INFORMATIKAI ALGORITMUSOK I.

Iványi Antal

alkotó szerkesztő

INFORMATIKAI

ALGORITMUSOK I.

ELTE Eötvös Kiadó, Budapest, 2004

A könyv az Oktatási Minisztérium támogatásával, a Felsőoktatási Pályázatok Irodája által lebonyolított felsőoktatási tankönyvtámogatási program keretében jelent meg.

Alkotó szerkesztő: Iványi Antal

Szerzők: Kása Zoltán (1.), Járai Antal és Kovács Attila (2.), Jörg Rothe (3. és 4.), Gyires Tibor (5.), Iványi Antal és Claudia Leopold (6.), Eberhard Zehendner (7.), Szidarovszky Ferenc (8.), Vizvári Béla (9.), Ulrich Tamm (10.), Balogh Ádám és Iványi Antal (11.), Demetrovics János és Sali Attila (12.), Miklós István (13.), Ingo Althöfer és Stefan Schwartz (14.), Szirmay-Kalos László (15.), Elek István és Sidló Csaba (16.), Galántai Aurél és Jeney András (17.)

Szakmai lektorok: Fekete István (1.), Rónyai Lajos (2.), Gonda János (3.), Ivanyos Gábor (4.), Tőke Pál (5.), Sima Dezső (6. és 7.), Mayer János (8.), Csirik János (9.), Fridli Sándor (10.), Varga László (11.), Kiss Attila (12.), Hunyadvári László és Katsányi István (13.), Szántai Tamás (14.), Vida János (15.), Meskó Attila (16.), Szántai Tamás (17.)

Nyelvi lektor: Biró Gabriella

Fordítók: Láng Zsuzsa (3.), Sidló Csaba (4.), Roszik János és Sztrik János (5.), Szakács Laura (7.), Pintér Miklós (8.), Sike Sándor (10.), Belényesi Viktor és Nikovits Tibor (14.)

A könyv címoldalán – a Szépművészeti Múzeum engedélyével és az ELTE Informatikai Karának támogatásával – Vasarely Victor *Dirac* című festménye látható. A borítóhoz felhasznált filmet a GOMA RT. bocsátotta rendelkezésünkre. A borítót Iványi Antal tervezte.

ISBN: 963 463 664 0

Kiadja az ELTE Eötvös Kiadó
1051 Budapest, Szerb utca 21.
Telefon: 411-6740, Fax: 485-52-26

Honlap: http://www.elte.hu/szervezet/eotvos_kiado.html

Elektronikus cím: eotvoskiado@ludens.elte.hu

Felelős kiadó: Pándi András

Nyomás és kötés: Debreceni Kinizsi Nyomda

Felelős vezető: Bördős János

Tartalomjegyzék

Tartalomjegyzék **5**
Előszó **9**

I. ALAPOK **12**
Bevezetés **13**
1. Rekurzív egyenletek (Kása Zoltán) **14**
1.1. Lineáris rekurzív egyenletek 15
1.2. Generátorfüggvények és rekurzív egyenletek 22
1.3. Numerikus megoldás 36
2. Komputeralgebra (Járai Antal és Kovács Attila) **38**
2.1. Adatábrázolás 39
2.2. Polinomok közös gyökei 44
2.3. Gröbner-bázis 63
2.4. Szimbolikus integrálás 71
2.5. Elmélet és gyakorlat 88
3. Kriptográfia (Jörg Rothe) **94**
3.1. Alapok 95
3.2. Kulcscsere Diffie és Hellman szerint 107
3.3. RSA és faktorizálás 110
3.4. Rivest, Rabi és Sherman protokolljai 116
3.5. Interaktív bizonyítási rendszerek és zéró ismeret 116
4. Bonyolultságelmélet (Jörg Rothe) **125**
4.1. Alapok 126
4.2. NP-teljesség 133
4.3. Az ítéletlogika kielégíthetőség-problémája 139
4.4. Gráfizomorfizmus és alsóság 144

II. HÁLÓZATOK **162**
Bevezetés **163**
5. Hálózatok szimulációja (Gyires Tibor) **164**
5.1. A szimuláció típusai 164
5.2. A telekommunikációs hálózatok modellezésének és szimulációjának szükségessége 165
5.3. A telekommunikációs hálózatok típusai 167
5.4. Teljesítményjellemzők szimulációhoz 169
5.5. A forgalom jellemzése 172
5.6. Szimulációs modellező rendszerek 178
5.7. Modellfejlesztési életciklus 189
5.8. A forgalom ingadozásának hatása nagy sebességű hálózatokra 195
5.9. Mérési adatok bemutatása 210
6. Párhuzamos számítások (Iványi Antal és Claudia Leopold) **222**
6.1. Párhuzamos architektúrák 224
6.2. Hatékonysági mértékek és optimalizálás 227
6.3. Párhuzamos programozás 235
6.4. Számítási modellek 242
6.5. PRAM algoritmusok 244
7. Szisztolikus rendszerek (Eberhard Zehendner) **268**
7.1. A szisztolika alapfogalmai 268
7.2. Tér-idő-leképezés és szisztolikus rács 278
7.3. A be/kiviteli séma levezetése 287
7.4. Vezérlési szempontok 295
7.5. Lineáris szisztolikus rácsok 306

III. FOLYTONOS OPTIMALIZÁCIÓ **312**
Előszó **313**
8. Játékelmélet (Szidarovszky Ferenc) **314**
8.1. Véges játékok 315
8.2. Folytonos játékok 320
8.3. Az oligopol feladat 349

IV. DISZKRÉT OPTIMALIZÁCIÓ **362**
Bevezetés **363**
9. Ütemezéselmélet (Vizvári Béla) **364**
9.1. Formális rendszer ütemezési feladatok osztályozására 365
9.2. A Gantt-diagramok 368
9.3. Ütemezési problémák egyetlen gépen 370
9.4. Ütemezési problémák párhuzamos berendezéseken 378
9.5. Az egyutas ütemezési probléma 389
9.6. A többutas ütemezési probléma 397

V. ADATBÁZISKEZELÉS **416**

Bevezetés **417**

10. Adattömörítés (Ulrich Tamm) **418**
10.1. Információelméleti eredmények 419
10.2. Aritmetikai kódolás és modellezés 429
10.3. Ziv–Lempel tömörítés 440
10.4. Burrows–Wheeler-transzformáció 443
10.5. Képtömörítés 447

11. Memóriagazdálkodás (Balogh Ádám és Iványi Antal) **456**
11.1. Partícionálás 456
11.2. Lapcserélési algoritmusok 470
11.3. Anomáliák 480
11.4. Állományok optimális elhelyezése 491

12. Relációs adatmodell tervezés (Demetrovics János és Sali Attila) **503**
12.1. Bevezetés 503
12.2. Funkcionális függőségek 504
12.3. Relációs sémák szétvágása 511
12.4. Általános függőségek 530

VI. ALKALMAZÁSOK **536**

Bevezetés **537**

13. Bioinformatika (Miklós István) **538**
13.1. Algoritmusok szekvenciákon 538
13.2. Algoritmusok fákon 552
13.3. Algoritmusok sztochasztikus nyelvtanokon 554
13.4. Szerkezetek összehasonlítása 560
13.5. Törzsfakészítés távolságon alapuló algoritmusokkal 562
13.6. Válogatott témák 572

14. Ember-gép kölcsönhatás (Ingo Althöfer és Stefan Schwarz) **580**
14.1. Több választási lehetőséget kínáló rendszerek 580
14.2. Több lehetséges megoldás előállítása 584
14.3. További interaktív problémamegoldó algoritmusok 600

15. Számítógépes grafika (Szirmay-Kalos László) **605**
15.1. Analitikus geometriai alapok 605
15.2. Ponthalmazok leírása egyenletekkel 606
15.3. Geometriai feldolgozó és tesszellációs algoritmusok 618
15.4. Tartalmazási algoritmusok 628
15.5. Mozgatás, torzítás, geometriai transzformációk 637
15.6. Megjelenítés sugárkövetéssel 645
15.7. Az inkrementális képszintézis algoritmusai 662

16. Térinformatika (Elek István és Sidló Csaba) **685**
16.1. A térinformatika adatmodelljei 685
16.2. Térbeli indexelés 687
16.3. Digitális szűrési eljárások 694
16.4. Mintavételezés 710
17. Tudományos számítások (Galántai Aurél és Jeney András) **717**
17.1. Lebegőpontos aritmetika és hibaelemzés 717
17.2. Lineáris egyenletrendszerek 727
17.3. Sajátértékszámítás 748
17.4. Numerikus programkönyvtárak és szoftvereszközök 758
Irodalomjegyzék **769**
Tárgymutató **787**
Színes ábrák és ismertetők **799**

Előszó

Az informatikai algoritmusok magyar nyelvű szakirodalma az utóbbi huszonöt évben alakult ki. Az első szakkönyvet Lovász László és Gács Péter írta 1978-ban [295]. Ezt a könyvet fordítások követték: 1982-ben Aho, Hopcroft és Ullman [6] könyve, 1987-ben Knuth háromkötetes monográfiája [258, 259, 260], majd 1987-ben Cormen, Leiserson és Rivest műve [89]. 1999-ben újra hazai szerzők következtek – Rónyai Lajos, Ivanyos Gábor és Szabó Réka [392] – majd 2002-ben megjelent Lynch *Osztott algoritmusok* című monográfiája [301].

Ezt 2003 tavaszán Iványi Antal *Párhuzamos algoritmusok* című könyve [221], majd 2003 őszén – *Új algoritmusok* címmel – Cormen, Leiserson, Rivest és Stein tankönyvének [90] fordítása követte.

A magyar informatikus hallgatók és gyakorlati szakemberek nagy érdeklődéssel fogadták az *Új algoritmusokat* – néhány hónap alatt a kiadott 2000 példány fele gazdára talált. Ez ösztönözte ennek a könyvnek a hazai szerzőit, hogy – külföldi kollégáik segítségével – további informatikai területek algoritmusait is összefoglalják.

A könyv tartalmát hat részre tagoljuk: *Alapok, Hálózatok, Diszkrét optimalizálás, Folytonos optimalizálás, Adatbázisok* és *Alkalmazások*.

Az első kötetbe azok a fejezetek került, amelyek szeptemberig elkészültek. Minden fejezet bemutat egy alkalmazási vagy elméleti szempontból lényeges területet és azokhoz kapcsolódó algoritmusokat. Az algoritmusok többségét szóban és olyan pszeudokóddal is megadjuk, amely a programozási tapasztalattal rendelkező olvasók számára könnyen érthető.

Az első kötet 247 ábrát, 157 pszeudokódot és 133 példát tartalmaz, amelyek elősegítik a tárgyalt algoritmusok működésének megértését. Az önálló tanulást az alfejezetek végén lévő gyakorlatok (összesen 269), az egyes témákban való elmélyülést pedig a fejezetek végén lévő (összesen 66) feladatok segítik. A fejezetek anyagával kapcsolatos friss és kiegészítő ismeretekre való utalások találhatók a fejezetek végén lévő *Megjegyzések a fejezethez* című részben. Az irodalomjegyzékben megadjuk egyrészt a felhasznált szakirodalom bibliográfiai adatait, másrészt – teljességre törekedve – felsoroljuk a magyar nyelvű forrásokat.

Az algoritmusok bemutatása az igényelt erőforrások – elsősorban futási idő és memória – elemzését is magában foglalja. A szakirodalomban szokásos módon felső korlátokat adunk meg a legrosszabb esetre jellemző erőforrásigényre, és esetenként a megoldandó probléma erőforrásigényére jellemző alsó korlátot is levezetünk.

A könyv kéziratát LaTeX kiadványszerkesztő eszköz segítségével készítettük, amelyet

az elmúlt hat év során Belényesi Viktorral és Locher Kornéllal fejlesztettünk ki, és korábban már három könyv kéziratának előállítására használtunk. Az ábrák többségét Locher Kornél rajzolta. Az irodalomjegyzéket Iványi Anna tette élővé.

Garey és Johnson klasszikus művét [149] követve mindazon algoritmusok futási idejét exponenciálisnak nevezzük, amelyekre nem adható polinomiális felső korlát.

Az *Új algoritmusok* példáját követve tizedespontot használunk.

Mindig különös gondot fordítunk könyveink külsejére. Az adott esetben olyan megoldást kerestünk, amely

- tükrözi a könyv tartalmi gazdagságát (az első kötet 17 és a második kötet hasonló számú fejezetét)
- és az alkotók szoros kötődését mind Magyarországhoz, mind pedig Európához.

Úgy gondoljuk, hogy a pécsi születésű Vásárhelyi Viktor – aki francia festőként Victor Vasarely néven vált világhírűvé – képeire jellemző a formák és színek gazdagsága, életútja pedig tükrözi kultúránk európai kötődését.

A budapesti és pécsi múzeumokban összesen közel 500 Vasarely-alkotás van. Ezek a művész ajándékai – a szülőföld iránti hála és tisztelet szimbólumai. Vasarely gazdag életművéből a könyv alkotói és majdani olvasói segítségével választottunk. A szavazók a *Dirac, Kubtuz, Rikka, Sixa* és a *Tupa-fond* című képeket emelték ki. Közülük két olyan képet – a *Dirac* és a *Kubtuz* című festményeket – választottuk, amelyeken szakaszokból kör alakul ki – szemléltetve az informatika azon alapvető tulajdonságát, hogy a folytonos valós világot diszkrét objektumokkal (bitekkel) írja le – és amelyekhez sikerült felhasználási engedélyt kapnunk.

Közismert, hogy az elmúlt évszázadban nemcsak művészeink, hanem sok kiváló tudósunk is külföldön ért fel a csúcsra. Nagy részükre azonban folyamatosan számíthat a hazai oktatás és tudományos élet. A hálózati szimulációs fejezet szerzője Gyires Tibor (Illinois Egyetem), a játékelméleti fejezetet pedig Szidarovszky Ferenc (Arizonai Műszaki Egyetem) írta. A második kötetben a megbízhatóságról szóló fejezetet Gács Péter (Bostoni Egyetem) írta, a belsőpontos módszerekről szóló fejezet egyik szerzője pedig Terlaky Tamás (McMaster Egyetem). Ma mind a négy szerző az adott terület vezető kutatója, amerikai egyetemek professzora – egykor magyar egyetemen tanultak, majd tanítottak.

A rekurziós fejezet szerzője Kása Zoltán (Babeş-Bolyai Tudományegyetem), a szisztolikus rendszerekről szóló fejezetet Szakács Laura (Babeş-Bolyai Tudományegyetem) fordította németről magyarra. Részvételük a könyv megszületésében a határainkon túli magyar nyelvű oktatással való szoros kapcsolatunk része.

Könyvünk tartalmi gazdagsága jó külföldi – elsősorban német – kapcsolatainknak is köszönhető. Az első kötet kriptográfiai és bonyolultságelméleti fejezetét Jörg Rothe (Düsseldorfi Egyetem), szisztolikus rendszerekkel foglalkozó fejezetét Eberhard Zehendner (Friedrich Schiller Egyetem) írta. Az adattömörítési fejezet szerzője Ulrich Tamm (Chemnitzi Egyetem), a párhuzamos programozásról szóló fejezet egyik szerzője Claudia Leopold (Kasseli Egyetem), az ember-gép kapcsolatokkal foglalkozó fejezet szerzői Ingo Althöfer és Stefan Schwartz (Friedrich Schiller Egyetem).

Az alkotók (szerzők, lektorok, fordítók és segítőtársaik) többsége a hazai informatikai felsőoktatás meghatározó intézményeinek – Budapesti Corvinus Egyetem, Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapesti Műszaki Főiskola, Debreceni Egyetem, Eötvös Loránd Tudományegyetem, Miskolci Egyetem, Pécsi Tudományegyetem, Szegedi

Tudományegyetem – oktatója.

Az *Oktatási Minisztérium* támogatásának köszönhetően ez a tankönyv nagyon kedvező áron juthat el az Olvasókhoz. Ugyancsak az Oktatási Minisztérium támogatásának köszönhető, hogy 2005 tavaszától a könyv elektronikus változata is mindenki számára szabadon hozzáférhető lesz.

Az alábbi kollégáknak köszönjük, hogy a tervezett könyv mindkét formáját támogatták: Fazekas Gábor egyetemi docens (Debreceni Egyetem Informatikai Karának dékánhelyettese), Imreh Balázs egyetemi docens (Szegedi Egyetem), Kása Zoltán egyetemi tanár (BBTE Matematikai és Informatikai Karának dékánhelyettese), Kozma László egyetemi docens (ELTE Informatikai Karának dékánja), Jörg Rothe egyetemi tanár (Heinrich Heine Universität, Düsseldorf), Sima Dezső főiskolai tanár (Budapesti Műszaki Főiskola Neumann János Informatikai Karának főigazgatója), Sidló Csaba PhD hallgató (ELTE Informatikai Doktori Iskola), Szeidl László egyetemi tanár (Pécsi Tudományegyetem Matematikai és Informatikai Intézet igazgatója), Szidarovszky Ferenc egyetemi tanár (Arizonai Műszaki Egyetem), Szirmay-Kalos László egyetemi tanár (BME Villamosmérnöki és Informatikai Kara), Terlaky Tamás egyetemi tanár (McMaster Egyetem, Hamilton)

Ugyancsak köszönjük azoknak a kollégáinknak a segítőkészségét, akiknek a lektori véleményét csatolni tudtuk a pályázathoz: Fekete István egyetemi docens (*Rekurziók* című fejezet), Fridli Sándor egyetemi docens (*Adattömörítés*), Gonda János egyetemi docens (*Kriptográfia*), Hunyadvári László egyetemi docens és Katsányi István PhD hallgató (*Bioinformatika*), Kiss Attila egyetemi docens (*Relációs adatbázisok tervezése*), Tőke Pál egyetemi docens (*Hálózatok szimulációja*), Vida János egyetemi docens (*Grafika*).

Az elektronikus változat elkészültéig a

`http://people.inf.elte.hu/tony/konyvek/infalg`

című honlapon találják meg olvasóink a könyv kiegészítését, amely többek között az élő irodalomjegyzéket, a névmutatót, a gyakorlatok és feladatok egy részének megoldását, működő programokat, a talált hibák jegyzékét tartalmazza. Ezen a honlapon keresztül fogjuk Olvasóinkat tájékoztatni az elektronikus változat hálózati címéről.

Köszönet illeti azokat – Belényesi Viktor, Thomas H. Cormen, Érsek Nándor, Pavel Fabian, Johan Gade, Iványi Anna, Kimo Johnson, Shamir Kuller, Lőrentey Károly, Stefan Schumann, Joel Seiferas, Beau Skinner, Ofer Springer, George Stephanides, Sekar Sundarraj, Szántai Tamás, Szeidl László, Ronald Tungl, Veszprémi Anna, David Wagner – akik észrevételeikkel segítettek a könyvünk alapjául szolgáló mű, az *Új algoritmusok* első kiadásának javításában.

Az előzetes érdeklődés alapján arra számítunk, hogy rövidesen újabb kiadásra lesz szükség. Ebben szeretnénk az első kiadás hibáit kijavítani. Ezért kérjük a könyv Olvasóit, hogy javaslataikat, észrevételeiket küldjék el a `tony@inf.elte.hu` címre – levelükben lehetőleg pontosan megjelölve a hiba előfordulási helyét, és megadva a javasolt szöveget.

A könyv honlapján megtalálható a szerzők és lektorok elektronikus címeit tartalmazó kolofonoldal. Olvasóink javaslataikkal, kérdéseikkel megkereshetik a könyv alkotóit.

Budapest, 2004. szeptember 22.

Iványi Antal

alkotó szerkesztő

I. ALAPOK

Bevezetés

Ebben az alapozó részben négy témakört tárgyalunk.

Az informatikai algoritmusok elemzése során gyakran előfordul, hogy például felismerjük az n és $n + 1$ méretű feladatok megoldási ideje közötti kapcsolatot – és ennek az úgynevezett rekurzív egyenletnek a felhasználásával szeretnénk közvetlenül felírni az n méretű bemenethez tartozó futási időt. Az első fejezet a *rekurzív egyenletek* leggyakrabban előforduló típusainak megoldási módszereit mutatja be.

A mai számítógépek sebessége és tárolókapacitása, valamint az elméleti eredmények számos olyan feladat kényelmes (mechanikus) megoldását lehetővé teszik, melyeket korábban nem, vagy csak nagy nehézségek árán tudtunk kezelni. Ezek egy része – mint a formális differenciálás és integrálás – a második fejezetben tárgyalt *komputeralgebrához* tartozik.

Az elektronikus kommunikáció hatalmas iramú terjedésével együtt nő a kommunikáció biztonságának jelentősége. Ezért a mai informatika egyik kulcsfontosságú területe a *kriptográfia,* mellyel a könyv harmadik fejezete foglalkozik.

Az algoritmusok elemzésének hagyományosan fontos része az erőforrásigény legrosszabb esetre vonatkozó felső korlátjának megadása. Az csak az utóbbi 15 évben vált természetessé, hogy az erőforrásigényre vonatkozó – a probléma és a megengedett algoritmusok tulajdonságain alapuló – alsó korlátokat is megadnak.

Például Donald Knuth *The Art of Computer Programming* című monográfiájának 1968-ban megjelent első kötetében szerepelt az aszimptotikus felső korlátok jellemzésére használt O-jelölés *(nagy ordo)* definíciója – ugyanakkor még nem szerepelt az alsó korlátok jellemzésére alkalmas Ω-jelölés, valamint a pontos nagyságrend megadására alkalmas Θ-jelölés. Az *Introduction to Algorithms* 1990-ben megjelent első kiadásában, a *Distributed Algorithms* 1996-ban megjelent első kiadásában, valamint Knuth könyvének 1997-ben megjelent harmadik kiadásában már az Ω-jelölés és a Θ-jelölés definíciója is szerepel.

A *negyedik fejezet* szerint a bonyolultságelmélet fontos feladata, hogy a problémákhoz és számítási modellekhez minél pontosabb alsó korlátokat adjon meg – ezzel is segítve a problémák erőforrásigény szerinti osztályozását.

A második kötetben fog megjelenni az *algebrai algoritmusok* elemzése.

1. Rekurzív egyenletek

Közismert a Fibonacci-számok rekurzív definíciója: ha F_n jelöli az n-edik Fibonacci-számot, akkor

$$F_0 = 0, \quad F_1 = 1 \ ,$$

$$F_{n+2} = F_{n+1} + F_n, \qquad \text{ha} \ \ n \geq 0 \ .$$

Szeretnénk explicit formában megadni F_n értékét tetszőleges n-re. A feladat tulajdonképpen olyan egyenlet megoldását kéri, amelyben az ismeretlen rekurzív módon van megadva, ezért ***rekurzív egyenletnek*** hívjuk. Itt a megoldás felfogható úgy, mint természetes számokon értelmezett függvény, mivel F_n minden n-re értelmezett. Az ilyen rekurzív egyenletet szokás még ***differenciaegyenletnek*** is nevezni, de nevezhetnénk akár ***diszkrét differenciálegyenletnek*** is.

1.1. definíció. *A* ***k-adrendű rekurzív egyenlet*** $(k \geq 1)$ *egy*

$$f(x_n, x_{n+1}, \ldots, , x_{n+k}) = 0, \qquad n \geq 0 \tag{1.1}$$

alakú egyenlet, ahol x_n-et kell explicit formában megadnunk.

Ahhoz, hogy egyértelműen meghatározhassuk x_n-et, meg kell adnunk k kezdőértéket, ezek általában $x_0, x_1, \ldots, x_{k-1}$. Ezek az értékadások ***kezdeti feltételeknek*** tekinthetők.

Mivel a Fibonacci-számokat definiáló egyenlet másodrendű rekurzív egyenlet, ezért ott két kezdeti értéket adunk meg.

Az (1.1) egyenletet és annak adott kezdeti feltételeit kielégítő $x_n = g(n)$ sorozatot az adott egyenlet ***partikuláris megoldásának*** nevezzük. Ha az $x_n = h(n, C_1, C_2, \ldots, C_k)$ sorozatból – a $C_1, C_2, \ldots, C_k$ állandók alkalmas megválasztásával – az (1.1) egyenlet minden partikuláris megoldása előállítható, akkor a sorozatot az egyenlet ***általános megoldásának*** nevezzük.

A rekurzív egyenletek megoldása általában nem egyszerű. A következőkben sajátos esetekben alkalmazható módszereket ismertetünk.

Az írásmódban függvény helyett inkább sorozatot használunk (ami tulajdonképpen természetes számokon értelmezett függvény). Így a jelölés egyszerűbb lesz, $x(n)$ helyett mindenhol x_n-t írunk.

A fejezet három részből áll. Az 1.1. alfejezetben a lineáris rekurzív egyenletek megoldásával, a 1.2. alfejezetben a generátorfüggvények felhasználásával, az 1.3. alfejezetben pedig lineáris rekurzív egyenletek numerikus megoldásával foglalkozunk.

1.1. Lineáris rekurzív egyenletek

Ha a rekurzív egyenlet

$$f_0(n)x_n + f_1(n)x_{n+1} + \cdots + f_k(n)x_{n+k} = f(n), \quad n \geq 0$$

alakú, ahol $f, f_0, f_1, \ldots, f_k$ természetes számokon értelmezett függvények, $f_0, f_k \neq 0$, és x_n-et kell explicit módon megadnunk, akkor ***lineáris*** rekurzív egyenletről beszélünk. Ha f azonosan nulla, akkor az egyenlet ***homogén,*** és különben ***inhomogén.*** Amennyiben az $f_0, f_1, \ldots, f_k$ függvények mindegyike állandó, akkor ***állandó együtthatós*** lineáris rekurzív egyenletről van szó.

1.1.1. Állandó együtthatós homogén lineáris rekurzív egyenletek

Legyen

$$a_0x_n + a_1x_{n+1} + \cdots + a_kx_{n+k} = 0, \qquad n \geq k\ , \tag{1.2}$$

ahol $a_0, a_1, \ldots, a_k$ valós állandók, $a_0, a_k \neq 0,\ k \geq 1$. Amennyiben adva van k kezdeti érték (leggyakrabban $x_0, x_1, \ldots, x_{k-1}$), az egyenlet megoldása egyértelműen meghatározható.

A megoldás érdekében rendeljük hozzá az egyenlethez a ***karakterisztikus egyenletét:***

$$a_0 + a_1r + \cdots + a_{k-1}r^{k-1} + a_kr^k = 0\ , \tag{1.3}$$

amely valós együtthatós egyenlet. Ennek az egyenletnek k gyöke van a komplex számok körében. Behelyettesítéssel ellenőrizhető, hogy ha r_0 valós megoldása a karakterisztikus egyenletnek, akkor $C_0r_0^n$ megoldása az (1.2) egyenletnek, ahol C_0 tetszőleges állandó.

Az (1.2) egyenlet általános megoldása

$$x_n = C_1x_n^{(1)} + C_2x_n^{(2)} + \cdots + C_kx_n^{(k)}\ ,$$

ahol $x_n^{(i)}$ $(i = 1, 2, \ldots, k)$ az (1.2) egyenlet lineárisan független megoldásai. A kezdeti feltételekből mindig meghatározhatók a $C_1, C_2, \ldots, C_k$ állandók egy k egyenletből álló egyenletrendszer megoldásával.

A lineárisan független megoldásokat a karakterisztikus egyenlet gyökei szolgáltatják a következők szerint. Minden gyökhöz hozzárendelhető egy ***fundamentálisnak*** nevezett megoldás.

Különböző valós gyökök

Legyenek $r_1, r_2, \ldots, r_p$ a karakterisztikus egyenlet egymástól különböző valós gyökei. Ekkor

$$r_1^n,\ r_2^n,\ \ldots,\ r_p^n$$

megoldásai az (1.2) rekurzív egyenletnek, és

$$C_1r_1^n + C_2r_2^n + \cdots + C_pr_p^n \tag{1.4}$$

is az, tetszőleges $C_1, C_2, \ldots, C_p$ állandókra. Ha $p = k$, akkor (1.4) a rekurzív egyenlet általános megoldása.

1.1. példa. Oldjuk meg az

$$x_{n+2} = x_{n+1} + x_n, \quad x_0 = 0,\ x_1 = 1$$

rekurzív egyenletet. A karakterisztikus egyenlet

$$r^2 - r - 1 = 0\ ,$$

amelynek gyökei

$$r_1 = \frac{1+\sqrt{5}}{2}\ ,\quad r_2 = \frac{1-\sqrt{5}}{2}\ .$$

Ezek valósak és egymástól különböznek, tehát az egyenlet általános megoldása

$$x_n = C_1\left(\frac{1+\sqrt{5}}{2}\right)^n + C_2\left(\frac{1-\sqrt{5}}{2}\right)^n\ .$$

A C_1 és C_2 meghatározhatók a kezdeti feltételekből. Ha figyelembe vesszük, hogy $x_0 = 0$, $x_1 = 1$, a következő egyenletrendszerhez jutunk:

$$\begin{aligned} C_1 + C_2 &= 0\ ,\\ C_1\frac{1+\sqrt{5}}{2} + C_2\frac{1-\sqrt{5}}{2} &= 1\ . \end{aligned}$$

Az egyenletrendszer megoldása $C_1 = 1/\sqrt{5},\ C_2 = -1/\sqrt{5}$. Így az általános megoldás

$$x_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n,$$

amely éppen F_n, az n-edik Fibonacci-szám.

Többszörös valós gyökök

Legyen r egy p-szeres gyöke a karakterisztikus egyenletnek. Ekkor

$$r^n,\ nr^n,\ n^2r^n,\ \ldots,\ n^{p-1}r^n$$

megoldásai az (1.2) rekurzív egyenletnek (az r többszörös gyökhöz tartozó fundamentális megoldások), és

$$(C_0 + C_1n + C_2n^2 + \cdots + C_{p-1}n^{p-1})r^n \tag{1.5}$$

is megoldás, tetszőleges $C_0,\ C_1,\ \ldots,\ C_{p-1}$ állandókra. Ha a karakterisztikus egyenletnek nincs más gyöke, akkor (1.5) a rekurzív egyenlet általános megoldása.

1.2. példa. Oldjuk meg az

$$x_{n+2} = 4x_{n+1} - 4x_n, \quad x_0 = 1,\ x_1 = 3$$

rekurzív egyenletet. A karakterisztikus egyenlet

$$r^2 - 4r + 4 = 0\ ,$$

amelynek $r = 2$ kétszeres gyöke. Ekkor

$$x_n = (C_0 + C_1n)2^n$$

megoldása az egyenletnek.

A kezdeti feltételekből

$$\begin{aligned} C_0 &= 1\,, \\ 2C_0 + 2C_1 &= 3\,. \end{aligned}$$

Innen $C_0 = 1$, $C_1 = 1/2$, azaz az általános megoldás

$$x_n = \left(1 + \frac{1}{2}n\right)2^n \quad \text{vagy} \quad x_n = (n+2)2^{n-1}\,.$$

Egyszeres komplex gyökök

Ha a trigonometrikus alakban felírt $a(\cos b + i \sin b)$ komplex szám gyöke a karakterisztikus egyenletnek, akkor az $a(\cos b - i \sin b)$ konjugált is az, mivel a karakterisztikus egyenlet valós együtthatós. Ekkor

$$a^n \cos bn \quad \text{és} \quad a^n \sin bn$$

megoldása az (1.2) rekurzív egyenletnek és

$$C_1 a^n \cos bn + C_2 a^n \sin bn \tag{1.6}$$

is az, tetszőleges C_1 és C_2 állandókra. Ha a karakterisztikus egyenletnek nincsenek más gyökei, akkor (1.6) általános megoldás.

1.3. példa. Oldjuk meg az

$$x_{n+2} = 2x_{n+1} - 2x_n, \quad x_0 = 0,\ x_1 = 1$$

rekurzív egyenletet. A karakterisztikus egyenlet

$$r^2 - 2r + 2 = 0\,,$$

amelynek gyökei $1 + i$ és $1 - i$, trigonometrikus alakban: $\sqrt{2}(\cos(\pi/4) + i\sin(\pi/4))$ és $\sqrt{2}(\cos(\pi/4) - i\sin(\pi/4))$. Ezért a rekurzív egyenletnek

$$x_n = C_1(\sqrt{2})^n \cos\frac{n\pi}{4} + C_2(\sqrt{2})^n \sin\frac{n\pi}{4}$$

megoldása. A kezdeti feltételekből

$$\begin{aligned} C_1 &= 0\,, \\ C_1\sqrt{2}\cos\frac{\pi}{4} + C_2\sqrt{2}\sin\frac{\pi}{4} &= 1\,. \end{aligned}$$

Innen azt kapjuk, hogy $C_1 = 0,\ C_2 = 1$. Az általános megoldás tehát

$$x_n = (\sqrt{2})^n \sin\frac{n\pi}{4}\,.$$

Többszörös komplex gyökök

Ha a trigonometrikus alakban felírt $a(\cos b + i \sin b)$ komplex szám p-szeres gyöke a karakterisztikus egyenletnek, akkor az $a(\cos b - i \sin b)$ konjugált is az.

Ekkor az (1.2) rekurzív egyenletnek

$$a^n \cos bn,\ na^n \cos bn,\ \ldots,\ n^{p-1}a^n \cos bn$$

és

$$a^n \sin bn,\ na^n \sin bn,\ \ldots,\ n^{p-1}a^n \sin bn$$

megoldásai. Ekkor megoldás

$$(C_0 + C_1 n + \cdots + C_{p-1}n^{p-1})a^n \cos bn + (D_0 + D_1 n + \cdots + D_{p-1}n^{p-1})a^n \sin bn$$

is, ahol $C_0, C_1, \ldots, C_{p-1}, D_0, D_1, \ldots, D_{p-1}$ tetszőleges állandók, amelyek meghatározhatók a kezdeti feltételekből. Ez általános megoldás, ha a karakterisztikus egyenletnek nincsenek más gyökei.

1.4. példa. Oldjuk meg az

$$x_{n+4} + 2x_{n+2} + x_n = 0,\quad x_0 = 0,\ x_1 = 1,\ x_2 = 2,\ x_3 = 3$$

rekurzív egyenletet. A karakterisztikus egyenlet

$$r^4 + 2r^2 + 1 = 0\,,$$

amely $(r^2+1)^2 = 0$ alakban is írható, és amelynek i és $-i$ kétszeres gyöke. Ezek trigonometrikus alakja

$$i = \cos\frac{\pi}{2} + i\sin\frac{\pi}{2},\quad \text{valamint}\quad -i = \cos\frac{\pi}{2} - i\sin\frac{\pi}{2}\,.$$

Ezért az általános megoldás

$$x_n = (C_0 + C_1 n)\cos\frac{n\pi}{2} + (D_0 + D_1 n)\sin\frac{n\pi}{2}\,.$$

A kezdeti feltételekből következik:

$$\begin{aligned}
C_0 &= 0\,,\\
(C_0 + C_1)\cos\frac{\pi}{2} + (D_0 + D_1)\sin\frac{\pi}{2} &= 1\,,\\
(C_0 + 2C_1)\cos\pi + (D_0 + 2D_1)\sin\pi &= 2\,,\\
(C_0 + 3C_1)\cos\frac{3\pi}{2} + (D_0 + 3D_1)\sin\frac{3\pi}{2} &= 3\,,
\end{aligned}$$

azaz

$$\begin{aligned}
C_0 &= 0\,,\\
D_0 + D_1 &= 1\,,\\
-2C_1 &= 2\,,\\
-D_0 - 3D_1 &= 3\,,
\end{aligned}$$

és innen $C_0 = 0$, $C_1 = -1$, $D_0 = 3$ és $D_1 = -2$. Az általános megoldás tehát

$$x_n = (3 - 2n)\sin\frac{n\pi}{2} - n\cos\frac{n\pi}{2}\,.$$

A most vizsgált négy eset segítségével bármilyen állandó együtthatós homogén egyenletet megoldhatunk.

1.5. példa. Oldjuk meg az

$$x_{n+3} = 4x_{n+2} - 6x_{n+1} + 4x_n, \quad x_0 = 0,\ x_1 = 1,\ x_2 = 1$$

rekurzív egyenletet. A karakterisztikus egyenlet

$$r^3 - 4r^2 + 6r - 4 = 0\ ,$$

amelynek gyökei: 2, $1 + i$ és $1 - i$. Ezért az általános megoldás

$$x_n = C_1 2^n + C_2(\sqrt{2})^n \cos\frac{n\pi}{4} + C_3(\sqrt{2})^n \sin\frac{n\pi}{4}\ .$$

Az állandók meghatározása után

$$x_n = -2^{n-1} + \frac{(\sqrt{2})^n}{2}\left(\cos\frac{n\pi}{4} + 3\sin\frac{n\pi}{4}\right)\ .$$

Általános megoldás

Az (1.2) k-adrendű homogén lineáris rekurzív egyenlethez rendelt karakterisztikus egyenletnek összesen k gyöke van a komplex számok között, amelyek nem feltétlenül különbözők. Legyenek ezek a gyökök a következők:

r_1 valós, p_1-szeres $(p_1 \geq 1)$,
r_2 valós, p_2-szeres, $(p_2 \geq 1)$,
...
r_t valós, p_t-szeres, $(p_t \geq 1)$,
$s_1 = a_1(\cos b_1 + i \sin b_1)$ komplex, q_1-szeres $(q_1 \geq 1)$,
$s_2 = a_2(\cos b_2 + i \sin b_2)$ komplex, q_2-szeres $(q_2 \geq 1)$,
...
$s_m = a_m(\cos b_m + i \sin b_m)$ komplex, q_m-szeres $(q_m \geq 1)$.

Mivel összesen k gyök van, $p_1 + p_2 + \cdots + p_t + 2(q_1 + q_2 + \cdots + q_m) = k$.
Ekkor az (1.2) rekurzív egyenlet általános megoldása

$$\begin{aligned} x_n &= \sum_{j=1}^{t}\left(C_0^{(j)} + C_1^{(j)} n + \cdots + C_{p_j-1}^{(j)} n^{p_j-1}\right) r_j^n \\ &+ \sum_{j=1}^{m}\left(D_0^{(j)} + D_1^{(j)} n + \cdots + D_{q_j-1}^{(j)} n^{q_j-1}\right) a_j^n \cos b_j n \\ &+ \sum_{j=1}^{m}\left(E_0^{(j)} + E_1^{(j)} n + \cdots + E_{q_j-1}^{(j)} n^{q_j-1}\right) a_j^n \sin b_j n\ , \end{aligned} \tag{1.7}$$

ahol

$C_0^{(j)},\ C_1^{(j)},\ \ldots,\ C_{p_j-1}^{(j)},\ j = 1, 2, \ldots, t$,

$D_0^{(l)}$, $E_0^{(l)}$, $D_1^{(l)}$, $E_1^{(l)}$, …, $D_{p_l-1}^{(l)}$, $E_{p_l-1}^{(l)}$, $l = 1, 2, \ldots, m$ állandók, melyek a kezdeti feltételekből meghatározhatók.

Az eddigiek a következő tételben foglalhatók össze.

1.2. tétel. *Legyen $k \geq 1$ egész, a_0, a_1, …, a_k valós számok, a_0, $a_k \neq 0$. Az (1.2) lineáris rekurzív egyenlet általános megoldása előállítható az (17.39) karakterisztikus egyenlet r_i gyökeiből képezett $n^j r_i^n$ alakú tagok lineáris kombinációjaként, ahol a p_i-szeres r_i gyök esetében $0 \leq j < p$ és a lineáris kombináció együtthatói a kezdeti feltételektől függnek.*

A tétel bizonyítását az Olvasóra hagyjuk (lásd 1.1-5. gyakorlat).

A megoldás lépéseit a következőképpen foglalhatjuk össze. Feltesszük, hogy az egyenlet együtthatóit az A tömb, a megoldás állandóit pedig a C tömb tartalmazza.

HOMOGÉN-LINEÁRIS

1 írjuk fel a rekurzív egyenlet karakterisztikus egyenletét
2 keressük meg a karakterisztikus egyenlet összes gyökét, multiplicitásukkal együtt
3 írjuk fel az (1.7) általános megoldást a gyökök alapján
4 a kezdeti feltételekből, ha léteznek, határozzuk meg az (1.7)-ben szereplő állandókat

1.1.2. Állandó együtthatós inhomogén lineáris rekurzív egyenletek

Az állandó együtthatós inhomogén lineáris rekurzív egyenlet általános alakja

$$a_0 x_n + a_1 x_{n+1} + \cdots + a_k x_{n+k} = f(n) \,, \tag{1.8}$$

ahol $a_0, a_1, \ldots, a_k$ valós állandók, $a_0, a_k \neq 0$, $k \geq 1$, és $f(n)$ nem azonosan nulla.

Az egyenlethez tartozó (1.2) homogén lineáris egyenletet az 1.2. tétel szerint meg tudjuk oldani. Ha ismerjük az (1.8) egyenlet egy partikuláris megoldását, akkor az (1.8) egyenlet általános megoldását is elő tudjuk állítani.

1.3. tétel. *Legyen $k \geq 1$ egész, a_0, a_1, …, a_k valós számok, a_0, $a_k \neq 0$. Ha $x_n^{(1)}$ az (1.8) lineáris inhomogén rekurzív egyenlet egy partikuláris megoldása és $x_n^{(0)}$ az (1.8) egyenlethez tartozó (1.2) homogén lineáris egyenlet általános megoldása, akkor*

$$x_n = x_n^{(0)} + x_n^{(1)}$$

általános megoldása az (1.8) egyenletnek.

A tétel bizonyítását meghagyjuk az Olvasónak (lásd 1.1-6. gyakorlat).

1.6. példa. Oldjuk meg az

$$x_{n+2} + x_{n+1} - 2x_n = 2^n, \quad x_0 = 0, \; x_1 = 1$$

rekurzív egyenletet. Először megoldjuk az

$$x_{n+2} + x_{n+1} - 2x_n = 0$$

$f(n)$	$x_n^{(1)}$
$n^p a^n$	$(C_0 + C_1 n + \cdots + C_p n^p)a^n$
$a^n n^p \sin bn$	$(C_0 + C_1 n + \cdots + C_p n^p)a^n \sin bn + (D_0 + D_1 n + \cdots + D_p n^p)a^n \cos bn$
$a^n n^p \cos bn$	$(C_0 + C_1 n + \cdots + C_p n^p)a^n \sin bn + (D_0 + D_1 n + \cdots + D_p n^p)a^n \cos bn$

1.1. ábra. A partikuláris megoldás alakja.

homogén egyenletet, amelynek általános megoldása

$$x_n^{(0)} = C_1(-2)^n + C_2 \ ,$$

mivel a karakterisztikus egyenlet gyökei -2 és 1. Könnyen ellenőrizhetjük, hogy $x_n^{(1)} = 2^{n-2}$ megoldása az eredeti, inhomogén egyenletnek. Az inhomogén egyenlet általános megoldása tehát

$$x_n = C_1(-2)^n + C_2 + 2^{n-2} \ .$$

A C_1 és C_2 állandókat meghatározhatjuk a kezdeti feltételekből. Ennek alapján az általános megoldás

$$x_n = -\frac{1}{4}(-2)^n + 2^{n-2} \quad \text{vagy} \quad x_n = \frac{2^n - (-2)^n}{4} \ ,$$

azaz

$$x_n = \begin{cases} 0, & \text{ha } n \text{ páros} \ , \\ 2^{n-1}, & \text{ha } n \text{ páratlan} \ . \end{cases}$$

A partikuláris megoldás meghatározható a *konstansok variálásának módszerével*. Léteznek azonban olyan esetek, amikor a partikuláris megoldást könnyebben is megkaphatjuk. Az 1.1. ábrán olyan $f(n)$ függvényeket adunk meg, amelyek esetében az $x_n^{(1)}$ partikuláris megoldás a táblázatban megadott alakban kereshető. Az állandókat az egyenletbe való behelyettesítéssel kaphatjuk meg.

Előző példánk esetében $f(n) = 2^n$, tehát az első esetet alkalmazzuk, amikor $a = 2$, $p = 0$, ezért a $C_0 2^n$-nel próbálkozunk. Behelyettesítés után azt kapjuk, hogy $C_0 = 1/4$, tehát a partikuláris megoldás

$$x_n^{(1)} = 2^{n-2} \ .$$

Gyakorlatok

1.1-1. Oldjuk meg az alábbi inhomogén lineáris rekurzív egyenletet:

$$H_n = 2H_{n-1} + 1, \quad \text{ha } n \geq 1, \text{ és} \quad H_0 = 0 \ .$$

(H_n itt a Hanoi-tornyai nevű feladat megoldásához szükséges – és egyben elégséges – lépések számát jelenti.)

1.1-2. Elemezzük a Hanoi-tornyaira vonatkozó feladatot abban az esetben, amikor úgy kell n korongot átrakni az A rúdról a C rúdra, hogy közben az A rúdról a C rúdra *nem* szabad korongot átrakni.

Útmutatás. Mutassuk meg, hogy ha az optimális algoritmus lépéseinek száma M_n és $n \geq 1$, akkor $M_n = 3M_{n-1} + 2$.

1.1-3. Oldjuk meg a következő rekurzív egyenletet:

$$(n+1)R_n = 2(2n-1)R_{n-1}, \quad \text{ha } n \geq 1, \text{ és} \quad R_0 = 1.$$

1.1-4. Oldjuk meg alábbi inhomogén lineáris rekurzív egyenletet:

$$x_n = 2^n - 2 + 2x_{n-1}, \quad \text{ha } n \geq 2, \quad \text{és} \quad x_1 = 0.$$

Útmutatás. Keressük a partikuláris megoldást $C_1 n 2^n + C_2$ alakban.

1.1-5.⋆ Bizonyítsuk be az 1.2. tételt.

1.1-6.⋆ Bizonyítsuk be az 1.3. tételt.

1.2. Generátorfüggvények és rekurzív egyenletek

A generátorfüggvényeket, többek között, felhasználhatjuk rekurzív egyenletek megoldására, bizonyos objektumok (pl. bináris fák) megszámolására, azonosságok bizonyítására, partíciós problémák megoldására. Az objektumok megszámolása rekurzív egyenletek felállításával és megoldásával történik. Ezek a rekurzív egyenletek általában nem lineárisak, megoldásukban segíthetnek a generátorfüggvények.

1.2.1. Értelmezés és műveletek

Egy $(a_n)_{n\geq 0} = \langle a_0, a_1, a_2, \ldots, a_n, \ldots \rangle$ végtelen számsorozathoz hozzárendelhetünk egy hatványsort a következőképpen:

$$A(z) = a_0 + a_1 z + a_2 z^2 + \cdots + a_n z^n + \cdots = \sum_{n\geq 0} a_n z^n \,,$$

amelyet az $(a_n)_{n\geq 0}$ számsorozat ***generátorfüggvényének*** nevezünk.

Például a Fibonacci-számok esetében a generátorfüggvény a következő:

$$F(z) = \sum_{n\geq 0} F_n z^n = z + z^2 + 2z^3 + 3z^4 + 5z^5 + 8z^6 + 13z^7 + \cdots \,.$$

Ha mindkét oldalt megszorozzuk z-vel, majd z^2-tel, a következőket kapjuk:

$$\begin{aligned} F(z) &= F_0 + F_1 z + F_2 z^2 + F_3 z^3 + \cdots + F_n z^n + \cdots \,, \\ zF(z) &= F_0 z + F_1 z^2 + F_2 z^3 + \cdots + F_{n-1} z^n + \cdots \,, \\ z^2 F(z) &= F_0 z^2 + F_1 z^3 + \cdots + F_{n-2} z^n + \cdots \,. \end{aligned}$$

Ha kivonjuk tagonként az első képletből a másodikat, majd a harmadikat, és figyelembe

vesszük a Fibonacci-számokat definiáló képletet, a következőt kapjuk:

$$F(z)(1 - z - z^2) = z \ ,$$

ahonnan

$$F(z) = \frac{z}{1 - z - z^2} \ . \tag{1.9}$$

A fenti számítások helyességét matematikailag igazolni lehet, de nem térünk ki erre. A generátorfüggvények segítségével, formális műveletek során kapott eredményeket a legtöbb esetben más módszerekkel is lehet igazolni.

Tekintsük az

$$A(z) = \sum_{n\geq 0} a_n z^n \text{ és } B(z) = \sum_{n\geq 0} b_n z^n$$

generátorfüggvényeket.

Az $A(z)$ és $B(z)$ generátorfüggvényeket akkor és csakis akkor mondjuk *egyenlőnek,* ha $a_n = b_n$ bármely n természetes számra.

Ezután a következő, generátorfüggvényekkel végezhető műveleteket definiáljuk: összeadás és valós számmal való szorzás, eltolás, szorzás, deriválás, integrálás.

Összeadás és valós számmal való szorzás

$$\alpha A(z) + \beta B(z) = \sum_{n\geq 0} (\alpha a_n + \beta b_n) z^n \ .$$

Eltolás

A

$$z^k A(z) = \sum_{n\geq 0} a_n z^{n+k} = \sum_{n\geq k} a_{n-k} z^n$$

generátorfüggvény a $< \underbrace{0, 0, \ldots, 0}_{k}, a_0, a_1, \ldots >$ számsorozatot jelképezi, míg az

$$\frac{1}{z^k}(A(z) - a_0 - a_1 z - a_2 z^2 - \cdots - a_{k-1} z^{k-1}) = \sum_{n\geq k} a_n z^{n-k} = \sum_{n\geq 0} a_{k+n} z^n$$

generátorfüggvény az $< a_k, a_{k+1}, a_{k+2}, \ldots >$ sorozatot.

1.7. példa. Legyen $A(z) = 1 + z + z^2 + \cdots$. Ekkor

$$\frac{1}{z}\Big(A(z) - 1\Big) = A(z) \quad \text{és} \quad A(z) = \frac{1}{1 - z} \ .$$

Szorzás

Ha $A(z)$ és $B(z)$ generátorfüggvények, akkor

$$\begin{aligned} A(z)B(z) &= (a_0 + a_1 z + \cdots + a_n z^n + \cdots)(b_0 + b_1 z + \cdots + b_n z^n + \cdots) \\ &= a_0 b_0 + (a_0 b_1 + a_1 b_0) z + (a_0 b_2 + a_1 b_1 + a_2 b_0) z^2 + \cdots \\ &= \sum_{n\geq 0} s_n z^n, \end{aligned}$$

ahol $s_n = \sum_{k=0}^{n} a_k b_{n-k}$.

Sajátos eset. Ha $b_n = 1$ bármely n természetes számra, akkor

$$A(z)\frac{1}{1-z} = \sum_{n\geq 0}\left(\sum_{k=0}^{n} a_k\right) z^n \ . \tag{1.10}$$

Ha még ezenkívül $a_n = 1$ is igaz bármely n természetes számra, akkor

$$\frac{1}{(1-z)^2} = \sum_{n\geq 0} (n+1) z^n \ . \tag{1.11}$$

Deriválás

$$A'(z) = a_1 + 2a_2 z + 3a_3 z^2 + \cdots = \sum_{n\geq 0} (n+1) a_{n+1} z^n \ .$$

1.8. példa. Az

$$A(z) = \sum_{n\geq 0} z^n = \frac{1}{1-z}$$

generátorfüggvény mindkét oldalát deriválva azt kapjuk, hogy

$$A'(z) = \sum_{n\geq 1} n z^{n-1} = \frac{1}{(1-z)^2} \ .$$

Integrálás

$$\int_0^z A(t)dt = a_0 z + \frac{1}{2} a_1 z^2 + \frac{1}{3} a_2 z^3 + \cdots = \sum_{n\geq 1} \frac{1}{n} a_{n-1} z^n \ .$$

1.9. példa. Legyen

$$\frac{1}{1-z} = 1 + z + z^2 + z^3 + \cdots$$

Mindkét oldalát integrálva azt kapjuk, hogy

$$\ln \frac{1}{1-z} = z + \frac{1}{2} z^2 + \frac{1}{3} z^3 + \cdots = \sum_{n\geq 1} \frac{1}{n} z^n \ .$$

Ha a két fenti generátorfüggvényt összeszorozzuk, akkor

$$\frac{1}{1-z} \ln \frac{1}{1-z} = \sum_{n\geq 1} H_n z^n \ ,$$

ahol $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ $(H_0 = 0, \quad H_1 = 1)$ az ún. *harmonikus számok*.

Argumentum cseréje
Legyen $A(z) = \sum_{n\geq 0} a_n z^n$, amely az $< a_0, a_1, a_2, \ldots >$ sorozatot jelképezi, akkor $A(cz) = \sum_{n\geq 0} c^n a_n z^n$ pedig az $< a_0, ca_1, c^2 a_2, \ldots c^n a_n, \ldots >$ sorozatot. Igazak még a következők is:

$$\frac{1}{2}\Big(A(z) + A(-z)\Big) = a_0 + a_2 z^2 + \cdots + a_{2n} z^{2n} + \cdots \ ,$$

$$\frac{1}{2}\Big(A(z) - A(-z)\Big) = a_1 z + a_3 z^3 + \cdots + a_{2n-1} z^{2n-1} + \cdots \ .$$

1.10. példa. Legyen $A(z) = 1 + z + z^2 + z^3 + \cdots = \dfrac{1}{1-z}$. Ekkor

$$1 + z^2 + z^4 + \cdots = \frac{1}{2}(A(z) + A(-z)) = \frac{1}{2}\left(\frac{1}{1-z} + \frac{1}{1+z}\right) = \frac{1}{1-z^2} \ ,$$

amely megkapható úgyis, hogy z-t z^2-tel helyettesítjük $A(z)$-ben.

Hasonlóképpen, megkaphatjuk a páratlan kitevőjű tagok összegét:

$$z + z^3 + z^5 + \cdots = \frac{1}{2}(A(z) - A(-z)) = \frac{1}{2}\left(\frac{1}{1-z} - \frac{1}{1+z}\right) = \frac{z}{1-z^2} \ .$$

A generátorfüggvények segítségével érdekes képleteket kaphatunk. Legyen például $A(z) = 1/(1-z) = 1 + z + z^2 + z^3 + \cdots$. Ekkor $zA(z(1+z)) = F(z)$, vagyis éppen a Fibonacci-számok generátorfüggvénye. A fenti képletből

$$zA(z(1+z)) = z + z^2(1+z) + z^3(1+z)^2 + z^4(1+z)^3 + \cdots \ .$$

A z^{n+1} együtthatója a bal oldalon éppen F_{n+1}, vagyis az $(n+1)$-edik Fibonacci-szám, míg a z^{n+1} jobb oldali együtthatója, a binomiális képlet alkalmazása után minden tagban

$$\sum_{k\geq 0} \binom{n-k}{k} .$$

Innen

$$F_{n+1} = \sum_{k\geq 0} \binom{n-k}{k} = \sum_{k=0}^{\lfloor \frac{n+1}{2} \rfloor} \binom{n-k}{k} . \tag{1.12}$$

Emlékeztetünk, hogy a binomiális képlet általánosítható tetszőleges valós r-re is, vagyis

$$(1+z)^r = \sum_{n\geq 0} \binom{r}{n} z^n \ ,$$

amely a binomiális együtthatók generátorfüggvénye. Itt $\binom{r}{n}$ a kombináció általánosítása valós r-re, vagyis

$$\binom{r}{n} = \begin{cases} \dfrac{r(r-1)(r-2)\ldots(r-n+1)}{n(n-1)\ldots 1}, & \text{ha } n > 0 \ , \\ 1, & \text{ha } n = 0 \ , \\ 0, & \text{ha } n < 0 \ . \end{cases}$$

A binomiális képlet fenti általánosításával (negatív r-re) egy, sok esetben hasznos képletet kapunk. Legyen

$$\frac{1}{(1-z)^m} = (1-z)^{-m} = \sum_{k\geq 0} \binom{-m}{k} (-z)^k \, .$$

Mivel egyszerű számítással igazolható, hogy

$$\binom{-m}{k} = (-1)^k \binom{m+k-1}{k} ,$$

a következő képletet kapjuk:

$$\frac{1}{(1-z)^{m+1}} = \sum_{k\geq 0} \binom{m+k}{k} z^k \, .$$

Ekkor

$$\frac{z^m}{(1-z)^{m+1}} = \sum_{k\geq 0} \binom{m+k}{k} z^{m+k} = \sum_{k\geq 0} \binom{m+k}{m} z^{m+k} = \sum_{k\geq 0} \binom{k}{m} z^k \, .$$

Innen pedig

$$\sum_{k\geq 0} \binom{k}{m} z^k = \frac{z^m}{(1-z)^{m+1}} \, , \tag{1.13}$$

ahol m természetes szám.

1.2.2. Rekurzív egyenletek megoldása generátorfüggvényekkel

Ha a megoldandó rekurzív egyenlet olyan, hogy a megoldás generátorfüggvénye sorba fejthető úgy, hogy az együtthatók zárt alakban felírhatók, akkor ez a módszer eredményre vezet. Legyen adott a következő rekurzív egyenlet:

$$F(x_n, x_{n-1}, \ldots, x_{n-k}) = 0 \, . \tag{1.14}$$

A megoldáshoz tekintsük az

$$X(z) = \sum_{n\geq 0} x_n z^n$$

generátorfüggvényt. Ha (1.14) felírható $G(X(z)) = 0$ egyenletként, amelyet meg tudunk oldani $X(z)$-re, majd $X(z)$-t sorba lehet fejteni úgy, hogy x_n zárt alakban felírható, akkor az (1.14) egyenletet sikerrel oldottuk meg.

A következőkben általános módszert adunk az inhomogén lineáris egyenletek megoldására. Ezután három nemlineáris feladatra mutatunk példát. Két esetben bináris fák valamilyen halmazának az elemeit számoljuk meg, a harmadikban pedig a bináris fák leveleit. A három nemlineáris rekurzív egyenlet az (1.15), (1.17) és (1.18), amelyeket a generátorfüggvények segítségével oldunk meg.

Állandó együtthatós inhomogén lineáris rekurzív egyenlet
Szorozzuk be z^n-nel az (1.8) egyenlet mindkét oldalát. Ekkor

$$a_0 x_n z^n + a_1 x_{n+1} z^n + \cdots + a_k x_{n+k} z^n = f(n) z^n \, .$$

Összegezzük tagonként a fenti egyenlet mindkét oldalát:

$$a_0 \sum_{n\ge 0} x_n z^n + a_1 \sum_{n\ge 0} x_{n+1} z^n + \cdots + a_k \sum_{n\ge 0} x_{n+k} z^n = \sum_{n\ge 0} f(n) z^n \ .$$

Innen átalakításokkal kapjuk, hogy

$$a_0 \sum_{n\ge 0} x_n z^n + \frac{a_1}{z} \sum_{n\ge 0} x_{n+1} z^{n+1} + \cdots + \frac{a_k}{z^k} \sum_{n\ge 0} x_{n+k} z^{n+k} = \sum_{n\ge 0} f(n) z^n \ .$$

Legyen

$$X(z) = \sum_{n\ge 0} x_n z^n \quad \text{és} \quad F(z) = \sum_{n\ge 0} f(n) z^n \ .$$

Ekkor az egyenletünk így alakul:

$$a_0 X(z) + \frac{a_1}{z}\Big(X(z) - x_0\Big) + \cdots + \frac{a_k}{z^k}\Big(X(z) - x_0 - x_1 z - \cdots - x_{k-1} z^{k-1}\Big) = F(z) \ .$$

Ezt az egyenletet meg lehet oldani $X(z)$-ben. Az $X(z)$ kifejezésében a jobb oldali racionális törtet fel lehet bontani elemi (parciális) törtekre, majd azokat sorba fejtve meghatározhatjuk az eredeti egyenlet x_n általános megoldását, figyelembe véve a kezdeti feltételeket.

1.11. példa. Oldjuk meg a fenti módszerrel a következő egyenletet:

$$x_{n+1} - 2x_n = 2^{n+1} - 2, \text{ ha } n \ge 0 \quad \text{és } x_0 = 0 \ .$$

Beszorzás és összegezés után

$$\frac{1}{z} \sum_{n\ge 0} x_{n+1} z^{n+1} - 2 \sum_{n\ge 0} x_n z^n = 2 \sum_{n\ge 0} 2^n z^n - 2 \sum_{n\ge 0} z^n \ ,$$

innen pedig

$$\frac{1}{z}\Big(X(z) - x_0\Big) - 2X(z) = \frac{2}{1-2z} - \frac{2}{1-z} \ .$$

Mivel $x_0 = 0$, az egyenlet megoldása a következő, miután a jobb oldalt elemi törtekre bontottuk:[1]

$$X(z) = \frac{2z}{(1-2z)^2} - \frac{2}{1-z} - \frac{2}{1-2z} \ .$$

Az

$$\frac{1}{1-2z} = \sum_{n\ge 0} 2^n z^n$$

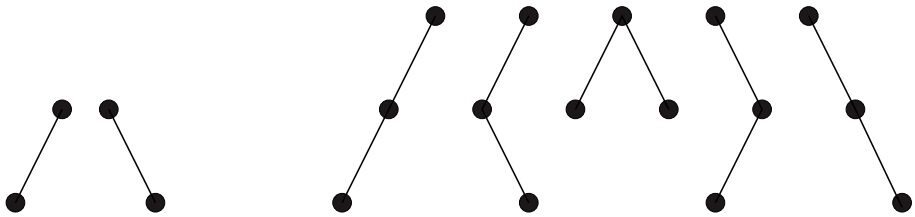
generátorfüggvény tagonkénti deriválásával a következőt kapjuk:

$$\frac{2}{(1-2z)^2} = \sum_{n\ge 1} n 2^n z^{n-1} \ .$$

Ezért

$$X(z) = \sum_{n\ge 0} n 2^n z^n + 2 \sum_{n\ge 0} z^n - 2 \sum_{n\ge 0} 2^n z^n = \sum_{n\ge 0} \Big((n-2)2^n + 2\Big) z^n \ ,$$

[1] Az elemi törtekre való bontást a határozatlan együtthatók módszerével végeztük.



1.2. ábra. Két- és háromcsúcsú bináris fák.

ahonnan

$$x_n = (n-2)2^n + 2 \; .$$

Bináris fák száma

Jelöljük b_n-nel az n csúcsú bináris fák számát. Ekkor $b_1 = 1$, $b_2 = 2$, $b_3 = 5$ (lásd az 1.2. ábrát). Legyen $b_0 = 1$. (Később látni fogjuk, hogy ez jó választás.)

Ha rögzítjük egy n csúcsú bináris fa gyökerét, akkor még $n-1$ csúcs marad a bal és jobb részfában összesen. Ha k csúcs van a bal oldali, $n-1-k$ pedig a jobb oldali részfában, akkor összesen $b_k b_{n-1-k}$ ilyen bináris fa létezik. Összegezve $k = 0, 1, \ldots, n-1$ értékekre, pontosan b_n-t kapjuk. Tehát tetszőleges $n \geq 1$ természetes számra a b_n-ben megoldandó rekurzív egyenlet a következő:

$$b_n = b_0 b_{n-1} + b_1 b_{n-2} + \cdots + b_{n-1} b_0 \; . \tag{1.15}$$

Ez még így is írható:

$$b_n = \sum_{k=0}^{n-1} b_k b_{n-1-k} \; .$$

A fenti rekurzív egyenlet mindkét oldalát z^n-nel szorozva, majd n szerint összegezve, a következőt kapjuk:

$$\sum_{n\geq 1} b_n z^n = \sum_{n\geq 1} \left(\sum_{k=0}^{n-1} b_k b_{n-1-k} \right) z^n \; . \tag{1.16}$$

Legyen $B(z) = \sum_{n\geq 0} b_n z^n$ a b_n számok generátorfüggvénye. Az (1.15) összefüggés bal oldala éppen $B(z) - 1$ (mivel $b_0 = 1$). A jobb oldal nagyon hasonlít két generátorfüggvény szorzatához. Hogy észrevegyük, melyik két függvényről van szó, használjuk a következő jelölést:

$$A(z) = zB(z) = \sum_{n\geq 0} b_n z^{n+1} = \sum_{n\geq 1} b_{n-1} z^n \; .$$

Ekkor az (1.16) jobb oldala éppen $A(z)B(z)$, ami egyenlő $zB^2(z)$-vel. Innen

$$B(z) - 1 = zB^2(z), \qquad B(0) = 1 \; .$$

Oldjuk meg ezt az egyenletet $B(z)$-re. Ekkor

$$B(z) = \frac{1 \pm \sqrt{1-4z}}{2z} .$$

Mivel $B(0) = 1$, csak a negatív jel megfelelő. Tehát

$$\begin{aligned}
B(z) &= \frac{1}{2z}\left(1 - \sqrt{1-4z}\right) = \frac{1}{2z}\left(1 - (1-4z)^{1/2}\right) \\
&= \frac{1}{2z}\left(1 - \sum_{n\geq 0}\binom{1/2}{n}(-4z)^n\right) = \frac{1}{2z}\left(1 - \sum_{n\geq 0}\binom{1/2}{n}(-1)^n 2^{2n} z^n\right) \\
&= \frac{1}{2z} - \binom{1/2}{0}\frac{2^0 z^0}{2z} + \binom{1/2}{1}\frac{2^2 z}{2z} - \cdots - \binom{1/2}{n}(-1)^n \frac{2^{2n} z^n}{2z} + \cdots \\
&= \binom{1/2}{1}2 - \binom{1/2}{2}2^3 z + \cdots - \binom{1/2}{n}(-1)^n 2^{2n-1} z^{n-1} + \cdots \\
&= \sum_{n\geq 0}\binom{1/2}{n+1}(-1)^n 2^{2n+1} z^n = \sum_{n\geq 0}\frac{1}{n+1}\binom{2n}{n} z^n .
\end{aligned}$$

Innen $b_n = \dfrac{1}{n+1}\binom{2n}{n}$.

Megjegyzés. Az utolsó átalakításnál felhasználtuk a következő, könnyen bizonyítható összefüggést:

$$\binom{1/2}{n+1} = \frac{(-1)^n}{2^{2n+1}(n+1)}\binom{2n}{n} .$$

Levelek száma *n* csúcsú bináris fák halmazában

Számítsuk ki az n csúcsú bináris fák halmazában a levelek (azaz első fokú csúcsok) számát. Jelöljük ezt a számot f_n-nel. Megjegyezzük, hogy a gyökeret akkor sem tekintjük levélnek, ha a fokszáma 1. Könnyű belátni, hogy $f_2 = 2$, $f_3 = 6$. Legyen $f_0 = 0$ és $f_1 = 1$ konvenció alapján.

Ahogy a bináris fák megszámolásánál, tekintsük most is az olyan n csúcsú bináris fákat, amelyeknek bal oldala k csúcsot, a jobb oldala pedig $n-k-1$ csúcsot tartalmaz. Bal oldalon b_k ilyen részfa van, jobb oldalon pedig b_{n-1-k}. Ha rögzítünk egy ilyen bal oldali részfát, akkor az összes jobb oldali részfát figyelembe véve, ott f_{n-1-k} levél van. Könnyen belátható tehát, hogy adott k-ra $b_{n-1-k}f_k + b_k f_{n-1-k}$ levél van. Ekkor, összegzés után

$$f_n = \sum_{k=0}^{n-1}\left(f_k b_{n-1-k} + b_k f_{n-1-k}\right) .$$

Egyszerű számítással azt kapjuk, hogy

$$f_n = 2(f_0 b_{n-1} + f_1 b_{n-2} + \cdots + f_{n-1}b_0), \quad n \geq 2 . \tag{1.17}$$

Ez a megoldandó rekurzív egyenlet, amelynek megoldása f_n. Legyen

$$F(z) = \sum_{n\geq 0} f_n z^n \quad \text{és} \quad B(z) = \sum_{n\geq 0} b_n z^n .$$

Az (1.17) összefüggés mindkét oldalát z^n-nel szorozva, majd n szerint összeadva

$$\sum_{n\geq 2} f_n z^n = 2\sum_{n\geq 2}\left(\sum_{k=0}^{n-1} f_k b_{n-1-k}\right) z^n \ .$$

De, mivel $f_0 = 0$ és $f_1 = 1$,

$$F(z) - z = 2zF(z)B(z) \ .$$

Innen

$$F(z) = \frac{z}{1-2zB(z)} \ ,$$

de mivel

$$B(z) = \frac{1}{2z}\left(1 - \sqrt{1-4z}\,\right) \ ,$$

következik, hogy

$$F(z) = \frac{z}{\sqrt{1-4z}} = z(1-4z)^{-1/2} = z\sum_{n\geq 0}\binom{-1/2}{n}(-4z)^n \ .$$

A számítások elvégzése után

$$F(z) = \sum_{n\geq 0}\binom{2n}{n} z^{n+1} = \sum_{n\geq 1}\binom{2n-2}{n-1} z^n \ ,$$

innen pedig

$$f_n = \binom{2n-2}{n-1} \quad \text{vagy} \quad f_{n+1} = \binom{2n}{n} = (n+1)b_n \ .$$

A kombináció általánosítása alapján f_0 és f_1 a konvenció alapján megadott értékekkel lesznek egyenlők.

n csúcsú, k levelű bináris fák száma

Egy kicsit nehezebb feladat: hány n csúcsú k levelű bináris fa létezik? Jelöljük ezek számát $b_n^{(k)}$-val. Könnyű belátni, hogy $b_n^{(k)} = 0$, ha $k > \lfloor (n+1)/2 \rfloor$. Egyszerű okoskodással ki lehet számítani a $k = 1$ esetet, vagyis $b_n^{(1)} = 2^{n-1}$ tetszőleges $n \geq 1$ természetes számra. Legyen $b_0^{(0)} = 1$ konvenció alapján. Akárcsak az előző feladatoknál, itt is a bal és jobb oldali részfákat vizsgáljuk meg. Ha a bal oldali részfában i csúcs és j levél van, akkor a jobb oldaliban $n-i-1$ csúcs és $k-j$ levél van. A $b_i^{(j)} b_{n-i-1}^{(k-j)}$ szorzat éppen ezeknek a fáknak a száma. Összegezve k és j szerint, a következő rekurzív képletet kapjuk:

$$b_n^{(k)} = 2b_{n-1}^{(k)} + \sum_{i=1}^{n-2}\sum_{j=1}^{k-1} b_i^{(j)} b_{n-i-1}^{(k-j)} \ . \tag{1.18}$$

Ennek a rekurzív egyenletnek a megoldására használjuk a következő generátorfüggvényt:

$$B^{(k)}(z) = \sum_{n\geq 0} b_n^{(k)} z^n, \qquad \text{ahol } k \geq 1 \ .$$

Az (1.18) egyenlet mindkét oldalát z^n-nel megszorozva, majd összeadva az $n = 0, 1, 2, \ldots$ értékekre, a következőt kapjuk:

$$\sum_{n\geq 1} b_n^{(k)} z^n = 2\sum_{n\geq 1} b_{n-1}^{(k)} z^n + \sum_{n\geq 1}\left(\sum_{i=1}^{n-2}\sum_{j=1}^{k-1} b_i^{(j)} b_{n-i-1}^{(k-j)}\right) z^n \ .$$

Az összegezés sorrendjét felcserélve

$$\sum_{n\geq 1} b_n^{(k)} z^n = 2\sum_{n\geq 1} b_{n-1}^{(k)} z^n + \sum_{j=1}^{k-1}\sum_{n\geq 1}\left(\sum_{i=1}^{n-2} b_i^{(j)} b_{n-i-1}^{(k-j)}\right) z^n \ .$$

Innen

$$B^{(k)}(z) = 2zB^{(k)}(z) + z\left(\sum_{j=1}^{k-1} B^{(j)}(z)B^{(k-j)}(z)\right)$$

vagy

$$B^{(k)}(z) = \frac{z}{1-2z}\left(\sum_{j=1}^{k-1} B^{(j)}(z)B^{(k-j)}(z)\right) \ . \tag{1.19}$$

Lépésről lépésre haladva, felírhatjuk a következőket.

$$B^{(2)}(z) = \frac{z}{1-2z}\Big(B^{(1)}(z)\Big)^2 \ ,$$

$$B^{(3)}(z) = \frac{2z^2}{(1-2z)^2}\Big(B^{(1)}(z)\Big)^3 \ ,$$

$$B^{(4)}(z) = \frac{5z^3}{(1-2z)^3}\Big(B^{(1)}(z)\Big)^4 \ .$$

Az általános megoldást megpróbáljuk a következő alakban keresni:

$$B^{(k)}(z) = \frac{c_k z^{k-1}}{(1-2z)^{k-1}}\Big(B^{(1)}(z)\Big)^k \ ,$$

ahol, amint láttuk, $c_2 = 1$, $c_3 = 2$, $c_4 = 5$. Az (1.19) képletbe behelyettesítve, a c_k számokra egy rekurzív összefüggést kapunk:

$$c_k = \sum_{i=1}^{k-1} c_i c_{k-i} \ .$$

Ezt szintén a generátorfüggvények segítségével oldjuk meg. Ha $k = 2$, akkor $c_2 = c_1 c_1$, és innen $c_1 = 1$. Legyen $c_0 = 1$. Ha $C(z) = \sum_{n\geq 0} c_n z^n$ a c_n számok generátorfüggvénye, akkor – figyelembe véve a generátorfüggvények szorzási képletét –

$$C(z) - 1 - z = (C(z) - 1)^2 \qquad \text{vagy} \qquad C^2(z) - 3C(z) + z + 2 = 0 \ ,$$

amelyet $C(z)$-re nézve megoldunk, és a

$$C(z) = \frac{3 - \sqrt{1-4z}}{2}$$

képletet kapjuk, mivel $C(0) = 1$ miatt csak a negatív előjel jó. Sorba fejtés után

$$\begin{aligned} C(z) &= \frac{3}{2} - \frac{1}{2}(1-4z)^{1/2} = \frac{3}{2} - \frac{1}{2}\sum_{n\geq 0}\frac{-1}{2n-1}\binom{2n}{n}z^n \\ &= \frac{3}{2} + \sum_{n\geq 0}\frac{1}{2(2n-1)}\binom{2n}{n}z^n = 1 + \sum_{n\geq 1}\frac{1}{2(2n-1)}\binom{2n}{n}z^n \ . \end{aligned}$$

Innen

$$c_n = \frac{1}{2(2n-1)}\binom{2n}{n}, \quad n \geq 1 \ .$$

Mivel $b_n^{(1)} = 2^{n-1}$, ha $n \geq 1$, könnyen ellenőrizhető, hogy $B^{(1)} = z/(1-2z)$. Tehát

$$B^{(k)}(z) = \frac{1}{2(2k-1)}\binom{2k}{k}\frac{z^{2k-1}}{(1-2z)^{2k-1}} \ .$$

Mivel azonban

$$\frac{1}{(1-z)^m} = \sum_{n\geq 0}\binom{n+m-1}{n}z^n \ ,$$

a következő eredményhez jutunk:

$$\begin{aligned} B^{(k)}(z) &= \frac{1}{2(2k-1)}\binom{2k}{k}\sum_{n\geq 0}\binom{2k+n-2}{n}2^n z^{2k+n-1} \\ &= \frac{1}{2(2k-1)}\binom{2k}{k}\sum_{n\geq 2k-1}\binom{n-1}{n-2k+1}2^{n-2k+1}z^n \ . \end{aligned}$$

Innen pedig

$$b_n^{(k)} = \frac{1}{2k-1}\binom{2k}{k}\binom{n-1}{2k-2}2^{n-2k}$$

vagy

$$b_n^{(k)} = \frac{1}{n}\binom{2k}{k}\binom{n}{2k-1}2^{n-2k} \ .$$

1.2.3. A Z-transzformáció módszere

Ha generátorfüggvényekkel oldunk meg egy inhomogén lineáris rekurzív egyenletet, akkor, amint láttuk, mindig egy racionális törtfüggvény sorba fejtése adja meg a megoldást. A Z-transzformáció módszerével ezt a sorba fejtést könnyebben elvégezhetjük. Legyen a racionális törtfüggvény $P(z)/Q(z)$, ahol $P(z)$ kisebb fokszámú, mint $Q(z)$. Ha ismerjük a nevező gyökeit, a törtfüggvényt elemi (vagy parciális) törtfüggvények összegére bonthatjuk a határozatlan együtthatók módszerével. Nézzük meg először azt az esetet, amikor a nevezőnek csak egyszeres (azaz egymástól különböző) gyökei vannak, és legyenek ezek $\alpha_1, \alpha_2, \ldots, \alpha_k$. Ekkor felírhatjuk, hogy

$$\frac{P(z)}{Q(z)} = \frac{A_1}{z-\alpha_1} + \cdots + \frac{A_i}{z-\alpha_i} + \cdots + \frac{A_k}{z-\alpha_k} \ .$$

Könnyen belátható, hogy

$$A_i = \lim_{z\to\alpha_i}(z-\alpha_i)\frac{P(z)}{Q(z)}, \qquad i = 1, 2, \ldots, k\ .$$

Másfelől

$$\frac{A_i}{z-\alpha_i} = \frac{A_i}{-\alpha_i\left(1-\dfrac{1}{\alpha_i}z\right)} = \frac{-A_i\beta_i}{1-\beta_i z}\ ,$$

ahol $\beta_i = 1/\alpha_i$. Ha most ezt az elemi törtet sorba fejtjük, akkor a következőt kapjuk:

$$\frac{-A_i\beta_i}{1-\beta_i z} = -A_i\beta_i\left(1+\beta_i z+\cdots+\beta_i^n z^n+\cdots\right)\ .$$

Innen a z^n együtthatója $-A_i\beta_i^{n+1}$, és jelöljük ezt $C_i(n)$-nel. Ekkor

$$C_i(n) = -A_i\beta_i^{n+1} = -\beta_i\lim_{z\to\alpha_i}(z-\alpha_i)\frac{P(z)}{Q(z)}\ ,$$

vagy

$$C_i(n) = -\beta_i^{n+1}\lim_{z\to\alpha_i}\frac{(z-\alpha_i)P(z)}{Q(z)}\ .$$

Ha most elvégezzük a $z\to 1/z$ átalakítást, és figyelembe vesszük, hogy $\beta_i = 1/\alpha_i$, akkor

$$C_i(n) = \lim_{z\to\beta_i}\left((z-\beta_i)z^{n-1}\frac{p(z)}{q(z)}\right)\ ,$$

ahol

$$\frac{p(z)}{q(z)} = \frac{P(1/z)}{Q(1/z)}\ .$$

Ekkor az $X(z)$ kifejtésében a z^n együtthatója éppen

$$C_1(n) + C_2(n) + \cdots + C_k(n)\ .$$

Könnyen belátható, hogy ha α gyöke a $Q(z)$ polinomnak, akkor $\beta = 1/\alpha$ gyöke a $q(z)$ polinomnak. Például, ha

$$\frac{P(z)}{Q(z)} = \frac{2z^z}{(1-z)(1-2z)}\ , \quad \text{akkor} \quad \frac{p(z)}{q(z)} = \frac{2}{(z-1)(z-2)}\ .$$

Amennyiben egy gyök többszörös, például β_i p-szeres, akkor a neki megfelelő részeredmény

$$C_i(n) = \frac{1}{(p-1)!}\lim_{z\to\beta_i}\frac{d^{p-1}}{dz^{p-1}}\left((z-\beta_i)^p z^{n-1}\frac{p(z)}{q(z)}\right)\ .$$

Itt $\dfrac{d^p}{dz^p}f(z)$ az $f(z)$ függvény p-edrendű deriváltját jelenti.

Az eddigiek a következő algoritmusban összegezhetők. Feltesszük, hogy az egyenlet együtthatóit a A, a megoldás állandóit pedig a C tömb tartalmazza.

LINEÁRIS-INHOMOGÉN(A, k, f)

1 legyen az egyenlet $a_0x_n + a_1x_{n+1} + \cdots + a_kx_{n+k} = f(n)$;
szorozzuk be az egyenlet mindkét oldalát z^n-nel és összegezzünk n szerint
2 hozzuk az egyenletet $X(z) = P(z)/Q(z)$ alakra, ahol $X(z) = \sum_{n\geq 0} x_nz^n$,
$P(z)$ és $Q(z)$ pedig polinomok
3 végezzük el a $z \to 1/z$ átalakítást, legyen az eredmény
$p(z)/q(z)$, ahol $p(z)$ és $q(z)$ polinomok
4 legyenek $q(z)$ gyökei:
β_1 p_1-szeres, $p_1 \geq 1$,
β_2 p_2-szeres, $p_2 \geq 1$,
...
β_k p_k-szeres, $p_k \geq 1$;
ekkor az eredeti egyenlet általános megoldása
$x_n = C_1(n) + C_2(n) + \cdots + C_k(n)$, ahol
$C_i(n) = 1/((p_i - 1)!) \lim_{z\to\beta_i} \frac{d^{p_i-1}}{dz^{p_i-1}} \left((z - \beta_i)^{p_i} z^{n-1} (p(z)/q(z))\right)$, $i = 1, 2, \ldots, k$.
5 **return** C

A módszer neve onnan ered, hogy ha egy generátorfüggvényben z helyébe $1/z$-t helyettesítünk, akkor megkapjuk a Z-transzformáltját, amelyre hasonló műveletek léteznek, mint a generátorfüggvényekre, és amelyre alkalmazva a reziduum-tételt, a fenti eredményhez jutunk.

1.12. példa. Oldjuk meg az

$$x_{n+1} - 2x_n = 2^{n+1} - 2, \quad \text{ha} \quad n \geq 0, \quad x_0 = 0$$

rekurzív egyenletet.

z^n-nel beszorozva és összegezve

$$\sum_{n\geq 0} x_{n+1}z^n - 2\sum_{n\geq 0} x_nz^n = \sum_{n\geq 0} 2^{n+1}z^n - \sum_{n\geq 0} 2z^n ,$$

azaz

$$\frac{1}{z}X(z) - 2X(z) = \frac{2}{1-2z} - \frac{2}{1-z}, \quad \text{ahol } X(z) = \sum_{n\geq 0} x_nz^n .$$

Innen

$$X(z) = \frac{2z^2}{(1-z)(1-2z)^2} .$$

A $z \to 1/z$ helyettesítést elvégezve

$$\frac{p(z)}{q(z)} = \frac{2z}{(z-1)(z-2)^2} ,$$

ahol a nevező gyökei: 1 egyszeres, 2 kétszeres gyök. Ezért

$$C_1 = \lim_{z\to 1} \frac{2z^n}{(z-2)^2} = 2 \qquad \text{és}$$

$$C_2 = \lim_{z\to 2} \frac{d}{dz}\left(\frac{2z^n}{z-1}\right) = 2\lim_{z\to 2} \frac{nz^{n-1}(z-1) - z^n}{(z-1)^2} = 2^n(n-2) .$$

Az általános megoldás tehát

$$x_n = 2^n(n-2) + 2, \quad n \geq 0 .$$

1.13. példa. Oldjuk meg az

$$x_{n+2} = 2x_{n+1} - 2x_n, \quad \text{ha} \quad n \geq 0, \quad x_0 = 1, \quad x_1 = 1$$

rekurzív egyenletet.

z^n-nel beszorozva és összegezve

$$\frac{1}{z^2}\sum_{n\geq 0} x_{n+2}z^{n+2} = \frac{2}{z}\sum_{n\geq 0} x_{n+1}z^{n+1} - 2\sum_{n\geq 0} x_n z^n ,$$

innen

$$\frac{1}{z^2}\Big(F(z) - z\Big) = \frac{2}{z}F(z) - 2F(z) ,$$

azaz

$$F(z)\left(\frac{1}{z^2} - \frac{2}{z} + 2\right) = -\frac{1}{z} .$$

Ekkor

$$F(1/z) = \frac{-z}{z^2 - 2z + 2} .$$

A nevező gyökei $1 + i$ és $1 - i$. Kiszámítjuk $C_1(n)$-t és $C_2(n)$-t:

$$C_1(n) = \lim_{z\to 1+i} \frac{-z^{n+1}}{z - (1-i)} = \frac{i(1+i)^n}{2} \quad \text{és}$$

$$C_2(n) = \lim_{z\to 1-i} \frac{-z^{n+1}}{z - (1+i)} = \frac{-i(1-i)^n}{2} .$$

Mivel

$$1 + i = \sqrt{2}\left(\cos\frac{\pi}{4} + i\sin\frac{\pi}{4}\right), \quad 1 - i = \sqrt{2}\left(\cos\frac{\pi}{4} - i\sin\frac{\pi}{4}\right) ,$$

hatványozás után

$$(1+i)^n = (\sqrt{2})^n\left(\cos\frac{n\pi}{4} + i\sin\frac{n\pi}{4}\right), \quad (1-i)^n = (\sqrt{2})^n\left(\cos\frac{n\pi}{4} - i\sin\frac{n\pi}{4}\right) ,$$

$$x_n = C_1(n) + C_2(n) = \left(\sqrt{2}\right)^n \sin\frac{n\pi}{4} .$$

Gyakorlatok

1.2-1. Számítsuk ki, hány olyan n csúcsú bináris fa van, amelynek sem a bal, sem pedig a jobb oldali részfája nem üres.

1.2-2. Számítsuk ki, hány olyan n csúcsú bináris fa van, amelyben minden levéltől különböző csúcsnak pontosan két gyereke van.

1.2-3. Oldjuk meg generátorfüggvény segítségével az alábbi rekurzív egyenletet.

$$H_n = 2H_{n-1} + 1, \quad H_0 = 0 .$$

(H_n itt a Hanoi-tornyai nevű feladat lépésszámát jelenti.)

1.2-4. Oldjuk meg Z-transzformáció segítségével az alábbi rekurzív egyenletet:

$$F_{n+2} = F_{n+1} + F_n + 1, \text{ ha } n \geq 0, \quad \text{és} \quad F_0 = 0, F_1 = 1 \ .$$

1.2-5. Oldjuk meg a következő egyenletrendszert:

$$\begin{aligned} u_n &= v_{n-1} + u_{n-2} \ , \\ v_n &= u_n + u_{n-1} \ , \end{aligned}$$

ahol $u_0 = 1, u_1 = 2, v_0 = 1$.

1.3. Numerikus megoldás

Leírunk egy függvényeljárást, amellyel lineáris rekurzív egyenleteket oldhatunk meg numerikusan. Az egyenletet a szokásos módon, a következő formában adjuk meg:

$$a_0 x_n + a_1 x_{n+1} + \cdots + a_k x_{n+k} = f(n) \ .$$

Az $a_0, a_1, \ldots, a_k$ együtthatókat az A, míg az $x_0, x_1, \ldots, x_{k-1}$ kezdőértékeket az X vektor tartalmazza. Hogy kiszámítsuk x_n értékét, sorra kiszámítjuk az $x_k, x_{k+1}, \ldots, x_n$ értékeket, minden alkalommal az X vektor első k elemében (azaz a $0, 1, \ldots, k-1$ indexű elemekben) őrizve meg a sorozat előző k értékét.

REKURZÍV(A, X, k, n, f)

```
 1 for j ← k to n
 2     do v ← A[0] · X[0]
 3        for i ← 1 to k − 1
 4            do v ← v + A[i] · X[i]
 5        v ← (f(j − k) − v)/A[k]
 6        if j ≠ n
 7          then for i ← 0 to k − 2
 8                    do X[i] ← X[i + 1]
 9               X[k − 1] ← v
10 return v
```

A 2–5. sorokban kiszámítjuk a következő x_j ($j = k, k+1, \ldots, n$) értékét (az előző k érték felhasználásával), ezt az értéket az algoritmusban v jelöli. A 7–9. sorokban, amennyiben még nem értük el az n-et, átmásoljuk az utolsó k értéket az X vektor első k elemébe. A 10. sor visszaadja az x_n értékét. Könnyen belátható, hogy az algoritmus futási ideje $\Theta(kn)$.

Gyakorlatok

1.3-1. Hány összeadást, kivonást, szorzást, osztást és értékadást végez a REKURZÍV algoritmus, ha az 1.4. példában szereplő adatokkal kiszámítja x_{1000} értékét?

Feladatok

1-1. Homogén egyenlet megoldhatósága generátorfüggvénnyel
Bizonyítsuk be, hogy tetszőleges homogén lineáris rekurzív egyenlet generátorfüggvénnyel való megoldáskor csak akkor fordulhat elő olyan eset, hogy nem tudjuk alkalmazni a megadott módszert, mivel az $X(z) = 0$ egyenlethez jutunk, ha az egyenlet megoldása $x_n = 0$ minden n-re.

1-2. Komplex gyökök Z-transzformáláskor
Vizsgáljuk meg, mi történik, ha a Z-transzformáció módszere alkalmazásakor a nevező gyökei komplex számok. A rekurzív egyenlet megoldásának mindig valósnak kell lennie. Biztosítja-e ezt a módszer?

Megjegyzések a fejezethez

Elaydi [119], Flajolet és Sedgewick [421], Greene és Knuth [175], valamint Mickens [327] könyve részletesen tárgyalja a rekurzív egyenletek megoldását.

Vannak a generátorfüggvényekkel foglalkozó, magyar nyelvű könyvek is – például Knuth [258], valamint Graham, Knuth és Patashnik [171]. Vilenkin műve [478] egyszerű módon tárgyal sok-sok feladatot – a könyv utolsó két fejezete rekurzív egyenletekkel és generátorfüggvényekkel foglalkozik.

Lovász [296] kombinatorikai problémákat és feladatokat tartalmazó könyvében is vannak generátorfüggvényekre vonatkozó feladatok.

A bináris fák megszámolása Knuth [258] könyvéből, a levelek megszámolása a bináris fák halmazában, valamint az n csúcsú k levelű bináris fák megszámolása Kása Zoltán [243] könyvéből valók.

2. Komputeralgebra

A különféle matematikai számítások elvégzésére képes informatikai eszközök nélkülözhetetlenek a modern tudományban és ipari technológiában. Képesek vagyunk kiszámolni bolygók, csillagok pályáját, vezérelni atomerőműveket, egyenletekkel leírni, modellezni a természet számos törvényét. Ezek a számítások alapvetően kétfélék lehetnek: *numerikusak és szimbolikusak*.

Ámbár a numerikus számítások nemcsak elemi aritmetikai műveleteket foglalnak magukban, hanem olyan műveleteket is, mint matematikai függvények numerikus értékének, polinomok gyökeinek vagy mátrixok sajátértékének meghatározása, ezek a műveletek alapvetően számokon értelmezettek, és ezek a számok a legtöbb esetben nem pontosak, pontosságuk az adott számítógépes architektúra lebegőpontos ábrázolási módjától függ. A szimbolikus számításokat matematikai vagy informatikai objektumokat jelentő szimbólumokon értelmezzük. Ezek az objektumok lehetnek számok (egészek, racionális számok, valós és komplex számok, algebrai számok), de lehetnek polinomok, racionális és trigonometrikus függvények, egyenletek, egyenletrendszerek, algebrai struktúrák elemei, vagy éppen halmazok, listák, táblázatok.

A szimbolikus számítások elvégzésére alkalmas számítógépes rendszereket (amelyek legtöbbször numerikus számításokra és az eredmények grafikus megjelenítésére egyaránt képesek) *komputeralgebra-rendszereknek* vagy *szimbolikus-algebrai rendszereknek* nevezzük. Az „algebra” szó a szimbolikus objektumokkal végzett műveletek algebrai eredetére utal.

A komputeralgebra-rendszerek mint *számítógépes programok* alapfeladata: (1) matematikai objektumok szimbolikus ábrázolása, (2) aritmetika ezekkel az objektumokkal. A komputeralgebra, mint *tudományterület* feladata pedig erre az aritmetikára épülő hatékony algoritmusok keresése, elemzése és megvalósítása tudományos kutatásokhoz és alkalmazásokhoz.

Mivel a komputeralgebra-rendszerek szimbolikusan, (lényegében) tetszőleges pontossággal és hibamentesen képesek számolni, először tisztázni kell, milyen adatszerkezeteket lehet hozzárendelni a különféle objektumokhoz. A 2.1. alfejezet a matematikai *objektumok ábrázolásának* problémakörét taglalja. A továbbiakban a szimbolikus algoritmusok közül ismertetjük azokat, melyek az idők folyamán a mindennapi tudomány és technika elengedhetetlen részévé váltak.

A természettudományok többsége jelenségeit, gondolatait matematikai egyenletekkel írja le. A lineáris egyenletek, egyenletrendszerek szimbolikus megoldásainak vizsgálata a jól ismert eliminációs módszereken alapul. A nemlineáris egyenletrendszerek megoldásai-

nak megkeresésére először megvizsgáljuk az *euklideszi algoritmus* különféle változatait és a *rezultánsmódszert*. A hatvanas évek közepén Buchberger doktori dolgozatában egy hatékony módszert dolgozott ki többváltozós polinomegyenletek szimbolikus megoldásainak meghatározására, amit ma *Gröbner-bázis elmélet* néven ismerünk. Buchberger munkájára csak évekkel később kezdtek felfigyelni, azóta a terület a komputeralgebra egyik legnépszerűbb ága. Ezekről lesz szó a 2.2. és a 2.3. alfejezetekben.

A következő bemutatandó terület a *szimbolikus integrálás*. Habár a probléma formális természete már több, mint 100 éve ismert (Liouville-elmélet), csak 1969-ben tudott Risch hatékony algoritmust adni annak eldöntésére, hogy ha adott egy valós elemi f függvény, akkor az $\int f(x)dx$ integrál is elemi-e és ha igen, az algoritmus meg is adja az integrál értékét. A módszerrel a 2.4. alfejezetben foglalkozunk.

A fejezet végén áttekintjük a szimbolikus algoritmusok elméleti és gyakorlati vonatkozásait (2.5. alfejezet), külön részt szánva napjaink komputeralgebra-rendszereinek.

2.1. Adatábrázolás

A komputeralgebrában a legkülönfélébb matematikai objektumok fordulnak elő. Ahhoz, hogy ezekkel az objektumokkal számolni lehessen, ábrázolni és tárolni kell őket a számítógép memóriájában. Ez elméleti és gyakorlati problémák egész sorát veti fel. Ebben az alfejezetben ezekről a kérdésekről lesz szó.

Tekintsük az *egészeket*. Egyrészt matematikai tanulmányainkból tudjuk, hogy halmazuk megszámlálható számosságú, viszont informatikai szemszögből azzal is tisztában vagyunk, hogy számítógépünk csak véges sok egész tárolására képes. Az, hogy mekkora a legnagyobb egész, amit minden további erőfeszítés nélkül ábrázolni tudunk, attól függ, hogy számítógépes architektúránkban mekkora a gépi szó mérete. Ez tipikusan $16, 32, 48$ vagy 64 bit. Az egy gépi szóban ábrázolható egészeket *egyszeres pontosságú egészeknek* nevezzük. Nem biztos, hogy egy tetszőleges egész szám elfér egy gépi szóban, vagyis olyan adatstruktúrára van szükség, ami több gépi szó felhasználásával tetszőlegesen nagy egész ábrázolására képes. Természetesen a „tetszőlegesen nagy” nem jelent „végtelen nagyot”, hiszen valamilyen tervezési igény vagy a memória mérete mindenképpen korlátot szab. Emellett olyan adatábrázolást kell megvalósítani, amelyre hatékony műveletek építhetők. Az egészek reprezentációjának alapvetően két útja van:

- ***helyiértékes*** (a hagyományos decimális számrendszerbeli ábrázolás általánosítása), amelyben egy n egészet $\sum_{i=0}^{k-1} d_i B^i$ alakban írunk fel, ahol a B alapszám akármilyen, egynél nagyobb egész lehet. A hatékonyság növelése miatt B-t úgy érdemes választani, hogy $B-1$ beleférjen egy gépi szóba. A d_i jegyek $(0 \le i \le k-1)$ vagy a kanonikus $\{0 \le d_i \le B-1\}$ vagy a szimmetrikus $\{-\lfloor B/2 \rfloor < d_i \le \lfloor B/2 \rfloor\}$ jegyhalmazból való egyszeres pontosságú egészek. Az így leírható *többszörös pontosságú egészek* lineáris listás $[d_0, d_1, \ldots, d_{k-1}]$ ábrázolásának számítógépes megvalósítása történhet dinamikusan vagy statikusan, attól függően, hogy a lineáris listát láncolt listaként vagy tömbként implementáljuk.

- ***moduláris,*** amelyben az n egész megfelelő számú, egyszeres pontosságú, páronként relatív prím modulusokkal vett moduláris képeinek lineáris listájaként adható meg. A moduláris képekből n a kínai maradéktétel segítségével rekonstruálható.

A moduláris alak gyorsabb az összeadás, kivonás és szorzás műveleteket tekintve, de lényegesen lassabb például oszthatósági vizsgálatoknál (amelyek sok esetben elkerülhetetlenek). Nyilvánvaló, hogy az adatstruktúra megválasztása erősen befolyásolja algoritmusaink sebességét.

2.1. példa. Az alábbi példában az egyszerűség kedvéért természetes számokkal dolgozunk. Tegyük fel, hogy olyan számítógép architektúránk van, ahol a gépi szó 32 bites, vagyis számítógépünk az $I_1 = [0, 2^{32} - 1] = [0, 4\,294\,967\,295]$ intervallum egészeivel képes egész aritmetikát végezni. Erre az aritmetikára építve az architektúránkon valósítsunk meg olyan egész aritmetikát, amellyel az $I_2 = [0, 10^{50}]$ intervallumban is számolni tudunk.

A helyiértékes ábrázoláshoz legyen $B = 10^4$, továbbá

$$\begin{aligned} n_1 &= 12345678909876543210123456789 0 , \\ n_2 &= 2110 . \end{aligned}$$

Ekkor

$$\begin{aligned} n_1 &= [7890, 3456, 1012, 5432, 9876, 7890, 3456, 12] , \\ n_2 &= [2110] , \\ n_1 + n_2 &= [0, 3457, 1012, 5432, 9876, 7890, 3456, 12] , \\ n_1 * n_2 &= [7900, 3824, 6049, 1733, 9506, 9983, 3824, 6049, 2] , \end{aligned}$$

ahol az összeadást és a szorzást helyiértékesen számoltuk.

A moduláris ábrázoláshoz válasszunk páronként relatív prím számokat az I_1 intervallumból úgy, hogy szorzatuk nagyobb legyen 10^{50}-nél. Legyenek például

$$\begin{aligned} m_1 = 4294967291,\ m_2 = 4294967279,\ m_3 = 4294967231 , \\ m_4 = 4294967197,\ m_5 = 4294967189,\ m_6 = 4294967161 \end{aligned}$$

prímek, ahol $\prod_{i=1}^{6} m_i > 10^{50}$. Egy I_2 intervallumbeli egészet tehát az I_1 intervallumból vett számhatossal ábrázolunk.

Ekkor

$$\begin{aligned} n_1 &\equiv 2009436698 \pmod{m_1}, \quad n_1 \equiv 961831343 \pmod{m_2} , \\ n_1 &\equiv 4253639097 \pmod{m_3}, \quad n_1 \equiv 1549708 \pmod{m_4} , \\ n_1 &\equiv 2459482973 \pmod{m_5}, \quad n_1 \equiv 3373507250 \pmod{m_6} , \end{aligned}$$

valamint $n_2 \equiv 2110 \pmod{m_i}$, $(1 \le i \le 6)$, vagyis

$$\begin{aligned} n_1 + n_2 &= [2009438808, 961833453, 4253641207, 1551818, 2459485083, 3373509360] , \\ n_1 * n_2 &= [778716563, 2239578042, 2991949111, 3269883880, 1188708718, 1339711723] , \end{aligned}$$

ahol az összeadás és a szorzás koordinátánként modulárisan elvégezve értendő.

Általánosabban, a matematikai objektumok ábrázolásának három absztrakciós szintjét érdemes megkülönböztetni:

1. *Az objektumok szintje.* Ezen a szinten az objektumok formális matematikai objektumoknak tekinthetők. Például $2+2$, $3*3-5$ és 4 ugyanazt az objektumot jelölik. Hasonlóan, az $(x+1)^2(x-1)$ és $x^3 + x^2 - x - 1$ polinomok az objektumok szintjén azonosnak tekinthetők.

2. *A forma szintje.* Itt már megkülönböztetjük az objektumok eltérő ábrázolásait. Például az $(x+1)^2(x-1)$ és x^3+x^2-x-1 polinomok ugyanannak a polinomnak különböző reprezentációi, hiszen az előbbi egy szorzat, utóbbi egy összeg.
3. *Az adatstruktúra szintje.* Itt a számítógép memóriájában eltérő ábrázolásokat tekintjük különbözőknek. Az x^3+x^2-x-1 polinomnak ezen a szinten többféle reprezentációja is lehet, a polinomot leírhatja például
   - egy együtthatókból álló tömb: $[-1,-1,1,1]$,
   - egy láncolt lista: $[-1,0] \to [-1,1] \to [1,2] \to [1,3]$.

A különböző matematikai objektumok számítógépes ábrázolásához a komputeralgebra-rendszerek tervezőinek dönteniük kell mind a forma, mind az adatstruktúra szintjén. A döntést olyan kérdések nehezítik, mint a reprezentáció memóriaigénye, olvashatósága, vagy az ábrázolás számítási ideje. Például az

$$\begin{aligned} f(x) &= (x-1)^2(x+1)^3(2x+3)^4 \\ &= 16x^9-80x^8+88x^7+160x^6-359x^5+x^4+390x^3-162x^2-135x+81 \end{aligned}$$

polinom szorzat alakja kifejezőbb, mint az összeg alakja, de utóbbi előnyösebb, ha mondjuk az x^5-es tag együtthatójára vagyunk kíváncsiak. A forma szintjére vonatkozó döntési nehézségeket szemlélteti az alábbi példa:

- $x^{1000}-1$ és $(x-1)(x^{999}+x^{998}+\cdots+x+1)$,
- $(x+1)^{1000}$ és $x^{1000}+1000x^{999}+\cdots+1000x+1$.

A matematikai objektumok minden igényt kielégítő ábrázolására tökéletes módszert nem ismerünk. A gyakorlatban az objektumoknak különböző reprezentációi is megengedettek. Ez azt a problémát veti fel, hogy ugyanazon objektum eltérő ábrázolása esetén meg kell tudnunk állapítani azok egyenlőségét, konvertálni kell tudnunk egyik alakból a másikba és az egyértelmű ábrázoláshoz egyszerűsítéseket kell tudnunk azokon végrehajtani. A forma szintjén például minden egész számot felírhatunk valamely B alapú számrendszerben, míg az adatstruktúra szintjén a forma szintjén kapott lineáris listát láncolt listaként vagy tömbként reprezentálhatjuk.

A *racionális számok* egészek párjaiból, a számlálóból és a nevezőből állnak. Memória takarékosság érdekében, valamint két racionális szám könnyű összehasonlíthatósága miatt célszerű a számláló és a nevező legnagyobb közös osztójával egyszerűsített alakot ábrázolni. Mivel az egészek euklideszi gyűrűt alkotnak, a legnagyobb közös osztó euklideszi algoritmussal gyorsan számolható. Az ábrázolás egyértelműségéhez a nevezőt érdemes pozitívnak választani, a racionális szám előjele így a számláló előjele lesz.

A *többváltozós polinomok* (az $R[x_1,x_2,\ldots,x_n]$ n-változós polinomgyűrű elemei, ahol R gyűrű) $a_1x^{e_1}+a_2x^{e_2}+\cdots+a_nx^{e_n}$ alakúak, ahol $a_i \in R\setminus\{0\}$, $e_i=(e_{i_1},\ldots,e_{i_n})$ és x^{e_i}-t írtunk $x_1^{e_{i_1}}x_2^{e_{i_2}}\ldots x_n^{e_{i_n}}$ helyett. A forma szintjén az alábbi reprezentációs lehetőségek adódnak.

1. Kiterjesztett vagy faktorizált ábrázolás, ahol a polinom összegként vagy szorzatként jelenik meg:
   - x^2y-x^2+y-1 ,
   - $\left(x^2+1\right)(y-1)$.

2. Rekurzív vagy disztributív ábrázolás (csak többváltozós esetben). Például kétváltozós polinomgyűrűben $f(x,y)$ tekinthető $R[x,y]$-belinek, $(R[x])[y]$-belinek vagy $(R[y])[x]$-belinek:

   - $x^2y^2 + x^2 + xy^2 - 1$,
   - $(x^2 + x)y^2 + x^2 - 1$,
   - $(y^2 + 1)x^2 + y^2x - 1$.

Az adatstruktúra szintjén ritka vagy teljes reprezentáció lehetséges, például a ritka $x^4 - 1$ polinom teljes ábrázolása $x^4 + 0x^3 + 0x^2 + 0x - 1$. A gyakorlatban a többváltozós polinomok ritka ábrázolása a célravezető.

A $\sum_{i=0}^{\infty} a_i x^i$ alakú *hatványsorok* legegyszerűbb ábrázolása az, ha valamilyen véges rendig adjuk csak meg az együtthatók sorozatát, így lényegében egyváltozós polinomoknak tekintjük őket. Ezzel a megközelítéssel az a probléma, hogy különböző hatványsorokhoz tartozhat ugyanaz a reprezentáció. Ezt elkerülendő, a hatványsort az együtthatói sorozatát generáló függvénnyel szokás leírni. A generáló függvény egy olyan kiszámítható f függvény, amire $f(i) = a_i$. A hatványsorokkal végzett műveletekhez ekkor elegendő azt ismerni, hogy hogyan kell az operandusok együtthatóiból előállítani a művelet eredményét reprezentáló sorozat együtthatóit. Például az f és g hatványsorok szorzatát jelölő h hatványsor együtthatóit a $h_i = \sum_{k=0}^{i} f_k g_{i-k}$ függvény segítségével származtathatjuk. Ekkor a h_i együtthatókat csak akkor kell ténylegesen kiszámolni, ha szükség van az értékükre. Ezt a komputeralgebrában is gyakran használt technikát ***késleltetett kiértékelésnek*** nevezzük.

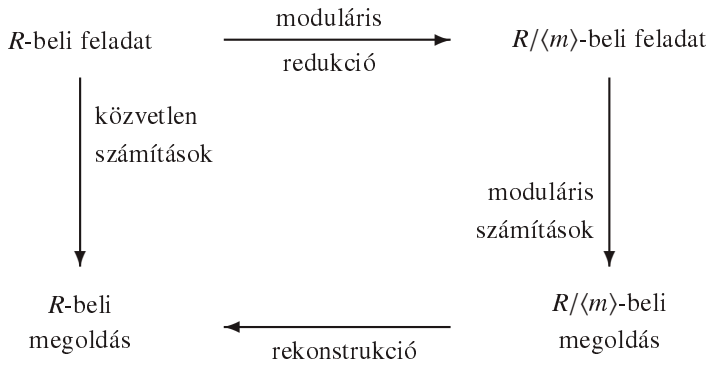
Mivel a komputeralgebra-rendszerek szimbolikusan számolnak, az algoritmusok *műveletigényének* vizsgálatán kívül mindig szükség van *memóriaigényük* vizsgálatára is, hiszen a memóriaigény befolyásolja a tényleges futási időt.[1] Tekintsünk egyszerű példaként egy n ismeretlenes, n egyenletből álló lineáris egyenletrendszert, ahol minden egyes egész együttható elfér a számítógép ω hosszúságú rekeszében. Kizárólag egész aritmetikát és Gauss-eliminációt alkalmazva a redukció eredményeként kapott együtthatók egyenként $2^{n-1}\omega$ tárhelyet igényelhetnek. Ha az együtthatók polinomok lennének és polinomaritmetikát használnánk, az eredmény polinomok együtthatóinak mérete, csakúgy mint a fokszámuk, exponenciális növekedést mutatna. A megfigyelt exponenciális növekedés ellenére a kapott végeredmény mégis „normális” méretű, hiszen a Cramer-szabály miatt a megoldások determinánsok hányadosaiként is megkaphatók, amelyek pedig közelítőleg csak $n\omega$ tárhelyet igényelnek. Ezt a jelenséget nevezzük ***köztes számítási tárrobbanásnak.*** Előfordulása gyakori a komputeralgebra algoritmusokban.

2.2. példa. Egész aritmetikát használva oldjuk meg az alábbi lineáris egyenletrendszert.

$$\begin{aligned} 37x + 22y + 22z &= 1 , \\ 31x - 14y - 25z &= 97 , \\ -11x + 13y + 15z &= -86 . \end{aligned}$$

Először a második egyenlet x változóját eliminálják. Szorozzuk meg az első sort 31-gyel, a másodikat -37-tel és adjuk össze őket. Ha ezt a módszert alkalmazzuk a harmadik egyenlet x változójának

[1] A futási időt mi a RAM-modellnek megfelelően műveletszámban mérjük. Ha Turing-gép modellt és konstans hosszú gépi szavakat használnánk, akkor ilyen probléma nem merül fel, mert a tár mindig alsó korlátja az időnek.



2.1. ábra. A moduláris algoritmusok általános sémája.

eliminációjára, az eredmény az alábbi lesz:

$$\begin{aligned} 37x + 22y + 22z &= 1\,, \\ 1200y + 1607z &= -3558\,, \\ 723y + 797z &= -3171\,. \end{aligned}$$

Most y eliminálásához a második sort 723-mal, a harmadikat -1200-zal szorozzuk, majd összeadjuk őket. Az eredmény:

$$\begin{aligned} 37x + 22y + 22z &= 1\,, \\ 1200y + 1607z &= -3558\,, \\ 205461z &= 1232766\,. \end{aligned}$$

Tovább folytatva az eljárást és sorban eliminálva a változókat végül azt kapjuk, hogy

$$\begin{aligned} 1874311479932400x &= 5622934439797200\,, \\ 246553200y &= -2712085200\,, \\ 205461z &= 1232766\,. \end{aligned}$$

Egyszerűsítés után $x = 3, y = -11, z = 6$ adódik. Természetesen, ha a számítások közben a legnagyobb közös osztókkal egyszerűsítünk, az együtthatók nagysága kevésbé drasztikusan nő.

A számítási tárrobbanás elkerülésére moduláris módszerek használatosak: ahelyett, hogy a számításainkat az R struktúra (pl. euklideszi gyűrű) egészeivel végeznénk, valamely faktorstruktúrában dolgozunk, majd az eredményt „visszatranszformáljuk” R-be (2.1. ábra). A moduláris redukció és a moduláris számítások általában hatékonyan elvégezhetők, a rekonstrukciós lépés pedig valamilyen interpolációs stratégiával történhet. Megjegyezzük, hogy a moduláris algoritmusok nagyon gyakoriak a komputeralgebrában, de nem univerzálisak.

2.2. Polinomok közös gyökei

Legyen R egy integritási tartomány, továbbá legyenek

$$f(x) = f_0 + f_1 x + \cdots + f_{m-1}x^{m-1} + f_m x^m \in R[x],\ f_m \neq 0\ , \tag{2.1}$$
$$g(x) = g_0 + g_1 x + \cdots + g_{n-1}x^{n-1} + g_n x^n \in R[x],\ g_n \neq 0 \tag{2.2}$$

tetszőleges polinomok, $n, m \in \mathbb{N}, n + m > 0$. Állapítsuk meg, hogy mi annak a szükséges és elégséges feltétele, hogy a két polinomnak legyen közös gyöke R-ben.

2.2.1. Klasszikus és bővített euklideszi algoritmus

Ha T test, akkor $T[x]$ euklideszi gyűrű. Emlékeztetőül, az R integritási tartományt euklideszi gyűrűnek nevezzük a $\varphi : R \setminus \{0\} \to \mathbb{N}$ euklideszi függvénnyel, ha bármely $a, b \in R$ $(b \neq 0)$ esetén létezik olyan $q, r \in R$, hogy $a = qb + r$, ahol $r = 0$ vagy $\varphi(r) < \varphi(b)$, továbbá minden $a, b \in R \setminus \{0\}$ esetén $\varphi(ab) \geq \varphi(a)$. A $q = a$ quo b elemet ***hányadosnak,*** az $r = a$ rem b elemet ***maradéknak*** nevezzük. Ha egy R euklideszi gyűrűben dolgozunk, azt szeretnénk, ha a legnagyobb közös osztó egyértelműen meghatározható lenne. Ehhez az R gyűrű egységszorzók által meghatározott ekvivalencia-osztályainak mindegyikéből egyetlen elem kiválasztása szükséges. (Például az egészek $\{0\}, \{-1, 1\}, \{-2, 2\}, \ldots$ osztályaiból mindig a nemnegatívat választjuk.) Így minden $a \in R$ egyértelműen írható fel

$$a = \mathrm{unit}(a) \cdot \mathrm{normal}(a)$$

alakban, ahol normal(a)-t az a elem ***normálalakjának*** nevezzük. Tekintsünk egy T test feletti $R = T[x]$ euklideszi gyűrűt. Ekkor az $a \in R$ elem normálalakja legyen a megfelelő normált főpolinom, vagyis normal$(a) = a/\,\mathrm{lc}(a)$, ahol lc(a) jelenti az a polinom főegyütthatóját. Foglaljuk össze a lényegesebb eseteket:

- ha $R = \mathbb{Z}$, akkor unit$(a) =$ sgn(a) $(a \neq 0)$ és $\varphi(a) =$ normal$(a) = |\,a\,|$,
- ha $R = T[x]$ (T test), akkor unit$(a) =$ lc(a) (az a polinom főegyütthatója a unit$(0) = 1$ megállapodással), normal$(a) = a/\,\mathrm{lc}(a)$ és $\varphi(a) = \deg a$.

Az alábbi algoritmus tetszőleges euklideszi gyűrűben kiszámítja két elem legnagyobb közös osztóját. Megjegyezzük, hogy a világ egyik legősibb algoritmusáról van szó, amit Euklidész már i. e. 300 körül ismert.

KLASSZIKUS-EUKLIDESZ(a, b)

```
1  c ← normal(a)
2  d ← normal(b)
3  while d ≠ 0
4        do r ← c rem d
5           c ← d
6           d ← r
7  return normal(c)
```

Az egészek gyűrűjében a 4. sor maradék képzése $c - \lfloor c/d \rfloor$-t jelenti. Ha $R = T[x]$, ahol T test, akkor a 4. sor maradék képzése az EGYHATÁROZATLANÚ-POLINOMOK-MARADÉKOS-OSZTÁSA(c, d)

iteráció	r	c	d
–	–	18	30
1	18	30	18
2	12	18	12
3	6	12	6
4	0	6	0

(a) KLASSZIKUS-EUKLIDESZ$(-18, 30)$ működése.

iteráció	r	c	d
–	–	$x^4 - \frac{17}{3}x^3 + \frac{13}{3}x^2 - \frac{23}{3}x + \frac{14}{3}$	$x^3 - \frac{20}{3}x^2 + 7x - 2$
1	$4x^2 - \frac{38}{3}x + \frac{20}{3}$	$x^3 - \frac{20}{3}x^2 + 7x - 2$	$4x^2 - \frac{38}{3}x + \frac{20}{3}$
2	$-\frac{23}{4}x + \frac{23}{6}$	$4x^2 - \frac{38}{3}x + \frac{20}{3}$	$-\frac{23}{4}x + \frac{23}{6}$
3	0	$-\frac{23}{4}x + \frac{23}{6}$	0

(b) KLASSZIKUS-EUKLIDESZ$(12x^4 - 68x^3 + 52x^2 - 92x + 56, -12x^3 + 80x^2 - 84x + 24)$ működése.

2.2. ábra. A KLASSZIKUS-EUKLIDESZ algoritmus működésének bemutatása $\mathbb{Z}$-ben és $\mathbb{Q}[x]$-ben. Az **(a)** esetben a bemenő adatok $a = -18, b = 30, a, b \in \mathbb{Z}$. A pszeudokód első két sora a bemenő számok abszolút értékét számolja ki. A harmadik sortól az hatodik sorig tartó ciklus négyszer fut le, a különböző iterációkban számolt r, c és d értékeket mutatja a táblázat. A KLASSZIKUS-EUKLIDESZ$(-18,30)$ algoritmus eredményül a 6-ot szolgáltatja. A **(b)** esetben a bemenő paraméterek $a = 12x^4 - 68x^3 + 52x^2 - 92x + 56, b = -12x^3 + 80x^2 - 84x + 24 \in \mathbb{Q}[x]$. A program első két sora a polinomok normálalakját eredményezi, majd a **while** ciklus háromszor fut le. Az algoritmus kimenete a normal$(c) = x - 2/3$ polinom.

algoritmussal számolható, melynek elemzését az 2.2-1. gyakorlatra hagyjuk. A 2.2. ábra a KLASSZIKUS-EUKLIDESZ működését mutatja $\mathbb{Z}$-ben és $\mathbb{Q}[x]$-ben. Megjegyezzük, hogy $\mathbb{Z}$-ben a program a **while** ciklusba mindig nemnegatív számokkal lép be, a maradék képzés mindig nemnegatív számot eredményez, így a 7. sorban a normalizálás felesleges.

A KLASSZIKUS-EUKLIDESZ algoritmus futási idejének vizsgálata előtt annak egy bővített változatával foglalkozunk.

BŐVÍTETT-EUKLIDESZ(a, b)

```
1  (r_0, u_0, v_0) ← (normal(a), 1, 0)
2  (r_1, u_1, v_1) ← (normal(b), 0, 1)
3  while r_1 ≠ 0
4        do q ← r_0 quo r_1
5           r ← r_0 − q r_1
6           u ← (u_0 − q u_1)
7           v ← (v_0 − q v_1)
8           (r_0, u_0, v_0) ← (r_1, u_1, v_1)
9           (r_1, u_1, v_1) ← (r, u, v)
10 return ( normal(r_0), u_0/(unit(a) · unit(r_0)), v_0/(unit(b) · unit(r_0)))
```

Ismert, hogy az R euklideszi gyűrűben az $a, b \in R$ elemek legnagyobb közös osztója alkalmas $u, v \in R$ elemekkel kifejezhető lnko$(a, b) = au + bv$ alakban. De nem csak egy ilyen számpár létezik. Ha ugyanis u_0, v_0 megfelelők, akkor $u_1 = u_0 + bt$ és $v_1 = v_0 - at$ is azok

minden $t \in R$ esetén:

$$au_1 + bv_1 = a(u_0 + bt) + b(v_0 - at) = au_0 + bv_0 = \text{lnko}(a, b) \ .$$

A Klasszikus-Euklidesz algoritmust úgy egészítettük ki, hogy eredményül ne csak a legnagyobb közös osztót szolgáltassa, hanem az iméntieknek megfelelően egy konkrét $u, v \in R$ számpárt is megadjon.

Legyen $a, b \in R$, ahol R euklideszi gyűrű a φ euklideszi függvénnyel. A Bővített-Euklidesz pszeudokód első két sora kezdeti értékadásainak megfelelően az

$$r_0 = u_0 a + v_0 b \quad \text{és} \quad r_1 = u_1 a + v_1 b \tag{2.3}$$

egyenletek nyilván teljesülnek. Megmutatjuk, hogy a (2.3) egyenlőségek a pszeudokód **while** ciklusának transzformációira invariánsak. Tegyük fel, hogy a ciklus valamely iterációjának végrehajtása előtt a (2.3) feltételek teljesülnek. Ekkor a pszeudokód 4–5. sora szerint

$$r = r_0 - qr_1 = u_0 a + v_0 b - q(au_1 + bv_1) = a(u_0 - qu_1) + b(v_0 - qv_1) \ ,$$

amiből a 6–7. sorok miatt

$$r = a(u_0 - qu_1) + b(v_0 - qv_1) = au + bv.$$

A 8–9. sorok olyan értékadásokat jelentenek, melyben u_0, v_0 felveszi u_1 és v_1, majd u_1, v_1 felveszi u és v értékeit, továbbá r_0, r_1 felveszi r_1 és r értékét. Ezért (2.3) egyenlőségei a **while** ciklus kiértékelése után is teljesülnek. Mivel a ciklus újabb és újabb végrehajtásakor $\varphi(r_1) < \varphi(r_0)$, így a 8–9. sorok értékadásai során keletkezett $\{\varphi(r_i)\}$ sorozat a természetes számok szigorúan monoton csökkenő sorozatát alkotja, ezért a vezérlés előbb utóbb kilép a **while** ciklusból. A legnagyobb közös osztó az algoritmus maradékos osztás sorozatának utolsó nem nulla maradéka, a 8–9. soroknak megfelelően r_0.

2.3. példa. Vizsgáljuk meg a Bővített-Euklidesz algoritmus maradéksorozatát az

$$a(x) = 63x^5 + 57x^4 - 59x^3 + 45x^2 - 8 \ , \tag{2.4}$$

$$b(x) = -77x^4 + 66x^3 + 54x^2 - 5x + 99 \tag{2.5}$$

polinomok esetében:

$$\begin{aligned}
r_0 &= x^5 + \frac{19}{21}x^4 - \frac{59}{63}x^3 + \frac{5}{7}x^2 - \frac{8}{63} \ , \\
r_1 &= x^4 - \frac{6}{7}x^3 - \frac{54}{77}x^2 + \frac{5}{77}x - \frac{9}{7} \ , \\
r_2 &= \frac{6185}{4851}x^3 + \frac{1016}{539}x^2 + \frac{1894}{1617}x + \frac{943}{441} \ , \\
r_3 &= \frac{771300096}{420796475}x^2 + \frac{224465568}{420796475}x + \frac{100658427}{38254225} \ , \\
r_4 &= -\frac{125209969836038125}{113868312759339264}x - \frac{3541728593586625}{101216278008301568} \ , \\
r_5 &= \frac{4717580163635699927436051 21}{1803229860333151158054368 75} \ .
\end{aligned}$$

Az pszeudokód 10. sorának végrehajtása előtt az u_0, v_0 változók értékei:

$$\begin{aligned} u_0 &= \frac{113868312759339264}{12520996983603 8125}x^3 - \frac{662639052858978337856562 24}{819649936515068708206531 25}x^2 \\ &\quad - \frac{172214445262403690128205666 1}{901614930166575579027184375}x + \frac{145175798748706922498167 8954}{901614930166575579027184375}, \\ v_0 &= -\frac{113868312759339264}{12520996983603 8125}x^4 - \frac{650693816081118388788135 36}{819649936515068708206531 25}x^3 \\ &\quad + \frac{178270505434627626751446079}{819649936515068708206531 25}x^2 + \frac{638085922305129542614635 3}{819649936515068708206531 25}x \\ &\quad - \frac{179818001183413133012445617}{819649936515068708206531 25}. \end{aligned}$$

A visszatérési értékek:

$$\begin{aligned} \mathrm{lnko}(a,b) &= 1, \\ u &= \frac{2580775248128}{467729710968369}x^3 - \frac{3823697946464}{779549518280615}x^2 \\ &\quad - \frac{27102209423483}{2338648554841845}x + \frac{7615669511954}{779549518280615}, \\ v &= \frac{703847794944}{155909903656123}x^4 + \frac{3072083769824}{779549518280615}x^3 \\ &\quad - \frac{25249752472633}{2338648554841845}x^2 - \frac{301255883677}{779549518280615}x + \frac{25468935587159}{2338648554841845}. \end{aligned}$$

Láthatjuk, hogy az együtthatók drasztikus növekedést mutatnak. Felvetődik a kérdés: miért nem normalizálunk a **while** ciklus *minden* iterációjában? Ez az ötlet vezet el a polinomok euklideszi algoritmusa normalizált változatához.

Bővített-Euklidesz-normalizált(a, b)

```
 1  e_0 ← unit(a)
 2  (r_0, u_0, v_0) ← (normal(a), e_0^{-1}, 0)
 3  e_1 ← unit(b)
 4  (r_1, u_1, v_1) ← (normal(b), 0, e_1^{-1})
 5  while r_1 ≠ 0
 6        do q ← r_0 quo r_1
 7           s ← r_0 rem r_1
 8           e ← unit(s)
 9           r ← normal(s)
10           u ← (u_0 − q u_1)/e
11           v ← (v_0 − q v_1)/e
12           (r_0, u_0, v_0) ← (r_1, u_1, v_1)
13           (r_1, u_1, v_1) ← (r, u, v)
14  return (r_0, u_0, v_0)
```

2.4. példa. Nézzük meg a Bővített-Euklidesz-normalizált algoritmus során keletkezett maradéksorozatot és az e együtthatósorozatot a korábbi (2.4), (2.5) polinomokra:

$$\begin{aligned}
r_0 &= x^5 + \frac{19}{21}x^4 - \frac{59}{63}x^3 + \frac{5}{7}x^2 - \frac{8}{63}, & e_0 &= 63\ ,\\
r_1 &= x^4 - \frac{6}{7}x^3 - \frac{54}{77}x^2 + \frac{5}{77}x - \frac{9}{7}, & e_1 &= -77\ ,\\
r_2 &= x^3 + \frac{9144}{6185}x^2 + \frac{5682}{6185}x + \frac{10373}{6185}, & e_2 &= \frac{6185}{4851}\ ,\\
r_3 &= x^2 + \frac{2338183}{8034376}x + \frac{369080899}{257100032}, & e_3 &= \frac{771300096}{420796475}\ ,\\
r_4 &= x + \frac{166651173}{5236962760}, & e_4 &= -\frac{222685475860375}{258204790837504}\ ,\\
r_5 &= 1, & e_5 &= \frac{156579848512133360531}{109703115798507270400}\ .
\end{aligned}$$

A pszeudokód 14. sorának végrehajtásakor az $\mathrm{lnko}(a,b) = r_0, u = u_0, v = v_0$ változók értékei:

$$\begin{aligned}
\mathrm{lnko}(a,b) &= 1,\\
u &= \frac{2580775248128}{467729710968369}x^3 - \frac{3823697946464}{779549518280615}x^2\\
&\quad - \frac{27102209423483}{2338648554841845}x + \frac{7615669511954}{779549518280615}\ ,\\
v &= \frac{703847794944}{155909903656123}x^4 + \frac{3072083769824}{779549518280615}x^3\\
&\quad - \frac{25249752472633}{2338648554841845}x^2 - \frac{301255883677}{779549518280615}x + \frac{25468935587159}{2338648554841845}\ .
\end{aligned}$$

$\mathbb{Q}[x]$-ben az együtthatók nagyságát tekintve az euklideszi algoritmus normalizált változatának előnye szembetűnő, de az együtthatók növekedését így sem kerültük el. A Bővített-Euklidesz-normalizált algoritmus gépi architektúra függő leírásához, elemzéséhez bevezetjük az alábbi jelölést. Legyen

$$\begin{aligned}
\lambda(a) &= \lfloor \log_2 |a|/w \rfloor + 1, \text{ ha } a \in \mathbb{Z}\setminus\{0\}, \text{ és } \lambda(0) = 0\ ,\\
\lambda(a) &= \max\{\lambda(b),\lambda(c)\}, \text{ ha } a = b/c \in \mathbb{Q},\ b,c \in \mathbb{Z},\ \mathrm{lnko}(b,c) = 1\ ,\\
\lambda(a) &= \max\{\lambda(b),\lambda(a_0),\ldots,\lambda(a_n)\}, \text{ ha } a = \sum_{0\le i\le n} a_i x^i/b \in \mathbb{Q}[x]\ ,\\
&\quad a_i \in \mathbb{Z},\ b \in \mathbb{N}^+,\ \mathrm{lnko}(b,a_0,\ldots,a_n) = 1\ ,
\end{aligned}$$

ahol w a számítógépes architektúra szóhossza bitekben. Könnyű meggondolni, hogy ha $a,b \in \mathbb{Z}[x]$ és $c,d \in \mathbb{Q}$, akkor

$$\begin{aligned}
\lambda(c+d) &\le \lambda(c) + \lambda(d) + 1\ ,\\
\lambda(a+b) &\le \max\{\lambda(a),\lambda(b)\} + 1\ ,\\
\lambda(cd),\lambda(c/d) &\le \lambda(c) + \lambda(d)\ ,\\
\lambda(ab) &\le \lambda(a) + \lambda(b) + \lambda(\min\{\deg a, \deg b\} + 1)\ .
\end{aligned}$$

Az alábbi tételeket bizonyítás nélkül közöljük.

2.1. tétel. *Ha $a,b \in \mathbb{Z}$ és $\lambda(a) = m \ge n = \lambda(b)$, akkor a* Klasszikus-Euklidesz *és a* Bővített-Euklidesz *algoritmusok $O(mn)$ gépi szóban mért elemi aritmetikai műveletet igényelnek.*

2.2. tétel. *Ha F test, $a, b, \in F[x]$, $\deg(a) = m \geq n = \deg(b)$, akkor a* Klasszikus-Euklidesz, *a* Bővített-Euklidesz *és a* Bővített-Euklidesz-normalizált *algoritmusok $O(mn)$ F-beli elemi aritmetikai műveletet igényelnek.*

Vajon az együtthatók imént látott növekedése pusztán csak a példaválasztásból fakad? Vizsgáljunk meg a Bővített-Euklidesz-Normalizált algoritmusban egyetlen maradékos osztást. Legyen $a = bq + e^*r$, ahol

$$\begin{aligned} a &= x^m + \frac{1}{c}\sum_{i=0}^{m-1} a_i x^i \in \mathbb{Q}[x] \ , \\ b &= x^n + \frac{1}{d}\sum_{i=0}^{n-1} b_i x^i \in \mathbb{Q}[x] \ , \end{aligned}$$

$r \in \mathbb{Q}[x]$ főpolinomok, $a_i, b_i \in \mathbb{Z}$, $e^* \in \mathbb{Q}$, $c, d \in \mathbb{N}^+$, és tekintsük az $n = m - 1$ esetet. Ekkor

$$\begin{aligned} q &= x + \frac{a_{m-1}d - b_{n-1}c}{cd} \ , \\ \lambda(q) &\leq \lambda(a) + \lambda(b) + 1 \ , \\ e^*r &= a - qb = \frac{acd^2 - xbcd^2 - (a_{m-1}d - b_{n-1}c)bd}{cd^2} \ , \\ \lambda(e^*r) &\leq \lambda(a) + 2\lambda(b) + 3 \ . \end{aligned} \tag{2.6}$$

Vegyük észre, hogy a (2.6) becslés az r maradék polinom együtthatóira is érvényes, vagyis $\lambda(r) \leq \lambda(a) + 2\lambda(b) + 3$. Így $\lambda(a) \sim \lambda(b)$ esetén maradékos osztásonként az együtthatók mérete legfeljebb kb. háromszorosára nőhet. Pszeudovéletlen polinomokra a becslés élesnek tűnik, a kísérletezni vágyó Olvasónak ajánljuk a 2-1. feladatot. A legrosszabb esetre kapott becslés azt sejteti, hogy

$$\lambda(r_l) = O(3^l \cdot \max\{\lambda(a), \lambda(b)\}),$$

ahol l jelöli a Bővített-Euklidesz-Normalizált algoritmus futási idejét, vagyis lényegében azt, hogy a **while** ciklus hányszor hajtódik végre. Szerencsére, ez az exponenciális növekedés nem teljesül az algoritmus minden iterációjában, végeredményben pedig az együtthatók növekedése a bemenet függvényében polinomiálisan korlátos. A későbbiekben látni fogjuk, hogy moduláris technika alkalmazásával az együtthatók növekedése teljesen elkerülhető.

Összefoglalva, az euklideszi algoritmus segítségével az $f, g \in R[x]$ (R test) polinomok legnagyobb közös osztóját kiszámítva f-nek és g-nek pontosan akkor van közös gyöke, ha a legnagyobb közös osztójuk nem konstans. Ha ugyanis $\mathrm{lnko}(f, g) = d \in R[x]$ nem konstans, akkor d gyökei f-nek és g-nek is gyökei, hiszen d osztója f-nek és g-nek is. Megfordítva, ha f-nek és g-nek van közös gyöke, akkor a legnagyobb közös osztójuk nem lehet konstans, mert a közös gyök ennek is gyöke.

2.2.2. Primitív euklideszi algoritmus

Amennyiben R euklideszi gyűrű vagy alaptételes gyűrű (amelyben érvényes a számelmélet alaptételének megfelelő állítás, miszerint bármely nem nulla és nem egység elem sorrendtől és egységszorzóktól eltekintve egyértelműen bontható irreducibilis elemek szorzatára), akkor a helyzet bonyolultabb, hiszen $R[x]$-ben nem feltétlenül létezik euklideszi algoritmus. Szerencsére, mégis több módszer kínálkozik, melyek használhatóságának két oka van:

(1) $R[x]$ alaptételes gyűrű, (2) alaptételes gyűrűben két vagy több elem legnagyobb közös osztója mindig létezik.

Az első kínálkozó módszer az, hogy a legnagyobb közös osztó számítását R hányadostestében végezzük el. A $p(x) \in R[x]$ polinomot ***primitív polinomnak*** nevezzük, ha nincs olyan R-beli prím, ami $p(x)$ összes együtthatóját osztaná. Gauss híres lemmája szerint primitív polinomok szorzata is primitív, melynek következménye, hogy f, g primitív polinomok esetén pontosan akkor lesz $d = \text{lnko}(f, g) \in R[x]$, ha $d = \text{lnko}(f, g) \in H[x]$, ahol H jelöli R hányadostestét. Vagyis az $R[x]$-beli legnagyobb közös osztó számítás visszavezethető $H[x]$-belire. Sajnos, ez a megközelítés nem igazán hatékony, mert a H hányadostestben használt aritmetika lényegesen költségesebb, mint az R-beli.

Második lehetőségként egy, az euklideszi algoritmushoz hasonló algoritmus segíthet: integritási tartomány feletti egyhatározatlanú polinomgyűrűben ún. pszeudo-maradékos osztást lehet definiálni. A (2.1), (2.2) polinomokat használva ha $m \geq n$, akkor létezik olyan $q, r \in R[x]$, hogy

$$g_n^{m-n+1} f = gq + r \ ,$$

ahol $r = 0$ vagy $\deg r < \deg g$. A q polinomot az f és g polinomok ***pszeudo-hányadosának,*** az r polinomot ***pszeudo-maradékának*** nevezzük. Jelölésben $q = \text{pquo}(f, g), r = \text{prem}(f, g)$.

2.5. példa. Legyen

$$f(x) = 12x^4 - 68x^3 + 52x^2 - 92x + 56 \in \mathbb{Z}[x] \ , \tag{2.7}$$
$$g(x) = -12x^3 + 80x^2 - 84x + 24 \in \mathbb{Z}[x] \ . \tag{2.8}$$

Ekkor $\text{pquo}(f, g) = -144(x + 1), \ \text{prem}(f, g) = 1152(6x^2 - 19x + 10)$.

Másrészt egységszorzótól eltekintve minden $f(x) \in R[x]$ polinom egyértelműen írható fel

$$f(x) = \text{cont}(f) \cdot \text{pp}(f)$$

alakban, ahol $\text{cont}(f) \in R$ és $\text{pp}(f) \in R[x]$ primitív polinom. Ekkor $\text{cont}(f)$-et f ***összetevőjének,*** $\text{pp}(f)$-et az $f(x)$ polinom ***primitív részének*** nevezzük. A felírások egyértelműsége az egységek normalizálásával érhető el. Például $\mathbb{Z}$-ben az egységek $\{-1, 1\}$ halmazából mindig a pozitívat választjuk.

Az alábbi algoritmus pszeudo-maradékos osztások sorozatát hajtja végre. Az algoritmus felhasználja a pszeudo-maradékot kiszámító prem() függvényt, feltételezi az R-beli legnagyobb közös osztó, valamint az $R[x]$-beli polinomok összetevőjének és primitív részének kiszámíthatóságát. Bemenete az $a, b \in R[x]$ polinomok, ahol R alaptételes gyűrű. Az algoritmus eredménye a $\text{lnko}(a, b) \in R[x]$ polinom.

iteráció	r	c	d
–	–	$3x^4 - 17x^3 + 13x^2 - 23x + 14$	$-3x^3 + 20x^2 - 21x + 6$
1	$108x^2 - 342x + 108$	$-3x^3 + 20x^2 - 21x + 6$	$6x^2 - 19x + 10$
2	$621x - 414$	$6x^2 - 19x + 10$	$3x - 2$
3	0	$3x - 2$	0

2.3. ábra. A Primitív-Euklidesz algoritmus működésének bemutatása az $a(x) = 12x^4 - 68x^3 + 52x^2 - 92x + 56$, $b(x) = -12x^3 + 80x^2 - 84x + 24 \in \mathbb{Z}[x]$ bemenő adatok esetén. A program első két sora a bemeneti polinomok primitív részét számolja ki. A harmadik sortól a hatodik sorig tartó ciklus háromszor fut le, a különböző iterációkban számolt r, c és d értékeket mutatja a táblázat. A program 7. sorában a γ változó értéke $\text{lnko}(4, 4) = 4$. A Primitív-Euklidesz(a, b) algoritmus eredményül $4 \cdot (3x - 2)$-t szolgáltat.

Primitív-Euklidesz(a, b)

```
1  c ← pp(f)
2  d ← pp(g)
3  while d ≠ 0
4        do r ← prem(c, d)
5           c ← d
6           d ← pp(r)
7  γ ← lnko(cont(a), cont(b))
8  δ ← γc
9  return δ
```

Az algoritmus működését a 2.3. ábra szemlélteti. A Primitív-Euklidesz algoritmus futási idejének nagyságrendje megegyezik az euklideszi algoritmus korábban látott változatainak futási idejével.

Primitív-Euklidesz algoritmus azért nagyon lényeges, mert az R test feletti többváltozós $R[x_1, x_2, \ldots x_t]$ polinomgyűrű alaptételes gyűrű, így az algoritmust úgy alkalmazzuk, hogy kiszámoljuk a legnagyobb közös osztót mondjuk $R[x_2 \ldots, x_t][x_1]$-ben, majd rekurzívan az $R[x_3, \ldots, x_t], \ldots, R[x_t]$ alaptételes gyűrűkben. Vagyis a többváltozós polinomgyűrűk rekurzív szemlélete természetes módon vezet a Primitív-Euklidesz algoritmus rekurzív alkalmazásához.

Észrevehetjük, hogy az algoritmus a korábban látottakhoz hasonlóan együttható növekedést mutat.

Vizsgáljuk meg részletesebben a $\mathbb{Z}[x]$ alaptételes gyűrűt. A legnagyobb közös osztó együtthatóinak nagyságára vonatkozó becslést az alábbi, bizonyítás nélkül közölt tétel mutatja.

2.3. tétel (Landau–Mignotte). *Legyen* $a(x) = \sum_{i=0}^{m} a_i x^i$, $b(x) = \sum_{i=0}^{n} b_i x^i \in \mathbb{Z}[x]$, $a_m \neq 0 \neq b_n$, *továbbá* $b(x) \mid a(x)$. *Ekkor*

$$\sum_{i=1}^{n} |b_i| \leq 2^n \left| \frac{b_n}{a_m} \right| \sqrt{\sum_{i=0}^{m} a_i^2} \,.$$

2.4. következmény. *Az előző tétel jelöléseivel az* $\mathrm{lnko}(a,b) \in \mathbb{Z}[x]$ *polinom bármely együtthatója abszolút értékben kisebb, mint*

$$2^{\min\{m,n\}} \cdot \mathrm{lnko}(a_m, b_n) \cdot \min\left\{ \frac{1}{|a_m|} \sqrt{\sum_{i=1}^{m} a_i^2}, \frac{1}{|b_n|} \sqrt{\sum_{i=1}^{n} b_i^2} \right\} .$$

Bizonyítás. Az a és b polinomok legnagyobb közös osztója nyilván osztja a-t és b-t, a foka pedig legfeljebb az a és b polinomok fokainak minimuma. Továbbá a legnagyobb közös osztó főegyütthatója osztója a_m-nek és b_n-nek is, így $\mathrm{lnko}(a_m, b_n)$-nek is. ■

2.6. példa. A 2.4. következmény szerint a (2.4), (2.5) polinomok legnagyobb közös osztója bármely együtthatójának abszolút értéke legfeljebb $\lfloor 32/9\sqrt{3197} \rfloor = 201$, a (2.7), (2.8) polinomok esetében pedig legfeljebb $\lfloor 32\sqrt{886} \rfloor = 952$.

2.2.3. A rezultáns

Az alábbiakban ismertetendő módszer a legáltalánosabb keretek között tárgyalja az (2.1), (2.2) polinomok közös gyökeire vonatkozó szükséges és elégséges feltételeket. További előnye, hogy magasabb fokú algebrai egyenletrendszerek megoldására is alkalmazható.

Legyen tehát R egy integritási tartomány és H a hányadosteste. Tekintsük H-nak azt a legszűkebb K bővítését, melyben a (2.1)-beli $f(x)$ polinom és a (2.2)-beli $g(x)$ polinom is lineáris faktorokra bomlik. Jelöljük az $f(x)$ polinom (K-beli) gyökeit $\alpha_1, \alpha_2, \ldots, \alpha_m$-nel, a $g(x)$ polinom gyökeit pedig $\beta_1, \beta_2, \ldots, \beta_n$-nel. Készítsük el a következő szorzatot:

$$\begin{aligned} \mathrm{res}(f,g) &= f_m^n g_n^m (\alpha_1 - \beta_1)(\alpha_1 - \beta_2) \cdots (\alpha_1 - \beta_n) \\ &\quad \cdot (\alpha_2 - \beta_1)(\alpha_2 - \beta_2) \cdots (\alpha_2 - \beta_n) \\ &\quad \vdots \\ &\quad \cdot (\alpha_m - \beta_1)(\alpha_m - \beta_2) \cdots (\alpha_m - \beta_n) \\ &= f_m^n g_n^m \prod_{i=1}^{m} \prod_{j=1}^{n} (\alpha_i - \beta_j) . \end{aligned}$$

Nyilvánvaló, hogy $\mathrm{res}(f,g)$ akkor és csak akkor lesz 0, ha valamilyen i-re és j-re $\alpha_i = \beta_j$, azaz ha f-nek és g-nek van közös gyöke. Ezt a $\mathrm{res}(f,g)$ szorzatot az f és g polinomok ***rezultánsának*** nevezzük. Vegyük észre, hogy a rezultáns értéke függ az f és g polinomok sorrendjétől, azonban a különböző sorrendben képzett rezultánsok legfeljebb csak előjelben térhetnek el egymástól:

$$\begin{aligned} \mathrm{res}(g,f) &= g_n^m f_m^n \prod_{j=1}^{n} \prod_{i=1}^{m} (\beta_j - \alpha_i) \\ &= (-1)^{mn} f_m^n g_n^m \prod_{i=1}^{m} \prod_{j=1}^{n} (\alpha_i - \beta_j) = (-1)^{mn} \mathrm{res}(f,g) . \end{aligned}$$

A rezultánsnak ez az alakja a gyakorlatban természetesen használhatatlan, mivel a gyökök ismeretét tételezi fel. Vizsgáljuk meg tehát a rezultáns különböző alakjait. Mivel

$$\begin{aligned} f(x) &= f_m(x-\alpha_1)(x-\alpha_2)\cdots(x-\alpha_m) \quad (f_m \neq 0)\,, \\ g(x) &= g_n(x-\beta_1)(x-\beta_2)\cdots(x-\beta_n) \quad (g_n \neq 0)\,, \end{aligned}$$

ezért

$$\begin{aligned} g(\alpha_i) &= g_n(\alpha_i-\beta_1)(\alpha_i-\beta_2)\cdots(\alpha_i-\beta_n) \\ &= g_n\prod_{j=1}^{n}(\alpha_i-\beta_j)\,. \end{aligned}$$

Így

$$\begin{aligned} \operatorname{res}(f,g) &= f_m^n\prod_{i=1}^{m}\Bigl(g_n\prod_{j=1}^{n}(\alpha_i-\beta_j)\Bigr) \\ &= f_m^n\prod_{i=1}^{m}g(\alpha_i) = (-1)^{mn}g_n^m\prod_{j=1}^{n}f(\beta_j)\,. \end{aligned}$$

Ámbár ez az alak sokkal barátságosabb, még mindig feltételezi legalább az egyik polinom gyökeinek ismeretét. Az alábbiakban azt nézzük meg, hogyan lehetne a rezultánst pusztán csak a polinomok együtthatói segítségével kifejezni. Ez a vizsgálat vezet el a rezultáns Sylvester-féle alakjához.

Tegyük fel, hogy a (2.1)-beli f és a (2.2)-beli g polinomoknak van közös gyöke. Ez azt jelenti, hogy van olyan $\alpha \in K$ szám, amelyre

$$\begin{aligned} f(\alpha) &= f_m\alpha^m + f_{m-1}\alpha^{m-1} + \cdots + f_1\alpha + f_0 = 0\,, \\ g(\alpha) &= g_n\alpha^n + g_{n-1}\alpha^{n-1} + \cdots + g_1\alpha + g_0 = 0\,. \end{aligned}$$

A két egyenletet szorozzuk meg rendre az $\alpha^{n-1}, \alpha^{n-2}, \ldots, \alpha, 1$, illetve az $\alpha^{m-1}, \alpha^{m-2}, \ldots, \alpha, 1$ számokkal. Ekkor az első egyenletből n, a második egyenletből m újabb egyenletet nyerünk. Ezt az $m+n$ egyenletet fogjuk úgy fel, mint egy $m+n$ ismeretlenre vonatkozó homogén lineáris egyenletrendszert, melynek $\alpha^{m+n-1}, \alpha^{m+n-2}, \ldots, \alpha, 1$ a megoldása. A megoldás nyilván nem-triviális, hiszen 1 is a gyökök között szerepel. Ismert, hogy az olyan homogén lineáris egyenletrendszernek, amely ugyanannyi egyenletből áll, mint ahány ismeretlent tartalmaz, csak abban az esetben van nemtriviális megoldása, ha a rendszer determinánsa zérus. Vagyis arra jutottunk, hogy f-nek és g-nek csak akkor lehet közös gyöke, ha a

$$D = \begin{vmatrix} f_m & \cdots & \cdots & \cdots & f_0 & & & \uparrow \\ & \ddots & & & & \ddots & & n \\ & & f_m & \cdots & \cdots & \cdots & f_0 & \downarrow \\ g_n & \cdots & \cdots & g_0 & & & & \uparrow \\ & \ddots & & & \ddots & & & m \\ & & \ddots & & & \ddots & & \\ & & & g_n & \cdots & \cdots & g_0 & \downarrow \end{vmatrix} \tag{2.9}$$

determináns nulla (a ki nem írt és nem pontozott helyeken mindenütt nullák állnak). A közös gyök létezésének tehát szükséges feltétele, hogy az $(m+n)$-edrendű D determináns 0 legyen. Az alábbiakban bebizonyítjuk, hogy a D determináns megegyezik az f és g polinomok rezultánsával, amiből az következik, hogy $D = 0$ a közös gyökök létezésének elégséges feltétele is. A (2.9) determinánst nevezzük az f és g polinomok rezultánsa ***Sylvester-féle alakjának.***

2.5. tétel. *A korábbi jelölésekkel*

$$D = f_m^n \prod_{i=1}^{m} g(\alpha_i) \ .$$

Bizonyítás. m-re vonatkozó teljes indukcióval dolgozunk. $m = 0$-ra $f = f_m = f_0$, így a jobb oldal f_0^n. A bal oldalon D egy n-edrendű determináns, melynek a főátlójában csupa f_0 áll, a többi helyen pedig nulla. Így $D = f_0^n$, az állítás tehát igaz. A továbbiakban tegyük fel, hogy $m > 0$ és hogy a bizonyítandó állítás $n-1$-re igaz. Ha tehát f helyett az

$$f^*(x) = f_m(x-\alpha_1)\cdots(x-\alpha_{m-1}) = f^*_{m-1}x^{m-1} + f^*_{m-2}x^{m-2} + \cdots + f^*_1 x + f^*_0$$

polinomot vesszük, akkor f^*ra és g-re az állítás teljesül:

$$D^* = \begin{vmatrix} f^*_{m-1} & \cdots & \cdots & \cdots & f^*_0 & & \\ & \ddots & & & & \ddots & \\ & & f^*_{m-1} & \cdots & \cdots & \cdots & f^*_0 \\ g_n & \cdots & \cdots & g_0 & & & \\ & \ddots & & & \ddots & & \\ & & \ddots & & & \ddots & \\ & & & g_n & \cdots & \cdots & g_0 \end{vmatrix} = f^{*m}_{m-1} \prod_{i=1}^{m-1} g(\alpha_i) \ .$$

Mivel $f = f^*(x-\alpha_m)$, ezért f és f^* együtthatói között az

$$f_m = f^*_{m-1}, f_{m-1} = f^*_{m-2} - f^*_{m-1}\alpha_m, \ldots, f_1 = f^*_0 - f^*_1\alpha_m, f_0 = -f^*_0\alpha_m$$

összefüggések állnak fenn. Így

$$D = \begin{vmatrix} f^*_{m-1} & f^*_{m-2} - f^*_{m-1}\alpha_m & \cdots & \cdots & -f^*_0\alpha_m & & \\ & \ddots & & & & \ddots & \\ & & f^*_{m-1} & \cdots & \cdots & \cdots & -f^*_0\alpha_m \\ g_n & \cdots & \cdots & g_0 & & & \\ & \ddots & & & \ddots & & \\ & & \ddots & & & \ddots & \\ & & & g_n & \cdots & \cdots & g_0 \end{vmatrix} .$$

A determinánst a következőképpen alakítjuk át: az első oszlop α_m-szeresét hozzáadjuk a második oszlophoz, az új második oszlop α_m-szeresét a harmadik oszlophoz stb., végig valamennyi oszlopon. Ezáltal az első n sorból eltűnnek az α_m-ek, vagyis az átalakított D első n sora megegyezik a fenti D^* első n sorával. Az utolsó m sorban az elsőből vonjuk ki a második α_m-szeresét, majd hasonlóan mindegyikből a rákövetkező α_m-szeresét. Végül D-ből az alábbi determináns lesz:

$$D = \begin{vmatrix} f_{m-1}^* & \cdots & \cdots & \cdots & f_0^* & & & \\ & \ddots & & & & \ddots & & \\ & & f_{m-1}^* & \cdots & \cdots & \cdots & f_0^* & \\ g_n & \cdots & \cdots & g_0 & & & & \\ & \ddots & & & \ddots & & & \\ & & \ddots & & & \ddots & & \\ & & & g_n & \cdots & \cdots & g_0 & \\ & & & & g_n & g_n\alpha_m + g_{n-1} & \cdots & g(\alpha_m) \end{vmatrix} .$$

Az utolsó oszlop szerint kifejtve, a $D = D^* g(\alpha_m)$ egyenlőséghez jutunk, amiből az indukciós feltevés alapján $D = f_m^n \prod_{i=1}^m g(\alpha_i)$ következik. ■

Azt kaptuk tehát, hogy $D = \operatorname{res}(f, g)$, vagyis az f és g polinomoknak akkor és csak akkor van közös gyökük K-ban, ha a D determináns eltűnik.

Algoritmikus szempontból magasabb fokú polinomok esetén a rezultáns Sylvester-féle alakjának kiszámolása egy nagy determináns kiszámítását jelenti. Az alábbi tétel szerint a pszeudo-maradékos osztás egyszerűsítheti a számításokat.

2.6. tétel. *A (2.1)-beli f és (2.2)-beli g polinomokra $m \geq n > 0$ esetén*

$$\begin{cases} \operatorname{res}(f, g) = 0, & \text{ha } \operatorname{prem}(f, g) = 0 , \\ g_n^{(m-n)(n-1)+d} \operatorname{res}(f, g) = (-1)^{mn} \operatorname{res}(g, r), & \text{ha } r = \operatorname{prem}(f, g) \neq 0 \text{és } d = \deg(r) . \end{cases}$$

Bizonyítás. A (2.9) determináns első sorát szorozzuk meg g_n^{m-n+1}-gyel. Legyenek $q = q_{m-n}x^{m-n} + \cdots + q_0 \in R[x]$ és $r = r_d x^d + \cdots + r_0 \in R[x]$ azok az egyértelműen meghatározott polinomok, melyekre

$$\begin{aligned} g_n^{m-n+1}(f_m x^m + \cdots + f_0) \quad = \quad & (q_{m-n}x^{m-n} + \cdots + q_0)(g_n x^n + \cdots + g_0) \\ & + r_d x^d + \cdots + r_0 , \end{aligned}$$

ahol $r = \operatorname{prem}(f, g)$. Ekkor a rezultáns $(n+1)$-edik sorát q_{m-n}-nel, az $(n+2)$-edik sorát q_{m-n-1}-gyel stb. szorozva, majd az első sorból kivonva a

$$g_n^{m-n+1}\operatorname{res}(f,g)=\begin{vmatrix}
0 & \cdots & 0 & r_d & \cdots & \cdots & r_0 & & & \\
 & f_m & \cdots & \cdots & \cdots & \cdots & \cdots & f_0 & & \\
 & & \ddots & & & & & & \ddots & \\
 & & & f_m & \cdots & \cdots & \cdots & \cdots & \cdots & f_0 \\
g_n & \cdots & \cdots & \cdots & g_0 & & & & & \\
 & \ddots & & & & \ddots & & & & \\
 & & \ddots & & & & \ddots & & & \\
 & & & \ddots & & & & \ddots & & \\
 & & & & \ddots & & & & \ddots & \\
 & & & & & g_n & \cdots & \cdots & \cdots & g_0
\end{vmatrix}$$

determinánst kapjuk. Itt r_d az első sor $(m-d+1)$-edik oszlopában van, r_0 pedig az első sor $(m+1)$-edik oszlopában.

Hasonló módon folytatva szorozzuk meg a második sort g_n^{m-n+1}-gyel, majd szorozzuk meg az $(n+2)$-edik, $(n+3)$-adik, … sort q_{m-n}-nel, q_{m-n-1}-gyel stb., és vonjuk ki őket a második sorból. Ugyanígy a harmadik, …, n-edik sorra. Az eredmény:

$$g_n^{n(m-n+1)}\operatorname{res}(f,g)=\begin{vmatrix}
 & & & r_d & \cdots & \cdots & r_0 & & & \\
 & & & & \ddots & & & \ddots & & \\
 & & & & & \ddots & & & \ddots & \\
 & & & & & & r_d & \cdots & \cdots & r_0 \\
g_n & \cdots & \cdots & \cdots & g_0 & & & & & \\
 & \ddots & & & & \ddots & & & & \\
 & & \ddots & & & & \ddots & & & \\
 & & & \ddots & & & & \ddots & & \\
 & & & & \ddots & & & & \ddots & \\
 & & & & & g_n & \cdots & \cdots & \cdots & g_0
\end{vmatrix}.$$

Sorcserék után azt kapjuk, hogy

$$g_n^{n(m-n+1)}\operatorname{res}(f,g)=(-1)^{mn}\begin{vmatrix} g_n & \cdots & \cdots & \cdots & g_0 & & & & & \\ & \ddots & & & & \ddots & & & & \\ & & \ddots & & & & \ddots & & & \\ & & & g_n & \cdots & \cdots & \cdots & g_0 & & \\ & & & & \ddots & & & & \ddots & \\ & & & & & g_n & \cdots & \cdots & \cdots & g_0 \\ & & & r_d & \cdots & \cdots & r_0 & & & \\ & & & & \ddots & & & \ddots & & \\ & & & & & \ddots & & & \ddots & \\ & & & & & & r_d & \cdots & \cdots & r_0 \end{vmatrix}.$$

Vegyük észre, hogy

$$\begin{vmatrix} g_n & \cdots & \cdots & \cdots & g_0 & & \\ & \ddots & & & & \ddots & \\ & & g_n & \cdots & \cdots & \cdots & g_0 \\ r_d & \cdots & \cdots & r_0 & & & \\ & \ddots & & & \ddots & & \\ & & \ddots & & & \ddots & \\ & & & r_d & \cdots & \cdots & r_0 \end{vmatrix} = \operatorname{res}(g,r)\ ,$$

ezért

$$g_n^{n(m-n+1)}\operatorname{res}(f,g)=(-1)^{mn}g_n^{m-d}\operatorname{res}(g,r)\ ,$$

amiből

$$g_n^{(m-n)(n-1)+d}\operatorname{res}(f,g)=(-1)^{mn}\operatorname{res}(g,r) \tag{2.10}$$

következik. ■

A (2.10) egyenlet egy nagyon fontos kapcsolatot ír le. Ahelyett, hogy az esetleg óriási méretű $\operatorname{res}(f,g)$ determinánst számítanánk ki, pszeudo-maradékos osztások sorozatát végezzük el, majd minden lépésnél (2.10)-et alkalmazzuk. Csak akkor számoljuk ki a rezultánst, ha már több pszeudo-maradékos osztás nem végezhető el. A tétel fontos következménye az alábbi

2.7. következmény. *Léteznek olyan* $u,v\in R[x]$ *polinomok, melyre* $\operatorname{res}(f,g)=fu+gv$, *ahol* $\deg u<\deg g$, $\deg v<\deg f$.

Bizonyítás. A rezultáns determináns alakjában az i-edik oszlopot szorozzuk meg x^{m+n-i}-vel és adjuk az utolsó oszlophoz minden $i=1,\dots,(m+n-1)$-re. Az eredmény az alábbi lesz:

$$\mathrm{res}(f,g)=\begin{vmatrix} f_m & \cdots & \cdots & f_0 & \cdots & x^{n-1}f \\ & \ddots & & & \ddots & \vdots \\ & & f_m & \cdots & \cdots & f \\ g_n & \cdots & \cdots & g_0 & \cdots & x^{m-1}g \\ & \ddots & & & \ddots & \vdots \\ & & g_n & \cdots & \cdots & g \end{vmatrix} .$$

A determinánst az utolsó oszlopa szerint kifejtve, majd f-et és g-t kiemelve kapjuk az állításban szereplő egyenlőséget a fokokra vonatkozó megszorításokkal. ■

A rezultánsmódszer legfontosabb előnye a korábban látott módszerekhez képest, hogy a bemeneti polinomok szimbolikus együtthatókat is tartalmazhatnak.

2.7. példa. Legyen

$$\begin{aligned} f(x) &= 2x^3-\xi x^2+x+3\in\mathbb{Q}[x]\,, \\ g(x) &= x^2-5x+6\in\mathbb{Q}[x]\,. \end{aligned}$$

Ekkor f és g $\mathbb{Q}$-beli közös gyökeinek létezését az euklideszi algoritmus variánsai segítségével nem tudjuk eldönteni, míg a rezultánsmódszerrel igen: pontosan akkor van közös gyök, ha

$$\mathrm{res}(f,g)=\begin{vmatrix} 2 & -\xi & 1 & 3 & \\ & 2 & -\xi & 1 & 3 \\ 1 & -5 & 6 & & \\ & 1 & -5 & 6 & \\ & & 1 & -5 & 6 \end{vmatrix} = 36\xi^2-429\xi+1260=3(4\xi-21)(3\xi-20)=0\,,$$

vagyis ha $\xi=20/3$, vagy $\xi=21/4$.

A rezultáns jelentősége nemcsak abban áll, hogy segítségével két polinom közös gyökének létezése eldönthető, hanem abban is, hogy használatával algebrai egyenletrendszerek rekurzív módon visszavezethetők egyismeretlenes algebrai egyenletek megoldására.

2.8. példa. Legyen

$$f(x,y) = x^2+xy+2x+y-1\in\mathbb{Z}[x,y]\,, \tag{2.11}$$

$$g(x,y) = x^2+3x-y^2+2y-1\in\mathbb{Z}[x,y]\,. \tag{2.12}$$

Értelmezzük az f és g polinomokat úgy, mint $(\mathbb{Z}[x])[y]$-beli elemeket. Pontosan akkor létezik közös gyökük, ha

$$\mathrm{res}_y(f,g)=\begin{vmatrix} x+1 & x^2+2x-1 & 0 \\ 0 & x+1 & x^2+2x-1 \\ -1 & 2 & x^2+3x-1 \end{vmatrix} = -x^3-2x^2+3x=0\,.$$

$\mathbb{Z}$-beli közös gyökök tehát az $x\in\{-3,0,1\}$ esetben létezhetnek. Minden x-hez (immáron $\mathbb{Z}[y]$-ban) visszahelyettesítéssel megoldjuk a (2.11), (2.12) egyenleteket, amikor is azt kapjuk, hogy az egyenletek egész megoldásai a $(-3,1),(0,1),(1,-1)$ számpárok.

Megjegyezzük, hogy a rezultánsmódszer többváltozós polinomegyenlet-rendszerek megoldásainak megkeresésére is alkalmas, ámbár nem igazán hatékony. Az egyik probléma az, hogy a determináns kiszámítása során számítási tárrobbanás lép fel. Megfigyelhetjük, hogy az egyhatározatlanú m és n-edfokú polinomok rezultánsa determináns alakjának kiszámítása a szokásos Gauss-eliminációval $O((m+n)^3)$-ös műveletigényű, míg az euklideszi algoritmus változatai kvadratikusak. A másik probléma, hogy a számítási bonyolultság erősen függ a határozatlanok sorrendjétől. Sokkal hatékonyabb, ha a polinomegyenlet-rendszer *összes változóját egyszerre eliminálju*k. Ez az út vezet el a többváltozós rezultánsok elméletéhez.

2.2.4. Moduláris legnagyobb közös osztó

A polinomok közös gyökeinek létezésére és meghatározására szolgáló eddigi módszerek mindegyikére jellemző volt a számítási tárrobbanás. Ösztönösen vetődik fel a kérdés: van-e lehetőség moduláris módszerek alkalmazására? Az alábbiakban az $a(x), b(x) \in \mathbb{Z}[x]$ esetet vizsgáljuk ($a, b \neq 0$). Tekintsük a (2.4), (2.5) $\in \mathbb{Z}[x]$ polinomokat és legyen $p = 13$ prím. Ekkor $\mathbb{Z}_p[x]$-ben a KLASSZIKUS-EUKLIDESZ algoritmus maradéksorozata az alábbi lesz:

$$\begin{aligned} r_0 &= 11x^5 + 5x^4 + 6x^3 + 6x^2 + 5 \,, \\ r_1 &= x^4 + x^3 + 2x^2 + 8x + 8 \,, \\ r_2 &= 3x^3 + 8x^2 + 12x + 1 \,, \\ r_3 &= x^2 + 10x + 10 \,, \\ r_4 &= 7x \,, \\ r_5 &= 10 \,. \end{aligned}$$

Azt kapjuk tehát, hogy $\mathbb{Z}_p[x]$-ben az a és b polinomok relatív prímek. Az alábbi tétel a $\mathbb{Z}[x]$-ben és $\mathbb{Z}_p[x]$-ben vett legnagyobb közös osztók közötti kapcsolatot írja le.

2.8. tétel. *Legyen $a, b \in \mathbb{Z}[x]$, $a, b \neq 0$. Legyen p olyan prím, amelyre $p \nmid \mathrm{lc}(a)$ és $p \nmid \mathrm{lc}(b)$. Legyen továbbá $c = \mathrm{lnko}(a, b) \in \mathbb{Z}[x]$, $a_p = a \bmod p$, $b_p = b \bmod p$ és $c_p = c \bmod p$. Ekkor*

(1) $\deg(\mathrm{lnko}(a_p, b_p)) \geq \deg(\mathrm{lnko}(a, b))$,

(2) ha $p \nmid \mathrm{res}(a/c, b/c)$, akkor $\mathrm{lnko}(a_p, b_p) = c_p$.

Bizonyítás. (1) bizonyítása: mivel $c_p \mid a_p$ és $c_p \mid b_p$ ezért $c_p \mid \mathrm{lnko}(a_p, b_p)$. Így

$$\deg(\mathrm{lnko}(a_p, b_p)) \geq \deg(\mathrm{lnko}(a, b) \bmod p).$$

Azonban a feltételek miatt $p \nmid \mathrm{lc}(\mathrm{lnko}(a, b))$, ezért

$$\deg(\mathrm{lnko}(a, b) \bmod p) = \deg(\mathrm{lnko}(a, b)) \,.$$

(2) bizonyítása: mivel $\mathrm{lnko}(a/c, b/c) = 1$, valamint c_p nemtriviális, ezért

$$\mathrm{lnko}(a_p, b_p) = c_p \cdot \mathrm{lnko}(a_p/c_p, b_p/c_p) \,. \tag{2.13}$$

Ha $\mathrm{lnko}(a_p, b_p) \neq c_p$, akkor (2.13) jobb oldala nemtriviális, így $\mathrm{res}(a_p/c_p, b_p/c_p) = 0$. De a rezultáns az együtthatók megfelelő szorzatainak összege, így $p \mid \mathrm{res}(a/c, b/c)$, ami ellentmondás. ■

2.9. következmény. *Véges sok olyan p prím van, amire $p \nmid \mathrm{lc}(a)$, $p \nmid \mathrm{lc}(b)$ és* $\deg(\mathrm{lnko}(a_p, b_p)) > \deg(\mathrm{lnko}(a, b))$. ■

Amikor a 2.8. tétel (1) állításában egyenlőség teljesül, akkor azt mondjuk, hogy p egy „szerencsés prím”. Máris körvonalazhatunk egy legnagyobb közös osztót kiszámoló moduláris algoritmust.

MODULÁRIS-LNKO-NAGYPRÍM(a, b)

```
1  M ← a Landau–Mignotte-konstans (a 2.4. következmény szerint)
2  H ← {}
3  while IGAZ
4        do p ← olyan prím, melyre p ≥ 2M, p ∉ H, p ∤ lc(a) és p ∤ lc(b)
5           c_p ← lnko(a_p, b_p)
6           if c_p | a és c_p | b
7              then return c_p
8              else H ← H ∪ {p}
```

Az algoritmus első sora a Landau–Mignotte-korlát kiszámítását kéri. A negyedik sor szerint olyan „elég nagy” prímet kell választani, amely nem osztja sem a, sem b főegyütthatóját. Az ötödik sor (például a $\mathbb{Z}_p[x]$-beli KLASSZIKUS-EUKLIDESZ algoritmussal) kiszámolja az a és b polinomok legnagyobb közös osztóját modulo p. A kapott polinom együtthatóit szimmetrikus ábrázolással tároljuk. A hatodik sor $c_p \mid a$ és $c_p \mid b$ teljesülését vizsgálja, melynek igazsága esetén c_p a keresett legnagyobb közös osztó. Ha ez nem teljesül, akkor p egy „szerencsétlen prím”, így új prímet választunk. Mivel a 2.8. tétel szerint csak véges sok „szerencsétlen prím” van, ezért az algoritmus előbb-utóbb véget ér. Amennyiben a prímeket megfelelő stratégia szerint választjuk, a H halmaz alkalmazása szükségtelen.

A MODULÁRIS-LNKO-NAGYPRÍM hátránya, hogy a bemeneti polinomok fokszámának növekedtével a Landau–Mignotte-konstans exponenciálisan nő, így ez esetben nagy prímekkel kell számolni. Felmerül a kérdés, hogyan módosítsuk az algoritmust, hogy „sok kis prímmel” számolhassunk? Mivel $\mathbb{Z}_p[x]$-ben a legnagyobb közös osztó az együtthatók konstanssal vett szorzata erejéig egyértelmű, ezért az új algoritmusban ügyelni kell a részpolinomok együtthatóira. Mielőtt tehát a kínai maradéktételt alkalmazzuk a különböző prímekkel vett moduláris legnagyobb közös osztók együtthatóira, minden lépésnél normalizálni kell az $\mathrm{lnko}(a_p, b_p)$ főegyütthatóját. Amennyiben a_m és b_n az a és b polinomok főegyütthatói, akkor $\mathrm{lnko}(a, b)$ főegyütthatója osztja $\mathrm{lnko}(a_m, b_n)$-t. Primitív a és b polinomok esetén $\mathrm{lnko}(a_p, b_p)$ főegyütthatóját ezért $\mathrm{lnko}(a_m, b_n)$ mod p-re normalizáljuk, majd a legvégén vesszük az eredmény polinom primitív részét. Ahogy a MODULÁRIS-LNKO-NAGYPRÍM algoritmusnál, a moduláris számítások eredményét most is szimmetrikus ábrázolással tároljuk. Ezek a meggondolások vezetnek az alábbi, kis prímeket használó moduláris legnagyobb közös osztó algoritmushoz.

MODULÁRIS-LNKO-KISPRÍMEK(a, b)

```
1  d ← lnko(lc(a), lc(b))
2  p ← olyan prím, melyre p ∤ d
3  H ← {p}
```

```
 4  P ← p
 5  c_p ← lnko(a_p, b_p)
 6  g_p ← (d mod p) · c_p
 7  (n, i, j) ← (3, 1, 1)
 8  while IGAZ
 9        do if j = 1
10              then if deg g_p = 0
11                        then return 1
12              (g, j, P) ← (g_p, 0, p)
13           if n ≤ i
14              then g ← pp(g)
15                   if g | a és g | b
16                      then return g
17           p ← olyan prím, melyre p ∤ d és p ∉ H
18           H ← H ∪ {p}
19           c_p ← lnko(a_p, b_p)
20           g_p ← (d mod p) · c_p
21           if deg g_p < deg g
22              then (i, j) ← (1, 1)
23           if j = 0
24              then if deg g_p = deg g
25                        then g_1 = EH-ÉPÍT(g, g_p, P, p)
26                             if g_1 = g
27                                then i ← i + 1
28                                else i ← 1
29                             P ← P · p
30                             g ← g_1
```

EH-ÉPÍT(a, b, m_1, m_2)

```
1  p ← 0
2  c ← 1/m_1 mod m_2
3  for i ← deg a downto 0
4      do r ← a_i mod m_1
5         s ← (b_i − r) mod m_2
6         p ← p + (r + s · m_1)x^i
7  return p
```

Észrevehetjük, hogy a MODULÁRIS-LNKO-KISPRÍMEK algoritmusban nincs szükség annyi kis prímre, mint amennyit a Landau–Mignotte-korlát meghatároz. Amennyiben a g polinom értéke néhány iteráción keresztül nem változik, a 13–16. sorokban teszteljük, hogy g valóban a legnagyobb közös osztó-e. Ezen iterációk számát tárolja a hatodik sor n változója. Megjegyezzük, hogy n értékét a bemeneti polinomoktól függően változtatni is lehetne. Az algoritmusban használt prímeket célszerű egy olyan előre eltárolt listából választani, amely az architektúrának megfelelő gépi szóban elférő prémeket tartalmazza; ilyenkor a H halmaz

használata szükségtelen. A 2.9. következmény miatt a MODULÁRIS-LNKO-KISPRÍMEK algoritmus befejeződik.

Az EH-ÉPÍT algoritmus a bemeneti a, b polinomok azonos fokú tagjainak együtthatóira felírt, modulo m_1 és m_2 vett két lineáris kongruenciából álló egyenletrendszer megoldását számolja ki a kínai maradéktételnek megfelelően. Nagyon fontos, hogy az eredmény polinom együtthatóit szimmetrikus ábrázolással tároljuk.

2.9. példa. Először vizsgáljuk meg MODULÁRIS-LNKO-KISPRÍMEK algoritmus működését a korábban is vizsgált (2.4), (2.5) polinomokra. A könnyebb érthetőség kedvéért kis prímekkel fogunk számolni. Emlékeztetőül,

$$\begin{aligned} a(x) &= 63x^5 + 57x^4 - 59x^3 + 45x^2 - 8 \in \mathbb{Z}[x] , \\ b(x) &= -77x^4 + 66x^3 + 54x^2 - 5x + 99 \in \mathbb{Z}[x] . \end{aligned}$$

Az algoritmus első hat sorának végrehajtása után a $p = 5$ választással $d = 7, c_p = x^2 + 3x + 2$ és $g_p = 2x^2 + x - 1$ lesznek. Mivel a 7. sor miatt a j változó értéke 1, ezért a 10–12. sorok végrehajtódnak. A g_p polinom nem nulla, ezért $g = 2x^2 + x - 1$, $j = 0$, valamint $P = 5$ lesznek a végrehajtás utáni változóértékek. A 13. sorban a feltétel értéke nem teljesül, így újabb prímet választunk. A $p = 7$ rossz választás, a $p = 11$ viszont megengedett. A 19–20. sorok szerint ekkor $c_p = 1$, $g_p = -4$. Mivel $\deg g_p < \deg g$, ezért $j = 1$ lesz és a 25–30. sorok nem hajtódnak végre. A g_p polinom konstans, így a 11. sorban a visszatérési érték 1 lesz, jelezve, hogy az a és b polinomok relatív prímek.

2.10. példa. Második példánkban tekintsük a korábbi

$$\begin{aligned} a(x) &= 12x^4 - 68x^3 + 52x^2 - 92x + 56 \in \mathbb{Z}[x] , \\ b(x) &= -12x^3 + 80x^2 - 84x + 24 \in \mathbb{Z}[x] , \end{aligned}$$

polinomokat. Legyen ismét $p = 5$. Az algoritmus első hat sora után $d = 12, c_p = x + 1, g_p = 2x + 2$. A 10–12. sorok végrehajtása után $P = 5, g = 2x + 2$ lesznek a változók értékei. A következő prím legyen $p = 7$. Így az új értékek $c_p = x + 4, g_p = -2x - 1$. Mivel $\deg g_p = \deg g$, ezért a 25–30. sorok után $P = 35$ és g új értéke $12x - 8$ lesz. Az i változó értéke továbbra is 1. A következő prímet válasszuk 11-nek. Ekkor $c_p = g_p = x + 3$. A g_p és g fokai megegyeznek, így g együtthatóit módosítjuk. Ekkor $g_1 = 12x - 8$ és mivel $g = g_1$, ezért $i = 2$, valamint $P = 385$ lesznek. Az új prím legyen 13. Ekkor $c_p = x + 8, g_p = -x + 5$. A g_p és g fokai továbbra is megegyeznek, ezért a 25–30 sorok végrehajtódnak, a változók értékei $g = 12x - 8, P = 4654, i = 3$ lesznek.

A 17–18. sorok végrehajtása után kiderül, hogy $g \mid a$ és $g \mid b$ ezért a legnagyobb közös osztó $g = 12x - 8$.

Az alábbi tételt bizonyítás nélkül közöljük.

2.10. tétel. *A* MODULÁRIS-LNKO-KISPRÍMEK *algoritmus megfelelően működik. Az algoritmus számítási bonyolultsága* $O(m^3(\log m + \lambda(K))^2)$ *gépi szóban mért művelet, ahol* $m = \min\{\deg a, \deg b\}$, *K pedig az a és b polinomok Landau–Mignotte-korlátja.*

Gyakorlatok

2.2-1. Legyen R egy kommutatív egységelemes gyűrű, $a = \sum_{i=0}^{m} a_i x^i \in R[x]$, $b = \sum_{i=0}^{n} b_i x^i \in R[x]$, továbbá b_n egység, $m \geq n \geq 0$. Az alábbi algoritmus az a és b polinomokkal végzett maradékos osztás eredményeként előállítja azokat a $q, r \in R[x]$ polinomokat, melyekre $a = qb + r$ és $\deg r < n$ vagy $r = 0$.

EGYHATÁROZATLANÚ-POLINOMOK-MARADÉKOS-OSZTÁSA(a, b)

```
1  r ← a
2  for i ← m − n downto 0
3      do if deg r = n + i
4          then q_i ← lc(r)/b_n
5               r ← r − q_i x^i b
6          else q_i ← 0
7  q ← Σ_{i=0}^{m−n} q_i x^i és r
8  return q
```

Bizonyítsuk be, hogy az algoritmus legfeljebb

$$(2 \deg b + 1)(\deg q + 1) = O(m^2)$$

R-beli műveletet igényel.

2.2-2. Mi a különbség $\mathbb{Z}$-ben a BŐVÍTETT-EUKLIDESZ és a BŐVÍTETT-EUKLIDESZ-NORMALIZÁLT algoritmusok között?

2.2-3. Bizonyítsuk be, hogy $\operatorname{res}(f \cdot g, h) = \operatorname{res}(f, h) \cdot \operatorname{res}(g, h)$.

2.2-4. Az $f(x) \in R[x]$ polinom ($\deg f = m$, $\operatorname{lc}(f) = f_m$) ***diszkriminánsának*** a

$$\operatorname{discr} f = \frac{(-1)^{\frac{m(m-1)}{2}}}{f_m} \operatorname{res}(f, f') \in R$$

elemet nevezzük, ahol f' az f x-beli deriváltját jelenti. Az f polinomnak nyilván akkor és csak akkor van többszörös gyöke, ha diszkriminánsa nulla. Számítsuk ki ($\operatorname{discr} f$)-et általános másod- és harmadfokú polinomok esetében.

2.3. Gröbner-bázis

Legyen F test, $R = F[x_1, x_2, \ldots, x_n]$ az F feletti n-határozatlanú polinomok gyűrűje, továbbá legyen $f_1, f_2, \ldots, f_s \in R$. Állapítsuk meg, hogy mi annak a szükséges és elégséges feltétele, hogy az $f_1, f_2, \ldots, f_s$ polinomoknak legyen közös gyöke R-ben. Látható, hogy a probléma az előző alfejezet $s = 2$ esetének bizonyos értelemben vett általánosítása.

Jelölje

$$I = \langle f_1, \ldots, f_s \rangle = \left\{ \sum_{1 \le i \le s} q_i f_i : q_i \in R \right\}$$

az $f_1, \ldots, f_s$ polinomok által generált ideált. Ekkor az $f_1, \ldots, f_s$ polinomok az ***I ideál bázisát*** alkotják. Az I ideál ***varietásán*** a

$$V(I) = \left\{ u \in F^n : f(u) = 0 \text{ minden } f \in I \text{ polinomra} \right\}$$

halmazt értjük. A varietás ismerete természetesen az $f_1, \ldots, f_s$ polinomok közös megoldásainak ismeretét is jelenti. A varietásról, illetve az I ideálról feltehető legfontosabb kérdések:

- $V(I) \neq \emptyset$?

- $V(I)$ „mekkora” ?
- Adott $f \in R$ esetén $f \in I$?
- $I = R$?

Az I ideál Gröbner-bázisa egy olyan bázis, ahol ezeket a kérdéseket könnyű megválaszolni. Mindenekelőtt vizsgáljuk meg az $n = 1$ esetet. Mivel $F[x]$ euklideszi gyűrű, ezért

$$\langle f_1, \ldots, f_s \rangle = \langle \text{lnko}(f_1, \ldots, f_s) \rangle \,. \tag{2.14}$$

Feltehetjük, hogy $s = 2$. Legyen $f, g \in F[x]$, és osszuk el maradékosan f-et g-vel. Ekkor egyértelműen léteznek olyan $q, r \in F[x]$ polinomok, melyekre $f = qg+r$, ahol $\deg r < \deg g$. Vagyis

$$f \in \langle g \rangle \Leftrightarrow r = 0 \,,$$

továbbá $V(g) = \{u_1, \ldots, u_d\}$, amennyiben $x - u_1, \ldots, x - u_d$ a $g \in F[x]$ polinom összes különböző lineáris faktora. Sajnos, a (2.14) egyenlőség két vagy több határozatlan esetén már nem teljesül. Sőt, akármilyen test feletti többváltozós polinomgyűrű nem euklideszi gyűrű, így a maradékos osztás lehetőségét is újra kell gondolni. Ebben az irányban haladunk tovább.

2.3.1. Monomiális rendezés

A ‘$\preceq$‘ $\subseteq \mathbb{N}^n$ teljes rendezési relációt ***megengedettnek*** nevezzük, ha

(i) $(0, \ldots, 0) \preceq v$ minden $v \in \mathbb{N}^n$-re,
(ii) minden $v_1, v_2, v \in \mathbb{N}^n$ esetén $v_1 \preceq v_2 \Rightarrow v_1 + v \preceq v_2 + v$.

Nem nehéz bizonyítani, hogy $\mathbb{N}^n$ minden megengedett rendezése egyben jólrendezés is (vagyis bármely nemüres részhalmazának van legkisebb eleme). A korábbi jelöléseket figyelembe véve tekintsük a

$$T = \{x_1^{i_1} \cdots x_n^{i_n}\}$$

halmazt, melynek elemeit ***monomoknak*** nevezzük. Vegyük észre, hogy T zárt az $F[x_1, \ldots, x_n]$-beli szorzásra, valamint a művelettel kommutatív monoidot alkot. Az $\mathbb{N}^n \to T$, $(i_1, \ldots, i_n) \mapsto x_1^{i_1} \cdots x_n^{i_n}$ leképezés izomorfizmus, ezért egy T-beli $\preceq$ megengedett teljes rendezésre

(i) $1 \preceq t$ minden $t \in T$-re,
(ii) minden $t_1, t_2, t \in T$ esetén $t_1 \prec t_2 \Rightarrow t_1 t \prec t_2 t$.

A T-beli megengedett rendezéseket ***monomiális rendezéseknek*** nevezzük. Tekintsünk néhány példát. Legyen

$$\alpha = x_1^{i_1} \cdots x_n^{i_n},\ \beta = x_1^{j_1} \cdots x_n^{j_n} \in T \,.$$

Ekkor az alábbi rendezéseket szokás definiálni.

- ***Lexikografikus rendezés.***
  $\alpha \prec_{lex} \beta \Leftrightarrow$ létezik olyan $l \in \{1, \ldots, n\}$, hogy $i_l < j_l$ és $i_{l+1} = j_{l+1}, \ldots, i_n = j_n$.
- ***Tiszta lexikografikus rendezés.***
  $\alpha \prec_{plex} \beta \Leftrightarrow$ létezik olyan $l \in \{1, \ldots, n\}$, hogy $i_l < j_l$ és $i_1 = j_1, \ldots, i_{l-1} = j_{l-1}$.

- ***Összfokszám szerint, majd lexikografikus rendezés.***
  $\alpha \prec_{grlex} \beta \Leftrightarrow i_1 + \cdots + i_n < j_1 + \cdots + j_n$ vagy $(i_1 + \cdots + i_n = j_1 + \cdots + j_n$ és $\alpha \prec_{lex} \beta)$.
- ***Összfokszám szerint, majd fordított lexikografikus rendezés.***
  $\alpha \prec_{tdeg} \beta \Leftrightarrow i_1 + \cdots + i_n < j_1 + \cdots + j_n$ vagy $(i_1 + \cdots + i_n = j_1 + \cdots + j_n$ és létezik olyan $l \in \{1, \ldots, n\}$, melyre $i_l > j_l$ és $i_{l+1} = j_{l+1}, \ldots, i_n = j_n)$.

A $\prec_{tdeg}$ monomiális rendezést némely szerzők $\prec_{grrevlex}$-nek jelölik („graded reverse lexicographic order"), hiszen a rendezés a monomok fokszámösszehasonlítás utáni lexikografikus rendezésének felel meg.

2.11. példa. Legyen $<=\prec_{plex}$. Ekkor $z \prec y \prec x$ esetén

$$\begin{aligned} 1 &\prec z \prec z^2 \prec \cdots \prec y \prec yz \prec yz^2 \prec \cdots \\ &\prec y^2 \prec y^2z \prec y^2z^2 \prec \cdots x \prec xz \prec xz^2 \prec \cdots \\ &\prec xy \prec xy^2 \prec \cdots \prec x^2 \prec \cdots . \end{aligned}$$

Legyen $<=\prec_{tdeg}$. Ekkor $z \prec y \prec x$ esetén

$$\begin{aligned} 1 &\prec z \prec y \prec x \\ &\prec z^2 \prec yz \prec xz \prec y^2 \prec xy \prec x^2 \\ &\prec z^3 \prec yz^2 \prec xz^2 \prec y^2z \prec xyz \\ &\prec x^2z \prec y^3 \prec xy^2 \prec x^2y \prec x^3 \prec \cdots . \end{aligned}$$

A továbbiakban mindig feltesszük, hogy valamilyen rögzített monomiális rendezés mellett dolgozunk. Legyen $f = \sum_{\alpha \in \mathbb{N}^n} c_\alpha x^\alpha \in R$ egy nem nulla polinom, $c_\alpha \in F$ és legyen adott egy $\prec$ monomiális rendezés. Ekkor $c_\alpha x^\alpha$ $(c_\alpha \neq 0)$ az f polinom ***tagjai,*** $\mathrm{mdeg}(f) = \max\{\alpha \in \mathbb{N}^n : c_\alpha \neq 0\}$ a polinom ***multifoka*** (a maximum a monomiális rendezés mellett értendő), $\mathrm{lc}(f) = c_{\mathrm{mdeg}(f)} \in F \setminus \{0\}$ a polinom ***főegyütthatója,*** $\mathrm{lm}(f) = x^{\mathrm{mdeg}(f)} \in R$ a polinom főmonomja, $\mathrm{lt}(f) = \mathrm{lc}(f) \cdot \mathrm{lm}(f) \in R$ a polinom ***főtagja.*** Legyenek továbbá $\mathrm{lt}(0) = \mathrm{lc}(0) = \mathrm{lm}(0) = 0$ és $\mathrm{mdeg}(0) = -\infty$.

2.12. példa. Tekintsük az $f(x, y, z) = 2xyz^2 - 3x^3 + 4y^4 - 5xy^2z \in \mathbb{Q}[x, y, z]$ polinomot. Amennyiben $<=\prec_{plex}$ és $z \prec y \prec x$, akkor

$$\mathrm{mdeg}(f) = (3, 0, 0),\ \mathrm{lt}(f) = -3x^3,\ \mathrm{lm}(f) = x^3,\ \mathrm{lc}(f) = -3\ ,$$

ha pedig $<=\prec_{tdeg}$ és $z \prec y \prec x$, akkor

$$\mathrm{mdeg}(f) = (0, 4, 0),\ \mathrm{lt}(f) = 4y^4,\ \mathrm{lm}(f) = y^4,\ \mathrm{lc}(f) = 4\ .$$

2.3.2. Többváltozós polinomok maradékos osztása

Ebben a pontban adott $f, f_1, \ldots, f_s \in R$ többváltozós polinomok és adott $\prec$ monomiális rendezés esetén olyan $q_1, \ldots, q_s \in R$ és $r \in R$ polinomokat keresünk, melyekre $f = q_1f_1 + \cdots + q_sf_s + r$ és r egyetlen monomja sem osztható $\mathrm{lt}(f_1), \ldots, \mathrm{lt}(f_s)$ egyikével sem.

TÖBBHATÁROZATLANÚ-POLINOMOK-MARADÉKOS-OSZTÁSA$(f, f_1, \ldots, f_s)$

```
 1  r ← 0
 2  p ← f
 3  for i ← 1 to s
 4      do q_i ← 0
 5  while p ≠ 0
 6        do if valamilyen i ∈ {1,...,s}-re lt(f_i) | lt(p)
 7              then az egyik ilyen i-re q_i ← q_i + lt(p)/ lt(f_i)
 8                   p ← p − lt(p)/ lt(f_i) f_i
 9              else r ← r + lt(p)
10                   p ← p − lt(p)
11  return q_1,...,q_s és r
```

Az algoritmus helyes működése abból következik, hogy az 5–10. sorok **while** ciklusának minden iterációjában fennállnak az alábbi tulajdonságok:

(i) $\mathrm{mdeg}(p) \preceq \mathrm{mdeg}(f)$ és $f = p + q_1 f_1 + \cdots + g_s f_s + r$,

(ii) $q_i \neq 0 \Rightarrow \mathrm{mdeg}(q_i f_i) \preceq \mathrm{mdeg}(f)$ minden $1 \leq i \leq s$ esetén,

(iii) r egyetlen tagja sem osztója egyik $\mathrm{lt}(f_i)$-nek sem.

Algoritmusunknak van egy gyenge pontja: a többváltozós polinomok maradékos osztása nem egyértelmű. A 7. sorban a megfelelő i értékek közül tetszőlegesen választhatunk.

2.13. példa. Legyen $f = x^2y + xy^2 + y^2 \in \mathbb{Q}[x, y]$, $f_1 = xy - 1$, $f_2 = y^2 - 1$, a monomiális rendezés $\prec_{plex}$, $y \prec_{plex} x$, és a 7. sorban mindig a legkisebb indexű megfelelő i értéket válasszuk. Ekkor az algoritmus eredménye $q_1 = x + y$, $q_2 = 1$, $r = x + y + 1$. De ha f_1 és f_2 szerepét felcseréljük, vagyis $f_1 = y^2 - 1$ és $f_2 = xy - 1$, akkor az algoritmus kimenete $q_1 = x + 1$, $q_2 = x$ és $r = 2x + 1$ lesz.

Az iménti példában látott módon (nevezetesen, hogy az pszeudokód 7. sorában mindig a legkisebb megfelelő pozitív i értéket választjuk) az algoritmus determinisztikussá tehető. Ilyenkor a $q_1, \ldots, q_s$ ***hányadosok*** és az r ***maradék*** egyértelmű, amit úgy jelölünk, hogy $r = f \operatorname{rem} (f_1, \ldots, f_s)$.

Az $s = 1$ esetben az algoritmus választ ad az ideál-tartalmazás problémájára: $f \in \langle f_1 \rangle$ akkor és csak akkor, ha az f polinom f_1 polinommal vett maradékos osztásakor a maradék nulla. Sajnos $s \geq 2$ esetén ez nem teljesül: a $\prec_{plex}$ monomiális rendezéssel

$$xy^2 - x \operatorname{rem} (xy + 1, y^2 - 1) = -x - y,$$

a hányadosok pedig $q_1 = y, q_2 = 0$. Másrészről viszont $xy^2 - x = x \cdot (y^2 - 1) + 0$, ami azt mutatja, hogy $xy^2 - x \in \langle xy + 1, y^2 - 1 \rangle$.

2.3.3. Monomiális ideálok és Hilbert-féle bázistétel

További célunk tetszőleges polinomideálhoz olyan bázis keresése, hogy ezzel a bázissal vett maradékos osztáskor a maradék egyértelmű legyen, így választ tudjunk adni az ideál-tartalmazás problémára. Vajon ilyen bázis egyáltalán létezik? És ha igen, véges elemszámú?

Az $I \subseteq R$ ideált ***monomiális ideálnak*** nevezzük, ha létezik olyan $A \subseteq \mathbb{N}^n$, melyre

$$I = \langle x^A \rangle = \langle \{x^\alpha \in T : \alpha \in A\} \rangle ,$$

vagyis az ideált monomok generálják.

2.11. lemma. *Legyen $I = \langle x^A \rangle \subseteq R$ egy monomiális ideál, és $\beta \in \mathbb{N}^n$. Ekkor*

$$x^\beta \in I \Leftrightarrow \exists \alpha \in A \quad x^\alpha \mid x^\beta \ .$$

Bizonyítás. A $\Leftarrow$ irány nyilvánvaló. Megfordítva, legyen $\alpha_1, \ldots, \alpha_s \in A$ és $q_1, \ldots, q_s \in R$, melyekre $x^\beta = \sum_i q_i x^{\alpha_i}$. Ekkor az összegnek legalább egy olyan $q_i x^{\alpha_i}$ tagja létezik, melyben x^β előfordul, így $x^{\alpha_i} \mid x^\beta$. ■

A lemma legfontosabb következménye, hogy monomiális ideálok pontosan akkor egyeznek meg, ha ugyanazokat a monomokat tartalmazzák.

2.12. lemma (Dickson-lemma). *Minden monomiális ideál végesen generálható, vagyis minden $A \subseteq \mathbb{N}^n$-hez létezik olyan $B \subseteq A$ véges halmaz, melyre $\langle x^A \rangle = \langle x^B \rangle$.*

2.13. lemma. *Legyen I egy ideál $R = F[x_1, \ldots, x_n]$-ben. Ha $G \subseteq I$ egy olyan véges halmaz, melyre $\langle \mathrm{lt}(G) \rangle = \langle \mathrm{lt}(I) \rangle$, akkor $\langle G \rangle = I$.*

Bizonyítás. Legyen $G = \{g_1, \ldots, g_s\}$. Ha $f \in I$ egy tetszőleges polinom, akkor G-vel vett maradékos osztás szerint $f = q_1 g_1 + \cdots + q_s g_s + r$, ahol $q_1, \ldots, q_s, r \in R$ olyanok, hogy vagy $r = 0$, vagy r egyetlen tagja sem osztható egyetlen g_i főtagjával sem. De ekkor $r = f - q_1 g_1 - \cdots - g_s g_s \in I$, és így $\mathrm{lt}(r) \in \mathrm{lt}(I) \subseteq \langle \mathrm{lt}(g_1), \ldots, \mathrm{lt}(g_s) \rangle$. A (2.11) lemma miatt ezért $r = 0$, vagyis $f \in \langle g_1, \ldots, g_s \rangle = \langle G \rangle$. ■

Amennyiben a Dickson-lemmát $\langle \mathrm{lt}(I) \rangle$-re alkalmazzuk, továbbá figyelembe vesszük, hogy a zérópolinom a $\langle 0 \rangle$ ideált generálja, az alábbi nevezetes eredményt kapjuk.

2.14. tétel (Hilbert-féle bázistétel). *Minden $I \subseteq R = F[x_1, \ldots, x_n]$ ideál végesen generálható, vagyis létezik olyan $G \subseteq I$ véges halmaz, melyre $\langle G \rangle = I$ és $\langle \mathrm{lt}(G) \rangle = \langle \mathrm{lt}(I) \rangle$.*

2.15. következmény (ideállánc-feltétel). *Legyen $I_1 \subseteq I_2 \subseteq \cdots$ R-beli ideálok egy növekvő lánca. Ekkor létezik olyan $n \in \mathbb{N}$, melyre $I_n = I_{n+1} = \cdots$.*

Bizonyítás. Legyen $I = \cup_{j \geq 1} I_j$. Ekkor I ideál, ami a Hilbert-féle bázistétel miatt végesen generálható. Legyen $I = \langle g_1, \ldots, g_s \rangle$. Az $n = \min\{j \geq 1 : g_1, \ldots, g_s \in I_j\}$ választással azt kapjuk, hogy $I_n = I_{n+1} = \cdots = I$. ■

Azokat a gyűrűket, melyekben teljesül az ideál-lánc feltétel, ***Noether-gyűrűknek*** nevezzük. Speciálisan, amennyiben F test, akkor $F[x_1, \ldots, x_n]$ Noether-gyűrű.

Legyen $\prec$ egy monomiális rendezés R-en és $I \subseteq R$ egy ideál. A $G \subseteq I$ véges halmazt az I ideál $\prec$ rendezésre vonatkozó ***Gröbner-bázisának*** nevezzük, ha $\langle \mathrm{lt}(G) \rangle = \langle \mathrm{lt}(I) \rangle$. A Hilbert-féle bázistétel következménye az alábbi

2.16. következmény. *$R = F[x_1, \ldots, x_n]$ minden I ideáljának van Gröbner-bázisa.*

Könnyű megmutatni, hogy egy G Gröbner-bázissal vett maradékos osztáskor a maradék nem függ a báziselemek sorrendjétől. Ilyenkor az $f \operatorname{rem} G = r \in R$ jelölés használatos. Az alábbi tétel szerint a Gröbner-bázis segítségével az ideál-tartalmazás problémája egyszerűen megválaszolható.

2.17. tétel. *Legyen G az $I \subseteq R$ ideál $\prec$ monomiális rendezésre vonatkozó Gröbner-bázisa és legyen $f \in R$. Ekkor $f \in I \Leftrightarrow f \operatorname{rem} G = 0$.*

Bizonyítás. Bebizonyítjuk, hogy egyértelműen létezik olyan $r \in R$, amelyre (1) $f - r \in I$, (2) r egyetlen tagja sem osztható $\operatorname{lt}(G)$ egyetlen monomjával sem. Ilyen r létezése a maradékos osztásból következik. Az egyértelműséghez feltesszük, hogy $f = h_1 + r_1 = h_2 + r_2$ valamilyen $h_1, h_2 \in I$-re és r_1 vagy r_2 egyik tagja sem osztható $\operatorname{lt}(G)$ egyetlen monomjával sem. Ekkor $r_1 - r_2 = h_2 - h_1 \in I$, továbbá a 2.11. lemma miatt $\operatorname{lt}(r_1 - r_2)$ osztható $\operatorname{lt}(g)$-vel valamely $g \in G$-re. Ez azt jelenti, hogy $r_1 - r_2 = 0$. ■

Ha tehát G az R egy Gröbner-bázisa, akkor minden $f, g, h \in R$ esetén $g = f \operatorname{rem} G$ és $h = f \operatorname{rem} G \Rightarrow g = h$.

2.3.4. A Buchberger-algoritmus

Észrevehetjük, hogy a Hilbert-féle bázistétel nem konstruktív: nem ad választ arra, hogyan konstruáljuk meg egy I ideál Gröbner-bázisát. Az alábbiakban úgy okoskodunk, hogy azt vizsgáljuk, egy véges halmaz mikor nem Gröbner-bázisa az I ideálnak.

Legyenek $g, h \in R$ nem nulla polinomok, $\alpha = (\alpha_1, \ldots, \alpha_n) = \operatorname{mdeg}(g)$, $\beta = (\beta_1, \ldots, \beta_n) = \operatorname{mdeg}(h)$, $\gamma = (\max\{\alpha_1, \beta_1\}, \ldots, \max\{\alpha_n, \beta_n\})$. A g és h polinomok ***S*-polinomján** az

$$S(g,h) = \frac{x^\gamma}{\operatorname{lt}(g)} g - \frac{x^\gamma}{\operatorname{lt}(h)} h \in R$$

polinomot értjük. Észrevehetjük, hogy $S(g,h) = -S(h,g)$, továbbá $x^\gamma / \operatorname{lt}(g), x^\gamma / \operatorname{lt}(h) \in R$ így $S(g,h) \in \langle g, h \rangle$. A most következő, bizonyítás nélkül közölt tétel egy egyszerű tesztet ad egy halmaz Gröbner-bázis mivoltára.

2.18. tétel. *A $G = \{g_1, \ldots, g_s\} \subseteq R$ halmaz akkor és csak akkor lesz a $\langle G \rangle$ ideál Gröbner-bázisa, ha*

$$S(g_i, g_j) \operatorname{rem} (g_1, \ldots, g_s) = 0 \ \textit{minden}\ 1 \le i < j \le s \ \textit{esetén}\,.$$

Az S-polinomok segítségével könnyen adható Gröbner-bázist konstruáló algoritmus (Buchberger, 1965): adott $f_1, \ldots, f_s \in R = F[x_1, \ldots, x_n]$ és adott $\prec$ monomiális rendezés mellett az alábbi algoritmus megadja az $I = \langle f_1, \ldots, f_s \rangle$ ideál egy $G \subseteq R$ Gröbner-bázisát.

GRÖBNER-BÁZIS$(f_1, \ldots, f_s)$

```
1  G ← {f_1, ..., f_s}
2  P ← {(f_i, f_j) | f_i, f_j ∈ G, i < j, f_i ≠ f_j}
3  while P ≠ ∅
4        do (f, g) ← egy tetszőleges pár P-ből
5           P ← P \ (f, g)
6           r ← S(f, g) rem G
7           if r ≠ 0
8              then G ← G ∪ {r}
9                   P ← P ∪ {(f, r) | f ∈ G}
10 return G
```

Először megmutatjuk, hogy a GRÖBNER-BÁZIS algoritmus helyesen működik. Az algoritmus futásának bármely pillanatában G az I ideál bázisa, hiszen kezdetben az, a továbbiakban pedig kizárólag olyan elemek kerülnek G-be, melyeket G elemeiből képzett S-polinomok G-vel vett maradékos osztásaként kapunk. Amennyiben az algoritmus befejeződik, az összes lehetséges képezhető S-polinom G-vel vett maradéka nulla, így a (2.18) tétel miatt G egy Gröbner-bázis.

Most bebizonyítjuk, hogy az algoritmus befejeződik. Legyenek G és G^* a pszeudokód **while** ciklusa két egymást követő végrehajtásakor kapott halmazok. Nyilván $G \subseteq G^*$, továbbá $\langle \mathrm{lt}(G) \rangle \subseteq \langle \mathrm{lt}(G^*) \rangle$. De a (2.15) következmény miatt az egymást követő iterációk $\langle \mathrm{lt}(G) \rangle$ ideállánca stabilizálódik, vagyis véges számú lépés után $\langle \mathrm{lt}(G) \rangle = \langle \mathrm{lt}(G^*) \rangle$ teljesül. Azt állítjuk, hogy ekkor $G = G^*$. Legyen $f, g \in G$ és $r = S(f, g)\,\mathrm{rem}\; G$. Ekkor $r \in G^*$ és vagy $r = 0$ vagy $\mathrm{lt}(r) \in \langle \mathrm{lt}(G^*) \rangle = \langle \mathrm{lt}(G) \rangle$, amiből a maradék képzésének definíciója miatt $r = 0$ következik.

2.14. példa. Legyen $F = \mathbb{Q}$, $<=<_{plex}$, $z < y < x$, $f_1 = x - y - z$, $f_2 = x + y - z^2$, $f_3 = x^2 + y^2 - 1$. Ekkor a pszeudokód első sora miatt $G = \{f_1, f_2, f_3\}$, a második sorból pedig $P = \{(f_1, f_2), (f_1, f_3), (f_2, f_3)\}$.

A **while** ciklus első végrehajtása során válasszuk az (f_1, f_2) párt. Ekkor $P = \{(f_1, f_3), (f_2, f_3)\}$, $S(f_1, f_2) = -2y - z + z^2$ és $r = f_4 = S(f_1, f_2)\,\mathrm{rem}\; G = -2y - z + z^2$. Ezért $G = \{f_1, f_2, f_3, f_4\}$ és $P = \{(f_1, f_3), (f_2, f_3), (f_1, f_4), (f_2, f_4), (f_3, f_4)\}$.

A ciklus második végrehajtásakor válasszuk az (f_1, f_3) párt. Ekkor $P = P \setminus \{f_1, f_3\}$, $S(f_1, f_3) = -xy - xz - y^2 + 1$, $r = f_5 = S(f_1, f_3)\,\mathrm{rem}\; G = -1/2z^4 - 1/2z^2 + 1$, így $G = \{f_i \mid 1 \le i \le 5\}$ és $P = \{(f_2, f_3), (f_1, f_4), \ldots, (f_3, f_4), (f_1, f_5), \ldots, (f_4, f_5)\}$.

A ciklus harmadik végrehajtása során válasszuk az (f_2, f_3) párt. Ekkor $P = P \setminus \{(f_2, f_3)\}$, $S(f_2, f_3) = xy - xz^2 - y^2 + 1$, $r = S(f_2, f_3)\,\mathrm{rem}\; G = 0$.

A **while** ciklus negyedik végrehajtása során válasszuk az (f_1, f_4) párt. Ekkor $P = P \setminus \{f_1, f_4\}$, $S(f_1, f_4) = 2y^2 + 2yz + xz - xz^2$, $r = S(f_1, f_4)\,\mathrm{rem}\; G = 0$.

Hasonlóan, az összes fennmaradó pár S-polinomjának G-vel vett maradékos osztásakor a maradék 0, így a visszatérő érték $G = \{x - y - z, x + y - z^2, x^2 + y^2 - 1, -2y - z + z^2, -1/2z^4 - 1/2z^2 + 1\}$ egy Gröbner-bázis.

2.3.5. Redukált Gröbner-bázis

A Buchberger-algoritmus által eredményezett Gröbner-bázis általában nem minimális és nem egyértelmű. Szerencsére egy kis ravaszkodással mindkettő elérhető.

2.19. lemma. *Ha G az $I \subseteq R$ ideál egy Gröbner-bázisa és* $\mathrm{lt}(g) \in \langle \mathrm{lt}(G \setminus \{g\})\rangle$*, akkor* $G \setminus \{g\}$ *is egy Gröbner-bázisa I-nek.*

Azt mondjuk, hogy a $G \subseteq R$ halmaz az $I = \langle G\rangle$ ideál ***minimális Gröbner-bázisa,*** ha Gröbner-bázis és minden $g \in G$ esetén

- $\mathrm{lc}(g) = 1$,
- $\mathrm{lt}(g) \notin \langle \mathrm{lt}(G \setminus \{g\})\rangle$.

A G Gröbner-bázis egy $g \in G$ eleme ***redukált*** a G-re nézve, ha g egyetlen monomja sincs az $\langle \mathrm{lt}(G \setminus \{g\})\rangle$ ideálban. Egy minimális G Gröbner-bázis redukált, ha G-re vonatkozóan minden eleme redukált.

2.20. tétel. *Minden ideálhoz egyértelműen létezik egy redukált Gröbner-bázis.*

2.15. példa. A 2.14. példát alapul véve nemcsak G, hanem $G' = \{x-y-z, -2y-z+z^2, -1/2z^4-1/2z^2+1\}$ is Gröbner-bázis. Nem nehéz megmutatni, hogy $G_r = \{x - 1/2z^2 - 1/2z, y - 1/2z^2 - 1/2z, z^4 + z^2 - z\}$ redukált Gröbner-bázis.

2.3.6. A Gröbner-bázis számítási bonyolultsága

A Gröbner-bázis elmélet kialakulása óta eltelt fél évszázad sem volt elég teljesen tisztázni az algoritmus számítási bonyolultságát. A tapasztalatok azt mutatják, hogy számítási tárrobbanással állunk szemben. De most, ellentétben az euklideszi algoritmus variánsainál látott számítási tárrobbanással, a növekedés ütemét nem lehet kordában tartani. A Gröbner-bázis számításnál fellépő számítási bonyolultságot szokás az *EXPSPACE*-teljes osztályba sorolni. Legyenek $f, f_1, \ldots, f_s \in F[x_1, \ldots, x_n]$ az F test feletti legfeljebb d-ed fokú polinomok ($<=\prec_{tdeg}$). Ha $f \in \langle f_1, f_2, \ldots f_s\rangle$, akkor

$$f = f_1 g_1 + \cdots + f_s g_s$$

olyan $g_1, \ldots, g_s \in F[x_1, \ldots, x_n]$ polinomokra, melyek fokai $\beta = \beta(n, d) = (2d)^{2^n}$-nel felülről korlátosak. Ez a duplán exponenciális korlát lényegében elkerülhetetlen, amit számos példa bizonyít. Sajnos, az $F = \mathbb{Q}$ esetben az ideál-tartalmazás probléma ilyen. Szerencsére, speciális esetekben drasztikus redukció érhető el. Ha $f = 1$ (a Hilbert-féle Nullstellensatz), akkor a $d = 2$ esetben $\beta = 2^{n+1}$, a $d > 2$ esetben pedig $\beta = d^n$. Márpedig a $V(f_1, \ldots, f_s)$ varietás pontosan akkor üres, ha $1 \in \langle f_1, f_2, \ldots f_s\rangle$, vagyis a polinomegyenlet-rendszerek megoldhatósága *PSPACE*-beli probléma. Számos eredmény szól amellett, hogy bizonyos feltételek mellett az (általános) ideál-tartalmazás probléma is *PSPACE*-beli. Ilyen feltétel például, hogy $\langle f_1, f_2, \ldots f_s\rangle$ zéró-dimenziós.

Ezen számítási bonyolultság ellenére a Gröbner-bázis elmélet komoly sikereket könyvelhet el: automatikus geometriai tételbizonyítás, robotok mozgásvezérlése és polinomegyenlet-rendszerek megoldása talán a legelterjedtebb alkalmazási területek. Az alábbiakban felsoroljuk azokat az területeket, ahol az elmélet sikeresen alkalmazható.

- *Polinomegyenletek ekvivalenciája.* Két polinomhalmaz pontosan akkor generálja ugyanazt az ideált, ha Gröbner-bázisuk megegyezik (tetszőleges monomiális rendezés mellett).

- *Polinomegyenletek megoldhatósága.* Az $f_i(x_1,\dots,x_n)=0,\ 1\le i\le s$ egyenletrendszer pontosan akkor oldható meg, ha az $1\notin\langle f_1,\dots,f_s\rangle$.
- *Polinomegyenletek megoldáshalmaza végessége.* Az $f_i(x_1,\dots,x_n)=0,\ 1\le i\le s$ egyenletrendszernek pontosan akkor van véges számú megoldása, ha $\langle f_1,\dots,f_s\rangle$-ben minden x_i változóhoz van olyan polinom, hogy az adott monomiális rendezés melletti főtagja x_i valamilyen hatványa.
- *Polinomegyenletek véges megoldásai száma.* Legyen az $f_i(x_1,\dots,x_n)=0,\ 1\le i\le s$ egyenletrendszernek véges sok megoldása. Ekkor multiplicitással számolva a polinomegyenlet megoldásszáma azon monomok halmazának számossága, melyek nem többszörösei a $\langle f_1,\dots,f_s\rangle$ Gröbner-bázisa elemei egyetlen főtagjának sem (tetszőleges monomiális rendezés mellett).
- **Kifejezések egyszerűsítése.**

A legutóbbira példát is mutatunk.

2.16. példa. Legyenek $a,b,c\in\mathbb{R}$ olyanok, hogy

$$a+b+c=3,\ \ a^2+b^2+c^2=9,\ \ a^3+b^3+c^3=24\ .$$

Számítsuk ki $a^4+b^4+c^4$ értékét. Legyenek tehát $f_1=a+b+c-3$, $f_2=a^2+b^2+c^2-9$ és $a^3+b^3+c^3-24$ az $\mathbb{R}[a,b,c]$ elemei, továbbá legyen $\prec=\prec_{plex}$, $c\prec b\prec a$. Ekkor az $\langle f_1,f_2,f_3\rangle$ Gröbner-bázisa

$$G=\{a+b+c-3,\ b^2+c^2-3b-3c+bc,\ 1-3c^2+c^3\}\ .$$

Így $a^4+b^4+c^4$ rem $G=69$, ami a feladat megoldása.

Gyakorlatok

2.3-1. Bizonyítsuk be, hogy a $\prec_{lex},\prec_{plex},\prec_{grlex}$ és $\prec_{tdeg}$ monomiális rendezések valóban megengedettek.

2.3-2. Legyen $\prec$ egy monomiális rendezés R-en, $f,g\in R\setminus\{0\}$. Lássuk be az alábbiakat:

(a) $\mathrm{mdeg}(fg)=\mathrm{mdeg}(f)+\mathrm{mdeg}(g)$,

(b) ha $f+g\neq 0$, akkor $\mathrm{mdeg}(f+g)\le\max\{\mathrm{mdeg}(f),\mathrm{mdeg}(g)\}$, ahol $\mathrm{mdeg}(f)\neq\mathrm{mdeg}(g)$ esetén egyenlőség teljesül.

2.3-3. Legyen $f=2x^4y^2z-3x^4yz^2+4xy^4z^2-5xy^2z^4+6x^2y^4z-7x^2yz^4\in\mathbb{Q}[x,y,z]$.

(a) Rendezzük a polinom tagjait $\prec_{plex},\prec_{grlex}$ és $\prec_{tdeg}$ monomiális rendezések esetén, ahol mindhárom esetben $z\prec y\prec x$.

(b) Mindhárom monomiális rendezés esetén határozzuk meg $\mathrm{mdeg}(f)$, $\mathrm{lc}(f)$, $\mathrm{lm}(f)$ és $\mathrm{lt}(f)$-et.

2.3-4.$\star$ Bizonyítsuk be a Dickson-lemmát.

2.3-5. Számítsuk ki az $I=\langle x^2+y-1,xy-x\rangle\subseteq\mathbb{Q}[x,y]$ ideál Gröbner-bázisát és redukált Gröbner-bázisát a $\prec=\prec_{lex}$ monomiális rendezés esetén, ahol $y\prec x$. Határozzuk meg, hogy az alábbi polinomok közül melyik eleme az ideálnak: $f_1=x^2+y^2-y$, $f_2=3xy^2-4xy+x+1$.

2.4. Szimbolikus integrálás

A határozatlan integrálás problémája egy adott f függvényhez olyan g függvényt találni, amelynek deriváltja f, azaz amelyre $g'(x)=f(x)$; ezen összefüggés jelölésére az

$\int f(x)\,dx = g(x)$ jelölést is használjuk. A bevezető analízis előadásokon a határozatlan integrálási problémák megoldására különböző módszerekkel próbálkozunk, amelyek között heurisztikus módon próbálunk választani: helyettesítések, trigonometrikus helyettesítések, parciális integrálás stb. Csak a racionális törtfüggvények integrálására szokás algoritmikus módszert használni.

Megmutatható, hogy a határozatlan integrálás teljes általánosságban algoritmussal megoldhatatlan probléma. Tehát csak arra van lehetőségünk, hogy minél nagyobb algoritmussal megoldható részt keressünk.

Az első lépés a probléma algebraizálása: teljesen elvonatkoztatunk minden analízisbeli fogalomtól, és a differenciálást úgy tekintjük, mint egy új (egyváltozós) algebrai műveletet, amely az összeadással és a szorzással adott kapcsolatban van, és ennek a műveletnek az „inverzét" keressük. Ez a felfogás vezetett a differenciálalgebra fogalmának bevezetéséhez.

A komputeralgebra rendszerek (például a MAPLE) integráló rutinja, hasonlóan hozzánk, először néhány heurisztikus módszerrel próbálkozik. Polinomok (vagy kicsit általánosabban, véges Laurent-sorok) integrálja könnyen meghatározható. Ezután egy egyszerű táblázatban való keresés következik (a MAPLE esetén például 35 alapintegrál felhasználása). Lehet persze könyvekből bevitt integráltáblázatokat is használni. Ezután speciális eseteket kereshetünk, amelyre megfelelő módszerek ismertek. Például

$$e^{ax+b}\sin(cx+d)\cdot p(x)$$

alakú integrálok esetén, ahol p polinom, parciális integrálás használható az integrál meghatározására. Ha a fenti módszerek sikertelenek, akkor helyettesítéssel próbálkozik az úgynevezett „beosztás a deriválttal" módszer formájában: ha az integrandus összetett kifejezés, akkor minden $f(x)$ részkifejezésére osztunk f deriváltjával, majd kipróbáljuk, hogy $u = f(x)$ helyettesítés után a kapott kifejezésből eltűnik-e x. Ezek az egyszerű módszerek meglepően sok határozatlan integrál kiszámítására elegendőek. Nagy előnyük, hogy az egyszerű problémákat gyorsan oldják meg. Ha nem vezetnek célhoz, akkor próbálkozunk az algoritmikus módszerekkel. Ezek között az első a racionális törtfüggvények integrálása. Mint látni fogjuk, a komputeralgebra rendszerekben használt változat lényegesen eltér a kézi számolásra használt változattól, mert a cél az, hogy a futási idő még bonyolult esetekben is kicsi legyen, és az eredmény a lehető legegyszerűbb formában keletkezzen. Az elemi függvények integrálására használt Risch-algoritmus a racionális törtfüggvények integrálására használt algoritmusokon alapul. Ezt is ismertetjük, de nem tárgyaljuk teljes részletességgel. A bizonyításokra inkább csak utalunk.

2.4.1. Racionális függvények integrálása

Ebben a pontban bevezetjük a differenciáltest és a differenciáltest bővítésének fogalmát, majd ismertetjük Hermite módszerét.

Differenciáltest

Legyen K egy nulla karakterisztikájú test, amelyen adott egy $f \mapsto f'$ leképezése K-nak önmagába az alábbi két tulajdonsággal:

(1) $(f+g)' = f' + g'$ (additivitás);

(2) $(fg)' = f'g + g'f$ (Leibniz-szabály).

Ekkor az $f \mapsto f'$ leképezést ***differenciáloperátornak, differenciálásnak*** vagy ***deriválásnak,*** K-t pedig ***differenciáltestnek*** nevezzük. A $C = \{c \in K : c' = 0\}$ halmaz a ***konstansok részteste*** K-ban. Ha $f' = g$, akkor azt is írjuk, hogy $f = \int g$. Nyilván bármely $c \in C$ konstansra $f + c = \int g$. Egy $0 \neq f \in K$ elem ***logaritmikus deriváltján*** f'/f-et értjük. (Formálisan ez „$\log(f)$ deriváltja”.)

2.21. tétel. *Az előző definíció jelöléseivel, a deriválás szokásos tulajdonságai teljesülnek:*

(1) $0' = 1' = (-1)' = 0$;

(2) *a differenciálás C-lineáris:* $(af + bg)' = af' + bg'$, *ha* $f, g \in K$, $a, b \in C$;

(3) *ha* $g \neq 0$, f *tetszőleges, akkor* $(f/g)' = (f'g - g'f)/g^2$;

(4) $(f^n)' = nf'f^{n-1}$, *ha* $0 \neq f \in K$ *és* $n \in \mathbb{Z}$;

(5) $\int fg' = fg - \int gf'$, *ha* $f, g \in K$ (***parciális integrálás***).

2.17. példa.

(1) Az előző definíció jelöléseivel, az $f \mapsto 0$ leképezés K-n a *triviális deriválás*, erre $C = K$.

(2) Legyen $K = \mathbb{Q}(x)$. Egyetlen olyan differenciálás van $\mathbb{Q}(x)$-en, a szokásos, amelyre $x' = 1$. Erre a konstansok $\mathbb{Q}$ elemei. Valóban, indukcióval $n' = 0$, ha $n \in \mathbb{N}$, és így $\mathbb{Z}$ illetve $\mathbb{Q}$ elemei is konstansok. Indukcióval adódik, hogy a hatványfüggvények deriváltja a szokásos, ahonnan a linearitás miatt a polinomoké is, így a hányados differenciálási szabálya szerint kapjuk az állítást. Nem nehéz kiszámolni, hogy a szokásos differenciálásra a konstansok $\mathbb{Q}$ elemei.

(3) Ha $K = C(x)$, ahol C tetszőleges nulla karakterisztikájú test, akkor egyetlen olyan differenciálás van K-n, a szokásos, amelyre a konstansok részteste C és $x' = 1$; az állítás hasonlóan adódik, mint az előző.

Ha C tetszőleges nulla karakterisztikájú test és $K = C(x)$ a szokásos differenciálással, akkor $1/x$ nem deriváltja semminek. (Az állítás bizonyítása nagyon hasonlít annak bizonyításához, hogy $\sqrt{2}$ irracionális, de kettővel való oszthatóság helyett az x polinommal való oszthatósággal kell dolgoznunk.)

A példa azt mutatja, hogy $1/x$ és más hasonló függvények integrálásához a differenciáltestet bővíteni kell. A racionális törtfüggvények integrálásához elég lesz logaritmusokkal bővíteni.

Differenciáltest bővítése

Legyen L egy differenciáltest, $K \subset L$ pedig egy részteste L-nek. Ha a differenciálás nem vezet ki K-ból, akkor azt mondjuk, hogy K egy ***differenciálrészteste*** L-nek, illetve hogy L egy ***differenciálbővítése*** K-nak. Ha valamely $f, g \in L$-re $f' = g'/g$, azaz ha f deriváltja a g logaritmikus deriváltja, akkor azt írjuk, hogy $f = \log g$. (Megjegyezzük, hogy log, ugyanúgy, mint $\int$, nem függvény, hanem reláció. Más szóval a log itt egy absztrakt fogalom, nem pedig egy meghatározott alapú logaritmus függvény.) Ha $g \in K$ választható, akkor azt mondjuk, hogy f ***logaritmikus*** K ***felett.***

2.18. példa.

(1) Legyen $g = x \in K = \mathbb{Q}(x)$, $L = \mathbb{Q}(x, f)$, ahol f egy új határozatlan, és legyen $f' = g'/g = 1/x$, azaz $f = \log(x)$. Ekkor $\int 1/x\,dx = \log(x)$.

(2) Hasonlóan,

$$\int \frac{1}{x^2-2} = \frac{\sqrt{2}}{4}\log(x-\sqrt{2}) - \frac{\sqrt{2}}{4}\log(x+\sqrt{2})$$

a $\mathbb{Q}(\sqrt{2})(x, \log(x-\sqrt{2}), \log(x+\sqrt{2}))$ differenciáltestben van.

(3) Mivel

$$\int \frac{1}{x^3+x} = \log(x) - \frac{1}{2}\log(x+i) - \frac{1}{2}\log(x-i) = \log(x) - \frac{1}{2}\log(x^2+1)\,,$$

az integrál tekinthető

$$\mathbb{Q}(x, \log(x), \log(x^2+1))$$

elemének is és

$$\mathbb{Q}(i)(x, \log(x), \log(x-i), \log(x+i))$$

elemének is. Nyilván célszerűbb az első lehetőséget választani, mert ekkor az alaptest bővítésére nincs szükség.

Hermite módszere

Legyen K egy nulla karakterisztikájú test, $f, g \in K[x]$ nemnulla és relatív prím polinomok. Az $\int f/g$ integrál kiszámításához Hermite módszerével olyan $a, b, c, d \in K[x]$ polinomokat találhatunk, amelyekre

$$\int \frac{f}{g} = \frac{c}{d} + \int \frac{a}{b}\,, \tag{2.15}$$

ahol $\deg a < \deg b$ és b négyzetmentes főpolinom. A c/d racionális függvényt az integrál ***racionális részének,*** az $\int a/b$ kifejezést pedig az integrál ***logaritmikus részének*** nevezzük. A módszer elkerüli g-nek a (felbontási testben vagy valamely még bővebb testben) lineáris tényezőkre való bontását, sőt, még a K felett irreducibilis tényezőkre való felbontást is.

Nyilván feltehetjük, hogy g főpolinom. Maradékos osztással $f = pg + h$, ahol $\deg h < \deg g$, így $f/g = p + h/g$. A p polinomrész integrálása triviális. Határozzuk meg g négyzetmentes felbontását, azaz keressünk olyan $g_1, \ldots, g_m \in K[x]$ négyzetmentes és páronként relatív prím főpolinomokat, amelyekre $g_m \neq 1$ és $g = g_1 g_2^2 \cdots g_m^m$. Bontsunk parciális törtekre (ez euklideszi algoritmussal megvalósítható):

$$\frac{h}{g} = \sum_{i=1}^{m} \sum_{j=1}^{i} \frac{h_{i,j}}{g_i^j}\,,$$

ahol minden $h_{i,j}$ foka kisebb, mint g_i foka.

A Hermite-redukció a következő lépés ismétlése: ha $j > 1$, akkor az $\int h_{i,j}/g_i^j$ integrált egy racionális függvény és egy, az eredetihez hasonló alakú integrál összegére redukáljuk, amelyben j eggyel kisebb. Felhasználva, hogy g_i négyzetmentes, azt kapjuk, hogy $\mathrm{lnko}(g_i, g_i') = 1$, így a bővített euklidészi algoritmussal kaphatunk olyan $s, t \in K[x]$ polinomokat, amelyekre $sg_i + tg_i' = h_{i,j}$ és $\deg s, \deg t < \deg g_i$. Innen parciális integrálással

$$\begin{aligned}\int \frac{h_{i,j}}{g_i^j} &= \int \frac{t \cdot g_i'}{g_i^j} + \int \frac{s}{g_i^{j-1}} \\ &= \frac{-t}{(j-1)g_i^{j-1}} + \int \frac{t'}{(j-1)g_i^{j-1}} + \int \frac{s}{g_i^{j-1}}\end{aligned}$$

$$= \frac{-t}{(j-1)g_i^{j-1}} + \int \frac{s + t'/(j-1)}{g_i^{j-1}} \, .$$

Megmutatható, hogy gyors algoritmusokat választva, ha $\deg f, \deg g < n$, akkor az eljárás $O(M(n)\log n)$ darab K testbeli műveletet igényel, ahol $M(n)$ a legfeljebb n-ed fokú polinomok szorzásához szükséges műveletek számának korlátja.

Hermite módszerének van olyan változata is, amely elkerüli h/g parciális törtekre bontását. Ha $m = 1$, akkor g négyzetmentes. Ha $m > 1$, akkor legyen

$$g^* = g_1 g_2^2 \cdots g_{m-1}^{m-1} = \frac{g}{g_m^m} \, .$$

Mivel $\mathrm{lnko}(g_m, g^* g_m') = 1$, léteznek olyan $s, t \in K[x]$ polinomok, amelyekre

$$s g_m + t g^* g_m' = h \, .$$

Mindkét oldalt osztva $g = g^* g_m^m$-el és parciálisan integrálva

$$\int \frac{h}{g} = \frac{-t}{(m-1)g_m^{m-1}} + \int \frac{s + g^* t'/(m-1)}{g^* g_m^{m-1}} \, ,$$

így m-et eggyel csökkentettük.

Megjegyezzük, hogy a és c a határozatlan együtthatók módszerével is meghatározhatók (Horowitz módszere). Osztás után feltehetjük, hogy $\deg f < \deg g$. Mint az algoritmusból látszik, $d = g_2 g_3^2 \cdots g_m^{m-1}$ és $b = g_1 g_2 \cdots g_m$ választható. (2.15) differenciálásával az a polinom $\deg b$ darab és a c polinom $\deg d$ darab együtthatójára, tehát összesen n darab együtthatóra egy lineáris egyenletrendszert kapunk. Ez a módszer általában nem olyan gyors, mint Hermite módszere.

Az alábbi algoritmus az x változó adott f/g racionális függvényére elvégzi a Hermite-redukciót.

HERMITE-REDUKCIÓ(f, g)

```
 1  p ← quo(f, g)
 2  h ← rem(f, g)
 3  (g[1], ..., g[m]) ← NÉGYZETMENTES(g)
 4  bontsuk (h/g)-t parciális törtekre, számítsuk ki a g[i]^j-hez tartozó h[i, j] számlálókat
 5  rac ← 0
 6  int ← 0
 7  for i ← 1 to m
 8      do int ← int + h[i, 1]/g[i]
 9      for j ← 2 to i
10          do n ← j
11          while n > 1
12                  do s és t meghatározása az s · g[i] + t · g[i]' = h[i, j] egyenletből
13                  n ← n − 1
14                  rac ← rac − (t/n)/g[i]^n
15                  h[i, n] ← s + t'/n
16          int ← int + h[i, 1]/g[i]
17  red ← rac + ∫ p + ∫ int
18  return red
```

Ha valamely nulla karakterisztikájú K testre az $\int a/b$ integrált akarjuk kiszámítani, ahol $a, b \in K[x]$ nem nulla relatív prím polinomok, $\deg a < \deg b$, b négyzetmentes és főpolinom, akkor eljárhatunk úgy, hogy a b polinom L felbontási testében felírjuk b-t gyöktényezős alakban: $b = \prod_{k=1}^{n}(x - \leftarrow_k)$, majd L felett parciális törtekre bontunk: $a/b = \sum_{k=1}^{n} c_k/(x - \leftarrow_k)$, végül integrálunk:

$$\int \frac{a}{b} = \sum_{k=1}^{n} c_k \log(x - \leftarrow_k) \in L(x, \log(x - \leftarrow_1), \ldots, \log(x - \leftarrow_n)) .$$

Ennek az eljárásnak, mint az $1/(x^3 + x)$ függvény példáján láttuk, a hátránya, hogy az L testbővítés foka túl magas lehet. Előfordulhat az is, hogy L foka K felett $n!$, ami teljesen kezelhetetlen esetekhez vezet. Másrészt az sem világos, hogy kell-e az adott esetben az alaptestet bővíteni, például az $1/(x^2 - 2)$ függvény esetében nem lehet-e az integrálást az alaptest bővítése nélkül elvégezni. A következő tétel lehetővé teszi, hogy L testbővítés fokát a lehető legkisebbre válasszuk.

2.22. tétel (Rothstein–Trager-féle integráló algoritmus). *Legyen K nulla karakterisztikájú test, $a, b \in K[x]$ nem nulla relatív prím polinomok, $\deg a < \deg b$, b négyzetmentes és főpolinom. Ha L algebrai bővítése K-nak, $c_1, \ldots, c_k \in L \setminus K$ páronként különböző, $v_1, \ldots, v_k \in L[x] \setminus L$ pedig négyzetmentes és páronként relatív prím főpolinomok, akkor az alábbiak ekvivalensek:*

(1) $$\int \frac{a}{b} = \sum_{i=1}^{k} c_i \log v_i ,$$

(2) *Az $r = \mathrm{res}_x(b, a - yb') \in K[y]$ polinom L felett lineáris tényezőkre bomlik, $c_1, \ldots, c_k$ pontosan az r különböző gyökei, és $v_i = \mathrm{lnko}(b, a - c_i b')$, ha $i = 1, \ldots, k$. Itt res_x az x határozatlanban vett rezultáns.*

2.19. példa. Tekintsük ismét az $\int 1/(x^3+x)\,dx$ integrál kiszámításának problémáját. Ebben az esetben

$$r = \mathrm{res}_x(x^3 + x, 1 - y(3x^2 + 1)) = -4z^3 + 3y + 1 = -(2y + 1)^2(y - 1) ,$$

amelynek gyökei $c_1 = 1$ és $c_2 = -1/2$. Így

$$\begin{aligned} v_1 &= \mathrm{lnko}(x^3 + x, 1 - (3x^2 + 1)) = x , \\ v_2 &= \mathrm{lnko}(x^3 + x, 1 + \frac{1}{2}(3x^2 + 1)) = x^2 + 1 . \end{aligned}$$

Az előző tétel alapján könnyen felírható integráló algoritmus kicsit javítható: $v_i = \mathrm{lnko}(b, a - c_i b')$-nek (az L test feletti számításokkal történő) kiszámítása helyett v_i megkapható K feletti számítással is, a Bővített-Euklidesz-normalizált algoritmus segítségével. Ezt egymástól függetlenül Trager, illetve Lazard és Rioboo vette észre. Nem nehéz belátni, hogy az így adódó teljes integráló algoritmus futásideje $O(nM(n)\lg n)$, ha $\deg f, \deg g < n$.

2.23. tétel (Lazard–Rioboo–Trager-formula). *Az előző tétel jelöléseivel, jelölje e a c_i-nek mint az $r = \mathrm{res}_x(b, a - yb')$ polinom gyökének a multiplicitását. Ekkor*

(1) $\deg v_i = e$;

(2) *ha* $w(x,y) \in K(y)[x]$ *jelöli a* b*-re és* $a - yb'$*-re* $K(y)[x]$*-ben végrehajtott* Bővített-Euklidesz-normalizált*-algoritmus* e*-ed fokú maradékát, akkor* $v_i = w(x, c_i)$.

Az alábbi algoritmus a Rothstein–Trager-módszer Lazard–Rioboo–Trager-féle javítása. Az x változó adott a/b racionális függvényére kiszámítjuk $\int a/b$-t, ahol b négyzetmentes főpolinom és $\deg a < \deg b$.

Logaritmikus-rész-integrál(a, b, x)

```
 1  a szubrezultáns algoritmussal legyen r(y) ← res_x(b, a − yb′), továbbá
 2  a számítás során legyen w_e(x, y) az e-ed fokú maradék
 3  (r_1(y), ..., r_k(y)) ←Négyzetmentes(r(y))
 4  int ← 0
 5  for i ← 1 to k
 6      do if r_i(y) ≠ 1
 7           then w(y) ← w_i(x, y) együtthatóinak lnko-ja
 8                (l(y), s(y), t(y)) ←Bővített-Euklidesz-normalizált(w(y), r_i(y))
 9                w_i(x, y) ←Primitív-rész(rem(s(y) · w_i(x, y), r_i(y)))
10                (r_{i,1}, ..., r_{i,k}) ←Faktorok(r_i)
11                for j ← 1 to k
12                    do d ← deg(r_{i,j})
13                       c ←Megoldás(r_{i,j}(y) = 0, y)
14                       if d = 1
15                          then int ← int + c · log(w_i(x, c))
16                          else for n ← 1 to d
17                                   do int ← int + c[n] · log(w_i(x, c[n]))
18  return int
```

2.20. példa. Tekintsük ismét az $\int 1/(x^2-2)\,dx$ integrál kiszámításának problémáját. Ebben az esetben

$$r = \operatorname{res}_x(x^2 - 2, 1 - y \cdot 2x) = -8y^2 + 1 .$$

A polinom irreducibilis $\mathbb{Q}[x]$-ben, így $\mathbb{Q}$ bővítése elkerülhetetlen. Az r gyökei $\pm 1/\sqrt{8}$. A $\mathbb{Q}(y)$ feletti Bővített-Euklidesz-normalizált-algoritmusból $w_1(x,y) = x - 1/(2y)$, így az integrál

$$\int \frac{1}{x^2-2}\,dx = \frac{1}{\sqrt{8}}\log(x - \sqrt{2}) - \frac{1}{\sqrt{8}}\log(x + \sqrt{2}) .$$

2.4.2. Risch integráló algoritmusa

Meglepő módon, a racionális függvények integrálására talált módszerek általánosíthatók a szokásos függvényeket (sin, exp stb.) és inverzeiket tartalmazó kifejezések integrálására. A komputeralgebra rendszerek meglepően bonyolult kifejezések integrálását is elvégzik, néha azonban látszólag igen egyszerű esetekben sem adják meg az integrált, például az

$\int x/(1+e^x)\,dx$ kifejezést kiértékeletlenül kapjuk vissza, vagy az eredmény nem elemi speciális függvényt tartalmaz, például az integrállogaritmus függvényt. Ez azért van, mert ezekben az esetekben az integrál nem is fejezhető ki „zárt alakban".

Bár a „zárt alakban" kifejezhető integrálokra vonatkozó alapvető eredményt Liouville 1833-ban találta, a megfelelő algoritmikus módszereket Risch csak 1968-ban fejlesztette ki.

Elemi függvények

A „zárt alakban" megadható függvényeknek azokat a függvényeket szokás tekinteni, amelyek felépíthetők a racionális függvények, az exponenciális és logaritmus függvény, a trigonometrikus és hiperbolikus függvények és inverzeik, valamint gyökvonások, illetve sokkal általánosabban polinom függvények „inverzei", azaz egyenletek gyökeinek képzése segítségével, tehát ezen függvények egymásba helyettesítésével.

Észrevehetjük, hogy míg az $\int 1/(1+x^2)\,dx$ integrált $\operatorname{arctg}(x)$ alakban szokás megadni, a racionális törtfüggvény integrálására megadott algoritmus az eredményt

$$\int \frac{1}{1+x^2}\,dx = \frac{i}{2}\log(x+i) - \frac{i}{2}\log(x-i)$$

alakban adja. Mivel $\mathbb{C}$-ben a trigonometrikus és hiperbolikus függvények kifejezhetők az exponenciális függvény, inverzeik pedig a logaritmus függvény segítségével, csak az exponenciális és a logaritmus függvényre szorítkozhatunk. Meglepő módon kiderül, hogy az integrálok kifejezéséhez (algebrai számokkal való bővítések mellett) itt is csak logaritmusokkal való bővítésekre van szükség.

Exponenciális elemek

Legyen L a K differenciáltest differenciálbővítése. Ha egy $\theta \in L$ elemhez van olyan $u \in K$, hogy $\theta'/\theta = u'$, azaz ha θ logaritmikus deriváltja K valamely elemének a deriváltja, akkor azt mondjuk, hogy θ ***exponenciális*** K felett, és azt írjuk $\theta = \exp(u)$. Ha csak az teljesül, hogy a $\theta \in L$ elemhez van olyan $u \in K$, hogy $\theta'/\theta = u$, azaz, ha θ logaritmikus deriváltja K eleme, akkor azt mondjuk, hogy θ ***hiperexponenciális*** K felett.

A K felett logaritmikus, exponenciális vagy hiperexponenciális elemek lehetnek algebraiak vagy transzcendensek K felett.

Elemi kiterjesztések

Legyen L a K differenciáltest differenciálbővítése. Ha

$$L = K(\theta_1, \theta_2, \ldots, \theta_n)\ ,$$

ahol $j = 1, 2, \ldots, n$-re θ_j logaritmikus, exponenciális vagy algebrai

$$K_{j-1} = K(\theta_1, \ldots, \theta_{j-1})$$

felett ($K_0 = K$), akkor azt mondjuk, hogy L a K ***elemi kiterjesztése.*** Ha $j = 1, 2, \ldots, n$-re θ_j transzcendens és logaritmikus vagy transzcendens és exponenciális K_{j-1} felett, akkor azt mondjuk, hogy L a K ***transzcendens elemi kiterjesztése.***

Legyen $C(x)$ a racionális függvények teste $C_{\mathbb{C}}$ konstans testtel és a szokásos differenciálással. Ennek egy elemi kiterjesztését ***elemi függvénytestnek***, transzcendens elemi kiterjesztését pedig ***transzcendens elemi függvénytestnek*** nevezzük.

2.21. példa. Az $f = \exp(x) + \exp(2x) + \exp(x/2)$ függvény $f = \theta_1 + \theta_2 + \theta_3 \in \mathbb{Q}(x, \theta_1, \theta_2, \theta_3)$ alakban írható, ahol $\theta_1 = \exp(x)$, $\theta_2 = \exp(2x)$, $\theta_3 = \exp(x/2)$. Nyilván θ_1 exponenciális $\mathbb{Q}(x)$ felett, θ_2 exponenciális $\mathbb{Q}(x, \theta_1)$ felett és θ_3 exponenciális $\mathbb{Q}(x, \theta_1, \theta_2)$ felett. Mivel $\theta_2 = \theta_1^2$, $\mathbb{Q}(x, \theta_1, \theta_2) = \mathbb{Q}(\theta_1)$, így f az egyszerűbb $f = \theta_1 + \theta_1^2 + \theta_3$ alakba írható. A θ_3 függvény nemcsak exponenciális $\mathbb{Q}(x, \theta_1)$ felett, hanem algebrai is, mivel $\theta_3^2 - \theta_1 = 0$, azaz $\theta_3 = \theta_1^{1/2}$. Így $f = \theta_1 + \theta_1^2 + \theta_1^{1/2} \in \mathbb{Q}(x, \theta_1, \theta_1^{1/2})$. De f még ennél egyszerűbb alakban is felírható:

$$f = \theta_3^2 + \theta_3^4 + \theta_3 \in \mathbb{Q}(x, \theta_3)\,.$$

2.22. példa. Az

$$f = \sqrt{\log(x^2 + 3x + 2)(\log(x + 1) + \log(x + 2))}$$

függvény felírható $f = \theta_4 \in \mathbb{Q}(x, \theta_1, \theta_2, \theta_3, \theta_4)$ alakban, ahol $\theta_1 = \log(x^2 + 3x + 1)$, $\theta_2 = \log(x + 1)$, $\theta_3 = \log(x+2)$, θ_4 pedig a $\theta_4^2 - \theta_1(\theta_2 + \theta_3) = 0$ algebrai egyenletnek tesz eleget, de sokkal egyszerűbben $f = \theta_1 \in \mathbb{Q}(x, \theta_1)$ alakban is.

2.23. példa. Az $f = \exp(\log(x)/2)$ függvény felírható $f = \theta_2 \in \mathbb{Q}(x, \theta_1, \theta_2)$ alakban, ahol $\theta_1 = \log(x)$ és $\theta_2 = \exp(\theta_1/2)$, így θ_1 logaritmikus $\mathbb{Q}(x)$ felett, θ_2 pedig exponenciális $\mathbb{Q}(x, \theta_1)$ felett. Azonban $\theta_2^2 - x = 0$, így θ_2 algebrai $\mathbb{Q}(x)$ felett, és $f(x) = x^{1/2}$.

Elemi függvények integrálása

Tetszőleges elemi függvénytestek elemeinek integrálját, ha elemi függvény, a Liouville-elv teljesen jellemezni fogja. Mindazonáltal az algebrai kiterjesztési lépések – ha nem csak a konstansok testét terjesztjük ki – nagy nehézségeket okoznak.

Itt csak a transzcendens elemi függvénytestek függvényeinek Risch-algoritmussal történő integrálásával foglalkozunk.

A gyakorlatban egy $\mathbb{Q}(\alpha_1, \ldots, \alpha_k)(x, \theta_1, \ldots, \theta_n)$ transzcendens elemi függvénytest valamely eleméről van szó, ahol $\alpha_1, \ldots, \alpha_k$ algebrai számok $\mathbb{Q}$ felett, az integrál pedig egy

$$\mathbb{Q}(\alpha_1, \ldots, \alpha_k, \ldots, \alpha_{k+h})(x, \theta_1, \ldots, \theta_n, \ldots, \theta_{n+m})$$

elemi függvénytest eleme lesz. Elvileg legegyszerűbb lenne a konstansok testét $\mathbb{C}$-nek választani, de mint a racionális függvények integrálásánál láttuk, ez nem lehetséges, mert csak bizonyos számtestekben, például algebrai számtestekben tudunk pontosan számolni, sőt, igyekeznünk kell az $\alpha_{k+1}, \ldots, \alpha_{k+h}$ algebrai számok számát és fokaikat a lehető legalacsonyabban tartani. Mindazonáltal a konstansok testének algebrai bővítéseit kezelhetjük dinamikusan, a szükségessé váló bővítésekről elképzelhetjük, hogy már előre elvégeztük őket, a gyakorlatban pedig mindig csak akkor hajtjuk végre a bővítést, ha szükségessé válik.

Miután a trigonometrikus és hiperbolikus függvények exponenciálisra, inverzeiknek pedig logaritmusra való konverzióját elvégeztük, az integrandust mint egy elemi függvénytest elemét kapjuk meg. A 2.21. és 2.22. példák mutatják, hogy bár lehet, hogy a függvény „első pillantásra” nem tűnik egy transzcendens elemi kiterjesztés elemének, mégis az, a 2.23. példa pedig azt, hogy lehet, hogy a függvény „első pillantásra” transzcendens elemi kiterjesztés elemének tűnik, mégsem az. Az első lépés tehát az, hogy a különböző exponenciális

és logaritmikus függvények közötti algebrai kapcsolatok vizsgálatával az integrandust mint transzcendens elemi függvénytest elemét állítjuk elő. Hogy ez hogyan lehetséges, azzal nem foglalkozunk. Hogy ez sikerült-e, az ellenőrizhető a Risch-től származó struktúra-tétel segítségével. A tétel bizonyítását elhagytuk. Szükségünk lesz egy definícióra.

Egy θ elem ***monomiális*** a K differenciáltest felett, ha transzcendens K felett, exponenciális vagy logaritmikus K felett, valamint K-ban és $K(\theta)$-ban ugyanazok a konstansok.

2.24. tétel (struktúra-tétel). *Legyen K a konstansok teste és $K_n = K(x, \theta_1, \ldots, \theta_n)$ egy differenciál-kiterjesztése $K(x)$-nek, amelyben a konstansok teste K. Tegyük fel, hogy minden θ_j vagy algebrai $K_{j-1} = K(x, \theta_1, \ldots, \theta_{j-1})$ felett, vagy $\theta_j = w_j$, ahol $w_j = \log(u_j)$ és $u_j \in K_{j-1}$, vagy $\theta_j = u_j$, ahol $u_j = \exp(w_j)$ és $w_j \in K_{j-1}$. Ekkor*

1. *$g = \log(f)$, ahol $f \in K_n \setminus K$, pontosan akkor monomiális K_n felett, ha nincs olyan*

$$f^k \cdot \prod u_j^{k_j}, \quad k, k_j \in \mathbb{Z}, k \neq 0$$

   szorzat, amely K-beli;

2. *$g = \exp(f)$, ahol $f \in K_n \setminus K$, pontosan akkor monomiális K_n felett, ha nincs olyan*

$$f + \sum c_j w_j, \quad c_j \in \mathbb{Q}$$

   lineáris kombináció, amely K-beli.

A szorzatképzés, illetve összegzés csak a logaritmikus és exponenciális lépésekre történik.

Az egész elmélet legfontosabb, klasszikus eredménye a következő tétel.

2.25. tétel (Liouville-elv). *Legyen K egy differenciáltest a C konstans testtel. Legyen L egy elemi differenciálkiterjesztése K-nak ugyanazzal a C konstans testtel. Tegyük fel, hogy $g' = f \in K$. Ekkor léteznek olyan $c_1, \ldots, c_m \in C$ konstansok és $v_0, v_1, \ldots, v_m \in K$ elemek, hogy*

$$f = v_0' + \sum_{j=1}^{m} c_j \frac{v_j'}{v_j} ,$$

azaz hogy

$$g = \int f = v_0 + \sum_{j=1}^{m} c_j \log(v_j) .$$

Figyeljük meg, hogy a helyzet hasonló, mint a racionális függvények integrálásánál.

A tételt nem bizonyítjuk. Bár a bizonyítás hosszadalmas, az ötlete könnyen elmondható. Először megmutatjuk, hogy egy transzcendens exponenciális bővítést differenciálással nem lehet „kiküszöbölni”, azaz egy racionális függvényét deriválva, abból az új elem nem tűnik el. Ez azon múlik, hogy a transzcendens exponenciális bővítő elem egy polinomját deriválva, a derivált újra az elem egy polinomja lesz, a polinom foka nem változik, és a derivált nem osztható az eredeti polinommal, kivéve, ha az monom volt. Utána megmutatjuk, hogy algebrai bővítésre nincs szükség az integrál kifejezéséhez. Ez lényegében azon múlik, hogy az algebrai bővítő elemet a minimálpolinomjába beírva nullát kapunk, és ezt az egyenletet differenciálva, a kapott egyenletből kifejezhető a bővítő elem deriváltja, mint az elem

racionális függvénye. Végül a transzcendens logaritmikus elemekkel való bővítéseket kell még megvizsgálnunk. Megmutatjuk, hogy egy ilyen bővítő elem differenciálással pontosan akkor küszöbölhető ki, ha egy konstans együtthatós elsőfokú polinomja szerepel. Ez azon múlik, hogy egy ilyen bővítő elem egy polinomját deriválva, annak egy polinomját kapjuk, aminek fokszáma vagy ugyanannyi, mint az eredetié, vagy eggyel kisebb, és ez utóbbi eset csak akkor fordulhat elő, ha a főegyüttható konstans.

Risch-algoritmus
Legyen K algebrai számtest $\mathbb{Q}$ felett, $K_n = K(x, \theta_1, \ldots, \theta_n)$ pedig transzcendens elemi függvénytest. Az algoritmus rekurzív n-ben: $\theta = \theta_n$ jelöléssel egy $f(\theta)/g(\theta) \in K_n = K_{n-1}(\theta)$ függvényt fogunk integrálni, ahol $K_{n-1} = K(x, \theta_1, \ldots, \theta_{n-1})$. (Az $n = 0$ eset a racionális függvények integrálása.) Feltehetjük, hogy f és g relatív prímek és g főpolinom. Az x szerinti differenciálás mellett használni fogjuk a θ szerinti deriválást is, ezt $d/d\theta$-val jelöljük. A továbbiakban csak az algoritmusokat ismertetjük.

Risch-algoritmus: logaritmikus eset
Az előző pont jelöléseivel, először feltesszük, hogy θ transzcendens és logaritmikus, $\theta' = u'/u$, $u \in K_{n-1}$. Maradékos osztással $f(\theta) = p(\theta)g(\theta) + h(\theta)$, ahonnan

$$\int \frac{f(\theta)}{g(\theta)} = \int p(\theta) + \int \frac{h(\theta)}{g(\theta)} \ .$$

A racionális függvények integrálásával ellentétben, most a polinomrész integrálása a nehezebb. Ezért a racionális rész integrálásával kezdjük.

Logaritmikus eset, racionális rész
Legyen $g(\theta) = g_1(\theta)g_2^2(\theta)\cdots g_m^m(\theta)$ a $g(\theta)$ négyzetmentes felbontása. Ekkor

$$\operatorname{lnko}\left((g_j(\theta), \frac{d}{d\theta}g_j(\theta)\right)) = 1$$

nyilván teljesül. Megmutatható, hogy a jóval erősebb $\operatorname{lnko}(g_j(\theta), g_j(\theta)') = 1$ feltétel is teljesül. Parciális törtekre bontással

$$\frac{h(\theta)}{g(\theta)} = \sum_{i=1}^{m}\sum_{j=1}^{i}\frac{h_{i,j}(\theta)}{g_i(\theta)^j} \ .$$

Hermite-redukciót fogunk használni: a bővített euklidészi algoritmussal kaphatunk olyan $s(\theta), t(\theta) \in K_{n-1}[\theta]$ polinomokat, amelyekre $s(\theta)g_i(\theta) + t(\theta)g_i(\theta)' = h_{i,j}(\theta)$ és $\deg s(\theta), \deg t(\theta) < \deg g_i(\theta)$. Innen parciális integrálással

$$\begin{aligned}
\int \frac{h_{i,j}(\theta)}{g_i^j(\theta)} &= \int \frac{t(\theta)\cdot g_i(\theta)'}{g_i(\theta)^j} + \int \frac{s(\theta)}{g_i(\theta)^{j-1}} \\
&= \frac{-t(\theta)}{(j-1)g_i(\theta)^{j-1}} + \int \frac{t(\theta)'}{(j-1)g_i(\theta)^{j-1}} + \int \frac{s(\theta)}{g_i(\theta)^{j-1}} \\
&= \frac{-t(\theta)}{(j-1)g_i(\theta)^{j-1}} + \int \frac{s(\theta) + t(\theta)'/(j-1)}{g_i(\theta)^{j-1}} \ .
\end{aligned}$$

Addig alkalmazva ezt az eljárást, amíg $j > 1$, azt kapjuk, hogy

$$\int \frac{h(\theta)}{g(\theta)} = \frac{c(\theta)}{d(\theta)} + \int \frac{a(\theta)}{b(\theta)} ,$$

ahol $a(\theta), b(\theta), c(\theta), d(\theta) \in K_{n-1}[\theta]$, $\deg a(\theta) < \deg b(\theta)$ és $b(\theta)$ négyzetmentes főpolinom.

Megmutatható, hogy az $\int a(\theta)/b(\theta)$ integrál kiszámítására a Rothstein–Trager-módszer alkalmazható. Számítsuk ki az

$$r(y) = \mathrm{res}_\theta(b(\theta), a(\theta) - y \cdot b(\theta)')$$

rezultánst. Megmutatható, hogy az integrál pontosan akkor elemi, ha $r(y) = r(y)s$ alakban írható, ahol $r(y) \in K[y]$ és $s \in K_{n-1}$. Ha tehát kiszámítjuk $r(y)$ primitív részét, ezt választjuk $r(y)$-nak, és $r(y)$ bármelyik együtthatója nem konstans, akkor nem létezik elemi integrál. Egyébként legyenek $c_1, \ldots, c_k$ az $r(y)$ különböző gyökei annak felbontási testében és legyen

$$v_i(\theta) = \mathrm{lnko}(b(\theta), a(\theta) - c_i b(\theta)') \in K_{n-1}(c_1, \ldots, c_k)[\theta] ,$$

ha $i = 1, \ldots, k$. Megmutatható, hogy

$$\int \frac{a(\theta)}{b(\theta)} = \sum_{i=1}^{k} c_i \log(v_i(\theta)) .$$

Tekintsünk néhány példát.

2.24. példa. Az $\int 1/\log(x)$ integrál integrandusa $1/\theta \in \mathbb{Q}(x, \theta)$, ahol $\theta = \log(x)$. Mivel

$$r(y) = \mathrm{res}_\theta(\theta, 1 - y/x) = 1 - y/x \in \mathbb{Q}(x)[y]$$

primitív polinom és van nem konstans együtthatója, az integrál nem elemi.

2.25. példa. Az $\int 1/(x\log(x))$ integrál integrandusa $1/(x\theta) \in \mathbb{Q}(x, \theta)$, ahol $\theta = \log(x)$. Itt

$$r(y) = \mathrm{res}_\theta(\theta, 1/x - y/x) = 1/x - y/x \in \mathbb{Q}(x)[y],$$

aminek primitív része $1 - y$. Ennek minden együtthatója konstans, így az integrál elemi, $c_1 = 1$, $v_1(\theta) = \mathrm{lnko}(\theta, 1/x - 1/x) = \theta$, így

$$\int \frac{1}{x\log(x)} = c_1 \log(v_1(\theta)) = \log(\log(x)) .$$

Logaritmikus eset, polinom rész

Marad a

$$p(\theta) = p_k\theta^k + p_{k-1}\theta^{k-1} + \cdots + p_0 \in K_{n-1}[\theta]$$

polinomrész integrálásának problémája. A Liouville-elv szerint $\int p(\theta)$ pontosan akkor elemi, ha

$$p(\theta) = v_0(\theta)' + \sum_{j=1}^{k} c_j \frac{v_j(\theta)'}{v_j(\theta)} ,$$

ahol $c_j \in K$ és $v_i \in K_{n-1}(\theta)$, ha $j = 0, 1, \ldots, m$, továbbá $K_{\mathbb{C}}$ a K valamely bővítése és $K_{n-1} = K(x, \theta_1, \ldots, \theta_{n-1})$. Meg fogjuk mutatni, hogy K lehet a K egy algebrai bővítése. Hasonlóan érvelve, mint a Liouville-elv bizonyításában, megmutatható, hogy $v_0(\theta) \in K_{n-1}[\theta]$ és $v_j(\theta) \in K_{n-1}$ (azaz független θ-tól), ha $j = 1, 2, \ldots, m$. Így

$$p(\theta) = v_0(\theta)' + \sum_{j=1}^{m} c_j \frac{v_j'}{v_j} \ .$$

A Liouville-elv bizonyításánál használt érveléssel azt is megkapjuk, hogy $v_0(\theta)$ foka legfeljebb $k+1$. így ha $v_0(\theta) = q_{k+1}\theta^{k+1} + q_k\theta^k + \cdots + q_0$, akkor

$$p_k\theta^k + p_{k-1}\theta^{k-1} + \cdots + p_0 = (q_{k+1}\theta^{k+1} + q_k\theta^k + \cdots + q_0)' + \sum_{j=1}^{m} c_j \frac{v_j'}{v_j} \ .$$

Innen a következő egyenletrendszert kapjuk:

$$\begin{aligned} 0 &= q_{k+1}' \ , \\ p_k &= (k+1)q_{k+1}\theta' + q_k' \ , \\ p_{k-1} &= kq_k\theta' + q_{k-1}' \ , \\ &\vdots \\ p_1 &= 2q_2\theta' + q_1' \ , \\ p_0 &= q_1\theta' + q_0' \ , \end{aligned}$$

ahol az utolsó egyenletben $q_0 = q_0 + \sum_{j=1}^{m} c_j \log(v_j)$. Az első egyenlet megoldása egyszerűen egy b_{k+1} konstans. Ezt visszahelyettesítve a következő egyenletbe és mindkét oldalt integrálva azt kapjuk, hogy

$$\int p_k = (k+1)b_{k+1} \cdot \theta + q_k \ .$$

Az integrálási eljárást rekurzív módon alkalmazva $p_k \in K_{n-1}$ integrálja kiszámítható, azonban ez az egyenlet csak akkor oldható meg, ha az integrál elemi, és legfeljebb egy logaritmikus bővítést használ és az éppen $\theta = \log(u)$. Ha ez nem teljesül, akkor $\int p(\theta)$ nem lehet elemi. Ha ez teljesül, akkor $\int p_k = c_k\theta + d_k$ valamely $c_k \in K$ és $d_k \in K_{n-1}$-el, ahonnan $b_{k+1} = c_{k+1}/(k+1) \in K$ és $q_k = d_k + b_k$ egy tetszőleges b_k integrációs konstanssal. Behelyettesítve q_k-t a következő egyenletbe és átrendezve

$$p_{k-1} - kd_k\theta' = kb_k\theta' + q_{k-1}' \ ,$$

vagyis mindkét oldalt integrálva

$$\int \left(p_{k-1} - kd_k\frac{u'}{u}\right) = kb_k\theta + q_{k-1}$$

adódik. A jobb oldalon az integrandus K_{n-1}-beli, így az integrációs eljárást rekurzív módon hívhatjuk. Ugyanúgy mint fent, az egyenlet csak akkor oldható meg, ha az integrál elemi, és legfeljebb egy logaritmikus bővítést használ és az éppen $\theta = \log(u)$. Tegyük fel, hogy ez

teljesül és

$$\int\left(p_{k-1}-k d_k\frac{u'}{u}\right)=c_{k-1}\theta+d_{k-1}\ ,$$

ahol $c_{k-1}\in K$ és $d_{k-1}\in K_{n-1}$. Ekkor a keresett megoldás $b_k=c_{k-1}/k\in K$ és $q_{k-1}=d_{k-1}+b_{k-1}$, ahol b_{k-1} egy tetszőleges integrációs konstans. Az eljárást folytatva, az utolsó előtti egyenlet megoldása $b_2=c_1/2\in K$ és $q_1=d_1+b_1$ valamely b_1 integrációs konstanssal. Behelyettesítve q_1-et az utolsó egyenletbe, átrendezve, majd integrálva

$$\int\left(p_0-d_1\frac{u'}{u}\right)=b_1\theta+q_0\ .$$

Ez alkalommal csak az a feltétel, hogy az integrál elemi függvény legyen. Ha elemi függvény, mondjuk

$$\int\left(p_0-d_1\frac{u'}{u}\right)=d_0\in K_{n-1}\ ,$$

akkor $b_1\in K$ a $\theta=\log(u)$ együtthatója d_0-ban és $q_0=d_0-b_1\log(u)$, az eredmény pedig

$$\int p(\theta)=b_{k+1}\theta^{k+1}+q_k\theta^k+\cdots+q_1\theta+q_0\ .$$

Nézzünk néhány példát.

2.26. példa. Az $\int\log(x)$ integrál integrandusa $\theta\in\mathbb{Q}(x,\theta)$, ahol $\theta=\log(x)$. Ha az integrál elemi, akkor

$$\int\theta=b_2\theta^2+q_1\theta+q_0$$

és $0=b_2'$, $1=2b_2\theta'+q_1'$, $0=q_1\theta'+q_0'$. Az ismeretlen b_2 konstanssal a második egyenletből $\int 1=2b_2\theta+q_1$. Mivel $\int 1=x+b_1$, azt kapjuk, hogy $b_2=0$, $q_1=x+b_1$. A harmadik egyenletből $-x\theta'=b_1\theta'+q_0'$. Mivel $\theta'=1/x$, integrálva $\int -1=b_1\theta+q_0$, és $\int -1=-x$, azt kapjuk, hogy $b_1=0$, $q_0=-x$, így $\int\log(x)=x\log(x)-x$.

2.27. példa. Az $\int\log(\log(x))$ integrál integrandusa $\theta_2\in\mathbb{Q}(x,\theta_1,\theta_2)$, ahol $\theta_1=\log(x)$ és $\theta_2=\log(\theta_1)$. Ha az integrál elemi, akkor

$$\int\theta_2=b_2\theta_2^2+q_1\theta_2+q_0$$

és $0=b_2'$, $1=2b_2\theta_2'+q_1'$, $0=q_1\theta_2'+q_0'$. Az ismeretlen b_2 konstanssal a második egyenletből $\int 1=2b_2\theta+q_1$. Mivel $\int 1=x+b_1$, azt kapjuk, hogy $b_2=0$, $q_1=x+b_1$. A harmadik egyenletből $-x\theta_2'=b_1\theta_2'+q_0'$. Mivel $\theta_2'=\theta_1'/\theta_1=1/(x\log(x))$, teljesülnie kell az

$$\int\frac{-1}{\log(x)}=b_1\theta_2+q_0$$

egyenlőségnek, azonban a 2.24. példából tudjuk, hogy a bal oldalon álló integrál nem elemi.

Risch-algoritmus: exponenciális eset

Most feltesszük, hogy θ transzcendens és exponenciális, $\theta'/\theta=u'$, $u\in K_{n-1}$. Maradékos osztással $f(\theta)=q(\theta)g(\theta)+h(\theta)$, ahonnan

$$\int\frac{f(\theta)}{g(\theta)}=\int q(\theta)+\int\frac{h(\theta)}{g(\theta)}\ .$$

Tervünk az, hogy a racionális részre Hermite módszerét fogjuk alkalmazni. Kellemetlen meglepetés ér azonban bennünket, mert a négyzetmentes felbontásban szereplő $g_j(\theta)$ függvényekre bár

$$\mathrm{lnko}\left((g_j(\theta), \frac{d}{d\theta}g_j(\theta)\right)) = 1$$

nyilván teljesül, a jóval erősebb $\mathrm{lnko}(g_j(\theta), g_j(\theta)') = 1$ feltétel már nem. Például ha $g_j(\theta) = \theta$, akkor

$$\mathrm{lnko}(g_j(\theta), g_j(\theta)') = \mathrm{lnko}(\theta, u'\theta) = \theta \ .$$

Megmutatható azonban, hogy ez a kellemetlen jelenség nem lép fel, ha $\theta \nmid g_j(\theta)$, ekkor már $\mathrm{lnko}(g_j(\theta), g_j(\theta)') = 1$. Elég tehát, ha a θ tényezőt eltávolítjuk a nevezőből. Legyen $g(\theta) = \theta^\ell g(\theta)$, ahol már $\theta \nmid g(\theta)$, és keressünk olyan $h(\theta), s(\theta) \in K_{n-1}[\theta]$ polinomokat, amelyekre $h(\theta)\theta^\ell + t(\theta)g(\theta) = h(\theta)$, $\deg h(\theta) < \deg g(\theta)$ és $\deg s(\theta) < \ell$. Mindkét oldalt osztva $g(\theta)$-val azt kapjuk, hogy

$$\frac{f(\theta(}{g(\theta)} = q(\theta) + \frac{t(\theta)}{\theta^l} + \frac{h(\theta)}{g(\theta)} \ .$$

A $p(\theta) = q(\theta) + t(\theta)/\theta^l$ jelöléssel $p(\theta)$ véges Laurent-sor, ennek integrálása azonban semmivel sem lesz nehezebb, mint egy polinom integrálása. Ez nem meglepő, ha meggondoljuk, hogy $\theta^{-1} = \exp(-u)$. Még így is, itt is a „polinomrész" integrálása a nehezebb. A másikkal kezdjük.

Exponenciális eset, racionális rész
Legyen $g(\theta) = g_1(\theta)g_2^2(\theta)\cdots g_m^m(\theta)$ a $g(\theta)$ négyzetmentes felbontása. Ekkor $\theta \nmid g_j(\theta)$ miatt $\mathrm{lnko}(g_j(\theta), g_j(\theta)') = 1$. Parciális törtekre bontással

$$\frac{h(\theta)}{g(\theta)} = \sum_{i=1}^{m}\sum_{j=1}^{i}\frac{h_{i,j}(\theta)}{g_i(\theta)^j} \ .$$

A Hermite-redukció ugyanúgy megy, mint a logaritmikus résznél. Azt kapjuk, hogy

$$\int \frac{h(\theta)}{g(\theta)} = \frac{c(\theta)}{d(\theta)} + \int \frac{a(\theta)}{b(\theta)} \ ,$$

ahol $a(\theta), b(\theta), c(\theta), d(\theta) \in K_{n-1}[\theta]$, $\deg a(\theta) < \deg b(\theta)$ és $b(\theta)$ négyzetmentes főpolinom, $\theta \nmid b(\theta)$.

Megmutatható, hogy az $\int a(\theta)/b(\theta)$ integrál kiszámítására a Rothstein–Trager-módszer alkalmazható. Számítsuk ki a

$$r(y) = \mathrm{res}_\theta(b(\theta), a(\theta) - y \cdot b(\theta)')$$

rezultánst. Megmutatható, hogy az integrál pontosan akkor elemi, ha $r(y) = r(y)s$ alakban írható, ahol $r(y) \in K[y]$ és $s \in K_{n-1}$. Ha tehát kiszámítjuk $r(y)$ primitív részét, ezt választjuk $r(y)$-nak, és $r(y)$ bármelyik együtthatója nem konstans, akkor nem létezik elemi integrál. Egyébként legyenek $c_1, \ldots, c_k$ az $r(y)$ különböző gyökei annak felbontási testében és legyen

$$v_i(\theta) = \mathrm{lnko}(b(\theta), a(\theta) - c_i b(\theta)') \in K_{n-1}(c_1, \ldots, c_k)[\theta] \ ,$$

ha $i = 1, \ldots, k$. Megmutatható, hogy

$$\int \frac{a(\theta)}{b(\theta)} = -\left((\sum_{i=1}^{k} c_i \deg v_i(\theta) \right) + \sum_{i=1}^{k} c_i \log(v_i(\theta)) \ .$$

Nézzünk néhány példát.

2.28. példa. Az $\int 1/(1+\exp(x))$ integrál integrandusa $1/(1+\theta) \in \mathbb{Q}(x,\theta)$, ahol $\theta = \exp(x)$. Mivel

$$r(y) = \mathrm{res}_\theta(\theta + 1, 1 - y\theta) = -1 - y \in \mathbb{Q}(x)[y]$$

primitív polinom és csak konstans együtthatói vannak, az integrál elemi, $c_1 = -1$, $v_1(\theta) = \mathrm{lnko}(\theta + 1, 1 + \theta) = 1 + \theta$, így

$$\int \frac{1}{1+\exp(x)} = -c_1 x \deg v_1(\theta) + c_1 \log(v_1(\theta)) = x - \log(\exp(x) + 1) \ .$$

2.29. példa. Az $\int x/(1+\exp(x))$ integrál integrandusa $x/(1+\theta) \in \mathbb{Q}(x,\theta)$, ahol $\theta = \exp(x)$. Mivel

$$r(y) = \mathrm{res}_\theta(\theta + 1, x - y\theta) = -x - y \in \mathbb{Q}(x)[y]$$

primitív polinom, amelynek van nem konstans együtthatója, az integrál nem elemi.

Exponenciális eset, polinom rész

Marad a

$$p(\theta) = \sum_{i=-\ell}^{k} p_i \theta^i \in K_{n-1}(\theta)$$

„polinomrész” integrálásának problémája. A Liouville-elv szerint $\int p(\theta)$ pontosan akkor elemi, ha

$$p(\theta) = v_0(\theta)' + \sum_{j=1}^{m} c_j \frac{v_j(\theta)'}{v_j(\theta)} \ ,$$

ahol $c_j \in K$ és $v_j \in K_{n-1}(\theta)$, ha $j = 0, 1, \ldots, m$, továbbá $K_\mathbb{C}$ a K valamely bővítése és $K_{n-1} = K(x, \theta_1, \ldots, \theta_{n-1})$. Megmutatható, hogy K lehet a K egy algebrai bővítése. Hasonlóan érvelve, mint a Liouville-elv bizonyításában, megmutatható, hogy az általánosság megszorítása nélkül feltehetjük: $v_j(\theta)$ vagy K_{n-1} eleme (azaz független θ-tól) vagy pedig főpolinom és irreducibilis $K_{n-1}[\theta]$-ban, ha $j = 1, 2, \ldots, m$. Továbbá az is belátható, hogy $v_0(\theta)$ nevezőjében nem lehet nem monom tényező, mivel egy ilyen tényező megmaradna a derivált nevezőjében is. Hasonlóan $v_j(\theta)$-nak ($j = 1, 2, \ldots, m$) sem lehet nem monom tényezője. Így azt kapjuk, hogy $v_j(\theta)$ vagy K_{n-1} eleme, vagy pedig $v_j(\theta) = \theta$, mivel ez az egyetlen irreducibilis monom, amely főpolinom. Ha azonban $v_j(\theta) = \theta$, akkor a megfelelő tag az összegben $c_j v_j(\theta)'/v_j(\theta) = c_j u'$, ami beolvasztható $v_0(\theta)'$-be. Ebből azt kapjuk, hogy ha $p(\theta)$-nak van elemi integrálja, akkor

$$p(\theta) = \left((\sum_{j=-\ell}^{k} q_j \theta^j \right)' + \sum_{j=1}^{m} c_j \frac{v_j'}{v_j} \ ,$$

ahol $q_j, v_j \in K_{n-1}$ és $c_j \in K$; az, hogy az összegzési határok az első összegben meg kell egyezzenek a $p(\theta)$ előállításában szereplő összegzési határokkal, következik abból, hogy

$$(q_j\theta^j)' = (q_j' + ju'g_j)\theta^j \ .$$

Az együtthatók összehasonlításával a

$$\begin{aligned} p_j &= q_j' + ju'q_j, \quad \text{ha } -\ell \le j \le k,\ j \ne 0 \ , \\ p_0 &= q_0' \ , \end{aligned}$$

egyenletrendszert kapjuk, ahol $q_0 = q_0 + \sum_{j=1}^{m} c_j \log(v_j)$. A $p_0 = q_0'$ egyenlet megoldása egyszerűen $q_0 = \int p_0$; ha ez az integrál nem elemi, akkor $\int p(\theta)$ sem elemi, ha viszont elemi, akkor meghatároztuk q_0-at. A $j \ne 0$ esetben egy differenciálegyenletet kell megoldanunk q_j meghatározásához, az úgynevezett ***Risch-differenciálegyenletet.*** A differenciálegyenlet $y' + fy = g$ alakú, ahol az adott f, g függvények K_{n-1} elemei és K_{n-1}-beli megoldásokat keresünk. Első pillantásra úgy tűnik, hogy az integrálás problémáját egy nehezebb problémával helyettesítettük, azonban az, hogy az egyenletek lineárisak, a megoldásnak pedig K_{n-1}-ben kell lenni, sokat segít. Ha valamelyik Risch-differenciálegyenletnek nincs K_{n-1}-beli megoldása, akkor $\int p(\theta)$ nem elemi, egyébként

$$\int p(\theta) = \sum_{j \ne 0} q_j\theta^j + q_0 \ .$$

A Risch-differenciálegyenlet algoritmussal megoldható, ezt nem részletezzük.

Tekintsünk néhány példát.

2.30. példa. Az $\int \exp(-x^2)$ integrál integrandusa $\theta \in \mathbb{Q}(x, \theta)$, ahol $\theta = \exp(-x^2)$. Ha az integrál elemi, akkor $\int \theta = q_1\theta$, ahol $q_1 \in \mathbb{C}(x)$. Nem nehéz belátni, hogy a differenciálegyenletnek nincs racionális megoldása, így $\int \exp(-x^2)$ nem elemi.

2.31. példa. Az $\int x^x$ integrál integrandusa $\exp((x\log(x)) = \theta_2 \in \mathbb{Q}(x, \theta_1, \theta_2)$, ahol $\theta_1 = \log(x)$ és $\theta_2 = \exp(x\theta_1)$. Ha az integrál elemi, akkor $\int \theta_2 = q_1\theta_2$, ahol $q_1 \in \mathbb{C}(x, \theta_1)$. Mindkét oldalt differenciálva $\theta_2 = q_1'\theta_2 + q_1(\theta_1 + 1)\theta_2$, ahonnan $1 = q_1' + (\theta_1 + 1)q_1$. Mivel θ_1 transzcendens $\mathbb{C}(x)$ felett, az együtthatók összehasonlításával $1 = q_1' + q_1$ és $0 = q_1$, aminek nincs megoldása. így $\int x^x$ nem elemi.

2.32. példa. Az

$$\int \frac{(4x^2 + 4x - 1)(\exp(x^2) + 1)(\exp(x^2) - 1)}{(x+1)^2}$$

integrál integrandusa

$$f(\theta) = \frac{4x^2 + 4x - 1}{(x+1)^2}(\theta^2 - 1) \in \mathbb{Q}(x, \theta) \ ,$$

ahol $\theta = \exp(x^2)$. Ha az integrál elemi, akkor $\int f(\theta) = q_2\theta^2 + q_0$ alakú, ahol

$$q_2' + 4xq_2 = \frac{4x^2 + 4x - 1}{(x+1)^2} \ ,$$

$$q_0' = -\frac{4x^2+4x-1}{(x+1)^2} .$$

A második egyenlet integrálható, és q_0 elemi. Az első egyenletnek megoldása $q_2 = 1/(1+x)$. Innen

$$\int f(\theta) = \frac{1}{x+1}\exp^2(x^2) - \frac{(2x+1)^2}{x+1} + 4\log(x+1) .$$

Gyakorlatok

2.4-1. Alkalmazzuk a Hermite-redukciót az alábbi $f(x) \in \mathbb{Q}(x)$ függvényre:

$$f(x) = \frac{441x^7 + 780x^6 - 286x^5 + 4085x^4 + 769x^3 + 3713x^2 - 43253x + 24500}{9x^6 + 6x^5 - 65x^4 + 20x^3 + 135x^2 - 154x + 49} .$$

2.4-2. Számítsuk ki az $\int f$ integrált, ahol

$$f(x) = \frac{36x^6 + 126x^5 + 183x^4 + 13807/6x^3 - 407x^2 - 3242/5x + 3044/15}{(x^2 + 7/6x + 1/3)^2(x - 2/5)^3} \in \mathbb{Q}(x) .$$

2.4-3. Alkalmazzuk a Risch-algoritmust az alábbi integrál kiszámítására:

$$\int \frac{x(x+1)\{(x^2e^{2x^2} - \log(x+1)^2) + 2xe^{3x^2}(x - (2x^3+2x^2+x+1)\log(x+1))\}}{((x+1)\log^2(x+1) - (x^3+x^2)e^{2x^2})^2} dx .$$

2.5. Elmélet és gyakorlat

A fejezet eddigi részében arra törekedtünk, hogy néhány lényeges szimbolikus algoritmus bemutatásán keresztül érzékeltessük a komputeralgebra tudományterületének algoritmustervezési problémáit. A következőkben az érdeklődő Olvasó áttekintést kaphat a szimbolikus algoritmusok kutatásának szélesebb világából.

2.5.1. Egyéb szimbolikus algoritmusok

A fejezetben bemutatott rezultánsmódszer és a Gröbner-bázis elmélet mellett létezik algoritmus nemlineáris egyenletek és egyenlőtlenségek *valós* szimbolikus gyökeinek meghatározására is (Collins).

Figyelemre méltó algoritmusok születtek differenciálegyenletek szimbolikus megoldásainak vizsgálata során. A Risch-algoritmushoz hasonló döntési eljárás létezik racionálisfüggvény-együtthatójú homogén másodrendű közönséges differenciálegyenletek zárt alakú megoldásainak kiszámítására. Magasabb rendű lineáris esetben az Abramov-eljárás a polinomegyüttható egyenletek zárt racionális megoldásait, a Bronstein-algoritmus pedig az $\exp(\int f(x)dx)$ alakú megoldásokat határozza meg. Parciális differenciálegyenletek esetében a Lie-féle szimmetria-módszerek állnak rendelkezésre. Létezik algoritmus formális hatványsorok és racionális függvények feletti lineáris differenciáloperátorok faktorizálására is.

A faktorizáláson alapuló eljárások komoly jelentőséggel bírnak a komputeralgebrai algoritmusok kutatásában. Olyannyira igaz ez a megállapítás, hogy sokan az egész tudományterület születését Berlekamp azon publikációjától számítják, amelyben hatékony algoritmust ad kis p karakterisztikájú véges testek feletti egyhatározatlanú polinomok faktorizációjára. Berlekamp eredményét később nagyobb p karakterisztikákra is kiterjesztette. Azért, hogy hasonlóan jó futási időt kapjon, az algoritmusába véletlen elemeket illesztett. A mai komputeralgebra rendszerek nagyobb véges testekre is rutinszerűen alkalmazzák Berlekamp eljárását talán anélkül, hogy a felhasználók többségének az algoritmus valószínűségi eredetéről tudomása lenne. A módszer könyvünk második kötetében kerül ismertetésre. Megjegyezzük, hogy véges testek feletti polinomok faktorizálására számos algoritmus létezik.

Nem sokkal azután, hogy a véges testek feletti polinomfaktorizáció megoldódott, Zassenhaus van der Waerden 1936-os *Moderne Algebra* könyvét alapul véve a p-adikus számok aritmetikájának ún. Hensel-lemmáját alkalmazta a faktorok kiterjesztésére. A „Hensel-lifting" – ahogy ma az eljárását nevezik – általános megközelítési módszer arra, hogyan rekonstruáljuk a faktorokat moduláris képeikből. Az interpolációval ellentétben, ami több képpont meglétét követeli meg, a Hensel-lifting kiindulásul csak egyetlen képpontot igényel. Az egész együtthatós polinomok faktorizációjára adott Berlekamp–Zassenhaus-féle algoritmus alapvető jelentőségű, mégis rejteget magában két csapdát. Az egyik, hogy bizonyos fajta polinomokra az algoritmus futási ideje exponenciális. Sajnos, az algebrai számtestekre épített polinomok faktorizációjánál sok ilyen „rossz" polinom kerül elő. A második, hogy többváltozós polinomokra hasonló reprezentációs probléma lép fel, mint amilyet ritka mátrixok Gauss-eliminációjánál tapasztalunk. Az első problémát egy, a számok geometriáján alapuló diofantoszi optimalizálás, a Lenstra–Lenstra–Lovász nevével fémjelzett ún. rácsredukciós algoritmus oldotta meg, amit a Berlekamp-módszerrel együtt használnak. Ezen polinomiális algoritmus mellé még egy olyan eljárás társul, amely arról gondoskodik, hogy a Hensel-lifting „jó" moduláris képről induljon és „megfelelő időben" véget érjen. A többváltozós polinomfaktorizáció említett reprezentációs problémájára is születtek megoldások. Ez a második olyan terület, ahol a véletlenítés kritikus szerepet kap a hatékony algoritmusok tervezésében. Megjegyezzük, hogy a gyakorlatban a Berlekamp–Zassenhaus–Hensel-féle algoritmus hatékonyabban működik, mint a Lenstra–Lenstra–Lovász-féle eljárás. Összevetésképpen a polinomfaktorizálás problémája tehát polinomiális időben megoldható, míg az N egész faktorizálására a bizonyítottan legjobb algoritmikus korlát $\widetilde{O}(N^{1/4})$ (Pollard és Strassen) a determinisztikus, és $L(N)^{1+o(1)}$ (Lenstra és Pomerance) a valószínűségi esetben, ahol $L(N) = e^{\sqrt{\ln N \ln\ln N}}$.

Valójában a heurisztikus, valószínűségi módszereknek egy új elmélete van születőben a komputeralgebrában egyrészt az számítási tárrobbanás elkerülésére, másrészt a determinisztikusan nagy futási idejű algoritmusok hatékonyságának növelésére. A valószínűségi algoritmusok esetében a nem megfelelő működésnek is van pozitív valószínűsége, ami vagy az esetleges rossz válaszban nyilvánul meg (Monte Carlo csoport), vagy, habár sohasem kapunk rossz választ (Las Vegas csoport), elképzelhető, hogy polinomiális időben nem kapunk semmit. A már említetteken kívül heurisztikákkal szép eredményeket értek el többek között polinomazonosság tesztelésénél, polinomok irreducibilitásának vizsgálatánál, mátrixok normálformáinak (Frobenius, Hilbert, Smith) meghatározásánál. Szerepük minden valószínűség szerint a jövőben is növekedni fog.

A fejezet eddigi részében a lényegesebb szimbolikus algoritmusokat tekintettük át. Már a bevezetőben említettük, hogy a komputeralgebra-rendszerek többsége képes numerikus

számítások elvégzésére is: a hagyományostól eltérően a számítási pontosságot a felhasználó határozhatja meg. Gyakran előfordul, hogy a szimbolikus és numerikus számításokat együtt célszerű használni. Tekintsük például egy differenciálegyenlet szimbolikusan kiszámolt hatványsor megoldását. A csonkított hatványsort ezután bizonyos pontokban a szokásos lebegőpontos aritmetikával kiértékelve a differenciálegyenlet megoldásának numerikus approximációját kapjuk. Amennyiben a megoldandó probléma valamilyen valós fizikai probléma approximációja, a szimbolikus számítások vonzó lehetőségei gyakran érvényüket vesztik; egyszerűen mert túl bonyolultak vagy túl lassúak ahhoz, hogy hasznosak vagy szükségesek legyenek, hiszen a megoldást is numerikusan keressük. Más esetekben, ha a probléma szimbolikusan kezelhetetlen, az egyetlen lehetséges út a numerikus approximációs módszerek használata. Ilyen akkor fordulhat elő, ha a létező szimbolikus algoritmus nem talál zárt megoldást (pl. nem elemi függvények integrálja stb.), vagy nem létezik a problémát kezelni tudó szimbolikus algoritmus. Ámbár egyre több numerikus algoritmusnak létezik szimbolikus megfelelője, a numerikus eljárások nagyon fontosak a komputeralgebrában. Gondoljunk csak a differenciál- és integrálszámítás területére: bizonyos esetekben a hagyományos algoritmusok – integráltranszformációk, hatványsor-approximációk, perturbációs módszerek – lehetnek a legcélravezetőbbek.

A komputeralgebra algoritmusok tervezésénél a jövőben egyre nagyobb szerepet kapnak a párhuzamos architektúrájú számítógépek. Habár sok meglévő algoritmus „ránézésre" párhuzamosítható, nem biztos, hogy a jó szekvenciális algoritmusok párhuzamos architektúrákon is a legjobbak lesznek: az optimum talán egy teljesen különböző eljárással érhető el.

2.5.2. A komputeralgebra-rendszerek áttekintése

A komputeralgebra-rendszerek fejlődésének története szoros kapcsolatban áll az informatika és az algoritmikus matematika fejlődésével. A számítástechnika kezdeti időszakában a különböző tudományterületek művelői szimbolikus számítási igényeik megkönnyítése és felgyorsítása érdekében kezdték el fejleszteni az első komputeralgebra-rendszereket, amelyek átdolgozva, folyamatosan megújulva és a sokadik változatban napjainkban is jelen vannak. A hetvenes években jelentek meg az ***általános célú*** komputeralgebra-rendszerek, amelyeket beépített adatstruktúrák, matematikai függvények és algoritmusok széles tárháza jellemez, minél nagyobb felhasználói területet próbálva meg lefedni. A jelentős számítógépes erőforrásigény miatt robbanásszerű elterjedésük a nyolcvanas évek elejére, a mikroprocesszor alapú munkaállomások megjelenésének idejére tehető. A jobb hardver környezet, a hatékonyabb erőforrás-gazdálkodás, rendszerfüggetlen, magasszintű alapnyelv használata, és nem utolsósorban a társadalmi-gazdasági igények fokozatosan piaci termékké érlelték az általános célú komputeralgebra-rendszereket, ami viszont erős javulást hozott a felhasználói felület és dokumentumkészítés terén.

Az alábbiakban a legismertebb általános és speciális célú komputeralgebra rendszereket, könyvtárakat soroljuk fel.

- Általános célú komputeralgebra rendszerek: Axiom, Derive, Form, GNU-calc, Jacal, Macsyma, Maxima, Maple, Distributed Maple, MathCAD, Matlab Symbolic Math Toolbox, Scilab, MAS, Mathematica, Mathview, Mock-Mma, MuPAD, Reduce, Risa.

- Algebra és számelmélet: Bergman, CoCoA, Felix, Fermat, GRB, Kan, Macaulay, Magma, Numbers, Pari, Simath, Singular.
- Algebrai geometria: Casa, Ganith.
- Csoportelmélet: Gap, LiE, Magma, Schur.
- Tenzor analízis: Cartan, FeynCalc, GRG, GRTensor, MathTensor, RedTen, Ricci, TTC.
- Komputeralgebra könyvtárak: Apfloat, BigNum, GNU MP, Kant, LiDiA, NTL, Saclib, Ubasic, Weyl, Zen.

Az általános célú komputeralgebra rendszerek többsége olyan tulajdonságokkal bír, mint

- interaktivitás,
- matematikai tények ismerete,
- matematikai objektumok kezelésére képes deklaratív[2], magas szintű programozási nyelv funkcionális programozási lehetőségekkel,
- az operációs rendszer és más programok felé való kiterjeszthetőség,
- szimbolikus és numerikus számítások integrálása,
- automatikus (optimalizált) C és Fortran kódszegmens generálása,
- grafikus felhasználói környezet,
- 2 és 3 dimenziós grafika, animáció,
- szövegszerkesztési lehetőség és automatikus LaTeX konverzió,
- on-line súgó.

A komputeralgebra-rendszereket ***matematikai szakértői rendszereknek*** is nevezik. Napjainkban az általános célú komputeralgebra-rendszerek szédítő iramú fejlődésének lehetünk szemtanúi, elsősorban tudásuknak és széles körű alkalmazhatóságuknak köszönhetően. Mégis hiba volna alábecsülni a speciális rendszereket, amelyek egyrészt igen fontos szerepet játszanak sok tudományterületen, másrészt sok esetben könnyebben kezelhetők és hatékonyabbak, jelölésrendszerük és algoritmusaik alacsony szintű programnyelvi implementációja miatt. Nagyon lényeges, hogy egy adott probléma megoldásához az arra legalkalmasabb komputeralgebra-rendszert válasszuk ki.

Feladatok

2-1. Maradékos osztás maradéksorozata együtthatói hosszának vizsgálata
Generáljunk két ($n = 10$)-ed fokú pszeudovéletlen polinomot $\mathbb{Z}[x]$-ben $l = 10$ decimális jegyből álló együtthatókkal. Hajtsunk végre egyetlen maradékos osztást ($\mathbb{Q}[x]$-ben) és számítsuk ki a maradék polinom és az eredeti polinom (λ függvénnyel meghatározott) maximális együtthatóhosszának arányát. Ismételjük meg a számításokat ($t = 20$)-szor és számítsunk átlagot. Mennyi lesz a kapott érték? Ismételjük meg a fenti kísérletet $l = 100, 500, 1000$ esetén.

[2] A deklaratív programozási nyelvek a kívánt eredményt specifikálják és nem a hozzájuk vezető utat, mint ahogy az imperatív nyelvek teszik.

2-2.* Moduláris-lnko-kisprímek *algoritmus szimulációs vizsgálata
Szimulációs vizsgálattal adjunk becslést a Moduláris-lnko-kisprímek algoritmusban az n változó optimális értékére. Használjunk véletlen polinomokat különböző fokszám és együttható-nagyság esetén.

2-3. Módosított pszeudo-maradékos osztás
Legyen $f, g \in \mathbb{Z}[x]$, $\deg f = m \geq n = \deg g$. A pszeudo-maradékos osztást módosítsuk oly módon, hogy a

$$g_n^s f = gq + r$$

egyenletnél az $s = m - n + 1$ kitevő helyett a lehető legkisebb olyan $s \in \mathbb{N}$ érték szerepeljen, amellyel $q, r \in \mathbb{Z}[x]$. Az így származtatott spquo() és sprem() eljárásokkal helyettesítve a Primitív-Euklidesz algoritmus pquo() és prem() eljárásait véletlen polinomokat alkalmazva vessük össze az algoritmusok tárigényét.

2-4. Redukált Gröbner-bázis konstrukciója
Tervezzünk algoritmust, amely adott G Gröbner-bázisból redukált Gröbner-bázist állít elő.

2-5. A Hermite-redukció megvalósítása
Valósítsuk meg a Hermite-redukciót valamilyen választott komputeralgebra nyelven.

2-6. Racionális törtfüggvény integrálása
Írjunk programot a racionális törtfüggvények integrálására.

Megjegyzések a fejezethez

A Klasszikus-Euklidesz és Bővített-Euklidesz algoritmusok nemnegatív egész bemenet esetén működő változata megtalálható [90]-ben. A rezultánsok elméletének természetes folytatásaként juthatunk el a szubrezultánsokhoz, melynek segítségével a Bővített-Euklidesz algoritmus során látott együttható-növekedés jól kordában tartható (lásd például [152, 480]).

A Gröbner-bázisokat B. Buchberger vezette be 1965-ben [62]. Polinomideálok bázisaival több szerző is foglalkozott ezt megelőzően. A legismertebb talán Hironaka, aki 1964-ben $\mathbb{C}$ feletti szingularitások feloldására hatványsorok ideáljainak bázisát használta. Munkájáért Fields-érmet kapott. Azonban Hironaka módszere nem volt konstruktív. A Gröbner-bázisokat az utóbbi két évtizedben számos algebrai struktúrára általánosították.

A differenciálalgebra alapjait J. F. Ritt fektette le 1948-ban [389]. A szimbolikus integrálás során használt négyzetmentes felbontás algoritmusa megtalálható például [480, 152] könyvekben. A Hermite-redukcióban a testbővítés foka lehető legkisebbre választásának fontosságát igazolja a [152]-ben található 11.11. példa, amelyben a felbontási test rendkívül magas fokú, míg az integrál egy másodfokú testbővítésben felírható. A Rothstein–Trager-féle integráló algoritmus bizonyítása megtalálható [480]-ben (22.8. tétel). Megjegyezzük, hogy az algoritmust Rothstein és Trager egymástól függetlenül találták. A Lazard–Rioboo–Trager-formula helyességének bizonyítása, a Logaritmikus-rész-integrál algoritmus futási idejének elemzése, az algebrai kiterjesztési lépések nehézségének kezelésével kapcsolatos eljárások vázlatos ismertetése, a $C(x)$ felett hiperexponenciális elem hiperexponenciális integráljának meghatározása (ha ilyen létezik), a Liouville-elv bizonyítása, a Risch-algoritmussal kapcsolatos állítások bizonyításai megtalálhatók a [480] könyvben.

A komputeralgebráról és a kapcsolódó tudományterületekről számos publikáció és

könyv született. Magyar nyelven [90], [171] és [264] érhető el. Angolul az érdeklődőknek az összefoglaló matematikai jellegű munkák közül felsorolunk néhányat: Caviness [71], Davenport és társai [100], von zur Gathen és társai [480], Geddes és társai [152], Knuth [258, 259, 260], Mignotte [328], Mishra [334], Pavelle és társai [362], Winkler [498].

A komputeralgebra informatikai oldaláról az érdeklődő Olvasó további információt találhat az alábbiakban: Christensen [75], Gonnet és Gruntz [164], Harper és társai [193], valamint a világhálón.

Az alkalmazásokról könyvek és cikkek nagyon széles választéka áll rendelkezésre, pl. Akritas [8], Cohen és társai (ed.) [85, 86], Grossman (ed.) [179], Hearn (ed.) [198], Kovács [263] és Odlyzko [352].

A komputeralgebra-rendszerek oktatásban betöltött szerepéről lásd pl. Karian [239] és Uhl [472] munkáját.

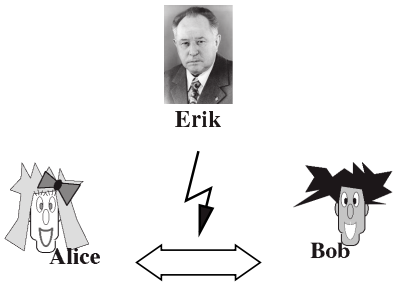
Konferencia kiadványok: AAECC, DISCO, EUROCAL, EUROSAM, ISSAC és SYMSAC.

Komputeralgebra folyóiratok: *Journal of Symbolic Computation* – Academic Press, *Applicable Algebra in Engineering, Communication and Computing* – Springer-Verlag, SIGSAM *Bulletin* – ACM Press.

Az Eötvös Loránd Tudományegyetem Informatikai Karának Komputeralgebra Tanszéke az oktatásban a [152, 328, 480, 498] munkákat veszi alapul.

3. Kriptográfia

Ebben a fejezetben a kriptográfiában használatos protokollokat, valamint alapvető problémákat és algoritmusokat mutatunk be. A kriptográfiában jellemző egyik alaphelyzetet láthatjuk a 3.1. ábrán. Aliz és Bob híreket cserélnek valamilyen nem biztonságos csatornán, például nyilvános telefonvonalon, vagy két hálózati kapcsolatban lévő számítógép közötti elektronikus posta segítségével. Erik megcsapolja a vezetéket, és lehallgatja a kommunikációt. Mivel Aliz és Bob tudják ezt, a küldeményeiket rejtjelezik.



3.1. ábra. A kriptográfiában tipikus alaphelyzet.

A 3.1. alfejezetben olyan szimmetrikus kriptográfiai rendszereket mutatunk be, amelyek az illetéktelen rejtjelfejtést hivatottak akadályozni. Szimmetrikusnak akkor nevezünk egy kriptográfiai rendszert, ha ugyanaz a kulcs használható rejtjelezésre (sifrírozásra) és visszafejtésre (desifrírozásra) is. Vajon hogyan kell Aliznak és Bobnak egy közös kulcsban megegyeznie, ha csupán egy megbízhatatlan csatornán tudnak kommunikálni egymással? Ha például Aliz választ egy kulcsot és rejtjelezi (mondjuk egy szimmetrikus rejtjelrendszerrel), majd elküldi Bobnak, azonnal felmerül a kérdés, milyen kulcsot használjanak ezeknek a kulcsoknak a rejtjelezéséhez.

Ez a paradoxnak tűnő kérdés – amely egy kicsit emlékeztet arra a másikra, hogy vajon a tojás, vagy a tyúk volt-e előbb – ***kulcscsere-problémaként*** ismert. A kriptográfia kezdetétől megoldhatatlannak tartották ezt a problémát. Annál nagyobb volt a meglepetés, amikor Whitfield Diffie és Martin Hellman 1976-ban megoldották. Egy protokollt javasoltak, amelyben Aliz és Bob információkat cserélnek, s ennek segítségével végül ki tudják számítani a közös kulcsukat. Az őket lehallgató Eriknek ugyanakkor fogalma sincs a kulcsukról, még akkor sem, ha az adatcserében résztvevő minden egyes bitet el tudja fogni. A 3.2. alfejezet bemutatja a Diffie–Hellman-protokollt.

A történet némileg ironikus része az, hogy éppen ez a protokoll, amely a szimmetrikus kriptográfiában oly fontos kulcscsere megoldhatatlannak tartott problémáját megoldotta, egy másik protokoll felfedezése előtt egyengette az utat, s ez utóbbiban a megbízhatatlan csatornán történő titkos kulcscsere már semmilyen szerepet nem játszik. Diffie és Hellman 1976-ban megjelent munkájukkal kaput nyitottak a modern ***nyilvános kulcsú kriptográfiának,*** és már két évvel később, 1978-ban, Rivest, Shamir és Adleman szélesre tárták ezt a kaput, amikor megalkották jól ismert RSA-rendszerükkel az első ***nyilvános kulcsú kriptorendszert.*** A 3.3. alfejezetben szerepel az RSA-rendszer, valamint egy, az RSA-n alapuló digitális aláírásra vonatkozó protokoll. Egy ilyen protokollal Aliz el tudja látni kézjegyével Bobnak küldött üzeneteit úgy, hogy Bob az üzenet küldőjének azonosságát ellenőrizni tudja. A digitális aláírások célja annak megakadályozása, hogy Erik Aliz üzenetét meghamisítva úgy tegyen, mintha azt Aliz küldte volna.

A Diffie–Hellman-protokoll biztonsága azon a feltételezésen alapszik, hogy a diszkrét logaritmus nem számítható ki hatékonyan. A moduláris hatványozás – amelynek inverz függvénye a diszkrét logaritmus – egy lehetséges egyirányú függvény. Az RSA biztonsága is egy probléma feltételezett nehézségén nyugszik, nevezetesen azon a feltételezésen, hogy nagy számok nem bonthatók hatékonyan prímtényezők szorzatára. A jogos címzett – Bob – ugyanakkor képes hatékonyan visszafejteni a rejtjelezett üzenetet, miközben egy általa választott szám faktorizációját saját „csapóajtó” információjaként használja fel.

A 3.4. alfejezetben bemutatunk egy Rivest–Sherman-protokollt, amelyik az úgynevezett erősen nem invertálható, asszociatív egyirányú függvényeken nyugszik, és a Diffie–Hellman-protokollhoz hasonlóan titkos kulcscserére szolgál. Ez a protokoll is módosítható oly módon, hogy digitális aláírás számára használható legyen.

Végül a 3.5. alfejezet az interaktív bizonyításrendszerek és a zéró-ismeretű protokollok érdekes területére vezet, amely a kriptográfiában gyakorlati felhasználásra kerül, nevezetesen a hitelesség problémájának megoldása során. Bemutatunk egy zéró-ismeretű protokollt a gráfizomorfizmus problémára. Másrészt ez a terület a bonyolultságelmélethez is kapcsolódik, és ezért a 4. fejezetben újból előkerül, ismét a gráfizomorfizmus problémával kapcsolatban.

3.1. Alapok

A ***kriptográfia*** évezredek óta létező művészet és tudomány, írások és üzenetek titkos anyagba történő olyan rejtjelezésével foglalkozik, amely illetéktelen személyek számára akadályozza a visszafejtést. Ebben az alfejezetben két klasszikus szimmetrikus kriptorendszert mutatunk be, a következő alfejezetekben pedig a manapság használatos legfontosabb protokollok és aszimmetrikus kriptorendszerek közül foglalkozunk néhánnyal. Protokollon két vagy több résztvevő közötti párbeszédet értünk, miközben egy résztvevő lehet egy ember, de lehet egy számítógép is. A kriptográfiai protokollok szövegek rejtjelezésére szolgálnak, oly módon, hogy a jogosult fogadó képes legyen egyszerűen és hatékonyan visszafejteni a rejtett szöveget. Egy protokoll algoritmusként is felfogható, melyet több résztvevő hajt végre.

A ***kriptoanalízis*** szintén több évezredes művészet és tudomány is, amely rejtjelezett üzenetek (illetéktelen) visszafejtésével és a létező kriptorendszerek feltörésével foglalkozik. A ***kriptológia*** ezt a két területet öleli fel, a kriptográfiát és a kriptoanalízist. Ebben

a fejezetben főként olyan ***kriptografikus*** algoritmusokra koncentrálunk, amelyek lehetővé teszik a biztonságos rejtjelezést. A kriptoanalízis algoritmusait – amelyekkel az ember próbálkozik, hogy kriptografikus protokollokat és rendszereket feltörjön –, megemlítjük, de nem vizsgáljuk teljes részletességgel.

3.1.1. Kriptográfia

A kriptográfia egyik jellemző alaphelyzete a 3.1. ábrán látható: Aliz és Bob olyan nem biztonságos csatornán kommunikálnak egymással, amelyet Erik lehallgat, és ezért az üzenetüket rejtjelezik egy kriptorendszer segítségével.

3.1. definíció (kriptorendszer). *A **kriptorendszer** egy $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ ötös a következő tulajdonságokkal:*

1. *$\mathcal{P}$, $\mathcal{C}$ és $\mathcal{K}$ véges halmazok, $\mathcal{P}$ a **nyílt szöveg tér**, $\mathcal{C}$ a **rejtett szöveg tér** és $\mathcal{K}$ a **kulcstér**. $\mathcal{P}$ elemeit **nyílt szövegnek** $\mathcal{C}$ elemeit pedig **rejtett szövegnek** nevezzük. Egy **üzenet** a nyílt szöveg szimbólumaiból álló szó.*
2. *$\mathcal{E} = \{E_k \mid k \in \mathcal{K}\}$ azoknak az $E_k : \mathcal{P} \to \mathcal{C}$ függvényeknek a családja, amelyeket a rejtjelezéshez használunk. $\mathcal{D} = \{D_k \mid k \in \mathcal{K}\}$ azoknak a $D_k : \mathcal{C} \to \mathcal{P}$ függvényeknek a családja, amelyeket a visszafejtéshez használunk.*
3. *Mindegyik $e \in \mathcal{K}$ kulcshoz van egy $d \in \mathcal{K}$ kulcs, melyekre minden $p \in \mathcal{P}$ nyílt szöveg esetén*

$$D_d(E_e(p)) = p. \tag{3.1}$$

*Egy kriptorendszerre azt mondjuk, hogy **szimmetrikus** (vagy titkos kulcsú), ha $d = e$ vagy legalábbis d „könnyen” kiszámítható e-ből. Egy **kriptorendszerre** azt mondjuk, hogy **aszimmetrikus** (vagy nyilvános kulcsú, „public-key”), ha $d \neq e$, és „gyakorlatilag nem végezhető el” az a feladat, hogy a d kulcsot e-ből kiszámítjuk. Ekkor e-t **nyilvános kulcsnak** nevezzük, és d az e**-hez tartozó titkos kulcs**.*

Néha különböző kulcstereket használnak a rejtjelezéshez és a visszafejtéshez, ami a fenti definíció megfelelő módosítását vonja maga után.

A klasszikus kriptorendszerek bemutatására példaként legyen a $\Sigma = \{\mathrm{A}, \mathrm{B}, \ldots, \mathrm{Z}\}$ ábécé a nyílt szöveg tér és ugyanakkor a rejtett szöveg tér is. Abból a célból, hogy a betűkkel úgy tudjunk számolni, mintha számok lennének, feleltessük meg Σ-nak $\mathbb{Z}_{26} = \{0, 1, \ldots, 25\}$-öt. A 0 szám tehát az A betűnek felel meg, az 1 a B betűnek stb. A nyílt szöveg szimbólumainak ez a természetes számokkal való kódolása természetesen nem tartozik hozzá a tényleges rejtjelezéshez, illetve visszafejtéshez.

Az üzenetek Σ^* elemei, ahol Σ^* a Σ fölötti szavak halmazát jelöli. Ha valamely $m \in \Sigma^*$ üzenet n hosszúságú blokkokra van felosztva, és blokkonként van rejtjelezve – ahogyan az sok kriptorendszerben szokásos –, akkor m egyes blokkjait $\mathbb{Z}_{26}^n$ elemeiként értelmezhetjük.

3.1. példa. Eltolásos rejtjelező. Az első példa egy monoalfabetikus szimmetrikus kriptorendszer. Legyen $\mathcal{K} = \mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}$. Az ***eltolásos rejtjel*** az a rejtett üzenet, amelyben a nyílt szöveg minden jelének az ábécében a k számú betűvel eltolt jel felel meg (modulo 26), ahol k rögzített szám. Az itt szereplő $k \in \mathbb{Z}_{26}$ a kulcs. Ha a rejtett szöveg minden jelét ugyanennek a k kulcsnak a felhasználásával visszatoljuk, akkor feltárul előttünk a nyílt szöveg. Az E_k rejtjelező függvényt és a D_k visszafejtő

m	B U D A P E S T K E L E T P Á R I Z S A
c	E X G D S H V W N H O H W S D U L C V D

3.2. ábra. Példa a Caesar-rejtjelezővel történő sifrírozásra.

függvényt mindegyik $k \in \mathbb{Z}_{26}$ kulcs esetén az alábbi módon definiáljuk:

$$\begin{aligned} E_k(m) &= (m + k) \bmod 26 \\ D_k(c) &= (c - k) \bmod 26, \end{aligned}$$

ahol a k-val való összeadást és kivonást jelenként, modulo 26 kell elvégezni.

A 3.2. ábrán látható egy magyar m üzenetnek a c kóddal történő rejtjelezése $k = 3$ esetén.[1] Az eltolással való rejtjelezés ezzel a speciális $k = 3$ kulccsal ***Caesar-rejtjelező*** néven is ismeretes, mert feltehetően a római császár ezt használta háborúiban a katonai üzenetek titkosítására.[2] Ez egy nagyon egyszerű helyettesítéses rejtjelező, amelyikben a nyílt szöveg minden betűjét a rejtjel ábécé egy meghatározott betűjével helyettesítenek.

Mivel a kulcstér nagyon kicsi, az eltolásos rejtjelet igen könnyű feltörni.

Ez a rejtjel már arra a támadásra is érzékeny, amelynek során a támadó csupán a rejtett szöveget ismeri (***rejtett szövegű támadás***, „ciphertext-only attacks"). Ha egyszerűen a 26 lehetséges kulcs mindegyikét végigpróbáljuk, kiderül, hogy melyik kulcs esetén adódik értelmes nyílt szöveg, ha pedig a rejtett szöveg elég hosszú, csak egy megfelelő rejtjelezés adódik.

A Caesar-rejtjelező ***monoalfabetikus kriptorendszer***, mert a nyílt szöveg minden betűjének a rejtett szövegben mindig ugyanaz a betű felel meg. Ezzel szemben egy ***polialfabetikus kriptorendszerben*** lehetséges, hogy ugyanannak a nyílt szövegbeli szimbólumnak különböző rejtett szövegbeli szimbólumok felelnek meg, attól függően, hogy a szöveg melyik helyén állnak. Egy ilyen polialfabetikus kriptorendszert, amelyik az eltolásos rejtjelezésre épül, de jóval nehezebb feltörni, Blaise de Vigenère (1523–1596) francia diplomata javasolt. Rendszere Leon Battista Alberti (szül. 1404) itáliai matematikus, Johannes Trithemius (szül. 1492) német apát és Giovanni Porta (szül. 1535) itáliai természettudós előmunkáira épül. Ez a rejtjelező úgy működik, mint az eltolásos rejtjelező, azzal a különbséggel, hogy a betűk, amelyekkel a nyílt szöveg egy szimbóluma sifrírozva van, most még a szövegbeli helyüknek megfelelően is változnak.

3.2. példa. *Vigenère-rejtjelező.* Ez a szimmetrikus polialfabetikus kriptorendszer egy úgynevezett ***Vigenère-négyzetet*** használ (lásd a 3.3. ábrát). Ez egy 26 sorból és 26 oszlopból álló mátrix. Mindegyik sorban az ábécé 26 betűje szerepel, sorról sorra mindig egy pozícióval balra tolva. Ez azt jelenti, hogy az egyes sorok felfoghatók egy-egy eltolásos rejtjelezőként, ahol a kulcsok sorban $0, 1, \ldots, 25$. A szimbólum szövegbeli helyétől függ az, hogy a Vigenère-négyzet melyik sorát használja az ember egy nyílt szövegbeli szimbólum rejtésére.

Az üzeneteket n állandó hosszúságú blokkra osztjuk és blokkonként rejtjelezzük, vagyis $\mathcal{K} = \mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}^n$. Az n blokkhosszat a rendszer ***periódusának*** is nevezik. Egy w szóban az i-edik betűt

[1] Az ékezetes betűk kódja ugyanaz, mint a megfelelő ékezet nélkülieké. *A fordító.*

[2] Történelmi megjegyzés: Gaius Julius Caesar *De Bello Gallico* című művében tudósít arról, hogy a gall háborúban (58–50 Kr.e.) hogyan küldött egy rejtjelezett üzenetet Q. Tullius Ciceronak (a híres szónok testvéröccsének), aki légiójával ostrom alatt volt. Az alkalmazott rendszer monoalfabetikus volt, és a latin betűket göröggel helyettesítette, Caesar írásaiból nem derül ki azonban az, hogy valóban a $k = 3$ kulcsú eltolásos rejtjelezőről volt-e szó. Ezt az információt később Suetonius adta.

0	A	B	C	D	E	F	G	**H**	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
19	**T**	U	V	W	X	Y	Z	**A**	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
25	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

3.3. ábra. Vigenère-négyzet: a „H” nyílt szöveget a „T” kulccsal rejtjelezve „A”-t kapunk.

k	T O N Y T O N Y T O N Y T O N Y T O N Y T O N Y
m	H U N G A R I A N I S A L L G R E E K T O G E R M A N S
c	A I A E T F V Y G W F Y E Z T P X S X R H U R P F O A Q

3.4. ábra. Példa a Vigenère-rejtjelezőt használó sifrírozásra.

w_i-vel jelöljük.

Az E_k rejtjelező függvényt és a D_k visszafejtő függvényt, amelyek mindketten $\mathbb{Z}_{26}^n$-ből $\mathbb{Z}_{26}^n$-be képeznek, mindegyik $k \in \mathbb{Z}_{26}^n$ kulcs esetén a következőképpen definiáljuk:

$$\begin{aligned} E_k(b) &= (b + k) \bmod 26 \\ D_k(c) &= (c - k) \bmod 26, \end{aligned}$$

ahol a k-val való összeadást és kivonást megint jelenként kell elvégezni modulo 26. Ez azt jelenti, hogy a megbeszélt $k \in \mathbb{Z}_{26}^n$ kulcsot betűről betűre a $b \in \mathbb{Z}_{26}^n$ blokk szimbólumai fölé írjuk. Ha a nyílt szöveg utolsó blokkjának kevesebb szimbóluma van, mint n, akkor a kulcsnak csak annyi jelét használjuk fel, amennyi szükséges. Amikor a nyílt szöveg i-edik b_i szimbólumát rejtjelezzük, fölötte a k_i kulcsszimbólum áll, és a Vigenère-négyzet k_i-edik sorát használjuk eltolásos rejtjelezőként.

Válasszuk például az $n = 4$ blokkhosszat, és a $k =$ TONY kulcsot. A 3.4. ábra egy angol nyelvű m nyílt szöveg rejtjelezését mutatja, amelyik hét blokkból áll, és a c rejtett szövegben a Vigenère-rejtjelezőt alkalmaztuk a k kulccsal. A nyílt szöveg első betűjéhez, a „H”-hoz a „T” kulcsszimbólumot rendeljük hozzá. A Vigenère-négyzet „H”-oszlopának és a “„T”-sorának kereszteződésében „A” található, ez lesz a rejtett szöveg első szimbóluma, amint azt a 3.3. ábrán láthatjuk.

Még számos további klasszikus kriptorendszer létezik, amelyeket azonban itt nem vizsgálunk meg közelebbről. Több lehetőség van arra is, hogy a kriptorendszereket jellegük

és tulajdonságaik alapján osztályozzuk. A 3.1. definíció megvilágítja a ***szimmetrikus*** és az ***aszimmetrikus*** kriptorendszerek közötti különbséget. A két bemutatott szimmetrikus rendszer (az eltolásos rejtjelező és a Vigenère-rejtjelező) rámutat a ***monoalfabetikus*** és a ***polialfabetikus*** rendszerek közötti eltérésekre. Mindkettő ***helyettesítéses rejtjelező***. Ezek szembeállíthatók a ***permutációs rejtjelezőkkel*** (más néven ***transzpozíciós rejtjelezőkkel***), amelyeknél a nyílt szöveg betűit nem a rejtjelábécé bizonyos betűi helyettesítenek, hanem csupán a szövegbeli helyük változik, egyébként azonban nem változnak meg.

Ha blokkonként n periódussal rejtjelezünk és Σ^n összes permutációinak halmazát alkalmazzuk kulcstérként, ahol Σ m betűs ábécé, akkor $m^n!$ különböző lehetőségünk van arra, hogy egy kulcsot kiválasszunk. Továbbá a ***blokkrejtjelezőt*** – amelyben, mint a Vigenère-rendszer esetén a nyílt szöveget blokkokra osztjuk, blokkonként rejtjelezzük –, egybevethetjük a folyamrejtjelekkel, amelyek egy folyamatos kulcsfolyamot hoznak létre. A blokkrejtjelezők különböző változatait vizsgálhatjuk. Egy fontos típust képviselnek az ***affin blokkrejtjelezők.*** Ezeket a következőképpen definiáljuk: az $E_{(A,\vec{b})}$ rejtjelező függvények, és a $D_{(A^{-1},\vec{b})}$ visszafejtő függvények $\mathbb{Z}_m^n$-ből $\mathbb{Z}_m^n$-be képeznek, és affin leképezések, ami azt jelenti, hogy az alábbi alakúak:

$$\begin{aligned} E_{(A,\vec{b})}(\vec{x}) &= A\vec{x} + \vec{b} \bmod m\,, \\ D_{(A^{-1},\vec{b})}(\vec{y}) &= A^{-1}(\vec{y} - \vec{b}) \bmod m\,. \end{aligned} \tag{3.2}$$

Itt $(A, \vec{b})$ és $(A^{-1}, \vec{b})$ a rejtjelezés illetve visszafejtés kulcsai; A $(n \times n)$-es mátrix $\mathbb{Z}_m$-beli elemekkel. A^{-1} az A inverz mátrixa; $\vec{x}$, $\vec{y}$ és $\vec{b}$ mind $\mathbb{Z}_m^n$-beli vektorok, a műveleteket pedig modulo m végezzük. Nézzünk néhány matematikai kiegészítést (lásd a 3.1.3. pontban a 3.2. definíciót is): egy – a $\mathbb{Z}_m$ gyűrű fölötti – $(n \times n)$-es A mátrixnak pontosan akkor van multiplikatív inverze, ha $\mathrm{lnko}(\det A, m) = 1$. A ***inverz mátrixát*** $A^{-1} = (\det A)^{-1} A_{\mathtt{adj}}$ segítségével definiáljuk, ahol $\det A$ az A determinánsa és $A_{\mathtt{adj}} = ((-1)^{i+j} \det A_{j,i})$ az A ***adjungált mátrixa***. A $\det A$ ***determinánsát*** rekurzív módon definiáljuk: $n = 1$ és $A = (a)$ esetén $\det A = a$; $n > 1$ esetén minden i-re, ahol $i \in \{1, 2, \ldots, n\}$, $\det A = \sum_{j=1}^{n} (-1)^{i+j} a_{i,j} \det A_{i,j}$, ahol $a_{i,j}$ az A-nak az (i, j) indexű eleme és az $(n-1) \times (n-1)$-es $A_{i,j}$ mátrixokat az A-ból úgy kapjuk, hogy az i-edik sort és a j-edik oszlopot töröljük. Egy mátrix determinánsát hatékonyan ki lehet számítani (lásd a 3-3. feladatot).

Példaként megemlítjük, hogy a Vigenère-rejtjelező affin rejtjelező, amelynek kulcstere, $\mathcal{K} = \mathbb{Z}_m^n$ pontosan m^n elemű (lásd a 3.2. példát). Ha (3.2)-ben a $\vec{b}$ nullvektor, akkor ***lineáris blokkrejtjelezőről*** van szó. Erre klasszikus példa a ***Hill-rejtjelező,*** amit 1929-ben Lester Hill talált ki. Ennél a rejtjelezőnél a kulcstér az összes olyan $(n \times n)$-es $\mathbb{Z}_m$-beli elemeket tartalmazó A mátrix, melyre $\mathrm{lnko}(\det A, m) = 1$. Emiatt a kulcsként megengedett mátrixok invertálhatók, s az A^{-1} inverz mátrixszal fejthetjük vissza az A mátrixszal rejtjelezett üzenetet. A Hill-rejtjelezőt az $E_A(\vec{x}) = A\vec{x} \bmod m$ rejtjelező függvénnyel és a $D_{A^{-1}}(\vec{y}) = A\vec{y} \bmod m$ visszafejtő függvénnyel definiáljuk. Ez a legáltalánosabb lineáris rejtjelező. A permutációs rejtjelezők szintén lineáris rejtjelezők, és így a Hill-rejtjelezők speciális esetei.

3.1.2. Kriptoanalízis

A kriptoanalízis feladata az, hogy adott kriptorendszereket feltörjön, elsősorban azért, hogy a szükséges kulcsot a visszafejtéshez megállapítsa. Attól függően, hogy milyen információk állnak a kriptoanalízis rendelkezésére, a támadások több típusát lehet megkülönböztetni, és

ezzel a vizsgált kriptorendszer biztonságát, illetve sérülékenységét jellemezni. Az eltolásos rejtjelezővel kapcsolatban már említettük a *rejtett szövegű támadást.* Ez a leggyengébb formája a támadásnak, és az a kriptorendszer, amelyik ennek a támadásnak sem áll ellen, nem sokat ér.

Az affin blokkrejtjelezők, mint például a Vigenère- és a Hill-rejtjelezők érzékenyek az olyan támadásokra, amelyeknél a támadó egy elcsípett rejtjelezett szöveget és a hozzátartozó nyílt szöveget is ismeri ***(ismert nyílt szövegű támadás,*** és ebből az alkalmazott kulcsra következtetni tud. Még inkább érzékenyek az olyan támadásokra, amelyeknél a támadó saját maga ki tud választani egy nyílt szöveget, és utána látja, hogy milyen rejtett szöveg tartozik hozzá ***(választott nyílt szövegű támadás, „chosen-plaintext attacks).***" A negyedik fajta támadás főként az aszimmetrikus kriptorendszerekre hatásos. Az ilyen (***rejtjelező kulcs támadás***, „encryption-key-attacks") esetén a támadó csupán a nyilvánosságra hozott kulcsot ismeri, de nem ismer rejtett üzenetet, és egyedül ebből az információból próbálja meghatározni a titkos kulcsot. A különbség az, hogy a támadónak van ideje számításokat végezni, a többi támadás esetén azonban sietnie kell, mert az üzenetet már elküldték. Ezért kell aszimmetrikus kriptorendszerek esetén a kulcsteret nagynak választani, s ezzel szavatolni a rendszer biztonságát. Ennek következménye az, hogy a gyakorlatban sokszor kevéssé hatékonyak az aszimmetrikus rendszerek.

Az ilyen támadások gyakran a rejtjelezett szövegben előforduló betűk gyakoriságát vizsgálják. Emellett a nyílt szöveg számára alkalmazott természetes nyelv redundanciáját is kihasználják. Például sok természetes nyelvben az „E" betű statisztikusan szignifikáns módon a leggyakrabban fordul elő. Hosszú „tipikus" szövegek vizsgálata szerint az „E" gyakorisága az angolban 12.31%, a franciában 15.87%, a németben pedig 18.46%. Más nyelvekben más betűk léphetnek fel a legnagyobb gyakorisággal. „Tipikus" finn szövegekben például 12.06%-kal az „A" a leggyakoribb betű.

A gyakoriságelemzés szemmel láthatóan hasznos a monoalfabetikus kriptorendszerek elleni támadás során. Ha például egy eltolásos rejtjelezővel rejtett német szövegben az „Y" betű lép fel leggyakrabban, ami a németben – akárcsak a legtöbb nyelvben – ritka, akkor ebből arra következtethetünk, hogy ez az „E" rejtett változata, az alkalmazott kulcs pedig az „U" ($k = 20$) (lásd a 3.3. ábrát). Az egyes betűk gyakorisága mellett vizsgálhatjuk a betűpárok (digrammok), a betűhármasok (trigrammok) stb. A támadásnak ez a módja működik a polialfabetikus kriptorendszerek esetében is, feltéve, hogy a periódus (vagyis a blokkhossz) ismert.

Az ismeretlen periódusú polialfabetikus kriptorendszerek ezzel szemben nagyobb biztonságot nyújtanak. A Vigenère-rejtjelező például hosszú ideig ellenállt minden feltörési kísérletnek. Csak 1863-ban, mintegy 300 évvel a feltalálása után, talált módszert Friedrich Wilhelm Kasiski német kriproanalitikus a Vigenère-rejtjelező feltörésére. Megmutatta, hogyan lehet meghatározni az alkalmazott periódust a kulcsszöveg szavainak ismétlődéséből akkor is, ha a periódus kezdetben ismeretlen volt. Végül a gyakoriságelemzés segítségével a rejtjelezett szöveg visszafejthető. Singh írja, hogy a rendkívül sokoldalú Charles Babbage, akit sokan kora zsenijének tartanak, a Kasiski-féle módszert valószínűleg korábban,1854-ben felfedezte, bár nem hozta nyilvánosságra.

A kriptográfia történetében mérföldkőként kell megemlítenünk Claude Shannonnak (1916–2001). a modern kód- és információelmélet atyjának úttörő munkáját. Shannon bebizonyította, hogy vannak kriptorendszerek, amelyek egy bizonyos szigorú matematikai értelemben ***tökéletes titkosságot*** tesznek lehetővé. Pontosabban szólva egy kriptorendszer ak-

kor tökéletes titkosságú, ha $|\mathcal{C}| = |\mathcal{K}|$, a kulcsok $\mathcal{K}$-ban egyenletes eloszlásúak, és minden $p \in \mathcal{P}$-re és minden $c \in \mathcal{C}$-re ***pontosan egy*** $k \in \mathcal{K}$ kulcs van, amelyre $E_k(p) = c$. Ez azt jelenti, hogy egy ilyen kriptorendszer a legtöbb gyakorlati célra nem használható, mert ahhoz, hogy tökéletes titkosságot garantálhassunk, egyrészt minden kulcsnak legalább olyan hosszúnak kell lennie, mint a rejtjelezendő üzenet, másrészt egyszeri használat után minden kulcsot el kell dobni. Gyakorlati célra megfelelő kriptorendszereket később fogunk ebben a fejezetben bemutatni.

3.1.3. Algebra, számelmélet és gráfelmélet

A később sorra kerülő algoritmusok és problémák közül néhánynak a megértéséhez segítségünkre lesz az algebra, és főként a csoport- és számelmélet köréből néhány alapvető fogalom és tétel. Vonatkozik ez mind a 3. fejezetbeli kriptorendszerekre és zéró-ismeretű protokollokra, mind pedig néhány, a 4. fejezetben vizsgált problémára. Az Olvasó megteheti, hogy ezt a fejezetet átugorja, és a szükséges fogalmaknak, eredményeknek csak akkor néz utána, ha később felbukkannak. A bizonyításoktól ebben a fejezetben többnyire eltekintünk.

3.2. definíció (csoport, gyűrű, test).

- $\mathfrak{G} = (S, \circ)$ ***csoport****, ha* S *nem üres halmaz,* $\circ$ *kétváltozós művelet* S*-en és teljesülnek a következő axiómák:*
  - *zártság:* $(\forall x \in S)\,(\forall y \in S)\,[x \circ y \in S]$*;*
  - *asszociativitás:* $(\forall x \in S)\,(\forall y \in S)\,(\forall z \in S)\,[(x \circ y) \circ z = x \circ (y \circ z)]$*;*
  - *semleges elem:* $(\exists e \in S)\,(\forall x \in S)\,[e \circ x = x \circ e = x]$*;*
  - *inverz elem:* $(\forall x \in S)\,(\exists x^{-1} \in S)\,[x \circ x^{-1} = x^{-1} \circ x = e]$.

  Az e *elemet a* $\mathfrak{G}$ *csoport* ***semleges elemének,*** *az* x^{-1} *elemet* x ***inverzének*** *nevezzük.* $\mathfrak{G}$ ***félcsoport****, ha* $\mathfrak{G}$*-ben az asszociativitás és a zártság teljesül a* $\circ$ *műveletre, akkor is, ha* $\mathfrak{G}$*-nek nincs semleges eleme, vagy nem minden elemnek van inverze. Egy* $\mathfrak{G} = (S, \circ)$ *félcsoport, vagy csoport* ***kommutatív,*** *ha* $x \circ y = y \circ x$ *minden* $x, y \in S$ *esetén teljesül. Egy* $\mathfrak{G}$ *véges csoport elemeinek a száma a* $\mathfrak{G}$ ***rendje*** *és ezt* $|\mathfrak{G}|$ *jelöli.*
- $\mathfrak{H} = (T, \circ)$*-t a* $\mathfrak{G} = (S, \circ)$ ***csoport részcsoportjának*** *nevezzük (*$\mathfrak{H} \leq \mathfrak{G}$ *jelöli), ha* $T \subseteq S$ *és* $\mathfrak{H}$ *kielégíti a csoportaxiómákat.*
- *Egy* $\mathfrak{R} = (S, +, \cdot)$ *hármas* ***gyűrű****, ha* $(S, +)$ *Abel-csoport,* $(S, \cdot)$ *félcsoport és a disztributivitási szabály érvényben van:*

  $$(\forall x \in S)\,(\forall y \in S)\,(\forall z \in S)\,[(x \cdot (y + z) = (x \cdot y) + (x \cdot z)) \wedge ((x + y) \cdot z = (x \cdot z) + (y \cdot z))].$$

  Egy $\mathfrak{R} = (S, +, \cdot)$ *gyűrű* ***kommutatív****, ha az* $(S, \cdot)$ *félcsoport kommutatív. Az* $(S, +)$ *csoport semleges elemét (ha létezik) az* $\mathfrak{R}$ *gyűrű* ***nullelemének*** *(röviden* ***nullának****), az* $(S, \cdot)$ *félcsoport semleges elemét az* $\mathfrak{R}$ *gyűrű* ***egységelemének*** *(röviden* ***egy****) nevezzük.*
- *Legyen* $\mathfrak{R} = (S, +, \cdot)$ *egységelemes gyűrű.* $\mathfrak{R}$ *egy* x *elemét pontosan akkor nevezzük* ***invertálhatónak*** *(vagy* $\mathfrak{R}$ ***egységének****), ha ez az elem az* $(S, \cdot)$ *félcsoportban invertálható.* $\mathfrak{R}$ *valamely* x *eleme* ***nullosztó,*** *ha nem nulla, és egy* $\mathfrak{R}$*-beli nem nulla* y *elemmel* $x \cdot y = y \cdot x = 0$.

- ***Test*** *egy kommutatív, egységelemes gyűrű, amelyben minden, a nullától különböző elem invertálható.*

3.3. példa. *Csoport, gyűrű, test.* Legyen $1 < k \in \mathbb{N}$. A $\mathbb{Z}_k = \{0, 1, \ldots, k-1\}$ halmaz az egész számok modulo k tekintett összeadásával véges csoport, amiben semleges elem a 0. $\mathbb{Z}_k$ a modulo k vett összeadással és szorzással kommutatív egységelemes gyűrű (lásd a 3. 3-1. feladatot). Abban az esetben, ha p prímszám, akkor $\mathbb{Z}_p$ a modulo k tekintett összeadásra és szorzásra test.

Legyen $\text{lnko}(n, m)$ az n és m számok legnagyobb közös osztója. Legyen $1 < k \in \mathbb{N}$ esetén $\mathbb{Z}_k^* = \{i \mid 1 \leq i \leq k \text{ és } \text{lnko}(i, k) = 1\}$. A számok modulo k vett szorzásával $\mathbb{Z}_k^*$ véges csoport az 1 semleges elemmel.

Ha a szövegösszefüggésből egyértelműen kiderül, hogy valamely $\mathfrak{G} = (S, \circ)$ csoportban a $\circ$ a művelet, akkor nem szükséges explicit módon megadni. A 3.3. példabeli $\mathbb{Z}_k^*$ csoport fontos szerepet játszik a 3.3. alfejezetben, ahol az RSA kriptorendszert mutatjuk be. Ennek a csoportnak a rendjét az ***Euler-féle*** φ ***függvény*** adja meg, vagyis $\varphi(k) = |\mathbb{Z}_k^*|$. φ alábbi tulajdonságai a definíció következményei:

- $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$ minden $m, n \in \mathbb{N}$ esetén, ha $\text{lnko}(m, n) = 1$ és
- $\varphi(p) = p - 1$ minden p prímszámra.

Ezeknek a tulajdonságoknak a bizonyítását az Olvasóra hagyjuk (lásd a 3.1-3. gyakorlatot). Főként a 3.3.1. pontban használjuk az alábbi állítást, amely ezekből a tulajdonságokból közvetlenül következik.

3.3. állítás. *Ha $n = p \cdot q$, ahol p és q különböző prímszámok, akkor $\varphi(n) = (p-1)(q-1)$.*

A Lagrange-tétel szerint egy véges, multiplikatív $\mathfrak{G}$ csoport minden a elemére és az e semleges elemre fennáll az $a^{|\mathfrak{G}|} = e$ összefüggés, ahol $|\mathfrak{G}|$ a csoport rendjét jelöli. Az alábbi, úgynevezett Euler-tétel ennek speciális esete (a $\mathbb{Z}_n^*$ csoportra). Az Euler-tétel speciális esete pedig, ha n prímszám, és a-nak nem osztója, „kis Fermat-tételként" lett ismert.

3.4. tétel (Euler). *Minden $a \in \mathbb{Z}_n^*$ esetén $a^{\varphi(n)} \equiv 1 \bmod n$.*

3.5. következmény (kis Fermat-tétel). *Ha p prímszám és $a \in \mathbb{Z}_p^*$, akkor $a^{p-1} \equiv 1 \bmod p$.*

A 4.4. alfejezetben gráfizomorfizmus problémára mutatunk algoritmusokat. Ez a probléma, amely a 3.5.2. pontban található zéró-ismeretű protokollnál fontos szerepet játszik, bizonyos csoportelméleti problémák speciális eseteként is felfogható. Különösen fontosak itt a ***permutációcsoportok.***

3.6. definíció (permutációcsoport).

- *Egy* ***permutáció*** *valamely halmaz bijektív leképezése önmagára. Egy $n \geq 1$ természetes szám esetén legyen $[n] = \{1, 2, \ldots, n\}$. Az $[n]$ összes permutációinak halmazát $\mathfrak{S}_n$ jelöli. Algoritmikus célokra a $\pi \in \mathfrak{S}_n$ permutációt n darab $[n] \times [n]$-beli $(i, \pi(i))$ rendezett párból álló listával reprezentáljuk.*
- *Ha $\mathfrak{S}_n$-en permutációk kompozíciójaként definiálunk egy műveletet, akkor $\mathfrak{S}_n$ csoport lesz. Ha π és τ $\mathfrak{S}_n$-beli permutációk, akkor $\pi\tau$* ***kompozíciójuk*** *az az $\mathfrak{S}_n$-beli permutáció, amelyet úgy kapunk, hogy először π-t, azután τ-t alkalmazzuk $[n]$ elemeire, tehát*

$(\pi\tau)(i) = \tau(\pi(i))$ *minden* $i \in [n]$ *esetén. Az* $\mathfrak{S}_n$ *permutációcsoport semleges eleme az* ***identikus permutáció,*** *amelyet a következőképpen definiálunk:* $\mathrm{id}(i) = i$ *minden* $i \in [n]$ *esetén.* $\mathfrak{S}_n$*-nek azt a részcsoportját, amely csupán az* id *elemet tartalmazza,* **id** *jelöli.*

- $\mathfrak{S}_n$*-nek valamely* $\mathfrak{T}$ *részhalmaza esetén a* $\mathfrak{T}$ ***által generált*** $\langle\mathfrak{T}\rangle$ ***permutációcsoportot*** $\mathfrak{S}_n$ *legszűkebb olyan részcsoportjaként definiáljuk, amely* $\mathfrak{T}$*-t tartalmazza. Az* $\mathfrak{S}_n$ *valamely* $\mathfrak{G}$ *részcsoportját az őt generáló halmazzal reprezentáljuk, amelyet* $\mathfrak{G}$ ***generátorrendszerének*** *is nevezzük.* $\mathfrak{G}$*-ben valamely* $i \in [n]$ ***elem pályáját*** $\mathfrak{G}(i) = \{\pi(i) \mid \pi \in \mathfrak{G}\}$ *definiálja.*
- $[n]$ *valamely* T *részhalmaza esetén* $\mathfrak{S}_n^T$ *az* $\mathfrak{S}_n$*-nek az a részcsoportja, amelyik* T *minden elemét önmagára képezi. Ha* $i \leq n$ *és* $\mathfrak{S}_n$ *részcsoportja* $\mathfrak{G}$*, akkor* $[i]$ $\mathfrak{G}$***-beli (pontonkénti) stabilizátorát*** *az alábbi módon definiáljuk:*

$$\mathfrak{G}^{(i)} = \{\pi \in \mathfrak{G} \mid \pi(j) = j \text{ minden } j \in [i] \text{ esetén}\} .$$

A $\mathfrak{G}^{(n)} = \mathbf{id}$ *és* $\mathfrak{G}^{(0)} = \mathfrak{G}$ *összefüggések fennállnak.*

- *Legyenek* $\mathfrak{G}$ *és* $\mathfrak{H}$ *permutációcsoportok, és* $\mathfrak{H} \leq \mathfrak{G}$*.* $\tau \in \mathfrak{G}$ *esetén* $\mathfrak{H}\tau = \{\pi\tau \mid \pi \in \mathfrak{H}\}$ $\mathfrak{H}$ ***jobb oldali mellékosztálya*** $\mathfrak{G}$***-ben****.* $\mathfrak{H}$ *két jobb oldali mellékosztálya* $\mathfrak{G}$*-ben azonos, vagy diszjunkt. Emiatt a* $\mathfrak{G}$ *permutációcsoportot* $\mathfrak{H}$ $\mathfrak{G}$*-beli jobb oldali mellékosztályai osztályozzák:*

$$\mathfrak{G} = \mathfrak{H}\tau_1 \cup \mathfrak{H}\tau_2 \cup \cdots \cup \mathfrak{H}\tau_k. \tag{3.3}$$

$\mathfrak{H}$ *mindegyik* $\mathfrak{G}$*-beli jobb oldali mellékosztályának a számossága* $|\mathfrak{H}|$*. A* $\{\tau_1, \tau_2, \ldots, \tau_k\}$ *(3.3) halmazt* $\mathfrak{H}$ $\mathfrak{G}$***-beli jobb oldali reprezentánsrendszerének*** *nevezzük.*

A pontonkénti stabilizátor fogalma különösen fontos olyan algoritmusok tervezésénél, amelyek permutációcsoportokkal dolgoznak. Az a lényeges struktúra, amelyet ilyenkor használunk azt úgy nevezzük, hogy $\mathfrak{G}$ permutációcsoportbeli ***stabilizátor lánc:***

$$\mathbf{id} = \mathfrak{G}^{(n)} \leq \mathfrak{G}^{(n-1)} \leq \cdots \leq \mathfrak{G}^{(1)} \leq \mathfrak{G}^{(0)} = \mathfrak{G} .$$

Minden i-re $1 \leq i \leq n$ esetén legyen $\mathfrak{T}_i$ a $\mathfrak{G}^{(i)}$-nek $\mathfrak{G}^{(i-1)}$-ben teljes jobb oldali reprezentánsrendszere. Ekkor azt mondjuk, hogy $\mathfrak{T} = \bigcup_{i=1}^{n-1} \mathfrak{T}_i$ ***erős generátora*** $\mathfrak{G}$***-nek,*** és $\mathfrak{G} = \langle\mathfrak{T}\rangle$. Ekkor minden $\pi \in \mathfrak{G}$ egyértelműen faktorizálható $\pi = \tau_1\tau_2\cdots\tau_n$ alakban, ahol $\tau_i \in \mathfrak{T}_i$. A permutációcsoportokra vonatkozó alábbi algoritmuselméleti eredmények hasznosak lesznek később, a 4.4. alfejezetben.

3.7. tétel. *Ha a* $\mathfrak{G} \leq \mathfrak{S}_n$ *permutációcsoport a generátorrendszerével adott, akkor fennáll a következő:*

1. *Minden* $i \in [n]$ *esetén* i $\mathfrak{G}(i)$ *pályája* $\mathfrak{G}$*-ben polinomiális idő alatt kiszámítható.*
2. *Az* $\mathbf{id} = \mathfrak{G}^{(n)} \leq \mathfrak{G}^{(n-1)} \leq \cdots \leq \mathfrak{G}^{(1)} \leq \mathfrak{G}^{(0)} = \mathfrak{G}$ *stabilizátorlánc kiszámítható* n*-ben polinomidő alatt, ami azt jelenti, hogy meghatározzuk minden* i*,* $1 \leq i \leq n$ *esetén* $\mathfrak{G}^{(i)}$ *teljes jobb oldali reprezentánsrendszerét* $\mathfrak{T}_i$*-t* $\mathfrak{G}^{(i-1)}$*-ben és ezzel* $\mathfrak{G}$ *egy erős generátorrendszerét.*

Azokat a fogalmakat, amelyeket a 3.6. definícióban vezettünk be a permutációcsoporttal kapcsolatban, most a gráfelméletből vett konkrét példák alapján fogjuk megvilágítani. Elsősorban gráfok automorfizmuscsoportját és izomorfizmuscsoportját vizsgáljuk. Ehhez szükségünk lesz néhány gráfelméleti fogalomra.

3.8. definíció (gráfok izomorfizmusa és automorfizmusa). *Egy G gráf egy véges $V(G)$ csúcshalmazból, és egy véges $E(G)$ élhalmazból áll, melyben lévő élek bizonyos csúcsokat összekötnek egymással. Feltesszük, hogy nincsenek párhuzamos- és hurokélek. Ebben az alfejezetben kizárólag irányítatlan gráfokat vizsgálunk, ami azt jelenti, hogy az éleknek nincs irányításuk és rendezetlen csúcspárokként foghatjuk fel őket. Két gráf, G és H,* ***diszjunkt egyesítését,*** *$G \cup H$-t, amelyek $V(G)$ és $V(H)$ csúcshalmazai átnevezéssel diszjunkttá tehetők, a $V(G) \cup V(H)$ csúcshalmazzal és az $E(G) \cup E(H)$ élhalmazzal definiáljuk.*

Legyen G és H két gráf ugyanannyi csúccsal. G ***és*** *H* ***közötti izomorfizmus*** *G csúcshalmazáról H csúcshalmazára képező éltartó bijekció. Állapodjunk meg abban, hogy $V(G) = \{1, 2, \ldots, n\} = V(H)$. Ekkor tehát G és H pontosan akkor izomorf (röviden $G \cong H$), ha van olyan $\pi \in \mathfrak{S}_n$ permutáció, hogy minden $i, j \in V(G)$ csúcsra fennáll a következő:*

$$\{i, j\} \in E(G) \iff \{\pi(i), \pi(j)\} \in E(H)\ . \tag{3.4}$$

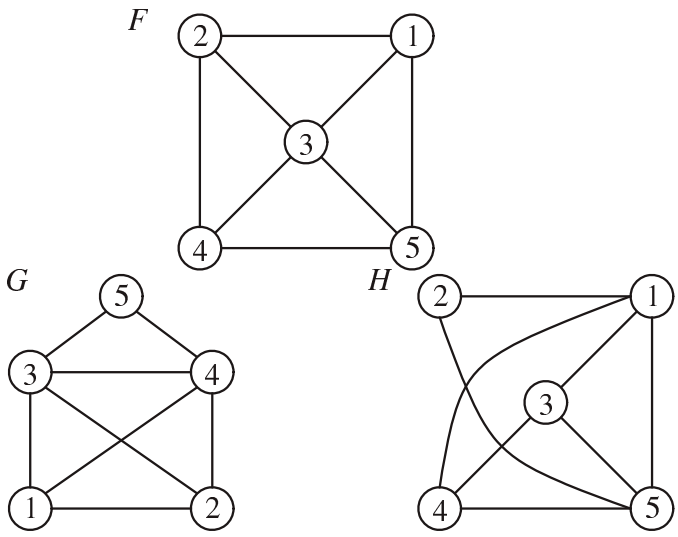
G ***egy automorfizmusa*** *G csúcshalmazának önmagára való éltartó bijekciója. Minden gráf esetén fennáll az* id *triviális automorfizmus. Jelöljük* $\mathrm{Iso}(G, H)$*-val a G és H közötti összes izomorfizmus halmazát,* $\mathrm{Aut}(G)$*-vel pedig G összes automorfizmusának halmazát. A* ***gráfizomorfizmus problémát*** *(röviden* `GI`*) és a* ***gráfautomorfizmus problémát*** *(röviden* `GA`*) a következő módon definiáljuk:*

$$\begin{aligned} \texttt{GI} &= \{(G, H) \mid G \text{ és } H \text{ izomorf gráfok}\}\ ; \\ \texttt{GA} &= \{G \mid G \text{ tartalmaz nem triviális automorfizmust}\}\ . \end{aligned}$$

Algoritmikus célokra a gráfokat vagy csúcs- és él-listájukkal, vagy csúcsmátrixukkal reprezentáljuk, amelyben az (i, j) helyen 1 áll, ha $\{i, j\}$ egy él a gráfban, különben pedig 0. Valamely gráfnak ehhez az előállításához elegendő a $\Sigma = \{0, 1\}$ ábécével való kódolás. Ha gráfpárokat kívánunk szemléltetni, akkor olyan bijektív párosítási $(\cdot, \cdot)$ függvényt alkalmazunk, amely $\Sigma^* \times \Sigma^*$-ról Σ^*-ra képez, polinomiális időben kiszámítható, és van polinomiális időben kiszámítható inverze.

3.4. példa. (Gráfok izomorfizmusa és automorfizmusa.) A 3.5. ábrán látható G és H gráfok izomorfak. Egy $\pi : V(G) \to V(H)$ izomorfizmus, amelynél (3.4)-nek megfelelően adjuk meg a csúcsok szomszédosságát, megadható például a következő módon: $\pi = \left(\begin{smallmatrix}1&2&3&4&5\\3&4&1&5&2\end{smallmatrix}\right)$, illetve ciklikus módon írva $\pi = (1\,3)(2\,4\,5)$. G és H között van még három további izomorfizmus, és így $|\mathrm{Iso}(G, H)| = 4$, (lásd a 3.1-4. gyakorlatot). G és H azonban nem izomorfak F-fel. Ez azonnal látható, ha a ***fokszámok*** sorozatát nézzük (vagyis minden csúcs esetén a csúcsra illeszkedő élek számát). A G-hez, illetve H-hoz tartozó sorozat különbözik az F-hez tartozótól: G és H esetén ez a sorozat $(2, 3, 3, 4, 4)$, miközben F esetén $(3, 3, 3, 3, 4)$. G egy nem triviális $\varphi : V(G) \to V(G)$ automorfizmusa például $\varphi = \left(\begin{smallmatrix}1&2&3&4&5\\2&1&4&3&5\end{smallmatrix}\right)$, illetve $\varphi = (1\,2)(3\,4)(5)$ segítségével adható meg, egy másik pedig $\tau = \left(\begin{smallmatrix}1&2&3&4&5\\1&2&4&3&5\end{smallmatrix}\right)$ illetve $\tau = (1)(2)(3\,4)(5)$ segítségével. G-nek van még két további automorfizmusa és így $|\mathrm{Aut}(G)| = 4$ (lásd a 3.1-4. gyakorlatot).

Az $\mathrm{Iso}(G, H)$, $\mathrm{Aut}(F)$, $\mathrm{Aut}(G)$ és $\mathrm{Aut}(H)$ permutációcsoportok $\mathfrak{S}_5$ részcsoportjai. $\mathrm{Aut}(G)$-ben az $\mathrm{Aut}(G)^{(5)} \leq \mathrm{Aut}(G)^{(4)} \leq \cdots \leq \mathrm{Aut}(G)^{(1)} \leq \mathrm{Aut}(G)^{(0)}$ stabilizátorlánc az $\mathrm{Aut}(G)^{(5)} = \mathrm{Aut}(G)^{(4)} =$



3.5. ábra. A három gráf: G és H izomorfak, de F nem izomorf velük.

$\mathrm{Aut}(G)^{(3)} = \mathbf{id}$, $\mathrm{Aut}(G)^{(2)} = \mathrm{Aut}(G)^{(1)} = \langle\{\mathrm{id}, \tau\}\rangle$ és $\mathrm{Aut}(G)^{(0)} = \mathrm{Aut}(G)$ részcsoportokból áll. G automorfizmuscsoportjában, $\mathrm{Aut}(G)$-ben az 1 és 2 csúcsok pályája $\{1, 2\}$, a 3 és 4 csúcsok pályája $\{3, 4\}$, és az 5 pályája $\{5\}$.

Az $\mathrm{Iso}(G, H)$ és $\mathrm{Aut}(G)$ permutációcsoportok rendje pontosan akkor azonos, ha G és H izomorfak. Ha ugyanis G és H izomorfak, akkor $|\mathrm{Iso}(G, H)| = |\mathrm{Aut}(G)|$ következik abból, hogy $\mathrm{Aut}(G) = \mathrm{Iso}(G, G)$. Másrészt ha $G \ncong H$, akkor $\mathrm{Iso}(G, H)$ üres, ugyanakkor $\mathrm{Aut}(G)$-ben mindig benne van az id triviális automorfizmus. Ebből következik a 3.9. lemma (3.5) állítása, amire később a 4.4. alfejezetben szükségünk lesz. A (3.6) igazolására tegyük fel, hogy G és H összefüggőek; ellenkező esetben G illetve H helyett a $\overline{G}$ illetve $\overline{H}$ komplementer gráfokat tekintjük, (lásd a 3.1-5. gyakorlatban). $G \cup H$ egy automorfizmusa, amely G és H csúcsait felcseréli, egy $\mathrm{Iso}(G, H)$-beli és egy $\mathrm{Iso}(H, G)$-beli izomorfizmusból tevődik össze. Így $|\mathrm{Aut}(G \cup H)| = |\mathrm{Aut}(G)| \cdot |\mathrm{Aut}(H)| + |\mathrm{Iso}(G, H)|^2$, amiből (3.5) felhasználásával következik (3.6).

3.9. lemma. *Bármely két G és H gráf esetén fennállnak a következők:*

$$|\mathrm{Iso}(G,H)| = \begin{cases} |\mathrm{Aut}(G)| = |\mathrm{Aut}(H)|, & \text{ha } G \cong H \\ 0, & \text{ha } G \ncong H \ ; \end{cases} \tag{3.5}$$

$$|\mathrm{Aut}(G \cup H)| = \begin{cases} 2 \cdot |\mathrm{Aut}(G)| \cdot |\mathrm{Aut}(H)|, & \text{ha } G \cong H \\ |\mathrm{Aut}(G)| \cdot |\mathrm{Aut}(H)|, & \text{ha } G \ncong H \ . \end{cases} \tag{3.6}$$

Ha G és H izomorf gráfok, és $\tau \in \mathrm{Iso}(G, H)$, akkor $\mathrm{Iso}(G, H) = \mathrm{Aut}(G)\tau$. Ez azt jelenti, hogy $\mathrm{Iso}(G, H)$ $\mathrm{Aut}(G)$-nek jobb oldali mellékosztálya $\mathfrak{S}_n$-ben. Mivel két jobb oldali mellékosztály azonos, vagy diszjunkt, $\mathfrak{S}_n$ (3.3) szerint $\mathrm{Aut}(G)$ jobb oldali mellékosztályaira bomlik:

$$\mathfrak{S}_n = \mathrm{Aut}(G)\tau_1 \cup \mathrm{Aut}(G)\tau_2 \cup \cdots \cup \mathrm{Aut}(G)\tau_k \ , \tag{3.7}$$

ahol $|\mathrm{Aut}(G)\tau_i| = |\mathrm{Aut}(G)|$ minden i, $1 \le i \le k$ esetén. Ezek szerint az $\mathfrak{S}_n$ permutációiból álló $\{\tau_1, \tau_2, \ldots, \tau_k\}$ halmaz $\mathrm{Aut}(G)$-nek $\mathfrak{S}_n$-beli jobb oldali reprezentánsrendszerét alkotja. Ha $\pi(G)$ azt a $H \cong G$ gráfot jelöli, amelyet akkor kapunk, ha a $\pi \in \mathfrak{S}_n$ permutációt G csúcsaira alkalmazzuk, akkor $\{\tau_i(G) \mid 1 \le i \le k\} = \{H \mid H \cong G\}$. Mivel $\mathfrak{S}_n$-ben pontosan $n! = n(n-1)\cdots 2 \cdot 1$ permutáció van, (3.7)-ből következik:

$$|\{H \mid H \cong G\}| = k = \frac{|\mathfrak{S}_n|}{|\mathrm{Aut}(G)|} = \frac{n!}{|\mathrm{Aut}(G)|} .$$

Ezzel beláttuk az alábbi következményt.

3.10. következmény. *Ha a G gráfnak n csúcsa van, akkor pontosan $n!/|\mathrm{Aut}(G)|$ vele izomorf gráf van.*

A 3.4. példa 3.5. ábráján lévő G gráffal így pontosan $5!/4 = 30$ gráf izomorf. A következő lemmára később, a 4.4. alfejezetben lesz szükségünk.

3.11. lemma. *Legyenek G és H n csúcsú gráfok. Definiáljuk az alábbi halmazt:*

$$A(G,H) \quad = \quad \{(F,\varphi) \mid F \cong G \text{ és } \varphi \in \mathrm{Aut}(F)\} \cup \{(F,\varphi) \mid F \cong H \text{ és } \varphi \in \mathrm{Aut}(F)\}.$$

Ekkor fennáll a következő:

$$|A(G,H)| \quad = \quad \begin{cases} n!, & \text{ha } G \cong H \\ 2n!, & \text{ha } G \ncong H. \end{cases}$$

Bizonyítás. Ha F és G izomorfak, akkor $|\mathrm{Aut}(F)| = |\mathrm{Aut}(G)|$. A 3.10. következményből adódik, hogy

$$|\{(F,\varphi) \mid F \cong G \text{ és } \varphi \in \mathrm{Aut}(F)\}| = \frac{n!}{|\mathrm{Aut}(F)|} \cdot |\mathrm{Aut}(F)| = n!,$$

és hasonlóan megmutatható az is, hogy $|\{(F,\varphi) \mid F \cong H \text{ és } \varphi \in \mathrm{Aut}(F)\}| = n!$. Ha G és H izomorfak, akkor

$$\{(F,\varphi) \mid F \cong G \text{ és } \varphi \in \mathrm{Aut}(F)\} \quad = \quad \{(F,\varphi) \mid F \cong H \text{ és } \varphi \in \mathrm{Aut}(F)\} .$$

Ebből következik, hogy $|A(G,H)| = n!$. Ha pedig G és H nem izomorfak, akkor $\{(F,\varphi) \mid F \cong G \text{ és } \varphi \in \mathrm{Aut}(F)\}$ és $\{(F,\varphi) \mid F \cong H \text{ és } \varphi \in \mathrm{Aut}(F)\}$ diszjunkt halmazok. Eszerint $|A(G,H)| = 2n!$. ■

Gyakorlatok

3.1-1. A 3.6. ábrán láthatunk két elfogott rejtjelezett szöveget, c_1-et és c_2-t. Tudjuk, hogy mindkettő ugyanannak az m nyílt szövegnek a rejtett változata, és az egyik eltolásos rejtjelezővel, a másik pedig Vigenère-rejtjelezővel lett sifrírozva. Mindkét szöveget fejtsük vissza. *Útmutatás.* A visszafejtés után megállapítható, hogy az egyik rejtjelező módszernél egy igaz, a másiknál egy hamis állítás adódik. Van esetleg értelme gyakorisági elemzést alkalmazni?

c_1	W K L V V H Q W H Q F H L V H Q F U B S W H G E B F D H V D U V N H B
c_2	N U O S J Y A Z E E W R O S V H P X Y G N R J B P W N K S R L F Q E P

3.6. ábra. Ugyanannak a nyílt szövegnek két rejtett változata a 3.1-1. gyakorlatban.

3.1-2. Mutassuk meg, hogy $\mathbb{Z}$ a szokásos összeadással és szorzással nullosztómentes gyűrű. Vajon $\mathbb{Z}$ test? Mit mondhatunk az $(\mathbb{N}, +)$, $(\mathbb{N}, \cdot)$ és $(\mathbb{N}, +, \cdot)$ algebrai struktúrák tulajdonságairól?

3.1-3. Bizonyítsuk be az Euler-féle φ-függvény alábbi tulajdonságait:

a. $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$ minden $m, n \in \mathbb{N}$ esetén, ha $\text{lnko}(m, n) = 1$.

b. $\varphi(p) = p - 1$ minden p prímszám esetén.

Ezeknek a tulajdonságoknak a segítségével bizonyítsuk be a 3.3 állítást.

3.1-4. Adottak az F, G és H gráfok a 3.5. ábrán.

a. Határozzuk meg a G és H közötti összes izomorfizmust.

b. Határozzuk meg F, G és H összes automorfizmusát.

c. A G és H közötti melyik izomorfizmus esetén lesz $\text{Iso}(G, H)$ $\text{Aut}(G)$-nak jobb oldali mellékosztálya $\mathfrak{S}_5$-ben, vagyis melyik $\tau \in \text{Iso}(G, H)$ esetén lesz $\text{Iso}(G, H) = \text{Aut}(G)\tau$? Határozzuk meg $\text{Aut}(F)$, $\text{Aut}(G)$ és $\text{Aut}(H)$ jobb oldali reprezentánsrendszerét $\mathfrak{S}_5$-ben.

d. Határozzuk meg F minden csúcsának pályáját $\text{Aut}(F)$-ben és H minden csúcsának pályáját $\text{Aut}(H)$-ban.

e. Határozzuk meg az $\text{Aut}(F) \leq \mathfrak{S}_5$ és $\text{Aut}(H) \leq \mathfrak{S}_5$ részcsoportok stabilizátorláncát.

f. Hány 5 csúcsú gráf izomorf F-fel?

3.1-5. Valamely G gráf komplementer gráfját a következőképpen definiáljuk: a csúcshalmaza $V(\overline{G}) = V(G)$, az élhalmaza pedig $E(\overline{G}) = \{\{i, j\} \mid i, j \in V(\overline{G}) \text{ és } \{i, j\} \notin E(G)\}$. Mutassuk meg, hogy:

a. $\text{Aut}(G) = \text{Aut}(\overline{G})$.

b. $\text{Iso}(G, H) = \text{Iso}(\overline{G}, \overline{H})$.

c. $\overline{G}$ összefüggő, ha G nem összefüggő.

3.2. Kulcscsere Diffie és Hellman szerint

Ebben az alfejezetben, valamint a következőkben is szükségünk lesz a 3.1.3. pontban látott számelméleti alapfogalmakra. Elsősorban a 3.3. példabeli $\mathbb{Z}_k^*$ multiplikatív csoportra és az Euler-féle φ függvényre gondolunk; a maradékosztálygyűrűk aritmetikáját a fejezet végén fogjuk megvilágítani a 3-1. feladatban.

A 3.7. ábra a kulcscserére vonatkozó Diffie–Hellman-protokollt mutatja. Ez a protokoll az r alapú, p modulusú moduláris exponenciális függvényen alapszik, ahol p prímszám és r primitív gyök modulo p. $r \in \mathbb{Z}_n^*$ ***primitív gyök modulo*** n, ha $r^d \not\equiv 1 \bmod n$ minden d, $1 \leq d < \varphi(n)$ esetén. Egy r modulo n primitív gyök előállítja az egész $\mathbb{Z}_n^*$ csoportot, ami azt jelenti, hogy $\mathbb{Z}_n^* = \{r^i \mid 0 \leq i < \varphi(n)\}$. Emlékeztetünk arra, hogy ha p prímszám, akkor a $\mathbb{Z}_p^*$ multiplikatív csoport rendje $\varphi(p) = p - 1$. $\mathbb{Z}_p^*$-nak pontosan $\varphi(p - 1)$ primitív gyöke van (lásd a 3.2-1. gyakorlatot is).

3.5. példa. Nézzük $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$-et. Mivel $\mathbb{Z}_4^* = \{1, 3\}$, így $\varphi(4) = 2$, tehát a modulo 5 primitív gyökök száma 2, modulo 5 primitív gyökök a 2 és a 3, így mind 2 mind 3 előállítja az egész $\mathbb{Z}_5^*$-ot,

Lépés	Aliz	Erik	Bob
1	Aliz és Bob megállapodnak egy nagy p prímszámban, és egy r modulo p primitív gyökben; p és r nyilvánosak		
2	véletlenszerűen választ egy nagy, titkos a számot, és kiszámítja a következőt: $\alpha = r^a \bmod p$		véletlenszerűen választ egy nagy, titkos b számot, és kiszámítja a következőt: $\beta = r^b \bmod p$
3		$\alpha \Rightarrow$ $\Leftarrow \beta$	
4	kiszámítja $k_A = \beta^a \bmod p$ értékét		kiszámítja $k_B = \alpha^b \bmod p$ értékét

3.7. ábra. Diffie és Hellman szerinti kulcscserére vonatkozó protokoll.

mivel

$$2^0 = 1; \quad 2^1 = 2; \quad 2^2 = 4; \quad 2^3 \equiv 3 \bmod 5 ;$$
$$3^0 = 1; \quad 3^1 = 3; \quad 3^2 \equiv 4 \bmod 5; \quad 3^3 \equiv 2 \bmod 5 .$$

Nem minden n számnak van modulo n primitív gyöke; a 8 a legkisebb ilyen példa. Az elemi számelméletből tudjuk, hogy valamely n számra pontosan akkor van modulo n primitív gyök, ha n vagy $\{1, 2, 4\}$-nek eleme, vagy pedig $n = q^k$, illetve $n = 2q^k$ alakú, ahol q páratlan prímszám.

3.12. definíció (diszkrét logaritmus). *Legyen p prímszám és r egy modulo p primitív gyök. Az r **alapú,** p **modulusú** $\mathtt{exp}_{r,p}$ **moduláris exponenciális függvényt,** amely $\mathbb{Z}_{p-1}$-ről $\mathbb{Z}_p^*$-re képez, $\mathtt{exp}_{r,p}(a) = r^a \bmod p$-vel definiáljuk. Az inverz függvényét **diszkrét logaritmusnak** nevezzük, ez rögzített p és r esetén az $\mathtt{exp}_{r,p}(a)$ értéket a-ra képezi. Fennáll, hogy $a = \log_r \mathtt{exp}_{r,p}(a) \bmod p$.*

A 3.7. ábra protokollja működik, mert $k_A = \beta^a = r^{ba} = r^{ab} = \alpha^b = k_B$ miatt (modulo p aritmetikával) Aliz valójában ugyanazt a kulcsot számolja ki, mint Bob. Ezt a kulcsot könnyen kiszámíthatják, mivel az $\mathtt{exp}_{r,p}$ moduláris exponenciális függvény a MODULÁRIS-HATVÁNYOZÓ algoritmussal hatékonyan meghatározható. Ezzel szemben Erik arra irányuló törekvése, hogy a kulcsukat meghatározza, nehézségekbe ütközik, mert a diszkrét logaritmus igen nehéz problémának számít. Az $\mathtt{exp}_{r,p}$ moduláris exponenciális függvény így egy lehetséges ***egyirányú függvény,*** olyan függvény, amelyik bár könnyen kiszámítható, de csak nehezen invertálható. Sajnos máig nem tudjuk, hogy van-e egyáltalán egyirányú függvény. S ami még inkább kár: jóllehet nem tudjuk, hogy van-e ilyen, a kriptográfiában az egyirányú függvények kulcsszerepet játszanak, mert sok kriptorendszer biztonsága azon a feltevésen alapszik, hogy egyirányú függvények valóban vannak.

MODULÁRIS-HATVÁNYOZÓ(a, b, m)

1 ▷ m a modulus, $b < m$ az alap és a a kitevő
2 állítsuk elő a kitevőt kettes számrendszerben: $a = \sum_{i=0}^{k} a_i 2^i$, ahol $a_i \in \{0, 1\}$.
3 számoljuk ki egymás után a b^{2^i} értékeket, ahol $0 \le i \le k$, felhasználva, hogy $b^{2^{i+1}} = \left(b^{2^i}\right)^2$.
4 számoljuk ki modulo m aritmetikával $b^a = \prod_{\substack{i=0 \\ a_i=1}}^{k} b^{2^i}$ értékét.
5 ▷ ha beleszámoltuk a szorzatba b^{2^i}-t, és kiszámítottuk $b^{2^{i+1}}$-t, akkor b^{2^i}-t kitörölhetjük
6 **return** b^a

A $b^a = \prod_{\substack{i=0 \\ a_i=1}}^{k} b^{2^i}$ kiszámítása helyes, mert a modulo m aritmetikában fennáll a követ-

kező:

$$b^a = b^{\sum_{i=0}^{k} a_i 2^i} = \prod_{i=0}^{k} \left(b^{2^i}\right)^{a_i} = \prod_{\substack{i=0 \\ a_i=1}}^{k} b^{2^i} .$$

Miért lehet hatékonyan kiszámítani az $\mathtt{exp}_{r,p}(a) = r^a \bmod p$ hatványfüggvényt? Ha ezt a számítást naivan hajtjuk végre, sok szorzásra lehet szükségünk az a kitevő mindegyik értékére. A MODULÁRIS-HATVÁNYOZÓ algoritmussal azonban Aliznak nem $a-1$ szorzást kell végeznie, mint a naiv módszernél, hanem csupán legfeljebb $2\log a$ szorzásra van szüksége. A MODULÁRIS-HATVÁNYOZÓ algoritmus egy exponenciális tényezővel gyorsítja a moduláris hatványozást.

3.6. példa. (MODULÁRIS-HATVÁNYOZÓ a Diffie–Hellman-protokollban.) Aliz és Bob a $p = 5$ prímszámot és az $r = 3$-at, mint modulo 5 primitív gyököt választja. Aliz az $a = 17$ titkos számot választja. Mivel Aliz Bobnak az α számot akarja küldeni, ki szeretné számítani az $\alpha = 3^{17} = 129140163 \equiv 3 \bmod 5$ számot. A kitevő bináris alakban felírva $17 = 1+16 = 2^0+2^4$. Aliz lépésenként számítja ki az értéket:

$$3^{2^0} = 3;\ 3^{2^1} = 3^2 \equiv 4 \bmod 5;\ 3^{2^2} \equiv 4^2 \equiv 1 \bmod 5;\ 3^{2^3} \equiv 1^2 \equiv 1 \bmod 5;\ 3^{2^4} \equiv 1^2 \equiv 1 \bmod 5.$$

Ebből kiszámítja a $3^{17} \equiv 3^{2^0} \cdot 3^{2^4} \equiv 3 \cdot 1 \equiv 3 \bmod 5$ értéket. Figyeljük meg, hogy Aliz ahelyett, hogy 16-szor szorzott volna, csupán négyszer emelt négyzetre és egyszer szorzott $\alpha = 3 \bmod 5$ meghatározása érdekében. Ha Bob saját magának a $b = 23$ titkos kitevőt választja, akkor ugyanezzel az eljárással ki tudja számítani a kulcs őrá eső részét, tehát $\beta = 3^{23} = 94143178827 \equiv 2 \bmod 5$, végül Aliz és Bob a közös titkos kulcsukat a 3.7. ábrán látható Diffie–Hellman-protokollal meghatározzák (lásd a 3.2-2. gyakorlatot). A protokoll biztonsága ebben az esetben természetesen nincs garantálva, hiszen $p = 5$-tel nagyon kicsi prímszámot választottak; ez a játékpélda csak a könnyebb megértést szolgálja.

Ha Erik figyelmesen lehallgatja Aliz és Bob társalgását, amely a 3.7. ábra szerinti Diffie–Hellman-protokoll szerint zajlik, akkor ismeri a p, r, α és β értékeket, amelyekből szeretné meghatározni a $k_A = k_B$ titkos kulcsukat. Eriknek ezt a problémáját ***Diffie–Hellman-problémának*** nevezik. Ha Erik meg tudná határozni az $a = \log_r \alpha \bmod p$ és a $b = \log_r \beta \bmod p$ titkos számokat, akkor, akárcsak Aliz és Bob, ki tudná számolni a $k_A = \beta^a \bmod p = \alpha^b \bmod p = k_B$ kulcsot, és megoldaná a Diffie–Hellman-problémát. Ez a probléma tehát nem nehezebb, mint a diszkrét logaritmus problémája. A fordított kérdés, hogy vajon a Diffie–Hellman-probléma ***legalább*** olyan nehéz-e, mint a diszkrét logaritmusé, tehát hogy ugyanolyan nehezek-e, ez máig csupán nem bizonyított sejtés. Mint sok más protokoll, a Diffie–Hellman-protokoll biztonságossága mindezidáig nem bizonyított.

Mivel szerencsére eddig sem a diszkrét logaritmus, sem a Diffie–Hellman-probléma hatékonyan nem oldható meg, ez a közvetlen támadás nem jelent valóságos veszélyt. Másrészt azonban vannak más, nem közvetlen támadások, amelyeknél a kulcsot nem közvetlenül a Diffie–Hellman-protokollban átadott α, illetve β értékekből határozzák meg. Így például a Diffie–Hellman nem biztonságos a „***középen állás támadással***” szemben. A fent leírt támadás ***passzív,*** ez pedig ***aktív*** abban az értelemben, hogy a támadó Erik nem elégszik meg a passzív hallgatózással, hanem megpróbálja a protokollt aktívan a saját ízlésének megfelelően megváltoztatni. Úgyszólván beáll „középre”, Aliz és Bob közé és elfogja Aliz Bobnak küldött $\alpha = r^a \bmod p$, és Bob Aliznak küldött $\beta = r^b \bmod p$ üzenetét. α és β helyett saját $\alpha_E = r^c \bmod p$ értékét továbbítja Bobnak, Aliznak pedig a szintén saját $\beta_E = r^d \bmod p$ értéket, miközben Erik a titkos c és d értékeket maga választotta. Ha most

Lépés	Aliz	Erik	Bob
1			Véletlenszerűen választ két nagy prímszámot, p-t és q-t és kiszámítja az $n = pq$ és $\varphi(n) = (p-1)(q-1)$ értékeket, nyilvános (n, e) kulcsát, és d titkos kulcsát, amelyek megfelelnek (3.8) és (3.9)-nek
2		$\Leftarrow (n, e)$	
3	$c = m^e \bmod n$ szerint rejtjelezi m-et		
4		$c \Rightarrow$	
5			visszafejti c-t $m = c^d \bmod n$ szerint

3.8. ábra. Az RSA-protokoll.

Aliz a $k_A = (\beta_E)^a \bmod p$ értéket kiszámítja, k_A valójában ezentúl nem a Bobbal – mint ahogy azt Aliz hiszi –, hanem az Erikkel való kommunikáció kulcsa, mert Erik ezt a kulcsot ugyanúgy értékeli ki, mint ők (modulo p aritmetikával): $k_E = \alpha^d = r^{ad} = r^{da} = (\beta_E)^a = k_A$ segítségével. Ugyanígy lefolytathat Erik egy észrevehetetlen kulcscserét Bobbal, és a jövőben vele kommunikálhat anélkül, hogy ezt Bob sejtené. A ***hitelességnek*** ezzel a problémájával később fogunk alaposabban foglalkozni a zéró-ismeretű protokollokról szóló 3.5. alfejezetben.

Gyakorlatok

3.2-1. *a.* Hány primitív gyök van $\mathbb{Z}_{13}^*$-ban, illetve $\mathbb{Z}_{14}^*$-ban?

b. Határozzuk meg $\mathbb{Z}_{13}^*$ valamint $\mathbb{Z}_{14}^*$ összes primitív gyökét, és bizonyítsuk be, hogy ezek valóban primitív gyökök.

c. Mutassuk meg az összes modulo 13, illetve modulo 14 primitív gyökről, hogy a teljes $\mathbb{Z}_{13}^*$-ot, illetve a teljes $\mathbb{Z}_{14}^*$-ot előállítják.

3.2-2. *a.* Határozzuk meg a $\beta = 3^{23} = 94143178827 \equiv 2 \bmod 5$ értéket (Bob 3.6. példabeli értékét) a MODULÁRIS-HATVÁNYOZÓ algoritmussal.

b. Határozzuk meg a 3.6. példában az α és β számokhoz Aliz és Bob közös titkos kulcsát a 3.7. ábrán lévő Diffie–Hellman-protokoll szerint.

3.3. RSA és faktorizálás

3.3.1. RSA

Az RSA-kriptorendszer, amelyet három feltalálójáról, Ron Rivest, Adi Shamir és Leonard Adleman-ról neveztek el, az első ismert ***nyilvános kulcsú kriptorendszer***. Manapság is igen népszerű, és sok kriptográfiai alkalmazásba beillesztik. A 3.8. ábrán összefoglaltuk az RSA protokoll egyes lépéseit, amelyeket végül részletesen is leírunk, valamint lásd a 3.7. példát.

Kulcsgenerálás. Bob választ véletlenszerűen két nagy prímszámot, p-t és q-t úgy, hogy $p \neq q$, és kiszámítja $n = pq$ szorzatukat. Azután választ egy $e \in \mathbb{N}$ kitevőt az alábbiak szerint

$$1 < e < \varphi(n) = (p-1)(q-1) \quad \text{és} \quad \text{lnko}(e, \varphi(n)) = 1\,, \tag{3.8}$$

majd meghatározza $e \bmod \varphi(n)$ inverzét, vagyis azt az egyértelmű d számot, amelyre:

$$1 < d < \varphi(n) \quad \text{és} \quad e \cdot d \equiv 1 \bmod \varphi(n). \tag{3.9}$$

Az (n, e) pár Bob *nyilvános kulcsa, d* pedig Bob *titkos kulcsa.*

Rejtjelezés. Mint már a 3.1. alfejezetben is, az üzenetek egy Σ ábécé feletti szavak és blokkonként, állandó hosszúságú blokkhosszal $|\Sigma|$-adikus természetes számokként kódoljuk őket. Ezeket a számokat azután rejtjelezzük. Legyen $m < n$ az üzenet egyik rejtjelezett blokkjának megfelelő szám, amelyet Aliz szeretne Bobnak küldeni. Aliz ismeri Bob nyilvános (n, e) kulcsát, és m-et rejtjelezi $c = E_{(n,e)}(m)$ számként, ahol a rejtjelező függvény definíciója az alábbi:

$$E_{(n,e)}(m) = m^e \bmod n \ .$$

Visszafejtés. Legyen c, $0 \leq c < n$ az a szám, amely a rejtjelezett szöveg egyik blokkjának kódja, amelyet Bob megkap, Erik pedig lehallgatja. Bob desifrírozza c-t a d titkos kulcsa és az alábbi visszafejtő függvény segítségével:

$$D_d(c) = c^d \bmod n \ .$$

Azt, hogy az RSA-módszer valóban kriptorendszer a 3.1. definíció értelmében, a 3.13. tétel állítja. Ennek a tételnek a bizonyítását az Olvasóra hagyjuk (lásd a 3.3-1. gyakorlatot).

3.13. tétel. *Legyenek (n, e) a nyilvános, d a titkos kulcsok az RSA-protokollban. Ekkor minden m, $0 \leq m < n$ üzenet esetén $m = (m^e)^d \bmod n$. Így az RSA (nyilvános kulcsú) kriptorendszer.*

Az RSA-eljárás hatékonysága érdekében megint a MODULÁRIS-HATVÁNYOZÓ algoritmust használjuk a gyors-hatványozásra. Hogyan válasszuk meg a 3.8. ábrán bemutatott RSA protokollban a p és q prímszámokat? Először is elég nagyoknak kell lenniük, egyébként Erik az n számot Bob nyilvános (n, e) kulcsában faktorizálná, és n p és q prímfaktorait meghatározná, így a kiterjesztett euklideszi algoritmussal könnyen meghatározhatná Bob d titkos kulcsát, amely e mod $\varphi(n)$ egyértelmű inverze, ahol $\varphi(n) = (p-1)(q-1)$. A p és q prímszámoknak titkosaknak kell maradniuk, s ezért elegendően nagyoknak kell lenniük. A gyakorlatban p-t és q-t mindig legalább 80 decimális jegyűnek kell választani. Elő kell tehát állítani ilyen nagyságú véletlen számokat, és tesztelni kell egy ismert véletlen prímteszttel, hogy vajon valóban prímek-e. Mivel a prímszámok elmélete szerint körülbelül $N/\ln N$ N-nél kisebb prímszám van, nagy a valószínűsége annak, hogy nem túl sok próbálkozás után prímszámra akadjunk.

Elméletileg *determinisztikusan* polinomiális időben eldönthető, hogy p és q prímek-e. Agrawal, Kayal és Saxena a közelmúltban azt a meglepő eredményt publikálták, hogy a PRIMES $= \{\text{bin}(n) \mid n \text{ prím}\}$ prímszámprobléma a P bonyolultsági osztályba tartozik. Ez az áttörés megoldotta a bonyolultságelmélet régóta nyitott problémáját, mert a gráfizomorfizmus probléma mellett a prímszámprobléma tűnt olyannak, amely se nem P-beli, se nem NP-teljes.[3] Bár az algoritmus futási idejének eredeti – $O(n^{12})$ – korlátját azóta $O(n^6)$-ra javították, az algoritmus a gyakorlati alkalmazások számára nem elég gyors.

A legkedveltebb véletlen prímteszt Rabin következő algoritmusa, amely Miller determinisztikus algoritmusának elvére épül.

[3] A P és NP bonyolultsági osztályokat a 4.1. alfejezetben definiáljuk, az NP-teljességet pedig a 4.2. alfejezetben.

MILLER–RABIN(n)

végezzük el az $n-1=2^k m$ átalakítást, ahol m és n páratlanok
válasszunk véletlenszerűen egy $z \in \{1,2,\ldots,n-1\}$ számot egyenletes eloszlással
számítsuk ki az $x = z^m \bmod n$ értéket
if $x \equiv 1 \bmod n$
    then return „n prímszám” és álljunk meg
    else for $j \leftarrow 0$ **to** $k-1$
        if $x \equiv -1 \bmod n$
            then return „n prímszám”
            else $x \leftarrow x^2 \bmod n$;
                return „n nem prímszám”

A MILLER–RABIN teszt úgynevezett ***Monte-Carlo-algoritmus,*** „nem” válasza megbízható, „igen” válasza azonban egy bizonyos hibavalószínűséggel értendő. A Miller–Rabin-teszthez hasonló Solovay és Strassen prímtesztje. Mindkettő $O(n^3)$ időben dolgozik. A Solovay–Strassen-teszt azonban kevésbé népszerű, mert a gyakorlatban nem olyan hatékony, mint a Miller–Rabin-teszt és kevésbé pontos.

Azon problémák osztályának, amelyek a Monte-Carlo-algoritmussal mindig megbízható „igen” válasszal oldódnak meg, külön nevük van: RP, a ***R***andomized ***P***olynomial Time kifejezésből származó betűszó. A komplementer osztály, coRP $= \{L \mid \overline{L} \in \mathrm{RP}\}$, mindazokat a problémákat tartalmazza, amelyekre a Monte-Carlo-algoritmus mindig megbízható „nem” választ ad. RP-t formálisan a nemdeterminisztikus polinomiális idejű Turing-gépekkel definiáljuk (röviden NPTM; lásd a 4.1. alfejezetet és főként a 4.1., 4.2. és 4.3. definíciókat), amelyeknek a működése véletlen folyamatként fogható fel: minden nemdeterminisztikus elágazásnál a gép úgymond pénzfeldobással működik, és a következő két elágazás mindegyike 1/2 valószínűséggel következik be. Az NPTM-ben megvalósuló utak száma szerint minden bemeneti adathoz tartozik egy bizonyos megvalósulási valószínűség, ahol hibák is felléphetnek. RP definíciója megkívánja, hogy az elfogadott bemeneti értékek esetén a hibavalószínűség soha nem lehet nagyobb 1/2-nél, miközben az elutasított bemenetek esetén egyáltalán nem léphet fel hiba.

3.14. definíció (randomizált polinomiális idő). *Az* RP *osztály pontosan azokat az A problémákat tartalmazza, amelyekhez van olyan NPTM M, hogy M mindegyik $x \in A$-t legalább* 1/2 *valószínűséggel fogadja el, $x \notin A$-t pedig* 0 *valószínűséggel fogadja el.*

3.15. tétel. *A* PRIMES *a* coRP *ben van.*

A fenti tétel azt mondja, hogy a Miller–Rabin-teszt a prímszámproblémára Monte-Carlo-algoritmus. A bizonyítást itt csak vázoljuk. Megmutatjuk, hogy a Miller–Rabin-teszt PRIMES-t egyoldalú hibavalószínűséggel fogadja el: ha a (binárisan előállított) n bemenő érték prímszám, akkor az algoritmus nem válaszolhatja tévesen azt, hogy n nem prímszám. Hogy ellentmondásra jussunk, tegyük fel, hogy n prím, de a Miller–Rabin-teszt azzal a kimenettel áll le, hogy „n nem prímszám”. Következésképpen $z^m \not\equiv 1 \bmod n$ áll fenn. Mivel x-et a **for** ciklus minden lefutásakor négyzetre emeljük, modulo n a z^m, z^{2m}, …, $z^{2^{k-1}m}$ egymás utáni értékeket vizsgáljuk. Ezen értékek egyikénél sem válaszolja azt az algoritmus, hogy n prím. Ebből következik, hogy $z^{2^j m} \not\equiv -1 \bmod n$ minden j, $0 \le j \le k-1$ esetén. Mivel

Üzenet	R S	A I	S T	H E	K E	Y T	O P	U B	L I	C K	E Y	C R	Y P	T O	G R	A P	H Y
m_b	460	8	487	190	264	643	379	521	294	62	128	69	641	508	173	15	206
c_b	697	387	206	449	165	724	631	365	189	600	325	262	332	472	354	665	673

3.9. ábra. Példa RSA-val történő rejtjelezésre.

$n-1=2^k m$, a kis-Fermat-tételből $z^{2^k m}\equiv 1 \bmod n$ következik (lásd a 3.5. következményt). Ezért $z^{2^{k-1}m}$ 1 négyzetgyöke modulo n. Mivel n prím, 1-nek modulo n csak két négyzetgyöke van, nevezetesen a $\pm 1 \bmod n$, (lásd a 3.3-2. gyakorlatot). $z^{2^{k-1}m}\not\equiv -1 \bmod n$ miatt tehát $z^{2^{k-1}m}\equiv 1 \bmod n$ teljesül. De ekkor megint $z^{2^{k-2}m}$ 1-nek négyzetgyöke modulo n. Mint fent, ebből megint $z^{2^{k-2}m}\equiv 1 \bmod n$ következik. Ezt a érvet ismételten alkalmazva végül azt kapjuk, hogy $z^m\equiv 1 \bmod n$, ami ellentmondás. Következésképpen a Miller–Rabin-teszt minden prímszám esetén helyes választ ad. Ha n nem prímszám, akkor megmutatható, hogy a Miller–Rabin-teszt valószínűsége nem lépi túl az $1/4$ küszöböt. Ismételt független tesztfuttatásokkal azt kapjuk, – természetesen a futási idő kárára, ami mindamellett $\log n$-ben polinomiális marad – hogy a hibavalószínűség tetszőlegesen közel kerül nullához.

3.7. példa. *RSA.* Bob a $p=67$ és $q=11$ prímszámokat választja. Így $n=67\cdot 11=737$ és $\varphi(n)=(p-1)(q-1)=66\cdot 10=660$. Ha most Bob a $\varphi(n)=660$-hoz a lehető legkisebb kitevőt választja, ami $e=7$, akkor az ő nyilvános kulcsa az $(n,e)=(737,7)$ pár. A kiterjesztett euklideszi algoritmus Bobnak a $d=283$ titkos kulcsot szolgáltatja, és fennáll $e\cdot d=7\cdot 283=1981\equiv 1 \bmod 660$ (lásd a 3.3-3. gyakorlatot). Amint a 3.1. alfejezetben is tettük, a $\Sigma=\{\mathrm{A},\mathrm{B},\ldots,\mathrm{Z}\}$ ábécét a $\mathbb{Z}_{26}=\{0,1,\ldots,25\}$ halmazzal azonosítjuk. Az üzenetek a Σ fölötti szavak, és azonos hosszúságú blokkonként 26-os számrendszerben előállított számokként kódoljuk. Példánkban a blokkhossz $\ell=\lfloor \log_{26} n\rfloor=\lfloor \log_{26} 737\rfloor=2$.

Egy $b=b_1b_2\cdots b_\ell$ blokkot, amelynek a hossza ℓ, és $b_i\in\mathbb{Z}_{26}$, az $m_b=\sum_{i=1}^{\ell} b_i\cdot 26^{\ell-i}$ szám ábrázolja. Az $\ell=\lfloor \log_{26} n\rfloor$ blokkhossz definíció következtében:

$$0\le m_b\le 25\cdot\sum_{i=1}^{\ell}26^{\ell-i}=26^\ell-1<n.$$

Az RSA rejtjelező függvénnyel a b blokkot, illetve a megfelelő m_b számot $c_b=(m_b)^e \bmod n$ szerint sifrírozzuk. A b blokknak megfelelő rejtett szöveg ekkor $c_b=c_0c_1\cdots c_\ell$, ahol $c_i\in\mathbb{Z}_{26}$. Az RSA tehát az ℓ hosszúságú blokkokat injektíven képezi le $\ell+1$ hosszúságú blokkokra. A 3.9. ábrán láthatjuk, hogy egy 34 hosszú üzenetet 17 darab 2 hosszú blokkba tördeltünk és az egyes blokkokat számokként rejtjeleztük. Például az első blokkot, amely „RS”, számmá transzformáltuk a következőképpen: „R”-nek a 17 és „S”-nek a 18 felel meg, és $17\cdot 26^1+18\cdot 26^0=442+18=460$.

A kapott c_b számot megint előállíthatjuk 26-os számrendszerben és $\ell+1$ lehet a hossz: $c_b=\sum_{i=0}^{\ell} c_i\cdot 26^{\ell-i}$, ahol $c_i\in\mathbb{Z}_{26}$ (lásd a 3.3-3. gyakorlatot is). Így az első blokk $697=676+21=1\cdot 26^2+0\cdot 26^1+21\cdot 26^0$, a rejtjelezett szövegben „BAV” lett belőle.

A visszafejtés is blokkonként történik. Ahhoz, hogy az első blokkot a $d=283$ titkos kulccsal visszafejtsük, kiszámoljuk a következőt: $697^{283} \bmod 737$, ismét gyors-hatványozással, a Moduláris-hatványozó algoritmus segítségével. Azért, hogy a számok ne legyenek nagyon nagyok, ajánlatos minden szorzásnál modulo $n=737$ redukálni a kapott értékeket. A kitevő bináris kifejtése $283=2^0+2^1+2^3+2^4+2^8$, és – amint kívántuk – adódik a következő:

$$697^{283}\equiv 697^{2^0}\cdot 697^{2^1}\cdot 697^{2^3}\cdot 697^{2^4}\cdot 697^{2^8}\equiv 697\cdot 126\cdot 9\cdot 81\cdot 15\equiv 460 \bmod 737.$$

Lépés	Aliz	Erik	Bob
1	A következő értékeket választja: $n = pq$, (n, e) nyilvános kulcs és d titkos kulcs, ahogyan Bob tette a 3.8. ábrán lévő RSA-protokollban		
2		$(n, e) \Rightarrow$	
3	Aláírja az m üzenetet: $\mathrm{sig}_A(m) = m^d \bmod n$		
4		$(m, \mathrm{sig}_A(m)) \Rightarrow$	
5			Ellenőrzi Aliz aláírását $m \equiv (\mathrm{sig}_A(m))^e \bmod n$ segítségével

3.10. ábra. Digitális aláírás RSA-val.

3.3.2. Digitális aláírás RSA segítségével

A 3.8. ábrán szereplő RSA nyilvános kulcsú kriptorendszer módosítható úgy, hogy protokollként digitális aláírásra alkalmas legyen. Ilyet mutattunk be a 3.10. ábrában. Meggyőződhetünk arról, hogy a protokoll működik (lásd a 3.3-4. gyakorlatot). Ez a protokoll érzékeny az olyan támadásra, amelynél a támadó maga meg tudja választani a rejtjelezendő nyílt szöveget (***választott szövegű támadás***, „chosen-plaintext attacks").

3.3.3. Az RSA biztonsága és lehetséges támadások az RSA ellen

Már említettük, hogy az RSA biztonsága döntő mértékben attól függ, hogy nagy számok nem könnyen faktorizálhatók. Mivel elszánt keresés ellenére sem sikerült ezidáig hatékony faktorizáló algoritmust találni, azt gyanítjuk, hogy ilyen algoritmus nincs, vagyis a faktorizálási probléma nehéz. Erre a sejtésre azonban még nem született bizonyítás. Önmagában ennek a sejtésnek a bizonyításából nem következne, hogy az RSA-rendszer biztonságos. Tudjuk, hogy az RSA feltörése legfeljebb olyan nehéz, mint a faktorizálási probléma, azt azonban nem tudjuk, hogy ugyanolyan nehéz-e. Elképzelhető az is, hogy az RSA könnyen feltörhető n faktorizálása nélkül is.

Az RSA-rendszer elleni lehetséges támadások listája helyett, amely különben is hiányos lenne, utalunk a további hivatkozásokat tartalmazó idevágó irodalomra, valamint a 3-4. feladatra. Mindegyik eddig ismert RSA elleni támadásra van megfelelő ellenintézkedés, amelyek meghiúsítják, vagy hatástalanítják a támadást, tehát a támadás sikerességének a valószínűsége elhanyagolhatóan kicsivé tehető. Főként a p és q prímszámokat, az n modulust, az e kitevőt és a d titkos kulcsot kell megfelelő gondossággal megválasztani.

Mivel az RSA elleni ***faktorizálási támadás*** különösen központi szerepet játszik, bemutatunk egy egyszerű ilyen támadást, amely John Pollard $(p-1)$-módszerén alapszik. A $(p-1)$-módszer p prímfaktorú összetett n szám esetén akkor működik, ha $p-1$ prímfaktorai kicsik. Ekkor ugyanis meg lehet határozni $p-1$ egyik ν többszörösét anélkül, hogy p-t ismernénk, és alkalmazhatjuk a kis Fermat-tételt (lásd a 3.5. következményt), amely szerint $a^\nu \equiv 1 \bmod p$ minden a egész szám esetén fennáll, amely p-hez relatív prím. Következésképpen p egy osztója $a^\nu - 1$-nek. Ha n nem osztója $a^\nu - 1$-nek, akkor $\mathrm{lnko}(a^\nu - 1, n)$ valódi osztója n-nek, s így n-et faktorizáltuk.

Hogyan lehet meghatározni $p-1$ egy ν többszörösét? Pollard $(p-1)$-eljárása ν számára azt a számot alkalmazza jelöltként, amely egy választott S korlát alatti összes prímszámhat-

Lépés	Aliz	Erik	Bob
1	véletlenszerűen választ két nagy számot, x-et és y-t, x-et titokban tartja, és kiszámolja az $x\sigma y$ értékét		
2		$(y, x\sigma y) \Rightarrow$	
3			véletlenszerűen választ egy z nagy számot, z-t titokban tartja, és kiszámolja az $y\sigma z$ értéket
4		$\Leftarrow y\sigma z$	
5	kiszámolja a $k_A = x\sigma(y\sigma z)$ értéket		kiszámolja a $k_B = (x\sigma y)\sigma z$ értéket

3.11. ábra. A kulcscserére vonatkozó, σ-n alapuló Rivest–Sherman-protokoll.

ványok szorzataként áll elő:

$$\nu = \prod_{q \text{ prím}, q^k \le S < q^{k+1}} q^k .$$

Ha $p-1$ mindegyik prímhatvány osztója kisebb S-nél, akkor ν többszöröse $p-1$-nek. Az algoritmus kiszámítja $\text{lnko}(a^\nu - 1, n)$ értékét egy alkalmas a alappal. Ha eközben nem sikerül n egyetlen valódi osztóját sem meghatározni, akkor az algoritmus újra indul egy új $S' > S$ értékkel.

Más faktorizálási módszer többek között a ***négyzetes szita***. Ez a következő egyszerű ötleten alapul. Az n szám faktorizálásához egy bizonyos szitával olyan a és b számokat keresünk, amelyekre:

$$a^2 \equiv b^2 \bmod n \quad \text{és} \quad a \not\equiv \pm b \bmod n \; . \tag{3.10}$$

Következésképpen n osztja az $a^2 - b^2 = (a-b)(a+b)$ számot, de nem osztja sem $a-b$-t, sem $a+b$-t. Így $\text{lnko}(a-b, n)$ egy nem triviális tényezője n-nek. A négyzetes szita mellett vannak más szitamódszerek is, amelyek ettől abban a módszerben különböznek, ahogyan a (3.10)-et kielégítő a és b számokat meghatározzák. Egy példa erre az „általános számtest-szita”.

Gyakorlatok

3.3-1.⋆ Bizonyítsuk be a 3.13. tételt. *Útmutatás.* A kis Fermat-tétel 3.5. következményére támaszkodva mutassuk meg, hogy $(m^e)^d \equiv m \bmod p$ és $(m^e)^d \equiv m \bmod q$. Mivel p és q prímszámok, $p \neq q$ és $n = pq$, a kínai maradéktételből következik az $(m^e)^d \equiv m \bmod n$ állítás.

3.3-2.⋆ A 3.15. tétel bizonyításának vázlatában felhasználtuk azt, hogy egy n prímszám esetén 1-nek csak két négyzetgyöke van modulo n, nevezetesen a $\pm 1 \bmod n$. Bizonyítsuk be ezt. *Útmutatás.* Használjuk fel azt, hogy r pontosan akkor négyzetgyöke 1-nek modulo n, ha n az $(r-1)(r+1)$ számot osztja.

3.3-3. *a.* Mutassuk meg, hogy a 3.7. példában szereplő $\varphi(n) = 660$ és $e = 7$ értékekre a kiterjesztett euklideszi algoritmus segítségével valóban a $d = 283$ titkos kulcs adódik, amely 7 mod 660 inverze.

b. Kódoljuk a 3.7. példában a 3.9. ábrabeli nyílt szöveget a $\Sigma = \{\text{A}, \text{B}, \ldots, \text{Z}\}$ ábécé betűivel mind a 17 blokk esetén.

c. Fejtsük vissza a 3.9. ábrán látható rejtett szövegnek mind a 17 blokkját, és mutassuk meg, hogy az eredeti nyílt szöveget kapjuk vissza.

3.3-4. Mutassuk meg, hogy a 3.10. ábrában szereplő digitális aláírásra vonatkozó RSA protokoll működik.

3.4. Rivest, Rabi és Sherman protokolljai

Rivest, Rabi és Sherman javasoltak kulcscserére és digitális aláírásra protokollokat. A 3.11. ábra kulcscsere protokollja Rivest-től és Sherman tól ered. Könnyen módosítható úgy, hogy digitális aláírásra alkalmas legyen (lásd 3.4-1. gyakorlatot).

Rivest és Sherman protokollja egy ***totális, erősen neminvertálható, asszociatív egyirányú függvényen*** alapszik. Ezen a következőt értjük. Egy totális (tehát mindenhol definiált) σ függvény, amelyik $\mathbb{N} \times \mathbb{N}$-ből $\mathbb{N}$-be képez, pontosan akkor ***asszociatív***, ha $(x\sigma y)\sigma z = x\sigma(y\sigma z)$ minden $x, y, z \in \mathbb{N}$ esetén teljesül, ahol a $\sigma(x, y)$ jelölés helyett az $x\sigma y$ jelölést alkalmaztuk. Ebből a tulajdonságból következik, hogy a protokoll működik, mert $k_A = x\sigma(y\sigma z) = (x\sigma y)\sigma z = k_B$ miatt Aliz és Bob valójában ugyanazt a kulcsot számítják ki.

Az erős neminvertálhatóságot itt nem kívánjuk formálisan definiálni. Tájékoztatásul annyit mondunk, hogy σ-t ***erősen neminvertálhatónak*** nevezzük, ha σ nem csupán egyirányú függvény, hanem akkor sem invertálható hatékonyan, ha a függvényérték mellett ehhez a függvényértékhez tartozó két argumentum egyikét is ismerjük. Ez a tulajdonság megakadályozza azt, hogy Erik y és $x\sigma y$, illetve $y\sigma z$ ismeretében a titkos x, illetve z számokat ki tudja számolni, amelynek a segítségével a $k_A = k_B$ kulcsot könnyen meg tudná határozni.

Gyakorlatok

3.4-1. Módosítsuk Rivest és Sherman 3.11. ábrán látható, kulcscserére vonatkozó protokollját úgy, hogy digitális aláírásra alkalmas protokollt kapjunk.

3.4-2. Melyik közvetlen támadás lenne a 3.11. ábrán szereplő Rivest–Sherman-protokoll ellen hatásos, és hogyan akadályozhatnánk meg? *Útmutatás.* Érveljünk a σ asszociatív egyirányú függvény „erős neminvertálhatóságának” fogalmával, amelyen a protokoll alapszik.

3.4-3. *a.* Adjunk formális definíciót az „erős neminvertálhatóság” fogalmára.

b. Adjunk formális definíciót az „asszociativitás” fogalmára ***parciális,*** $\mathbb{N} \times \mathbb{N}$-ből $\mathbb{N}$-be képező függvények számára. Egy ilyen definíciónak nem kell mindenhol érvényesnek lennie. Mi okoz gondot a következő definíció kísérletnél: „Egy (esetleg parciális) $\sigma : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ függvényt ***asszociatívnak*** nevezzük, ha $x\sigma(y\sigma z) = (x\sigma y)\sigma z$ minden olyan $x, y, z \in \mathbb{N}$ esetén érvényben van, amikor az (x, y), (y, z), $(x, y\sigma z)$ és $(x\sigma y, z)$ párok mindnyájan σ értelmezési tartományában vannak.”

Útmutatás. Ezeknek a fogalmaknak kielégítő definícióját találjuk Hemaspaandra munkájában.

3.5. Interaktív bizonyítási rendszerek és zéró ismeret

3.5.1. Interaktív bizonyítási rendszerek, Artúr–Merlin-játékok és zéróismeretű protokollok

Az a probléma, amelyet a Diffie–Hellman-protokoll elleni ***„középen állás támadásnál”*** a 3.2. alfejezetben említettünk, nyilvánvalóan abból adódik, hogy Bob a protokoll lejátszása előtt nem győződött meg beszélgető partnere valódi kilétéről. Bob azt hiszi, hogy Alizzal hajtja végre a protokollt, a valóságban azonban Erikkel. Másképpen megfogalmazva Aliz-

nak az a feladata, hogy Bobot kétséget kizáróan meggyőzze személyazonosságáról. A kriptográfiának ezt a feladatát ***személyazonosításnak*** nevezzük. A digitális aláírással szemben, amely az elektronikus úton közvetített ***dokumentum,*** mint például az e-mail valódiságát hitelesíti, most azoknak a ***személyeknek*** a hitelesítéséről van szó, akik a protokollban a résztvevő felek. Ennek a fogalomnak további értelmezése is van: „személy", vagy „résztvevő" nem csak egy élő személy lehet, hanem például egy számítógép, amelyik egy másik számítógéppel automatikusan lefolytat egy protokollt.

Önmaga hitelességének bizonyítására Aliz felhasználhatna egy csupán általa ismert titkos adatot, mondjuk a PIN kódját (***P***ersonal ***I***dentifaction ***N***umber), vagy más információt, amit rajta kívül más nem ismer. Itt azonban van egy bökkenő. Aliz kénytelen titkát Bobnak elárulni személyazonossága valódiságának bizonyítására. Ekkor azonban az nem lesz többé már titok. Bob egy harmadik féllel lefolytatott protokoll során közölhetné mondjuk Kristóffal, hogy ő Aliz, mert már ismeri Aliz titkát. Az a kérdés tehát, hogyan bizonyíthatná az ember egy titok ismeretét anélkül, hogy magát a titkot elárulná. Pontosan erről van szó a zéró-ismeretű protokollokban. Ezek speciális interaktív bizonyítási rendszerek, amelyeket Goldwasser, Micali és Rackoff vezettek be. Tőlük függetlenül fejlesztette ki Babai és Moran az ezzel lényegében ekvivalens Artúr–Merlin-játékok tervét, amelyet most informálisan leírunk.

A hatalmas Merlin varázsló, akit egy M NP gép reprezentál, és a bizalmatlan Artúr király, akit egy A randomizált polinomidejű gép reprezentál, közösen meg akarnak oldani egy L problémát, tehát el akarják dönteni, hogy egy x bemenet L-hez tartozik-e, vagy sem. Mindegyik bemenő adatért játszanak, miközben felváltva lépnek. Merlin szándéka mindig az, hogy Artúrt meggyőzze arról, hogy a közös x bemenő szavuk L-hez tartozik, függetlenül attól, hogy ez valóban így van-e. Merlin egy lépése az „$x \in L$" (állítólagos) bizonyítására vonatkozó adat. Ezt megkapja egy $M(x, y)$ szimulációval, ahol x a bemenetet és y az eddigi lépéseket kódolja. Az y szó tehát leírja az M eddigi nemdeterminisztikus választásait, illetve A eddigi véletlen választásait.

Artúr király azonban bizalmatlan. A hatalmas varázsló állítólagos bizonyítását természetesen nem tudja közvetlenül ellenőrizni, ehhez nincs meg a számítási ereje. Kétségbe tudja azonban vonni Merlin bizonyítását, és képes egy ügyes kihívással válaszolni, amelyben a Merlin szolgáltatta bizonyítás ***véletlenszerűen választott*** részleteihez ellenőrizhető tanúsítványt követel. Hogy Artúrt kielégítse, Merlinnek lenyűgöző valószínűséggel meg kell őt győznie bizonyításának helyességéről. Artúr egyik lépése tehát abból áll, hogy $A(x, y)$ számításokat szimuláljon, ahol x megint a bemenetet, y pedig a játék eddigi lefutását írják le.

Az Artúr–Merlin-játék gondolata kifejezhető váltakozva egzisztenciális és valószínűségi kvantorokkal, ahol az előbbiek Merlin NP-számítását, az utóbbiak pedig Artúr randomizált polinomiális számításait formalizálják.[4] Ilyen módon bonyolultsági osztályok hierarchiáját definiálhatjuk, az úgynevezett Artúr–Merlin-hierarchiát. Itt az MA osztály definíciójára szorítkozunk, ez egy olyan két lépéses Artúr–Merlin-játéknak felel meg, amelynél Merlin lép először.

3.16. definíció (MA az Artúr–Merlin-hierarchiában). *Az* MA *osztály pontosan azokat az L problémákat tartalmazza, amelyekre van egy M NPTM és egy A randomizált polinomidejű*

[4] Ez hasonlít a polinomiálisidő-hierarchia lépéseinek $\exists$ és $\forall$ kvantorokkal váltakozva történő jellemzéséhez, (lásd a 4.4. alfejezetet és főként a 4.16. tétel 3. részét).

Turing-gép, hogy minden x bemenetre fennáll:

- *Ha $x \in L$, akkor van egy olyan y út $M(x)$-ben, hogy $A(x,y) \geq 3/4$ valószínűséggel elfogad, (vagyis Arthur Merlinnek az „$x \in L$”-re vonatkozó y bizonyítását nem tudja megcáfolni, és Merlin nyer).*
- *Ha $x \notin L$, akkor $M(x)$ mindegyik y útját $A(x,y) \geq 3/4$ valószínűséggel elutasítja (vagyis Artúrt Merlin „$x \in L$”-re vonatkozó hamis bizonyítása nem téveszti meg, és nyer).*

Ennek megfelelően lehet definiálni az AM, MAM, AMA, ... *osztályokat (lásd a 3.5-1. gyakorlatot).*

A 3/4 valószínűségi küszöb a 3.16. definícióban, amely szerint Artúr elfogad vagy elutasít, önkényesen lett megválasztva, és eleinte nem tűnik elég nagynak. Valójában lehet növelni a siker valószínűségét, és 1-hez tetszőlegesen közel hozni. Másként megfogalmazva, a definícióban alkalmazhatunk $1/2 + \varepsilon$ valószínűséget egy tetszőleges rögzített $\varepsilon > 0$ konstanssal, s így még mindig ugyanazt az osztályt kapjuk. Továbbá ismert az, hogy a lépések egy konstans száma esetén ez a hierarchia beleolvad az AM osztályba („stabilizálódik”): $\mathrm{NP} \subseteq \mathrm{MA} \subseteq \mathrm{AM} = \mathrm{AMA} = \mathrm{MAM} = \cdots$. Az a kérdés azonban még nyitott, hogy az $\mathrm{NP} \subseteq \mathrm{MA} \subseteq \mathrm{AM}$ tartalmazás valódi-e.

A fent említett ***interaktív bizonyítási rendszerek*** az Artúr–Merlin-játékok változatai. Jelentéktelen különbség a terminológiában az, hogy Merlin az „igazoló”, Artúr pedig az „ellenőrző”, és a kommunikáció nem játékként zajlik, hanem protokoll formájában. Első pillantásra a két modell között az a jelentős különbség, hogy Artúr véletlen bitje nyilvánosan – és elsősorban Merlin számára – ismert, ezzel ellentétben az ellenőrző véletlen bitje az interaktív bizonyítási rendszerekben titkos. Goldwasser és Sipser megmutatták azonban, hogy valójában lényegtelen az, hogy a véletlen bit titkos, vagy nyilvános. Tehát az Artúr–Merlin-játékok és az interaktív bizonyítási rendszerek egymással ekvivalensek.

Ha a játékban konstans számú lépés helyett polinomiálisan sokat engedünk meg – és többet a polinomiális időkorlátozás miatt nem lehet –, akkor az IP osztályt kapjuk. Definíció szerint IP tartalmazza az egész NP-t, és főként a gráfizomorfizmus-problémát. Később látni fogjuk, hogy IP tartalmaz $\mathrm{coNP} = \{\overline{L} \mid L \in \mathrm{NP}\}$-beli problémákat is, amelyekről feltesszük, hogy nincsenek NP-ben. A 4.21. tétel bizonyítása elsősorban azt mutatja meg, hogy a gráfizomorfizmus-probléma komplementere AM-ben és így IP-ben van. Shamir egyik híres eredménye azt mondja, hogy IP még PSPACE-szel is megegyezik, azon problémák osztályával, amelyek a polinomiális térben eldönthetők.

Térjünk azonban vissza a hitelesség előbb vázolt problémájához, és a zéró-ismeretű protokollok fogalmához. Nézzük az alapgondolatot. Tegyük fel, hogy Artúr és Merlin egyik játékukat játsszák. Az interaktív bizonyítási rendszerek terminológiája szerint ebben az IP-protokollban Merlin nehéz bizonyítást küld Artúrnak. Hogy honnan varázsolta ezt a bizonyítást, az Merlin titka, és mivel csak ő ismeri, ezzel hitelesíteni tudja magát Artúr előtt.

Amit egyikük sem tud: a gonosz varázsló Marvin varázsital segítségével Merlin hasonmásává változott, és Artúr előtt Merlinként akarja kiadni magát. Merlin titkát azonban nem ismeri. Hasonlóképpen nincs meg neki Merlin hatalmas varázslóereje sem, mágiája nem nagyobb, mint egy közönséges randomizált polinomidejű Turing-gép számítási ereje. Ugyanolyan csekély, mint amennyire Artúr képes Marvinnak Merlin-bizonyítást találni. Ennek ellenére megkísérli, hogy a Merlin és Artúr közötti kommunikációt szimulálja. Egy IP-protokoll pontosan akkor ***zéró-ismeretű,*** ha az az információ, amely Marvin és Artúr között kicserélésre kerül, nem különbözik a Merlin és Artúr közötti kommuniká-

ciótól. Ekkor Marvin, aki Merlin titkos bizonyítását nem ismeri, természetesen nem tud róla a szimuláló protokollba semmiféle információt sem áramoltatni. Bár abban a helyzetben van, hogy az eredeti protokollt pontosan lemásolta, úgy, hogy egy független megfigyelő semmiféle különbséget nem tudna felfedezni, a protokoll semmilyen információt nem tud közölni: Ahol nincs, ott ne keress!

3.17. definíció (zéró-ismeretű protokoll). *$L \in$ IP esetén legyen M NPTM és A pedig randomizált polinomidejű Turing-gép úgy, hogy (M, A) interaktív bizonyítási rendszer L számára. Az (M, A) IP-protokoll pontosan akkor* ***zéró-ismeretű protokoll L számára,*** *ha van egy olyan $\widehat{M}$ randomizált polinomidejű Turing-gép, hogy $(\widehat{M}, A)$ az (M, A) eredeti protokollt szimulálja, és minden $x \in L$ esetén az (M, A)-beli, illetve az $(\widehat{M}, A)$-beli kommunikációt reprezentáló $(m_1, m_2, \ldots, m_k)$ és $(\widehat{m}_1, \widehat{m}_2, \ldots, \widehat{m}_k)$ sorozatoknál azonos a pénzfeldobás eloszlása.*

A fent definiált fogalmat az irodalomban úgy nevezik, hogy „***tökéletes zéró ismeret becsületes igazolóval***” (honest-verifier perfect zero-knowledge). Ez a következőket jelenti: (a) feltesszük, hogy az ellenőrző Artúr ***becsületes*** (aminek nem feltétlenül kell teljesülnie a kriptográfiai alkalmazásokban), és (b) megkívánjuk, hogy a szimuláló protokollban közölt információ ***tökéletesen*** megegyezzék az eredeti protokollban közölt információval. Az első feltétel némileg idealista, a második alkalmasint túl szigorú. Ezért vizsgálják a zéró-ismeretű protokollok más változatait (lásd a fejezet végén levő megjegyzéseket).

3.5.2. Zéró-ismeretű protokoll gráfizomorfizmusra

Most egy konkrét példát vizsgálunk. Már említettük, hogy GI NP-ben van, a komplementer probléma pedig $\overline{\text{GI}}$, AM-ben (lásd a 4.21. tétel bizonyítását). Így mindkét probléma IP-ben van. Most adunk egy zéró-ismeretű protokollt GI-re.

Bár manapság nincs hatékony eljárás GI-re, Merlin meg tudja oldani ezt a problémát, mert GI NP-ben van. Jóllehet nem szükséges, hogy ezt tegye. Megteheti, hogy egyszerűen választ egy nagy G_0 gráfot, amelynek n csúcsa van, valamint véletlenszerűen egy $\pi \in \mathfrak{S}_n$ permutációt, és előállítja a $G_1 = \pi(G_0)$ gráfot. A (G_0, G_1) párt nyilvánosságra hozza, a G_0 és G_1 közötti π izomorfizmust saját információjaként titokban tartja. A 3.12. ábra mutatja a Merlin és Artúr közötti IP-protokollt.

Természetesen Merlin nem küldheti el Artúrnak egyszerűen π-t, mert akkor elárulná a titkát. Annak bizonyítására, hogy az adott G_0 és G_1 gráfok valóban izomorfak, Merlin választ véletlenszerűen egyenletes eloszlással egy ρ izomorfizmust és egy a bitet, és előállítja a $H = \rho(G_a)$ gráfot. Ezután elküldi Artúrnak H-t. Ő egy kihívással válaszol: küld Merlinnek egy véletlenszerűen egyenletes eloszlással választott b bitet, és követel tőle egy σ izomorfizmust G_b és H között. Ezt pontosan akkor fogadja el Artúr, ha Merlin σ-jára valóban teljesül $\sigma(G_b) = H$.

A protokoll működik, mert Merlin ismeri saját titkos π izomorfizmusát és véletlenszerűen választott ρ permutációját. Nem okoz problémát Merlinnek, hogy kiszámolja a σ izomorfizmust G_b és H között és Artúr előtt hitelesítse magát. A π titkot azonban nem árulja el. Mivel G_0 és G_1 izomorfak, Artúr 1 valószínűséggel elfogad. Itt egyáltalán nem kell vizsgálni két nem izomorf gráf esetét, mivel Merlin a protokollnak megfelelően izomorf G_0 és G_1 gráfokat választ (lásd a 4.21. tétel bizonyítását is).

Tegyük fel, hogy Marvin szeretné magát Artúr előtt Merlinként kiadni. Ismeri a G_0 és

Lépés	Merlin		Artúr
1	választ egy ρ véletlen permutációt $V(G_0)$-ra és egy $a \in \{0,1\}$ bitet, kiszámolja $H = \rho(G_a)$-t		
2		$H \Rightarrow$	
3			választ egy $b \in \{0,1\}$ véletlen bitet, majd G_b és H között követel egy izomorfizmust
4		$\Leftarrow b$	
5	kiszámolja a σ izomorfizmust $\sigma(G_b) = H$-val: ha $b = a$, akkor $\sigma = \rho$; ha $0 = b \neq a = 1$, akkor $\sigma = \pi\rho$; ha $1 = b \neq a = 0$, akkor $\sigma = \pi^{-1}\rho$.		
6		$\sigma \Rightarrow$	
7			ellenőrzi, hogy $\sigma(G_b) = H$, és akkor fogadja el, ha ez teljesül

3.12. ábra. Goldreich, Micali és Wigderson GI-re vonatkozó zéró-ismeretű protokollja.

Lépés	Marvin		Artúr
1	választ egy ρ véletlen permutációt $V(G_0)$-ra, és egy $a \in \{0,1\}$ bitet, kiszámolja $H = \rho(G_a)$-t		
2		$H \Rightarrow$	
3			választ egy $b \in \{0,1\}$ véletlen bitet, és követel egy izomorfizmust G_b és H között
4		$\Leftarrow b$	
5	ha $b \neq a$, akkor $\widehat{M}$ kitöröl minden ebben a körben eddig közvetített információt és ismétel; ha $b = a$, akkor $\widehat{M}$ elküldi $\sigma = \rho$-t		
6		$\sigma \Rightarrow$	
7			$b = a$ esetén $\sigma(G_b) = H$, ezért Artúr elfogadja Marvin hamis személyazonosságát

3.13. ábra. A GI-re vonatkozó zéró-ismeretű protokoll szimulációja π ismerete nélkül.

G_1 gráfokat, de nem ismeri a titkos π izomorfizmust. Mégis szeretné színlelni, hogy ismeri π-t. Ha az Artúr által választott b bit véletlenül megegyezik az a bittel, amelyet Marvin ***előre*** határozott meg, akkor ő nyer. Ha azonban $b \neq a$, akkor a $\sigma = \pi\rho$ vagy $\sigma = \pi^{-1}\rho$ kiszámítása igényli π ismeretét. Mivel GI egy randomizált polinomidejű Turing-gép számára túl nehéz, és hatékonyan nem számítható ki, Marvin elég nagy G_0 és G_1 gráfok esetén nem tudja meghatározni a π izomorfizmust. Nem ismerve a π-t, csak találgathat. Az ő esélye arra, hogy véletlenül elcsípjen egy b bitet, amelyre $b = a$, legfeljebb $1/2$. Természetesen Marvin mindig találgathat, és így az ő sikervalószínűsége pontosan $1/2$. Ha Artúr azt kívánja, hogy r független körben végezzék el a protokollt, akkor a csalás valószínűsége a 2^{-r} értékre szorítható le. Ez már $r = 20$ esetén is elenyészően csekély: Marvin sikervalószínűsége kisebb, mint egy a millióhoz.

Meg kell még mutatnunk, hogy a 3.12. ábrán szereplő protokoll valóban zéró-ismeretű protokoll. A 3.13. ábrán látható egy szimuláló protokoll Marvinnal, aki Merlin π titkát nem ismeri, de azt színleli, hogy ismeri. Az az információ, amelyik a protokoll egy lefutása alatt kicserélésre kerül, a (H, b, σ) hármasként adható meg. Ha Marvin véletlenszerűen választ egy a bitet, és $a = b$, akkor egyszerűen elküldi $\sigma = \rho$-t és nyer: Artúr, vagy bárki más független megfigyelő nem tud semmilyen szabálytalanságot felfedezni. Másrészt, ha $a \neq b$,

Marvin csalása lelepleződik. Ez azonban nem okoz gondot az álnok varázslónak: egyszerűen kitörli a szimuláló protokollból ezt a kört és újra próbálkozik. Így (H, b, σ) formájú hármasok sorozatát tudja előállítani, amelyek a Merlin és Artúr közötti eredeti protokoll hármasaiból álló megfelelő sorozataitól megkülönböztethetetlenek. Így Goldreich, Micali és Wigderson `GI` protokollja zéró-ismeretű protokoll.

Gyakorlatok

3.5-1. Definiáljuk az Artúr–Merlin-hierarchia AM, MAM, AMA, ... osztályait a 3.16. definíció MA osztályához hasonlóan.

3.5-2. Melyik osztályt kapjuk egy olyan Artúr–Merlin-játékban, amelyik csak egy lépésből áll, amelyet Merlin, illetve amelyet Artúr végez?

3.5-3. Folytassuk le a 3.12. ábrán látható zéró-ismeretű protokollt a $G_0 = G$ és a $G_1 = H$ gráfok, valamint a G és H közötti $\pi = \left(\begin{smallmatrix}1&2&3&4&5\\3&4&1&5&2\end{smallmatrix}\right)$ izomorfizmus esetén, ahol G és H a 3.1.3. pontban a 3.4. példabeli gráfok. Játsszuk le ezt az Artúr–Merlin-játékot egy magunk választotta ρ izomorfizmussal $a \in \{0,1\}$ és $b \in \{0,1\}$ mindegyik változatára. Ismételjük meg ezt a játékot egy számunkra ismeretlen ρ izomorfizmussal, amelyet valaki más választott.

3.5-4. Módosítsuk a 3.12. ábra protokollját úgy, hogy Marvin csalásvalószínűsége a 2^{-10} érték alá kerüljön. Adjuk meg ennek a valószínűségnek egy formálisan helyes elemzését.

Feladatok

3-1. Aritmetika a $\mathbb{Z}_k$ maradékosztálygyűrűben

Legyenek $k \in \mathbb{N}$ és $x, y, z \in \mathbb{Z}$. Azt mondjuk, hogy x ***kongruens*** y***-nal modulo*** k (röviden: $x \equiv y \bmod k$), ha k osztja az $y - x$ különbséget.

Az $x + k\mathbb{Z} = \{y \in \mathbb{Z} \mid y \equiv x \bmod k\}$ halmazt x mod k ***vett maradékosztályának*** nevezzük. Például 3 mod 7 maradékosztálya a $3 + 7\mathbb{Z} = \{3, 3 \pm 7, 3 \pm 2 \cdot 7, \ldots\} = \{3, 10, -4, 17, -11, \ldots\}$ halmaz. ***Egy*** x mod k ***maradékosztály reprezentánsa*** legyen mindig a legkisebb természetes szám $x + k\mathbb{Z}$-ben; például 3 reprezentálja a 3 mod 7 maradékosztályt. Az összes mod k vett maradékosztály reprezentánsa $\mathbb{Z}_k = \{0, 1, \ldots, k-1\}$. $\mathbb{Z}_k$-n definiálunk ***összeadást*** az $(x + k\mathbb{Z}) + (y + k\mathbb{Z}) = (x + y) + k\mathbb{Z}$ és ***szorzást*** az $(x + k\mathbb{Z}) \cdot (y + k\mathbb{Z}) = (x \cdot y) + k\mathbb{Z}$ szabályok segítségével. A modulo 7 vett aritmetikában például $(3+7\mathbb{Z})+(6+7\mathbb{Z}) = (3+6)+7\mathbb{Z} = 2+7\mathbb{Z}$ és $(3 + 7\mathbb{Z}) \cdot (4 + 7\mathbb{Z}) = (3 \cdot 4) + 7\mathbb{Z} = 5 + 7\mathbb{Z}$.

Bizonyítsuk be, hogy modulo k aritmetikát alkalmazva:

a. $(\mathbb{Z}_k, +, \cdot)$ kommutatív egységelemes gyűrű;

b. A 3.3. példában definiált $\mathbb{Z}_k^*$ halmaz multiplikatív csoport;

c. $(\mathbb{Z}_p, +, \cdot)$ minden p prímszám esetén test. Milyen struktúra $(\mathbb{Z}_p^* \cup \{0\}, +, \cdot)$, ha p prímszám?

d. Bizonyítsuk be, hogy egy csoport semleges eleme, valamint minden csoportelem inverze egyértelműen meghatározott.

e. Mutassuk meg, hogy egy kommutatív egységelemes $\mathfrak{R}$ gyűrű invertálható elemei csoportot alkotnak, az $\mathfrak{R}$ úgynevezett ***egységcsoportját***. Mi a $\mathbb{Z}_k$ gyűrű egységcsoportja?

f. Határozzuk meg a $\mathbb{Z}_k$ maradékosztálygyűrű nullosztóit. Mutassuk meg, hogy $\mathbb{Z}_k$-ban nincs nullosztó, ha k prímszám.

3-2. Faizomorfizmus
Speciális gráfosztályok esetén, például a fák osztályában, a GI probléma hatékonyan megoldható. Egy (irányítatlan) ***fa*** összefüggő, körmentes gráf, ahol egy ***kör*** egymásután következő élekből áll, úgy, hogy a kiindulási csúcsba visszatérünk, és minden csúcs legfeljebb egyszer szerepel benne. Egy fa ***levelei*** az 1-fokú csúcsok. Tervezzünk hatékony algoritmust a ***faizomorfizmusra***, amely P-ben van. Ezt a problémát így definiáljuk:

$$\mathtt{TI} \quad = \quad \{(G, H) \mid G \text{ és } H \text{ izomorf fák}\} .$$

Útmutatás. Jelöljük meg fokozatosan az adott fák csúcsait megfelelő számsorozatokkal, és mindegyik lépésben hasonlítsuk össze a kapott jelölő sorozatokat. Kezdjük a levelekkel és lépésről lépésre haladjunk a fa belseje felé, amíg az összes csúcsot meg nem jelöltük.

3-3. Determináns számítása
Írjuk meg egy olyan algoritmus pszeudokódját, amelyik egy mátrix determinánsát hatékonyan kiszámítja. Ültessük át az algoritmust egy tetszőleges programnyelvre. Ki lehet-e hatékonyan számítani egy mátrix inverzét?

3-4. Alacsony kitevőjű támadás

a. A 3.8. ábrán látható RSA-rendszerben az $e = 3$ kitevő hatékonysági okok miatt népszerűségnek örvend. Ez azonban veszélyes lehet. Tegyük fel, hogy Aliz, Bob és Kristóf ugyanazt az m üzenetet ugyanazzal a nyilvános $e = 3$ kitevővel, de esetleg különböző n_A, n_B és n_C modulusokkal rejtjelezik. Erik elcsípi a három keletkező rejtett szöveget: $c_i = m^3 \bmod n_i$, ahol $i \in \{A, B, C\}$. Ezután Erik az m üzenetet könnyen vissza tudja fejteni. Hogyan?

Útmutatás. Erik ismeri a kínai maradéktételt. Egy javasolt érték a kitevő számára $e = 2^{16} + 1$, amelynek bináris előállításában csak két egyes van, s így a MODULÁRIS-HATVÁNYOZÓ algoritmus nagyon gyorsan feldolgozza.

b. A fent leírt támadásokat ki lehet terjeszteni k darab olyan rejtett szövegre, amelyek egymással kapcsolatban vannak. Legyenek ismertek valamilyen a_i és b_i $1 \leq i \leq k$ értékek, és tegyük fel, hogy k darab $c_i = (a_i m + b_i)^e \bmod n_i$ üzenetet küldtek el és mindegyiket el is fogták, ahol $k > e(e+1)/2$ és $\min(n_i) > 2^{e^2}$. Hogyan tudja egy támadó kideríteni, hogy mi az eredeti m üzenet?

Útmutatás. Az úgynevezett rácsredukciós technikával (lásd például Miccianció és Goldwasser cikkét).

c. Hogyan lehet ezt a támadást elhárítani?

Megjegyzések a fejezethez

Diffie és Hellman nyilvános kulcsú kriptográfiáról szóló munkája [111]. Az angol, francia, német és finn nyelvű szövegek betűinek gyakoriságáról [407]-ban olvashatunk. Charles

Babbageról Singh ír a könyvében [433]. Claude Shannon korszakalkotó munkája a tökéletes titkosságú kriptorendszerekről [426]. Ron Rivest, Adi Shamir és Leonard Adleman [391] munkájukban írták le az első ismert nyilvános kulcsú kriptorendszert. Agrawal, Kayal és Saxena[4] munkájukban publikálták azt az eredményt, hogy a PRIMES = {bin(n) | n prím} prímszámprobléma a P bonyolultsági osztályba tartozik.

John Pollard $(p-1)$-módszeréről [375]-ben olvashatunk. Rabin és Miller *véletlen prímtesztje* a [330, 381] cikkekben, Solovay és Strassen prímtesztje pedig a [436] cikkben található meg. ***Választott szövegű támadásról*** olvashatunk a [398] dolgozatban. Az RSA-rendszer elleni lehetséges támadásokról a további utalásokat tartalmazó [55]-ben lehet olvasni. A négyzetes szita leírását megtalálhatjuk például [439]-ben, az „általános számtest-szita" leírását pedig az [288] cikkben.

Stinsonnál [439] olvashatunk arról, hogy ha p és q prímszámok, $p \neq q$ és $n = pq$, a kínai maradéktételből hogyan következik az $(m^e)^d \equiv m \mod n$ állítás. A 3.4-3. gyakorlat megoldásához kielégítő definíciót találunk a [201, 202] munkákban. A zéró-ismeretű protokollokat Goldwasser, Micali és Rackoff [162] vezették be. Az Artúr–Merlin-játékok tervét Babai és Moran [23, 24] fejlesztették ki.

Goldwasser és Sipser [163] munkájában olvashatunk arról, hogy az interaktív bizonyítási rendszerek esetén lényegtelen az, hogy a véletlen bit titkos, vagy nyilvános. A [425]cikkben található meg Shamir híres eredménye arról, hogy IP PSPACE-szel megegyezik. A GI problémára adott zéró-ismeretű protokoll Goldreich, Micali és Wigderson munkájáig [160] nyúlik vissza.

A faizomorfizmusra vonatkozó 3-2. feladathoz lásd [246]-t is. A 3-4. feladatban említett rácsredukciós technikát lásd például [326]-ben. Az itt említett támadás Johan Håstad [186] munkájára nyúlik vissza, és Don Coppersmith [88] javította.

Singh [433] könyve szép bepillantást nyújt a kriptológia fejlődésének történetébe, az ókori gyökerektől kezdve a modern rejtjelező eljárásokig. Például ott olvashatjuk, hogy a brit Állami Kommunikációs Központ (Government Communications Headquarters; GCHQ) egyik különleges egysége, a Kommunikációs-elektronikai Biztonsági Csoport (Communications Electronics Security Group; CESG) azt állítja, hogy Ellis, Cocks és Williamson nevű munkatársai mind a 3.8. ábrán látható RSA-rendszert, mind a 3.7. ábrabeli Diffie–Hellman-protokollt hamarabb feltalálták, mint Rivest, Shamir és Adleman, illetve hamarabb, mint Diffie és Hellman, érdekes módon fordított sorrendben. Az RSA-rendszer jószerével minden, a kriptográfiáról szóló könyvben le van írva. Az RSA elleni támadásokról, mint amilyen a 3.3. alfejezetben szerepel, átfogó listát találunk például a [55, 398] áttekintő cikkekben.

Prímszámtesztek, mint amilyen a Miller–Rabin-algoritmus, valamint faktorizáló algoritmusok szintén sok könyvben megtalálhatók, például [159, 407, 439] -ben.

Az erősen neminvertálható, asszociatív egyirányú függvények fogalma, amelyen az 3.11. ábrán látható titkos kulcscserére vonatkozó protokoll alapszik, Rivest-től és Shermantól származik. Ennek a protokollnak a módosítása, amely digitális aláírásra alkalmas, Rabinak és Sherman-nak köszönhető, akik [380] munkájukban azt is bebizonyították, hogy kommutatív, asszociatív egyirányú függvény pontosan akkor létezik, ha P $\neq$ NP. Kétségtelen, hogy az általuk konstruált egyirányú függvények sem nem totálisak, sem erősen nem invertálhatók, még P $\neq$ NP fennállása esetén sem. Hemaspaandra és Rothe [202] megmutatták, hogy totális, erősen nem invertálható, kommutatív, asszociatív egyirányú függvény pontosan akkor adódik, ha P $\neq$ NP teljesül (lásd a [46, 201] cikkeket is).

Az interaktív bizonyítási rendszerek és zéró-ismeretű protokollok területére vonatkozólag a legjobb és legátfogóbb forrást Goldwasser, Micali és Rackoff mutatták be [162] munkájukban, ez Goldreich könyvében [159] a 4. fejezet. Hasonlóképpen szép kimutatást találhatunk például Köbler és társai [247] és Papadimitriou [358] könyvében, valamint a [158, 161, 398] áttekintő cikkekben. Az Artúr–Merlin-játékokat főként Babai és Moran [23, 24], valamint Zachos és Heller [505] vizsgálták.

[159] részletesen foglalkozik a zéró-ismeretű protokollok olyan változataival, amelyek technikai részletekben térnek el a 3.17. definíciótól. Némi ízelítő található róluk például a [158, 161, 398] dolgozatokban.

A kriptográfiával foglalkozik magyar nyelvű könyvében Ködmön József [249], Buttyán Levente és Vajda István [64], valamint Simon Singh [433].

4. Bonyolultságelmélet

A 3. fejezetben kriptográfia módszerek és protokollok szempontjából fontos algoritmusokkal ismerkedtünk meg, mint amilyen a square-and-multiply algoritmus. Az algoritmusok fejlesztői akkor boldogok, ha sikerül egy probléma hatékony megoldását előállítaniuk. Ennek során viszont sok fontos probléma sajnos megmakacsolja magát, és konok módon ellenáll minden kísérletnek, melynek célja hatékony megoldó algoritmus kifejlesztése lenne. Ilyen problémák például a Boole-kifejezések kielégíthetőség problémája, a párosítás és a gráfizomorfizmus problémái, melyeket a fejezetben részletesen tárgyalunk.

A bonyolultságelmélet egyik legfontosabb feladata ilyen és hasonló problémák *kiszámíthatósági bonyolultság* szerinti osztályozása. Miközben az algoritmusfejlesztő akkor elégedett, ha egy, a problémáját megoldó konkrét algoritmus fejlesztése során – konkrét futási idő mellett – meg tud őrizni egy, a lehetőségekhez mérten legjobb *felső bonyolultsági korlátot*, a bonyolultságelméleti szakember egy lehető legjobb *alsó korlát* meghatározásán fáradozik. Az algoritmusok elmélete és a bonyolultságelmélet ilyen szempontból kiegészítik egymást. Ha a felső és alsó korlát egybeesik, akkor a problémát osztályoztuk.

Annak bizonyítéka, hogy egy probléma nem oldható meg hatékonyan, gyakran „negatív” hatású és egyáltalán nem kívánatos. Van azonban a bizonyításnak egy pozitív aspektusa is: a kriptográfia tárgykörében (lásd a 3. fejezetet) épp a rossz hatékonyság hiányában vagyunk érdekeltek. Annak bizonyítása, hogy egyes problémák csak rossz hatékonysággal oldhatók meg, mint például a faktorizálás probléma vagy a diszkrét logaritmus, titkosított üzenetek átvitelekor a biztonság növekedését jelentik.

A 4.1. alfejezetben lefektetjük a bonyolultságelmélet alapjait, különös tekintettel a P és NP bonyolultsági osztályok definícióira. Nagyon fontos kérdés, hogy ez a két osztály különböző-e vagy sem. Ez évtizedek óta a bonyolultságelmélet és az egész elméleti informatika központi kérdése. A mai napig sem a $P \neq NP$ sejtést, sem pedig P és NP egyenlőségét nem sikerült bizonyítani. A 4.2. alfejezet rövid bevezetőt ad az NP-teljesség elméletébe, amely ezzel a kérdéskörrel különösen intenzíven foglalkozik.

A leghíresebb NP-teljes problémák egyike a SAT, az ítéletlogika kielégíthetőség problémája: *kielégíthető-e* egy adott Boole-formula igazságértékeket behelyettesítve változóiba, vagyis igazzá teheti-e valamely behelyettesítés? A SAT probléma NP-teljessége miatt nagyon valószínűtlen, hogy SAT hatékony (és determinisztikus) megoldó algoritmussal rendelkezzen. A 4.3. alfejezetben bemutatunk SAT-hoz egy determinisztikus és egy valószínűségi algoritmust, melyek mindketten exponenciális időben működnek. Futási idejük a *gyakorlatban fontos* bemenetméretek mellett elviselhető szinten tartható még úgy is, hogy ezek az

algoritmusok *aszimptotikusan rossz hatékonyságúak*, tehát nagyon nagy bemenetekre csillagászati méretű ráfordítást igényelnek.

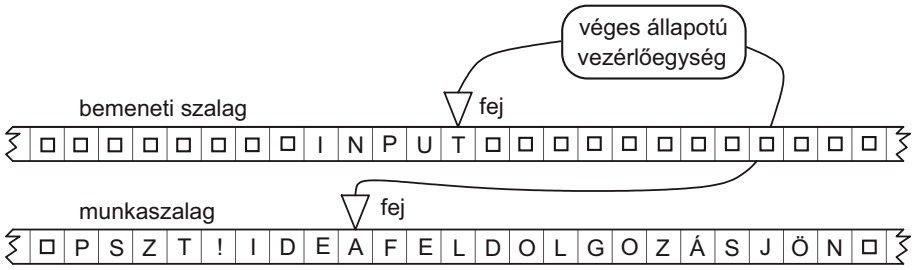
A 4.4. alfejezetben újra elővesszük a GI gráfizomorfizmus problémát, amit a 3.1.3. alfejezet 3.8. definíciójában ismertettünk, és a zéró ismeretű protokolloknál játszott szerepet. Ez a probléma azon NP-beli természetes problémák egyike, melyek valószínűleg (az elfogadhatónak tűnő P $\neq$ NP feltevés mellett) nem oldhatók meg hatékonyan, és nem is NP-teljesek. Ilyen szempontból GI különleges helyzetet élvez az NP-beli problémák halmazában. Az erre utaló jelek az úgynevezett *alsóság*-elméletből (angolul „lowness theory") erednek, amit a 4.4. alfejezetben ismertetünk. Többek között megmutatjuk, hogy GI az NP-n belüli alsó-hierarchiában helyezkedik el, ami nagyban valószínűsíti, hogy GI nem NP-teljes. Ezen kívül megmutatjuk, hogy GI az SPP bonyolultsági osztályban fekszik, és ezáltal *alsóbb* („low") az ismert valószínűségi bonyolultságosztályoknál. A formális megfogalmazást kerülve egy halmaz *alsóbb* egy C bonyolultsági osztálynál, ha mint „orákulum", mint jós semmiféle hasznos információt nem szolgáltat a C-számítások számára. A fent nevezett következmény bizonyításához, annak belátásához, hogy GI alsóbb bizonyos bonyolultsági osztályoknál, nagyon hasznosnak mutatkoznak egyes csoportelméleti algoritmusok.

4.1. Alapok

A bevezetésben említettük, hogy a bonyolultságelmélet többek között alsó korlátok meghatározásával foglalkozik. Ennek nehézsége abban rejlik, hogy nem elég a vizsgált problémát megoldó *egyetlen* konkrét algoritmus futási idejét elemezni, hanem meg kell mutatnunk, hogy a problémát megoldó *összes* elképzelhető algoritmus futási ideje *legalább akkora,* mint a felmutatott alsó korlát. Ezek közé az algoritmusok közé olyanok is tartozhatnak, melyeket talán még ki sem találtak, ebből következően első lépésként az algoritmus fogalmát formális és matematikailag pontos módon kell megragadnunk, máskülönben nem beszélhetünk az elképzelhető algoritmusok összességéről.

Az 1930-as évek óta már sok különböző formális algoritmus-modellre tettek javaslatot. Ezek a modellek ekvivalensek egymással olyan értelemben, miszerint bármelyik modell átalakítható egy tetszőlegesen választott másik ilyen modellé. Kissé pongyolán fogalmazva ez a transzformáció felfogható mint egyfajta – programnyelvek közti – fordítás. Az eddigi ismert modellek ekvivalenciája alapján az úgynevezett *Church-tézis* felteszi, hogy minden egyes algoritmusmodell pontosan a - természeténél fogva elég bizonytalanul körülhatárolható - „intuitív kiszámítások" fogalmát írja le. A bonyolultságelméletben használt szokásos algoritmusmodell az 1936-ban Alan Turing (1912–1954) által bevezetett Turing-gép. A Turing-gép a számítógépek egy nagyon egyszerű absztrakt modellje, melyet a következőkben szintaxisán és szemantikáján keresztül definiálunk. Eközben egyúttal bevezetünk két különböző kiszámíthatóságelméleti vezérgondolatot: a determinisztikusságot és a nemdeterminisztikusságot. Célszerűen először az általánosabb nem-determinisztikus Turing-gépeket írjuk le, amiből aztán speciális esetként közvetlenül adódnak a determinisztikus Turing-gépek.

Mindenek előtt a Turing-gép néhány technikai részletét és működését ismertetjük. Egy Turing-gép rendelkezik k darab mindkét irányban végtelen munkaszalaggal, melyek mezőkre vannak felosztva. A mezőkben betűk (jelek) állhatnak. Azt a tényt, hogy egy mező üres, egy speciális jellel, a $\square$-szimbólummal jelöljük. A tényleges számítás a munkasza-



4.1. ábra. Turing-gép.

lagokon történik. Egy számítás kezdetekor a bemeneti szó egy meghatározott szalagon, a bemeneti szalagon található, aminek minden bemeneti szóhoz nem használt további mezője a □ jelet tartalmazza. A számítás befejeztével az eredmény egy másik meghatározott szalagon, a kimeneti szalagon jelenik meg. Minden szalaghoz egy-egy író-olvasó fej férhet hozzá, amely a gép minden lépésében felülírhatja az aktuálisan olvasott betűt, ezután elmozdulhat egy mezőt jobbra vagy balra, illetve állva maradhat az aktuális mezőn. Eközben a gép aktuális állapota megváltozhat, melyet a belső memóriájában (*véges állapotú vezérlőegység,* angolul „finite control") feljegyez.[1] A 4.1. ábrán egy két szalagos Turing-gép látható.

4.1. definíció (a Turing-gép szintaxisa). *Egy k* ***szalagos nemdeterminisztikus Turing-gép*** *(röviden k-szalagos NTM) egy $M = (\Sigma, \Gamma, Z, \delta, z_0, \square, F)$ hetes, ahol Σ a* ***bemeneti ábécé****, Γ a munka ábécé (szalagjelek) melyre $\Sigma \subseteq \Gamma$, Z véges* ***állapothalmaz,*** *ahol $Z \cap \Gamma = \emptyset$, $\delta : Z \times \Gamma^k \to \mathfrak{P}(Z \times \Gamma^k \times \{L, R, N\}^k)$ az* átmenetfüggvény*, $z_0 \in Z$ a* ***kezdőállapot****, $\square \in \Gamma - \Sigma$ az* ***üres jel*** *és $F \subseteq Z$ a* ***végállapotok halmaza****. $\mathfrak{P}(S)$ egy S halmaz* ***hatványhalmazát*** *jelöli, tehát S összes részhalmazának halmazát.*

Minden $z, z' \in Z$, $x \in \{L, R, N\}$ és $a, b \in \Gamma$ mellett $(z', b, x) \in \delta(z, a)$ helyett a rövidített $(z, a) \mapsto (z', b, x)$ jelölést is használhatjuk. Ha az aktuális állapot z, a fej pedig egy a feliratú mezőn áll, akkor a fenti Turing-gép parancs a következő lépések végrehajtását jelenti:

- *a felülírását b-vel,*
- *egy új z' állapot felvételét és*
- *az $x \in \{L, R, N\}$-nak megfelelő fejmozgást, azaz egy mezőnyi lépést* balra *(L,* az angol „left" után), vagy egy lépést *jobbra* (R, azaz „right"), vagy a fej helyben marad az aktuális mezőn (N mint „neutral", *semleges*).

A k ***szalagos determinisztikus Turing-gép*** *(röviden a k-szalagos DTM) speciális eseténél akkor jutunk, ha a δ átmenetfüggvény $Z \times \Gamma^k$-ról $Z \times \Gamma^k \times \{L, R, N\}^k$-ra képez.*

A $k = 1$ esetben az 1-szalagos Turing-gépekhez jutunk, melyeket NTM-ként, illetve DTM-ként rövidítünk. Minden k-szalagos NTM illetve k-szalagos DTM szimulálható megfelelő egy szalagos Turing-géppel, amelynél a számítási idő legfeljebb négyzete az eredeti-

[1] Lehetséges további megkötések bevezetése is, például lehet a bemeneti szalag csak olvasható, illetve a kimeneti szalag csak írható. Hasonlóképp a technikai részletek kidolgozásakor számos további megoldás lehetséges, megkövetelhetjük, hogy meghatározott fejek csak meghatározott irányba mozdulhassanak el, vagy a szalagok csak egyik irányba legyenek végtelenek és így tovább.

nek. Ésszerű lehet több szalag használata, amennyiben a hatékonyság szerepét nem hanyagolhatjuk el.

A Turing-gépek felfoghatók mint olyan elfogadó automaták, melyek nyelveket (szóhalmazokat) fogadnak el. Emellett a Turing-gépek használhatók függvények kiszámításához is.

4.2. definíció (a Turing-gép szemantikája). *Legyen* $M = (\Sigma, \Gamma, Z, \delta, z_0, \square, F)$ *egy NTM.* M ***egy konfigurációja*** *egy* $k \in \Gamma^* Z \Gamma^*$ *szó. Ha* $k = \alpha z \beta$, *akkor* $\alpha\beta$ *jelentése az aktuális szalagfelirat (az aktuális szó a fej által már meglátogatott szalagrészről), a fej* β *első szimbólumán áll,* z *pedig az* M *aktuális állapota.*

M *konfigurációinak* $\mathfrak{K}_M = \Gamma^* Z \Gamma^*$ *halmazára definiálunk egy* $\vdash_M$ *bináris relációt, amely a* $k \in \mathfrak{K}_M$ *konfigurációból* $k' \in \mathfrak{K}_M$ *konfigurációba vezető átmenetet írja le a* δ *átmenetfüggvény segítségével. Legyen minden* α *és* β Γ^**-beli szóra, ahol* $\alpha = a_1 a_2 \cdots a_m$ *és* $\beta = b_1 b_2 \cdots b_n$, $m \geq 0$, $n \geq 1$ *és minden* $z \in Z$*-re:*

$$\alpha z \beta \vdash_M \begin{cases} a_1 a_2 \cdots a_m z' c b_2 \cdots b_n\,, & \text{ha } (z, b_1) \mapsto (z', c, N),\ m \geq 0 \text{ és } n \geq 1\,, \\ a_1 a_2 \cdots a_m c z' b_2 \cdots b_n\,, & \text{ha } (z, b_1) \mapsto (z', c, R),\ m \geq 0 \text{ és } n \geq 2, \\ a_1 a_2 \cdots a_{m-1} z' a_m c b_2 \cdots b_n, & \text{ha } (z, b_1) \mapsto (z', c, L),\ m \geq 1 \text{ és } n \geq 1. \end{cases}$$

Megvizsgálandó még két speciális eset:

1. *Ha* $n = 1$ *és* $(z, b_1) \mapsto (z', c, R)$ *(tehát* M *jobbra mozdul és egy* $\square$*-szimbólumot talál), akkor* $a_1 a_2 \cdots a_m z b_1 \vdash_M a_1 a_2 \cdots a_m c z' \square$.
2. *Ha* $m = 0$ *és* $(z, b_1) \mapsto (z', c, L)$ *(tehát* M *balra mozdul és egy* $\square$*-szimbólumot talál), akkor* $z b_1 b_2 \cdots b_n \vdash_M z' \square c b_2 \cdots b_n$.

Az M ***kezdőkonfigurációja*** x ***bemenet mellett*** *mindig* $z_0 x$, *az* M ***termináló konfigurációi*** x ***bemenet mellett*** *pedig* $\alpha z \beta$ *alakúak, ahol* $z \in F$ *és* $\alpha, \beta \in \Gamma^*$.

Legyen $\vdash_M$ *reflexív, tranzitív lezártja* $\vdash_M^*$. *Eszerint* $k, k' \in \mathfrak{K}_M$*-re* $k \vdash_M^* k'$ *akkor és csak akkor, ha létezik egy véges* $k_0, k_1, \ldots, k_t$ $\mathfrak{K}_M$*-beli konfiguráció-sorozat, melyre*

$$k = k_0 \vdash_M k_1 \vdash_M \cdots \vdash_M k_t = k',$$

ahol lehetséges, hogy $k = k_0 = k_t = k'$. *Ha az* M *kezdőkonfigurációja* x *bemenet mellett* $k_0 = z_0 x$, *akkor ezt a sorozatot* $M(x)$ ***véges kiszámításának*** *nevezzük, ha a* k'*-beli állapot végállapot, és azt mondjuk, hogy* M ***megáll az*** x ***bemenetre***. *Az* M ***által elfogadott nyelvet*** *a következőképp definiáljuk:*

$$L(M) = \{x \in \Sigma^* \mid z_0 x \vdash_M^* \alpha z \beta,\ z \in F \text{ és } \alpha, \beta \in \Gamma^*\}.$$

Az M *termináló állapotainak* F *halmazát feloszthatjuk az* F_a elfogadó végállapotok *és a* F_r elutasító végállapotok *halmazaira, ahol* $F = F_a \cup F_r$ *és* $F_a \cap F_r = \emptyset$. *Ebben az esetben* $L(M) = \{x \in \Sigma^* \mid z_0 x \vdash_M^* \alpha z \beta,\ z \in F_a \text{ és } \alpha, \beta \in \Gamma^*\}$ *az* M *által elfogadott nyelv.*

Az M ***kiszámít egy*** $f : \Sigma^* \to \Delta^*$ ***szófüggvényt***, *amennyiben minden* $x \in \Sigma^*$*-ra és minden* $y \in \Delta^*$*-ra*

1. $x \in D_f \iff M$ *az* x *bemenet mellett véges sok lépésben megáll,*
2. *minden* $x \in D_f$*-re:* $f(x) = y \iff z_0 x \vdash_M^* z y$ *egy megfelelő* $z \in F$*-re,*

$(z_0, a) \mapsto (z_1, \$, R)$	$(z_2, \$) \mapsto (z_2, \$, R)$	$(z_5, c) \mapsto (z_5, c, L)$
$(z_1, a) \mapsto (z_1, a, R)$	$(z_3, c) \mapsto (z_3, c, R)$	$(z_5, \$) \mapsto (z_5, \$, L)$
$(z_1, b) \mapsto (z_2, \$, R)$	$(z_3, \square) \mapsto (z_4, \square, L)$	$(z_5, b) \mapsto (z_5, b, L)$
$(z_1\$) \mapsto (z_1, \$, R)$	$(z_4, \$) \mapsto (z_4, \$, L)$	$(z_5, a) \mapsto (z_5, a, L)$
$(z_2, b) \mapsto (z_2, b, R)$	$(z_4, \square) \mapsto (z_6, \square, R)$	$(z_5, \square) \mapsto (z_0, \square, R)$
$(z_2, c) \mapsto (z_3, \$, R)$	$(z_4, c) \mapsto (z_5, c, L)$	$(z_0, \$) \mapsto (z_0, \$, R)$

4.2. ábra. Az M Turing-gép $L = \{a^n b^n c^n \mid n \geq 1\}$ nyelvhez tartozó δ parancsai.

*ahol D_f az f **értelmezési tartományát** jelöli. Azokat a szófüggvényeket, melyeket egy Turing-gép kiszámít, **kiszámítható szófüggvényeknek** nevezzük. Egy $f : \mathbb{N}^k \to \mathbb{N}$ függvény kiszámítható, ha egy $g : \{0, 1, \#\}^* \to \{0, 1\}^*$,*

$$g(bin(x_1)\#bin(x_2)\#\cdots\#bin(x_k)) = bin(f(x_1, x_2, \ldots, x_k))$$

által definiált g szófüggvények segítségével kiszámítható. A $bin(n)$ jelölés az $n \in \mathbb{N}$ kettes számrendszerbeli ábrázolását jelenti felvezető nullák nélkül, mint például $bin(17) = 10001$.

Mivel egy NTM minden konfigurációja több rákövetkező konfigurációval rendelkezhet, ezért természetesen adódik egy *kiszámítási fa*, melynek gyökere a kezdőkonfiguráció, levelei pedig a termináló konfigurációk. A fák speciális gráfok (lásd a 3.8. definíciót a 3.1.3. pontban, valamint a 3-2. feladatot), tehát csúcsokból és élekből állnak. Az M kiszámítási fájának csúcsai megfelelnek az M x bemenet melletti konfigurációi. Két k és k' konfiguráció között pontosan akkor vezet egy irányított él k-ból k'-be, ha $k \vdash_M k'$. Egy út az M kiszámítási fájában egy $k_0 \vdash_M k_1 \vdash_M \cdots \vdash_M k_l \vdash_M \cdots$ konfiguráció-sorozat, tehát $M(x)$ egy számításának felel meg. Egy NTM kiszámítási fája tartalmazhat végtelen utakat. Egy DTM esetében a kezdőkonfiguráció kivételével minden konfigurációt egyértelműen (*determinisztikusan*) meghatározza a megelőző konfigurációja, ezért a kiszámítási fa elfajul egy lineáris lánccá. A lánc a kezdőkonfigurációval indul, és – amennyiben a gép megáll az adott bemenetre – a termináló konfigurációval végződik, ellenkező esetben pedig a lánc végtelen.

4.1. példa. *Turing-gép.* Legyen egy L nyelv a következő: $L = \{a^n b^n c^n \mid n \geq 1\}$. Egy, az L nyelvet elfogadó Turing-gép definíciója az

$$M = (\{a, b, c\}, \{a, b, c, \$, \square\}, \{z_0, z_1, \ldots, z_6\}, \delta, z_0, \square, \{z_6\}),$$

ahol a δ átmenetfüggvénynek megfelelő Turing-gép parancsok listáját a 4.2. ábra tartalmazza. A 4.3. ábra egyenként jellemzi az M állapotait azok jelentésével, valamint a hozzájuk tartozó elvégzendő lépésekkel.

A bonyolultságelmélet szakemberei alapos emberek, szívesen teszik rendbe, foglalják rendszerbe a fontos problémák óriási, változatos halmazát. Ezen cél érdekében osztályozzák és katalogizálják a problémákat, majd bonyolultsági osztályokba sorolják őket. Minden egyes ilyen osztályba olyan problémák kerülnek, melyek a megoldásukhoz egy meghatározott bonyolultsági mérték szerint nagyjából azonos ráfordításokat igényelnek. A legelterjedtebb bonyolultsági mértékek az *időmérték* (egy algoritmus által a megoldáshoz szükséges lépések száma) és a *tármérték* (a számítógépen szükséges tárhely). Mi most az időmérték tárgyalására szorítkozunk.

Z	Jelentés	Teendő
z_0	kezdőállapot	új ciklust kezdeni
z_1	felismertük a-t	megkeresni a következő b-t
z_2	együttesen felismertük a, b-t	megkeresni a következő c-t
z_3	a, b, c ki van írva	megkeresni a jobb oldali szélt
z_4	jobb oldali szélre értünk	visszalépni és ellenőrizni, hogy minden a, b, c ki van-e írva
z_5	a teszt sikertelen	visszalépni a bal szélre és új ciklust kezdeni
z_6	a teszt sikerült	elfogadás

4.3. ábra. Az M állapotainak értelmezése.

Az algoritmus által a megoldáshoz szükséges „idő” fogalmán az elvégzett lépések számát értjük, mint a bemenet méretének függvényét. Formális modellünk, a Turing-gép esetében egy lépés a Turing-gép δ átmenetfüggvényének egyszeri alkalmazását jelenti, tehát a számítás egy konfigurációjának átmenetét a következőbe. Esetünkben a hagyományos *legrosszabb eset* bonyolultsági modell vizsgálatára szorítkozunk. Ennek megfelelően egy Turing-gép időfüggvényéhez a lehetséges tetszőleges n hosszú bemenetek közül azokat a döntő hatású bemeneteket vizsgáljuk, melyek mellett a gép a legtöbb lépést igényli – tehát a legrosszabb esetet feltételezzük. Ezzel ellentétben *átlagos eset* bonyolultság esetén egy algoritmus várható futási idejét mint egy (tetszőlegesen választott méretű bemenethez adott) valószínűségi eloszlásnak megfelelő átlagot kezeljük.

A következőkben definiáljuk a determinisztikus és nemdeterminisztikus időbonyolultsági osztályokat.

4.3. definíció (determinisztikus és nemdeterminisztikus időbonyolultság).

- *Legyen M egy DTM, melyre $L(M) \subseteq \Sigma^*$, valamint legyen $x \in \Sigma^*$ egy bemenet. Definiáljuk az $M(x)$ **időfüggvényét**, amely Σ^*-ból $\mathbb{N}$-be képez a következőképp:*

$$\mathrm{Time}_M(x) \quad = \quad \begin{cases} m, & \text{ha } M(x) \text{ pontosan } m+1 \text{ konfigurációval rendelkezik}, \\ \text{nem definiált} & \text{egyébként}. \end{cases}$$

Definiáljuk a $\mathrm{time}_M : \mathbb{N} \to \mathbb{N}$ *függvényt:*

$$\mathrm{time}_M(n) \quad = \quad \begin{cases} \max_{x:|x|=n} \mathrm{Time}_M(x), & \text{ha } \mathrm{Time}_M(x) \text{ definiált minden} \\ & |x| = n \text{ méretű } x\text{-re}, \\ \text{nem definiált} & \text{egyébként}. \end{cases}$$

- *Legyen M egy NTM, melyre $L(M) \subseteq \Sigma^*$, valamint legyen $x \in \Sigma^*$ egy bemenet. Definiáljuk az $M(x)$ **időfüggvényét**, amely Σ^*-ból $\mathbb{N}$-be képez a következőképp:*

$$\mathrm{NTime}_M(x) \quad = \quad \begin{cases} \min\{\mathrm{Time}_M(x,\alpha) \mid M \text{ az } \alpha \text{ úton elfogadja } x\text{-et}\}, & \text{ha } x \in L(M), \\ \text{nem definiált}, & \text{egyébként}. \end{cases}$$

Definiáljuk az $\mathrm{NTime}_M : \mathbb{N} \to \mathbb{N}$ *függvényt:*

$$\mathrm{ntime}_M(n) \quad = \quad \begin{cases} \max_{x:|x|=n} \mathrm{NTime}_M(x) & \text{ha } \mathrm{NTime}_M(x) \text{ értelmezve van minden olyan} \\ & x\text{–re}, \text{melyre}, |x| = n, \\ \text{nem definiált}, & \text{egyébként}. \end{cases}$$

$t(n)$	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$	$n = 60$
n	.00001 mp	.00002 mp	.00003 mp	.00004 mp	.00005 mp	.00006 mp
n^2	.0001 mp	.0004 mp	.0009 mp	.0016 mp	.0025 mp	.0036 mp
n^3	.001 mp	.008 mp	.027 mp	.064 mp	.125 mp	.256 mp
n^5	.1 mp	3.2 mp	24.3 mp	1.7 perc	5.2 perc	13.0 perc
2^n	.001 mp	1.0 mp	17.9 perc	12.7 nap	35.7 év	366 évszd
3^n	.059 mp	58 perc	6.5 év	3855 évszd	$2 \cdot 10^8$ évszd	$1.3 \cdot 10^{13}$ évszd

4.4. ábra. Néhány polinomiális és exponenciális időfüggvény összevetése.

- *Legyen t egy $\mathbb{N}$-ből $\mathbb{N}$-be képező kiszámítható függvény. Definiáljuk a* ***determinisztikus*** *és a* ***nemdeterminisztikus időbonyolultsági osztályokat*** *t* ***időfüggvény mellett*** *a következőképp:*

$$\mathrm{DTIME}(t) = \left\{ A \;\middle|\; \begin{array}{l} A = L(M) \text{ egy } M \text{ DTM-el és} \\ \text{minden } n \in \mathbb{N}\text{-re } \mathrm{time}_M(n) \leq t(n) \end{array} \right\},$$

$$\mathrm{NTIME}(t) = \left\{ A \;\middle|\; \begin{array}{l} A = L(M) \text{ egy } M \text{ NTM-el és} \\ \text{minden } n \in \mathbb{N}\text{-re } \mathrm{ntime}_M(n) \leq t(n) \end{array} \right\}.$$

- *Legyen* $\mathbb{P}\mathrm{ol}$ *az összes polinom halmaza. Definiáljuk a* P *és* NP *bonyolultsági osztályokat a következőképp:*

$$\mathrm{P} = \bigcup_{t \in \mathbb{P}\mathrm{ol}} \mathrm{DTIME}(t) \qquad \text{és} \qquad \mathrm{NP} = \bigcup_{t \in \mathbb{P}\mathrm{ol}} \mathrm{NTIME}(t).$$

DPTM-*el illetve* NPTM-*el jelöljük a polinomiális idejű DTM-eket illetve NTM-eket.*

Miért is olyan fontosak a P és az NP polinom idejű osztályok? Nyilvánvaló, hogy az exponenciális futási idejű algoritmusok általában bajosan nevezhetők hatékonynak. A 4.4. ábrán összehasonlítjuk néhány polinomiális és exponenciális időfüggvény növekedési ütemét, gyakorlati szempontból fontos bemeneti méretekre. Ehhez egy olyan számítógépből indulunk ki, amely másodpercenként egymillió művelet elvégzésére képes. Látható, hogy míg a polinomiális idejű algoritmusok a bemenet $n = 60$ méretéig elfogadható idő alatt előállítják az eredményt, addig például egy $t(n) = 3^n$ futási idejű algoritmus a bemenet viszonylag szerényebb, $n = 30$ méreténél 6 évnél is tovább dolgozna, az $n = 40$ méretű problémákhoz már majdnem 400 évezredre, és körülbelül $n = 50$-től pedig valóban csillagászati mértékű időre lenne szüksége.

Az utóbbi évtizedekben a számítógéptudományok és a hardver-technológia meggyőző fejlődését figyelhettük meg. Ha azonban fel is tesszük, hogy a gyorsabb és gyorsabb processzorok fejlesztési üteme megmarad az eddigi szinten, a 4.5. ábra szerint ez sem segít lényegesen az exponenciális futási idejű algoritmusok teljes végrehajtási idejének csökkentésében. Mi történne, ha egy olyan számítógépet használnánk, amely 100-szor vagy akár 1000-szer gyorsabb napjaink leggyorsabb számítógépeinél? Egy $t_i(n)$, $1 \leq i \leq 6$ függvényhez jelölje N_i azt a maximális feladatméretet, amelyet egy $t_i(n)$ időfüggvényű algoritmus egy órán belül meg tud oldani. A 4.5. ábra alapján látható, hogy még ezerszeres sebességnövekedés is csak körülbelül tízzel növeli N_5 értékét $t_5(n) = 2^n$ esetén. Ezzel szemben egy n^5 időfüggvényű algoritmus ugyanekkora sebességnövekedést feltételezve hozzávetőleg négyszer nagyobb problémákat tud megoldani egy óra alatt.

$t_i(n)$	Mai számítógép	100-szor gyorsabb	1000-szer gyorsabb
$t_1(n) = n$	N_1	$100 \cdot N_1$	$1000 \cdot N_1$
$t_2(n) = n^2$	N_2	$10 \cdot N_2$	$31.6 \cdot N_2$
$t_3(n) = n^3$	N_3	$4.64 \cdot N_3$	$10 \cdot N_3$
$t_4(n) = n^5$	N_4	$2.5 \cdot N_4$	$3.98 \cdot N_4$
$t_5(n) = 2^n$	N_5	$N_5 + 6.64$	$N_5 + 9.97$
$t_6(n) = 3^n$	N_6	$N_6 + 4.19$	$N_6 + 6.29$

4.5. ábra. Mi lenne, ha a számítógépek gyorsabbak lennének?

A következő dogma azt a széles körben elterjedt nézetet fogalmazza meg, miszerint a polinomiális idejű algoritmusok hatékonynak tekinthetők, míg a csak exponenciális alsó korláttal rendelkező algoritmusok kimondottan rosszak, hatékonynak nem nevezhetők.

4.4. dogma. *A polinomiális idő a hatékonyság intuitív fogalmának felel meg. Az exponenciális idő a rossz hatékonyság intuitív fogalmának felel meg.*

Természetesen egy dogma mindig is csak egy dogma marad, ennélfogva 4.4-et is kritikusan kell szemlélnünk. Egy $n^{10^{77}}$ lépésben dolgozó algoritmus például ugyan formálisan nézve egy $\mathbb{N}$ feletti, polinom lépésszámú algoritmus, de a polinom fokszáma éppen olyan nagy, mint az egész látható univerzumban lévő összes atom mostanság becsült darabszáma. Emiatt egy ilyen algoritmus egyáltalán nem hatékony, használata még a legkisebb problémaméret mellett is gyakorlatilag értelmetlen. Másrészt viszont egy $2^{0.00001 \cdot n}$ exponenciális időkorlát éppenséggel elfogadható lehet a gyakorlatban fontos problémaméretek mellett. Természetesen, ha valamikor is jelentkezik az exponenciális növekedés, a $0.00001 \cdot n$ kitevő esetében ez először csak nagyon nagy n mellett esedékes. Ez a két szélsőséges eset a valóságban mindenesetre szinte soha nem fordul elő. A természetes P-beli problémák lenyűgöző többsége megoldható olyan algoritmussal, aminek futási ideje alacsony fokú polinom, mint $O(n^2)$ vagy $O(n^3)$. Negyed-, ötöd- vagy ennél magasabb fokú polinomok csak nagyon ritkán lépnek fel.

A P osztály a 4.4. dogma szerint pontosan a hatékonyan megoldható problémákat tartalmazza. Az NP osztály sok olyan, gyakorlatban fontos problémát tartalmaz, melyekre a mai napig nem találtak hatékony megoldó algoritmust, mint például a kielégíthetőség és a gráfizomorfizmus problémái. Ezeket a 4. fejezetben részletesen vizsgáljuk. A kérdés, hogy a P és NP osztályok azonosak-e, szintén máig megoldatlan. Ez a híres P–NP probléma, amit tekinthetünk az elméleti informatika legfontosabb máig nyitott kérdésének. A kérdés különösen nagy szerepet játszik a kriptográfia területén, mivel a legtöbb ma használatos titkosítási rendszer azon a feltevésen alapszik, hogy bizonyos problémák nehezen megoldhatók. Ide tartoznak a faktorizálási probléma és a diszkrét logaritmus problémája, melyeket a 3. fejezetben vizsgálunk részletesebben. Amennyiben sikerülne a P = NP egyenlőséget bizonyítani, minden ilyen titkosítási rendszer elvesztené biztonságos tulajdonságát, és ezáltal haszontalanná válna.

A P–NP kérdés különösen nagy szerepet játszott az NP-teljesség elméletének létrejöttében. Ez az elmélet módszereket biztosít a legnehezebb NP-beli problémák alsó korlátainak bizonyításához. Egy ilyen bizonyításnál egyetlen egy nehéz NP-beli problémából kell kiindulni. Számos további NP-probléma NP-nehézsége ezután egy visszavezetés segítségével következik, ami során egy problémát egy másikká alakítunk át. Érdekes módon – mivel

ezek visszavezetések – léteznek olyan hatékony algoritmusok, melyekkel lehetővé válik a nehéz problémák NP-nehézségének bizonyítása. Azokat a problémákat, melyek NP-beliek és NP-nehezek, NP-teljes problémáknak nevezzük. Ezek nem tartozhatnak P-be, ezért nem is lehetnek hatékonyan megoldhatók, kivéve, ha esetleg P = NP fennáll. Az NP-teljesség elméletét a 4.2. alfejezetben mutatjuk be.

Gyakorlatok

4.1-1. Be lehet-e bizonyítani a Church-tézist? A választ indokoljuk.

4.1-2. Tekintsük a 4.1. példában szereplő M Turing-gépet.

(a) Adjuk meg M konfigurációinak sorozatát $x = a^3b^3c^2$, illetve $y = a^3b^3c^3$ bemenet mellett.

(b) Bizonyítsuk be M helyességét, azaz igazoljuk az $L(M) = \{a^n b^n c^n \mid n \geq 1\}$ egyenlőséget.

(c) Adjunk becslést az M futási idejére.

4.1-3. Mutassuk meg, hogy a 3.8. definícióban szereplő `GI` és `GA` problémák NP-beliek.

4.2. NP-teljesség

Az NP-teljesség elmélete módszereket biztosít NP-beli problémák alsó korlátainak igazolásához. Egy NP-probléma NP-*ben teljes*, ha az osztály legnehezebb problémái közé tartozik. Ennél fogva egy X probléma NP-nehézségének belátásához NP összes problémáját össze kell hasonlítani X-el, és meg kell mutatni, hogy X legalább olyan nehéz, mint az aktuálisan vizsgált másik probléma. Két probléma bonyolultsága polinomiális idejű visszavezetések segítségével mérhető össze egymással. A sok különbözőképp definiálható visszavezethetőségi típus közül számunkra most az úgynevezett *sok-egy-visszavezethetőség* lényeges, amit $\leq_{\mathrm{m}}^{\mathrm{p}}$-vel fogunk jelölni. Mivel ebben az alfejezetben nem foglalkozunk más visszavezethetőségi osztállyal, ezért egyszerűen „visszavezethetőség”-ről beszélünk. A 4.4. alfejezetben általánosabb visszavezethetőségeket is megismerhetünk, az úgynevezett *Turing-visszavezethetőséget* és az (*erős*) *nemdeterminisztikus Turing-visszavezethetőséget.*

4.5. definíció (visszavezethetőség, NP-teljesség). *Egy A halmaz* ***visszavezethető*** *egy B halmazra (formálisan $A \leq_{\mathrm{m}}^{\mathrm{p}} B$, ha létezik egy polinomiális időben kiszámítható r függvény, melyre minden $x \in \Sigma^*$ esetén $x \in A \iff r(x) \in B$. Egy B halmaz* $\leq_{\mathrm{m}}^{\mathrm{p}}$***-nehéz*** NP***-ben,*** *ha minden $A \in$ NP halmazra $A \leq_{\mathrm{m}}^{\mathrm{p}} B$. Egy B halmaz* $\leq_{\mathrm{m}}^{\mathrm{p}}$***-teljes*** NP***-ben*** *(vagy röviden* NP***-teljes****), ha $B \leq_{\mathrm{m}}^{\mathrm{p}}$-nehéz* NP*-ben és $B \in$* NP.

Egy adott X probléma NP-nehézségének bizonyításhoz látszólag végtelen sok hatékony algoritmus szükséges, majd minden egyes ilyen NP-beli problémát hatékonyan vissza kell vezetni X-re. Egy alapvető kutatási eredmény szerint azonban nem szükséges *végtelen sok* ilyen visszavezetés elvégzése, elég egyetlen NP-teljes V probléma visszavezetése X-re. Mivel a $\leq_{\mathrm{m}}^{\mathrm{p}}$-visszavezethetőség tranzitív (lásd a 4.2-2. gyakorlatot) valamint V NP-nehéz, ezért az $A \leq_{\mathrm{m}}^{\mathrm{p}} V \leq_{\mathrm{m}}^{\mathrm{p}} X$ visszavezetéssel következik X NP-nehézsége. Itt A befutja az NP-beli nyelveket.

1971-ben Stephen Cook megtalálta az első ilyen NP-teljes problémák egyikét, az ítéletlogikai kifejezések kielégíthetőségének problémáját („satisfiability problem”), amit röviden `SAT`-tal jelölünk. Sok NP-teljességi bizonyításhoz elég, ha a kielégíthetőség probléma egy

úgynevezett 3-SAT megszorításából indulunk ki, ahol az adott Boole-formulák konjunktív normálformában állnak rendelkezésre, és minden klóz pontosan három literált tartalmaz. A 3-SAT is NP-teljes. A diszjunktív normálformájú Boole-kifejezések kielégíthetőségének eldöntése hatékonyan megoldható.

4.6. definíció (kielégíthetőség probléma). *A* HAMIS *és az* IGAZ *Boole-konstansokat rendre* 0*-val és* 1*-gyel jelöljük. Legyenek* $x_1, x_2, \ldots, x_m$ *Boole-változók, tehát* $x_i \in \{0,1\}$ *minden* i*-re. A változókat és azok negáltját* ***literáloknak*** *nevezzük. Egy* φ ***Boole-formula kielégíthető,*** *ha létezik* φ *változóinak olyan behelyettesítése, ami a formulát igazzá teszi. Egy* φ *Boole-formula* ***konjunktív normálformájú*** *(röviden* ***KNF****,) ha* $\varphi(x_1, x_2, \ldots, x_m) = \bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{k_i} \ell_{i,j} \right)$ *formájú, ahol az* $\ell_{i,j}$ *literálok* $\{x_1, x_2, \ldots, x_m\}$*-ből valók. A literálok* $\bigvee_{j=1}^{k_i} \ell_{i,j}$ *diszjunkcióit* φ ***klózainak*** *hívjuk. Egy* φ *Boole-formula* k***-KNF,*** *ha* φ *KNF és* φ *minden klóza pontosan* k *literált tartalmaz. Definiáljuk a következő két problémahalmazt:*

$$\begin{aligned} \texttt{SAT} &= \{\varphi \mid \varphi \textit{ kielégíthető KNF Boole-formula}\}\,, \\ \texttt{3-SAT} &= \{\varphi \mid \varphi \textit{ kielégíthető 3-KNF Boole-formula}\}\,. \end{aligned}$$

4.2. példa. *Boole-kifejezések.* A következő két formula kielégíthető Boole-kifejezés (lásd még a 4.2-1. gyakorlatot):

$$\begin{aligned} \varphi(w,x,y,z) &= (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee \neg z) \wedge (w \vee \neg y \vee z) \wedge (\neg w \vee \neg x \vee z)\,; \\ \psi(w,x,y,z) &= (\neg w \vee x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (\neg w \vee y \vee z) \wedge (w \vee \neg x \vee \neg z)\,. \end{aligned}$$

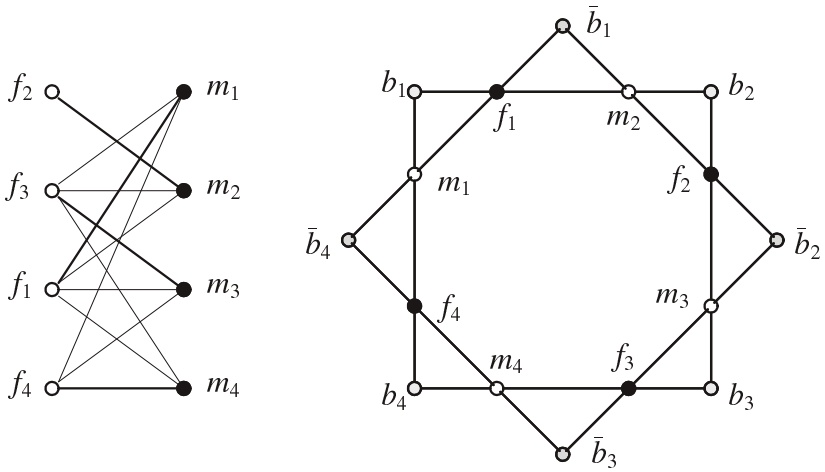
Itt φ egy 3-KNF formula, ezért φ 3-SAT-beli. Ezzel szemben ψ nem 3-KNF, mivel az első klóz négy literált tartalmaz, ezért ψ ugyan SAT-beli, de nem 3-SAT-beli.

A 4.7. tétel Cook korábban említett eredménye, amelyben az elsők között adott meg SAT-tal egy NP-teljes problémát. A bizonyítás alapötlete szerint kódoljuk egy tetszőleges M NPTM működését egy $\varphi_{M,x}$ Boole-formulával úgy, hogy $\varphi_{M,x}$ pontosan azon x-ekre legyen kielégíthető az x bemenettel, melyeket az M elfogad. A kielégíthetőség problémából kiinduló visszavezetésekhez sokszor célszerű, ha az adott kifejezések szigorú 3-KNF formában állnak rendelkezésre. Ez megoldható, mivel SAT visszavezethető 3-SAT-ra, ezért 3-SAT szintén NP-teljes (lásd az *Új algoritmusok* megfelelő fejezetét).

4.7. tétel (Cook). *A* SAT *és* 3-SAT *problémák* NP*-teljesek.*

Napjainkra több ezer NP-teljes problémát találtak már. Mi most a nagy választékból szemléltetési célokra a *háromdimenziós párosítás problémát* választjuk ki, és egy 3-SAT-ból való visszavezetéssel megmutatjuk NP-teljességét. A párosítási feladatoknál egymáshoz illő párokat vagy hármasokat alakítunk ki. Egy *kétdimenziós* (vagy *bipartite*) *párosítás* egymáshoz illő párok halmaza, egy *háromdimenziós* (vagy *tripartite*) *párosítás*, más néven *hármasítás* egymáshoz illő hármasok halmaza. Kétdimenziós párosítások jól szemléltethetők (irányítatlan) gráfokkal, lásd bővebben a 3.8. definíciót.

4.8. definíció (kétdimenziós párosítás probléma). *Egy* $2n$ *csúccsal rendelkező* G *gráf* ***páros****, ha a csúcshalmaza két* n *méretű, diszjunkt* V_1 *és* V_2 *halmazra osztható, ahol mindkettő* ***független halmaz****, azaz sem* V_1*, sem pedig* V_2 *csúcsai nincsenek élekkel összekötve. Csak*



4.6. ábra. *Balra*: A házasítás probléma megoldása. *Jobbra*: Az R reláció igazságérték-komponensei.

*V_1 és V_2-beli csúcsok között fordulhat elő él. G **tökéletes kétdimenziós párosítása** egy olyan n csúcsú $M \subseteq E(G)$ halmaz, melynél minden két különböző $\{v, w\}$ és $\{x, y\}$ M-beli élre $(v, x \in V_1,\ w, y \in W_2)$ $v \neq x$ és $w \neq y$. A **kétdimenziós párosítás probléma** kérdése az, hogy egy adott páros gráfhoz létezik-e kétdimenziós párosítás.*

4.3. példa. (Kétdimenziós párosítás probléma.) Képzeljünk el n darab házasodni szándékozó hölgyet és n darab házasodni szándékozó férfit, akik egy páros gráf csúcsait alkotják. A $V(G)$ csúcshalmaz tehát két részre bomlik, $V_{\text{vőlegény}} = \{f_1, f_2, \ldots, f_n\}$ és $V_{\text{menyasszony}} = \{m_1, m_2, \ldots, m_n\}$, valamint $V(G) = V_{\text{menyasszony}} \cup V_{\text{vőlegény}}$ és $V_{\text{menyasszony}} \cap V_{\text{vőlegény}} = \emptyset$. A $V_{\text{vőlegény}}$-beli és a $V_{\text{menyasszony}}$-beli csúcsokat élek köthetik össze, de nincs él $V_{\text{vőlegény}}$-beli csúcsok vagy $V_{\text{menyasszony}}$-beli csúcsok között. Egy kétdimenziós párosítás akkor áll elő, ha rendezhető n darab esküvő az n menyasszony és n vőlegény között úgy, hogy (Garey és Johnson művéből idézve) „a poligámia elkerülendő, valamint mindenki elfogadható feleséget illetve férjet kapjon”. Emiatt az értelmezés miatt a kétdimenziós párosítás problémát *házasítás problémának* is nevezzük. A 4.6. ábrán (bal oldalon) látható a házasítás probléma egy megoldása, ahol a vastagon nyomtatott élek a négy új házaspárt jelölik. Ismert, hogy a házasítás probléma hatékonyan megoldható. A valós élet gyakran igazolja az állítást: *Házasodni könnyű!*

A következőkben általánosítjuk a páros gráfokat és a kétdimenziós párosításokat három dimenzióra.

4.9. definíció (háromdimenziós párosítás probléma). *Legyen U, V és W három, páronként diszjunkt, n méretű halmaz. Legyen $R \subseteq U \times V \times W$ egy hármas reláció, tehát (u, v, w) hármasok halmaza, ahol $u \in U$, $v \in V$ és $w \in W$. R **egy háromdimenziós párosítása** egy n méretű $M \subseteq R$ halmaz, ahol minden két különböző (u, v, w) és $(\hat{u}, \hat{v}, \hat{w})$ hármasra $u \neq \hat{u}$, $v \neq \hat{v}$ és $w \neq \hat{w}$. Ez azt jelenti, hogy egy háromdimenziós párosítás semelyik két eleme nem illeszkedhet egymáshoz valamely koordinátájában. Definiáljuk tehát a **háromdimenziós párosítás***

problémát *a következőképp:*

$$\text{3-DM} = \left\{ (R,U,V,W) \;\middle|\; \begin{array}{l} U,\ V \text{ és } W \text{ páronként diszjunkt, nem üres, azonos méretű} \\ \text{halmaz és } R \subseteq U \times V \times W \text{ egy hármas reláció, amely} \\ \text{egy } |U| \text{ méretű háromdimenziós párosítást tartalmaz} \end{array} \right\}.$$

4.4. példa. *Háromdimenziós párosítás probléma.* Kilenc hónap telt el. Egy reggel az n darab boldog házastárspárunk útra kel a kórházba. Néhány órával később n csecsemő születik, akik hangos sírás mellett rögtön jelentősen megnövelik szüleik életének bonyolultságát például azáltal, hogy elcserélik a névtáblájukat, amin rajta áll, hogy melyik szülőpárhoz tartoznak. Ez nagy zűrzavarhoz vezet a szülőszobában. Még tovább rontva a helyzeten az újdonsült apák mindegyike – talán az izgalmas pillanatoktól összezavarva és a többi nő szépségétől elcsábítva – kijelenti, hogy sosem látta azt a hölgyet, aki makacsul kitart amellett, hogy az ő gyermekét hozta épp világra. Ehelyett hűtlen módon kijelenti, hogy egy *másik* hölgy a házastársa, aki az első mellett közvetlenül balra fekszik. A káosz teljes! A szülőszoba főnővére nehéz problémával szembesül: melyik szülőpároshoz melyik csecsemő tartozik? Másképp fogalmazva n darab boldog, harmonikus és páronként diszjunkt családot kell újra összeállítania, tehát egy háromdimenziós párosítás feladatot kell megoldania n édesapa, n édesanya és n csecsemő között. Nem csoda, hogy – szemben a kétdimenziós párosítás feladat hatékony megoldhatóságával – a 3-DM probléma NP-teljes. Végül a főnővérnek $3n$ darab vérvételt és kifinomult DNS teszteket kell elvégeznie, ám ennek tárgyalása már meghaladná könyvünk terjedelmét. Ismét átfogalmazva a 3-DM NP-teljességét a valós élet tapasztalataira: *Ha jönnek a gyermekek, nagyon nehéz feladat boldog és harmonikus, diszjunkt családokban élni!*

4.10. tétel. *A* 3-DM *probléma* NP*-teljes.*

Bizonyítás. A 4.2-4. gyakorlat szerint könnyen belátható, hogy 3-DM NP-beli. A 3-DM NP-nehézségének bizonyítása mögötti intuíciót legkönnyebb akkor megérteni, ha először megnézzük, hogyan jár el a szülőszobai főnővér a háromdimenziós párosítás probléma megoldásakor. Először is mindenkit ellát a teremben egy névtáblával, olyannal, ami megbízhatóan nem fog ismét eltűnni. Feltesszük, hogy az édesanyák az $m_1, m_2, \ldots, m_n$, az édesapák az $f_1, f_2, \ldots, f_n$, a csecsemők pedig a $b_1, b_2, \ldots, b_n$ elnevezéseket kapják. Ezután a főnővér egy újabb adag $\{\bar{b}_1, \bar{b}_2, \ldots, \bar{b}_n\}$ csecsemőt állít elő, ahol minden egyes $\bar{b}_i$ a b_i-vel megegyező klón[2], azaz ugyanúgy néz ki, és a DNS-e b_i-vel azonos öröklődési információkat tartalmaz. Ezt elvégezve két kört alakít ki a $4n$ emberből. A $2n$ darab szülő egy belső kört formáz, amiben apák és anyák váltakoznak. A külső körben az n csecsemő és n klónja helyezkedik el, ugyancsak váltakozva. A két kör szomszédos személyei kapcsolatban állnak egymással úgy, ahogy a 4.6. ábra (jobb oldalon) $n = 4$ esetében mutatja: minden apa két anyával és két csecsemővel van összekötve.

A következő gondolatmenetben az i indexek modulo n értendők, a példában $n = 4$. Vizsgáljunk minden i-t, melyre[3] i modulo $n = 4$. Az f_i-edik apa az m_{i-1}-edik anyával tartja magát házasnak, és vele közös gyermekének pedig az $(i-1)$ csecsemőt. Ezzel szemben az m_i anya ahhoz ragaszkodik, hogy *ő* az f_i apa felesége, és a közös gyermekük pedig az i-edik csecsemő. Mindkét ellentmondó állítást egy-egy háromszög szemlélteti a 4.6. ábrán (jobb oldalon), melyek csúcsai f_i, m_{i-1} és $\bar{b}_{i-1}$, illetve m_i, f_i és b_i. A $2n$ darab háromszög

[2] A csecsemőklónozás technikai kérdései és az ezzel kapcsolatban felmerülő etikai kérdések szintén túlmutatnak ezen könyv határain, ennélfogva szép csendben mellőzzük őket.

[3] A modulo n aritmetikáját a 3. fejezet végén, a 3-1. példával kapcsolatban ismertettük.

mindegyike egy-egy potenciális családot képez. A főnővérnek azt kell megállapítania, hogy mely háromszögek reprezentálják az *eredeti* n darab családot, és melyek nem. Az egyetlen lehetőség n darab diszjunkt család létrehozására az, ha vagy minden háromszögbe egy b_i csecsemőt, vagy pedig minden háromszögbe egy $\bar{b}_i$ klóncsecsemőt választunk. Azáltal, hogy a $3n$ vérvételt és a fentebb említett DNS-teszteket elvégezte, a főnővér meghozhatja a helyes döntést és minden apához kijelölheti igazi feleségét és igazi gyermekét. A főnővér tehát így állítja elő az n darab eredeti családot. A fennmaradó n darab csecsemőt (és ez a főnővér módszerének szomorú oldala – és a csecsemőklónozásé úgy általában) adoptálják, vagy árvaházakban helyezik el.

A bonyolultságelmélettel foglalkozók nem nagyon értenek a DNS-tesztekhez vagy a klónozáshoz, szerencsére azonban a kielégíthetőség-problémát jól ismerik. A 3-DM probléma NP-nehézségének belátásához definiáljuk 3-SAT egy visszavezetését 3-DM-re. Legyen adott φ Boole-formula 3-KNF-ben, vagyis $\varphi(x_1, x_2, \ldots, x_\ell) = C_1 \wedge C_2 \wedge \cdots \wedge C_n$, ahol φ minden C_j klóza pontosan három literált tartalmaz. Előállítandó polinomiális időben egy (R, U, V, W) négyes 3-DM-ből, ahol $R \subseteq U \times V \times W$ egy hármas reláció a páronként diszjunkt, nem üres, azonos méretű U, V és W halmazokkal úgy, hogy

$$\varphi \text{ kielégíthető} \iff R \text{ egy } |U| \text{ méretű, } M \text{ háromdimenziós párosítást tartalmaz} . \qquad (4.1)$$

R különböző jellegű hármasokból áll, melyek mögött mindig más és más szándék rejtőzik. Azonos jellegű hármasokat elem-csoportokká, komponensekké fogunk össze. Az első ilyen komponens azokból az R-beli hármasokból áll, melyek φ formula változóinak olyan meghatározott hozzárendelését biztosítják, ami konzisztens φ összes klózára nézve. Ennek megfelelően ha ugyanaz a változó előfordul különböző klózokban, akkor minden egyes ilyen előforduláshoz ugyanazt az igazságértéket kell hozzárendelni, ennélfogva ezeket az alkotóelemeket R-ből *igazságérték-komponensnek* nevezzük.

Minden φ-beli x_i változóhoz készítünk pontosan $2n$ darab U-beli $b^i_1, b^i_2, \ldots, b^i_n$ és $\bar{b}^i_1, \bar{b}^i_2, \ldots, \bar{b}^i_n$ elemet, ahol n a φ klózainak száma. A b^i_j reprezentálja x_i, a $\bar{b}^i_j$ pedig $\neg x_i$ előfordulását φ j-edik C_j klózában. Mivel az összes literál nem fordul elő minden klózban, ezért néhány b^i_j vagy $\bar{b}^i_j$ nem feleltethető meg valamely φ-beli literál előfordulásának. Ezeken kívül minden φ-beli x_i változóhoz létrehozunk további n darab V-beli $m^i_1, m^i_2, \ldots, m^i_n$ és n darab W-beli $f^i_1, f^i_2, \ldots, f^i_n$ elemet, melyek a 4.6. ábrán (jobb oldalon) lévő belső kört alkotják, ha $n = 4$ és az ábrán nem szerepelnek a fenti indexek. Kapcsoljuk most össze az m^i_j, f^i_j és b^i_j, továbbá az f^i_j, m^i_{j-1} és $\bar{b}^i_{j-1}$ elemeket egymással, ahogy a 4.6. jobb oldali ábráján vázoltuk. Az így létrehozott komponensek háromszögei megfelelnek az R hármasainak. A belső körből m^i_j és f^i_j csak azokban a komponensekben fordulnak elő, melyek az x_i változónak felelnek meg, miközben a külső körből b^i_j és $\bar{b}^i_j$ más komponensekben is előfordulhatnak. Formálisan az X igazságérték-komponensek $X = \bigcup_{i=1}^{\ell} X_i$ alakúak, ahol $X_i = F_i \cup T_i$ minden φ-beli x_i-re, a következő két halmaz által definiálva:

$$\begin{aligned} F_i &= \{(b^i_j, m^i_j, f^i_j) \mid 1 \le j \le n\} ; \\ T_i &= \{(\bar{b}^i_j, m^i_j, f^i_{j+1}) \mid 1 \le j < n\} \cup \{(\bar{b}^i_n, m^i_n, f^i_1)\} . \end{aligned}$$

Mivel a belső kör m^i_j és f^i_j elemei csak X_i komponenseiként szerepelhetnek, ezért R minden M párosítása pontosan n darab hármast tartalmaz, vagy minden ilyen hármast F_i-ből,

vagy minden hármast T_i-ből. Az F_i és T_i közötti választás biztosítja az x_i változók megfeleltetését az *igaz* vagy a *hamis* értékeknek. Mivel X_i tartalmazza φ összes x_i előfordulását, ezért ez a választás az egész formulára nézve konzisztens, következésképp R minden egyes M párosítása a φ formula egy hozzárendelését határozza meg úgy, hogy a hozzárendelés minden x_i változója akkor és csak akkor lehet igaz, ha $M \cap X_i = T_i$.

Most hozzáillesztjük R-hez az $Y = \bigcup_{j=1}^{n} Y_j$ hármasok halmazát úgy, hogy minden Y_j a φ megfelelő C_j klózát ellenőrizze. Ennek megfelelően az Y komponenst R *kielégíthetőségi-komponensének* nevezzük. Minden C_j klózhoz készítünk két $v_j \in V$ és $w_j \in W$ elemet, melyek csak Y_j-ben fordulnak elő. Ezen kívül Y_j három további elemet tartalmaz az $\bigcup_{i=1}^{\ell} \left(\{b_j^i\} \cup \{\bar{b}_j^i\}\right)$ halmazból, melyek megfelelnek C_j három literáljának, és R más komponenseiben is előfordulhatnak. Formálisan Y_j-t φ minden C_j klózára a következő halmazzal definiáljuk:

$$Y_j = \{(b_j^i, v_j, w_j) \mid x_i \text{ előfordul } C_j\text{-ben}\} \cup \{(\bar{b}_j^i, v_j, w_j) \mid \neg x_i \text{ előfordul } C_j\text{-ben}\} .$$

Mivel a v_j és w_j $(1 \le j \le n)$ elemek egyike sem szerepelhet az R halmaz Y_j-től különböző bármelyik másik hármasában, ezért R minden M háromdimenziós párosítása pontosan egy Y_j-beli hármast tartalmaz, vagy (b_j^i, v_j, w_j)-t, vagy pedig $(\bar{b}_j^i, v_j, w_j)$-t. Továbbá M pontosan akkor tartalmaz egy Y_j-beli hármast – vagy b_j^i-t (ha x_i előfordul C_j-ben), vagy pedig $\bar{b}_j^i$-t (ha $\neg x_i$ fordul elő C_j-ben) – ha ez az elem nem fordul elő az $M \cap X_i$ hármasai között. Ez pedig pontosan akkor következik be, ha az M igazságérték-komponensével meghatározott hozzárendelés kielégíti a C_j klózt.

Mostanra U pontosan $2n\ell$ elemet tartalmaz, azonban V és W mindössze $n\ell + n$ elemű. Bővítsük további $n(\ell - 1)$ elemmel V-t és W-t, amivel a három halmaz azonos méretű lesz. Először is hozzáadjuk a $v_{n+1}, v_{n+2}, \ldots, v_{n\ell}$ elemeket V-hez, valamint a $w_{n+1}, w_{n+2}, \ldots, w_{n\ell}$ elemeket W-hez. Ezeken kívül kiegészítjük R-et hármasok következő halmazával:

$$\begin{aligned} Z = {} & \{(b_j^i, v_k, w_k) \mid 1 \le i \le \ell \text{ és } 1 \le j \le n \text{ és } n+1 \le k \le n\ell\} \cup \\ & \{(\bar{b}_j^i, v_k, w_k) \mid 1 \le i \le \ell \text{ és } 1 \le j \le n \text{ és } n+1 \le k \le n\ell\} . \end{aligned}$$

A vicc pedig az, hogy ha létezik $R - Z$ egy háromdimenziós párosítása, és az összes, R igazságérték- és kielégíthetőségi-komponenseivel biztosított feltétel teljesül, akkor ez a háromdimenziós párosítás U-ból pontosan $n(\ell - 1)$ elemet szabadon hagy, amit aztán egy egyértelműen meghatározott Z-beli (v_k, w_k) párral „párosíthatunk”. Az $R - Z$ háromdimenziós párosításainak ilyen bővítése R egy háromdimenziós párosítását adja. Formálisan az U, V és W halmazokat a következőképp definiáljuk:

$$\begin{aligned} U &= \{b_j^i \mid 1 \le i \le \ell \text{ és } 1 \le j \le n\} \cup \{\bar{b}_j^i \mid 1 \le i \le \ell \text{ és } 1 \le j \le n\} ; \\ V &= \{m_j^i \mid 1 \le i \le \ell \text{ és } 1 \le j \le n\} \cup \{v_k \mid 1 \le k \le n\ell\} ; \\ W &= \{f_j^i \mid 1 \le i \le \ell \text{ és } 1 \le j \le n\} \cup \{w_k \mid 1 \le k \le n\ell\} . \end{aligned}$$

Az $R \subseteq U \times V \times W$ relációt a $R = X \cup Y \cup Z$ egyenlőség határozza meg. Mivel R pontosan $2n\ell + 3n + 2n^2\ell(\ell - 1)$ hármast tartalmaz, tehát polinomiális számút a φ formula méretére nézve, és R szerkezete könnyen meghatározható φ szerkezetéből, ezért a visszavezetés polinomiális időben kiszámítható. A (4.1) állítás következik az R konstrukciója közbeni észrevételekből.

(4.1) formális bizonyítását a 4.2-5. gyakorlat keretében az Olvasóra bízzuk. ■

Gyakorlatok

4.2-1. Adjunk meg egy-egy kielégítő behelyettesítést a 4.2. példában szereplő φ és ψ Boole-formulákhoz.

4.2-2. Mutassuk meg az $\leq_{\mathrm{m}}^{\mathrm{p}}$-visszavezethetőség tranzitivitását: $(A \leq_{\mathrm{m}}^{\mathrm{p}} B \wedge B \leq_{\mathrm{m}}^{\mathrm{p}} C) \Longrightarrow A \leq_{\mathrm{m}}^{\mathrm{p}} C$.

4.2-3. Adjunk meg egy SAT $\leq_{\mathrm{m}}^{\mathrm{p}}$ 3-SAT visszavezetést. Alakítsuk át hozzá egy adott Boole-formula minden olyan klózát KNF klózzá, amely csak egy, csak kettő vagy háromnál több literált tartalmaz. Eközben a formula kielégíthetősége ne változzon.

4.2-4. Mutassuk meg, hogy a SAT és 3-DM problémák NP-beliek.

4.2-5. Lássuk be a 4.10. bizonyításban szereplő (4.1) egyenlőséget.

4.3. Az ítéletlogika kielégíthetőség-problémája

4.3.1. 3-SAT determinisztikus időbonyolultsága

A SAT kielégíthetőség probléma és annak 3-SAT megszorítása a 4.7. tétel alapján NP-teljes. Ha SAT esetleg P-beli lenne, akkor az általánosan elfogadott sejtéssel szemben P = NP rögtön következne, ezért nagyon valószínűtlennek tűnik, hogy léteznének SAT-hoz vagy 3-SAT-hoz hatékony determinisztikus algoritmusok. Akkor viszont mekkora a legjobb algoritmusok futási ideje 3-SAT-ra? Mivel a formula pontos szerkezete nyilvánvalóan befolyásolhatja a futási időt, ezért mi ebben a pontban 3-SAT problémára koncentrálunk, ahol minden klóz pontosan három literálból áll. Az itt bemutatott eredmények közvetlenül átvihetők k-SAT-ra, SAT megszorítására pontosan k literállal klózonként. 3-SAT „naiv” determinisztikus algoritmusa a következőképp működik: egy adott n-változós φ Boole-formulánál egymás után az összes lehetséges behelyettesítést kipróbáljuk, kiértékelve a formulát a mindenkori értékekkel. Ha φ valamely behelyettesítésre igaz, az algoritmus elfogad, ellenkező esetben mind a 2^n behelyettesítést eredmény nélkül kipróbálja, és az algoritmus elutasít. Ez a módszer nyilvánvalóan $O(2^n)$ idővel dolgozik. Megoldható-e a probléma gyorsabban?

Igen, megoldható gyorsabban is. De mielőtt megmutatnánk *hogyan*, a következő kérdést szeretnénk feltenni: *miért?* Mi haszna annak, ha 3-SAT felső időkorlátját $O(2^n)$ alá szorítjuk, valahová $O(c^n)$-re, ahol a c konstansra $1 < c < 2$ teljesül, ami még mindig egy exponenciális időkorlát? A haszon ott jelentkezik, hogy elérhető annak az n_0 küszöbértéknek a hátrébb tolása, ahonnan kezdve az exponenciális idő jellemző lesz, azaz ahol az algoritmus abszolút futási ideje $n \geq n_0$ méretű bemenetek mellett elviselhetetlenül nagyra nő. A 3-SAT „naiv” algoritmusának $O(2^n)$ korlátját hozzávetőleg annyira le lehet szorítani, hogy egy $O(c^n)$ algoritmussal kétszeres méretű bemenetek feldolgozása válik lehetővé azonos idő alatt, ami nagy gyakorlati jelentőséggel bír. Pontosan ez az eset áll fenn $c = \sqrt{2} \approx 1.4142$ mellett, ahol az algoritmus $O(\sqrt{2}^{2n}) = O(2^n)$ időben dolgozik (lásd még a 159. oldalon a 4.10. táblázatot).

A következőkben bemutatunk egy algoritmust 3-SAT-hoz, amely a *visszalépésen* alapul. Ez az algoritmusfejlesztési technika azon problémák megoldására alkalmas, ahol a megoldás n darab alkotóelemből tevődik össze, melyeknél több választási lehetőség adódik.

Például 3-SAT egy megoldása egy kielégítő behelyettesítés, ami n darab igazságértékéből áll, továbbá minden ilyen igazságértékhez két választási lehetőség adódik: IGAZ vagy HAMIS, illetve 1 vagy 0. Az ötlet ezután abból áll, hogy az üres megoldásból (részleges behelyettesítés, ahol a változók nincsenek kitöltve) kiindulva lépésről lépésre, a *Backtracking*-eljárás rekurzív hívásával a probléma mind nagyobb részleges megoldását hozzuk létre, amíg meg nem találjuk a teljes megoldást, már amennyiben az létezik. A keletkező rekurziós fában[4] a gyökeret megjelöljük az üres megoldással, a probléma teljes megoldásai pedig a levelek szintjére kerülnek. Ha az algoritmus végrehajtása során megállapítjuk, hogy a rekurziós fa aktuális ága „halott", vagyis az addig konstruált részmegoldás semmi esetben sem egészíthető ki a probléma teljes megoldásává, akkor az eddig elért csúcs alatti részfát nyugodtan levághatjuk. A meghívandó eljárást visszaléptethetjük, próbálkozhatunk az előzőleg konstruált részmegoldás újabb folytatásával. Ennek a visszalépésnek köszönheti nevét a *visszalépés* algoritmikai elve. A rekurziós fa „halott" részeinek levágásával időt tudunk megtakarítani.

BACKTRACKING-SAT(φ, β)

```
1  if β-ban φ minden változója ki van töltve
2     then return φ(β)
3     elseif β hamissá teszi φ egy klózát                ▷ Halott ág.
4           then return 0
5     elseif BACKTRACKING-SAT(φ, β0)
6           then return 1
7     else return BACKTRACKING-SAT(φ, β1)
```

A BACKTRACKING-SAT algoritmus egy φ Boole-formula bemenet és φ valamely változóinak β részleges hozzárendelése mellett egy Boole-értéket ad vissza: 1-et, ha a β parciális megoldás egy minden változót tartalmazó φ-t kielégítő hozzárendeléssé bővíthető, és 0-át egyébként. A parciális hozzárendeléseket ebben az esetben a $\{0, 1\}$ ábécé fölötti, n-nél nem hosszabb szavaknak tekintjük. Az algoritmus kezdő hívása a BACKTRACKING-SAT(φ, λ), ahol λ az üres hozzárendelés. Látható, hogy ha az előzetesen konstruált β parciális hozzárendelés φ valamely klózát hamissá teszi, akkor az már soha nem lesz kiegészítve egy φ-t kielégítő hozzárendeléssé, a rekurziós fa aktuális csúcs alatti részfáját levágjuk (lásd még a 4.3-1. gyakorlatot).

A BACKTRACKING-SAT futási idejének felső becsléséhez az adott φ formulából egy tetszőlegesen választott C_j klózt vizsgálunk. A φ minden β kitöltendő hozzárendelése – és ebből most elsősorban a három C_j-ben előforduló változó – igazságértékeket kell kapjon értékül. A 0 vagy 1 sorozatok $2^3 = 8$ lehetőségéből az algoritmus biztosan kiválasztja azt az egyet, amely C_j-t hamissá teszi. A szóban forgó csúcs BACKTRACKING-SAT(φ, β) rekurziós fájában tehát egy „halott" részfához vezet, amit nyugodtan levághatunk. A φ formula szerkezetéből adódóan további „halott" részfák állnak elő, melyeket nem kell figyelembe venni, ebből adódik a BACKTRACKING-SAT egy felső korlátjára legrosszabb esetben $O((2^3 - 1)^{n/3}) = O(\sqrt[3]{7}^n) \approx O(1.9129^n)$, ami a „naiv" 3-SAT algoritmus $O(2^n)$ korlátját még

[4] A félreértések elkerülése végett hangsúlyozni kell, hogy a rekurziós fa más, mint egy NTM kiszámítási fája. A BACKTRACKING-SAT algoritmus teljesen determinisztikusan működik, egy rekurziós fa mélységi bejárásának megfelelően. Egy ilyen fa belső csúcsai az algoritmus rekurzív hívásait reprezentálják, a gyökere az első hívást, a leveleknél pedig további hívások nélkül terminál az algoritmus. A rekurziós fa egy $\hat{k}$ csúcsa pontosan akkor gyereke egy k csúcsnak, ha a $\hat{k}$ hívás az algoritmus k után kiváltott számításán belül következik be, továbbá nincs olyan $k^\star$ hívás, ami k-n belül van és $k^\star$-on belül van.

mindig javítja valamelyest.

A 3-SAT determinisztikus időbonyolultsága még lejjebb szorítható, például Monien és Speckenmeyer oszd-meg-és-uralkodj algoritmusa $O(1.618^n)$ felső korláttal rendelkezik. A felső korlát világrekordját determinisztikus 3-SAT-algoritmusra jelenleg egy $O(1.481^n)$ korlátos, lokális keresésen alapuló megoldás tartja, melyet Dantsin és társai készítettek. Vannak más, nemdeterminisztikus megközelítések is. Ezek közül egyet mutatunk be, egy Schöning munkáiból merítő, úgynevezett „véletlen séta" algoritmust.

4.3.2. 3-SAT valószínűségi időbonyolultsága

Egy *véletlen séta* egy (véletlen) bejárás adott struktúrán, például egy euklideszi térben, egy végtelen rácson vagy egy gráfon. Esetünkben a gráfok bejárása érdekes, azon belül is olyan gráfok bejárása, melyek egy meghatározott sztochasztikus automatát reprezentálnak. Egy sztochasztikus automata egy speciális *véges automata*.

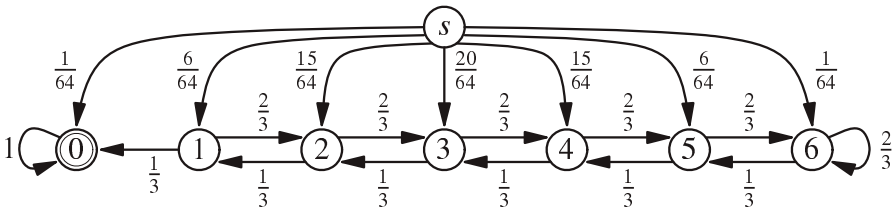
Egy véges automata szemléltethető állapotgráfjának segítségével. A véges automata állapotait csúcsok, az állapotok közti átmeneteket irányított, egy Σ ábécé szimbólumaival címkézett élek jelölik. Egy csúcs a kitüntetett *kezdőállapot* szerepet kapja, ennél kezdődik az automata működése, és akkor fejeződik be, ha a teljes bemenet feldolgozásra került. Az automata minden lépésben pontosan egy bemeneti szimbólumot olvas. Az állapotgráf bizonyos csúcsait *végállapotként* jelöljük meg, és amennyiben egy végállapotot érünk el a feldolgozás végén, akkor az automata elfogadja a bemenetet.

Véges automatákkal szavakat ismerhetünk fel. Egy Σ^*-beli $w = w_1 w_2 \cdots w_n$ szót az automata pontosan akkor fogad el, ha a kezdőállapotból kiindulva w minden egyes w_i szimbólumát a megfelelő sorrendben elolvassa, az állapotátmeneteket mindig a w_i-vel jelölt él követésének megfelelően hajtja végre, és végül elér egy végállapotot. Egy véges automata által elfogadott nyelv pontosan az ilyen módon elfogadott szavakból áll.

Egy $\mathcal{S}$ *sztochasztikus automata* éleit utólagosan számokkal is felcímkézzük. Az $\mathcal{S}$ állapotgráfjában egy u-ból v-be vezető élhez rendelt $p_{u,v}$ $(0 \leq p_{u,v} \leq 1)$ érték annak valószínűségét adja meg, hogy $\mathcal{S}$ az u állapotból v állapotba lép. Egy sztochasztikus automata (véletlen) állapotátmeneteinek folyamatát a sztochasztika tárgykörében *Markov-láncoknak*, a végállapotokat pedig *abszorpciós* állapotoknak nevezhetjük. Egy $\mathcal{S}$ sztochasztikus automata esetében így természetesen egy w szó elfogadása (és ezzel az $\mathcal{S}$ által elfogadott nyelv definiálása) csak adott valószínűséggel következik be, mely valószínűséget a w feldolgozása során bejárt élek címkéiből ismerhetjük.

Esetünkben azonban nem a nyelvfelismerés érdekes a sztochasztikus automata feladatai közül. Az automatát egy bejárásra szeretnénk alkalmazni, amit a Random-SAT valószínűségi algoritmus valósít meg. A Random-SAT algoritmus megpróbál az n-változós φ Boole-formulához egy kielégítő behelyettesítést találni, amennyiben létezik ilyen.

Random-SAT valamely φ bemenet mellett először is készít egy véletlenszerű β kezdeti hozzárendelést, ahol β minden egyes bitje függetlenül választott egyenletes eloszlás mellett, tehát β minden bitje 0 vagy 1 értéket vesz fel 1/2–1/2 valószínűséggel. A behelyettesítéseket ismét mint $\{0,1\}$ feletti n-hosszú szavakat kezeljük. Tegyük fel, hogy φ kielégíthető, és legyen $\tilde{\beta}$ φ tetszőlegesen választott kielégítő behelyettesítése φ-be. Legyen X az a valószínűségi változó, ami a β *és* $\tilde{\beta}$ *Hamming-távolságát* fejezi ki, vagyis azon bitek számát, ahol β és $\tilde{\beta}$ nem egyeznek meg. X nyilvánvalóan a $j \in \{0, 1, \ldots, n\}$ értékeket veheti fel, valamint binomiális eloszlású n és 1/2 paraméterekkel. Ennek megfelelően $X = j$ valószínűsége $\binom{n}{j} 2^{-n}$.



4.7. ábra. Egy RANDOM-SAT bejárás sztochasztikus automatájának állapotgráfja.

RANDOM-SAT(φ)

```
1  for i ← 1 to ⌈(4/3)^n⌉                          ▷ n a φ változóinak száma.
2      do válasszunk egy β ∈ {0,1}^n behelyettesítést, egyenletes eloszlás szerint
3      for j ← 1 to n
4          do if φ(β) = 1
5                  then return β                    ▷ β kielégíti φ-t.
6                  else válasszunk egy C = (x ∨ y ∨ z) klózt, ahol C(β) = 0
7                       válasszunk véletlenszerűen egy ℓ ∈ {x,y,z} literált,
                                egyenletes eloszlás szerint
8                       állapítsuk meg β ℓ-edik lefoglalt β_ℓ ∈ {0,1} bitjét
9                       változtassuk meg β-ban β_ℓ -t (1 − β_ℓ)-re
10 return „φ kielégíthetetlen”
```

A RANDOM-SAT algoritmus ezután ellenőrzi, hogy a kezdetben választott β hozzárendelés kielégíti-e a φ formulát, és amennyiben igen, akkor elfogadja a bemenetet. Egyébként pedig ha β nem teszi igazzá φ-t, akkor léteznie kell egy φ-beli klóznak, amit β nem elégít ki. A RANDOM-SAT tetszőlegesen kiválaszt egy ilyen klózt, a kiválasztott klózban egyenletes eloszlás mellett véletlenszerűen kijelöl egy literált, és megcseréli az ehhez a literálhoz rendelt bitet az aktuális β behelyettesítésben. Ezt n-szer megismétli, majd amennyiben az aktuális behelyettesítés még mindig nem elégíti ki a φ formulát, akkor a RANDOM-SAT újraindul egy új kezdeti hozzárendeléssel és megismétli a teljes fent leírt próbát t-szer, ahol $t = \lceil (4/3)^n \rceil$.

A 4.7. ábrán egy $\mathcal{S}$ sztochasztikus automata szerepel, melynek éleit nem szimbólumokkal, hanem mindössze átmenet-valószínűségekkel címkéztük. A RANDOM-SAT algoritmus φ bemenet melletti működését a következőkben mint $\mathcal{S}$ egy bejárását mutatjuk be. Az s kezdőállapotból kiindulva – amit később már sosem érintünk – RANDOM-SAT(φ) az n és 1/2 paraméterű binomiális eloszlásnak megfelelően lép tovább egy $j \in \{0, 1, \ldots, n\}$ állapotba. Ezt mutatja a kép felső része egy $n = 6$ változós φ Boole-formula esetében. Egy ilyen j állapot azt jelenti, hogy a véletlenszerűen választott β kezdeti behelyettesítés és a $\tilde{\beta}$ kielégítő behelyettesítés állandó közötti Hamming-távolság j. Amíg $j \neq 0$, addig RANDOM-SAT(φ) a belső **for** ciklusban a kielégítő megoldást keresve mindig megváltoztatja az aktuális β behelyettesítés egy β_ℓ bitjét $(1 - \beta_\ell)$-re. Az $\mathcal{S}$ bejárásban ez megfelel egy lépésnek a $j - 1$ állapotba balra vagy a $j + 1$ állapotba jobbra, ahol csak n-nél kisebb vagy n-nel egyenlő állapot érhető el.

Az állandóra választott $\tilde{\beta}$ behelyettesítés kielégíti φ-t, tehát φ minden klózából legalább egy literált igazzá tesz. Ha minden klózban *pontosan* egyet kijelölünk ezek közül a $\tilde{\beta}$ által

kielégített literálok közül, úgy pontosan akkor lépünk egy lépést balra, ha ezt az ℓ literált RANDOM-SAT(φ) kiválasztja. Egy balra lépés valószínűsége ($j > 0$-ból $j - 1$-be) nyilvánvalóan mindig 1/3, a jobbra lépés (j-ből $j + 1$-be) valószínűsége pedig mindig 2/3.

Bármikor is érjük el a $j = 0$ állapotot, ott β és $\tilde{\beta}$ Hamming-távolsága 0 lesz. Ennek megfelelően β kielégíti a φ formulát, és RANDOM-SAT(φ) visszaadja β aktuális értékét, majd megáll elfogadó állapotban. Egy $j \neq 0$ állapotban természetesen belefuthatunk egy $\tilde{\beta}$-tól különböző kielégítő behelyettesítésbe is, de mivel ez a lehetőség az elfogadás valószínűségét csak növelheti, ezért az elfogadás valószínűségének becslésénél most figyelmen kívül hagyjuk. Ha a $j = 0$ állapotot nem érjük el β behelyettesítés n-szeri bitcseréjével, akkor a kezdő behelyettesítést olyan rosszul választottuk, hogy RANDOM-SAT(φ) egyszerűen eldobja, és egy új kezdeti értékkel próbál ismét szerencsét.

Mivel annak valószínűsége, hogy a (számunkra kedvező) 0 végállapottól jobbra távolodjunk nagyobb, mint annak valószínűsége, hogy 0 irányába balra haladjunk, azt gondolhatjuk, hogy RANDOM-SAT nem túl nagy valószínűséggel jár sikerrel. Annak esélyét sem szabad azonban alábecsülni, hogy s-ből rögtön a nulladik lépés után a 0 közelébe kerülünk. Minél közelebbi a 0-hoz a kezdő pozíciónk, annál nagyobb annak valószínűsége, hogy a rákövetkező véletlen jobbra- vagy balra-lépések lefutásával 0-ba ütközünk.

RANDOM-SAT eredményességének valószínűségére és az algoritmus futási idejére vonatkozó elemzést itt most nem mutatjuk be teljes részletességében, csak vázlatszerűen. Az egyszerűség kedvéért tegyük fel, hogy n a 3 egész értékű többszöröse. Legyen p_i annak valószínűsége, hogy RANDOM-SAT n lépésen belül eléri a 0 állapotot, azzal a feltétellel, hogy RANDOM-SAT a bejárás nulladik lépésében (β kezdeti véletlen hozzárendelés megválasztásakor) az $i \leq n/3$ állapotba kerül. Amennyiben például kezdetben az $n/3$ állapotba kerülünk, akkor legfeljebb $n/3$ lépés megengedett a „rossz” irányba jobbra, ami kiegyenlíthető a „helyes” irányba, balra tett lépésekkel. Egyéb esetekben a 0 állapot nem érhető el n lépésben. Általában az i állapotból kiindulva legfeljebb $(n - i)/2$ lépést tehetünk jobbra, amiből a következő becslés adódik p_i-re:

$$p_i = \binom{n}{\frac{n-i}{2}} \left(\frac{2}{3}\right)^{(n-i)/2} \left(\frac{1}{3}\right)^{n-(n-i)/2} . \tag{4.2}$$

Legyen továbbá q_i annak valószínűsége, hogy RANDOM-SAT a bejárás nulladik lépésében az $i \leq n/3$ állapotba jut. Ekkor természetesen fennáll

$$q_i = \binom{n}{i} \cdot 2^{-n} . \tag{4.3}$$

Végül pedig legyen p az eredményesség valószínűsége, amikor RANDOM-SAT a külső *for* ciklus egy körében eléri a 0 állapotot. Ez lehetséges $j > n/3$ állapotokból is. Ennélfogva fennáll

$$p \geq \sum_{i=0}^{n/3} p_i \cdot q_i .$$

Ez az összeg közelíthető az entrópiafüggvény segítségével, továbbá az összeg tagjaiban (4.2) és (4.3) binomiális együtthatói közelíthetők a Stirling-formulával, így végül $\Omega((3/4)^n)$ adódik p alsó korlátjának.

A hibás futás elkerülése érdekében a RANDOM-SAT algoritmus összesen t független kísérletet hajt végre, melyek mindegyike új kezdőértékkel indul, és legalább a fent megadott

hozzávetőleges $(3/4)^n$ valószínűséggel sikerrel jár. Mivel a kísérletek függetlensége miatt ezek a valószínűségi értékek szorzódnak, ezért RANDOM-SAT eredményes futásának valószínűsége – tehát annak valószínűsége, hogy RANDOM-SAT megadja φ egy kielégítő behelyettesítését, amennyiben az létezik – nagyon közel van 1-hez. Ha pedig φ kielégíthetetlen, RANDOM-SAT sosem követ el hibát, tehát ezekben az esetekben a kimenet mindig „φ kielégíthetetlen" lesz.

Az algoritmus futási ideje megegyezik az eredményes futás $p \approx (3/4)^n$ valószínűségének reciprokával, mivel egy hiba valószínűsége (amikor a t próba során egyszer sem találunk kielégítő behelyettesítést, pedig φ kielégíthető) $(1-p)^t \leq e^{-t \cdot p}$-vel becsülhető. Amennyiben tehát nem szeretnénk egy előre adott ε hibavalószínűségi értéket átlépni, elegendő t értékét úgy megválasztani, hogy $e^{-t \cdot p} \leq \varepsilon$ illetve $t \geq \ln(1/\varepsilon)/p$ fennálljon. A konstans tényezőktől eltekintve ez $t = \lceil (4/3)^n \rceil$ választásával elérhető, az algoritmus futási ideje tehát $O((4/3)^n)$.

Gyakorlatok

4.3-1. Indítsuk el a BACKTRACKING-SAT algoritmust $\varphi = (\neg x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (\neg u \vee y \vee z) \wedge (u \vee \neg y \vee z)$ Boole-formulával, és konstruáljuk meg lépésről lépésre φ egy kielégítő behelyettesítését. Rajzoljuk fel az így kialakuló rekurziós fát, és határozzuk meg a fa azon részeit, melyeket levágunk, mert nem vezethetnek megoldáshoz.

4.4. Gráfizomorfizmus és alsóság

A következő alfejezetben szükségünk lesz a 3.1.3. alfejezetben leírt csoport- és gráfelméleti alapokra. Különösen fontos felidézni a 3.6. definíció permutációcsoport fogalmát, a `GI` gráfizomorfizmus problémát, valamint a 3.8. definíció `GA` gráfautomorfizmus problémáját (lásd még a 3.4. példát is).

4.4.1. Visszavezethetőségek és bonyolultsági hierarchiák

A 4.2. pontban megismerkedtünk a hatékonyan megoldható házasítás problémával, valamint az NP-teljes `SAT`, `3-SAT` és `3-DM` problémákkal. Könnyen meggondolhatjuk, hogy P = NP pontosan akkor teljesül, ha *minden* NP-beli probléma – beleértve az NP-teljes problémákat is – P-ben helyezkedik el. Amennyiben P ≠ NP, akkor P-ben nem lehetnek NP-teljes problémák. Megoldható-e vajon hatékonyan minden NP-beli probléma a hihetőnek tűnő P ≠ NP feltevés mellett, tehát P-beliek-e, vagy pedig NP-teljesek? Tudunk-e vajon olyan NP-beli problémákat adni P ≠ NP feltevés mellett, melyek nem oldhatók meg hatékonyan, de nem is NP-teljesek? A választ a következő tétel adja meg.

4.11. tétel (Ladner). *Ha* P ≠ NP, *akkor léteznek olyan problémák* NP*-ben, melyek se nem* P*-beliek, se nem* NP*-teljesek.*

A Ladner által megadott problémák valamelyest „mesterségesek" olyan szempontból, hogy közvetlenül a 4.11. tétel bizonyításához készültek. Vannak azonban olyan természetes problémák is NP-ben, melyek megfelelő jelöltek arra, hogy se P-beliek, se pedig NP-teljesek ne legyenek. Ezek közül egy `GI`, a gráfizomorfizmus probléma, és ezt szeretnénk most bizonyítani. Ehhez definiálunk NP-n belül két, Schöning által bevezetett bonyolultsági osztályhierarchiát, az úgynevezett *alsó-*, valamint a *felső-hierarchiát.* Ahhoz, hogy

ezt a két hierarchiát definiálni tudjuk, be kell vezetnünk az NP-re épülő *polinomiális idő-hierarchia* fogalmát, ennek definíciójához pedig szükségünk van egy, a 4.5. definícióban bevezetett sok-egy visszavezethetőségnél általánosabb visszavezethetőségre, nevezetesen a *Turing-visszavezethetőség*, a $\leq_T^p$ fogalmára. Definiáljuk továbbá a *nemdeterminisztikus* és az *erősen nemdeterminisztikus Turing-visszavezethetőség* – $\leq_T^{NP}$ és $\leq_{sT}^{NP}$ – fogalmakat, melyek a polinomiális idő-hierarchia esetében fontosak. Ezek a visszavezethetőségek az *orákulumos Turing-gép* fogalmára építkeznek. Most tehát definiáljuk a nevezett fogalmakat.

4.12. definíció (orákulumos Turing-gép). *Az **orákulum-halmaz** (röviden **orákulum**) szavak egy halmaza. Egy B orákulummal rendelkező* orákulumos M Turing-gép *egy olyan Turing-gép, ami rendelkezik egy speciális **kérdés-szalaggal**, állapotainak halmaza pedig tartalmaz egy speciális $z_?$ **kérdés-állapotot**, valamint $z_{\mathtt{yes}}$ és $z_{\mathtt{no}}$ **válasz-állapotokat**. Amíg M nincs a $z_?$ állapotban, addig pontosan úgy viselkedik, mint egy szokványos Turing-gép. Ha azonban működése során $z_?$ kérdés-állapotba kerül, akkor megszakítja futását, és kérdést intéz az orákulumhoz az éppen a kérdés-szalagon álló q szóról. Az orákulumot egyfajta „fekete dobozként" képzelhetjük el: B orákulum egy lépésben eldönti, hogy q szó B-beli-e, függetlenül attól, hogy ennek eldöntése milyen nehézségű feladat. Ha $q \in B$, akkor M a következő lépésben $z_{\mathtt{yes}}$ válasz-állapotba kerül és folytatja működését. Ellenkező esetben (ha $q \notin B$) M $z_{\mathtt{no}}$ állapotban folytatja működését. Ekkor azt mondjuk, hogy az M Turing-géppel x **bemenet kiszámítása relatív B orákulumra nézve**, a kiszámítást pedig $M^B(x)$-el jelölünk. Legyen $L(M^B)$ az M^B által elfogadott nyelv. Egy C bonyolultsági osztályt **relativizálhatónak** nevezünk, ha reprezentálható orákulumos Turing-gépekkel (üres orákulum-halmaz mellett) az előzőeknek megfelelően. Definiáljuk egy C relativizálható bonyolultság-osztály és egy B orákulum mellett a B-re relatív C osztályt a következőképp:*

$$C^B = \{L(M^B) \mid M \text{ egy } C\text{-t reprezentáló orákulumos Turing-gép}\} .$$

Ha $\mathcal{B}$ halmazok egy osztálya, akkor legyen $C^{\mathcal{B}} = \bigcup_{B\in\mathcal{B}} C^B$.

NPOTM (illetve DPOTM) jelöli a *nemdeterminisztikus* (illetve a *determinisztikus*) *polinomiális idejű orákulumos Turing-gépeket.* Definiálhatók például a következő osztályok:

$$\mathrm{NP}^{\mathrm{NP}} = \bigcup_{B\in\mathrm{NP}} \mathrm{NP}^B = \{L(M^B) \mid M \text{ egy NPOTM, } B \text{ pedig NP-beli}\} ;$$
$$\mathrm{P}^{\mathrm{NP}} = \bigcup_{B\in\mathrm{NP}} \mathrm{P}^B = \{L(M^B) \mid M \text{ egy DPOTM, } B \text{ pedig NP-beli}\} .$$

Amennyiben az orákulum üres halmaz ($\emptyset$), akkor a nem relativizálható $\mathrm{NP} = \mathrm{NP}^\emptyset$ illetve $\mathrm{P} = \mathrm{P}^\emptyset$ osztályokhoz jutunk. NPOTM helyett ekkor NPTM-ről beszélhetünk, DPOTM helyett pedig DPTM-ről. Különösen jól alkalmazhatók az orákulumos Turing-gépek keresési feladatokhoz, mint amilyen a prefixkeresés is, ahogy a következő példa mutatja. Az itt alkalmazott Pre-Iso NP-orákulum szolgáltatja azt az információt, hogy hogyan lehet az üres szóból kiindulva bitről bitre haladva az NP-beli `GI` probléma legkisebb megoldását előállítani, amennyiben létezik ilyen megoldás.

4.5. példa. *Prefixkeresés a legkisebb izomorfizmus után orákulumos Turing-géppel.*

A `GI` gráfizomorfizmus problémát az 3.1.3. alfejezet 3.8. definíciójában fogalmaztuk meg. Legyenek G és H két adott gráf $n \geq 1$ darab csúccsal. A G és H közti izomorfizmusokat „$(G, H) \in$ `GI`"

megoldásának nevezzük. Az $\mathrm{Iso}(G,H)$ izomorfizmus-csoport tartalmazza a „$(G,H) \in \mathtt{GI}$” összes megoldását, valamint $\mathrm{Iso}(G,H) \neq \emptyset \iff (G,H) \in \mathtt{GI}$. Ha $(G,H) \in \mathtt{GI}$, akkor egy lexikografikusan legkisebb megoldás előállítása a célunk, ellenkező esetben pedig az üres λ szó kimenet segítségével tudatnunk kell, hogy „$(G,H) \notin \mathtt{GI}$”. Tehát, ki szeretnénk számítani a következőképp definiált f függvényt:

$$f(G,H) \quad = \quad \begin{cases} \min\{\pi \,|\, \pi \in \mathrm{Iso}(G,H)\}, & \text{ha } (G,H) \in \mathtt{GI}\ , \\ \lambda, & \text{ha } (G,H) \notin \mathtt{GI}\ , \end{cases}$$

ahol a lexikografikus rendezésre vonatkozó minimumot $\mathfrak{S}_n$-ből képezzük. A $\mathfrak{S}_n$ definíciója a következő: kezeljük a $\pi \in \mathfrak{S}_n$ egy permutációját mint n hosszú, $[n] = \{1,2,\ldots,n\}$ ábécé fölötti $\pi(1)\pi(2)\cdots\pi(n)$ szót. Pontosan akkor írható $\pi < \sigma$ ($\pi,\sigma \in \mathfrak{S}_n$), ha létezik egy olyan $j \in [n]$, melyre minden $i < j$ esetén $\pi(i) = \sigma(i)$, valamint $\pi(j) < \sigma(j)$. Amennyiben egy $\sigma \in \mathfrak{S}_n$ permutációból kitörlünk néhány $(i,\sigma(i))$ párt, akkor *parciális permutációt* kapunk, amit szintén kezelhetünk $[n]$ fölötti szóként. A $\sigma \in \mathfrak{S}_n$ egy $k \leq n$ *hosszú prefixe* σ egy olyan parciális permutációja, amely minden $(i,\sigma(i))$ párt tartalmaz $i \leq k$ mellett, viszont egyetlen $(i,\sigma(i))$ párt sem $i > k$ esetén. A $k = 0$ esetben σ prefixe a λ üres szó, valamint $k = n$ esetben a teljes permutáció. Ha π a $\sigma \in \mathfrak{S}_n$ egy $k < n$ hosszú prefixe és $w = i_1 i_2 \cdots i_{|w|}$ egy $[n]$ fölötti $|w| \leq n-k$ hosszú szó, akkor jelölje πw azt a parciális permutációt, ami π-t a $(k+1,i_1),(k+2,i_2),\ldots,(k+|w|,i_{|w|})$ párokra bővíti. Ha $1 \leq j \leq |w|$ esetén $\sigma(k+j) = i_j$, akkor πw is σ egy prefixe. Definiáljuk G és H gráfok mellett az $\mathrm{Iso}(G,H)$-beli izomorfizmusok prefixeinek halmazát:

$$\text{Pre-Iso} \quad = \quad \{(G,H,\pi) \,|\, (\exists w \in \{1,2,\ldots,n\}^*)\,[w = i_1 i_2 \cdots i_{n-|\pi|} \text{ és } \pi w \in \mathrm{Iso}(G,H)]\}\ .$$

Figyeljük meg, hogy $n \geq 1$-re a λ üres szót egy permutáció sem képzi le $\mathfrak{S}_n$-be, valamint hogy $\mathrm{Iso}(G,H) = \emptyset$ pontosan akkor áll fenn, ha $(G,H,\lambda) \notin$ Pre-Iso, ami viszont akkor és csak akkor igaz, ha $(G,H) \notin \mathtt{GI}$.

N-Pre-Iso(G,H)

```
1  if (G,H,λ) ∉ Pre-Iso
2    then return 0
3    else π ← λ
4         j ← 0
5         while j < n                                  ▷ G és H mindig n csúcsú.
6               do i ← 1
7                  while (G,H,πi) ∉ Pre-Iso
8                        do i ← i + 1
9                        π ← πi
10                       j ← j + 1
11        return π
```

Az N DPOTM prefixkereséssel, a Pre-Iso orákulum segítségével számítja ki az f függvényt (lásd még a 4.4-2. gyakorlatot). Jelölje FP a polinomiális időben kiszámítható függvények osztályát. Ekkor $f \in \mathrm{FP}^{\text{Pre-Iso}}$. Mivel Pre-Iso egy NP-beli halmaz (lásd a 4.4-2. gyakorlatot), ezért $f \in \mathrm{FP}^{\mathrm{NP}}$ következik.

A 4.5. példa alapján elláthatók orákulummal a függvényeket kiszámító Turing-gépek is, valamint relativizálhatók függvény osztályokra, mint például az FP. Másrészt alkalmazhatók orákulumként halmazok helyett függvények is. A 4.12. definícióval összevetve az $f : \Sigma^* \to \Sigma^*$ függvényorákulum egy q kérdésre egy lépésben nem az „igen” vagy „nem”

válaszokat adja, hanem $|f(q)|$ lépésben visszaadja az $f(q)$ függvényértékeket. A következő tétel kimondja, hogy konstruálható teljes izomorfizmus két izomorf gráf közötti parciális izomorfizmusból egy f függvényorákulum segítségével.

4.13. tétel. *Legyenek G és H izomorf gráfok. Legyen f egy függvényorákulum, melyre $f(G,H) = (x,y)$, ahol $x \in V(G)$, valamint fennáll $y \in V(H)$, és $\sigma(x) = y$ egy $\sigma \in \mathrm{Iso}(G,H)$ izomorfizmussal. Ekkor létezik egy M DPOTM, amely az f orákulummal kiszámol egy $\varphi \in \mathrm{Iso}(G,H)$ izomorfizmust.*

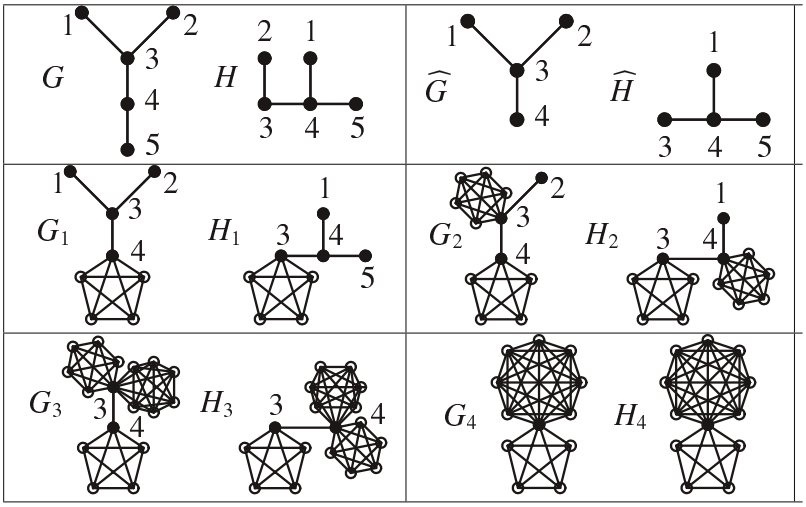
A 4.13. tétel szerint tehát az NP-beli GI probléma egy teljes megoldásának konstrukciója visszavezethető a GI egy parciális megoldására (érdemes lehet ezt összevetni a Backtracking-SAT algoritmussal, ami bitről bitre bővíti a 3-SAT kifejezések parciális megoldását, amíg azok teljesek nem lesznek).

Képzeljük el, hogy Merlin, a 3.5. alfejezet 3.12. ábráján szereplő GI probléma zéró ismeretű protokolljában egy $\sigma \in \mathrm{Iso}(G,H)$ izomorfizmust küld Artúrnak, ahogy az kívánta, sajnos azonban az átvitelnél némely bitek elvesznek vagy használhatatlanná válnak. Artúr tehát csak egy parciális π izomorfizmust kap meg σ-ból. Rekonstruálhat azonban Merlin segítségével a 4.13. tételnek köszönhetően egy $\varphi \in \mathrm{Iso}(G,H)$ teljes izomorfizmust még akkor is, ha π csak egyetlen csúcspárból áll. Figyeljük meg, hogy φ-nek nem kell a Merlin által eredetileg elküldött σ izomorfizmusnak lennie.

A 4.6. példa a bizonyítás alapötletét szemlélteti konkrét gráfokkal. A formális bizonyítástól most eltekintünk. Egy lényeges tulajdonság, amit itt kihasználunk GI úgynevezett *ön-visszavezethetősége*. Anélkül, hogy belemennénk a technikai részletekbe, ezt a fontos fogalmat a következőképp írhatjuk le: egy A halmaz pontosan akkor *ön-visszavezethető*, ha létezik egy M DPOTM, ami egy A orákulum segítségével elfogadja magát az A halmazt. M az x bemeneti szóról egyszerűen megkérdezhetné az A orákulumot, amikor is x A-beliségének eldöntése természetesen triviális lenne, ezért az ön-visszavezethetőségnél megtiltjuk a bemenetre magára vonatkozó kérdéseket. Ehelyett az M csak olyan kérdéseket tehet fel az A orákulumnak, melyek *kisebbek* a bemenetnél, ahol a kisebb jelzőt a hagyományos lexikografikus rendezésből általánosított értelemben használjuk. Ezt a definíciót a 4.13. tételnek megfelelően szintén átültethetjük halmazok helyett függvényekre. **4.6. példa.**

(Teljes izomorfizmus előállítása parciális izomorfizmusból.) A 4.8. ábra két izomorf G és H gráfjára $\mathrm{Iso}(G,H) = \{\sigma,\varphi\}$, ahol $\sigma = \left(\begin{smallmatrix}1&2&3&4&5\\1&5&4&3&2\end{smallmatrix}\right)$ és $\varphi = \left(\begin{smallmatrix}1&2&3&4&5\\5&1&4&3&2\end{smallmatrix}\right)$. A következőkben leírunk és körüljárunk egy kudarchoz vezető naiv megközelítést. Tegyük fel, hogy a 4.13. tétel algoritmusa az f orákulumhoz intézett kérdésekkel meghatároz egy (x,y) csúcspárt, ahol $\sigma(x) = y$ vagy $\varphi(x) = y$. Így felismerve (x,y)-t az algoritmus törli x-et G-ből illetve y-t H-ból, és hasonló módon fokozatosan addig halad, amíg a gráfok üresek nem lesznek. Ezzel biztosítottuk, hogy az algoritmus legfeljebb $n = 5$ lépésben terminál jon, és remélhetőleg a tárolt csúcspárok sorrendje a keresett $\mathrm{Iso}(G,H)$-beli izomorfizmust adja, tehát vagy σ-t, vagy pedig φ-t. Ez azonban nem feltétlenül következik be. Tegyük fel például, hogy az első (G,H) párra vonatkozó kérdésre az f orákulum az $(5,2)$ csúcspárt választolja. A 4.13. tétel algoritmusa ekkor egyszerűen törli G-ből az 5, valamint H-ból a 2 csúcsokat (valamint az 5-ből illetve a 2-ből kiinduló éleket). Ekkor a 4.8. ábra $\widehat{G}$ és $\widehat{H}$ gráfjaihoz jutunk. Az $\mathrm{Iso}(\widehat{G},\widehat{H})$ azonban hat izomorfizmust tartalmaz, amiből csak kettő *összeegyeztethető* a $(5,2)$ párral (lásd a 4.4-3. gyakorlatot). Ez tehát azt jelenti, hogy $\mathrm{Iso}(\widehat{G},\widehat{H})$ hat izomorfizmusából csak kettő parciális izomorfizmusa σ-nak és φ-nek. Ezután az algoritmus következő lépésében eltárolhatja például az új $(4,5)$ párt, ami sem σ-hoz, sem pedig φ-hez nem tartozik.

Annak érdekében, hogy kizárhassuk ezeket az eseteket, a 4.13. tétel f orákulumos M DPOTM-ja másképp jár el. Nem csak egyszerűen törli azokat a csúcspárokat, melyeket az orákulum segítségével



4.8. ábra. Példa a 4.13. tételben szereplő konstrukcióra.

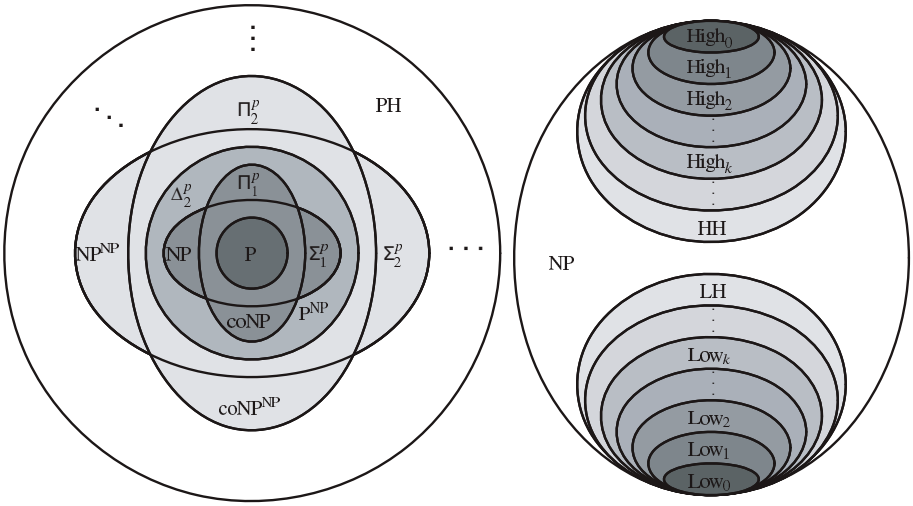
meghatározott, hanem elegendően nagy méretű klikkek segítségével kijelöli a törölt csúcsok szomszédos csúcsait. Egy k méretű *klikk* olyan gráf, aminek k darab csúcsa páronként egy-egy éllel mindenhol össze van kötve. Példánkban tehát az első $(5,2)$ csúcspár törlése után a G-beli 4 és H-beli 3 csúcsokat egy 5 méretű klikkel jelöljük meg. Ebből adódik a (G_1, H_1) új pár, lásd a 4.8. ábrán. Figyeljük meg, hogy most minden $\pi \in \mathrm{Iso}(G_1, H_1)$ izomorfizmus összeegyeztethető a $(5,2)$ csúcspárral az eredeti $\mathrm{Iso}(G,H)$-beli σ-ból és φ-ből.

Ha M fokozatosan e szerint az eljárás szerint halad, akkor az orákulum válaszok egy meghatározott sorozatára például az 4.8. ábrán szereplő (G_2, H_2), (G_3, H_3) és (G_4, H_4) gráfpárok állhatnak elő. Ekkor a (G_4, H_4)-nél egyértelműen meghatároztuk az utolsó $(4,3)$ csúcspárt, M pedig ebben az esetben előállította a $\varphi = \left(\begin{smallmatrix}1&2&3&4&5\\5&1&4&3&2\end{smallmatrix}\right)$ teljes izomorfizmust $\mathrm{Iso}(G,H)$-ból.

Most pedig különböző visszavezethetőségeket definiálunk az orákulumos Turing-gép fogalmából kiindulva. Minden itt vizsgált visszavezethetőség hatékonyan – tehát polinomiális időben – kiszámítható.

4.14. definíció (Turing-visszavezethetőség). *Legyen $\Sigma = \{0,1\}$ egy bináris ábécé, legyenek A és B ezen Σ feletti szavak halmazai, és legyen $\mathcal{C}$ egy bonyolultsági osztály. A $\mathcal{C}$-beli halmazok komplementereinek osztályát definiáljuk mint* $\mathrm{co}\mathcal{C} = \{\overline{L} \mid L \in \mathcal{C}\}$. *Definiáljuk a következő visszavezethetőségeket:*

- *Turing-visszavezethetőség:* $A \leq_{\mathrm{T}}^{\mathrm{p}} B \iff A = L(M^B)$ *egy M DPOTM-ra.*
- *Nemdeterminisztikus Turing-visszavezethetőség:* $A \leq_{\mathrm{T}}^{\mathrm{NP}} B \iff A = L(M^B)$ *egy M NPOTM-ra.*
- *Erős nemdeterminisztikus Turing-visszavezethetőség:* $A \leq_{\mathrm{sT}}^{\mathrm{NP}} B \iff A \in \mathrm{NP}^B \cap \mathrm{coNP}^B$.
- *Ha $\leq_r$ egy fent definiált visszavezethetőség, akkor egy B halmazt pontosan akkor nevezünk* ***$\leq_r$-nehéznek $\mathcal{C}$-ben****, ha $A \leq_r B$ teljesül minden $A \in \mathcal{C}$ halmazra. Egy B halmaz pontosan akkor* ***$\leq_r$-teljes $\mathcal{C}$-ben****, ha B $\leq_r$-nehéz $\mathcal{C}$-re és $B \in \mathcal{C}$.*
- $\mathrm{P}^{\mathcal{C}} = \{A \mid (\exists B \in \mathcal{C})\,[A \leq_{\mathrm{T}}^{\mathrm{p}} B]\}$ *halmaz a $\mathcal{C}$* lezárása $\leq_{\mathrm{T}}^{\mathrm{p}}$-visszavezethetőség mellett.



4.9. ábra. A polinomiális idő-, valamint az alsó- és felső-hierarchiák.

- $\mathrm{NP}^C = \{A \mid (\exists B \in C)\,[A \leq_{\mathrm{T}}^{\mathrm{NP}} B]\}$ *halmaz a* C lezárása $\leq_{\mathrm{T}}^{\mathrm{NP}}$-visszavezethetőség mellett.

A 4.14. definíció $\leq_{\mathrm{T}}^{\mathrm{p}}$- és $\leq_{\mathrm{T}}^{\mathrm{NP}}$-visszavezethetőségeinek segítségével bevezetjük a polinomiális idő-hierarchiát, valamint az NP-ben definiált alsó- és felső-hierarchiákat.

4.15. definíció (polinomiális idő-hierarchia). *A* $\mathrm{PH} = \bigcup_{k\geq 0} \Sigma_k^p$ polinomiális idő-hierarchiát *a következőképp definiáljuk:* $\Delta_0^p = \Sigma_0^p = \Pi_0^p = \mathrm{P}$, $\Delta_{i+1}^p = \mathrm{P}^{\Sigma_i^p}$, $\Sigma_{i+1}^p = \mathrm{NP}^{\Sigma_i^p}$ *és* $\Pi_{i+1}^p = \mathrm{co}\Sigma_{i+1}^p$ *minden* $i \geq 0$*-ra.*

Ebből speciális esetként $\Delta_1^p = \mathrm{P}^{\Sigma_0^p} = \mathrm{P}^{\mathrm{P}} = \mathrm{P}$ és $\Sigma_1^p = \mathrm{NP}^{\Sigma_0^p} = \mathrm{NP}^{\mathrm{P}} = \mathrm{NP}$ és $\Pi_1^p = \mathrm{co}\Sigma_1^p = \mathrm{coNP}$ adódik. A következő bizonyítás nélküli tétel megadja ezen hierarchiák néhány tulajdonságát (lásd még a 4-2. feladatnál).

4.16. tétel (Meyer és Stockmeyer). *Minden egyes* $i \geq 1$*-re:*

1. $\Sigma_{i-1}^p \cup \Pi_{i-1}^p \subseteq \Delta_i^p \subseteq \Sigma_i^p \cap \Pi_i^p$.
2. Σ_i^p, Π_i^p, Δ_i^p *és* PH *mind* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-lezártak,* Δ_i^p *továbbá a* $\leq_{\mathrm{T}}^{\mathrm{p}}$*-visszavezetésekre nézve is lezárt.*
3. Σ_i^p *pontosan azt az* A *halmazt tartalmazza, amelyre léteznek* $B \in \mathrm{P}$ *halmaz és egy* p *polinom úgy, hogy minden* $x \in \Sigma^*$*-ra:* $x \in A \iff (\exists^p w_1)\,(\forall^p w_2) \cdots (\mathfrak{Q}^p w_i)\,[(x, w_1, w_2, \ldots, w_i) \in B]$*, ahol a* $\exists^p$ *és* $\forall^p$ *kvantorok polinomiális hosszkorlátúak és* $\mathfrak{Q}^p = \exists^p$ *ha* i *páratlan, vagy* $\mathfrak{Q}^p = \forall^p$ *ha* i *páros.*
4. *Ha* $\Sigma_{i-1}^p = \Sigma_i^p$*, akkor* PH *összeomlik* $\Sigma_{i-1}^p = \Pi_{i-1}^p = \Delta_i^p = \Sigma_i^p = \Pi_i^p = \cdots =$ PH*-ba.*
5. *Ha* $\Sigma_i^p = \Pi_i^p$*, akkor* PH *összeomlik* $\Sigma_i^p = \Pi_i^p = \Delta_{i+1}^p = \Sigma_{i+1}^p = \Pi_{i+1}^p = \cdots =$ PH*-ba.*
6. Σ_i^p*-ben,* Π_i^p*-ben és* Δ_i^p*-ben léteznek* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-teljes problémák. Amennyiben* PH*-ban létezik* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-teljes probléma, akkor* PH *egy véges szintjébe omlik össze, vagyis egy adott* k*-val* $\mathrm{PH} = \Sigma_k^p = \Pi_k^p$.

4.17. definíció (az NP-beli alsó- és felső-hierarchia). *Legyen $k \geq 0$ mellett az*

- *az* NP*-beli* $\mathrm{LH} = \bigcup_{k\geq 0} \mathrm{Low}_k$ ***alsó-hierarchia*** *k-adik szintje* $\mathrm{Low}_k = \{L \in \mathrm{NP} \mid \Sigma_k^{p,L} \subseteq \Sigma_k^p\}$,
- *az* NP*-beli* $\mathrm{HH} = \bigcup_{k\geq 0} \mathrm{High}_k$ ***felső-hierarchia*** *k-adik szintje* $\mathrm{High}_k = \{H \in \mathrm{NP} \mid \Sigma_{k+1}^p \subseteq \Sigma_k^{p,H}\}$.

Egy L halmaz tehát akkor és csak akkor Low_k-beli, ha egy Σ_k^p kiszámításhoz orákulumként használhatatlan, vagyis minden információt, amit L kínálhat, egy Σ_k^p-gép is kiszámíthat, önmagában, orákulum nélkül. Másrészről egy H halmaz High_k-ból olyan gazdag hasznos információkban, hogy egy Σ_k^p-gép kiszámítási erejét egy NP-halmazban maximális értékre növeli. Egy High_k-beli H orákulum segítségével tehát egy Σ_k^p-gép minden Σ_{k+1}^p kiszámítást szimulálhat. Egy Σ_k^p-gép számára tehát H pontosan olyan sokat nyújt, mint egy NP-teljes halmaz. A fent definiált hierarchiák egymásba ágyazott szerkezetét a 4.9. ábra szemlélteti, ahol a világosabb tónusú bonyolultsági osztályok tartalmazzák a sötétebb árnyalatú osztályokat. Egyik ilyen $\mathcal{C} \subseteq \mathcal{D}$ tartalmazásról sem ismert, hogy *valódi*-e, tehát vajon $\mathcal{C} \neq \mathcal{D}$ fennáll-e. A kérdés, hogy $\Sigma_k^p \neq \Sigma_{k+1}^p$ igaz-e, $k = 0$ esetében pontosan a P-versus-NP kérdés. Most (ismét bizonyítás nélkül) felsoroljuk a hierarchiák néhány fontosabb tulajdonságát (lásd még a 4-2. feladatot). Ladner tétele (4.11.) közvetlenül következik a 4.18. tétel utolsó állításának $n = 0$ speciális esetéből.

4.18. tétel (Schöning).

1. $\mathrm{Low}_0 = \mathrm{P}$, $\mathrm{Low}_1 = \mathrm{NP} \cap \mathrm{coNP}$ *és* $\mathrm{NP} \cap \mathrm{coAM} \subseteq \mathrm{Low}_2$.
2. $\mathrm{High}_0 = \{H \mid H \leq_{\mathrm{T}}^{\mathrm{p}}$*-teljes* NP*-ben*$\}$ *és* $\mathrm{High}_1 = \{H \mid H \leq_{\mathrm{sT}}^{\mathrm{NP}}$*-teljes* NP*-ben*$\}$.
3. $\mathrm{Low}_0 \subseteq \mathrm{Low}_1 \subseteq \cdots \subseteq \mathrm{Low}_k \subseteq \cdots \subseteq \mathrm{LH} \subseteq \mathrm{NP}$.
4. $\mathrm{High}_0 \subseteq \mathrm{High}_1 \subseteq \cdots \subseteq \mathrm{High}_k \subseteq \cdots \subseteq \mathrm{HH} \subseteq \mathrm{NP}$.
5. *Minden* $n \geq 0$*-ra* $\mathrm{Low}_n \cap \mathrm{High}_n$ *pontosan akkor nem üres, ha* $\Sigma_n^p = \Sigma_{n+1}^p = \cdots = \mathrm{PH}$.
6. *Minden* $n \geq 0$*-ra* NP*-ben akkor és csak akkor léteznek sem* Low_n*-beli, sem pedig* High_n*-beli halmazok, ha* $\Sigma_n^p \neq \Sigma_{n+1}^p$. NP*-ben pontosan akkor léteznek sem* LH*-beli, sem pedig* HH*-beli halmazok, ha* PH *valódi végtelen halmaz, tehát nem omlik össze egy véges szintjére.*

4.4.2. A gráfizomorfizmus alsó-hierarchiabeli

Hozzákezdve az említett eredmény bizonyításához belátjuk, hogy `GI` Low_2-beli, ami egy erős fegyvertény `GI` NP-teljessége ellen. Ha `GI` NP-teljes lenne, akkor $\mathrm{High}_0 \subseteq \mathrm{High}_2$-beli is lenne, mivel a 4.18. tétel alapján High_0 pontosan NP $\leq_{\mathrm{T}}^{\mathrm{p}}$-teljes részét tartalmazza, így tehát az NP $\leq_{\mathrm{m}}^{\mathrm{p}}$-teljes részét is. Szintén a 4.18. tételből következik viszont, hogy $\mathrm{Low}_2 \cap \mathrm{High}_2$ akkor és csak akkor nem üres, ha PH összeomlik Σ_2^p-be, ami viszont nagyon valószínűtlennek tűnik.

A tétel bizonyításához szükségünk van még az úgynevezett hasítás lemmára, lásd 4.20. lemma. A hasítás egy dinamikus adatrendező eljárás. Minden adategységhez egy kulcs tartozik, ami azt egyértelműen azonosítja. A kulcsok U halmaza meglehetősen nagy, a következőkben mint *univerzum* tekintjük, mivel a ténylegesen felhasznált kulcsok $V \subseteq U$

halmaza jóval kisebb lehet. A célunk az, hogy U elemeit egy *hasító függvény* segítségével bejegyezzük egy $T = \{0, 1, \ldots, k-1\}$ *hasító táblázatba*, ahol több U-beli kulcshoz is tartozhat ugyanaz a T-beli cím. Lehetőség szerint különböző V-beli kulcsoknak különböző T-beli címet kell kapniuk, tehát elkerülendők a ténylegesen használt kulcsok *ütközései*. Másképp fogalmazva a hasító függvény lehetőség szerint legyen injektív V-re.

A sok különböző ismert hasító eljárás változat közül számunkra leginkább az *univerzális hasítás* az érdekes. Ennél az alapötlet abból áll, hogy egy alkalmasan választott hasító függvény családból *véletlenszerűen* választunk egy hasító függvényt. Ez a hasítás módszer univerzális olyan értelemben, hogy már nem függ egy meghatározott V halmaztól, hanem *minden* megfelelően kicsi V halmazon nagy valószínűséggel elkerüli az ütközéseket. A valószínűség itt a hasító függvény véletlenszerű választására vonatkozik. A következőkben a kulcsokat mint $\Sigma = \{0, 1\}$ ábécé fölött kódolt szavakat kezeljük, és Σ^n-el jelöljük a Σ^*-beli, n hosszú szavak halmazát.

4.19. definíció (hasítás). *Legyen $\Sigma = \{0, 1\}$, valamint legyenek m és t természetes számok, $t > m$. Egy $h : \Sigma^t \to \Sigma^m$* hasító függvény *lineáris leképezés, ami egy $(t \times m)$ Boole-mátrixszal adott, mégpedig $B_h = (b_{i,j})_{i,j}$-vel, ahol $b_{i,j} \in \{0, 1\}$. Minden $x \in \Sigma^t$ és $1 \leq j \leq m$ mellett az $y = h(x) \in \Sigma^m$ j-edik bitje mint $y_j = (b_{1,j} \wedge x_1) \oplus (b_{2,j} \wedge x_2) \oplus \cdots \oplus (b_{t,j} \wedge x_t)$ adódik, ahol $\oplus$ a logikai paritásműveletet jelenti, tehát $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 1 \iff |\{i \mid a_i = 1\}| \equiv 1 \bmod 2$.*

Legyen $\mathcal{H}_{t,m} = \{h : \Sigma^t \to \Sigma^m \mid B_h$ egy $(t \times m)$ Boole-mátrix$\}$ hasító függvények egy családja, t és m paraméterekkel. A $\mathcal{H}_{t,m}$-re egyenletes eloszlást tételezünk fel: $\mathcal{H}_{t,m}$-ből egy h hasító függvényt emelünk ki, amelynél a B_h-beli $b_{i,j}$-ket függetlenül és egyenletes eloszlás mellett választjuk.

Legyen $V \subseteq \Sigma^t$. A $\mathcal{H}_{t,m}$ egy $\widehat{\mathcal{H}}$ részcsaládjára akkor lép fel ütközés *V-n, ha*

$$(\exists v \in V)\,(\forall h \in \widehat{\mathcal{H}})\,(\exists x \in V)\,[v \neq x \wedge h(v) = h(x)]\ .$$

Egyéb esetekben $\widehat{\mathcal{H}}$ a V-n ütközésmentes.

Egy V fölötti ütközés azt jelenti tehát, hogy megsérül egy tetszőlegesen választott $\widehat{\mathcal{H}}$ családbeli hasító függvény injektivitása V-n. A következő lemma kimondja, hogy minden kellően kis V halmazon $\mathcal{H}_{t,m}$ egy véletlenszerűen választott részcsaládja ütközésmentes, ha pedig V túl nagy, az ütközés elkerülhetetlen. A 4.20. lemma bizonyításától eltekintünk.

4.20. lemma (hasítás lemma). *Legyenek $t, m \in \mathbb{N}$ paraméterek, $V \subseteq \Sigma^t$ és $\widehat{\mathcal{H}} = (h_1, h_2, \ldots, h_{m+1})$ egy egyenletes eloszlás mellett véletlenül választott hasító függvény család $\mathcal{H}_{t,m}$-ből. Legyen*

$$K(V) = \{\widehat{\mathcal{H}} \mid (\exists v \in V)\,(\forall h \in \widehat{\mathcal{H}})\,(\exists x \in V)\,[v \neq x \wedge h(v) = h(x)]\}$$

$\widehat{\mathcal{H}}$-ban egy ütközés fellépésének eseménye V-n. Ekkor

1. *ha $|V| \leq 2^{m-1}$, akkor $K(V)$ legfeljebb $1/4$ valószínűséggel következik be,*
2. *ha $|V| > (m+1)2^m$, akkor $K(V)$ 1 valószínűséggel következik be.*

A 3.5. alfejezetben definiáltuk az Artúr-Merlin hierarchiát és említettük, hogy ez a hierarchia a második szintjére omlik össze. Számunkra most a coAM osztály érdekes, lásd a 3.16. definíciót.

4.21. tétel (Schöning). *A* GI *probléma* Low_2*-beli.*

Bizonyítás. A 4.18. tétel alapján minden coAM-beli NP-halmaz Low_2-beli. GI Low_2-beliségének belátásához elégséges tehát azt megmutatni, hogy GI coAM-beli. Legyenek G és H n csúccsal rendelkező gráfok. A hasítás lemmát szeretnénk alkalmazni. Ígéretesnek tűnik, hogy a 3.11. lemmában definiált

$$A(G,H) \quad = \quad \{(F,\varphi) \mid F \cong G \text{ és } \varphi \in \mathrm{Aut}(F)\} \cup \{(F,\varphi) \mid F \cong H \text{ és } \varphi \in \mathrm{Aut}(F)\}$$

halmaz a 4.20. lemma V-jének szerepét vegye fel. A 3.11. lemma szerint $|A(G,H)| = n!$ amennyiben $G \cong H$ és $|A(G,H)| = 2n!$, ha $G \ncong H$. Ahhoz, hogy a GI-t megoldó coAM-gép polinomiális időben dolgozzon, a hasítás lemma t és m paraméterei n-ed fokú polinomok kell legyenek. A lemma alkalmazásához az $m = m(n)$ polinomot úgy kellene választanunk, hogy

$$n! \leq 2^{m-1} < (m+1)2^m < 2n! \tag{4.4}$$

fennálljon, mivel ekkor lesz a $V = A(G,H)$ halmaz elég nagy ahhoz, hogy elegendően nagy valószínűséggel meg tudjunk különböztetni két izomorf G és H gráfot két nem izomorf gráftól. Sajnos nem létezik olyan m polinom, hogy a (4.4) egyenlőtlenség fennálljon, ehelyett választunk egy másik V halmazt, amivel a felső és alsó korlátok közötti rést elég naggyá tudjuk tenni.

Legyen $V = A(G,H)^n = \underbrace{A(G,H) \times A(G,H) \times \cdots \times A(G,H)}_{n\text{-szer}}$. Ezzel (4.4) a következőképp alakul:

$$(n!)^n \leq 2^{m-1} < (m+1)2^m < (2n!)^n \,, \tag{4.5}$$

ahol az új egyenlőtlenség kielégíthető az $m = m(n) = 1 + \lceil n \log n! \rceil$ választással.

Készítsünk egy M coAM-gépet GI-hez a következőképp. Az n csúcsú G és H gráfok bemenete mellett M mindenekelőtt kiszámítja az m paramétert. A $V = A(G,H)^n$ halmaz n darab (F,φ) formájú párost tartalmaz, ahol F egy n csúcsú gráf és $\varphi \in \mathrm{Aut}(F)$. Tekintsük a V elemeit mint $t(n)$ hosszú, $\Sigma = \{0,1\}$ ábécé fölötti szavakat, ahol t egy alkalmasan választott polinom. Ekkor M egyenletes eloszlás mellett véletlenszerűen választ egy $\widehat{\mathcal{H}} = (h_1, h_2, \ldots, h_{m+1})$ hasító függvény családot $\mathcal{H}_{t,m}$-ből, ami megfelel Artúr lépésének. Minden $h_i \in \widehat{\mathcal{H}}$ hasító függvényt egy $(t \times m)$ Boole-mátrixszal reprezentálunk, ezért az $m+1$ darab, $\widehat{\mathcal{H}}$-beli h_i hasító függvény ábrázolható mint $p(n)$ hosszú $z_{\widehat{\mathcal{H}}} \in \Sigma^*$ szó, ahol p egy alkalmas polinom. Módosítva a hasítás lemma $K(V)$ ütközés predikátumát

$$B = \{(G,H,z_{\widehat{\mathcal{H}}}) \mid (\exists v \in V)\,(\forall i : 1 \leq i \leq m+1)\,(\exists x \in V)\,[v \neq x \wedge h_i(v) = h_i(x)]\} \,.$$

A B-ben elforduló $\forall$ kvantor csak polinomiálisan sok i-t határoz meg, ezért determinisztikus módon polinomiális időben kiértékelhető, valamint B mindkét $\exists$ kvantora összefogható *egy* darab polinomiális hosszra korlátozott $\exists$ kvantorrá. A 4.16. tétel alapján tehát B egy Σ_1^p = NP-beli halmaz. Legyen N egy NPTM B-hez. A $z_{\widehat{\mathcal{H}}}$ készülő szóhoz – amit $m+1$ függetlenül és egyenletes eloszlás mellett választott hasító függvény reprezentál – M szimulálja az $N(G,H,z_{\widehat{\mathcal{H}}})$ kiszámítását, ami megfelel Merlin lépésének. Az M pontosan akkor fogadja el a (G,H) bemenetét, ha $N(G,H,z_{\widehat{\mathcal{H}}})$ elfogad.

Becsüljük meg (a $z_{\widehat{\mathcal{H}}}$-ban kódolt hasító függvény véletlenszerű választása mellett) annak valószínűségét, hogy M elfogadja (G,H) bemenetét. Ha G és H izomorfak, akkor a 3.11.

lemma alapján $|A(G,H)| = n!$. A (4.5) egyenlőtlenségből $|V| = (n!)^n \leq 2^{m-1}$ következik. A 4.20. lemma alapján annak valószínűsége, hogy $(G, H, z_{\widehat{H}})$ B-beli, következésképp $M(G,H)$ elfogad, legfeljebb 1/4. Ha ezzel szemben G és H nem izomorfak, akkor a 3.11. lemma szerint $|A(G,H)| = 2n!$, a (4.5) egyenlőtlenségből pedig $|V| = (2n!)^n > (m+1)2^m$ adódik. A 4.20. lemma alapján ekkor annak valószínűsége, hogy $(G, H, z_{\widehat{H}})$ B-beli, tehát $M(G,H)$ elfogad, közel 1. Ebből következik tehát, hogy GI coAM-beli. ■

4.4.3. A gráfizomorfizmus SPP-beli

A 3.3.1. alfejezet 3.14. definíciójában bevezettük az RP valószínűségi osztályt. A következő alfejezetben két további, itt definiált valószínűségi osztály játszik szerepet: pp és SPP, melyek a *Probabilistic Polynomialtime* és a *Stoic Probabilistic Polynomialtime* angol kifejezések rövidítései.

4.22. definíció (PP és SPP). *A* pp ***osztály*** *azokat az A problémákat foglalja magában, melyekhez létezik egy M NPTM úgy, hogy minden x bemenetre: ha $x \in A$, akkor $M(x)$ 1/2-nél nem kisebb valószínűség mellett elfogad, és ha $x \notin A$, akkor $M(x)$ valamely 1/2-nél kisebb valószínűséggel fogad el.*

Jelölje $\mathrm{acc}_M(x)$ *az $M(x)$ elfogadó útjainak számát egy M NPTM-nél x bemenet mellett, valamint* $\mathrm{rej}_M(x)$ *az $M(x)$ visszautasító útjainak számát. Legyen továbbá* $\mathrm{gap}_M(x) = \mathrm{acc}_M(x) - \mathrm{rej}_M(x)$.

Az SPP *osztály azokból az A problémákból áll, melyekre létezik olyan M NPTM, hogy minden x-re* $(x \in A \implies \mathrm{gap}_M(x) = 1)$ *és* $(x \notin A \implies \mathrm{gap}_M(x) = 0)$ *teljesül.*

Egy SPP-gép „sztoikus” tehát abban az értelemben, hogy a „rése” – tehát az elfogadó és a visszautasító utak különbsége – mindig csak két értéket vesz fel az exponenciális számú lehetséges érték közül. A pp-vel ellentétben az SPP így egy úgynevezett *ígéret osztály* („promise class”), mivel egy M SPP-gép azt „ígéri”, hogy minden x-re $\mathrm{gap}_M(x) \in \{0,1\}$ teljesül (lásd még a 4.4-4. gyakorlatot).

Az alsóság fogalmát tetszőleges relativizálható C bonyolultsági osztályra definiálhatjuk: egy A halmaz akkor és csak akkor C-*alsó*, ha $C^A = C$ teljesül. Speciálisan minden k-ra a 4.17. definíció alsóság-hierarchiájának Low_k szintje pontosan azokból az NP-halmazokból áll, melyek Σ_k^p-alsó tulajdonságúak. Minden, a fent definiált SPP osztályba tartozó halmaz pp-alsó. A következő tételben ezek mellett bizonyítás nélkül összefoglalunk néhány hasonló tulajdonságot.

4.23. tétel.

1. *Az* SPP *osztály* pp-*alsó, tehát* $\mathrm{pp}^{\mathrm{SPP}} = \mathrm{pp}$.
2. *Az* SPP *önmagával alsó (angolul „self-low”), tehát* $\mathrm{SPP}^{\mathrm{SPP}} = \mathrm{SPP}$.
3. *Legyen $A \in$ NP egy N NPTM-el egyeztetve és $L \in \mathrm{SPP}^A$ egy M NPOTM-mal egyeztetve úgy, hogy az* M-A(x) *minden x bemenetre csak olyan q kérdést tesz fel, melyre* $\mathrm{acc}_N(q) \leq 1$ *fennáll. Ekkor L* SPP*-beli.*
4. *Legyen $A \in$ NP egy N NPTM-mel egyeztetve és $f \in \mathrm{FP}^A$ egy M DPOTM-ma egyeztetve úgy, hogy* M-A(x) *minden x bemenet mellett csak olyan q kérdést tesz fel, melyre* $\mathrm{acc}_N(q) \leq 1$. *Ekkor f* $\mathrm{FP}^{\mathrm{SPP}}$*-beli.*

A következő tétel kimondja, hogy egy jobb oldali co-halmaz lexikografikusan legkisebb permutációja hatékonyan kiszámítható. A $\mathfrak{S}_n$ lexikografikus rendezését a 4.5. példában definiáljuk.

4.24. tétel. *Legyen $\mathfrak{G} \leq \mathfrak{S}_n$ egy permutációcsoport $\mathfrak{G} = \langle G \rangle$ mellett és $\pi \in \mathfrak{S}_n$ egy permutáció. Ekkor létezik olyan algoritmus, amely a (G, π) bemenetre polinomiális időben meghatározza a $\mathfrak{S}_n$-beli $\mathfrak{G}$ jobb oldali co-halmazát, $\mathfrak{G}\pi$-t.*

Bizonyítás. A LERC algoritmus a lexikografikusan legkisebb permutáció kiszámítását végzi a $\mathfrak{S}_n$-beli $\mathfrak{G}$ jobb oldali co-halmazában $\mathfrak{G}\pi$-ben, ahol a $\mathfrak{G}$ permutációcsoport egy G generátorfüggvényen keresztül adott, lásd a 3.1.3. pont 3.6. definícióját.

A 3.7. tétel segítségével polinomiális időben számítható az $\mathbf{id} = \mathfrak{G}^{(n)} \leq \mathfrak{G}^{(n-1)} \leq \cdots \leq \mathfrak{G}^{(1)} \leq \mathfrak{G}^{(0)} = \mathfrak{G}$, $\mathfrak{G}$ stabilizátor lánc. Pontosabban fogalmazva minden egyes i-re meghatározzuk $\mathfrak{G}^{(i)}$ jobb oldali T_i reprezentáns rendszerét, ahol $1 \leq i \leq n$, és ebből képezzük $\mathfrak{G}$ egy erősebb $S = \bigcup_{i=1}^{n-1} T_i$ generátorát. Mivel $\varphi_0 = \pi$ és $\mathfrak{G}^{(n-1)} = \mathfrak{G}^{(n)} = \mathbf{id}$, ezért a bizonyításhoz elég a LERC algoritmus helyességét megmutatni, tehát azt, hogy $0 \leq i \leq n-1$ esetén minden i-re a $\mathfrak{G}^{(i+1)}\varphi_{i+1}$-beli $\mathfrak{G}^{(i)}\varphi_i$ legkisebb permutációját kapjuk.

LERC(G, π)

1 számítsuk ki a $\mathfrak{G}$-beli stabilizátorok $\mathfrak{G}^{(n)} \leq \mathfrak{G}^{(n-1)} \leq \cdots \leq \mathfrak{G}^{(1)} \leq \mathfrak{G}^{(0)}$ láncát
2 $\quad\varphi_0 \leftarrow \pi$
3 $\quad$**for** $i \leftarrow 0$ **to** $n-1$
4 $\quad\quad$**do** $x \leftarrow i+1$
5 $\quad\quad\quad$számítsuk ki a $\mathfrak{G}^{(i)}(x)$ pályán az y elemet, melyre $\varphi_i(y)$ minimális
6 $\quad\quad\quad$határozzunk meg egy τ_i permutációt $\mathfrak{G}^{(i)}$-ben $\tau_i(x) = y$ mellett
7 $\quad\quad\quad\varphi_{i+1} \leftarrow \tau_i\varphi_i$
8 $\quad$**return** φ_n

Ebből indukcióval következik, hogy $\mathfrak{G}^{(n)}\varphi_n = \{\varphi_n\}$ a $\mathfrak{G}\pi = \mathfrak{G}^{(0)}\varphi_0$ lexikografikusan legkisebb permutációja. Következésképp tehát a LERC algoritmus valóban $\mathfrak{G}\pi$ lexikografikusan legkisebb φ_n permutációját adja vissza.

A fenti állítás bizonyításához jelöljük egy $\mathfrak{H} \leq \mathfrak{S}_n$ permutációcsoportbeli $x \in [n]$ elemek pályáját $\mathfrak{H}(x)$-el. Legyen τ_i az a $\mathfrak{G}^{(i)}$-beli permutáció, amely a $i+1$-et a $\mathfrak{G}^{(i)}(i+1)$ pályában lévő y-ra képzi le, és amelyre $\varphi_i(y) = x$ a $\{\varphi_i(z) \mid z \in \mathfrak{G}^{(i)}(i+1)\}$ halmaz minimális eleme. A 3.7. tétel alapján a $\mathfrak{G}^{(i)}(i+1)$ pálya polinomiális időben kiszámítható, és mivel $\mathfrak{G}^{(i)}(i+1)$ legfeljebb $n-i$ elemet tartalmaz, ezért y hatékonyan meghatározható. Az algoritmus definíciója alapján fennáll $\varphi_{i+1} = \tau_i\varphi_i$. Mivel $\mathfrak{G}^{(i)}$ minden permutációja minden $[i]$-beli elemet önmagára képez le és mivel $\tau_i \in \mathfrak{G}^{(i)}$, ezért minden j-re, amelyre $1 \leq j \leq i$, minden $\tau \in \mathfrak{G}^{(i)}$-re és minden $\sigma \in \mathfrak{G}^{(i+1)}$-re:

$$(\sigma\varphi_{i+1})(j) = \varphi_{i+1}(j) = (\tau_i\varphi_i)(j) = \varphi_i(j) = (\tau\varphi_i)(j).$$

Ebből speciálisan következik, hogy minden $\mathfrak{G}^{(i)}\varphi_i$-beli μ lexikografikusan legkisebb permutációra fennáll, hogy minden $\mathfrak{G}^{(i+1)}\varphi_{i+1}$-beli permutációnak egyeznie kell μ-vel az első i

elemre.

Ezen kívül minden $\sigma \in \mathfrak{G}^{(i+1)}$-re és a fent definiált $x = \varphi_i(y)$ elemre fennáll

$$(\sigma\varphi_{i+1})(i+1) = \varphi_{i+1}(i+1) = (\tau_i\varphi_i)(i+1) = x.$$

Természetesen $\mathfrak{G}^{(i+1)}\varphi_{i+1} = \{\varphi \in \mathfrak{G}^{(i)}\varphi_i \mid \varphi(i+1) = x\}$. Az állítás ezután abból a tényből következik, hogy a $\mathfrak{G}^{(i)}\varphi_i$-beli lexikografikusan legkisebb μ permutációra fennáll $\mu(i+1) = x$.

Ezzel megmutattuk, hogy a LERC algoritmus hatékonyan és helyesen működik. ■

A 4.24. tétel könnyen kibővíthető a 4.25. következménnyé (lásd a 4-3. feladatot). Ehhez kapcsolódóan bebizonyítjuk az alfejezet fő tételét, a 4.26. tételt.

4.25. következmény. *Legyen $\mathfrak{G} \leq \mathfrak{S}_n$ egy permutációcsoport ahol $\mathfrak{G} = \langle G \rangle$, valamint legyen π és ψ két permutáció $\mathfrak{S}_n$-ben. Ekkor létezik olyan algoritmus, amely (G, π, ψ) bemenet mellett polinomiális időben meghatározza $\psi\mathfrak{G}\pi$ lexikografikusan legkisebb permutációját.*

4.26. tétel (Arvind és Kurur). *A* `GI` *probléma* SPP*-beli.*

Bizonyítás. Az `AUTO` probléma a következőképp definiált: számítsuk ki egy adott G gráfhoz az $\mathrm{Aut}(G)$ automorfizmus-csoportot (a fogalmak értelmezéséhez lásd a 3.6. definíciót és az azt követő bekezdést, valamint a 3.8. definíciót). Mathon eredményei alapján alapján az `AUTO` és a `GI` problémák Turing-ekvivalensek, tehát $\texttt{AUTO} \in \mathrm{FP}^{\texttt{GI}}$ és $\texttt{GI} \in \mathrm{P}^{\texttt{AUTO}}$, ezért elég `AUTO` $\mathrm{FP}^{\mathrm{SPP}}$-beliségét megmutatni. Ezután SPP 4.23. tételbeli önmagával alsó („self-low") tulajdonságából következik, hogy $\texttt{GI}\ \mathrm{P}^{\texttt{AUTO}} \subseteq \mathrm{SPP}^{\mathrm{SPP}} \subseteq$ SPP-beli, amivel a tételt be is láttuk.

Célunk tehát, hogy találjunk `AUTO`-hoz egy $\mathrm{FP}^{\mathrm{SPP}}$-algoritmust. Ennek az algoritmusnak egy adott G gráf esetén egy $S = \bigcup_{i=1}^{n-1} T_i$ erős generátort kell kiszámítania $\mathrm{Aut}(G)$-hez, ahol

$$\mathbf{id} = \mathrm{Aut}(G)^{(n)} \leq \mathrm{Aut}(G)^{(n-1)} \leq \cdots \leq \mathrm{Aut}(G)^{(1)} \leq \mathrm{Aut}(G)^{(0)} = \mathrm{Aut}(G)$$

az $\mathrm{Aut}(G)$ stabilizátorainak lánca, T_i $(1 \leq i \leq n)$ pedig $\mathrm{Aut}(G)^{(i)}$ egy teljes jobb oldali reprezentáns rendszere $\mathrm{Aut}(G)^{(i-1)}$-ben. A triviális $\mathrm{Aut}(G)^{(n)} = \mathbf{id}$ esettel kezdve lépésről lépésre építünk fel egy erős generátort $\mathrm{Aut}(G)^{(i)}$-hez, csökkenő i mellett, végül tehát kapunk egy erős generátort $\mathrm{Aut}(G)^{(0)} = \mathrm{Aut}(G)$-hez. Tegyük fel tehát, hogy már találtunk egy $S_i = \bigcup_{j=i}^{n-1} T_j$ erős generátort $\mathrm{Aut}(G)^{(0)} = \mathrm{Aut}(G)$-hez. Most leírjuk hogyan határozza meg az $\mathrm{FP}^{\mathrm{SPP}}$-algoritmus $\mathrm{Aut}(G)^{(i)}$ egy teljes jobb oldali reprezentáns rendszerét $\mathrm{Aut}(G)^{(i-1)}$-ben, T_{i-1}-et. Ehhez definiáljuk az

$$A = \left\{ (G, S, i, j, \pi) \;\middle|\; \begin{array}{l} S \subseteq \mathrm{Aut}(G) \text{ és } \langle S \rangle \text{ egy pontonkénti stabilizátora } [i]\text{-nek} \\ \mathrm{Aut}(G)\text{-ben, } \pi \text{ egy pontonként } [i-1] \text{ által stabilizált} \\ \text{parciális permutáció és } \pi(i) = j, \text{ valamint létezik egy} \\ \tau \in \mathrm{Aut}(G)^{(i-1)}, \text{ ahol } \tau(i) = j \text{ és } \mathrm{LERC}(S, \tau) \text{ a } \pi \text{ bővítése} \end{array} \right\}$$

halmazt. A 4.24. tétel alapján a $\langle S \rangle\tau$ jobb oldali co-halmazhoz tartozó $\mathrm{LERC}(S, \tau)$ lexikografikusan legkisebb permutáció polinomiális időben kiszámítható a LERC algoritmus segítségével. A π parciális permutáció a (G, S, i, j, π) bemenet része, mivel az A halmazt mint orákulumot szeretnénk használni a lexikografikusan legkisebb $\tau \in \mathrm{Aut}(G)^{(i-1)}$, $\tau(i) = j$ melletti permutáció prefixkereséses meghatározásakor (vessük ezt össze a 4.5. példa N-Pre-Iso

algoritmusával).

Az N algoritmus egy NPTM A orákulumhoz, tehát A NP-beli. Eldöntendő, hogy ha $\tau(i) = j$, akkor fennáll-e $\langle S\rangle\tau$ jobb oldali co-halmaz minden σ permutációjára $\sigma(i) = j$.

$N(G, S, i, j, \pi)$

1 ellenőrizzük, hogy $S \subseteq \mathrm{Aut}(G)^{(i)}$
2 találjunk ki nemdeterminisztikusan egy $\tau \in \mathfrak{S}_n$ permutációt ▷ G n csúcsú.
3 **if** $\tau \in \mathrm{Aut}(G)^{(i-1)}$ és $\tau(i) = j$ és τ a π bővítése és $\tau = \mathrm{LERC}(S, \tau)$
4 **then** elfogadás és megállás
5 **else** elutasítás és megállás

Most megmutatjuk, hogy N elfogadó útjainak száma (G, S, i, j, π) bemenet mellett vagy 0 vagy 1, amennyiben $\langle S\rangle = \mathrm{Aut}(G)^{(i)}$, valamint általában fennáll

$$\mathrm{acc}_N(G, S, i, j, \pi) \in \left\{0, |\mathrm{Aut}(G)^{(i)}|/|\langle S\rangle|\right\} .$$

Először egy pszeudokódot adunk meg.

M-A(G)

1 $T_i \leftarrow \{\}$ minden i-re, ahol $0 \le i \le n-2$ ▷ G n csúcsú.
2 ▷ T_i az $\mathrm{Aut}(G)^{(i+1)}$ egy teljes jobb oldali reprezentáns rendszere lesz $\mathrm{Aut}(G)^{(i)}$-ben.
3 $S_i \leftarrow \emptyset$ minden i-re, ahol $0 \le i \le n-2$
4 $S_{n-1} \leftarrow \{\mathrm{id}\}$ ▷ S_i egy erős generátor lesz $\mathrm{Aut}(G)^{(i)}$-hez.
5 **for** $i \leftarrow n-1$ **downto** 1
6 **do** ▷ Az i-edik iteráció kezdetén S_i-t már megtaláltuk, most S_{i-1}-et számítjuk ki.
7 legyen $\pi : [i-1] \to [n]$ parciális permutáció $\pi(a) = a$-val minden $a \in [i-1]$-re
8 ▷ $i = 1$ esetén π a definiálatlan parciális permutáció.
9 **for** $j \leftarrow i+1$ **to** n
10 **do** $\hat{\pi} \leftarrow \pi j$ ▷ Tehát $\hat{\pi}$ bővíti π-t az (i, j) párral $\hat{\pi}(i) = j$ mellett.
11 **if** $(G, S_i, i, j, \hat{\pi}) \in A$
12 **then** ▷ A prefixkeresés előállítja a legkisebb $\mathrm{Aut}(G)^{(i-1)}$-beli,
i-ről j-re képező permutációt.
13 **for** $k \leftarrow i+1$ **to** n
14 **do** keressük meg az ℓ elemet, $\hat{\pi}$ leképezésein kívül,
$(G, S_i, i, j, \hat{\pi}\ell) \in A$ mellett
15 $\hat{\pi} \leftarrow \hat{\pi}\ell$
16 ▷ $\hat{\pi}$ most egy teljes $\mathfrak{S}_n$-beli permutáció.
17 $T_{i-1} \leftarrow T_{i-1} \cup \hat{\pi}$
18 ▷ T_{i-1} most az $\mathrm{Aut}(G)^{(i)}$ egy teljes jobb oldali reprezentáns
rendszere $\mathrm{Aut}(G)^{(i-1)}$-ben.
19 $S_{i-1} \leftarrow S_i \cup T_{i-1}$
20 **return** S_0 ▷ S_0 egy erős generátor $\mathrm{Aut}(G) = \mathrm{Aut}(G)^{(0)}$-hoz.

Tegyük fel, hogy (G, S, i, j, π) A-beli. Ha $\tau(i) = j$ egy $\tau \in \mathrm{Aut}(G)^{(i-1)}$ mellett és $j > i$, akkor az $\langle S \rangle \tau$ jobb oldali co-halmaz pontosan azokból az $\mathrm{Aut}(G)^{(i-1)}$-beli permutációkból áll, melyek i-t j-re képezik le. Következésképp $N(G, S, i, j, \pi)$ egyetlen elfogadó útja megfelel az egyértelműen meghatározott lexikografikusan legkisebb $\tau = \mathrm{LERC}(S, \tau)$ permutációnak. Ha viszont $\langle S \rangle$ az $\mathrm{Aut}(G)^{(i)}$ egy valódi alcsoportja, akkor $\mathrm{Aut}(G)^{(i)}\tau$ ábrázolható az $\langle S \rangle$ $k = |\mathrm{Aut}(G)^{(i)}|/|\langle S \rangle|$ darab diszjunkt jobb oldali co-halmazának egyesítéseként. Általában tehát $N(G, S, i, j, \pi)$ akkor rendelkezik k elfogadó úttal, ha (G, S, i, j, π) A-beli, egyébként pedig nincs elfogadó útja.

Az M-A egy FP^A-algoritmus AUTO megoldásához. Itt az M DPOTM csak olyan $q = (G, S_i, i, j, \pi)$ kérdéseket tesz fel az A orákulumnak, melyekre $\langle S_i \rangle = \mathrm{Aut}(G)^{(i)}$, következésképp minden valóban feltett q kérdésre fennáll $\mathrm{acc}_N(q) \leq 1$. A 4.23. tétel 4. része alapján ebből $\mathtt{AUTO} \in \mathrm{FP}^{\mathrm{SPP}}$ következik.

Az állítást – miszerint M-A(G) S_0 kimenete egy erős generátor $\mathrm{Aut}(G) = \mathrm{Aut}(G)^{(0)}$-hoz – n szerinti teljes indukcióval mutatjuk meg. Az indukció kezdete az $n-1$, $S_{n-1} = \{\mathrm{id}\}$ pedig természetesen $\mathrm{Aut}(G)^{(n-1)} = \mathbf{id}$-t állítja elő. Az indukciós lépéshez feltesszük, hogy az i-edik iteráció kezdetekor már rendelkezésre áll egy S_i erős generátor $\mathrm{Aut}(G)^{(i)}$-hez. Megmutatjuk, hogy az i-edik iteráció végére az $S_{i-1} = S_i \cup T_{i-1}$ halmaz egy erős generátor $\mathrm{Aut}(G)^{(i-1)}$-hez. Minden j-hez, ahol $i + 1 \leq j \leq n$ a „$(G, S_i, i, j, \hat{\pi}) \in A$?” kérdéssel megvizsgáljuk, hogy $\mathrm{Aut}(G)^{(i-1)}$-ben létezik-e olyan permutáció, amely i-t j-re képezi le. A rákövetkező perfixkeresés az A orákulumhoz intézett megfelelő kérdésekkel előállítja a lexikografikusan legkisebb $\hat{\pi}$ permutációt $\mathrm{Aut}(G)^{(i-1)}$-ben $\hat{\pi}(i) = j$ mellett. Mint fent megállapítottuk ekkor A-hoz csak $\mathrm{acc}_N(q) \leq 1$-nek megfelelő q kérdéseket intézünk, mivel S_i egy erős generátor $\mathrm{Aut}(G)^{(i)}$-hez, tehát $\langle S_i \rangle = \mathrm{Aut}(G)^{(i)}$. A permutáció előállítása után az i-edik iteráció végén T_{i-1} $\mathrm{Aut}(G)^{(i)}$ egy teljes jobb oldali átlója $\mathrm{Aut}(G)^{(i-1)}$-ben, következésképp $S_{i-1} = S_i \cup T_{i-1}$ egy erős generátor $\mathrm{Aut}(G)^{(i-1)}$-hez. Végül n iteráció után megtalálunk egy S_0 erős generátort $\mathrm{Aut}(G) = \mathrm{Aut}(G)^{(0)}$-hoz. ■

A 4.23. tétel első két állításából közvetlenül adódik a 4.27. következmény.

4.27. következmény. GI *alsó* SPP*-re és* pp*-re, tehát* $\mathrm{SPP}^{\mathtt{GI}} = \mathrm{SPP}$ *és* $\mathrm{pp}^{\mathtt{GI}} = \mathrm{pp}$.

Gyakorlatok

4.4-1. A 4.14. definíció alapján fennáll $A \leq_{\mathrm{T}}^{\mathrm{p}} B \iff A \in \mathrm{P}^B$. Mutassuk meg, hogy $A \leq_{\mathrm{T}}^{\mathrm{p}} B \iff \mathrm{P}^A \subseteq \mathrm{P}^B$.

4.4-2. Mutassuk meg, hogy a 4.5. példában definiált Pre-Iso halmaz NP-beli, valamint hogy az N-Pre-Iso algoritmus N gépe egy DPOTM, tehát polinomiális időben dolgozik.

4.4-3. Határozzuk meg a 4.6. példában definiált $\widehat{G}$ és $\widehat{H}$ gráfokhoz az $\mathrm{Iso}(\widehat{G}, \widehat{H})$ izomorfizmuscsoportot.

4.4-4. A következő osztályok közül melyek „ígéret”-osztályok: NP és coNP, RP és coRP, AM és coAM, MA és coMA? Tartozhatnak-e az „ígéret”-osztályokhoz teljes problémák? A válaszokat indokoljuk.

Feladatok

4-1. Erős NPOTM
Egy *erős NPOTM* egy három végállapot típussal rendelkező NPOTM, vagyis a végállapotok F halmaza F_a, F_r és $F_?$ halmazokra bontható. Ekkor teljesül, hogy amennyiben $x \in A$, akkor $M^B(x)$ rendelkezik egy olyan úttal, ami egy F_a-beli állapotban végződik, és nem rendelkezik olyan úttal, amely F_r-beli állapotban végződik. Ha azonban $x \notin A$, akkor $M^B(x)$ rendelkezik egy olyan úttal, ami egy F_r-beli állapottal végződik, és nem rendelkezik olyan úttal, ami F_a-beli állapottal végződik. Az $M^B(x)$ mindkét esetben végződhet bizonyos utakon $F_?$-beli állapotban, ami megfelel a „nem tudom” válasznak. Az erős NPOTM tehát olyan gép, ami sosem hazudik. Bizonyítsuk a következő két állítást:

- $A \leq_{\mathrm{sT}}^{\mathrm{NP}} B \iff$ létezik M erős NPOTM, mellyel $A = L(M^B)$.
- $A \leq_{\mathrm{sT}}^{\mathrm{NP}} B \iff \mathrm{NP}^A \subseteq \mathrm{NP}^B$.

Útmutatás. Általánosítsuk a 4.4-1. gyakorlatot.

4-2. Bizonyítás
Bizonyítsuk a 4.16. és 4.18. tételek állításait.

4-3. Bizonyítás módosítása
Módosítsuk a 4.24. tétel bizonyítását úgy, hogy abból adódjon a 4.25. következmény.

Megjegyzések a fejezethez

A 3. és 4. fejezetek egyes részei a [400] könyvön alapulnak. Amit itt csak erősen tömörített formában tudtunk bemutatni, ott megtalálható átfogó és minden technikai részletre kiterjedő változatban, terjedelmes példákkal, nagyobb számokkal, szebb és több ábrával, pontosabb magyarázatokkal és kidolgozottabb bizonyításokkal. Ilyen módon megtalálhatók [400]-ben olyan bizonyítások, melyekről itt most lemondtunk, mint a 4.16., 4.18. és 4.23. tételek, valamint a 4.20. lemma bizonyításai. Ehhez képest a 3. és ezen 4. fejezet előnye, hogy rövid, feszes és tömör, mindazonáltal érthető és pontos.

Bonyolultságelmélethez nagyobb háttéranyag található például a [203, 358, 482, 490] könyvekben. A Turing-gépet Alan Turing vezette be úttörő munkájában [470]. A 4.4. és a 4.5. táblázatot Garey és Johnson [149] készítették. A klasszikus [149] az NP-teljesség elméletéhez még ma is értékes forrásnak számít. A 4.7. tételnél említett több ezer NP-teljes probléma egy több száz darabos gyűjteménye szintén megtalálható Garey és Johnson [149] könyvében. A 4.10. tétel bizonyítása megtalálható más könyvekben is, például [149] és [358]. A 3-SAT probléma determinisztikus időbonyolultságánál említett oszd-meg-és-uralkodj algoritmust Monien [336], a lokális keresésen alapuló megoldást pedig Dantsin és társai [98] publikálták. A véletlen séta algoritmus Schöning [416, 418] munkáin alapszik. A $\leq_{\mathrm{T}}^{\mathrm{p}}$-visszavezethetőséget Cook [87], a $\leq_{\mathrm{m}}^{\mathrm{p}}$-visszavezethetőséget Karp [241] vezették be. Ladner, Lynch és Selman [271] cikke átfogó és mélyreható mű a bonyolultság-korlátozott visszavezethetőségek tanulmányozásához. A 4.4-1. gyakorlat, valamint a 4-1. feladat alapjait Selman [423] cikke jelenti.

Az eddigi legjobb $O(1.481^n)$ felső korlátot Dantsin és társai [98] érték el k-SAT determinisztikus időbonyolultságára $k \geq 3$ esetén. Schöning itt bemutatott valószínűségi algo-

Algoritmus	Típus	3-SAT	4-SAT	5-SAT	6-SAT
vISSZALÉPÉS	det.	$O(1.913^n)$	$O(1.968^n)$	$O(1.987^n)$	$O(1.995^n)$
Monien és Speckenmeyer [336]	det.	$O(1.618^n)$	$O(1.839^n)$	$O(1.928^n)$	$O(1.966^n)$
Dantsin et al. [98]	det.	$\boldsymbol{O(1.481^n)}$	$\boldsymbol{O(1.6^n)}$	$\boldsymbol{O(1.667^n)}$	$\boldsymbol{O(1.75^n)}$
Paturi et al. [361]	val.	$O(1.362^n)$	$O(1.476^n)$	$\boldsymbol{O(1.569^n)}$	$\boldsymbol{O(1.637^n)}$
Schöning [416]	val.	$O(1.334^n)$	$O(1.5^n)$	$O(1.6^n)$	$O(1.667^n)$
Iwama és Tamaki [223]	val.	$\boldsymbol{O(1.324^n)}$	$\boldsymbol{O(1.474^n)}$	—	—

4.10. ábra. Néhány – a kielégíthetőség problémát megoldó – algoritmus futási ideje.

ritmusa a „korlátozott lokális keresés ismétléssel” [416] ötleten alapszik. A **k-SAT** problémához $k \geq 4$ mellett Paturi és társai [361] algoritmusa kissé még jobb. A jelenlegi legjobb valószínűségi algoritmus **3-SAT**-hoz és **4-SAT**-hoz Iwama és Tamaki [223] munkáján alapul. Az ő algoritmusuk ügyesen ötvözi Paturi et al. [361] és Schöning [416] algoritmusait, és megközelítőleg $O(1.324^n)$ futási idővel rendelkezik. A **k-SAT** ezen algoritmusa $k \geq 5$ mellett nem jobb mint Paturi et al. [361] algoritmusa.

A 4.10. ábra áttekintést nyújt a fejezetben tárgyalt és néhány további kielégíthetőség problémát megoldó algoritmusról. A jelenlegi legjobb algoritmusok félkövér szedéssel szerepelnek.

A gráfizomorfizmus problémával Köbler, Schöning és Torán [247] könyve foglalkozik átfogó jelleggel, elsősorban bonyolultságelméleti szempontok szerint. Hoffman [208] csoportelméleti algoritmusokat vizsgált **GI**-hoz. A 4.11. tételt Ladner [270] publikálta, az alacsony- és magas bonyolultsági osztályhierarchiákat Schöning [414] vezette be. A 4.13. tételnél használt lexikografikus rendezés megtalálható Rothe [400] cikkében. Gál et al. [155] mutatták meg, hogy teljes izomorfizmus konstruálható $O(\lg n)$ méretű parciális izomorfizmusból két, egyenként n csúcsú izomorf gráfhoz. Ezt az eredményt a 4.13. tétel optimálisra javítja fel, kimondja, hogy ehhez már egy 1 méretű parciális izomorfizmus is elegendő. Ezen tétel, valamint a 4.6. példa Größe, Rothe és Wechsung [177] munkájára alapszik. A polinomiális idő-hierarchiát Meyer és Stockmeyer [323, 440] vezették be, akik bizonyították többek között a 4.16. tétel állításait. Az alsó- és felső-hierarchiát Schöning vezette be [414], valamint bizonyította a 4.18. tétel állításait [414] és megmutatta, hogy **GI** Low_2-beli [415]. Köbler et al. [245, 246] közöltek először eredményeket **GI** alsóságára vonatkozóan valószínűségi osztályok, mint amilyen pp ellenében. Eredményeiket Arvind és Kurur [20] javították, akik bebizonyították, hogy **GI** még SPP-beli is. Az univerzális hasítás módszerét Carter és Wegman [69] írták le 1979-ben. A 4.20. lemma Carter és Wegman [69] munkáján alapszik. Az SPP a Valiant [475] által bevezetett UP osztály általánosítása. Az SPP és más „ígéret” osztályok alapos tanulmányozásával számos munka foglalkozik, például [20, 56, 127, 194, 245, 246, 382, 399], a 4.23. tételben felsorolt tulajdonságok is megtalálhatók ezen művekben [127, 246, 247]. A 4.26. tétel bizonyításához szükséges Turing-ekvivalenciát **AUTO** és **GI** problémák közt Mathlon [312] mutatta meg (lásd még [247]).

Magyar nyelvű, bonyolultságelmélettel foglalkozó szakkönyvek Cormen, Leiserson, Rivest és Stein [90], Lovász [297], Papadimitriou [358], valamint Rónyai Lajos, Ivanyos Gábor és Szabó Réka [392] műve.

A szerző köszönetet mond Uwe Schöning-nek hasznos észrevételeiért ezen fejezet egy korábbi változatához. Leginkább Uwe Schöning előadásfóliáin alapul a 4.3. alfejezet RANDOM-SAT algoritmusának valószínűségi elemzése, amely az eredeti érvelés egy egysze-

rűsítése. Ennek részletesebb elemzése olvasható könyvében [417]. Ezen kívül hálás köszönet illeti Dietrich Stoyant, Sigurd Assingt és Holger Spakowskit a 3. és 4. fejezetek korábbi változatainak korrektúra-olvasásáért. Támogatta a szerzőt a Német Kutatási Társaság (Deutsche Forschungsgemeinschaft – DFG) a RO 1202/9-1 azonosítószám alatt.

II. HÁLÓZATOK

Bevezetés

Ez a rész három fejezetből áll.

Az *ötödik* fejezet hálózatok szimulációjával foglalkozik.

A *hatodik* fejezet címe *Párhuzamos számítások*. A fejezetben áttekintést adunk a párhuzamos számításokhoz használt leggyakoribb architektúrákról, bemutatjuk a legnépszerűbb számítási modelleket és végül alapvető informatikai feladaok megoldásra alkalmas párhuzamos algoritmusokat mutatunk be.

A *hetedik* fejezet a szisztolikus rendszerekről szóló alapvető ismereteket rendszerezi, és bemutatja néhány ismert módszer szisztolikus rendszerekkel való megvalósítását.

5. Hálózatok szimulációja

Ebben a fejezetben olyan módszereket és technikákat tárgyalunk, melyekkel szimulálhatjuk különböző hálózati számítógéprendszerek és alkalmazások működését. Egy hálózati rendszer vagy alkalmazás teljesítményének a hálózat fizikai megépítését, illetve az alkalmazás kibocsátását megelőző jellemzésére a hálózattervezés és menedzselés területén a szimuláció az egyik legelterjedtebb módszer.

5.1. A szimuláció típusai

5.1.1. Rendszerek, modellek és diszkrét-esemény szimuláció

Egy hálózati rendszer hálózati elemek – mint például a forgalomirányítók, kapcsolók, kapcsolatok, felhasználók és alkalmazások – olyan halmaza, melynek elemei együttműködnek valamilyen cél érdekében. A szimulációs tanulmány tárgya lehet egy olyan rendszer is, mely része egy másik rendszernek, mint például egy alhálózat. A hálózati rendszer állapota azoknak a lényeges változóknak és paramétereknek a halmaza, melyek leírják a rendszert egy bizonyos időben, mely magában foglalja az adott dolog vizsgálatát. Például, ha egy kapcsolat kihasználtságára vagyunk kíváncsiak, akkor csak a link által másodpercenként átvitt bitek számát és a kapcsolat teljes kapacitását akarjuk tudni, és nem a kapcsolat által összekötött kapcsolókban az egyes kapuk számára rendelkezésre álló puffermennyiséget.

A hálózat fizikai modelljének megépítése helyett egy matematikai modellt építünk, mely tükrözi a hálózatelemek működését és a közöttük lévő logikai és mennyiségi kapcsolatokat. A hálózatelemek közötti kapcsolatok megváltoztatásával a hálózatot annak fizikai megépítése nélkül elemezhetjük azt feltételezve, hogy a modell a valós rendszerhez hasonlóan viselkedik, tehát ez egy érvényes modell. Például, a kapcsolat kihasználtságát analitikusan kiszámíthatjuk az $U = D/T$ képlet felhasználásával, ahol D az adott idő alatt elküldött adatmennyiség, T pedig a kapcsolat kapacitása bit/másodpercben. Ez nagyon egyszerű modell, amely ritkán alkalmazható valós feladatok esetén. Sajnos, a problémák nagy része túl bonyolult ahhoz, hogy a felmerülő kérdéseket egyszerű matematikai egyenletekkel megválaszolhassuk. A nagyon bonyolult esetekben a szimulációs technika jobb.

A szimulációs modellek többféleképpen osztályozhatók. A leggyakoribb osztályozások a következők.

- *Statikus és dinamikus szimulációs modellek.* A statikus modell az időtől függetlenül jellemzi a rendszert. A dinamikus modell pedig egy időben változó rendszert ír le.
- *Sztochasztikus és determinisztikus modellek.* Ha a modell egy olyan rendszert ír le, amely véletlentől függő elemeket tartalmaz, akkor sztochasztikus modellnek nevezzük, egyébként determinisztikusnak. A hálózati modelleket megalapozó sorbanállási rendszerek véletlenszerű komponenseket tartalmaznak, mint például a csomagok beérkezési ideje egy adott sorba, a sorok kiszolgálási ideje, egy kapcsoló kapu kimenete stb.
- *Diszkrét és folytonos modellek.* A folytonos modell egy olyan rendszert ábrázol, melynek változói időben folyamatosan változnak. Például a differenciálegyenletek definiálják az összefüggéseket egyes állapotváltozók változásának a mértékére az idő változása szerint. Egy diszkrét modell egy olyan rendszert jellemez, ahol az állapotváltozók diszkrét időpillanatokban változnak meg. Az egyes események ezekben a diszkrét időpontokban fordulhatnak elő és változtathatják meg a rendszer állapotát. Például, egy csomag érkezése egy forgalomirányítóhoz egy bizonyos időpillanatban egy olyan esemény, amely megváltoztatja az forgalomirányító kapu-pufferének állapotát.

A továbbiakban mi dinamikus, sztochasztikus és diszkrét modelleket tételezünk fel, és diszkrét-esemény szimulációs modellekként hivatkozunk rájuk.

A számítógépes kommunikáció összetett természetének következtében a hálózati modellek is összetettek. A számítógépes programok fejlesztése az egyes szimulációs problémákra egy lehetőség, de ez nagyon sokáig tart és nem hatékony. Az utóbbi időben a szimulációs és modellező csomagok alkalmazása szokásossá vált, ami megtakarítja a kódolási időt és lehetővé teszi a modellező számára, hogy a programozási részletek helyett a modellezési problémára koncentráljon. Első pillantásra ezeknek a hálózatszimulációs és modellezési csomagoknak – mint például a COMNET, OPNET – használata azzal a veszéllyel jár, hogy a modellezőnek meg kell bíznia olyan modellezési technikákban és rejtett eljárásokban, amelyek jogilag védettek lehetnek, valamint nem nyilvánosak. A következő alfejezetekben azt tárgyaljuk, hogy a szimulációs módszertan hogyan győzi le az előbb említett félelmeinket az érvényesítési eljárások alkalmazásával, melyek célja, hogy megbizonyosodjunk arról, vajon a valós hálózati rendszer ugyanúgy működik-e majd, mint ahogyan azt a szimulációs modell előrejelezte.

5.2. A telekommunikációs hálózatok modellezésének és szimulációjának szükségessége

5.2.1. Szimuláció és emuláció

Az egyre több adat, számítógép, tároló rendszer és hálózat világában a rendszerek tervezése és menedzselése egyre nagyobb kihívássá válik. Amint a hálózatok gyorsabbá, nagyobbá és bonyolultabbá válnak, a hagyományos számítások már nem ésszerű megközelítései az új hálózati technológiáknál egy új hálózatterv megvalósításának és több millió dolláros befektetéseknek az érvényesítésénél. Ezek a bonyolult számítások és táblázatok már nem megfelelő eszközök a hálózati forgalom sztochasztikus természete, valamint az egész rendszer összetettsége miatt.

A különböző szervezetek egyre inkább azoktól az új hálózati technológiáktól és hálózati alkalmazásoktól függenek, melyek támogatják az üzleti szükségleteiket. Ennek eredményeképp, a gyenge hálózat-teljesítménynek súlyos következményei lehetnek az üzleti tevékenységükre nézve. Az egyes tervezési célok eléréséhez tartozó különböző alternatív megoldások kiértékelésében a hálózattervezők egyre inkább számítanak az olyan módszerekre, melyek segítik őket a számos javaslat kiértékelésében a végső döntés meghozatala és a valódi rendszer megépítése előtt. Egy széleskörűen elfogadott módszer a szimulációs teljesítmény-előrejelzés. A hálózattervező használhat egy szimulációs modellt a tervezési változatok elemzésére és egy új, vagy egy már létező rendszer módosításainak tanulmányozására azok fizikai megépítése nélkül. Egy szimulációs modell tükrözheti egy hálózat topológiáját és a hálózatban elvégzett feladatokat, hogy segítségével statisztikai eredményeket kapjuk a hálózat teljesítményéről.

Fontos a szimuláció és az emuláció közötti különbség megértése. Az emuláció célja az, hogy utánozza az eredeti hálózatot, és reprodukáljon minden eseményt, amely az egyes hálózatelemekben és alkalmazásokban bekövetkezik. Szimuláció esetén a cél az, hogy olyan statisztikai eredményeket állítsunk elő, amelyek kifejezik az egyes hálózatelemek viselkedését és funkcióit. Egy diszkrét-esemény szimuláció során meg akarjuk figyelni az eseményeket, amint azok megtörténnek az adott időben, és teljesítmény-jellemzőket szeretnénk meghatározni, hogy következtetéseket vonhassunk le a hálózat teljesítményéről, mint például a kapcsolat-kihasználtságokról, a válaszidőkről, a forgalomirányítók puffereinek méreteiről stb.

A nagyon sok hálózatelemet tartalmazó hálózatok szimulációja esetén olyan nagy modellt kaphatunk, amelynek elemezése a szimuláció alatt előállított nagy mennyiségű statisztika miatt bonyolult. Ezért ajánlatos a hálózatnak csak azon részeit modellezni, amelyek lényegesek a szimulációval megkapni kívánt statisztikákat illetően. Továbbá szükségszerű, hogy a modell csak azokat a részleteket tartalmazza, amelyek a szimuláció céljai miatt fontosak. A hálózattervezők általában a következő célokat tűzik ki:

- *Teljesítménymodellezés.* Statisztikák előállítása a kapcsolatok, forgalomirányítók, kapcsolók, pufferek stb. teljesítmény paramétereiről.
- *Hibaelemzés.* A hálózatelemek hibás működése következményeinek elemzése.
- *Hálózattervezés.* Különböző hálózattervek statisztikáinak összehasonlítása az egyes tervezési javaslatok követelményeinek kiértékelése céljából.
- *Hálózati erőforrások tervezése.* Olyan változtatások hatásának mérése a hálózat teljesítményén, mint pl. az új felhasználókkal, alkalmazásokkal vagy hálózatelemekkel való bővítés.

A céloktól függően ugyanahhoz a hálózathoz különböző szimulációs modellekre lehet szükség. Például, ha a modellező meg akarja határozni egy protokoll új szolgáltatása által okozott többletterhelést a kommunikációs kapcsolatokon, akkor a modell linkjeinek csak az új szolgáltatás által előállított forgalmat kell reprezentálnia. Ha viszont a modellező egy alkalmazás válaszidejét akarja vizsgálni maximális forgalom-terhelés esetén, akkor mellőzheti az előző modellbeli protokoll új szolgáltatásának megfelelő forgalmat.

Egy másik fontos kérdés a modell finomsága, azaz hogy milyen szintű részletekig modellezzük az egyes hálózatelemeket. Például el kell döntenünk, hogy egy forgalomirányító belső felépítését akarjuk modellezni, vagy pedig egy egész csomagkapcsolt hálózatot. Az első esetben meg kell adnunk a forgalomirányító belső komponenseit, a processzorok szá-

mát és sebességét, a buszok típusait, a kapuk számát, a kapu-pufferek mennyiségét továbbá az ezen komponensek közötti kölcsönhatásokat is. De ha alkalmazásszinten akarjuk vizsgálni a válaszidőt az egész csomagkapcsolt hálózatban, akkor a forgalomirányítók belső részletei helyett inkább az alkalmazások és protokollok típusait, a hálózat topológiáját és a kapcsolatok kapacitását kell meghatároznunk. Habár a forgalomirányítók alacsonyszintű működése hatással van a végpontok közötti válaszidőre, ennek részletes modellezése az egész hálózatot tekintve nem járul hozzá meghatározóan a szimulációs eredményekhez. A nanoszekundum nagyságrendű belső működési részletek modellezése nem járul hozzá jelentősen a végpontok közötti mikroszekundum vagy másodperc nagyságrendű késleltetés vizsgálatához. A nagyobb modell-finomságból eredő további pontosságot erősen ellensúlyozza a modell bonyolultsága, és ezeknek a részleteknek a modellbe foglalása által igényelt munka és idő.

Az egyszerűsítés történhet statisztikai függvények alkalmazásával is. Például a cellahibákat egy ATM hálózatban nem feltétlenül kell úgy modellezni, hogy egy kommunikációs kapcsolaton a cella fejlécében megváltoztatunk egy bitet, ezáltal idézve elő rossz CRC-t a fogadónál. Használhatunk egy statisztikai függvényt is annak eldöntésére, hogy egy cella megsérült vagy elveszett-e, tehát így a cella részleteit nem kell megadnunk a cellahibák modellezéséhez.

Ezek a példák azt szemléltetik, hogy a hálózatszimuláció célja nem a hálózat emulációja, hanem a funkcionalitás bizonyos vizsgálatok céljára történő reprodukciója.

5.3. A telekommunikációs hálózatok típusai

5.3.1. Modellezési konstrukciók

Egy kommunikációs hálózat hálózatelemekből, csomópontokból (küldőkből és fogadókból) és az őket összekötő kommunikációs közegekből áll. A számos hálózat osztályozási kritérium közül mi kettőt használunk: az átviteli technológiát és méretet. A méret vagy távolság szintén meghatározza a hálózatban alkalmazott technikát, amely például lehet vezetékes vagy vezetéknélküli. Két vagy több hálózat összekapcsolását ***internetworknek*** nevezzük. A legismertebb internetwork az Internet.

Az átviteli technológiától függően nagyjából két csoportba oszthatjuk a hálózatokat, üzenetszórásos és pont-pont kapcsolatú hálózatok:

- Az *üzenetszórásos hálózatokban* egy kommunikációs csatorna van megosztva az összes csomópont között. A kommunikáció úgy történik, hogy a csomópontok a minden más csomópont által küldött csomagokat vagy kereteket is megkapják. A keretben lévő cím határozza meg a címzettet vagy címzetteket, és csak a címzett csomópontok dolgozzák fel a keretet. Az üzenetszórásos technológiák azt is lehetővé teszik, hogy a keretet az összes csomópontnak címezzük. Az ilyen üzenetszórásos kereteket a hálózat összes csomópontja feldolgozza. A kereteket címezhetjük csomópontok egy csoportjának a tagjainak is, vagy közülük egy tetszőleges csomópontnak is. Az előbbit többesküldésnek, az utóbbit bárkinek való küldésnek nevezzük.
- A *pont-pont kapcsolatú hálózatok* csomópontpárok közötti kapcsolatok sokaságából állnak. Itt egy adott küldőtől egy adott címzetthez küldött csomagot vagy keretet esetleg

más csomópontokon keresztül kell továbbítani. Ezek tárolják és továbbítják azt, amíg el nem éri célját.

A másik osztályozási szempont, vagyis a lefedett fizikai terület nagysága alapján a következőképpen csoportosíthatjuk a hálózatokat:

- A *személyes hálózati környezetek* (**P**ersonal **A**rea **N**etwork – PAN) egy adott személy igényeit elégítik ki. Például egy billentyűzetből, egérből és PDA-ból („Personal Digital Assistant" – Digitális Személyi Asszisztens – mozaikszava) álló, vezetéknélküli hálózat tekinthető egy személyes hálózati környezetnek.
- A *helyi hálózatok* (**L**ocal **A**rea **N**etwork – LAN) egy korlátozott méretű területet fednek le, tipikusan magánszemélyek, részlegek, kisebb otthoni szervezetek vagy épületek adott emeletei birtokolják őket. A helyi hálózat munkaállomásokat, kiszolgálókat és megosztott erőforrásokat kapcsol össze. A LAN-ok csoportosíthatók továbbá az átviteli technológia (melynek sebességét bit/másodpercben mérjük) és a hálózati topológia szerint. Az átviteli technológiák sebessége a hagyományos 10 Mbps-tól 10 Gbps sebességűig terjed. Topológia szerint vannak busz és gyűrű hálózatok, továbbá kapcsolt LAN-ok.
- A *városi hálózatok* (**M**etropolitan **A**rea **N**etwork – MAN) nagyobb területet hidalnak át, például egy várost. Egy széleskörűen alkalmazott MAN a kábel-tv hálózat, amely nem csak az egyirányú TV adásokat, hanem kétirányú Internet szolgáltatásokat is biztosít az átviteli spektrum nem használt részében. Más MAN technológiák például az *optikai szálas elosztott adatinterfész* és a következőkben tárgyalt vezetéknélküli IEEE (**I**nstitute of **E**lectrical and **E**lectronical **E**ngineers) technológiák.
- A *nagy kiterjedésű hálózatok* (**W**ide **A**rea **N**etwork – WAN) nagy földrajzi területet fednek le, egy államot, országot vagy akár egy kontinenst is. Egy WAN alhálózatokban összekapcsolt hosztokból (kliensekből és szerverekből) áll. Az alhálózatok továbbítják az üzeneteket a forrástól a cél hosztig. Egy alhálózat tartalmazhat számos átviteli vonalat, melyek mindegyike speciális hardver eszközöket, úgynevezett forgalomirányítókat kapcsol össze. Az átviteli vonal több különböző közeg is lehet, rézdrót, optikai üvegszál, vezetéknélküli kapcsolat stb. Mikor egy üzenetet el kell küldeni egy vagy több hosztnak, a küldő az üzenetet kisebb részekre osztja, melyeket ***csomagoknak*** nevezünk. Amikor egy csomag megérkezik egy átviteli vonalon, a forgalomirányító tárolja azt, kiválaszt egy kimenő vonalat és azon továbbítja. A kimenő vonal kiválasztása egy forgalomirányítási algoritmus alapján történik. Végül az egyesével továbbított csomagokból a cél hoszt(ok) újra összeállítják az eredeti üzenetet.

A vezetéknélküli hálózatok a következő osztályokba sorolhatók: rövidtávú rádióhálózatok, vezetéknélküli LAN-ok és vezetéknélküli WAN-ok.

- Rövidtávú hálózatoknál az egyes eszközök 6–10 méteren belül vannak összekapcsolva rövidtávú rádiókapcsolatokkal. Ilyenek például a Bluetooth, a különböző komponensek, digitális kamerák, a ***globális helymeghatározó rendszer*** („Global Positioning System" – GPS) eszközei, a fejhallgatók, számítógépek, szkennerek, monitorok és billentyűzetek. A komponensek elsődleges-másodlagos viszonyban vannak. A fő rendszeregység, az elsődleges komponens irányítja a másodlagos komponensek működését. Az elsődleges komponens határozza meg, hogy milyen címeket használnak a másodlagos eszközök, továbbá, hogy melyik frekvenciákon továbbíthatnak adatokat.

- A vezetéknélküli LAN számítógépekből és rádió-modemmel és antennával felszerelt hozzáférési pontokból áll, melyek lehetővé teszik az adatok küldését és fogadását. A számítógépek kommunikálhatnak egymással közvetlenül vagy egy hozzáférési ponton keresztül, ami más hálózatokhoz is kapcsolhatja őket. A lefedett terület általában 100 méter körül van. A vezetéknélküli LAN protokollok az IEEE 802.11 szabványok családjába tartoznak, és 11-108 Mbps sebességet biztosítanak.
- A vezetéknélküli WAN-ok lehetnek kis és nagy sávszélességű hálózatok is. A kis sávszélességű rádió-hálózatok eddig kifejlesztett három generációját a celluláris telefonrendszerek használják. Az első generációt csak hang-kommunikációra tervezték, ez analóg jelzést használt. A második generáció szintén csak hangot továbbított, de digitális átviteli technológiára alapozva. Az aktuális harmadik generáció digitális és hangot és adatokat is továbbít legfeljebb 2 Mbps sebességgel. Negyedik és további generációs celluláris rendszerek is fejlesztés alatt vannak. A nagy sávszélességű WAN-ok a telefonrendszerek használatát elkerülve biztosítanak nagy sebességű hozzáférést az otthonokból és cégektől. Az IEEE 802.16 szabvány az IEEE 802.11 szabvánnyal ellentétben épületekhez és nem mobil állomásokhoz továbbítja a szolgáltatásokat, és sokkal magasabb, 10–66 GHz frekvencia-tartományban üzemel. A távolság az épületek között több kilométer is lehet.
- A vezetékes és vezetéknélküli otthoni hálózatok a különböző, Interneten keresztül elérhető eszközök összekapcsolásával egyre inkább népszerűbbek lesznek. Az otthoni hálózatok állhatnak PC-kből, laptopokból, PDA-kból, TV-kből, DVD-kből, videókamerákból, MP3 lejátszókból, mikrohullámú sütőkből, hűtőszekrényekből, lámpákból, riasztókból, kihasználtság-mérőkből stb. Sok háztartás ma már fel van szerelve nagy sebességű Internet-hozzáféréssel (kábel modem, DSL stb.), amin keresztül igény szerint tölthetnek le zenét és filmeket.

A kommunikációs hálózatok különböző komponensei és típusai megfelelnek a szimulációs modellépítés konstrukcióinak és különböző lépéseinek. Tipikusan a hálózati topológiát építik fel először, majd hozzáadják a forgalom forrásait, céljait, a terhelést és a hálózati működés paramétereinek beállítását. A szimulációt vezérlő paraméterek meghatározzák a kísérletet és a szimuláció lefutását. A szimuláció indítása előtt a szimuláció alatti és utáni elemzéshez különböző statisztikai jelentések aktiválhatók. Statisztikai eloszlások állnak rendelkezésre ahhoz, hogy leírjuk a beépített analitikus eloszlások speciális paraméterezéseit. Amint a modell elkészült, a modellező új modellkönyvtárakat készíthet, melyek újra felhasználhatók más modellekben is.

5.4. Teljesítményjellemzők szimulációhoz

5.4.1. Teljesítménymértékek

Ebben az alfejezetben a hálózatok olyan jellemzőinek egy nem kimerítő listáját tekintjük át, melyek meghatározó hatással vannak a hálózat teljesítményére. Általában ezek a jellemzők képezik a hálózat-modellezésnek, azaz a számítógéphálózatok statisztikai elemzésének, tervezésének és optimalizálásának céljait. Alapvetően a hálózatmodelleket úgy készítjük,

hogy megadjuk egy sorbanállási rendszerben az igény érkezési és kiszolgálási intenzitások statisztikai eloszlását, ami azután meghatározza ezeket a jellemzőket.

- *Kapcsolat kapacitás.* A csatorna vagy kapcsolat-kapacitás a kapcsolat által időegységenként továbbított üzenetek száma. Ezt általában bit/másodpercben mérik. Az információelmélet egyik leghíresebb eredménye Shannon csatornakódolási tétele: „Minden csatornára létezik olyan kód, ami lehetővé teszi a csatornán való hibamentes adattovábbítást R intenzitással, feltéve, hogy $R \leq C$, ahol C a csatorna kapacitása." Egyenlőség csak akkor lehetséges, ha a jel/zaj hányados („Signal-to-Noise Ration" – SNR) végtelen.
- *Sávszélesség.* A sávszélesség a hálózati jelek számára rendelkezésre álló legalacsonyabb és legmagasabb frekvenciák közötti különbség. A sávszélesség kifejezést szintén használják egy adott kapcsolat vagy protokoll áteresztőképességének leírására, kilobit, megabit, terabit stb. /másodpercben mérve.
- *Válaszidő.* A ***válaszidő*** az az idő, amire a hálózati rendszernek szüksége van ahhoz, hogy megválaszolja egy igényforrás kérését. A válaszidő tartalmazza a célállomáshoz való továbbítás idejét, a feldolgozási időt a forrásnál, a célnál, valamint a közbeeső hálózatelemeknél, továbbá tartalmazza a válasznak a forráshoz való átvitelének idejét. Az átlagos válaszidő a hálózat teljesítményének fontos mértéke. A felhasználók számára a minél kisebb válaszidő a legkedvezőbb. A válaszidő statisztikáknak (átlag és szórás) állandóknak kell lenniük, és nem szabad, hogy napszaktól függjenek. Meg kell jegyeznünk viszont, hogy az alacsony átlagos válaszidő nem garantálja azt, hogy nem lesznek hálózati torlódások miatt rendkívül hosszú válaszidők is.
- *Késés.* A ***késés*** vagy késleltetés az az időmennyiség, ami egységnyi adatnak egy hálózati kapcsolaton való továbbításához szükséges. A késleltetés és a sávszélesség az a két tényező, ami meghatározza egy kapcsolat sebességét. A késleltetés tartalmazza a terjedési késleltetést (azt az időt, ami alatt az elektronikus vagy optikai jelek megteszik a két pont közötti távolságot) és a feldolgozási időt. Például egy földi állomásnak egy másik földi állomással való műholdas kommunikációs kapcsolatának késése (legalább 34000 km mindkét út) körülbelül 270 milliszekundum. Az USA keleti és nyugati partjai közötti válaszolási késleltetés körülbelül 100 milliszekundum, globálisan 125 milliszekundum. Egy több szegmensből álló adatútnak a forrás és cél végpontok közötti késleltetését nem csak a jel terjedési sebessége, hanem az útvonalon lévő hálózati eszközök, forgalomirányítók, kapcsolók is befolyásolják, melyek tárolják, feldolgozzák, irányítják, kapcsolják, és becsomagolják az adatokat. A hibás csomagok és cellák, a jelveszteség, a kapcsolat és eszközhibák, valamint túlterhelések szintén hozzájárulnak a hálózati késleltetéshez. Hibás cellák és csomagok esetén szükség van azoknak a forrástól való teljes újraküldésére. Az ilyen a csomagokat általában eldobják – azt feltételezve, hogy később újraküldik őket. Ez lassulást és a pufferek túlcsordulását okozhatja.
- *Forgalomirányító protokollok.* A forrástól a célig a hálózati forgalom valamilyen útvonalon halad keresztül. Egy helyi hálózatban ez az útvonal nem egy problémás kérdés, mivel csak egy út lehetséges bármely forrás és cél között. Ha viszont a hálózat számos céget kapcsol össze számos útvonallal, forgalomirányítóval és kapcsolattal, a legjobb út vagy útvonalak megkeresése igen fontossá válik. Egy útvonal többféle különböző kapacitású, késleltetésű és megbízhatóságú kapcsolatból állhat. Létesítésük forgalomirányító protokollok segítségével történik. Ezeknek a protokolloknak az a célja, hogy a

torlódást elkerülve egy optimális, vagy közel optimális útvonalat találjanak a forrás és a cél között.

- *Forgalomszervezés.* A forgalomszervezés koncepcióját felhasználva jelenleg egy újfajta forgalomirányítási technikát fejlesztenek. A folyamatos hálózatkapacitás-növelés helyett inkább a hálózati erőforrások optimális kiosztását alkalmazzák a torlódások elkerülésére. A forgalomszervezést úgy valósítják meg, hogy a forgalomfolyamokat a fizikai hálózati topológiára képezik le előre meghatározott útvonalak mentén. Fő cél a forgalomirányítók és kapcsolók továbbító kapacitásainak optimális kiosztása. A forgalomszervezés megteremti annak lehetőségét, hogy a forgalomfolyam eltérjen a hagyományos forgalomirányítási protokollok által kiszámított optimális útvonaltól egy kevésbé zsúfolt hálózatrész felé. Kiegyenlíti a terhelést a kapcsolatokon, forgalomirányítókon és kapcsolókon úgy, hogy ezen hálózatelemek egyike se legyen túl kevéssé vagy túlságosan is kihasználva.
- *Protokoll többletterhelés.* A protokollüzenetek és az alkalmazások adatai protokoll-adategységekbe vannak ágyazva, mint például keretek, csomagok és cellák. A hálózattervezők egyik fő aggálya a protokollok többletterhelése. Ez a többletterhelés a következő kérdést veti fel. Valójában milyen gyorsan tudunk adatokat továbbítani egy adott kommunikációs útvonallal és protokollcsomaggal, azaz mekkora sávszélesség marad valójában az alkalmazásoknak? A legtöbb protokoll további többletterhelést is bevezet sávszélességen belüli protokoll-menedzselési funkciókkal kapcsolatban. A kapcsolatfenntartó csomagok, hálózati riasztások, vezérlő és ellenőrző üzenetek, a poll, select és különböző jelzőüzenetek az adatfolyamokkal együtt továbbítódnak.
- *Erős ingadozás.* A hálózati torlódás leggyakoribb oka a forgalom erős ingadozása. Egyes nem régi eredmények nyilvánvalóvá tették a nagy sebességű Internet forgalmának erősen ingadozó voltát és azt, hogy a változékonysága a korábbi feltételezésekkel szemben nem jelezhető előre. Megmutatták, hogy a hálózati forgalom gyakran hasonló statisztikai tulajdonságokkal rendelkezik. A gyakran vagy mindig erősen ingadozó forgalom statisztikailag leírható a *hosszú távú függőség* fogalmának felhasználásával. A hosszú távon függő forgalomnak mindig megfigyelhető erős ingadozása. Az előbbiek egyik következménye az, hogy a különböző adatfolyamok összekeverése, mint ahogyan az az Interneten is történik, nem eredményez egyenletes forgalmat. A helyi és nagy kiterjedésű hálózatokban végzett mérések bebizonyították, hogy a széleskörűen használt Markov-folyamaton alapuló modellek nem alkalmazhatók napjaink hálózati forgalmának leírására. Ha a forgalom Markov-folyamat lenne, az erős ingadozás kiegyenlítődne egy hosszú időintervallum átlagolása során, ellentmondva a megfigyelt forgalomjellemzőknek. Az erősen ingadozó forgalom káros következményeit a 5.9. alfejezetben lévő esettanulmányban vizsgáljuk meg.
- *Keret méret.* A hálózattervezők gyakran aggódnak a nagyméretű keretek miatt, ugyanis az ilyen elveszett keretek és újraküldésük esetén sokkal gyorsabban fel tudják tölteni a forgalomirányítók puffereit mint a kisebbek. Mivel a feldolgozási késleltetés ugyanakkora a nagyobb kereteknél, mint a kisebbeknél, azaz a nagyobb csomagok látszólag hatékonyabbak, de a forgalomirányítók és kapcsolók gyorsabban fel tudják dolgozni a belső soraikat kisebb csomagok esetén. A nagyobb keretek esetén szükség lehet azok feldarabolására is, mivel a *maximális adatátviteli egység* („Maximum Transmission Unit” - MTU) mérete korlátozza a méretüket. Az MTU egy olyan paraméter, ami

meghatározza egy IP interfész által átvihető legnagyobb csomag méretét. Másrészt, a kisebb keretek több ütközést okozhatnak az Ethernet hálózatokban vagy kisebb kihasználtságot a WAN kapcsolatokon.

- *Az eldobott csomagok aránya.* A csomagok az OSI architektúra adatkapcsolati és hálózati rétegében kerülhetnek eldobásra. A szállítási réteg puffereket kezel a nyugtázatlan csomagok számára, és szükség esetén újraküldi őket, hogy ezáltal egy hibamentes kapcsolatot biztosítson a küldő és a fogadó között. Az alsóbb rétegekben eldobott csomagok aránya meghatározza a szállítási rétegben újraküldött csomagok arányát. A forgalomirányítók és kapcsolók a belső pufferek hiánya miatt is eldobhatnak csomagokat. Ha a WAN kapcsolatok zsúfolttá válnak, akkor a pufferek gyorsabban feltöltődnek, ami pedig időtúllépéseket és újraküldéseket okoz a szállítási rétegben. A TCP *lassú indulás algoritmusa* a válaszidő folyamatos becslésével és a továbbítási intenzitásnak a korábbi ingadozásától függő beállításával próbálja elkerülni a torlódást.

5.5. A forgalom jellemzése

5.5.1. Hálózati statisztikák

A kommunikációs hálózatok véletlentől függő jellemzőkkel továbbítják az adatokat. A hálózat jellemzőinek mértékei véletlen folyamatokból vett statisztikai minták. Ilyen például a válaszidő, a kapcsolat kihasználtság, az üzenetek beérkezési időköze stb. Ebben az alfejezetben áttekintjük a hálózatmodellezésben és teljesítménybecslésben legfontosabb statisztikákat.

A vizsgált hálózatjellemzőnek megfelelő eloszlástípus kiválasztása után a következő lépés az eloszlás paramétereinek becslése. Erre sok esetben a minta átlagát vagy közepét és szórását használják. A fejlettebb szoftver-eszközök tartalmazzák a szükséges számításokat ezekhez a becslésekhez. Az átlag úgy értelmezhető, mint az az érték, ami körül a minta értékei csoportosulnak. A következő egyenletek akkor alkalmazhatók, ha rendelkezésre állnak diszkrét vagy folytonos alapadatok. Legyen $X_1, X_2, \ldots, X_n$ a rendelkezésre álló, n méretű minta. Ennek az átlagát a következő képlet adja meg:

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n} .$$

A minta S^2 szórásnégyzete pedig a következőképpen definiálható:

$$S^2 = \frac{\sum_{i=1}^{n} {X_i}^2 - n\overline{X}^2}{n-1} .$$

Ha az adatok diszkrétek és egy gyakorisági eloszlásba vannak csoportosítva, akkor az előbbi egyenletek a következőképpen módosulnak:

$$\overline{X} = \frac{\sum_{j=1}^{k} f_j X_j}{n} ,$$

$$S^2 = \frac{\sum_{j=1}^{k} f_j {X_j}^2 - n\overline{X}^2}{n-1} ,$$

eloszlás	paraméter(ek)	becslés(ek)
Poisson	α	$\hat{\alpha} = \overline{X}$
exponenciális	λ	$\hat{\lambda} = 1/\overline{X}$
egyenletes	b	$\hat{b} = ((n+1)/n)[\max(X)]$ (torzítatlan)
normális	μ, σ^2	$\hat{\mu} = \overline{X}$ $\hat{\sigma}^2 = S^2$ (torzítatlan)

5.1. ábra. Leggyakoribb eloszlások paramétereinek becslése.

ahol k az X különböző értékeinek száma, f_j pedig az X_j érték gyakorisága. Az S szórás az S^2 szórásnégyzet négyzetgyöke.

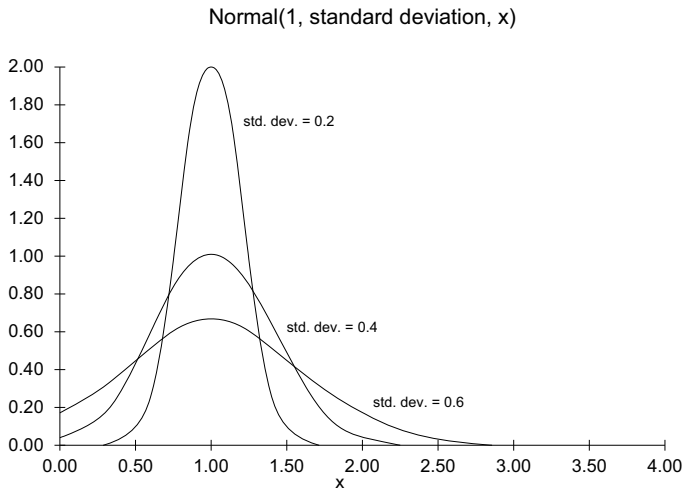
A szórásnégyzet és a szórás szemlélteti a mintáknak az átlagérték körüli eltérését. A kis eltérés a mintáknak egy erős, középre irányuló tendenciáját mutatja. A nagy eltérés viszont kis tendenciát és nagy statisztikai véletlenszerűséget mutat.

Az eloszlás paramétereinek numerikus becslései azért szükségesek, hogy az adott eloszláscsaládot egy eloszlásra redukálhassuk és tesztelhessük az adott feltevést. Az 5.1. ábra a hálózatmodellezésben előforduló leggyakoribb eloszlások paramétereinek becsléseit foglalja össze. Jelölje α az adott paramétert, $\hat{\alpha}$ pedig a becslését. Ha eltávolítjuk a torzítást a σ^2 becslésétől a normális eloszlásnál és a b-étől az egyenletes eloszlásnál, akkor ezek a becslések a mintaadatokon alapuló maximum likelihood becslések.

A valószínűségeloszlások a valós világban előforduló véletlenszerű változásokat írják le. Habár a változásokat véletlennek nevezzük, a véletlenszerűségnek vannak különböző mértékei; a különböző eloszlások megfelelnek annak, ahogyan a változások bekövetkeznek. Ezért a különböző eloszlásokat különböző szimulációs célokra használják. A valószínűségeloszlások eloszlásfüggvényekkel vannak reprezentálva. Ezek megmutatják, hogy egy bizonyos érték mennyire valószínű. A kumulatív eloszlásfüggvény annak a valószínűségét adja meg, hogy egy bizonyos értekkel egyező, vagy annál kisebb értéket választunk. Például, ha a kumulatív eloszlásfüggvény az 1 értéknél 0.85, akkor ebből az eloszlásból választva az esetek 85%-ában 1-nél kisebb számot kapunk. A kumulatív eloszlásfüggvény értéke egy adott pontban megegyezik a megfelelő sűrűségfüggvény-görbe alatt a ponttól balra lévő területtel. Mivel a sűrűségfüggvény alatti teljes terület egyenlő 1-gyel, ezért a kumulatív eloszlásfüggvény 1-hez tart, ahogyan a pozitív irányban haladunk előre. A modellezési esetek többségében a modellezőnek nem kell tudnia minden részletet ahhoz, hogy sikeresen felépítsen egy szimulációs modellt. Elég azt tudnia, hogy melyik eloszlás a legmegfelelőbb az adott esetre.

Most ismertetjük a leggyakoribb statisztikai eloszlásokat. A megfelelő sűrűségfüggvények ábrázolására a COMNET szimulációs modellezési eszközt használjuk. Gyakorlati szempontból egy sűrűségfüggvény egy hisztogrammal közelíthető az összes előfordulás gyakoriságainak valószínűségekké való konvertálásával.

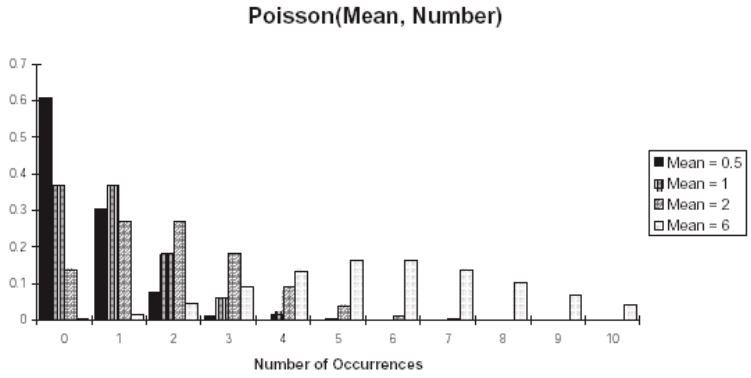
- *Normális eloszlás.* Tipikusan egy összetett folyamat eloszlásának modellezésére használható, ahol a folyamat komponens folyamatok összegeként írható le. Például, egy fájl hálózaton való átvitelének ideje (válaszidő) a fájlt felépítő blokkok átviteli idejeinek az összegével egyezik meg. A modellező eszközökben a normális eloszlás két pozitív valós számmal adható meg: a várható értékkel és a szórással. A függvényértékek szintén pozitív valós számok. Az x paraméter azt adja meg, hogy melyik véletlen számsoroza-



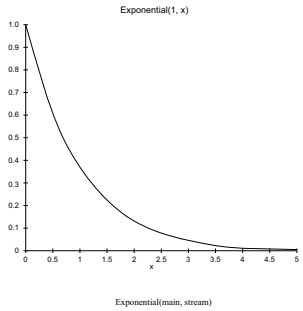
5.2. ábra. Normális eloszlás.

tot alkalmazzuk a mintakészítéshez. Ez az eloszlás szintén gyakran használt üzenetek méretének modellezésére. Például, egy üzenet leírható 20000 bájt átlagos mérettel és 5000 bájt szórással.

- *Poisson-eloszlás.* Ez az eloszlás az egy bizonyos időintervallumban bekövetkező független események előfordulásának számát modellezi. Például egy csomagfolyamból egy cél által egy másodperc vagy perc alatt megkapott csomagok számát. A modellező eszközökben a Poisson-eloszlás egy pozitív valós számmal, a várható értékkel adható meg. A „szám” paraméter az 5.3. ábrán azt adja meg, hogy melyik véletlen számsorozat lesz alkalmazva a mintakészítéshez. Amikor ez az eloszlás egy időintervallummal együtt van megadva és egy egész számot ad, olyankor gyakran használják az adott intervallumban bekövetkező beérkezések számának leírására. Szimulációkor viszont sokkal hasznosabb, ha ez az információ a beérkezések közötti időközként van kifejezve. Erre a célra az exponenciális eloszlást használják.
- *Exponenciális eloszlás.* Ez az eloszlás a független események között eltelt időt modellezi, mint például a csomagfolyam egy forrása által küldött csomagok esetén a beérkezési időközt. Fontos megjegyezni, hogy ha az események bekövetkezése közötti idő exponenciális, akkor a bekövetkezett események száma Poisson-eloszlású. A modellező eszközökben az exponenciális eloszlás egy pozitív valós számmal, az eloszlás várható értékével, és egy x paraméterrel adható meg, ami pedig azt mondja meg, melyik véletlen számsorozatot alkalmazzuk a mintakészítéshez. Az eloszlás további alkalmazási területei még például: adatbázis tranzakciók, billentyű leütések, fájl hozzáférések, elektronikus levelek, névfeloldási kérések, HTTP lekérdezések, X-window protokoll üzenetváltások stb. közötti idők modellezése.
- *Egyenletes eloszlás.* Az egyenletes eloszlás olyan adatokat modellez, melyek egy intervallumból vesznek fel értékeket, mindegyiket egyenlő valószínűséggel. Az eloszlás teljesen meghatározható a legkisebb és a legnagyobb lehetséges érték megadásával. Diszkrét adatok esetén a megfelelő diszkrét egyenletes eloszlás is használható. A cso-



5.3. ábra. Poisson-eloszlás.

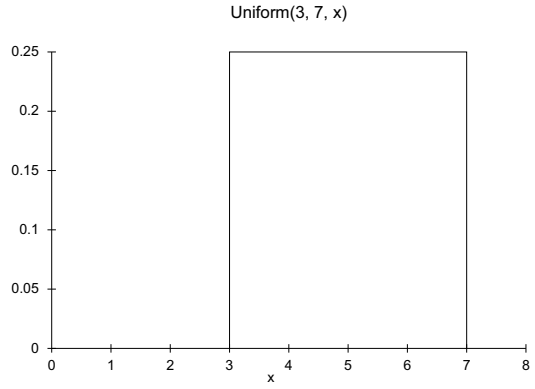


5.4. ábra. Exponenciális eloszlás.

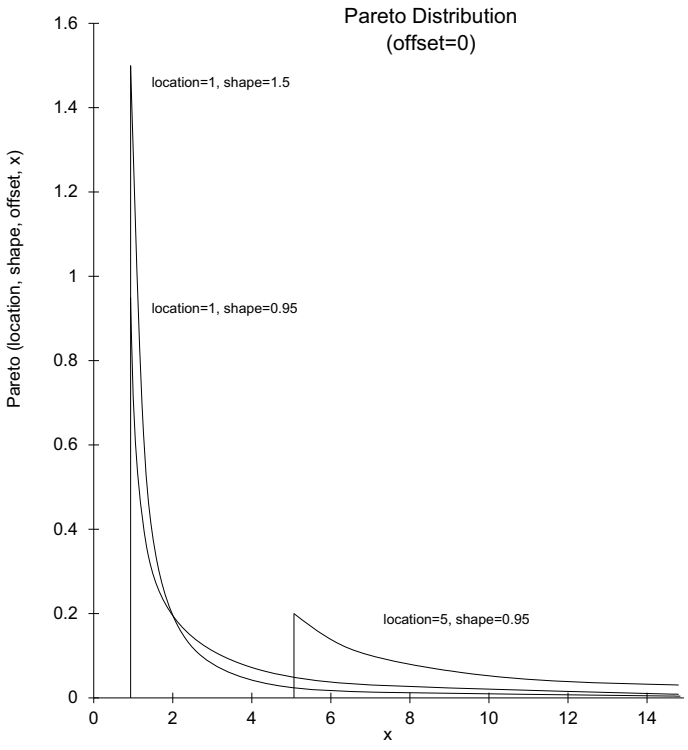
maghosszakat gyakran modellezik egyenletes eloszlással. A modellező eszközökben ez az eloszlás három pozitív valós számmal adható meg. Az első kettő a legkisebb és a legnagyobb lehetséges értéket adja meg, a harmadik pedig azt, hogy melyik véletlen számsorozat legyen alkalmazva a mintakészítéshez.

- *Pareto-eloszlás.* A Pareto-eloszlás egy hatvány típusú eloszlás erősen ingadozó tulajdonsággal rendelkező források modellezésére (nem hosszú távon függő forgalomra). Az eloszlás erősen csúcsos, de a vége lassan csökken. Megadása a következő három paraméterrel történik: hely, alak és eltolás. A hely megadja ahol az eloszlás kezdődik, az alak meghatározza, hogy a vége milyen gyorsan ereszkedik, az eltolás pedig eltolja az eloszlást.

A valószínűségeloszlások egy közös alkalmazási célja az, hogy definiálják a hálózatok különböző paramétereit. A hálózatok egy tipikus modellezési célú paramétere több üzenet esetén az üzenetek közötti idő. A megadott időn egy üzenet indulásától a következő üzenet indulásáig eltelt időt értjük. Ahogyan azt korábban tárgyaltuk, a beérkezési időközökre leggyakrabban használt eloszlás az exponenciális. Az exponenciális eloszlás számára megadandó paraméterek a várható érték és a véletlen folyamszám. A hálózati forgalmat gyakran írják le Poisson folyamatként. Ez általában azt jelenti, hogy az üzenetek számát figyelték

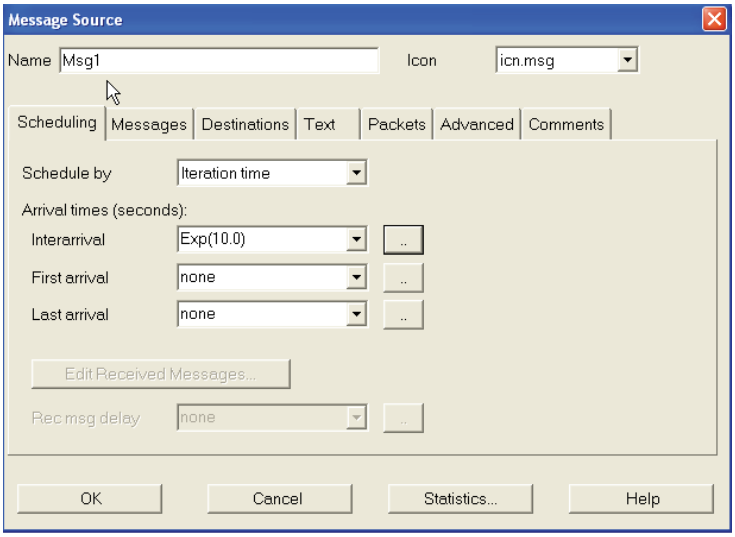


5.5. ábra. Egyenletes eloszlás.

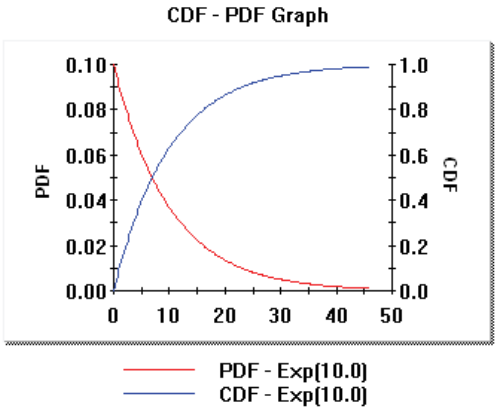


5.6. ábra. Pareto-eloszlás.

meg egymást követő időintervallumokban, és a megfigyelések száma egy intervallumban Poisson-eloszlású. A modellező eszközökben nem az időegységenkénti üzenetek számát

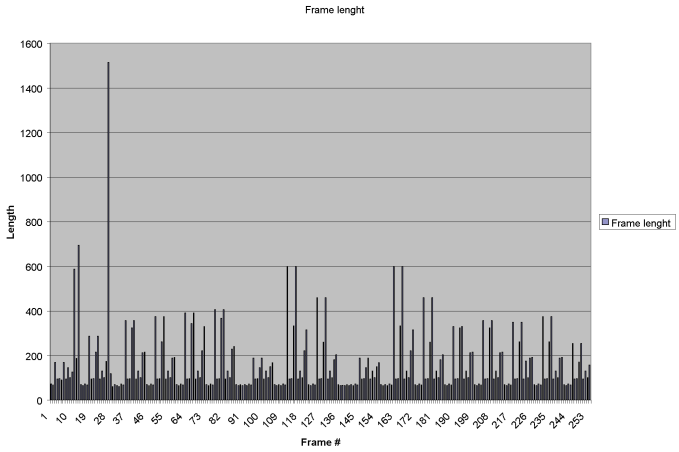


5.7. ábra. Exponenciális eloszlás átlagosan 10 másodperces beérkezési időközzel.



5.8. ábra. Az Exp(10.0) beérkezési időköz sűrűségfüggvénye.

nem adják meg, hanem inkább az üzenetek beérkezési időközét. Bebizonyítható, hogy ha az egységnyi időintervallumonkénti üzenetszám Poisson-eloszlású, akkor a beérkezési időköz exponenciális eloszlású. A 5.7. ábrán látható párbeszédablakban a beérkezési időköz eloszlás COMNET-ben az Exp(10.0) kifejezéssel van definiálva. Ez azt jelenti, hogy az egy üzenet és az utána következő üzenet indulása közötti idő exponenciális eloszlást követ 10 másodperces átlaggal. A 5.8. ábra a megfelelő sűrűségfüggvényt szemlélteti.



5.9. ábra. Anomáliák a kerethosszakban.**10-es ábra!!!**

Több szimulációs modell is a különböző forgalomfolyamok szimulációjára összpontosít. A forgalomfolyamok szimulálhatók egyrészt a forgalom feltételezett jellemzőinek megadásával, vagy pedig a tanulmányozott alkalmazás működése során nyert valódi adatok felhasználásával megalapozva. Ez utóbbit a következő alfejezetben fogjuk tárgyalni.

A hálózatmodellezők általában a hálózatról meglévő adatok elemzésével kezdik a modellezést, hogy ezáltal képet kapjanak a hálózat jellemzőiről. Ez segíti az alkalmazási szinten lévő folyamatok elég mély megértését ahhoz, hogy az egyes hálózatelemeket a modellezési konstrukciókhoz tudják rendelni. Különféle eszközök is használhatók a modellépítés előtt. Az előzetes elemzés után a modellező figyelmen kívül hagyhatja az olyan folyamatokat és eseményeket, amelyek nem fontosak a szóban forgó tanulmányban. Például, az adatbázis tranzakciókról nyert adatok a kerethossz nagy változékonyságát mutatják. Az 5.9. ábra segít elképzelni ezeket anomáliákat.

Ugyanezeknek az adatoknak a vizsgálata (19.1. ábra) a keretek beérkezési időközeinek nagy változékonyságát is felfedi.

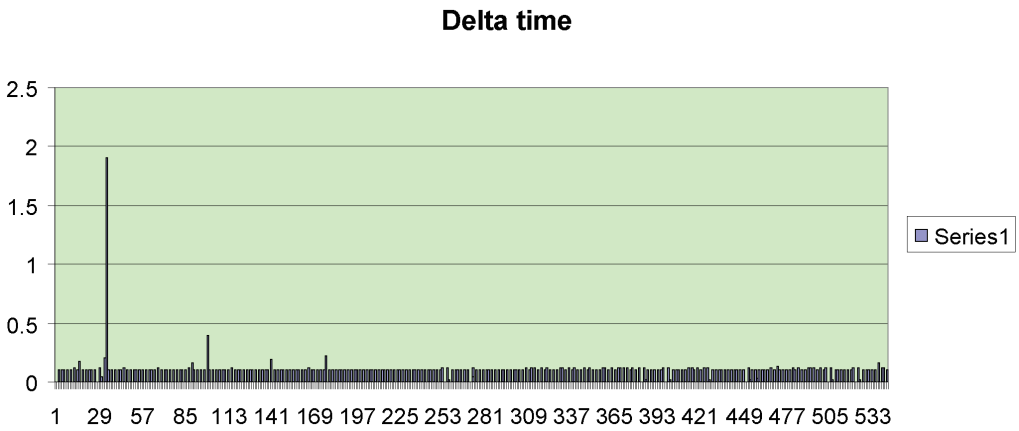
A kumulatív valószínűségeloszlás-függvény közelítésével például a kerethosszak hisztogramja (5.11. ábra) segíti a modellezőt az eloszlástípus meghatározásában.

5.6. Szimulációs modellező rendszerek

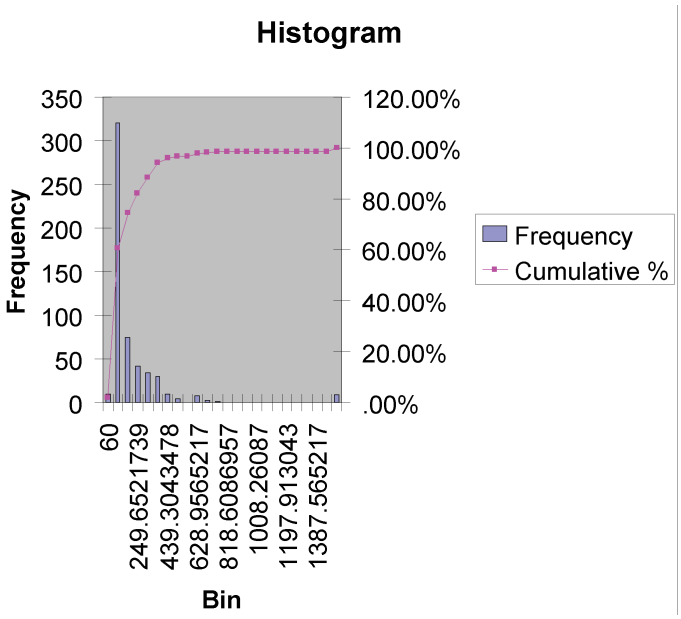
5.6.1. Adatgyűjtő eszközök és hálózatelemzők

Ebben az alfejezet összefoglaljuk a széleskörűen alkalmazott OPNET és COMNET diszkrét-esemény szimulációs eszközök fő jellemzőit, és az ezeket támogató hálózatelemzőket, a Network Associates Sniffer-ét és a az OPNET Alkalmazásjellemző Környezetét.

Az OPNET („Optimized Network Engineering Tools” – *Optimalizált hálózattervező eszközök*) egy széleskörűen alkalmazható szimulációs rendszer, mely alkalmas kommunikációs hálózatok és elosztott rendszerek modellezésére részletes protokoll és teljesítmé-



5.10. ábra. A beérkezési időközök nagy változékonysága. Az ábra színes változata megtalálható a 807. oldalon.



5.11. ábra. A kerethosszak hisztogramja. Az ábra színes változata a 807. oldalon látható.

nyelemzéssel. Az OPNET számos eszközt tartalmaz, melyek a modellezési és szimulációs projektek egyes szakaszainak megfelelően három kategóriába sorolhatók. Ezek a szakaszok a következők: modellspecifikáció, adatgyűjtés és szimuláció, valamint az elemzés.

5.6.2. Modellspecifikáció

A modellspecifikáció során a modellező a tanulmányozott hálózati rendszer egy reprezentációját hozza létre. Az OPNET támogatja a modellek újrafelhasználását, azaz a modellek alapozhatók olyan beágyazott modellekre, melyeket korábban készítettek el és modellkönyvtárakban tároltak. A specifikációszerkesztőkkel a modellek különböző szintű részletezettséggel adhatók meg. Ezek a szerkesztők az aktuális hálózati rendszer hierarchikus struktúrájának megfelelően osztályokba sorolják az modellezési információkat. A legmagasabb szintű szerkesztő, a *Projektszerkesztő* alakítja ki a hálózati topológiák, alhálózatok, és kapcsolatok modelljeiből álló hálózatmodelleket, melyeket pedig a *Csomópontszerkesztővel* adhatunk meg. A Csomópontszerkesztő írja le a csomópontok belső szerkezetét, funkcionális elemeiket és a közöttük lévő adatfolyamokat. A csomópontok olyan folyamatmodell-modulokból állnak, melyeket a *Folyamatszerkesztővel* adhatunk meg. A hálózat hierarchiájának legalacsonyabb szintjén a folyamatmodellek írják le az adott modul viselkedését a protokollokkal, algoritmusokkal és alkalmazásokkal kapcsolatban, véges automatákat és egy magasszintű nyelvet felhasználva.

Számos más szerkesztő is elérhető a folyamat- vagy csomópont-szintű modellek által hivatkozott különböző adatmodellek definiálására, mint amilyenek például a csomagformátumok és a folyamatok közötti vezérlőinformációk. További szerkesztőkkel készíthetünk, módosíthatunk vagy csak megtekinthetünk különböző sűrűségfüggvényeket, melyekkel különböző eseményeket irányíthatunk. Ilyen például a csomagok küldése vagy fogadása között eltelt idő meghatározása. A modellspecifikáció-szerkesztők egy grafikus felületet biztosítanak a felhasználónak, mellyel változtathatja a modelleket ábrázoló objektumokat és a megfelelő folyamatokat. Mindegyik szerkesztő a modell egy adott absztrakciós szintjének megfelelő objektumokat és műveleteket adhatja meg. Ezért a Projektszerkesztő adja meg a hálózat csomópontjait és kapcsolatait, a Csomópontszerkesztő a processzorokat, sorokat és a hálózat csomópontjaiban lévő adó és fogadó egységeket, a Folyamatszerkesztő pedig a folyamatok állapotait és átmeneteit. A 5.12. ábra az egyes szerkesztők absztrakciós szintjeit szemlélteti.

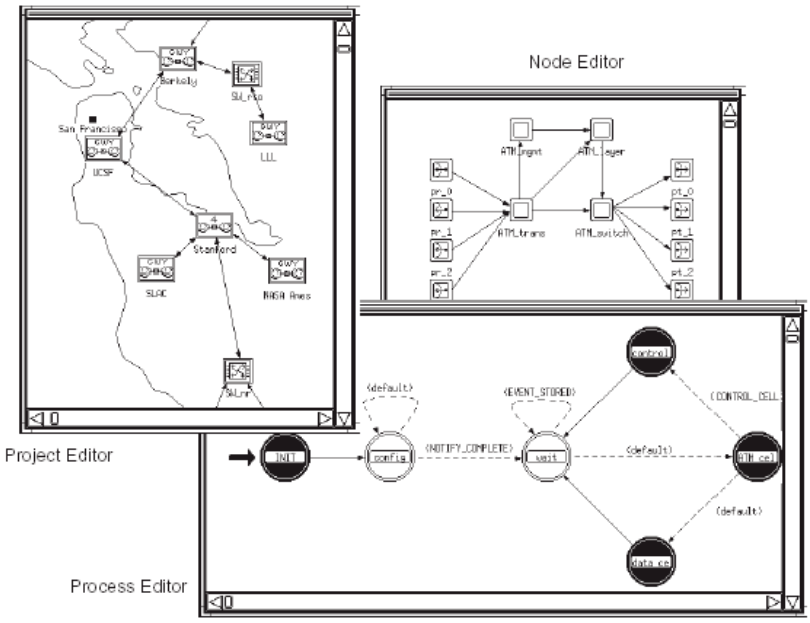
5.6.3. Adatgyűjtés és szimuláció

Az OPNET a szimuláció alatt többféle kimenetet tud készíteni attól függően, hogy a modellező ezt hogyan definiálja. A modellezők legtöbb esetben beépített adattípusokat is használhatnak: kimenetvektorokat és skalárokat, valamint animációkat.

- A kimenetvektorok idősorozatok szimulációs adatait reprezentálják. Ezek a vektorok idő-érték párokat tartalmazó bejegyzések listájából állnak. A bejegyzések első értéke tekinthető a független, a második pedig a függő változónak.

- A skaláris statisztikák a szimuláció alatt gyűjtött statisztikákból származtatott egyedi értékek. Ilyen például az átlagos átviteli ráta, az eldobott cellák számának maximuma, az átlagos válaszidő és más egyéb statisztikák.

- Az OPNET a szimuláció alatt vagy az után megtekinthető animációkat is tud készíteni. A modellező definiálhat többféle animációt is, például csomagfolyamokat, állapotátmeneteket és statisztikákat.



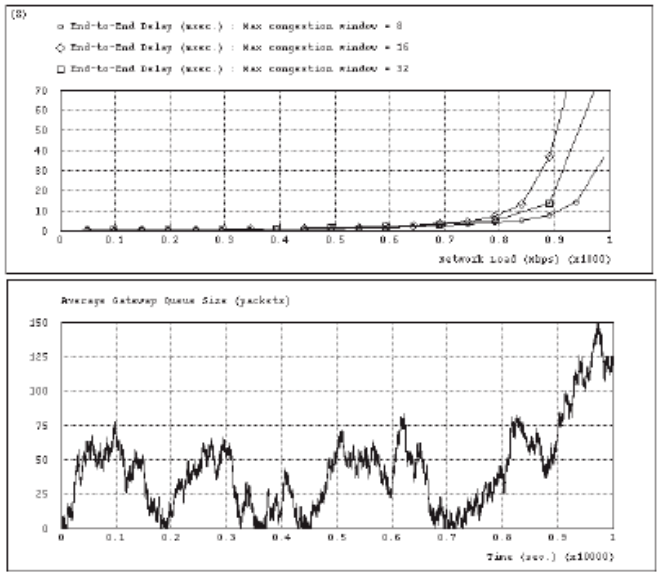
5.12. ábra. A három absztrakciós szintet a Projekt-, Csomópont- és Folyamatszerkesztők adják meg.

5.6.4. Elemzés

A szimuláció alatt összegyűjtött adatok többsége kimenetvektor és skalár állományokban tárolódik. Ezeknek az adatoknak az elemzésére az OPNET egy *Elemzőeszköz* nevű segédeszközt nyújt, amely ábrázoló és numerikus feldolgozó-funkciók gyűjteménye. Az Elemzőeszköz az adatokat grafikonok és nyomkövetések formájában jeleníti meg. A nyomkövetések az X és Y tengelyek értékpárjainak listáját tartalmazzák. Tárolásukra és megjelenítésükre elemzőtáblákat használ. Az Elemzőeszköz a szimulációs eredmények feldolgozására és új nyomkövetések készítésére módszerek széles választékát támogatja. Ebben benne van a hisztogramok, sűrűség és kumulatív eloszlásfüggvények, valamint a konfidencia intervallumok számítása is. Támogatja továbbá a matematikai szűrők használatát is a vektor vagy nyomkövetési adatok feldolgozásához. A matematikai szűrők előre definiált számításokra, valamint statisztikai és aritmetikai operátorokra alapozott hierarchikus blokkdiagrammokként vannak definiálva. A következő két ábra (5.13. és 5.14.) az Elemzőeszköz által készített grafikonokat szemlélteti.

A 5.14. ábrán látható Elemzőeszköz négy grafikont jelenít meg egyszerre.

Népszerű diszkrét-esemény szimulációs rendszer a COMNET is, melyet majd a 5.9. alfejezetben fogunk röviden tárgyalni.



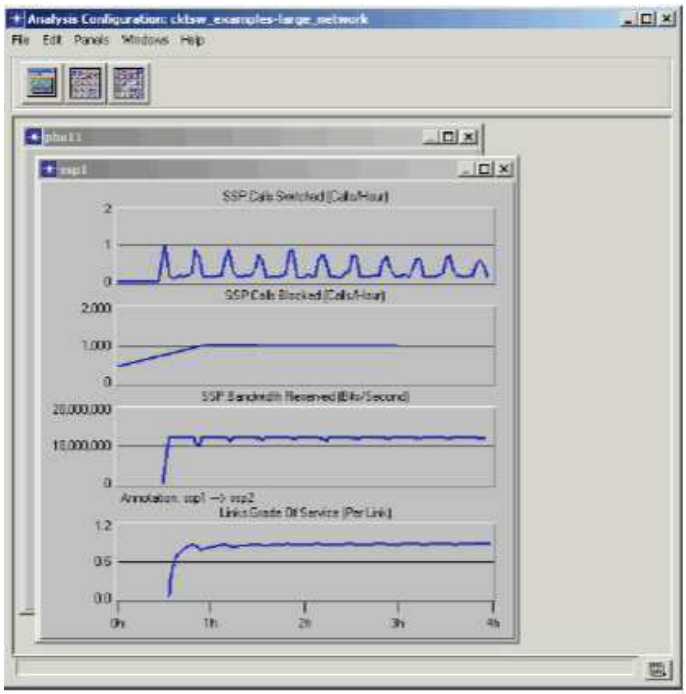
5.13. ábra. Egy példa a skaláris adatok (felső grafikon) valamint vektor adatok (alsó grafikon) grafikus megjelenítésére.

5.6.5. Hálózatelemzők – az Alkalmazásjellemző Kör0nyezet

Egyre nagyobb az érdeklődés az alkalmazás életciklusán keresztüli, a fejlesztéstől az üzembe helyezésig tartó teljesítmény-előrejelzésre, mérésre, modellezésre és megállapításra. Az alkalmazások teljesítményének előrejelzése különösen fontos olyan alkalmazási területeken, mint amilyen például az elektronikus kereskedelem. Az egyre inkább versengő elektronikus kereskedelemnél az alkalmazás teljesítménye különösen fontos lehet ott, ahol nagyon szoros a verseny. A teljesítmény így a bevételekre is hatással van. Ha egy alkalmazás gyengén teljesít, akkor az alkalmazás helyett inkább mindig a hálózatot hibáztatják. Ezeket a teljesítményproblémákat sokminden okozhatja, például az alkalmazás tervezési hibája vagy a lassú adatbázisszerverek. Az *Alkalmazásjellemző környezethez* („Application Characterization Environment” – ACE) és a Network Associates Sniffer-jéhez hasonló eszközök használatával a modellezők módszertanokat fejleszthetnek ki arra, hogy azonosítani tudják az alkalmazás-lelassulások forrását és megoldják az azokat előidéző problémákat. Az alkalmazás elemzése után a modellezők javaslatokat tehetnek a teljesítmény optimalizálására, aminek eredményei gyorsabb alkalmazások és jobb válaszidők lehetnek.

Az Alkalmazásjellemző környezet a hálózati alkalmazások szemléltetésére, elemzésére és hibáinak elhárítására alkalmas eszköz. A hálózatok szervezői és alkalmazásfejlesztői a következőkre használhatják.

- Hálózatok és alkalmazások szűk keresztmetszetének meghatározására.
- Hálózati és alkalmazásproblémák felderítésére.



5.14. ábra. Négy grafikont megjelenítő Elemzőeszköz.

- Várható hálózati módosításoknak a meglévő alkalmazások válaszidejeire gyakorolt hatásának elemzésére.
- Az alkalmazásoknak változó konfigurációk és hálózati feltételek melletti teljesítményének előrejelzésére.

Egy alkalmazás teljesítményét a hálózat jellemzői határozzák meg, melyeket különböző komponensek befolyásolnak. A következő lista néhány ilyen tulajdonságot és a kapcsolódó hálózatelemeket tartalmazza:

- *Hálózati közeg*
  - Sávszélesség (torlódás, erős ingadozás)
  - Késleltetés (TCP ablakméret, nagy késleltetésű eszközök, csevegő alkalmazások)
- *Csomópontok*
- *Kliensek*
  - A felhasználók számára szükséges idő
  - Feldolgozási idő
  - Kiéheztetés
- *Kiszolgálók*

  - Feldolgozási idő
  - Többrétegű várakozó adatok
  - Kiéheztetés

- *Alkalmazások*

  - Alkalmazás-fordulók (túl sok forduló – csevegő alkalmazások)
  - Szálak (egy és többszálúság)
  - Adat profil (erősen ingadozó, túl sok adatfeldolgozás)

Az alkalmazások elemzése két fázist igényel:

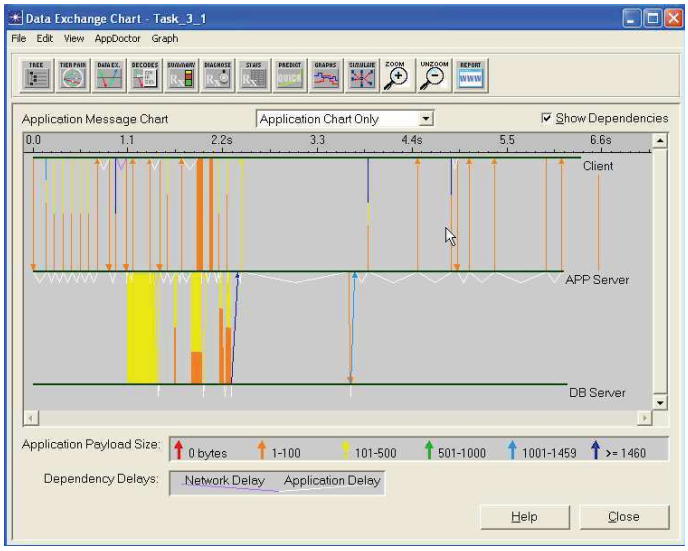
- Az alkalmazás futása közben csomag-nyomkövetések készítése ahhoz, hogy egy alapkonfigurációs modellt készíthessünk az alkalmazás modellezéséhez. Ehhez használhatjuk az ACE vagy bármely más hálózatelemzők eszközeit. Ezek a nyomkövetések stratégiailag telepített ügynök-alkalmazásokkal készíthetők el.
- A nyomkövetés-állomány importálásával az alkalmazás tranzakcióinak ábrázolása, amit alkalmazásfeladatnak nevezünk, az alkalmazás által generált üzenetek és protokoll adategységek további elemzése céljára.

Az alkalmazásfeladat elkészítése után a következő műveleteket végezhetjük el a forgalom nyomkövetési eredményein:

- A csomag-nyomkövetés eredményeinek megjelenítése és szerkesztése a protokollverem különböző szintjein, külön ablakokban. Ezeket az ablakokat arra is használhatjuk, hogy eltávolítsuk vagy töröljük az alkalmazásfeladat különböző részeit. Így a minket érdeklő tranzakciókra tudjuk fordítani a legnagyobb figyelmet.
- Alkalmazási szintbeli elemzés elvégzése a szűk keresztmetszetek felismerésére. Külön megmérhetjük a válaszidő egyes összetevőit, mint például az alkalmazási szinten eltöltött időt, a feldolgozási időt és a hálózati adattovábbítás idejét, továbbá részletes statisztikákat tekinthetünk meg a hálózatról és az alkalmazásról. A csomag-nyomkövetések eredményei alapján elemezhetjük a hálózati és alkalmazásprotokollok adategységeit is.
- Az alkalmazás teljesítményének előrejelzése különböző módosítási ötletek és tervezett változtatások esetén.

A következőkben a részletekbe bocsátkozás nélkül illusztráljuk egy egyszerű háromrétegű alkalmazáson keresztül az előbb említettek jellemzőit. Egy távoli alkalmazásszerverhez hozzáférő kliens (ami adatbázis szervertől igényel információkat) esetén szeretnénk meghatározni a nagy válaszidő okait. A kliens egy ADSL vonalon csatlakozik az Internetre, az alkalmazásszerver és az adatbázis szerver között pedig egy 100 Mbps sebességű Ethernet kapcsolat van. Azonosítani akarjuk a nagy válaszidő okát, valamint megoldási lehetőségeket szeretnénk ajánlani. Ehhez nyomkövetési ügynököket telepítünk a kliens és az alkalmazásszerver valamint a két szerver közé. Az ügynökök egyidejűleg mindkét helyen gyűjtik az információkat a tranzakciók alatt. Ezután a nyomkövetési információk egyesíthetők és szinkronizálhatók, hogy ezáltal a hálózat és az egyes rétegek késleltetésének a lehető legjobb elemzési lehetőségét tudják nyújtani.

A nyomkövetési információknak az ACE-ba való importálása után a tranzakciókat az *Adatcsere diagramon* („Data Exchange Chart”) elemezzük, ami ábrázolja az alkalmazás üzeneteinek a rétegek közötti továbbítását (lásd a 19.4. ábrát).



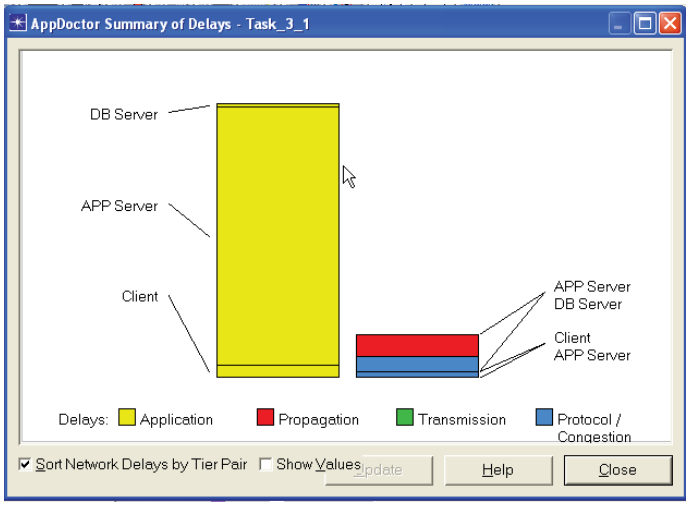
5.15. ábra. Adatcsere diagram. Az ábra színes változata megtalálható a 808. oldalon.

Az Adatcsere diagram megmutatja a kliens és a szerverek között átvitt különböző méretű csomagokat. A tranzakció teljes válaszideje körülbelül 6 másodperc. A „Függőségek megjelenítése” jelölőnégyzet bejelölése esetén fehér vonalak jelzik a nagy feldolgozási késleltetéseket az alkalmazásszerver és a kliens rétegeknél. További elemzés céljára készíttethetünk egy „Késleltetések összegzése” ablakot, melyben az alkalmazás válaszideje a következő négy általános kategóriára van osztva: az alkalmazás késleltetése, terjedési késleltetés, átviteli késleltetés és protokoll/torlódás miatti késleltetés (lásd az 5.17. ábrát). Ennek a diagramnak a segítségével megfigyelhetjük az alkalmazáshoz és a hálózathoz kapcsolódó késleltetések viszonyát a kliens és a szerverek közötti tranzakció alatt. Látható, hogy az alkalmazás késleltetése jóval több, mint a terjedési, átviteli és a protokoll/torlódás miatti késleltetés.

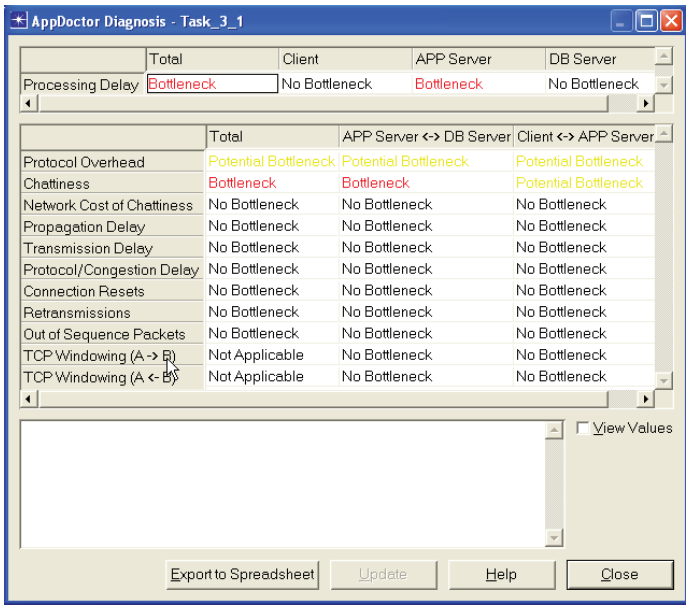
A „Diagnózis” funkció (5.17. ábra) a lehetséges szűk keresztmetszeteknek egy sokkal részletesebb elemzési lehetőségét biztosítja olyan tényezők elemzésével, melyek gyakran okoznak teljesítményproblémákat a hálózati alkalmazásokban. Az egy adott küszöb feletti értékek szűk keresztmetszetként, vagy lehetséges szűk keresztmetszetként vannak megjelölve.

A tranzakció diagnózisa igazolja, hogy az elsődleges szűk keresztmetszet az alkalmazásszerver feldolgozási késleltetése miatt van. A feldolgozási késleltetést lassú állomány input/output, CPU feldolgozás vagy memóriahozzáférés okozhatja. Az elemzés egy másik szűk keresztmetszetet is felfed, ami az alkalmazás csevegőssége. Ez tehát a következő feladatunk. Az alkalmazás viselkedését az alkalmazás-fordulókkal kapcsolatban vizsgáljuk, amihez a tranzakció-statisztikákból juthatunk hozzá. Egy alkalmazás-forduló az alkalmazás-üzenetfolyam irányának megváltozását jelenti.

A tranzakció statisztikáiból (5.18. ábra) kiderül, hogy az alkalmazás-fordulók száma nagy, azaz a tranzakció által egy időben küldött adatok mérete kicsi. Ez jelentős alkal-

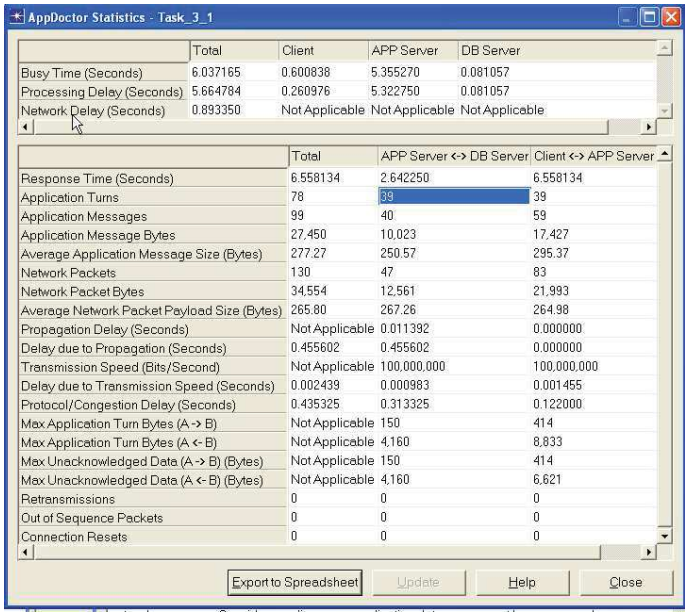


5.16. ábra. Késleltetések összegzése. Az ábra színes változata a 808. oldalon látható.



5.17. ábra. Diagnózis ablak.

mazásbeli és hálózati késleltetéseket okozhat. Továbbá, az alkalmazás feldolgozási idejének jelentős része telhet el a sok kérés és válasz feldolgozásával. A Diagnózis ablak egy „Csevegősség” szűk keresztmetszetet jelez „Csevegősség hálózati költsége” szűk keresztmetszet nélkül, ami a következőket jelenti:



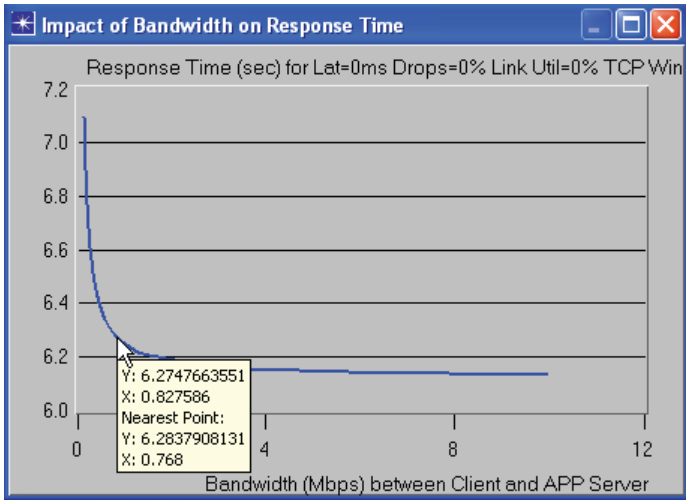
5.18. ábra. Statisztikák ablak.

- Az alkalmazás nem generál jelentős hálózati forgalmat a csevegősség miatt.
- Az alkalmazás jelentős feldolgozási késleltetést okoz a sok kis alkalmazási szintbeli kérés és válasz kezelése miatt.
- Az alkalmazás „Csevegősség hálózati költsége” drasztikusan nőhet egy nagyobb késleltetésű hálózatban.

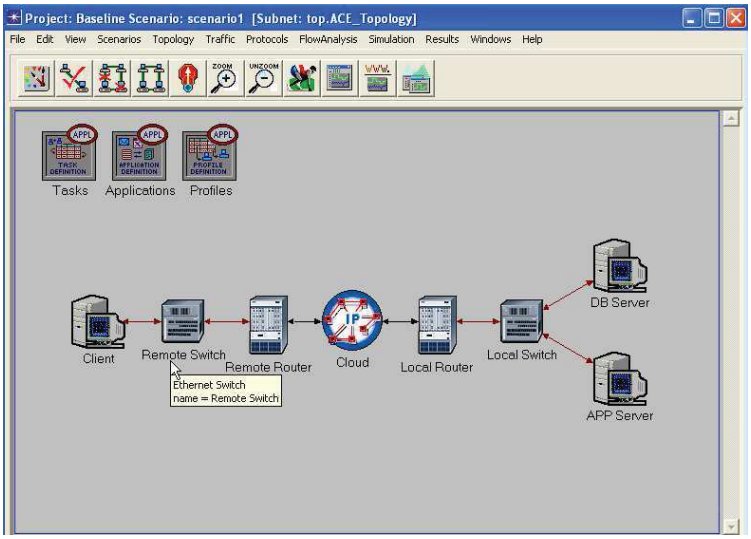
Azt lehet ajánlani, hogy az alkalmazásnak kevesebb és nagyobb üzeneteket kellene küldenie. Ezzel hatékonyabban használná ki a rétegek és a hálózat erőforrásait. Például, egy adatbázis alkalmazásnak nem szabad egy rekordhalmazt rekordonként egyesével elküldenie.

Vajon jelentősen csökkenne-e a válaszidő akkor, ha nagyobb sávszélességet adnánk (**??**. ábra) a kliens és az alkalmazásszerver közötti kapcsolathoz? Ennek a kérdésnek a megválaszolása azért fontos, mert a nagyobb sávszélesség sokba kerül. A becslés funkciót felhasználva megválaszolhatjuk ezt a kérdést. A következő diagramon a sávszélességet 128 Kbps-ról 10 Mbps-ra növeltük. Látható, hogy körülbelül 827 Kbps után nincs jelentős javulás a válaszidőben, azaz az ajánlott legnagyobb sávszélesség ennél az alkalmazásnál nem nagyobb, mint 827 Kbps, ami pedig biztosítható egy nagyobb sebességű DSL vonallal is.

Az alkalmazás teljesítményének elemzése után a forgalom nyomkövetési adataiból azonnal el tudjuk készíteni az induló alapkonfigurációs modellt további szimulációs célokra, amit a 5.20. ábra szemléltet.



5.19. ábra. A sávszélesség hozzáadásának hatása a válaszidőre. Az ábra színes változata a 809. oldalon látható.



5.20. ábra. Alapkonfigurációs modell további szimulációs tanulmányokra.

5.6.6. Sniffer

Egy másik népszerű hálózatelemző a Network Associates cég *Sniffer* nevű terméke. (A Network Associates ezt nemrég nevezte át Netasyst-re.) Ez egy hatékony hálózatmegjelenítő eszköz, ami a következő feladatok megoldását teszi lehetővé:

- A hálózati forgalom részletes elemzés céljára történő nyomkövetését.

- Az „Expert Analyzer” segítségével problémák diagnosztizálását.
- A hálózati tevékenységek valós idejű megfigyelését.
- Részletes kihasználtság és hibastatisztikák gyűjtését az egyes állomásokról, párbeszédekről vagy a hálózat bármely részéről.
- A korábbi kihasználtság és hibainformációk mentését alapkonfigurációs elemzésre.
- Látható és hallható riasztások létrehozását, továbbá a hálózat adminisztrátorainak értesítését problémák észlelése esetén.
- A hálózat aktív eszközökkel való vizsgálatát forgalomszimulációs célokra, válaszidő mérésre, hop-ok számlálására és problémák elhárítására.

5.7. Modellfejlesztési életciklus

5.7.1. Egy keretrendszer hálózatmodellezők számára

A hálózatmodellezésnek számos megközelítése van. Egy lehetséges módszer egy modell készítése a hálózati topológia alapján a hálózati forgalom statisztikai közelítésével. Néhány módosítás után a modellező meg tudja vizsgálni néhány paraméter változtatásának a hálózat vagy az alkalmazás teljesítményére gyakorolt hatását. Ennél a megközelítésnél fontosabb a teljesítménykülönbségek vizsgálata annál, hogy egy olyan modellt használjunk, ami valós hálózati forgalmon alapszik. Például, bizonyos kliens-szerver tranzakciókat feltételezve meg akarjuk mérni a válaszidő változását a kapcsolat-kihasználtság függvényében. Ebben az esetben nem különösebben fontos, hogy egy a valódi hálózati forgalmon alapuló modellt alkalmazzunk. Elég megadni egy gyakori felhasználó vagy tervező által becsült adatforgalom mennyiségét, majd ezen adatmennyiség esetén vizsgáljuk meg, hogy a válaszidő mennyivel nő, amikor a kapcsolat-kihasználtság növekszik az eredetihez képest.

A hálózatmodellezés leggyakrabban használt megközelítése a megelőző hálózatszervezés módszertanát követi. Ez magában foglalja az valós hálózati forgalmon alapuló hálózati modell készítését a hálózat jelenlegi és a jövőbeni viselkedésének szimulálására és az új alkalmazások hozzáadásának hatásainak előrejelzésére. A modellező és szimulációs eszközök alkalmazásával a hálózatszervezők változtatni tudnak a modellen, új berendezéseket, munkaállomásokat, kiszolgálókat és alkalmazásokat adhatnak hozzá. Ezen kívül az egyes kapcsolatokat nagyobb sebességűekre cserélhetik, és tesztelhetik őket a hálózat tényleges megváltoztatása előtt. A következőkben ezt az egyetemek, vállalatok és az ipar által széleskörűen elfogadott megközelítési módot követjük. A következő bekezdésekben a ***modellfejlesztési életciklusnak*** nevezett modellezési lépéseket tekintjük át, amelyeket már több nagyméretű vállalati hálózat modellezésében alkalmaztunk. A modellfejlesztési életciklus tehát a következő lépésekből áll:

- A hálózat topológiájának és komponenseinek azonosítása.
- Adatgyűjtés.
- Az alapkonfigurációs modell elkészítése és érvényesítése, és ennek felhasználásával szimulációs tanulmányok végzése.
- Az alkalmazásmodell elkészítése az alkalmazások által generált forgalom részleteinek felhasználásával.

- Az alapkonfigurációs és az alkalmazásmodell integrációja valamint a szimulációs tanulmányok befejezése.
- További adatgyűjtés a hálózat növekedése és változásai után, továbbá miután további részleteket tudtunk meg az alkalmazásokról.
- Ezeknek a lépéseknek a megismétlése.

A következőkben a fenti lépéseket részletezzük:

A hálózat topológiájának és komponenseinek azonosítása
A topológia a hálózat fizikai komponenseit (forgalomirányítók, áramkörök és kiszolgálók) és ezek kapcsolatát írja le. Tartalmazza az egyes hálózati berendezések helyét és konfigurációjuk leírását, azt hogy milyen típusú és sebességű áramkörökkel vannak összekapcsolva, a LAN-ok és WAN-ok típusát, a kiszolgálók helyét, a címzési módokat, az alkalmazások és protokollok listáját stb.

Adatgyűjtés
Az alapkonfigurációs modell felépítéséhez szükség van a topológia és forgalom adataira. A modellezők a topológia jellemzőit megadhatják manuálisan, vagy pedig hálózatmenedzselő eszközök és a hálózati berendezések konfigurációs állományainak felhasználásával. Számos hálózatmenedzselő eszköz használja az *Egyszerű hálózati menedzsment protokollt* („Simple Network Management Protocol" – SNMP), mellyel a forgalomirányítókban és más berendezésekben futó SNMP ügynökök által karbantartott *Menedzsment információs adatbázisból* („Management Information Base" – MIB) lehet lekérdezni. Ezt a folyamatot nevezik SNMP hálózat-feltérképezésnek. A kérdéses hálózat topológiájának felméréséhez topológia-adatokat importálhatunk a forgalomirányítók konfigurációs állományaiból. Egyes teljesítménymenedzselő eszközök hálózatmenedzsment platformok (például HP OpenView vagy IBM NetView) térkép állományaiból is tudnak adatokat importálni. Ezek exportálási funkcióját alkalmazva a kapott térkép fájl importálható lesz a modellezéshez.

Az alapkonfigurációs modellhez szükséges forgalmi adatok különböző forrásokból származhatnak: interjúkból és hálózati dokumentumokból származó forgalomleírásokból, tervezési vagy karbantartási dokumentumokból, MIB/SNMP jelentésekből és hálózatelemző vagy *Távoli megfigyelő* („Remote Monitoring" – RMON) nyomkövetési eredményekből. Az RMON egy hálózatmenedzselő protokoll, ami lehetővé teszi a hálózati információk gyűjtését az egyes csomópontokban. Az RMON nyomkövetési eredményeit RMON szondák gyűjtik, melyek szabványuktól függően a hálózati architektúra különböző szintjein gyűjtik az adatokat. A 5.21. ábra a leggyakrabban használt szabványokat és adatgyűjtési szinteket foglalja össze.

A hálózati forgalom osztályozható használat és alkalmazás alapú adatokként. A legfontosabb különbség a két osztály között az, hogy milyen fokú részleteket biztosítanak az adatok, valamint hogy milyen következtetések vonhatók le belőlük. A felosztás egyértelműen megadható két egymás melletti OSI réteg, a szállítási és a viszonyréteg segítségével. A használat alapú adatok a szállítási réteggel kapcsolatos teljesítménykérdésekkel kapcsolatos vizsgálatokra szolgálnak, az alkalmazás alapú adatok pedig a hálózati architektúra szállítási rétege fölötti rétegekkel kapcsolatos elemzésekre. (Intenet terminológiában ez ekvivalens a TCP szint és az e fölötti alkalmazási szint közötti vágással.)

	RMON1	RMON2	Enterprise RMON
Ethernet/Token Ring	X	X	X
MAC réteg megfigyelése	X	X	X
hálózati réteg megfigyelése		X	X
alkalmazási réteg megfigyelése		X	X
kapcsolt LAN, Frame Relay, ATM			X
VLAN támogatása			X
alkalmazás válaszideje			X

5.21. ábra. Az RMON szabványok összehasonlítása.

A használat alapú adatok gyűjtésének célja a teljes forgalom méretének meghatározása az alkalmazásoknak a hálózaton történő megvalósítása előtt. Ezek az adatok a forgalomirányítókban vagy egyéb hálózati berendezésekben lévő SNMP ügynököktől gyűjthetők össze. A forgalomirányítókhoz vagy kapcsolókhoz küldött SNMP kérdések statisztikákat biztosítanak az egyes LAN interfészeken, WAN áramkörökön vagy permanens virtuális áramkör („Permanent Virtual Circuit" – PVC) interfészeken keresztül küldött bájtok pontos számáról. Ezen adatok segítségével kiszámíthatjuk az egyes áramkörökön elérhető sávszélesség kihasználtságát.

Az alkalmazás alapú adatok gyűjtésének célja pedig az egy alkalmazás által generált adatok mennyiségének és az alkalmazás igényeinek a típusának meghatározása. Ez lehetővé teszi a modellezők számára az alkalmazás viselkedésének megértését és az alkalmazási szintbeli forgalom jellemzését. A forgalomelemzőkből, RMON2 kompatibilis szondákból, a Sniffer-ből vagy a NETScout Manager-ből származó adatok különböző részleteket biztosítanak az alkalmazás hálózati forgalmáról. A stratégiailag elhelyezett adatgyűjtő berendezések elég adatot tudnak gyűjteni ahhoz, hogy világosan lássuk az alkalmazás forgalmának viselkedését. A forgalomelemzők tipikusan a következő alkalmazási szintbeli adatokat gyűjtik:

- Az alkalmazások típusait.
- A hálózati rétegbeli címmel (azaz IP címmel) rendelkező hosztokat.
- Két hoszt közötti hálózati párbeszédek hosszát (a kezdés és a befejezés időpontja).
- Az egyes párbeszédek során mindkét irányban az elküldött bájtok számát.
- A párbeszédek során mindkét irányban a csomagok átlagos méretét.
- A forgalom erős ingadozását.
- Csomaghossz eloszlásokat.
- Csomag-beérkezési időköz eloszlásokat.
- Csomagtovábbítási protokollokat.
- Forgalom profilt, azaz üzenet és csomag méreteket, beérkezési időközöket és feldolgozási késleltetést.
- Egy alkalmazásnak egy tipikus felhasználó általi használatának gyakoriságát.
- A résztvevő csomópontok fontosabb kölcsönhatásait, események sorozatait.

Az alapkonfigurációs modell elkészítése és érvényesítése, és ennek felhasználásával szimulációs tanulmányok végzése
Az alapkonfigurációs modell készítésének célja az, hogy a vizsgált hálózat pontos modelljét adjuk meg. Ez a modell a hálózat jelenlegi állapotát tükrözi. A tanulmányok pedig az ezen a modellen elvégzett módosítások hatásait mérik fel. A modell könnyen érvényesíthető, ugyanis az előrejelzéseinek összhangban kell lenniük az aktuális hálózaton végzett mérésekkel. Az alapkonfigurációs modell általában csak olyan alapvető teljesítmény-mértékeket jelez elő, mint az erőforrás kihasználtság és a válaszidő.

Az alapkonfigurációs modell a topológiának és a korábban összegyűjtött használat alapú forgalomadatoknak az egyesítésével jön létre. Ezt érvényesíteni kell az aktuális hálózat teljesítmény-paramétereivel, azaz igazolni kell, hogy a modell a valós hálózati működéshez hasonlóan működik. Az alapkonfigurációs modell használható az aktuális hálózat elemzésére, és szolgálhat a további alkalmazás és kapacitástervek alapjául is. A modellezőeszközök importálási funkcióját alkalmazva a modellkészítés kezdhető az életciklus adatgyűjtési lépésében összegyűjtött topológiaadatok importálásával. A különböző hálózatmenedzselő rendszerek, mint például a HP OpenView vagy a Network Associate Sniffer-je, a topológia adatait általában topológia-állományokban (.top vagy .csv) tárolják. A forgalomadatokat tartalmazó állományok a következőképpen csoportosíthatók:

- Párbeszédek résztvevőinek kommunikációjával kapcsolatos állományok, melyek összegyűjtve tartalmazzák a hálózat terheltségének információit, a hosztneveket, a küldött csomagok és bájtok számát az egyes hosztpárok esetén. Az adatok lehetővé teszik a modellezőeszköz számára, hogy megtartsa a forgalom erősen ingadozó voltát. Ezek az állományok különböző adatgyűjtő eszközök segítségével nyerhetők.
- Esemény-nyomkövetési állományok, melyek összegzett információk helyett az egyes párbeszédekhez kapcsolódó hálózatterheltségéről tartalmaznak információkat. A szimuláció során az állomány alapján eseményről eseményre lejátszhatók a hálózati tevékenységek.

A szimuláció előtt a modellezőnek a következő szimulációs paraméterekkel kapcsolatban kell döntenie:

- *Futási idő.* A ***futási időnek*** nagyobbnak kell lennie, mint a leghosszabb üzenetnek a hálózatban való késése. Ezen idő alatt a szimulációnak elegendő számú eseményt kell létrehoznia, hogy a modell elég mintát tudjon generálni minden eseményből.
- *Felmelegedési periódus.* A szimuláció ***felmelegedési periódusa*** az az idő, ami a csomagok, pufferek, üzenetsorok, áramkörök és a modell különböző egyéb elemeinek inicializálásához kell. A felmelegedési periódus megegyezik egy tipikus üzenet hosztok közötti késleltetésével. A szimuláció felmelegedési ideje azért szükséges, hogy biztosítsuk az egyensúlyi állapot elérését az adatgyűjtés megkezdése előtt.
- *Többszöri ismétlések.* Olyan esetekben, amikor a statisztikák nem eléggé közelítik a valós értékeket, szükség lehet egy adott modell többszöri lefuttatára is. Szintén többszöri ismétlésekre van szükség az érvényesítés előtt, amikor több másolatot futtatunk le azért, hogy meghatározzuk a statisztikáknak a másolatok közötti ingadozását. Ennek okai leggyakrabban a ritka események.
- *Konfidenciaintervallum.* A konfidenciaintervallum segítségével becsüljük meg a populáció-paraméter valószínűsíthető méretét. Ez megad egy becsült értéktartományt,

ami egy adott valószínűséggel tartalmazza a becsült paramétert. A leggyakrabban használt intervallumok a 95% és 99% konfidenciaintervallumok, melyek 0.95 és 0.99 valószínűséggel tartalmazzák az adott paramétert. A szimulációban a konfidenciaintervallum egy mutatót biztosít a szimulációs eredmények pontosságára nézve. A kevesebb ismétlés szélesebb konfidenciaintervallumot és kisebb pontosságot eredményez.

Több modellezőeszközben a topológia és forgalomadatokat tartalmazó állományok importálása után az alapkonfigurációs modell automatikusan elkészül. Ezt ellenőrizni kell a konstrukciós hibák elkerülése végett, majd érvényesíteni a következő lépések végrehajtásával:

- Egy előzetes futtatással ellenőrizni kell, hogy minden forrás-cél pár jelen van-e a modellben.
- Egy felmelegedési periódust is tartalmazó hosszabb szimulációval meg kell mérni a küldött és fogadott üzenetek számát és a kapcsolatok kihasználtságát, hogy ezzel igazoljuk a megfelelő méretű forgalommennyiség átvitelét.

Az alapkonfigurációs modell érvényesítése igazolja, hogy a szimuláció ugyanazokat a teljesítmény-paramétereket szolgáltatja, mint amiket a fizikai hálózaton mértünk. Általában a következő hálózatjellemzők mérhetők mind a modellben, mind pedig a fizikai hálózatban.

- Küldött és fogadott csomagok száma.
- Pufferhasználat.
- Csomagkésleltetés.
- Kapcsolat-kihasználtság.
- Csomópontok CPU kihasználtsága.

A konfidenciaintervallumok és az egymástól független minták száma befolyásolja azt, hogy modell és a valós hálózat között milyen szoros egyezés várható. A legtöbb esetben a legjobb, amit várhatunk, egy átfedés a szimulációval előrejelzett értékek és a mért adatok konfidenciaintervalluma között. A nagyon szoros egyezés elérése túl sok mintát igényelne a hálózatból, és a szimuláció túl sokszori ismétlését.

Az alkalmazásmodell elkészítése az alkalmazások által generált forgalom részleteinek felhasználásával

Az alkalmazásmodelleket akkor tanulmányozzák, ha egy hálózati alkalmazásnak a hálózat teljesítményére gyakorolt hatását, vagy pedig ha a hálózatnak az alkalmazás teljesítményére gyakorolt hatását kell kiértékelni. Az alkalmazásmodellek a hálózati csomópontok között az alkalmazás futása alatt generált forgalomról szolgáltatnak részleteket. Az alkalmazásmodellek építésének lépései hasonlóak az alapkonfigurációs modellnél tárgyaltakhoz:

- Adatok gyűjtése az alkalmazás eseményeiről és a felhasználói profilokról.
- Az alkalmazás adatainak importálása a szimulációs modellbe manuálisan vagy automatikusan.
- A modellezési hibák azonosítása és javítása.
- A modell érvényesítése.

Az alapkonfigurációs és az alkalmazási modell integrációja és a szimulációs tanulmányok befejezése
Az alkalmazásmodell(ek) és az alapkonfigurációs modell integrációja a következő lépések szerint történik:

- Kezdjük a használat alapú adatokból készített alapkonfigurációs modellel.
- Az alkalmazáshasználati esetek információit felhasználva (a felhasználók helye, száma, tranzakciók gyakorisága) meghatározzuk, hogy hová és hogyan töltsük be az alkalmazásprofilokat az alapkonfigurációs modellbe.
- Az előző lépésben generált alkalmazásprofilokat az alapkonfigurációs modellhez adjuk, melyek a tanulmányozott alkalmazás által generált forgalmat ábrázolják.

A szimulációs tanulmányok befejezése a következő lépésekből áll:

- Futtassuk a modellt vagy szimulációt egy modellezési eszköz felhasználásával.
- Elemezzük az eredményeket: hasonlítsuk össze a cél tranzakciók teljesítmény paramétereit a szimuláció kezdetekor felállított célokkal.
- Elemezzük a különböző hálózatelemek kihasználtságát és teljesítményét, különösen ott ahol a célokat nem értük el.

A tipikus szimulációs tanulmányok a következő eseteket tartalmazzák:

- Kapacitáselemzés

  A kapacitás elemzésekor a hálózat paramétereinek változásait tanulmányozzuk, mint például:

  – Felhasználók számának és helyének változása.
  – Hálózatelemek kapacitásának változása.
  – Hálózati technológiák változása.

  A modellezőt érdekelhetik például a különböző változtatásoknak a következő hálózatparaméterekre gyakorolt hatásai:

  – Kapcsolók és forgalomirányítók kihasználtsága.
  – Kommunikációs kapcsolatok kihasználtsága.
  – Pufferkihasználtság.
  – Az újraküldött és az elveszett csomagok száma.

- Válaszidő elemzése

  A válaszidő elemzésének köre az üzenet és csomagtovábbítási késleltetések tanulmányozása:

  – Alkalmazási és hálózati rétegbeli csomagtovábbítási késleltetés.
  – Csomag válaszidő.
  – Üzenet/csomag késleltetések.
  – Az alkalmazás válaszideje.

- Alkalmazáselemzés

  Egy alkalmazással kapcsolatos tanulmányhoz az alkalmazás válaszidő arányának meghatározása tartozik, relatívan az egyes hálózati komponensek és alkalmazások késleltetéséhez. Az alkalmazások elemzése statisztikákat biztosít a hálózat és az alkalmazások különböző teljesítményjellemzőiről, beleértve az előző alfejezetben tárgyaltakat is.

További adatgyűjtés a hálózat növekedése és változásai után, továbbá miután további részleteket tudtunk meg az alkalmazásokról
A következőkben bemutatott fázisnak a célja az, hogy elemezze vagy előrejelezze a hálózat teljesítményét az aktuális feltételek és a hálózat terhelésének megváltozása (új alkalmazások, felhasználók vagy hálózati struktúra) esetén:

- Azonosítsuk a hálózati infrastruktúra azon módosításait, melyek megváltoztatják a hálózati erőforrások kapacitásigényét.
- Az áttervezés tartalmazhatja a megnövekedett vagy lecsökkent kapacitást, a hálózatelemek áthelyezését vagy a megváltozott kommunikációs technológiát.
- Módosítsuk a modellt ezeknek a változásoknak megfelelően.
- Mérjük fel az alkalmazás fejlesztési vagy telepítési terveknek a hálózatra gyakorolt hatását.
- Mérjük fel az üzleti feltételek és tervek (pl. új felhasználók, telephelyek hozzáadása) hatását a hálózatra.
- Használjunk folyamatos mérési technikákat a használati tendenciák, különösen az Internet és intranet használathoz kapcsolódók figyelésére.

5.8. A forgalom ingadozásának hatása nagy sebességű hálózatokra

5.8.1. Bevezetés

A helyi hálózati és nagy kiterjedésű hálózati forgalommal kapcsolatos jelenlegi mérések azt bizonyítják, hogy a széleskörűen elterjedt Markov-folyamat modellek nem alkalmazhatók napjaink hálózati forgalmának leírására. Ha a forgalom Markov-folyamat lenne, akkor a forgalom erős ingadozása hosszú idő átlagolása során kiegyenlítődne, ami ellentétes a forgalom megfigyelt jellemzőivel. A valós fogalommal kapcsolatos mérések azt is bizonyítják, hogy a forgalom erős ingadozása nagyon gyakran jelen van. A gyakran vagy mindig erősen ingadozó forgalom statisztikailag jellemezhető az *önhasonlóság* fogalmának felhasználásával. Az önhasonlóságot gyakran összekapcsolják fraktálgeometriai objektumokkal, melyek a nagyításuktól függetlenül egyformának látszanak. A sztochasztikus folyamatok, idősorok esetében az önhasonlóság kifejezés a folyamat eloszlására utal, amely azonos marad ha különböző időintervallumban nézünk. Az önhasonló idősorok jelentős ingadozásokat tartalmaznak, bármely időintervallumban kiugróan magas értékek sokaságával. A hálózati forgalom jellemzői – mint például a csomagok száma másodpercenként, a bájtok száma másodpercenként vagy a keretek hossza – tekinthetők sztochasztikus idősoroknak. Ezért a

forgalom ingadozásának meghatározása megegyezik a megfelelő idősorok önhasonlóságának jellemzésével.

A hálózati forgalom önhasonlóságával foglalkozó cikkek megmutatják, hogy a csomagvesztés, a puffer kihasználtság és a válaszidő teljesen más lesz akkor, ha a szimuláció valós forgalomadatokat vagy az önhasonlóságot is tartalmazó mesterséges adatokat használ.

A háttérről

Legyen $X = (X_t : t = 0, 1, 2, \ldots)$ egy stacionárius kovariancia folyamat. Egy ilyen folyamathoz tartozik egy $\mu = \mathrm{E}[X_t]$ várható érték konstans, egy $\sigma^2 = \mathrm{E}\left[(X_t - \mu)^2\right]$ szórásnégyzet és egy $r(k) = \mathrm{E}\left[(X_t - \mu)(X_{t+k} - \mu)\right] / \mathrm{E}\left[(X_t - \mu)^2\right]$ $(k = 0, 1, 2, \ldots)$ autokorrelációs függvény, ami csak k-tól függ. Feltételezzük, hogy X-nek van egy

$$r(k) \sim \alpha k^{-\beta}, \quad k \to \infty \tag{5.1}$$

alakú autokorrelációs függvénye, ahol $0 < \beta < 1$ és α egy pozitív konstans. Reprezentálja $X^{(m)} = (X_{(k)}^{(m)} : k = 1, 2, 3, \; m = 1, 2, 3, \ldots)$ az új idősorokat, melyeket az eredeti X sorok feletti m méretű, egymást nem fedő blokkok átlagolásával kapunk. Minden $m = 1, 2, 3, \ldots, X^{(m)}$-re $X_k^{(m)} = (X_{km-m+1} + \cdots + X_{km})/m, (k \geq 1)$. Továbbá jelölje $r^{(m)}$ az egyesített $X^{(m)}$ idősorok autokorrelációs függvényeit.

Az önhasonlóság definíciója

Az X folyamatot ***önhasonlónak*** nevezzük $H = 1 - \beta/2$ önhasonlósági paraméterrel, ha a megfelelő $X^{(m)}$ egyesített folyamatnak azonos korrelációs struktúrája van, mint X-nek, azaz $r^{(m)}(k) = r(k)$ minden $m = 1, 2, \ldots$ $(k = 1, 2, 3, \ldots)$ esetén.

Az X stacionárius kovariancia folyamatot ***aszimptotikusan önhasonlónak*** nevezzük $H = 1 - \beta/2$ önhasonlósági paraméterrel, ha minden eléggé nagy k-ra $r^{(m)}(k) \to r(k)$, ha $m \to \infty$, $0.5 \leq H \leq 1$.

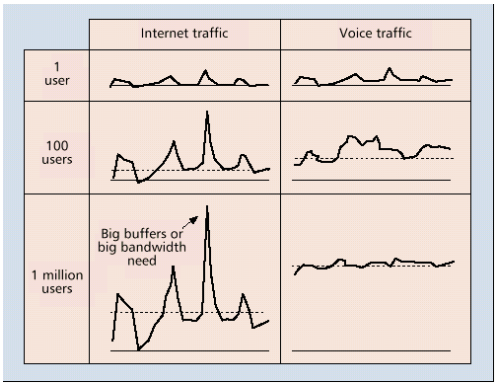
A hosszú távú függőség definíciója

Egy stacionárius folyamat ***hosszú távon függő,*** ha az autokorrelációs értékek összege végtelenhez tart: $\sum_k r(k) \to \infty$. Egyébként a folyamat ***rövid távon függő.*** A definíciókból származtatható, hogy míg a rövid távon függő folyamatoknak exponenciálisan fogyó, addig a hosszú távon függő folyamatoknak hiperbolikusan fogyó autokorrelációik vannak, azaz a kapcsolódó eloszlás nehéz farkú. A nehéz farokkal rendelkező eloszlású valószínűségi változók nagy valószínűséggel generálnak különösen nagy értékeket.

Az önhasonlóság fokát a H vagy ***Hurst-paraméter*** fejezi ki. A paraméter a folyamat autokorrelációs függvényének a fogyás sebességét mutatja. Amint $H \to 1$, az önhasonlóság és a hosszú távú függőség mértéke is nő. A hosszú távon függő önhasonló folyamatok esetén $H > 0.5$.

Forgalommodellek

A forgalommodellezés a hagyományos hangátviteli hálózatok modellezéséből ered. Korábban a modellek legtöbbje azon a feltevésen alapult, hogy a modellezett folyamatok markoviak (vagy általánosabban rövid távon függők). Azonban napjaink nagy sebességű digitális csomag-alapú hálózatai a hálózati szolgáltatások és technológiák különbözőségének köszönhetően sokkal bonyolultabbak és erősebben ingadozók, mint a hagyományos hangforgalom.



5.22. ábra. Az Internetes hálózati forgalom önhasonló természete.

Válaszul az új fejlesztésekre számos kifinomult sztochasztikus modellt hoztak létre, mint például a Markov-modulált Poisson-folyamatok, a fluid-flow modellek, a markovi érkezési folyamatok, a kötegelt markovi érkezési-folyamat modellek, a csomagsorozat modellek és a *Transzformáció-kiterjesztés-minta* („Transform-Expand-Sample") modellek. Ezek a modellek leginkább a kapcsolódó sorbanállási probléma analitikus megoldására helyezik a hangsúlyt. Általában nem vetik őket össze a forgalom valós jellemzőivel, és nem bizonyítják, hogy eredményeik összhangban vannak a valós forgalomadatok statisztikai jellemzőivel.

A modellek egy másik csoportja megpróbálja követni a valós forgalomadatok statisztikai jellemzőit. Viszont hosszú ideig a hálózatokkal kapcsolatos kutatások nem rendelkeztek megfelelő forgalmi mérésekkel. Azonban az utóbbi években nagy mennyiségű – a Webre és a nagy sebességű hálózatokra vonatkozó – mérési adatot gyűjtöttek össze és tettek elérhetővé. Ezeknek az adatoknak egy része nagy felbontású, órákon, napokon vagy heteken keresztül mért forgalmi adatokat tartalmaz. Más részük hetekről, hónapokról, évekről biztosít információkat. A nagy felbontású adatok statisztikai elemzései bebizonyították, hogy a csomag-alapú hálózatok valós forgalmi adatai alátámasztják az önhasonlóságot. Ezek az eredmények rámutatnak a tradicionális modellek és a mért forgalmi adatok közötti eltérésekre. Míg a hagyományos modellekben a feltételezett folyamatok rövid távon függők, a mért forgalmi adatok bizonyítják a hosszú távú függőséget. A 5.22. ábrán az Internet-forgalom és a hangforgalom közötti különbség figyelhető meg különböző felhasználószám esetén. Amint a hangfolyamok száma nő, a forgalom egyre egyenletesebb lesz, ellentétben az Internet-forgalommal.

A rövid távon függő sorbanállási modellekkel ellentétben a hosszútávú függőséggel rendelkező modellekkel kapcsolatban eddig jóval kevesebb elméleti eredmény született.

Az önhasonló modelleknek két fő csoportja van: a fraktális Gauss-zajok és a fraktális ARIMA folyamatok. A Gauss-modellek pontosan leírják több forgalomfolyam egyesítését. Az M/Pareto modellt olyan hálózati forgalommodellezésben használták, ahol a Gauss-modell alkalmazásához nem állt rendelkezésre elegendő számú egyesített adat.

Fekete doboz és strukturális modellek

A hagyományos idősorok elemzését ***fekete doboz modellezésnek*** nevezzük. Ezzel ellentétben a szerkezeti modellezés arra a környezetre összpontosít, amelyben a modell adatait gyűjtötték, azaz a hálózati komponensek hierarchiájára, melyekből napjaink kommunikációs rendszerei felépülnek. Míg az előbbi szerzők elismerik, hogy a fekete doboz modellek hasznosak lehetnek más környezetekben, a modern csomag-alapú hálózatok dinamikus és bonyolult természetének megértésére alkalmatlannak tartják őket. Ezeknek a modelleknek nem sok haszna van napjaink hálózatainak tervezésében, irányításában és ellenőrzésében sem. Hogy az empirikusan megfigyelt jelenségeknek, mint amilyen a hosszútávú függőség is, fizikai magyarázatát adjuk, a fekete doboz modelleket strukturális modellekre kell cserélnünk. A strukturális forgalommodelleknek egy jól alkalmazható jellemzőjük, hogy figyelembe veszik napjaink hálózatainak rétegelt felépítését, és elemezni tudják a hozzájuk kapcsolódó hálózatparamétereket, amelyek alapvetően meghatározzák a hálózat teljesítményét és működését. Az idősor-modellek általában ezeket a részleteket fekete dobozokként kezelik. Mivel a valós hálózatok bonyolult rendszerek, a fekete doboz modellek sok esetben számos paraméterről feltételezik, hogy pontosan ábrázolják a valós rendszert. A hálózattervezők számára, akik igen fontos alkalmazói a forgalommodellezésnek, a fekete doboz modellek nem túl hasznosak. Egy bonyolult hálózati környezetben ritkán lehetséges megmérni vagy megbecsülni a modell nagy számú paraméterét. A hálózattervezők számára a modellnek egyszerűnek és értelmesnek kell lennie az adott hálózatra nézve. A modell támaszkodhat valós hálózati mérésekre, és az eredményeknek helytállóknak kell lenniük a valós hálózat teljesítményére és működésére nézve.

Sokáig a forgalommodelleket valódi hálózatokban gyűjtött adatok felhasználása nélkül készítették. Ezek a modellek nem voltak alkalmazhatók gyakorlati hálózattervezésre. Napjainkban a nagy mennyiségű hálózati forgalmi mérések elérhetősége, és a hálózati struktúra növekvő bonyolultsága miatt egyre inkább alkalmazzák az ***Ockham borotvája*** nevű elvet. (Ockham borotvája egy középkori filozófus, William Ockham elve volt. Eszerint az elv szerint a modellezőknek nem szabad a minimálisan szükségesnél több feltételezést tenniük. Ez az alapelv, melyet a takarékosság elvének is hívnak, motiválja az összes tudományos modellezést és elméletépítést. A modellezőknek az adott jelenség leírására alkalmas ekvivalens modellek közül a legegyszerűbbet kell választaniuk. Ez az elv bármely modell esetén segíti a modellezőket abban, hogy csak azokat a változókat vegyék be a modellbe, amelyek valóban szükségesek a jelenség magyarázatához. Így a modellek kifejlesztése egyszerűbbé válik, csökken az inkonzisztencia, a félreérthetőség és a redundancia lehetősége.)

A szerkezeti modellek bemutatják, hogyan magyarázza a forgalom dinamikájának részleteit az egyes hosztok szintjén a hosztok közötti párbeszédek hálózati forgalomának önhasonló természete. A szerkezeti forgalommodelleknek fizikai jelentése van az adott hálózati környezetben, és nyomatékosítják a hosszútávú függőség túlsúlyát az egyes hosztok közötti párbeszédek által generált csomag érkezési mintákban. Ezek a modellek betekintést nyújtanak abba, hogy az egyes hálózati kapcsolatok hogyan viselkednek helyi és nagy kiterjedésű hálózatokban. Habár ezek a modellek az összeállított forgalmi minták fizikai struktúrájának figyelembevételével túlmennek a fekete doboz modellezési módszertanon, nem tartalmazzák a kapcsolatok, forgalomirányítók, kapcsolók és ezek véges kapacitásainak a forgalmi útvonalakon egybefonódó struktúráját.

Crovella és Bestavros megmutatták, hogy a World Wide Web forgalma az önhasonlóságával egyező tulajdonságokat mutat, továbbá hogy a Weben elérhető állományméretek

eloszlása miatt az átviteli idők nehéz farkúak lehetnek. Szintén megmutatták, hogy elsődlegesen a felhasználók „gondolkodási idejének" hatása miatt a csendes idők szintén nehéz farkúak lehetnek.

Az erős ingadozás hatása a nagy sebességű hálózatokra
Az erős ingadozásnak a hálózati torlódásokra gyakorolt hatásai a következők:

- A veszteségeket is tartalmazó torlódási periódusok meglehetősen hosszúak lehetnek, és erősen koncentráltak.
- A pufferméret lineáris növekedése nem okozza a csomagvesztés jelentős csökkenését.
- Az aktív kapcsolatok számának csekély növekedése is okozhat jelentős csomagvesztésnövekedést.

Az eredmények azt mutatják, hogy a csomagforgalom hullám jellegű. A hegyes csúcsok okozzák a tényleges adatvesztéseket, a kisebb hullámzások pedig a kiemelkedéseken lovagolnak tovább.

Egy másik terület, ahol az erős ingadozás befolyásolhatja a hálózat teljesítményét, az olyan ütemezésű kapcsolat, amely forgalomosztályok közötti prioritásokat tartalmaz. Egy olyan környezetben, ahol a nagyobb prioritású osztályra nincs sávszélességkorlátozás (a fizikai sávszélességen kívül), az interaktív forgalom prioritást kaphat a nagy mennyiségű adatforgalommal szemben. Ha a magasabb prioritású osztály hosszú időn keresztül erősen ingadozik, akkor ezen osztály erős ingadozásai hosszú időre akadályozhatják az alacsonyabb prioritású forgalmat.

Az erős ingadozás olyan hálózatokra is hatással lehet, ahol az engedélyezésvezérlési mechanizmus nem az egyes kapcsolatok forgalom-paraméterein, hanem a közelmúlt forgalmi mérésein alapszik. Az az engedélyezésvezérlés, amely csak a közelmúlt forgalmát veszi figyelembe, félrevezethető egy hosszú, egészen alacsony forgalmi intenzitású periódust követően.

5.8.2. Modellparaméterek

A kliensek és szerverek közötti tranzakciók aktív és azt követő inaktív periódusokból állnak. Ezek a tranzakciók az egyes irányokban küldött csomagok csoportjaiból tevődnek össze. A csomagcsoportokat erős ingadozásoknak nevezzük. A forgalom erősen ingadozó volta a következő időparaméterekkel jellemezhető:

- ***Tranzakció érkezési időköz*** („Transaction Interarrival Time" – TIAT): Egy adott tranzakció első csomagja és a következő tranzakció első csomagja között eltelt idő.
- ***Erős ingadozás érkezési időköz*** („Burst Interarrival Time"): az erős ingadozások között eltelt idő, $1/\lambda$, ahol λ az erős ingadozások beérkezési intenzitása.
- ***Csomag érkezési időköz*** („Packet Interarrival Time"): az erős ingadozásokban a csomagok érkezése között eltelt idő, $1/r$, ahol r a csomagok beérkezési intenzitása.

A Hurst-paraméter
Előre látható, hogy az egyre több és többféle forgalom gyorsan folytatódó többszolgáltatású hálózatokba való egyesítése végül a forgalom kiegyenlítődését fogja eredményezni. Ha elegendő az egyesítés mértéke, a folyamat Gauss-folyamatokkal modellezhető. Jelenleg

viszont a hálózati forgalom nem mutat a gaussihoz közeli jellemzőket. A hálózatok nagy részében az egyesítés mértéke nem elég nagy ahhoz, hogy kiegyenlítse az erősen ingadozó forgalom negatív hatását. Azonban addig, amíg a forgalom gaussivá nem válik, a létező módszerek pontos méréseket és előrejelzéseket nyújthatnak az erősen ingadozó forgalomról.

A módszerek többsége a Hurst-paraméter becslésén alapszik – minél nagyobb a H értéke, annál nagyobb az ingadozás, következésképpen rosszabb a kapcsolók és a forgalomirányítók teljesítménye az egyes forgalmi útvonalak mentén. Egyes módszerek megbízhatóbbak másoknál. A megbízhatóság számos tényezőtől függ, például a becslési technikától, a mintaérettől, az időintervallumtól, a forgalmi politikától stb. A publikált mérések alapján megvizsgáltuk a legkisebb becslési hibával rendelkező módszereket[1]. Ezek közül az *Újraskálázott módosított tartomány* („Rescaled Adjusted Range" – R/S)) módszert választottuk, mivel ennek megvalósítását megtaláltuk a hálózatról letölthető Benoit csomagban. Módszerünkhöz az ezen csomag által kiszámított Hurst-paraméter szolgál bemenetként.

Az M/Pareto forgalommodell és a Hurst-paraméter

Ismert, hogy az M/Pareto modell alkalmas a hosszú erős ingadozásokat tartalmazó, hosszú távon függő forgalomfolyamok modellezésére. A modellt eredetileg az ATM pufferszintjeinek az elemzésére javasolták, később az Ethernet, VBR videó és egykiszolgálós sorbanállási rendszerekben IP csomagfolyamok teljesítményének előrejelzésére is használták. A modellt itt nem csak egykiszolgálós sorra alkalmazzuk, hanem olyan összetett rendszerre, melyben kapcsolatok, kapcsolók és forgalomirányítók befolyásolják az egyes hálózatelemek teljesítményét.

Az M/Pareto modell egymást átfedő, λ beérkezési intenzitású erős ingadozások Poisson-folyamata. Az ingadozások r intenzitással generálnak csomagokat. Minden ingadozás az intervalluma kezdetétől egy Pareto-eloszlású ideig folytatódik. A Pareto-eloszlás alkalmazása eredményezi a hosszú távon függő forgalmat jellemző nagyon hosszú ingadozásokat.

Annak a valószínűsége, hogy egy Pareto-eloszlású X valószínűségi változó túllép egy x határt:

$$\Pr\{X > x\} = \begin{cases} \left(\frac{x}{\delta}\right)^{\gamma}, & x \geq \delta \\ 1, & \text{egyébként,} \end{cases} \qquad (5.2)$$

$$1 < \gamma < 2, \delta > 0.$$

Az X várható értéke, az erős ingadozások $\mu = \delta\gamma/(\gamma - 1)$ várható hossza és ennek szórásnégyzete végtelen. Egy t intervallumot feltételezve ebben az intervallumban a csomagok M átlagos száma:

$$M = \lambda tr\delta\gamma/(\gamma - 1)\,, \text{ és} \qquad (5.3)$$

$$\lambda = \frac{M(\gamma - 1)}{tr\delta\gamma}\,. \qquad (5.4)$$

Az M/Pareto modell aszimptotikusan önhasonló, és a Hurst-paraméterre teljesül, hogy

$$H = \frac{3 - \gamma}{2}\,. \qquad (5.5)$$

[1] Szórásnégyzet, együttes szórásnégyzet, Higuchi, maradék-szórásnégyzet, Újraskálázott módosított tartomány, Whittle becslés, periodogram, regressziós maradéktag

eltelt idő (másodperc)	átlagos sávszélesség kihasználtság %	összes bájt/ másodperc	bejövő bájtok/ másodperc	kimenő bájtok/ másodperc
299	2.1	297.0	159.2	137.8
300	2.2	310.3	157.3	153.0
301	2.1	296.8	164.4	132.4
302	2.7	373.2	204.7	168.5
...	...	...	...	...

5.23. ábra. Nyomkövetési eredmények.

bájtok átlagos száma	üzenet késés (milliszekundum)	pufferszint (bájt)	eldobott csomagok száma	a kapcsolat sávszélesség kihasználtsága (%)		
				56 Kbps Frame Relay	ATM DS-3 szegmens	100 Mbps Ethernet
440.4279	78.687	0.04	0	3.14603	0.06	0.0031

5.24. ábra. A mért hálózatjellemzők.

5.8.3. A Hurst-paraméter megvalósítása a COMNET modellezőeszközben

A Hurst-paramétert és az M/Pareto modell egy módosított változatát a COMNET diszkrét-esemény szimulációs rendszerben valósítottuk meg. A diszkrét-esemény szimuláció segítségével valósághű hálózatjellemzőket kaphatunk. Ilyen jellemző például a kapcsolatok kihasználtsága és a kapcsolók, forgalomirányítók teljesítménye. Módszerünkkel felmérhetjük az összeállított erősen ingadozó forgalom káros következményeit, és előrejelezhetjük hatását a teljes hálózat teljesítményére.

Forgalmi mérések

Az alapkonfigurációs modell felépítéséhez egy nagyméretű intézményi hálózatban a Concord Network Health nevű hálózatelemző rendszerrel gyűjtöttünk össze nyomkövetési információkat. Különböző széles és keskeny sávú kapcsolatokon végeztünk méréseket, például 45 Mbps ATM, 56 Kbps, és 128 Kbps Frame Relay összeköttetéseken. A Concord Network Health rendszer segítségével adott ideig forgalmi méréseket végezhetünk az egyes hálózati csomópontoknál, mint például forgalomirányítóknál, kapcsolóknál. Az időintervallumot 6000 másodpercre állítottuk be, és a következőket mértük: a küldött és fogadott bájtok és csomagok másodpercenkénti számát, a csomag késleltetést, az eldobott csomagok számát stb. A rendszer nem képes mérni az ingadozásokban lévő csomagok számát és az ingadozások hosszát, mint ahogy azt az előbbi M/Pareto modellben feltételeztük. Emiatt a korlátozás miatt a rendelkezésre álló adatoknak megfelelően kissé módosítjuk a forgalommodellt. ötpercenként készítünk pillanatfelvételeket a egy keskeny sávú Frame Relay kapcsolat forgalomáról egy távoli kliens és egy intézmény szervere között a 5.23. ábra szerinti formában.

A küldött bájtok átlagos számát, az üzenetkésést, a kliens helyi forgalomirányítójának a pufferszintjét, az eldobott csomagok számát, és az 56 Kbps sebességű kapcsolatnak, az ATM hálózat DS-3 szegmensének és a 100 Mbps sebességű Ethernet kapcsolatnak a célállomásnál lévő átlagos kihasználtságait a 5.24. ábra foglalja össze.

A COMNET a tranzakciókat a következőkkel reprezentálja: üzenetforrás, cél, üzenet méret, valamint az útvonalon lévő kommunikációs berendezések és kapcsolatok. Az üze-

netküldési intenzitás egy beérkezési időköz eloszlással van megadva, azaz az egymást követő csomagok között eltelt idővel. Az M/Pareto modellben a Poisson-eloszlás λ intenzitással generál ingadozásokat vagy üzeneteket. Ezt az információt a szimulációban úgy adjuk meg, hogy az egymást követő érkezések közötti időintervallum hossza átlagosan $1/\lambda$. Erre a célra az exponenciális eloszlást használjuk. Az exponenciális eloszlásnak a beérkezések közötti időre való alkalmazásával egy Poisson-eloszlás szerinti érkezési mintát fogunk kapunk. COMNET-ben a beérkezések közötti időt az $\mathrm{Exp}(1/\lambda)$ függvénnyel implementáltuk. A modellben a Concord Network Health-ben lévő mintának megfelelően 1 másodpercnek állítottuk be a beérkezési időközt, ami megegyezik $\lambda = 1$/másodperc beérkezési intenzitással.

Az M/Pareto modellben az egyes erős ingadozások Pareto-eloszlású ideig tartanak. Mivel a Concord Network Health eszköz nem tudta mérni az ingadozások hosszát, ezért azt feltételezzük, hogy az ingadozást az egy üzenetben küldött vagy fogadott bájtok másodpercenkénti száma jellemzi. Mivel az ATM cellakezelő algoritmusa biztosítja, hogy azonos hosszúságú üzenetek azonos idő alatt legyenek feldolgozva, ezért a hosszabb üzenetek hosszabb feldolgozási időt igényelnek. Így azt mondhatjuk, hogy az ingadozások időtartamának eloszlása megegyezik az ingadozások méretének eloszlásával. Ezért módosíthatjuk az M/Pareto modellt úgy, hogy a Pareto-eloszlású ingadozás-időtartamot a Pareto-eloszlású ingadozás-mérettel helyettesítjük. Ezután δ-t nem az ingadozások átlagos időtartamából, hanem azok átlagos méretéből kapjuk.

COMNET-ben a Pareto-eloszlású ingadozásméretet két paraméterrel definiáljuk, a hely és az alak paraméterekkel. A hely paraméter megfelel (5.2) δ-jának, az alak pedig a γ-nak, ami a (5.5) egyenlettel a következőképpen számítható ki:

$$\gamma = 3 - 2H. \tag{5.6}$$

A Pareto-eloszlásnak végtelen várható értéke és szórása lehet. Ha viszont az alak paraméter nagyobb, mint 2, akkor mindkettő véges lesz. Ha az alak paraméter nagyobb, mint 1 és kisebb vagy egyenlő, mint 2, akkor a várható érték véges, viszont a szórásnégyzet végtelen. Továbbá ha ez kisebb vagy egyenlő 1-nél, akkor mind a várható érték, mind a szórásnégyzet végtelen.

A Pareto-eloszlás várható értékéből a következőt kapjuk:

$$\delta = \frac{\mu \cdot (\gamma - 1)}{\gamma}\,. \tag{5.7}$$

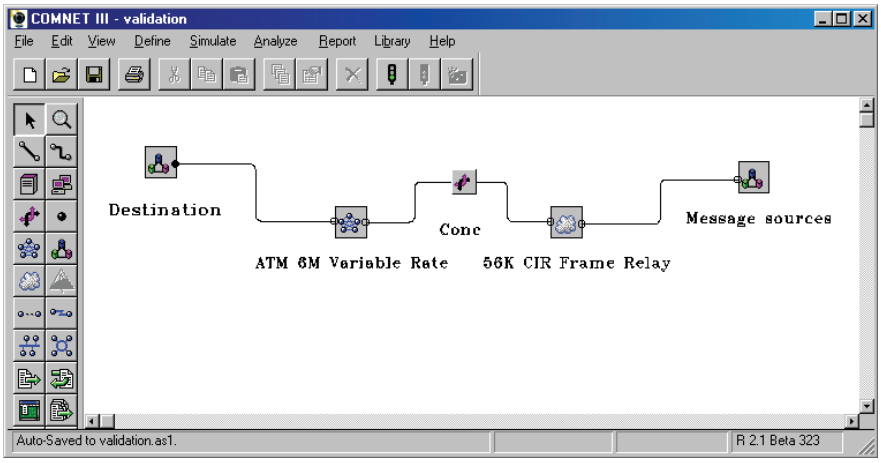
A (5.6) és (5.7) összefüggések lehetővé teszik az erősen ingadozó forgalom modellezését a valós nyomkövetési eredmények alapján a következő lépésekben:

a. Nyomkövetési adatok gyűjtése a Concord Network Health hálózatelemző felhasználásával.

b. A H Hurst-paraméter kiszámítása a nyomkövetési adatokból a Benoit csomag segítségével.

c. A COMNET eszköz exponenciális és Pareto-eloszlásainak felhasználásával, az előzőleg kiszámított paraméterekkel az érkezések közötti idő és az üzenetek méterének eloszlásának megadása.

d. Forgalomgenerálás a módosított M/Pareto modell szerint, és a hálózat teljesítményjellemzőinek mérése.



5.25. ábra. A hálózati topológiának az a része, ahol a méréseket végeztük.

Az előbbi lépesekkel generált forgalom erősen ingadozó olyan H paraméterrel, melyet valós forgalmi adatokból számítottunk ki.

5.8.4. Az alapkonfigurációs modell érvényesítése

Az alapkonfigurációs modellt úgy érvényesítjük, hogy az 56 Kbps sebességű Frame Relay és a 6 Mbps sebességű ATM kapcsolatok különböző paramétereit összehasonlítjuk a valós hálózatról a Concord Network Health hálózatelemző segítségével kapott adatokkal. Az egyszerűség kedvéért csak az „összes bájtok/másodperc” oszlopot vizsgáljuk. A valós forgalomnak a Benoit csomag által kiszámított Hurst-paraméter értéke $H = 0.55$. A hálózat topológiája a 5.25. ábrán látható.

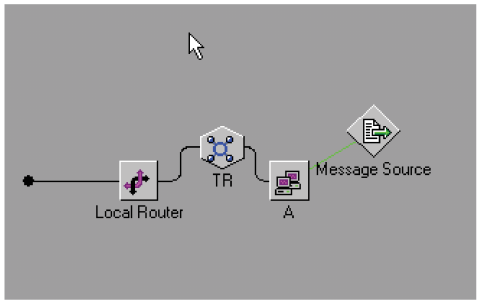
Az „üzenetforrások” ikon egy alhálózatot reprezentál, mely egy token ring hálózatból, egy helyi forgalomirányítóból és egy A kliensből áll, mely a „Célhálózat” alhálózatban lévő B kiszolgálónak küld üzeneteket (5.26. ábra).

A beérkezési időköz és az üzenetek mérete az Exp(1) exponenciális és a Par(208.42, 1.9) Pareto függvényekkel van definiálva. A Pareto-eloszlás helyét (208.42) és alakját (1.9) a (5.7) és (5.7) képletek segítségével számítottuk ki az erős ingadozások átlagos hosszának (ez 440 bájt a 5.24. ábra alapján) és a $H = 0.55$ paraméternek a behelyettesítésével (5.27. ábra).

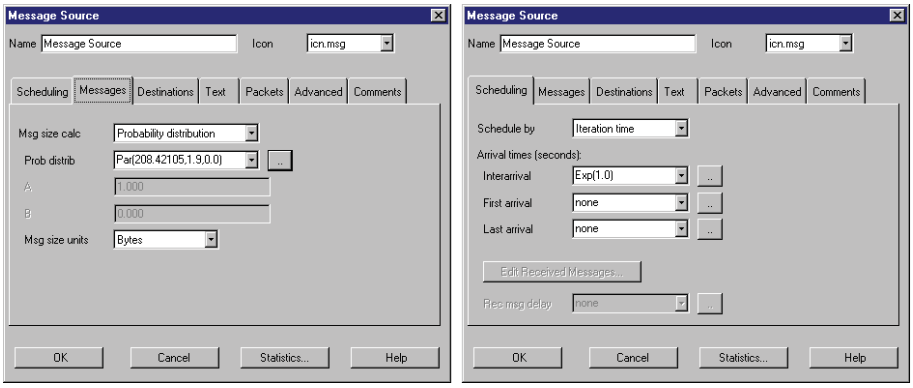
A 5.28. grafikon a megfelelő nehéz farkú Pareto-eloszlást és a kumulatív eloszlásfüggvényt szemlélteti (Az X tengelyen a bájtok számát ábrázoltuk).

A „Frame Relay” ikon egy kerettovábbítási felhőt reprezentál 56 K információátviteli intenzitással („Committed Information Rate” – CIR). A „Conc” forgalomirányító a Frame Relay hálózatot egy 6 Mbps sebességű, *változó intenzitásirányítással* („Variable Rate Control” – VBR) rendelkező ATM hálózathoz kapcsolja. Ezt a 5.29. és 5.30. ábra szemlélteti.

A „Célhálózat” jelöli a B kiszolgálót tartalmazó alhálózatot, melyet a 5.31. ábra szem-



5.26. ábra. Az „üzenetforrás” távoli kliens.



5.27. ábra. A kliens által küldött üzenetek beérkezési időköze és mérete.

léltet.

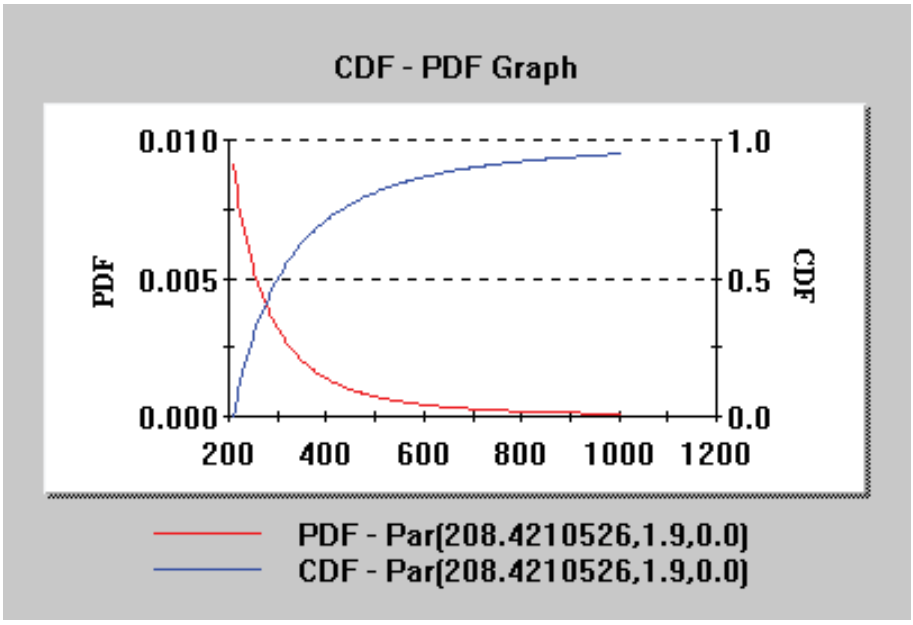
A modell eredményei a Frame Relay kapcsolat kihasználtságát tekintve (0.035 ~ 3.5%) majdnem azonosak a valós mérésekkel (3.1%) (5.32. ábra).

Az üzenetek késleltetése szintén nagyon közel van a kliens és a szerver közötti mért értékhez (78 milliszekundum) (5.33. ábra).

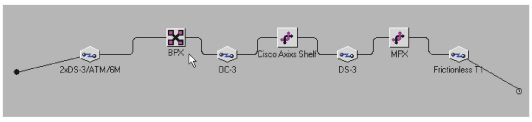
A 5.34. ábrán látható, hogy a kliens forgalomirányítójának input puffer szintje körülbelül azonos a mért értékkel.

Hasonlóan, az ATM hálózat DS-3 kapcsolat-szegmensének és a célhálózat Ethernet kapcsolatának kihasználtsága is jól közelíti a valós hálózat méréseit (5.35. ábra).

Az is megfigyelhető a modell nyomkövetési eredményeiből, hogy a $H = 0.55$ Hurst-paraméterrel a modell majdnem ugyanolyan erősen ingadozó forgalmat generál, mint a valós hálózat. Továbbá az eldobott csomagok száma a modellben és a méréseknél is nulla. Így tehát van egy olyan kiinduló modellünk, amely jól reprezentálja a valós hálózatot.



5.28. ábra. A Pareto-eloszlás 400 bájt várható érték és $H = 0.55$ Hurst-paraméter esetén. Az ábra színes változata a 809. oldalon látható.

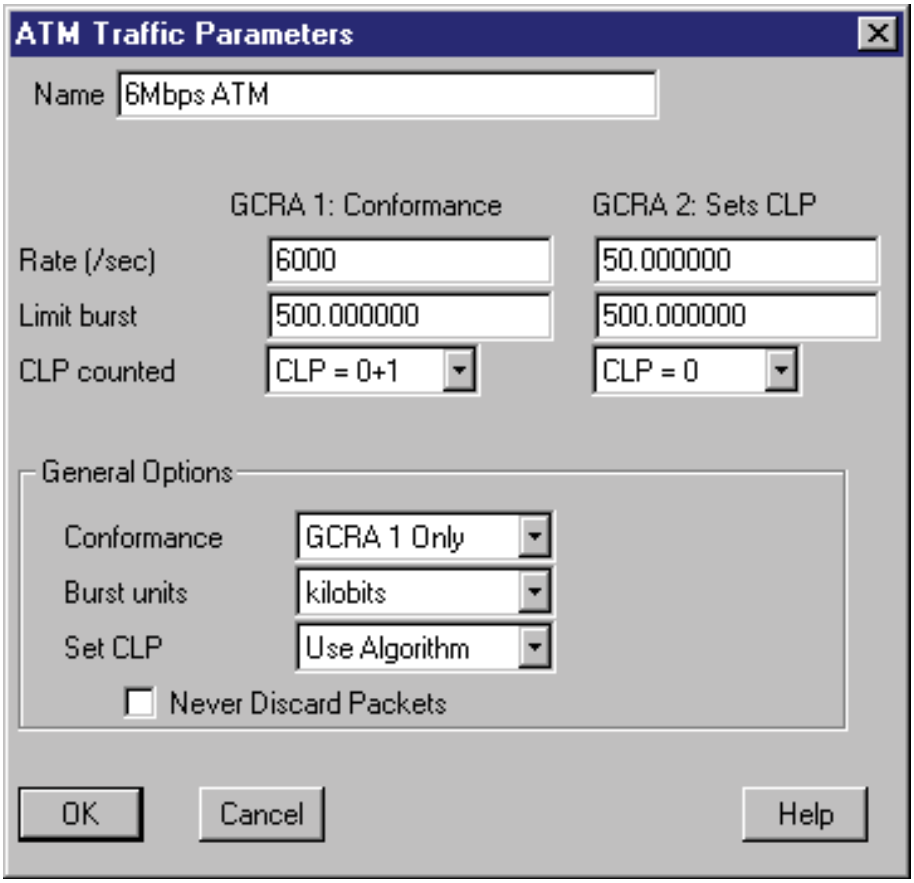


5.29. ábra. A 6 Mbps sebességű, változó intenzitásirányítású ATM hálózat belső kapcsolatai.

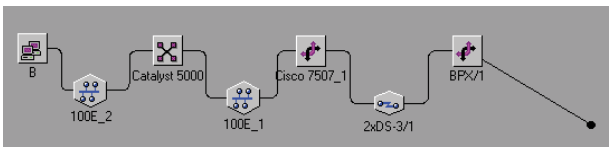
5.8.5. A forgalom erős ingadozásának következményei

Módszerünk illusztrálására kifejlesztettünk egy COMNET szimulációs modellt a forgalom erős ingadozásának a hálózati kapcsolatokra, az üzenetek késleltetésére, a forgalomirányítók input pufferere és a nagy számú felhasználótól származó összetett forgalom miatt eldobott csomagok számára gyakorolt következményeinek mérésére. A modell a 5.3. alfejezetben leírt módon valósítja meg a Hurst-paramétert. Hogy a ritka események is megfelelő számban előforduljanak, a szimulációt 6000, 16000 és 18000 másodpercig ismételtük. Az eredményt mindegyik esetben nagyon hasonlónak találtuk.

Az erősen ingadozó forgalom forrásainak topológiája
Az „üzenetforrás” alhálózatok az előbbi alapkonfigurációs modell szerint továbbítják az üzeneteket, azonos mérettel de különböző ingadozási paraméterrel: $H = 0.95$, $H = 0.75$ és $H = 0.55$. Kezdetben négy alhálózat működését szimuláltuk, alhálózatonként négy felhasz-



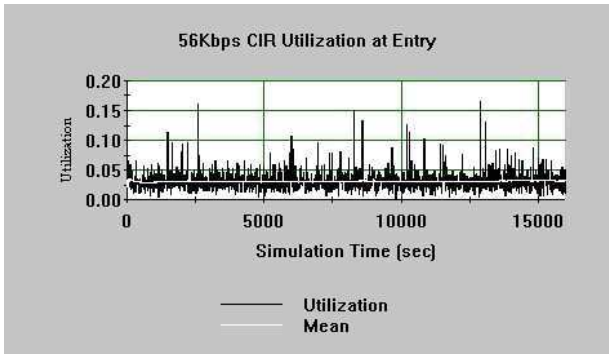
5.30. ábra. A 6 Mbps sebességű ATM kapcsolat jellemzői.



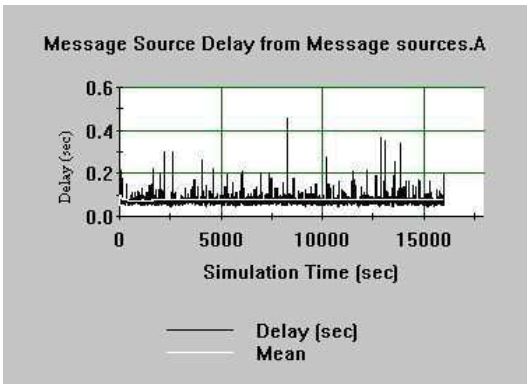
5.31. ábra. A „Célhálózat” alhálózat.

nálóval, melyek mindegyike az előzőekkel megegyezően ugyanolyan mennyiségű adatot küldött (átlagosan 440 bájtot másodpercenként) (5.36. ábra).

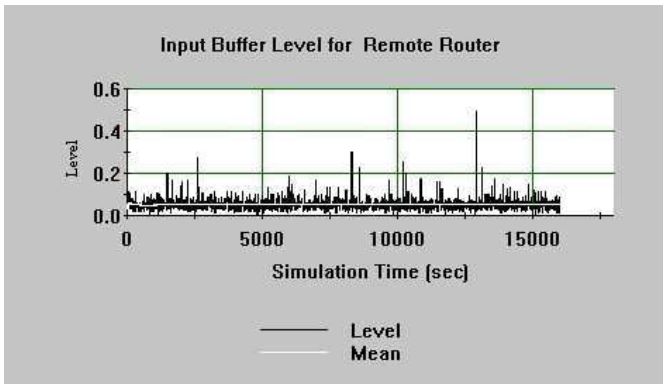
Kapcsolat-kihasználtság és üzenetkésleltetés
Először is egy Frame Relay kapcsolaton szeretnénk mérni és illusztrálni a különösen magas kihasználtság és üzenetkésleltetés csúcsokat. A modell forgalmát adó üzenetek méretét



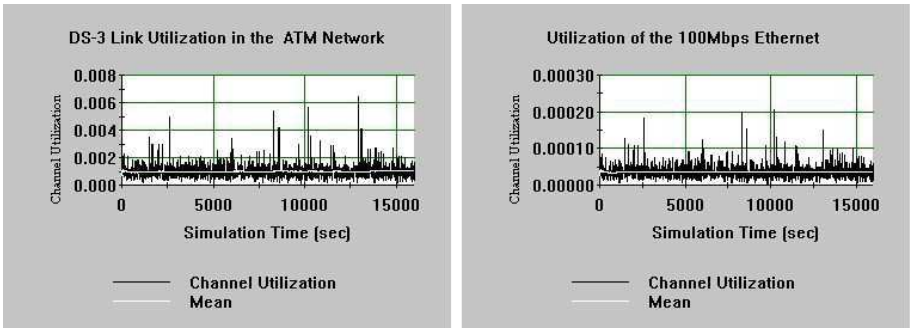
5.32. ábra. A Frame Relay kapcsolat kihasználtsága az alapkonfigurációs modellben.



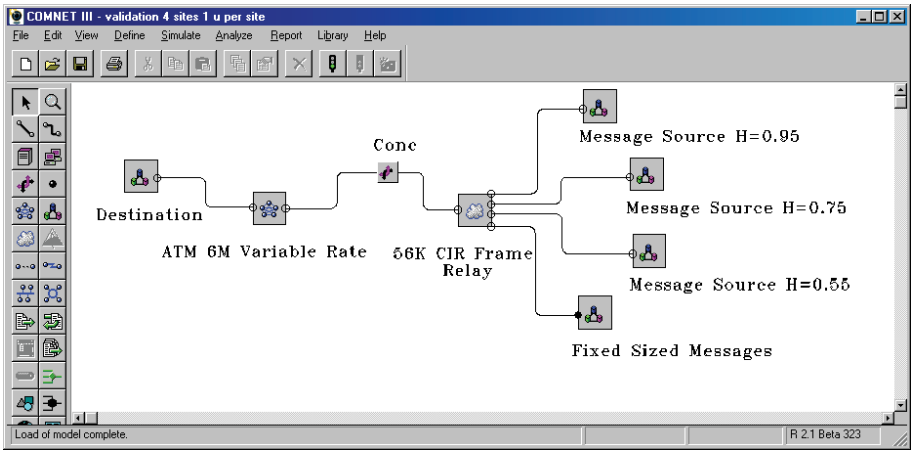
5.33. ábra. Az üzenetek késleltetése a kliens és a szerver között.



5.34. ábra. A távoli forgalomirányító input puffer szintje.



5.35. ábra. A DS-3 kapcsolat és a célhálózat Ethernet kapcsolatának kihasználtsága.



5.36. ábra. Erősen ingadozó forgalom forrásainak topológiája különböző Hurst-paraméterekkel.

	azonos méretű üzenetek	$H = 0.55$	$H = 0.75$	$H = 0.95$
átlagérték	0.12	0.13	0.13	0.14
csúcsérték	0.18	0.48	1	1

5.37. ábra. A szimulált kapcsolat-kihasználtságok átlagos és csúcsértékei.

különböző Hurst-paraméterek határozzák meg. Az üzenetek az összehasonlíthatóság érdekében azonos méretűek. A COMNET eszköznek van egy nyomkövetési opciója is, mellyel adatokat tud gyűjteni a modell által generált forgalomról. Azt is ellenőriztük, hogy a különböző Hurst-paraméterek esetén generált forgalomfolyamokból a Benoit-csomag hasonló Hurst-paramétereket számít ki.

A 5.37. ábrán az egyes szimulált esetek kapcsolat-kihasználtságainak átlagos és csúcsértékei láthatók. A kihasználtságok nem százalékban, hanem [0,1] intervallumbeli értékekként vannak kifejezve.

A következő alfejezetben található ábrák jól láthatóvá teszik, hogy annak ellenére hogy

	azonos méretű üzenetek	$H = 0.55$	$H = 0.75$	$H = 0.95$
átlagos válaszidő (ms)	75.960	65.61	87.880	311.553
válaszidő csúcsérték (ms)	110.06	3510.9	32418.7	112458.08
standard eltérés	0.470	75.471	716.080	4341.24

5.38. ábra. Válaszidő és erős ingadozás.

	azonos méretű üzenetek	$H = 0.55$	$H = 0.75$	$H = 0.95$
elfogadott csomagok	13282	12038	12068	12622
blokkolt csomagok	1687	3146	3369	7250
átlagos pufferhasználat bájtokban	56000858	61001835	62058222	763510495

5.39. ábra. Az eldobott cellák száma és az erős ingadozás közötti kapcsolat.

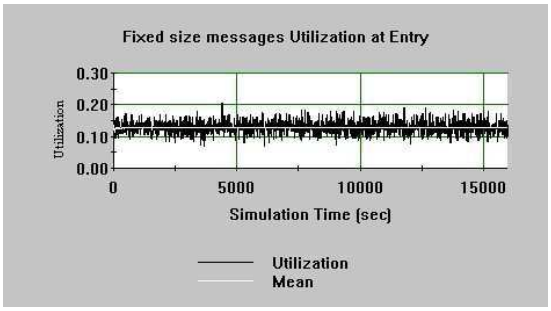
a kapcsolat átlagos kihasználtsága közel azonos, a csúcsértékek gyakorisága és mérete növekszik az ingadozás erősödésével, ez pedig cellavesztéseket okoz a forgalomirányítókban és a kapcsolókban. A válaszidőre a 5.38. ábrán látható eredményeket kaptuk.

Az A függelék grafikonjai grafikusan illusztrálják a különböző Hurst-paraméterek és a válaszidők közötti kapcsolatot.

Az input pufferek szintje nagy számú felhasználó esetén
Megmértük az erősen ingadozó cellaforgalom miatt az ATM hálózat egy forgalomirányítójának input pufferénél eldobott cellák számát is. Körülbelül 600 felhasználó összegzett forgalmát szimuláltuk, melyek a valós forgalmi mérésekkel egyezően ugyanannyi bájtot küldtek el másodpercenként. Az 5.39. ábra a blokkolt csomagok számát foglalja össze az egyes eseteknél.

5.8.6. Következtetések

Az előzőekben egy diszkrét-esemény szimulációs módszert mutattunk be, amely alkalmas különböző erősen ingadozó forgalmat továbbító hálózatok jellemzőinek mérésére. Korábbi tanulmányok bebizonyították, hogy az erősen ingadozó adatfolyamok egyesítése szintén erősen ingadozó adatfolyamot eredményez. Ezért a hagyományos, hálózattervezésre használt módszerek és modellek módosítására van szükség. A mi módszertanunkat a fekete doboz modellek helyett a strukturált modellek csoportjába soroljuk. A strukturális modellek arra a környezetre összpontosítanak, amiben a modellek adatai össze lettek gyűjtve, azaz a hálózati komponensek hierarchiájára, melyekből napjaink kommunikációs rendszerei felépülnek. Habár a fekete doboz modellek hasznosak lehetnek más környezetekben, nem olyan könnyű őket alkalmazni napjaink hálózatainak tervezésében, irányításában és ellenőrzésében. Egy jól ismert modellt, az M/Pareto modellt implementáltuk a COMNET diszkrét-esemény szimulációs csomag felhasználásával. Ez lehetővé teszi az önhasonló forgalom káros következményeinek elemzését nem csak egy egykiszolgálós sorra, hanem különböző kapcsolódó hálózati komponensek összteljesítményére nézve is. Valós hálózati nyomkövetési információk felhasználásával olyan modellt készítettünk és érvényesítettünk, mellyel mérni és grafikusan illusztrálni tudtuk az erősen ingadozó forgalomnak a kapcsolatok kihasználtságra, az üzenetkésleltetésekre és a pufferek teljesítményére gyakorolt hatását Frame Relay and ATM hálózatokban. Megmutattuk, hogy az erősödő ingadozás nagyon nagy kapcsolat-kihasználtságot, válaszidőt és sok eldobott csomagot eredményez, továbbá



5.40. ábra. A Frame Relay link kihasználtsága azonos méretű üzenetek esetén.

szimulációval meghatároztunk különböző teljesítményjellemzőket.

A csomagválasztás hangsúlyozza olyan eszközöknek a szükségességét, melyek hasznosak lehetnek nem csak elméleti szakemberek, hanem hálózattervezők és mérnökök számára is. Ez a cikk a meglévő, jól ismert elméleti eredmények és ezeknek a mindennapi gyakorlati hálózatelemzésben és modellezésben való alkalmazása közötti rést szeretné csökkenteni. Meglehetősen jó volna, ha a mérő, megfigyelő és irányító eszközökben rendelkezésre állnának a megfelelő forgalommodellek. Az itt tárgyalt modell segítheti a forgalommodellezés elsődleges felhasználóit, a hálózattervezőket és mérnököket abban, hogy megértsék a hálózati forgalom dinamikus természetét, továbbá támogathatja őket mindennapi munkájukban.

5.9. Mérési adatok bemutatása

5.9.1. Kapcsolat-kihasználtsági mérések

Az 5.40–5.42. ábrák azt szemléltetik, hogy bár a különböző Hurst-paraméterek esetén az átlagos kapcsolat-kihasználtságok majdnem azonosak, az ingadozás növekedésével a csúcsértékek gyakorisága és mérete nő, ez pedig a forgalomirányítókban és kapcsolókban cellaveszteséket okoz. A kihasználtságok nem százalékban, hanem [0,1] intervallumbeli értékekként vannak kifejezve.

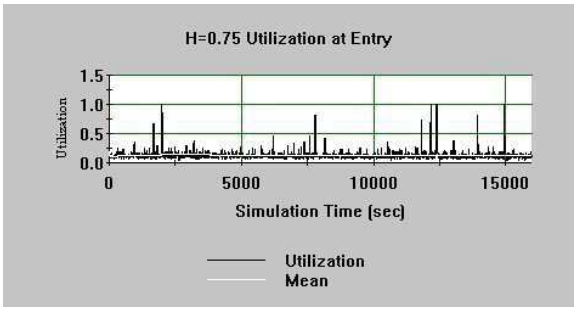
5.9.2. Üzenetkésleltetési mérések

A 5.43–5.45. ábrák a különböző Hurst-paraméterek és a válaszidők közötti kapcsolatot szemléltetik.

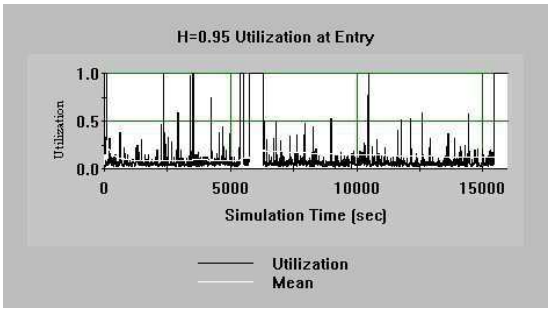
Gyakorlatok

5.9-1. Nevezzünk meg néhány, a következő fogalmakhoz kapcsolódó tulajdonságot, eseményt, tevékenységet és állapotváltozót:

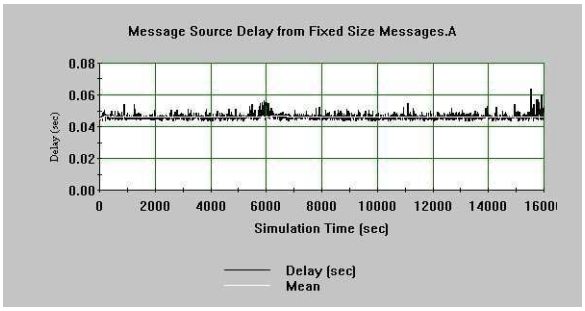
- Kiszolgáló
- Kliens
- Ethernet



5.41. ábra. A Frame Relay link kihasználtsága $H = 0.75$ Hurst-paraméter esetén (magasabb csúcsokkal).



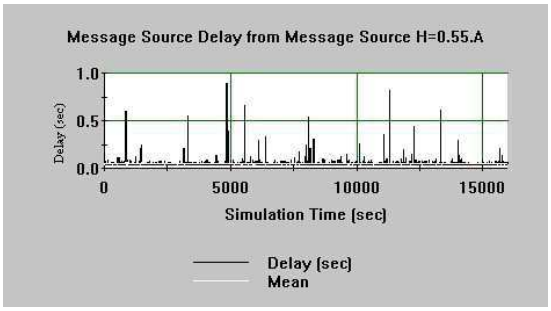
5.42. ábra. A Frame Relay link kihasználtsága $H = 0.95$ Hurst-paraméter esetén (sok magas csúccsal).



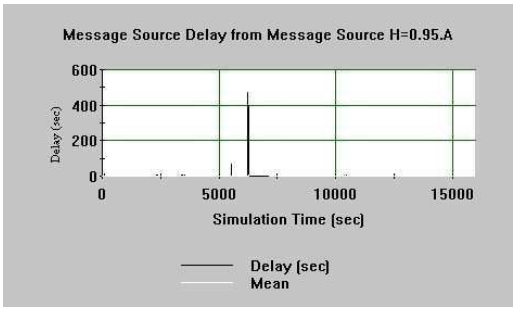
5.43. ábra. Üzenetkésleltetés azonos méretű üzenetek esetén.

- Csomagkapcsolt hálózat
- Híváslétesítés celluláris mobil hálózatban
- A TCP *lassú indulás algoritmusa*

5.9-2. Olvassunk el egy cikket a hálózatok szimulációjának alkalmazásáról, és írjunk beszámolót arról, hogyan közelíti meg a cikk a modell érvényesítést.



5.44. ábra. Üzenetkésleltetés $H = 0.55$ esetén (magasabb válaszidő csúcsokkal).



5.45. ábra. Üzenetkésleltetés $H = 0.95$ esetén (nagyon magas válaszidő csúccsal).

5.9-3. Ez a gyakorlat feltételezi, hogy rendelkezésre áll valamilyen hálózatelemző szoftver (például Lananalyzer for Windows vagy bármilyen más eszköz), mely elemezni tudja a hálózati forgalmat és információkat tud gyűjteni róla. A következőkben mi az előbb említett eszközt használjuk.

- Kezdjük el egy állomány továbbítását a helyi hálózatban egy kliens és egy kiszolgáló között. Figyeljük meg a részletes statisztikákat az adatátviteli vonal kihasználtságáról és a másodpercenként átvitt csomagok számáról, majd mentse el a megfelelő grafikonokat.
- A Lananalyzer Help menüjénél válasszuk ki és olvassuk el a „Capturing and Analyzing Packets” fejezetet.
- Az állomány átvitele során csak a kliens és a kiszolgáló közötti csomagokat vizsgáljuk.
- Mentsük el a csomagokról gyűjtött nyomkövetési információkat .csv formátumban. Elemezzük ezt az állományt táblázatkezelő felhasználásával. Figyeljük meg, vannake szokatlan protokoll-események, mint például a csomagok között eltelt túl hosszú idő, túl sok hibás csomag stb.

5.9-4. Ebben a gyakorlatban a Sniffer különböző hálózatelemzési és alapkonfiguráció készítési funkcióit vizsgáljuk. Az alapkonfiguráció definiálja a hálózatot jellemző tevékenységeket, és ennek ismeretében fel tudjuk ismerni a tipikustól eltérő működést. Ezt okozhatja

valamilyen probléma, vagy a hálózat növekedése is. Az alapkonfigurációs adatokat akkor kell gyűjteni, amikor a hálózati működése tipikusnak mondható. Egyes statisztikák készítéséhez, mint például a sávszélesség-kihasználtság vagy a csomagok száma másodpercenként, egy olyan grafikont kell készíteni, amely egy adott időintervallumban ábrázolja az információkat. Erre azért van szükség, mert az olyan mintavétel, amely túl rövid időintervallumban gyűjt adatokat, félrevezető lehet. Egy vagy több hálózatkomponens hozzáadása után érdemes egy alapkonfigurációt készíteni, így később össze lehet hasonlítani a hozzáadás előtti és utáni tevékenységeket. Az összegyűjtött adatok exportálhatók más programok, például táblázatkezelők és modellezőeszközök számára, amelyekkel további elemzéseket készíthetünk és amelyek segítik az összegyűjtött adatok kezelését.

A Sniffer egy nagyon jól alkalmazható hálózatelemző eszköz. Számos jól integrált funkciót tartalmaz, melyeket a következőkre használhatunk:

- Forgalomnyomkövetési információk gyűjtése részletes elemzés céljára.
- Problémák megállapítására az Expert Analyzer alkalmazásával.
- A hálózati tevékenységeknek valós idejű megfigyelésére.
- Részletes kihasználtsági és hibastatisztikák gyűjtésére az egyes állomásokról, párbeszédekről vagy a hálózat bármely részéről.
- A korábbi kihasználtsági és hibainformációknak alapkonfigurációs elemzés céljára történő tárolására.
- Problémák esetén az adminisztrátorokat figyelmeztető látható és hallható riasztások létrehozására.
- A hálózat aktív eszközökkel történő forgalomszimulációs vizsgálatára, válaszidő mérésre, hop számlálásra és hibaelhárításra.
- A Monitor menü History Samples pontja lehetővé teszi a hálózati tevékenységeknek egy időintervallumon keresztül való rögzítését. Ezek az adatok használhatók az alapkonfiguráció elkészítéséhez, ami segíti az egyes határértékek beállítását, melyeknek a normálistól eltérő működés esetén történő átlépése kiváltja az egyes riasztásokat. Továbbá ezek az adatok szintén hasznosak a hálózat terhelésének a hosszú távú változásainak meghatározására, így tervezhetővé válnak a jövőbeli hálózatbővítések.
- Egyidejűleg legfeljebb 10 hálózati tevékenység figyelhető meg vele. Egy adott tevékenység megfigyelésére több statisztikakészítés is elindítható, így egyidejűleg mind a rövid, mind a hosszú távú tendenciák rögzíthetők. A korábbi minták megfigyelérése rendelkezésre álló hálózati események az Adapter párbeszédablakban kiválasztott adaptertől függenek. Például, egy token ring hálózat esetén a különböző token ring kerettípusok mintái (mint például a beacon keretek) figyelhetők meg, Frame Relay hálózat

esetén pedig a különböző Frame Relay kerettípusok (például LMI keretek) mintái. A megfigyelhető események adapterenként változnak.

Gyakorlati feladatok:

- Állítsunk be egy szűrőt (Capture/Define filter) a saját PC-je és valamelyik távoli munkaállomás között az IP forgalom mintavételezésére.
- Állítsuk be a Monitor/History Samples/Multiple History-nál a következőket: Octets/s (Oktet/másodperc), Utilization (Kihasználtság), Packets/s (Csomag/másodperc), Collision/s (Ütközés/másodperc), és Broadcasts/s (Üzenetszórás/másodperc).
- Állítsuk be a mintavételi intervallumot 1 másodpercre (jobb klikk a Multiple ikonon és ott properties, Sample).
- Indítsuk el a hálózat megfigyelését (jobb klikk a Multiple ikonon, majd Start Sample).
- Szimuláljunk valamilyen szokásos hálózati forgalmat, például töltsünk le egy nagyméretű állományt egy kiszolgálótól.
- Rögzítsük a „Multiple History”-t ezalatt a „szokásos hálózati forgalom” alatt. Ezt tekintjük az alapkonfigurációnak.
- Állítsuk a Tools/Options/MAC/Threshold-nál az oktet/másodperc értékét az alapkonfigurációs érték 10-szeresére(???). Definiáljunk egy riasztást az oktet/másodpercre: Amikor eléri ezt a határértéket, küldessünk egy levelet a saját elektronikus címünkre. Az 5.46. ábrán azt feltételezzük, hogy ez a határérték 1000.
- A riasztást a 5.47. ábrán látható módon adjuk meg.
- Ezután állítsuk be az SMTP kiszolgálót a helyi, saját levelezőszerverére (5.48. ábra).
- A probléma komolyságát (Severity) állítsuk kritikusra (Critical) (5.49. ábra).
- Gyűjtsünk nyomkövetési információkat a forgalomról (Capture/Start) az állomány letöltése alatt.
- A letöltés befejeződése után állítsuk le az információgyűjtést (Capture/Stop majd Display).
- Az Expert Decode opcióval elemezzük a csomagok TCP/IP rétegeit!
- Nézzük meg, megérkezett-e a Sniffer Pro-tól a „riasztás e-mail”. Várhatóan az alábbihoz hasonló levelet fogjuk kapni, mely jelzi az oktet/másodperc határérték túllépését:

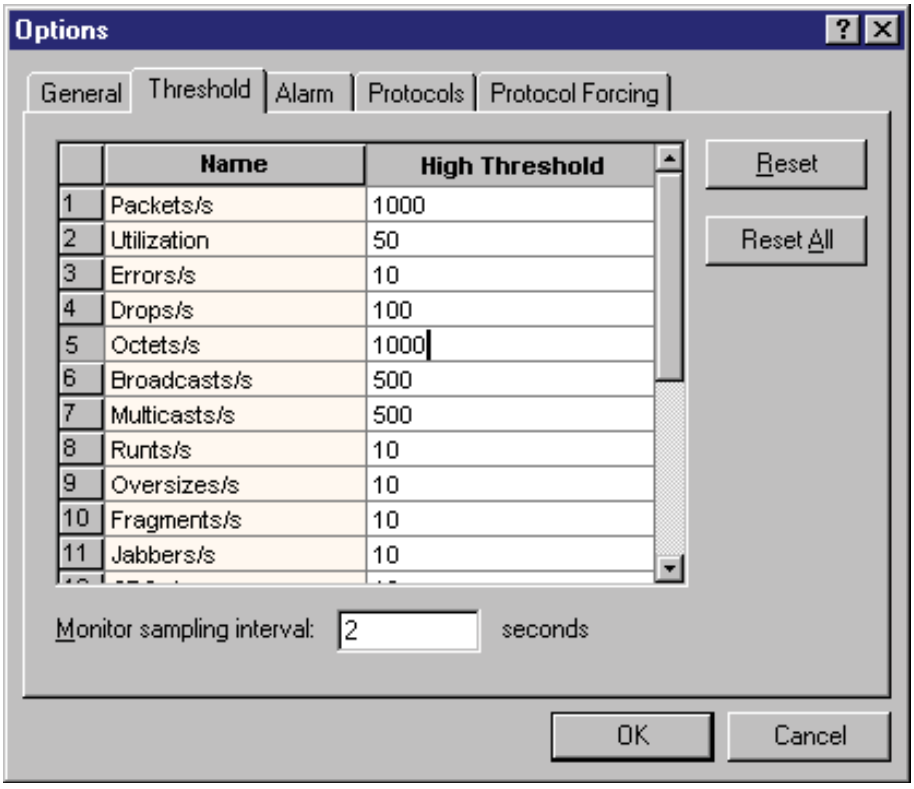
```
From: ...
Subject: Octets/s: current value = 22086, High Threshold = 9000
To: ...

This event occurred on ...
```

Mentse el a következő állományokat:

- A „Baseline screens”-t
- A Baseline Multiple History.csv állományt
- A „riasztás e-mail”-t

5.9-5. A gyakorlat célja egy alapkonfigurációs modell felépítése és érvényesítése egy hálózatmodellező eszköz felhasználásával. Feltételezzük, hogy a modellező számára elérhető



5.46. ábra. Beállítások.

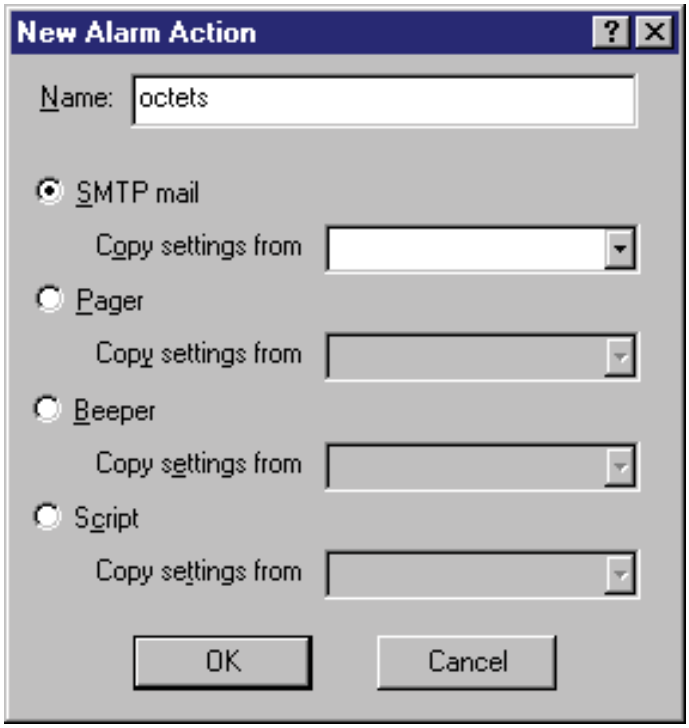
egy szimulációs modellezőeszköz, például a COMNET vagy az OPNET.

Először gyűjtsünk válaszidő statisztikákat egy távoli számítógép pingelésével. A ping parancs a hálózaton egy adott klienstől egy kiszolgálóhoz küldött csomagok oda-vissza útjának idejét méri. A parancs egy lehetséges formátuma a következő: ping hosztnév -n x -l y -w z > fájlnév. Itt „x” a küldendő csomagok száma, „y” a csomaghossz bájtokban, „z” az időtúllépési érték és a „fájlnév” az állomány neve, amibe az összegyűjtött statisztikák kerülnek.

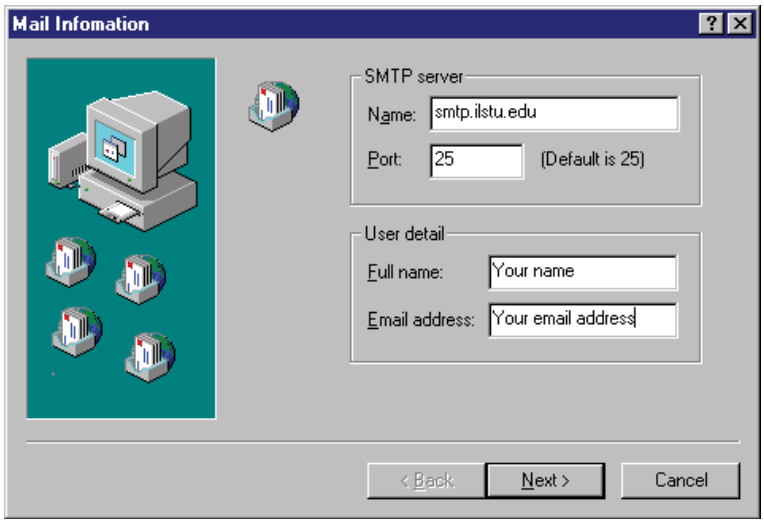
Például, a ping 138.87.169.13 -n 5 -l 64 > c:
ping.txt parancs a következő állományt hozza létre:

```
Pinging 138.87.169.13 with 64 bytes of data:
Reply from 138.87.169.13: bytes=64 time=178ms TTL=124
Reply from 138.87.169.13: bytes=64 time=133ms TTL=124
Reply from 138.87.169.13: bytes=64 time=130ms TTL=124
Reply from 138.87.169.13: bytes=64 time=127ms TTL=124
Reply from 138.87.169.13: bytes=64 time=127ms TTL=124
```

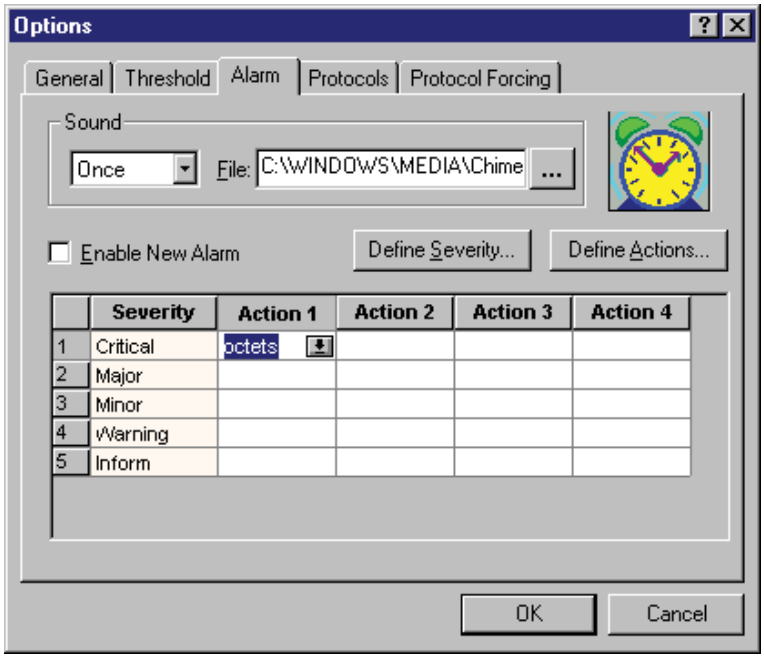
- Egy táblázatkezelő felhasználásával készítsünk egy hisztogramot ezekről az időkről és a csomagok sorszámáról.



5.47. ábra. Új riasztás akció.



5.48. ábra. Levelezési információk.



5.49. ábra. Beállítások.

- Készítsünk hisztogramot a válaszok számáról és a válaszidőkről.
- Készítsük el a válaszidők kumulatív sűrűségfüggvényét, az eloszlás farkánál a részleteket is feltüntetve.
- Készítsük el az átvitelek alapkonfigurációs modelljét. A forgalomjellemzőket az előző lépésben készített kumulatív sűrűségfüggvény határozzuk meg.
- Érvényesítsük a modellt.
- Mennyi a kapcsolat kihasználtsága 32 és 64 bájt hosszúságú üzenetek esetén?

5.9-6. Feltételezzük, hogy a modellező számára elérhető egy szimulációs modellezőeszköz, mint például a COMNET, OPNET stb. Ebben a gyakorlatban néhány gyakran használt kép-állomány helyét szeretnénk meghatározni egy laborban. Az előrejelzések szerint a következő évben az új kliensek hozzáadása megháromszorozza ezeknek az állományoknak a használatát. Ezek tárolhatók vagy a kiszolgálón, vagy a kliens munkaállomásokon. A könnyebb karbantartás miatt a kiszolgálón való tárolást részesítjük előnyben. Az aktuális hálózatnak egy alapkonfigurációs modelljét fogjuk elkészíteni, és megmérjük a kapcsolat-kihasználtságot az állományátvitelek alatt. Továbbá érvényesítjük a modellt az aktuális forgalomjellemzőkkel. A forgalom skálázásával előrejelezhetjük a kapcsolat-kihasználtságot a munkaállomások hozzáadása után megháromszorozódott forgalom esetén.

- Készítsük el az alapkonfigurációs modell topológiáját.
- Gyűjtsünk forgalom-nyomkövetési információkat az átvitel alatt, és importáljuk őket.

- Futtassuk és érvényesítsük a modellt (Az átvitt üzenetek számának a modellben meg kell egyeznie a nyomkövetési állományban lévő számmal, a szimuláció ideje egyenlőnek kell lennie a csomagok közötti idők – („Interpacket Time") - összegével, és a kapcsolat-kihasználtságnak közel egyenlőnek kell lennie a nyomkövetés alatti átlagos kihasználtsággal).
- Írassuk ki az átvitt üzenetek számáról, az üzenetkésleltetésről, a protokollok általi kapcsolat-kihasználtságról és a teljes kapcsolat-kihasználtságról szóló jelentéseket.
- Háromszorozzuk meg a forgalmat.
- Írassuk ki az átvitt üzenetek számáról, az üzenetkésleltetésről, a protokollok általi kapcsolat-kihasználtságról és a teljes kapcsolat-kihasználtságról szóló jelentéseket.
- Ha a kapcsolat-kihasználtság az alapkonfigurációs határérték alatt van, akkor a kiszolgálón hagyjuk a kép-állományokat, egyébként a munkaállomásokra helyezzük át őket.
- Kérdés: Hol érdemesebb tárolni ezeket, a klienseken vagy a kiszolgálón?

5.9-7. Ennek a gyakorlatnak a célja az osztott és a kapcsolt Ethernet teljesítményének összehasonlítása. Megmutatjuk, hogy az örökség („legacy") vagy osztott Ethernet átalakítása kapcsolt Ethernetté csak akkor indokolt, ha az ütközések száma meghalad egy adott határértéket.

a) Készítsük el egy osztott Ethernetet használó helyi hálózati kliens/szerver alkalmazás modelljét. A modell tartalmaz egy 10Base5 Ethernetet, mely egy kiszolgálót (Webszerver) és három számítógépcsoportot kapcsol össze (Kliens 1, Kliens 2, Kliens 3). Minden csoportnak három számítógép a tagja, továbbá minden csoportnak van egy „Web Request" nevű üzeneteket generáló forrása. A kiszolgálónak ezekre egy „Web Server" alkalmazása válaszol. Minden „Web Request" forgalmat generál a kiszolgálóhoz. Amikor a kiszolgáló megkap egy „Web Request" üzenetet, olyankor egy „Web Response" üzenetet generál és elküldi a megfelelő kliensnek.

- Minden „Web Request" egy 10000 byte hosszú üzenetet jelent, amit a forrás minden Exp(5) másodpercben küld a Webszerverhez. Állítsa az üzenet szövegét „Web Request"-re.
- A Webszerver visszaküld egy üzenetet „Web Response" szöveggel. Az üzenet mérete 10000 és 100000 bájt között változik, ezt a Geo(10000, 100000) eloszlás határozza meg. A szerver csak a kapott „Web Request" üzenetekre válaszol. Állítsa a válaszüzenetet „Web Response"-ra.
- A többi paraméternél használjuk az alapértelmezett értékeket.
- Válasszuk a „Csatornakihasználtságot" („Channel Utilization") és az „Ütközési statisztikák"-at („Collision Stats") a „Kapcsolatok beszámolói"-nál („Links Reports").
- Válasszuk az „Üzenetkésleltetés"-t az „Üzenet + válaszjelentés"-nél („Message + Response Source Report").
- Futtassuk a szimulációt 100 másodpercig. Megadhatjuk az animáció opciót is.
- Írassuk ki a „Kapcsolat-kihasználtság"-ot és az „Ütközési statisztikák"-at megjelenítő jelentést, valamint a forgalom forrásai közötti üzenetkésleltetésről szóló jelentést.

b) A válaszidő csökkentése érdekében alakítsuk át az osztott LAN-t kapcsolt LAN-ná. A kliens-szerver paramétereket és az Ethernet sebességét változatlanul hagyva helyezzünk

el egy Ethernet kapcsolót a kliensek és a szerver közé. (A szerver egy full-duplex 10Base5 kapcsolattal csatlakozik a kapcsolóhoz.)

- Írassuk ki a „Kapcsolat-kihasználtság”-ot és az „Ütközési statisztikák”-at megjelenítő jelentést, valamint a forgalom forrásai közötti üzenetkésleltetésről szóló jelentést.

c) Mindkét modellben cseréljük a 10Base5 kapcsolatokat 10 BaseT kapcsolatokra. Ekkor az előző esetekkel ellentétben nem azonos mértékű relatív javulást fogunk tapasztalni a válaszidőben. Magyarázzuk meg, miért.

5.9-8. Egy vállalat helyi hálózatának egy része két alhálózatból áll. Mindkettő egy-egy osztályt szolgál ki. Az egyik hálózat az IEEE 802.3 CSMA/CD 10BaseT Ethernet szabvány szerint üzemel, a másik pedig az IEEE 802.5 16Mbps Token Ring szabvány szerint. A két hálózat egy Cisco 2500-as forgalomirányítóval van összekapcsolva. Az Ethernet LAN 10 számítógépet tartalmaz, melyek közül egy e-mail szervernek van kijelölve mindkét osztály számára. A token ring LAN szintén 10 számítógépet tartalmaz, melyek közül egy fájlszervernek van kijelölve az osztályok számára.

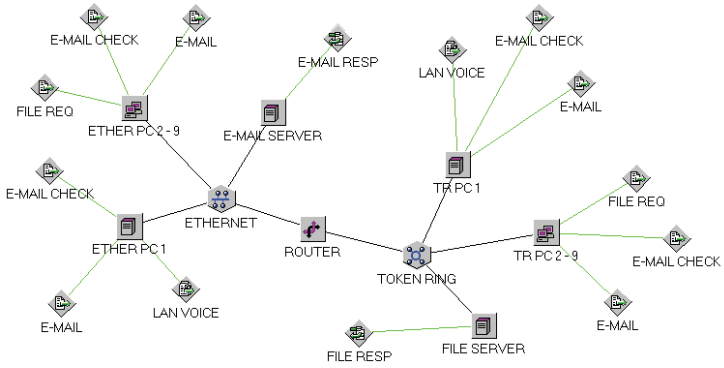
A vállalat jelenleg mindkét osztályra alkalmazottak felvételét tervezi. Habár a jelenlegi hálózati konfiguráció valószínűleg nem lesz képes kiszolgálni az új alkalmazottakat, a vállalatnak nincs semmilyen módszere a hálózat kihasználtságának vagy a késleltetésének mérésére. A vállalat az új alkalmazottak felvétele előtt szeretné felmérni ezeket az aktuális alapkonfigurációs szinteket. Az alkalmazottak egyébként is már mindkét osztályról panaszkodtak a fájlszervertől való letöltés lassúsága miatt.

A LAN-okon keresztülfolyó közös forgalomnak egy felmérése szerint a forgalom többsége a következő forrásokból származik: elektronikus levelezésből, a különböző alkalmazások állomány-átviteléből és a hang alapú levelezési rendszerből, amely lehetővé teszi a vezetők számára, hogy hallható üzeneteket küldjenek az osztályukon dolgozó alkalmazottaknak. Az üzenetjellemzők statisztikai leírásának alapjául az alkalmazottakkal folytatott beszélgetések és az üzenetek átlagos méretének becslése szolgált.

Az elektronikus levelezést minden alkalmazott használja mindkét osztályon. Az interjúkból kiderült, hogy a levélküldések időköze leírható egy 900 másodperc várható értékű exponenciális eloszlással. A levelek mérete pedig leírható egy olyan egyenletes eloszlással, mely szerint a levélméret 500 és 2000 bájt között van. Az összes e-mail az Ethernet hálózaton lévő e-mail szerverhez továbbítódik, ahol pedig a megfelelő felhasználó postafiókjában kerül tárolásra.

Ha egy felhasználó el akar olvasni egy levelet, akkor azt az e-mail szervertől kell lekérnie. A postafiók ellenőrzéseinek ideje leírható egy Poisson-eloszlással, melynek várható értéke 900 másodperc. Az erre használt üzeneteknek a mérete 60 bájt. Ha egy felhasználó le szeretne tölteni egy levelet, akkor a szerver beolvassa a felhasználóhoz tartozó állományt és továbbítja a kért levelet az adott alkalmazott számítógépére. Az állomány beolvasásának és a benne lévő üzenetek feldolgozásának ideje leírható egy egyenletes eloszlással, mely 3 és 5 másodperc közötti értékeket vehet fel. A levelek mérete egy normális eloszlással írható le, melynek várható értéke 40000 bájt és standard eltérése 10000 bájt.

Mindkét osztályon nyolc alkalmazott van, akiknek saját számítógépe van, és akik a fájlszervertől is kérnek le állományokat. Ezeknek a kéréseknek az érkezési időköze leírható egy 900 másodperc várható értékű exponenciális eloszlással. A kérések mérete egyenletes eloszlást követ, 10 bájt minimummal és 20 bájt maximummal. Ezeket a kéréseket kizárólag a token ring hálózatban lévő fájlszervernek küldik. Egy kérés érkezésekor a szerver beol-



5.50. ábra. Hálózati topológia. Az ábra színes változata a 810. oldalon látható.

vassa a kért állományt és elküldi az adott számítógépnek. Ez a feldolgozás csak nagyon kis késleltetést jelent. Az állományok mérete egy normális eloszlással írható le, melynek várható értéke 200000 bájt és standard eltérése 25000 bájt.

A hang alapú üzenetküldést mindkét osztályon csak a vezetők használják, akik általában csak a saját részlegükben lévő alkalmazottaknak küldenek ilyen üzeneteket. A továbbító alkalmazás először kapcsolatot létesít az alkalmazott számítógépével. Ha ez felépült, akkor továbbítja az üzenetet. Ezek mérete normális eloszlással írható le, melynek várható értéke 50000 bájt és standard eltérése 1200 bájt. Beérkezési időközük szintén normális eloszlással írható le, melynek várható értéke 1000 másodperc és standard eltérése 10 bájt.

Az összes üzenetforrás a TCP/IP protokollkészletet használja, és a csomagok elkészítésének becsült ideje 0.01 milliszekundum.

A hálózat topológiájának hasonlónak kell lennie a 5.50. ábrán a COMNET-ben láthatóhoz.

A következő jelentések használhatók a szimulációnál:

- Kapcsolatjelentések: Csatorna-kihasználtság és Ütközési statisztikák az egyes kapcsolatok esetén.
- Csomópontjelentések: A beérkezett üzenetek száma az egyes csomópontok esetén.
- Üzenet és válasz jelentések: Az üzenetek késleltetése az egyes csomópontok esetén.
- Session Source jelentések: Az üzenetek késleltetése az egyes csomópontok esetén.

A modell futtatásakor jóval nagyobb válaszidőt fog észlelni a fájlszervernél. Ha a szolgáltatás-minőségi szint kisebb válaszidőt igényel, akkor milyen megoldást lehet javasolni annak csökkentésére? Jó megoldás lenne még egy fájlszerver üzembe helyezése az

Ethernet hálózatban? Mi mást lehetne még módosítani?

Megjegyzések a fejezethez

Law és Kelton monográfiája [280] jó áttekintést ad a hálózati rendszerekről. A hálózatok osztályozásával kapcsolatban két, magyarul is megjelent monográfiát ajánlunk, melyek szerzői Sima, Fountain és Kacsuk [432], illetve Tanenbaum [457]

A valószínűségszámítás alapjaival kapcsolatban Prékopa András [377] és Rényi Alfréd [393] könyveit ajánljuk. A leggyakoribb statisztikai eloszlásokat Banks könyve [29] alapján foglaltuk össze. A sűrűségfüggvények ábrázolására használt COMNET szimulációs modellezési eszközt ismertetése megtalálható a CACI két kiadványában [65, 235].

A szimuláció matematikai hátterével kapcsolatban Kátai Imre jegyzetét [244], a sorbanállás elméletével kapcsolatban pedig Kleinrock könyvét [256] és Sztrik János hálózatról letölthető tananyagát [452] ajánljuk.

A csatornakapacitás definíciója megtalálható például az Interneten is elérhető szótárakban [217, 489]. Az információ- és kódelmélettel kapcsolatos részletek megtalálhatók például Jones és Jones könyvében [234].

A hosszú távú függőséggel foglalkoznak például Taqqu és társai [287, 460].

A hálózatmodellezésben előforduló leggyakoribb eloszlások becsléseit leíró 5.1. ábra Banks, Carson és Nelson könyvéből [29] származik.

Az OPNET szoftver és annak dokumentációja letölthető a [356]-ben megadott címről. Ez a dokumentáció részletesen tárgyalja a szimuláció egyes szakaszait is.

A forgalom ingadozásának hatását Gyires Tibor cikke [184] alapján elemeztük.

A hálózati forgalommal kapcsolatos mérésekről számolnak be Leland és társai [285, 286], valamint Crovella és Bestavros [94].

Hálózatok önhasonlóságával foglalkoznak Erramilli, Narayan és Willinger [121], Willinger és társai [497], valamint Beran [42]. Mandelbrot [307], Paxson és Floyd [363], valamint Mandelbrot és van Ness [308] tanulmányozták a hosszú távon függő folyamatokat.

Forgalomirányítási modellek találhatók például a következő művekben: [17, 199, 227, 320, 347, 348, 359, 497].

Forgalmi adatokat tartalmaz [43, 114, 181, 363]. Az 5.22. ábra Listanti és Eramo cikkéből [292] származik.

A hosszútávú függőséget elemzi Addie, Zukerman és Neame [3], Duffield és O'Connell [113], valamint Erramilli, Narayan és Willinger [121].

A *fekete doboz* modellezés kifejezést Willinger és Paxson [495] vezette be 1997-ben.

Az *Ockham borotvája* nevű elvről például Francis Heylighen honlapja [206] tartalmaz adatokat. A Snifferről a Network Associates cég honlapján [316] találhatók további részletek.

Willinger, Taqqu, Sherman és Wilson [496] egy szerkezeti modellt elemeznek. Crovella és Bestavros [94] a World Wide Web forgalmát elemezték.

Az erős ingadozásnak a hálózati torlódásokra gyakorolt hatásával foglakozott például Neuts [347], valamint Molnár Sándor, Vidács Attila és Nilsson [335].

A Hurst-paraméter hatását elemzi Addie, Zukerman és Neame [3].

A Benoit-csomag letölthető a hálózatról [468].

A Pareto-modellt vizsgálta Addie, Zukerman és Neame [3].

6. Párhuzamos számítások

A folyamatos szintjén párhuzamos számítások[1] fő célja, hogy a feladatokat több processzor segítségével gyorsabban oldjuk meg, mint egy processzoron. Ezek a processzorok tartozhatnak egyetlen számítógéphez vagy különböző számítógépekhez, amelyek egy hálózaton keresztül kommunikálnak. A párhuzamos végrehajthatósághoz mindkét esetben szükség van arra, hogy a feladatot egyidejűleg megoldható alfeladatokra osszuk.

Bár a párhuzamos számítások története korábban kezdődött, az első párhuzamos számítógépnek a 64 processzort tartalmazó ILLIAC IV-et szokták tekinteni, amely 1972-ben kezdett működni. A párhuzamos számítógépek a nyolcvan évek végén indultak gyors fejlődésnek, amikor is több új céget alapítottak különböző párhuzamos számítógépek gyártására. Sajnos abban az időben nehezen indult el a szoftver fejlesztése és a kifejlesztett szoftver nem volt hordozható. Ezért a párhuzamos számítógépeket csak a tudományos és műszaki élet leginkább számításigényes területein alkalmazták, mivel a piac túl kicsi volt ahhoz, hogy a jelentős fejlesztési költségeket meg tudja fizetni. Ezért számos cég abba is hagyta a fejlesztést.

A dolog pozitív oldala volt az a felismerés, hogy olcsó párhuzamos számítógépek hozhatók létre az elterjedt személyi számítógépek vagy munkaállomások összekapcsolásával. Ahogy a hálózatok gyorsabbá váltak, ezek az úgynevezett *klaszterek* rövidesen ugyanolyan nagyságrendű sebességet értek el, mint a speciális célú gépek. Jelenleg a világ 500 legnagyobb számítógépének folyamatosan frissített listáján a gépek 42 százaléka klaszter.

A párhuzamos számítások eszköztárát gyarapítják azok a multiprocesszoros gépek is, amelyeket ugyan a web és más területek kiszolgálására terveztek, de párhuzamos számítások elvégzésére is alkalmasak. Végül a szoftver hordozhatóságával kapcsolatos problémák is megoldódtak a párhuzamos programozás széles körben elterjesztett szabványai segítségével. A két legfontosabb szabvány az MPI és OpenMP – ezeket a 6.3. alfejezetben fogjuk ismertetni.

Összegezve, jelenleg elfogadható költségű hardverbázis áll rendelkezésre. A terület azonban még nem érte el fejlődési határait, mivel a párhuzamos szoftver fejlesztése komoly nehézségeket okoz. Míg egy soros program megírásához elegendő egy algoritmust találni, azaz elemi műveleteknek az adott feladatot megoldó sorozatát, majd az algoritmust egy programozási nyelven leírni, addig a párhuzamos számítások további kihívásokat jelentenek:

[1] Párhuzamos számítás több szinten valósítható meg, így utasítás-szinten, szál-szinten és folyamat-szinten. Ez a fejezet elsődlegesen a folyamat szintű párhuzamos feldolgozással foglalkozik. *A lektor.*

- Az elemi műveleteket olyan taszkokká kell csoportosítani, melyek párhuzamosan megoldhatók.
- A taszkokat ütemezni kell a processzorokra.
- Az adatokat az architektúrától függően el kell osztani a memóriamodulok között.
- A folyamatokat és szálakat kezelni kell, azaz elindítani, megállítani és így tovább. A kommunikációt és a szinkronizációt meg kell szervezni.

Természetesen nem elég a műveletek csoportosítását, az ütemezést és így tovább megoldani – olyan megoldásokat kell találni, amelyek gyors programokhoz vezetnek. A teljesítménymértékeket és a teljesítmény optimalizálásának módszereit a 6.2. alfejezetben tárgyaljuk, ahol a fent említett feladatok megoldását is vizsgáljuk. A különböző párhuzamos architektúrák és programozási modellek – a soros modellektől eltérően – különböző algoritmusokat részesítenek előnyben.

Ennek következtében a párhuzamos algoritmusok bonyolultabbak, mint a sorosak. A párhuzamos algoritmusok vizsgálatához gyakran alkalmaznak egyszerűsített modelleket. Például a népszerű PRAM (lásd a 6.4. alfejezetet) modell nem veszi figyelembe a kommunikációs és szinkronizálási költségeket.

Hasonlítsuk össze a párhuzamos számításokat az osztott és a konkurens számításokkal. Az *osztott számítások* összekapcsolt processzorokat használnak és a megoldandó problémákat kisebb részfeladatokra osztják, de a felosztás céljai különbözők lehetnek. Míg a *párhuzamos számításoknál* a részfeladatok *egyidejűleg* oldódnak meg, az osztott számításoknál a részfeladatok *különböző helyeken* oldódnak meg, *különböző erőforrásokat felhasználva.* Ezek a tulajdonságok átfedhetik egymást, ezért számos alkalmazás egyszerre párhuzamos és osztott, a lényeg azonban különböző. A párhuzamos számításoknál a felépítés homogén, és a főcél a számítások gyorsabb elvégzése, az osztott számításoknál a felépítés heterogén és az alkalmazások számára gyakran kedvező a különböző erőforrások összekapcsolása. A párhuzamos rendszerek általában hosszabb ideig léteznek és kiszámíthatók, míg az osztott alkalmazások olyan elemekből állnak, melyek a futás során kapcsolódtak össze.

A *konkurens számítások* nem feltétlenül igényelnek több processzort, és azt a körülményt emelik ki, hogy egyidejűleg több részszámítás van folyamatban. Ez a fontos tulajdonság garantálja, hogy az eredmény a részszámítások tetszőleges sorrendje és átfedése mellett is helyes lesz. Ezért a párhuzamosság és a konkurencia viszonya ahhoz hasonlítható, hogy egyidejűleg több könyvet olvasunk (ami a gyakorlatban valószínűleg nem oldható meg). Így a konkurens számítás vagy párhuzamos, vagy nem párhuzamos, a párhuzamos számítás viszont mindig konkurens. Egyetlen kivétel az adatok párhuzamossága, melyben egyetlen program utasításait alkalmazzuk párhuzamosan különböző adatokra. Ezt a megközelítést követi a SIMD architektúra.

A nagy sebességnek köszönhetően a párhuzamos számítások tipikus alkalmazási területei a természettudományok és a műszaki feladatok megoldása, különösen a numerikus számítások és a szimuláció. Ezeknek az alkalmazásoknak egyre nagyobb a számítási igénye, mivel a nagyobb számítási kapacitás részletesebb modellek alkalmazását és így pontosabb eredmények elérését teszi lehetővé. A párhuzamos számítások másik előnye a nagyobb memóriakapacitás, ami lehetővé teszi, hogy több adatot tároljunk az olyan memóriaszinteken, mint a gyorsítótár.

A fejezet további részének felépítése a következő. Az 6.1. című alfejezetben röviden áttekintjük és osztályozzuk a jelenlegi párhuzamos architektúrákat. Azután a 6.2. alfejezet-

ben olyan alapfogalmakat vezetünk be, mint a taszk és a folyamat, majd elemezzük a hatékonysági mértékeket és a hatékonyság javítására szolgáló általános módszereket. Ezután a 6.3. alfejezet a párhuzamos programozás modelljeit ismerteti, elsősorban a népszerű MPI és OpenMP szabványokra koncentrálva. Erre a gyakorlati alapra épül a fejezet hátralévő része, amely formalizált módon tárgyalja a legegyszerűbb párhuzamos algoritmusokat és tervezési módszereket. A soros algoritmusoktól eltérően a párhuzamos algoritmusok esetén nincs általánosan elfogadott tervezési és elemzési módszer, bár számos modellt javasoltak erre a célra. A modellek mindegyike bizonyos kompromisszumot valósít meg az egymásnak ellentmondó célok – nevezetesen a valódi architektúrák pontos tükrözése és a tervezés, valamint elemzés egyszerűsége – között. A 6.4. alfejezet áttekintést ad a modellekről, a 6.5. alfejezet pedig konkrét algoritmusokat mutat be – többek között a kiválasztás, összefésülés és rendezés feladatok megoldására.

Az algoritmusok elemzése során mindvégig nagy figyelmet fordítunk az adott feladat soros és párhuzamos megoldási idejének, valamint a kétféle megoldás során végzett munkának az összehasonlítására. Az algoritmusok elemzésének korszerű felfogását követve nem elégszünk meg a vizsgált hatékonysági mérték értékének egy felső korlátjával, hanem lehetőleg pontosan jellemezzük a legrosszabb esetben várható futási idő és a végzett munka nagyságrendjét.

6.1. Párhuzamos architektúrák

A párhuzamos architektúrák egyik egyszerű és közismert osztályozását Flynn javasolta. A számítógépeket négy osztályba sorolta: SISD, SIMD, MISD, és MIMD architektúrák.

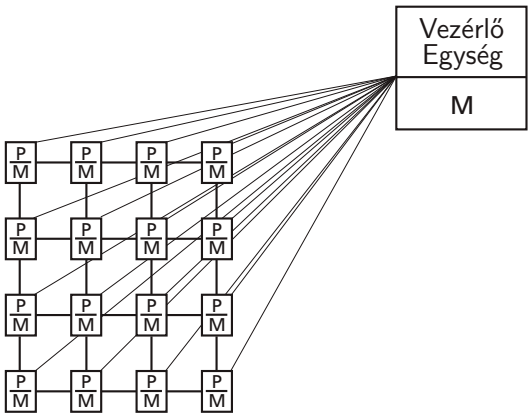
- SI *egyszeres utasításáramot* (**S**imple **I**nstruction stream) jelent, azaz egyidejűleg egyetlen utasítás hajtódhat végre.
- MI *többszörös utasításáramot* (**M**ultiple **I**nstruction stream) jelent, azaz különböző processzorok egyidejűleg különböző utasításokat is végrehajthatnak.
- SD *egyszeres adatáramot* (**S**imple **D**ata stream) jelent, azaz egyidejűleg egyetlen adattal hajtódhatnak végre műveletek.
- MD *többszörös adatáramot* (**M**ultiple **D**ata stream) jelent, azaz egyidejűleg különböző adatokkal is végezhetők műveletek.

A SISM felépítés a Neumann-elvű számítógépeket jelenti. MISD típusú számítógépeket valószínűleg sohasem építettek. A korai számítógépek SIMD felépítésűek voltak, ma a legtöbb párhuzamos számítógép MIMD felépítésű. Bár ennek a sémának kicsi az osztályozási ereje, mégis széles körben alkalmazzák.

A következő osztályozás a párhuzamos gépeket a SIMD, SMP, ccNUMA, nccNUMA, NORMA, clusters, és rács osztályokba sorolja.

SIMD architektúrák

Amint azt a 6.1. ábra mutatja, a SIMD számítógép egy nagyteljesítményű vezérlő processzorból és több, kisebb teljesítményű feldolgozó elemből áll. A feldolgozó egységeket gyakran rácsszerűen kötik össze úgy, hogy minden egység a közvetlen szomszédaival kommunikál. A vezérlő processzor – a soros gépek processzorához hasonlóan – egymás után ol-



6.1. ábra. SIMD architektúra.

vassa és dekódolja az utasításokat. A soros esetben a processzor a beolvasott utasítást a saját memóriájában lévő adatokkal hajtja végre. A párhuzamos esetben a vezérlő processzor minden feldolgozó egységhez elküldi a beolvasott utasítást és azok egyidejűleg hajtják végre ezt az utasítást a saját memóriájukban tárolt adatokon. Példaként tekintsük az LD reg, 100 utasítást. Ennek hatására minden processzor betölti a 100-as memóriacím tartalmát a regiszterbe, de a 100-as cím processzoronként különböző memóriarekeszt jelent. Tehát a processzorok – a SIMD felépítésnek megfelelően – azonos utasítást hajtanak végre különböző adatokkal. Az

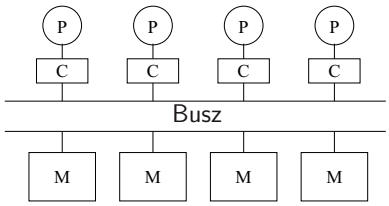
if *test* then *if_branch* else *else_branch*,

utasítás hatására a processzorok egyidejűleg elvégzik a tesztelést, azután egy részük az *if_branch* ágon folytatja, míg mások tétlenek maradnak, és végül az előbbiek maradnak tétlenek és az utóbbiak végzik el az *else_branch* ágat. Emiatt a SIMD felépítésű számítógépek szabályos szerkezetű számítások elvégzésére alkalmasak. Ez a felépítés történetileg lényeges, de ma már nem alkalmazzák.

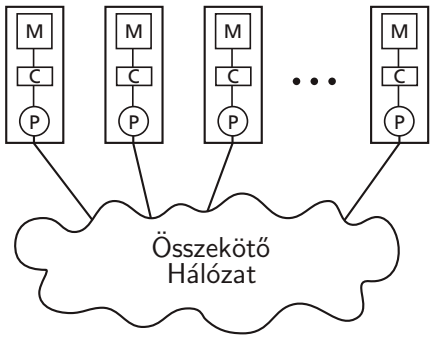
Szimmetrikus multiprocesszorok

A szimmetrikus multiprocesszorok (SMP) egy közös memóriához kapcsolódó processzorokat tartalmaznak. A hardver minden processzornak hozzáférést biztosít – a szokásos betölt/tárol műveletekkel – minden memóriarekeszhez. Ezért a programokat – beleértve az operációs rendszert is – elég egy példányban tárolni. A memória fizikailag modulokra bontható, de a hozzáférési idő azonos minden processzor-modul párra (ez indokolja a szimmetrikus jelzőt). A processzorokat egy adatátviteli vonal, egy mátrix-kapcsoló vagy egy kapcsolóhálózat köti össze a memóriával (lásd 6.2. ábra). A memóriához való hozzáférés mindegyik esetben késleltetéssel történik, ami részben a hálózati erőforrásokért folyó verseny következménye, és a processzorok számával nő.

Minden processzornak egy- vagy többszintes gyors hozzáférésű gyorsítótára is van. A fő memória és a gyorsítótár között az adatok a gyorsító adatátviteli vonalakon mozognak. Ha az egyes adatelemeket több gyorsítótárban is tároljuk, akkor koherenciaproblémák lépnek fel. Például *rossz megosztásról* beszélünk, ha több processzor is hozzáfér ugyanahhoz a gyorsítóvonalhoz, de annak különböző részét használják.



6.2. ábra. Busz-alapú SMP-architektúra.



6.3. ábra. ccNUMA architektúra.

Gyorsítótáras NUMA architektúrák

LeopoldParallelFig_02.07

A ccNUMA (gyorsítótárral kohérens nem-egyenletes memóriahozzáférés) nem egységes memóriahozzáférést jelent, ellentétben az előző osztály szimmetrikus hozzáférésével. A 6.3. ábra mutatja ezeknek a számítógépeknek a felépítését.

Az ábra szerint minden processzorhoz tartozik egy *helyi memória* (gyorsítótár), amelyhez a hozzáférés gyorsabb, mint a memória további, *távoli memóriának* nevezett részéhez. A teljes memóriához szabványos betöltő/tároló műveletekkel férhetünk hozzá, ezért a programokat – az operációs rendszerek programjait is – csak egyszer kell tárolni. Az SMP-khez hasonlóan minden processzornak egy vagy többszintes gyorsítótára van; a gyorsítótár szintjeinek összhangját a hardver biztosítja.

Gyorsítótár nélküli NUMA architektúrák

Az nccNUMA architektúrák abban különböznek a ccNUMA architektúráktól, hogy a hardver csak a helyi memóriából származó adatokat tölti be a gyorsítótárba. A távoli memóriához való hozzáférés a szokásos beviteli/kiviteli műveletekkel megvalósítható, de először az operációs rendszernek kell a megfelelő lapot a helyi memóriába töltenie. Ez a különbség egyszerűsíti a hardver tervezését, és ezért az nccNUMA felépítésű gépek több processzort tartalmaznak. Hátrány viszont, hogy az operációs rendszer bonyolultabb, a távoli memóriához való hozzáférés ideje nagyobb. A 6.3. ábrán bemutatott felépítés az nccNUMA típusú számítógépekre is vonatkozik.

Távoli memóriához való gyors hozzáférés nélküli architektúrák

A NORMA (**NO R**emote **M**emory **A**ccess Architectures) architektúra abban különbözik az

előző osztálytól, hogy a távoli memóriához lassú beviteli/kiviteli műveletekkel lehet hozzáférni, szemben az előző osztály betöltő/tároló műveleteivel. Amint azt a 6.3. ábra mutatja, minden csúcs egy processzorból, gyorsító memóriából és helyi memóriából áll, tartalmazza az operációs rendszer egy saját példányát, vagy legalább annak központi részét. Míg az SMP, ccNUMA és nccNUMA architektúrákat rendszerint a közös memóriájú gépekhez sorolják, a NORMA típusú gépeket, klasztereket és grideket az osztott memóriájúakhoz.

Klaszterek

A ***klaszter*** olyan párhuzamos vagy osztott számítógéprendszer, amely egységes felépítésű, egymással összekötött számítógépekből áll. A számítógép itt személyi számítógépet, munkaállomást vagy – egyre gyakrabban – szimmetrikus multiprocesszort jelent, azaz olyan hálózati elemet, amely processzorból (vagy processzorokból), memóriából, esetleg perifériákból és operációs rendszerből áll. Az egységes feldolgozó elemekből álló rendszert SSI tulajdonságúnak (**S**ingle **S**ystem **I**mage) mondjuk, ha csak a rendszerbe lehet bejelentkezni, az egyes elemekre nem – vagy ha egyetlen fájlrendszer van. Természetesen az SSI tulajdonság fokozatos, ezért a határvonal az osztott rendszerek és klaszterek között nem éles. A NORMA rendszerek és klaszterek közötti határvonal is bizonytalan, mivel attól is függ, hogy eleve egységes rendszert terveztek vagy meglévő, eredetileg különálló komponenseket kapcsoltak össze.

A klaszterek osztályozhatók aszerint, hogy párhuzamos számításokra, nagyméretű számításokra vagy széles körű hozzáférhetőségre alkalmazzák-e őket. A párhuzamos számításokra használt klaszterek tovább oszthatók a ***kiemelt klaszterekre,*** melyeket eleve úgy terveztek, hogy párhuzamos gépekként üzemeljenek, vagy ***kampusz-klaszterekre,*** amelyek az idő egy részében párhuzamos gépekként alkalmazott osztott rendszerek. A dedikált klaszterek elemei rendszerint nem rendelkeznek perifériákkal és nagy sebességű hálózati vonalakkal vannak összekötve. A kampusz-klaszterek viszont gyakran asztali személyi számítógépek, melyek között a belső kommunikáció közönséges hálózati vonalak segítségével valósul meg.

Gridek

Foster és Kesselman szerint „A grid egy hardver/szoftver infrastruktúra az erőforrások közös használatához és a problémák megoldásához”. A gridek lehetővé teszik az olyan erőforrásokhoz való összehangolt hozzáférést, mint a processzorok, memóriák, adatok, perifériák stb. A gridek abban különböznek a párhuzamos számítási architektúrától, hogy nagyok, heterogének és dinamikusan változnak. A gridek vezérlése ezért bonyolult feladat.

6.2. Hatékonysági mértékek és optimalizálás

Ebben az alfejezetben először a gyakorlatban használt hatékonysági mértékeket és optimalizálási módszereket, azután pedig az algoritmusok aszimptotikus elemzésénél felhasználható fogalmakat tekintjük át.

6.2.1. Hatékonyság a gyakorlatban

Amint azt a bevezetésben leírtuk, a párhuzamos számítások során a feladatokat ***taszkokra*** bontjuk és a taszkokat egymástól függetlenül megoldjuk. A taszkokat ***folyamatokként*** vagy ***szálakként*** valósítjuk meg. Más szavakkal, a folyamatok végrehajtás alatt lévő programok. Minden folyamattal kapcsolatban tárolunk olyan erőforrásokkal kapcsolatos adatokat, mint a memóriaszegmensek, fájlok és jelek, míg a szálak a folyamatokon belül léteznek és a többszörös szálak megosztják egymás között az erőforrásokat. Adott folyamat szálai hozzáférnek a közös memóriához, míg a folyamatok rendszerint üzenetekkel kommunikálnak. Minden szálnak van egy utasításszámlálója vagy más regiszterértéke, valamint egy verme a helyi változók tárolására. A folyamatokat tekinthetjük az erőforráshasználatot segítő egységnek, míg a szálakat a központi feldolgozó egységen való végrehajtást segítő egységnek. Mivel kevesebb információ tárolására van szükség, gyorsabban lehet szálakat létrehozni, megszüntetni és egyik szálról a másikra átkapcsolni, mint ugyanezeket a műveleteket folyamatokkal elvégezni.

Az architektúrától függ, hogy szálakat vagy folyamatokat alkalmazunk-e. A közös memóriájú gépeken a szálak rendszerint gyorsabban futtathatók, ugyanakkor viszont a folyamatokat felhasználhatjuk a programok hordozhatóságának biztosítására. A szálak alkalmazhatók, ha van egy szoftver réteg (osztott közös memória), amely megvalósítja a közös memória absztrakcióját, de ezeknek a szálaknak nagyobbak a kommunikációs költségei.

Míg a taszkok fogalma a problémákhoz kapcsolódik, a folyamatok és szálak fogalma a megvalósításhoz kötődik. Amikor egy algoritmust tervezünk, rendszerint nagyszámú taszkot azonosítunk, amelyek potenciálisan futhatnak párhuzamosan, majd ezek közül többet ugyanarra a folyamatra vagy szálra képezünk le.

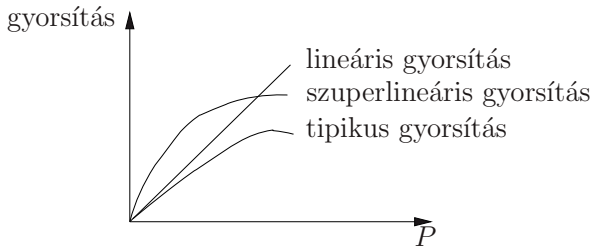
Párhuzamos programok két stílusban írhatók és ezek a stílusok keveredhetnek: az egyik stílusra ***adatpárhuzamosság*** jellemző, azaz ugyanazt a műveletet egyidejűleg különböző adatokra alkalmazzuk. Ez a művelet lehet egy gépi utasítás, mint a SIMD architektúrákban, vagy egy összetett művelet, például egy függvény alkalmazása. Utóbbi esetben különböző processzorok különböző műveleteket hajtanak végre. A másik stílus jellemzője a ***taszkpárhuzamosság,*** azaz a folyamatok/szálak különböző taszkokat hajtanak végre. Mivel egy függvény külső konstrukcióként tartalmazhat például **if** vagy **case** parancsot, az adatpárhuzamosság és taszkpárhuzamosság közti határvonal nem éles.

A folyamatok segítségével megvalósított párhuzamos programok tovább osztályozhatók: az SPMD (**S**ingle **P**rogram **M**ultiple **D**ata) programozási stílus azt jelenti, hogy minden folyamat ugyanazt a programot futtatja, míg az MPMD (**M**ultiple **P**rogram **M**ultiple **D**ata) stílus alkalmazásakor a folyamatok különböző programokat futtatnak. Az MPMD programok taszkpárhuzamosak, míg az SPMD programok mind taszkpárhuzamosak, mind pedig adatpárhuzamosak lehetnek. SPMD módban a taszkpárhuzamosságot feltételes utasítások segítségével fejezzük ki.

Mivel a párhuzamos számítások elsődleges célja a programok gyors futtatása, a teljesítménymértékek fontos szerepet játszanak ezen a területen. Természetes abszolút mérték a végrehajtási idő, de még gyakrabban használnak egy relatív mértéket, a gyorsítást. Adott problémára a ***gyorsítást*** a

$$gyorsítás(p) = T_1/T_p$$

hányadossal definiáljuk, ahol T_1 a leggyorsabb ismert soros algoritmus futási ideje p processzoron. A környezettől függően a gyorsítás vonatkozhat p folyamat vagy p szál alkalma-



6.4. ábra. Ideális, tipikus és szuperlineáris gyorsítási görbék.

zására is. A gyorsítással kapcsolatos, de ritkábban használt mérték a ***hatékonyság,*** melyet a

$$hatékonyság(p) = gyorsítás(p)/p$$

összefüggéssel definiálunk.

Ettől a definíciótól függetlenül a *hatékonyságot* a jó teljesítmény szinonimájaként is használják.

A 6.4. ábra bemutatja az ideális, tipikus és szuperlineáris gyorsítási görbéket. Az ideális görbe azt a feltevést tükrözi, hogy ha a processzorok számát megkétszerezzük, akkor a végrehajtási idő felére csökken. Ezért az ideális gyorsítás egységnyi hatékonyságnak felel meg. A szuperlineáris gyorsítás a gyorsításnak köszönhetően fordulhat elő, azaz akkor, ha a több processzor alkalmazása megnöveli a gyorsítótár méretét, és így több adathozzáférés szolgálható ki a gyorsítótárból – ahelyett, hogy a lassú központi memóriához kellene fordulni.

A tipikus gyorsítás az ideális gyorsításnál kisebb és csak bizonyos processzorszámig nő. Emellett több processzor alkalmazása lassítja a programot. A tipikus és ideális gyorsítás közötti különbségnek több oka van:

- *Amdahl törvénye* azt mondja, hogy minden programnak van egy s soros hányada, amely nem párhuzamosítható. Ezért $T_p > s$, és így $gyorsítás(p) < T_1/s$, azaz a gyorsítás mindig kisebb, mint egy konstans érték. Szerencsére egy másik gyakorlati megfigyelés, a ***Gustafson–Barsis-törvény*** csökkenti az Amdahl-törvény gyakorlati szerepét. A Gustafson-Barsis-törvény szerint a tipikus alkalmazásokban a párhuzamos algoritmus alkalmazása nem korlátozódik egy rögzített méretű probléma megoldására, hanem egyre nagyobb méretű feladatokat oldunk meg. Ebben az esetben s lassabban nőhet, mint T_1, és így T_1/s már nem konstans.
- A taszkkezelés, azaz a taszkok indítása, megállítása, megszakítása, valamint a folyamatok és szálak ütemezése bizonyos többletterhelést okoz. Továbbá az is rendszerint igaz, hogy az elvégzendő munkát nem lehet egyenletesen elosztani a folyamatok/szálak között.
- A kommunikáció és a szinkronizáció lassítja a programot. A kommunikáció az adatcserét jelenti, míg a szinkronizáció más típusú koordinációt jelent, például a kölcsönös kizárás biztosítását. A kommunikációs és szinkronizációs költségek még a nagy sebességű vonalakkal rendelkező hálózatokban is nagyságrendileg nagyobbak, mint a számítási költségek. Ennek oka a fizikai átviteli költségek mellett a protokoll számítási többletigénye és a hálózati erőforrásokért való versenyzés okozta késedelem.

A fenti tényezők hatásának minimalizálásával a teljesítmény javítható. Amdahl törvényét nehéz megkerülni, kivéve azt az esetet, ha új algoritmust tudunk tervezni, amelyre nézve s értéke kisebb – esetleg olyan áron, hogy T_1 értéke nagyobb lesz. Az algoritmikus fogalmakat később tekintjük át – most a gyakorlatban használt jellemzőket mutatjuk be.

Amint korábban láttuk, a taszkokat olyan folyamatokként vagy szálakként valósítjuk meg, amelyek rendszerint többszörös taszkokra vonatkoznak. A nagyobb teljesítmény érdekében a folyamatok/szálak szemcsézettségét az architektúrának megfelelően választjuk meg. Sok folyamat/szál szükségtelenül megnöveli a taszkkezelési költségeket, míg kevés folyamat/szál a számítógép alacsony kihasználtságát eredményezi. Ezért célszerű több folyamatot/szálat ugyanahhoz a processzorhoz rendelni, mivel így a processzorok átkapcsolhatnak, amikor bevitel/kivitel vagy más ok miatt várakozniuk kellene. A nagy szemcséjű folyamatok/szálak előnye a jobb kommunikáció/számítások arány, míg a finomabb szemcséjű folyamatok/szálak alkalmasabbak a terhelés egyenletes elosztására.

A terheléskiegyenlítés végezhető statikus vagy dinamikus módon. Ha a taszkok futási ideje előre megbecsülhető, akkor a statikus sémákat részesítjük előnyben. Ezeknél a módszereknél a programozó minden folyamathoz/szálhoz hozzárendel bizonyos számú folyamatot/szálat úgy, hogy az összköltség minden processzoron hasonló nagyságú legyen. A dinamikus megoldásra példa a mester-szolga séma. Ebben a sémában először a mester folyamat hozzárendel minden processzorhoz egy folyamatot. Ezután valahányszor egy szolga befejez egy taszkot, értesíti erről a mestert és az új taszkot rendel a processzorhoz mindaddig, amíg minden taszk végre nem hajtódik. Ez a séma jó terheléskiegyenlítést biztosít, aminek az ára a taszkkezelés okozta többletterhelés.

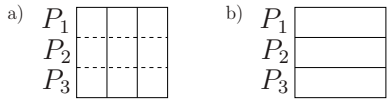
A hatékonyság növelésének legjobb eszköze rendszerint a kommunikációs/szinkronizációs költségek csökkentése. Természetesen javulás érhető el az architektúra vagy a rendszerszoftver változtatásával, például a *késleltetésnek* – azaz a távolról érkező adatra várás idejének – csökkentésével, vagy a ***sávszélesség*** – azaz az időegység alatt átvitt adatmennyiség – növelésével.

Az algoritmustervező vagy felhasználói programozó csökkentheti a kommunikációs/szinkronizációs költségeket a beavatkozások számának csökkentésével. Az egyik fontos megközelítés ennek a minimalizálásnak az elvégzésére a helyi optimalizálás. A ***lokalitás*** a (soros vagy párhuzamos) programok olyan tulajdonsága, amely tükrözi az azonos adatokhoz időben vagy térben koncentrálva való hozzáférés mértékét. Az osztott memóriájú architektúrákban például az adatokat annál a processzornál célszerű tárolni, ahol felhasználjuk őket. A lokalitás javítható kódtranszformációval, adattranszformációval vagy a kettő kombinációjával. Például tekintsük a következő programrészlet végrehajtását három processzoron:

```
for (i=0; i<N; i++) in parallel
  for (j=0; j<N; j++)
    f(A[i][j]);
```

Itt az *in parallel* kifejezés azt jelenti, hogy az iterációk egyenletesen vannak elosztva a processzorok között úgy, hogy P_0 futtatja az $i = 0, \ldots, i/3$ iterációt, P_1 futtatja az $i = i/3 + 1, \ldots, 2i/3$ iterációkat és P_2 futtatja az $i = 2i/3+1, \ldots, N-1$ iterációkat (az indexek szükség szerint kerekítve értendők). Feltesszük, hogy az f függvény mellékhatásoktól mentes.

A 6.5(a) ábrán lévő adatokkal a lokalitás kicsi, mivel sok hozzáférés szükséges a távoli memóriához. A lokalitás javítható úgy, hogy az adatok elosztását a 6.5(b) ábrán láthatóra



6.5. ábra. Lokalitás optimalizálása az adatok transzformálásával.

változtatjuk, vagy a programot változtatjuk a következőre:

```
for (j=0; j<N; j++) in parallel
  for (i=0; i<N; i++)
    f(A[i][j]);
```

A második lehetőség, a kódtranszformáció előnye, hogy alkalmazható szelektíven a kód egy részére, míg az adattranszformáció a program egészére hat és míg az egyik helyen javulást okoz, egy másik helyen hátrányos lehet. Az adatelosztások mindig helyesek, míg a kódtranszformációnak figyelembe kell vennie az *adatfüggőségeket,* melyek az utasítások sorrendjéből következnek. Például az

```
a = 3;          (1)
b = a;          (2)
```

kódrészletben az (1) és (2) utasítások között adatfüggőség van. A két utasítás sorrendjének megváltoztatása hibás programot eredményezne.

Közös memóriájú architektúrákon a programozó nem határozza meg az adatok eloszlását. A programok gyorsabban futnak, ha az együtt felhasznált adatok a gyorsítótárnak ugyanazon az adatátviteli vonalán vannak. A közös memóriájú architektúráknál az adatok elosztását a fordítóprogram végzi, például a C nyelvben sorfolytonosan. A programozónak csak közvetett befolyása van azzal, hogyan deklarálja az adatszerkezeteket.

A kommunikációs költségek csökkentésének egy másik módja az adatok másolása. Például kifizetődő a gyakran használt adatok több processzoron való tárolása, vagy rövid számítások megismétlése az eredmények elküldése helyett.

A szinkronizáció szükséges a helyességhez, de lassítja a program végrehajtását, egyrészt a saját végrehajtási ideje miatt, másrészt azzal, hogy a processzorokat arra kényszeríti, hogy egymásra várjanak. Ezért a szinkronizáció széles körű alkalmazását célszerű elkerülni. Például a kritikus szakaszok (melyekben a folyamatok/szálak ugyanahhoz az erőforráshoz igényelnek kizárólagos hozzáférést) számát célszerű minimális értéken tartani. *Sorossá tételnek* nevezzük azt, amikor legfeljebb egy folyamat aktív olyankor, amikor van várakozó folyamat.

Végül a hatékonyság javítható a késleltetés elrejtésével, azaz a kommunikáció és a számítás közötti párhuzamossággal. Például egy folyamat valamivel előbb elkezd egy távoli helyről való olvasást (előre való betöltés) vagy a következő számítással párhuzamosan végzi a távoli memóriába való írást.

6.2.2. Hatékonyság az elemzésekben

Az algoritmusok elemzése során az igényelt erőforrások mennyiségét *abszolút* és *relatív* mennyiségekkel jellemezzük.

Ezeknek a mennyiségeknek a célszerű megválasztása attól is függ, hogy az algoritmus konkrét vagy absztrakt gépen (számítási modellen) fut.

Jelöljük $W(n,\pi,\mathcal{A})$-val, illetve $W(n,\pi,p,\mathcal{P})$-vel azoknak a lépéseknek a számát, amelyekkel a π problémát az $\mathcal{A}$ soros, illetve a $\mathcal{P}$ párhuzamos algoritmus – (utóbbi p processzort felhasználva) – n méretű feladatokra legrosszabb esetben megoldja.

Hasonlóképpen jelöljük $B(n,\pi,\mathcal{A})$-val, illetve $B(n,\pi,p,\mathcal{P})$-vel azoknak a lépéseknek a számát, amelyekkel a π problémát az $\mathcal{A}$ soros, illetve a $\mathcal{P}$ párhuzamos algoritmus (utóbbi p processzort felhasználva) – n méretű feladatokra legjobb esetben megoldja.

Jelöljük $N(n,\pi)$-vel, illetve $N(n,\pi,p)$-vel azoknak a lépéseknek a számát, amelyekre az n méretű π feladat megoldásához mindenképpen szüksége van bármely soros, illetve bármely párhuzamos algoritmusnak – utóbbinak akkor, ha legfeljebb p processzort vehet igénybe.

Tegyük fel, hogy minden n-re adott a π feladat n méretű konkrét előfordulásainak $D(\pi,n)$ eloszlásfüggvénye. Ekkor legyen $E(n,\pi,\mathcal{A})$, illetve $E(n,\pi,p,\mathcal{P})$ annak az időnek a várható értéke, amennyi alatt a π problémát n méretű feladatokra megoldja az $\mathcal{A}$ soros, illetve a $\mathcal{P}$ párhuzamos algoritmus – utóbbi p processzort felhasználva.

Az elemzések során gyakori az a feltételezés, hogy az azonos méretű bemenetek előfordulási valószínűsége azonos. Ilyenkor ***átlagos*** lépésszámról beszélünk, amit $A(n,\mathcal{A})$-val jelölünk.

A W, B, N, E és A jellemzők függnek attól a számítási modelltől is, amelynek alapul vételével az algoritmust megvalósítjuk. Az egyszerűség kedvéért feltesszük, az algoritmus egyúttal a számítási modellt is egyértelműen meghatározza.

Amennyiben a szövegkörnyezet alapján egyértelmű, hogy mely problémáról van szó, akkor a jelölésekből π-t elhagyjuk.

Ezek között a jellemzők között érvényesek az

$$\begin{aligned} N(n) &\leq B(n,\mathcal{A}) && (6.1)\\ &\leq E(n,\mathcal{A}) && (6.2)\\ &\leq W(n,\mathcal{A}) && (6.3) \end{aligned}$$

egyenlőtlenségek. Hasonlóképpen a párhuzamos algoritmusok jellemző adataira az

$$\begin{aligned} N(n,p) &\leq B(n,\mathcal{P},p) && (6.4)\\ &\leq E(n,\mathcal{P},p) && (6.5)\\ &\leq W(n,\mathcal{P},p) && (6.6) \end{aligned}$$

egyenlőtlenségek teljesülnek. Az átlagos lépésszámra pedig a

$$\begin{aligned} B(n,\mathcal{A}) &\leq A(n,\mathcal{A}) && (6.7)\\ &\leq W(n,\mathcal{A})\,, && (6.8) \end{aligned}$$

illetve

$$\begin{aligned} B(n,\mathcal{P},p) &\leq A(n,\mathcal{P},p) && (6.9)\\ &\leq W(n,\mathcal{P},p) && (6.10) \end{aligned}$$

áll fenn.

Hangsúlyozzuk, hogy ezek a jelölések nem csak lépésszámra, hanem az algoritmusok más jellemzőire – például memóriaigény, küldött üzenetek száma – is alkalmazhatók.

Ezután relatív jellemzőket, úgynevezett *hatékonysági mértékeket* definiálunk.

A relatív lépésszám azt mutatja, hányszor kisebb a párhuzamos algoritmus lépésszáma a soros algoritmus lépésszámánál.

A $\mathcal{P}$ párhuzamos algoritmusnak az $\mathcal{A}$ soros algoritmusra vonatkozó ***gyorsítása*** (vagy relatív lépésszáma)

$$g(n, \mathcal{A}, \mathcal{P}) = \frac{W(n, \mathcal{A})}{W(n, p, \mathcal{P})} \ . \tag{6.11}$$

Ha a gyorsítás egyenesen arányos a felhasznált processzorok számával, akkor lineáris gyorsításról beszélünk. Ha a $\mathcal{P}$ párhuzamos és az $\mathcal{A}$ soros algoritmusra

$$\frac{W(n, \mathcal{A})}{W(n, p, \mathcal{P})} = \Theta(p) \ , \tag{6.12}$$

akkor $\mathcal{P}$ $\mathcal{A}$-ra vonatkozó gyorsítása ***lineáris***.

Ha a $\mathcal{P}$ párhuzamos és az $\mathcal{A}$ soros algoritmusra

$$\frac{W(n, \mathcal{A})}{W(n, p, \mathcal{P})} = o(p) \ , \tag{6.13}$$

akkor $\mathcal{P}$ $\mathcal{A}$-ra vonatkozó gyorsítása ***szublineáris.***

Ha a $\mathcal{P}$ párhuzamos és az $\mathcal{A}$ soros algoritmusra

$$\frac{W(n, \mathcal{A})}{W(n, p, \mathcal{P})} = \omega(p) \ , \tag{6.14}$$

akkor $\mathcal{P}$ $\mathcal{A}$-ra vonatkozó gyorsítása ***szuperlineáris.***

A párhuzamos algoritmusok esetében fontos jellemző az $m(n, p, \mathcal{P})$ ***munka***, amit a lépésszám és a processzorszám szorzatával definiálunk. Akkor is ez a szokásos definíció, ha a processzorok egy része csak a lépésszám egy részében dolgozik:

$$m(n, p, \mathcal{P}) = pW(n, p, \mathcal{P}) \ . \tag{6.15}$$

Érdemes hangsúlyozni, hogy – különösen az aszinkron algoritmusok esetében – a ténylegesen elvégzett lépések száma lényegesen kevesebb lehet, mint amit a fenti (6.15) képlet szerint kapunk.

Egy $\mathcal{P}$ párhuzamos algoritmusnak az ugyanazon problémát megoldó soros algoritmusra vonatkozó $h(n, p, \mathcal{P}, \mathcal{A})$ ***hatékonysága*** a két algoritmus munkájának hányadosa:

$$h(n, p, \mathcal{P}, \mathcal{A}) = \frac{W(n, A)}{pW(n, p, \mathcal{P})} \ . \tag{6.16}$$

Abban a természetes esetben, amikor a párhuzamos algoritmus munkája legalább akkora, mint a soros algoritmusé, a hatékonyság nulla és egy közötti érték – és a viszonylag nagy érték a kedvező.

Központi fogalom a párhuzamos algoritmusok elemzésével kapcsolatban a munkahatékonyság. Ha a $\mathcal{P}$ párhuzamos és $\mathcal{A}$ soros algoritmusra

$$pW(n, p, \mathcal{P}) = O(W(n, \mathcal{A})) \ , \tag{6.17}$$

akkor $\mathcal{P}$ $\mathcal{A}$-ra nézve ***munkahatékony.***

Ez a meghatározás egyenértékű a

$$\frac{pW(n,p,\mathcal{P})}{W(n,\mathcal{A})} = O(1) \tag{6.18}$$

egyenlőség előírásával. Eszerint egy párhuzamos algoritmus csak akkor munkahatékony, ha összes munkája nagyságrendileg nem nagyobb, mint a soros algoritmus munkája.

Egy párhuzamos algoritmus akkor és csak akkor munkahatékony, ha a relatív lépésszáma lineáris. Egy munkahatékony párhuzamos algoritmus hatékonysága $\Theta(1)$.

Ha egy algoritmus egy feladat megoldásához adott erőforrásból csak $O(N(n))$ mennyiséget használ fel, akkor az algoritmust az adott erőforrásra, számítási modellre (és processzorszámra) nézve ***aszimptotikusan optimálisnak*** nevezzük.

Ha egy $\mathcal{A}$ ($\mathcal{P}$) algoritmus egy feladat megoldásához adott erőforrásból a bemenet minden lehetséges $n \geq 1$ mérete esetében csak az okvetlenül szükséges $N(n,\mathcal{A})$ – illetve $N(n,p,\mathcal{A})$ – mennyiséget használja fel, azaz

$$W(n,\mathcal{A}) = N(n,\mathcal{A}) \ , \tag{6.19}$$

illetve

$$W(n,p,\mathcal{P}) = N(n,p,\mathcal{P}) \ , \tag{6.20}$$

akkor az algoritmust az adott erőforrásra (és az adott számítási modellre) nézve ***abszolút optimálisnak*** nevezzük és azt mondjuk, hogy $W(n,p,\mathcal{P}) = N(n,p,\mathcal{P})$ a vizsgált feladat ***pontos bonyolultsága.***

Két algoritmus összehasonlításakor a

$$W(n,\mathcal{A}) = \Theta(W(n,\mathcal{B})) \tag{6.21}$$

esetben azt mondjuk, hogy a $\mathcal{A}$ és $\mathcal{B}$ algoritmus lépésszámának növekedési sebessége aszimptotikusan ***azonos nagyságrendű.***

Amikor két algoritmus lépésszámát (például a legrosszabb esetben) összehasonlítjuk, akkor gyakran találunk olyan váltási helyeket, melyeknél kisebb méretű bemenetekre az egyik, míg nagyobb méretű bemenetekre a másik algoritmus lépésszáma kedvezőbb. A formális definíció a következő: ha a pozitív egész helyeken értelmezett $f(n)$ és $g(n)$ függvényekre, valamint a $v \geq 0$ pozitív egész számra teljesül, hogy

1. $f(v) = g(v)$;
2. $(f(v-1) - g(v-1))(f(v+1) - g(v+1)) < 0$,

akkor a v számot az $f(n)$ és $g(n)$ függvények ***váltási helyének*** nevezzük.

Például két mátrix szorzatának a definíció alapján történő és a Strassen-algoritmussal történő kiszámítását összehasonlítva (lásd például Cormen, Leiserson, Rivest és Stein többször idézett új könyvét) azt kapjuk, hogy kis mátrixok esetén a hagyományos módszer, míg nagy mátrixok esetén a Strassen-algoritmus az előnyösebb – egy váltási hely van, melynek értéke körülbelül 20.

Gyakorlatok

6.2-1. Határozzuk meg azokat a részfeladatokat, amelyek a szokásos mátrixszorzás során párhuzamosíthatók. Lehetőleg minél több részfeladatot adjunk meg. Ezután ajánljunk

különböző lehetőségeket a részfeladatok (kisebb számú) szálakra való felbontására, majd hasonlítsuk össze ezeket a leképezéseket a közös memóriájú architektúrára való leképezés hatékonyságával.

6.2-2. Tekintsünk egy párhuzamos programot, melynek bemenete egy n egész szám és kimenete a $2 \leq p \leq n$ feltételnek eleget tevő prím számok sorozata. A T_i taszk meghatározza, hogy i prímszám-e. Ezt úgy dönti el, hogy az i számot rendre elosztja a potenciális osztókkal, azaz a $2, \ldots, \sqrt{i}$ számokkal. A programot rögzített számú folyamattal és szállal valósítsuk meg. Ajánljunk különböző lehetőségeket ennek megoldására, elemezzük előnyeiket és hátrányaikat. Vegyük figyelembe mind a statikus, mind pedig a dinamikus terhelés-kiegyensúlyozó sémákat.

6.2-3. Határozzuk meg a következő stencil kód adatösszefüggéseit:

```
for (t=0; t<tmax; t++)
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      a[i][j] += a[i-1][j] + a[i][j-1];
```

Alakítsuk át úgy a kódot, hogy párhuzamosítható legyen.

6.3. Párhuzamos programozás

Részben a különböző architektúrák alkalmazása, részben pedig a terület újdonsága miatt a párhuzamos programozás számos modellje ismert. A legnépszerűbb modellek ma az üzenetküldés, amelyet az MPI (**M**essage **P**assing **I**nterface) szabvány és a strukturált közös memóriájú programozás, amelyet az OpenMP szabvány egységesít. Ezeket a programozási modelleket az 6.3.1., illetve a 6.3.2. pontokban tárgyaljuk. További fontos modelleket, mint a szálprogramozás, adatpárhuzamosság és automatikus párhuzamosítás a 6.3.3. pontban vázolunk.

6.3.1. MPI programozás

Amint a neve mutatja, az MPI az üzenetküldő programozási modellen alapul. Ebben a modellben több program fut párhuzamosan és kommunikál egymással üzeneteket küldve és kapva. A folyamatok nem férnek hozzá a közös memóriához, minden kommunikáció közvetlen üzenetcserével történik. A kommunikáció pontosan két folyamatot tételez fel: az egyik a *küld* műveletet, a másik pedig a *fogad* műveletet hajtja végre. Az üzenetküldés mellett az MPI csoportos műveleteket és kommunikációs eszközöket is tartalmaz.

Az üzenetküldés asszimetrikus abban az értelemben, hogy a küldőnek fel kell fednie kilétét, míg a fogadó vagy felfedi kilétét vagy kinyilvánítja készségét, hogy bármilyen forrásból adatokat kapjon.

Mivel mind a küldőnek, mind pedig a fogadónak aktívan részt kell vennie a kommunikációban, a programozónak előre meg kell terveznie, mikor fognak egy adott folyamatpár elemei egymással kommunikálni. Üzenet cseréjére több célból is sor kerülhet:

- a programozó által előre megtervezett adatok olyan tulajdonságainak cseréje, mint az adatok mérete vagy típusa;
- olyan vezérlő információ cseréje, amely a későbbi üzenetcserére vonatkozik;
- szinkronizáció, melyet azzal érünk el, hogy a beérkező üzenet tájékoztatja a fogadót a küldő állapotáról. Amint később látni fogjuk, ez kiegészíthető azzal, hogy a küldő kap

információt a fogadó állapotáról. Megjegyezzük, hogy a szinkronizáció a kommunikáció speciális esete.

Az MPI szabványt 1994-ben vezette be az MPI fórum, amely hardver- és szoftverkereskedők, kutatóintézetek és egyetemek egy csoportja. A lényegesen kiterjesztett MPI-2 változat 1997-ben jelent meg. Az MPI-2-nek ugyanazok az alaptulajdonságai, de kiegészítő függvényosztályokat vezet be.

Az MPI számos olyan könyvtári függvényt alkalmaz, amelyek összekötik a C, C++ és FORTRAN nyelveket. Az MPI legtöbb funkciója a folyamatok közötti kommunikációval kapcsolatos és nyitva hagy olyan folyamatkezelési problémákat, mint a folyamatok elindítása és megállítása – bár ez alól az MPI-2-ben számos kivétel van. Ezeket a lehetőségeket a szabványon kívül kell megoldani, és ezért a megoldások nem hordozhatók. Ezért és további okokból az MPI programok tipikusan rögzített folyamathalmazt alkalmaznak – és ezek a folyamatok egyszerre indulnak el akkor, amikor a program végrehajtása megkezdődik. A programok kódolhatók SPMD vagy MPMD stílusban. Párhuzamos programokat írhatunk úgy, hogy mindössze hat alapfüggvényt alkalmazunk:

- Az MPI_Init függvényt minden más MPI függvény előtt meg kell hívni.
- Az MPI_Finalize függvényt az utolsó MPI függvény után kell meghívni.
- Az MPI_Comm_size függvény megadja a programban lévő folyamatok számát.
- Az MPI_Comm_rank függvény megadja a hívó folyamat számát, a számozást nullától kezdve.
- Az MPI_Send függvény üzenetet küld. Ennek a függvénynek a következő paraméterei vannak:
  - az üzenet címe, mérete és adattípusa, valamint a fogadó sorszáma;
  - üzenet címkéje, ami egy olyan szám, amely az üzenetet ahhoz hasonló módon jellemzi, ahogyan az elektronikus levelet a tárgya;
  - kommunikátor, amely folyamatok egy csoportja, ahogy azt a továbbiakban elmondjuk.
- Az MPI_Recv függvény egy üzenetet kap. A függvénynek ugyanazok a paraméterei, mint az MPI_Send függvénynek, azzal az eltéréssel, hogy az üzenet méretére csak egy felső korlátot kell megadni, továbbá a küldőre egy szabad paraméter adható meg, és egy státusznak nevezett paraméter adja meg a fogadott üzenet küldőjét, méretét és címkét.

A 6.6. ábra egy MPI programot mutat be.

Bár a fenti funkciók elegendők egy egyszerű program megírásához, számos további funkció segít az MPI programok hatékonyságának és/vagy szerkezetének javításában. Az MPI-1 többek között a következő függvényosztályokat támogatja:

- *Alternatív funkciók a páronkénti kommunikációhoz.* Az MPI_Send függvény, amelyet szabványos küldő függvénynek is neveznek, vagy a fogadó által elküldött üzenetet, vagy a rendszer által a bufferba tett üzenet ad vissza. A két lehetőség közül az MPI választ. Az MPI_Send függvény különböző változatai a következő lehetőségek valamelyikét választják: szinkronizált módban a függvény csak akkor küld üzenetet, ha először a fogadó kapott üzenetet – ezzel szinkronizál mindkét irányban. A buffer módban a rendszernek akkor kell az üzenetet tárolnia, ha a fogadó még nem alkalmazta az MPI_Recv függvényt.

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"

int main (int argc, char **argv) {
  char msg[20];
  int me, total, tag=99;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &me);
  MPI_Comm_size(MPI_COMM_WORLD, &total);

  if (me==0) {
    strcpy(msg, "Hello");
    MPI_Send(msg, strlen(msg)+1, MPI_CHAR, 1, tag,
             MPI_COMM_WORLD);
  }
  else if (me==1) {
    MPI_Recv(msg, 20, MPI_CHAR, 0, tag, MPI_COMM_WORLD,
             &status);
    printf("Received: %s \n", msg);
  }
  MPI_Finalize();
  return 0;
}
```

6.6. ábra. Egy egyszerű MPI program.

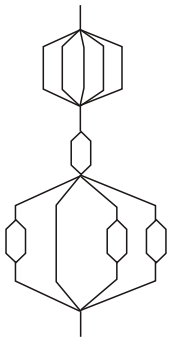
Mind a küldőnél, mind pedig a fogadónál a függvényeknek a szabványos, szinkronizált és buffer módban is van blokkolt és blokkolatlan változata. A blokkolt változatot korábban leírtuk. A blokkolatlan változatok közvetlenül hívás után válaszolnak, és hagyják, hogy a küldő/fogadó folytassa a program végrehajtását mindaddig, míg a rendszer a háttérben befejezi a kommunikációt. A blokkolatlan kommunikációnak az MPI_Wait vagy MPI_Test függvényekkel kell befejeződnie azért, hogy biztosak lehessünk abban, hogy a kommunikáció befejeződött és a buffer újra felhasználható. A befejezési függvények lehetővé teszik a többszörös igényekre való várakozást.

Az MPI programok holtpontra juthatnak, ha például a P_0 folyamat először az MPI_Send függvényt alkalmazza a P_1 folyamattal kapcsolatban, majd ugyanezt teszi a P_1 folyamat P_0-lal kapcsolatban, ezután pedig ugyanez történik fordított szereposztással. Az MPI támogatja a kombinált küld/fogad függvényt.

Számos programban a folyamatpárok ismételten cserélnek adatokat ugyanazon buffer felhasználásával. A kommunikációs többletterhelés csökkentése érdekében címeket tartalmazó címkéket alkalmazhatunk – ezt ***állandó kommunikációnak*** nevezzük. Végül az MPI_Probe és MPI_Iprobe függvények lehetővé teszik, hogy az üzenet méretét és más jellemzőit az üzenet fogadása előtt megvizsgálhassuk.

- *Adattípusokat kezelő függvények.* Az üzenetküldés egyszerű formájában azonos típusú (például `float`) adatokat továbbítunk. Az MPI modell megengedi, hogy egy üzenetben különböző típusú adatokat továbbítsunk, és azt is, hogy nem folyamatos bufferekből küldjünk adatokat, például egy tömb minden második elemét küldjük el. Ehhez az MPI két függvényosztályt definiál: a felhasználó által definiált adattípusok adatpozíciókat és típusokat írnak le, míg a pakolási függvények abban segítenek, hogy egy bufferbe több adat kerüljön. Az MPI modell azzal is támogatja a heterogenitást, hogy szükség esetén automatikusan konvertálja az adatokat.
- *Kollektív kommunikációs függvények.* Ezek támogatják az olyan kommunikációs mintákat, mint az üzenetszórás. Bár minden kommunikációs minta megvalósítható küld/fogad függvények sorozatával, a csoportos függvények előnyösebbek, mivel javítják a program tömörségét és érthetőségét, és gyakran van optimizált megvalósításuk. Továbbá a megvalósítások kihasználhatják az architektúra sajátosságait, és így a másik gépre átvitt program az új gépen is hatékonyan futhat, felhasználva az adott gépre optimizált megvalósítást.
- *Csoport- és kommunikátor-vezérlő függvények.* Amint azt már említettük, a küld és fogad függvények tartalmaznak egy kommunikátor argumentumot, amely leírja a folyamatok egy csoportját. Technikai értelemben a kommunikátor egy osztott adatszerkezet, amely minden folyamatnak megmondja, hogyan érheti el a csoportjában lévő többi folyamatot, és ***attribútumoknak*** nevezett kiegészítő információt is tartalmaz. Ugyanaz a csoport különböző kommunikátorokkal is leírható. Az üzenetcsere csak akkor megy végbe, ha `MPI_Send` és `MPI_Recv` függvények kommunikátor argumentumai megfelelnek egymásnak. Ezért a kommunikátorok alkalmazása felbontja a programok üzeneteit diszjunkt halmazokra úgy, hogy azok a halmazok azután nem hatnak egymásra. Ily módon a kommunikátorok segítenek a programok strukturálásában és hozzájárulnak a programok helyességéhez. Az MPI segítségével megvalósított könyvtáraknál a kommunikátorok lehetővé teszik, hogy a könyvtári forgalmat és az alkalmazói programok forgalmát szétválasszuk. A csoportkommunikátorok a csoportos kommunikációhoz szükségesek. Az adatszerkezetben lévő attribútumok tartalmazhatnak olyan alkalmazás-specifikus információt, mint a hibakezelő. Az eddig leírt – csoporton belüli – kommunikátorok mellett az MPI támogatja a külső kommunikátorokat is.

  Az MPI-2 négy további függvénycsoportot alkalmaz:
- *Dinamikus folyamatkezelő függvények.* Ezekkel a függvényekkel a program futása során új MPI folyamatok indíthatók el. Továbbá, az egymástól függetlenül elindított MPI programok (mindegyik több folyamatból áll) ügyfél/kiszolgáló mechanizmus segítségével kapcsolatba kerülhetnek egymással.
- *Egyoldalú kommunikációs függvények.* Az egyoldalú kommunikáció a közös memória segítségével történő kommunikáció egyik típusa, melyben a folyamatok egy csoportja saját címterének egy részét közös memóriaként használja. A kommunikáció a közös memóriába való írással és az onnan való olvasással történik, Az egyoldalú kommunikáció abban különbözik a többi közös memóriás kommunikációtól – például az OpenMP-től – hogy a memóriához való hozzáféréshez explicit függvényekre van szükség.
- *Párhuzamos B/K függvények.* Számos függvény lehetővé teszi, hogy a többszörös folyamatok csoportosan olvashassanak/írhassanak ugyanabból/ugyanabba a fájlba.



6.7. ábra. Egy OpenMP program szerkezete.

- *Kollektív kommunikációs függvények interkommunikátorokhoz.* Ezek a függvények általánosítják a csoportos kommunikáció fogalmát interkommunikátorokra. Például egy csoport egy folyamata üzenetet szórhat egy másik csoport minden folyamatához.

6.3.2. OpenMP programozás

Az OpenMP neve onnan származik, hogy nyitott szabvány a multiprogramozáshoz, azaz a közös memóriájú architektúrákhoz. A közös memória miatt ebben a pontban folyamatok helyett szálakról lesz szó.

A közös memóriával történő kommunikáció gyökeresen különbözik az üzenetküldéstől. Míg az üzenetküldés közvetlenül feltételez két folyamatot, a közös memória elválasztja a folyamatokat egy közbülső elem beiktatásával. Küld/fogad helyett olvas/ír kifejezéseket használunk, azaz egy szál ír a memóriába, egy másik szál pedig később olvas a memóriából. A szálaknak nem kell egymásról tudniuk, és egy beírt értéket több szál is olvashat. Az olvasás és az írás között tetszőleges hosszúságú idő telhet el. Ebben az esetben a szinkronizációt explicit módon kell szervezni: az olvasónak tudnia kell, mikor fejeződött be az írás, és el kell kerülni azt, hogy ugyanazokat az adatokat egyidejűleg több szál is átalakíthassa.

Az OpenMP modell – az 6.3.1. pontban ismertetett MPI és a 6.3.3. alfejezetben ismertetett egyéb modellektől eltérően – nem tartalmaz egyoldalú kommunikációt. Az OpenMP abban különbözik a többi modelltől, hogy elágazásegyesítő szerkezete van, ahogyan azt a 6.7. ábra mutatja. A program végrehajtása egyetlen – ***mesterszálnak*** nevezett szál végrehajtásával kezdődik, majd a párhuzamos szakaszban több szál jön létre – ezek egyike a mesterszál. A párhuzamos szakaszok egymásba ágyazhatók, de a párhuzamos szálaknak egyszerre kell befejeződniük. Amint az ábra mutatja, a párhuzamos szakaszok sorozatot is alkothatnak úgy, hogy az egyes szakaszokban lévő szálak száma különböző.

Az OpenMP modell a könyvtári függvények helyett fordítóprogram parancsokat használ. Ezek a parancsok olyan útmutatásokat tartalmaznak, melyeket a fordítóprogram vagy figyelembe vesz, vagy nem. Például egy soros fordítóprogram figyelmen kívül hagyja ezeket a parancsokat. Az OpenMP modell támogatja a fokozatos párhuzamosítást, melyben egy soros programból indulunk ki, a hatékonyság szempontjából kritikus részeken fordítóprogram parancsokat alkalmazunk, majd később szükség esetén további parancsokat alkalmazunk.

Az OpenMP-t 1998-ban vezették be, 2.0 változata 2002-ben jelent meg. A fordítóprogramot vezérlő parancsok mellett az OpenMP néhány könyvtári funkciót és környezeti változót is alkalmaz. A szabvány C, C++ és FORTRAN nyelvekhez is rendelkezésre áll.

```
#include <omp.h>
#define N 100
double a[N][N], b[N], c[N];
int main() {
  int i, j;
  double h;
  /* a kezd  rt kek be ll t s t elhagytuk */
  omp_set_num_threads(4);
  #pragma omp parallel for shared(a,b,c) private(j)
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      c[i] += a[i][j] * b[j];
  /* az eredm nyek kivitel t elhagytuk */
}
```

6.8. ábra. Mátrix-vektor szorzás az OpenMP-ben párhuzamos ciklus alkalmazásával.

OpenMP-ben könnyebb programozni, mint az MPI szabvány szerint, mivel a fordítóprogram részekre bontja az elvégzendő munkát. Aki OpenMP-ben programoz, meghatározza a szálak számát, azután pedig a következő módok egyikén írja le a munka megosztását:

- *Explicit módon.* Egy szál az omp_get_thread_num paranccsal saját számot igényelhet. Egy egy feltételes utasítás értékeli ezt a számot és explicit módon taszkokat rendel a szálhoz – az MPI programok SPMD stílusához hasonló módon.
- *Párhuzamos ciklus.* A fordítóprogramot vezérlő #pragma omp parallel for parancs azt jelzi, hogy a következő for parancs végrehajtható párhuzamosan úgy, hogy minden szál több iterációt (taszkot) hajt végre. Erre egy példát mutat a 6.8. ábra. A programozó a munkát a schedule(static) vagy schedule(dynamic) parancsokkal befolyásolhatja. A statikus ütemezés azt jelenti, hogy minden szál az egymást követő iterációk hasonló méretű blokkját kapja meg. A dinamikus ütemezés azt jelenti, hogy először minden szálhoz egy iterációt rendelünk, azután pedig valahányszor egy szál befejezi a kapott iterációt, kap egy újabbat, ahogyan azt korábban az MPI programozásnál a mester/szolga paradigmára leírtuk. Itt azonban – a mester/szolga megoldástól eltérően – a fordítóprogram dönti el, hogy melyik szál melyik taszkot hajtsa végre és a fordítóprogram határozza meg a szükséges kommunikációt is.
- *Taszkpárhuzamos szakaszok.* A #pragma omp parallel sections lehetővé teszi azon taszkok listájának leírását, amelyeket a rendelkezésre álló szálakhoz rendeltünk.

A szálak a közös memórián keresztül kommunikálnak, azaz a közös változókba írnak vagy azokat olvassák. A változóknak csak egy része közös, míg mások csak egy speciális szálhoz tartoznak. Azokat a szabályokat, amelyek meghatározzák, hogy egy változó közös vagy nem, a programozó felülírhatja.

Több OpenMP parancs azzal a szinkronizációval kapcsolatos, amely a kölcsönös kizáráshoz szükséges és lehetővé teszi a közös memória egységes kezelését. Bizonyos szinkronizációs parancsokat a fordítóprogram ad meg. Például a párhuzamos ciklus végén minden szál megvárja a többi szálat, mielőtt új ciklust kezdene.

6.3.3. Más programozási modellek

Bár az MPI és OpenMP a legnépszerűbb modellek, más megközelítéseknek is van gyakorlati jelentősége. Itt a szálprogramozást, a nagy teljesítményű FORTRAN-t és az automatikus párhuzamosítást mutatjuk be.

Az OpenMP-hez hasonlóan a ***szálprogramozásban*** – például a *phtreads* könyvtárban vagy a Java nyelv szálainál – közös memóriát alkalmaznak. A szálak alacsonyabb absztrakciós szinten működnek, mint az OpenMP, amelyben a programozó a felelős a szálkezelés és a munkamegosztás minden részletéért. A szálakat egyesével konkrétan létre kell hozni, és hozzájuk kell rendelni az elvégzendő feladatot. A szálprogramozás a taszkpárhuzamosságra koncentrál, míg az OpenMP programozás az adatpárhuzamosságra. A szálprogramok strukturálatlanok lehetnek, azaz bármely szál létrehozhat új szálat vagy megállíthat egy másik szálat. Az OpenMP programokat gyakran alakítják át szálprogramokká.

Az adatpárhuzamosság más programozási stílushoz vezet, amelyet az olyan nyelvek támogatnak, mint például a a nagyteljesítményű FORTRAN (**H**igh **P**erformance **F**ortran). Míg az adatpárhuzamosság kifejezhető az MPI, OpenMP stb. eszközeivel, az adatpárhuzamos nyelvek másra helyezik a fő hangsúlyt. A HPF egyik fontos konstrukciója a párhuzamos ciklus, melynek iterációi egymástól függetlenül – azaz kommunikáció nélkül – végrehajthatók. Az adatpárhuzamos stílus könnyebben érthetővé teszi a programokat, mivel nem kell figyelmet fordítani a konkurens tevékenységekre. Hátránya, hogy nehézségeket okozhat az alkalmazások ilyen struktúrába kényszerítése. A HPF egyetlen osztott címtérrel rendelkezik, és a nyelv jelentős része az adatok mozgatásával foglalkozik. Míg az MPI programozók az adatokat explicit módon elküldik a megfelelő helyre, a HPF programozók az adatok szétosztását ahhoz hasonló absztrakt szinten írják le, mint ahogy az OpenMP programozók megadják a párhuzamos ciklusok ütemezését. A részletet a fordítóprogramra bízzák. Az OpenMP egyik fontos fogalma a tulajdonos-szabály, mely szerint egy értékadó utasítás baloldali változójának tulajdonosa végzi el a műveletet. Ezért az adatok elosztásából adódik a számítások elosztása.

A ciklusok párhuzamosítása – különösen a tudományos számítások területén – jelentős teljesítmény-növekedést eredményez. Ez a párhuzamosítás gyakran elvégezhető automatikusan, párhuzamosító fordítóprogramok segítségével. Ezek a fordítóprogramok ellenőrzik az adatösszefüggéseket. Számos program szerkezete átalakítható a függőség csökkentése érdekében, a külső és a belső ciklusok felcserélével. A párhuzamosító fordítóprogramok fontos programosztályokra felismerik ezeket az átalakítási lehetőségeket.

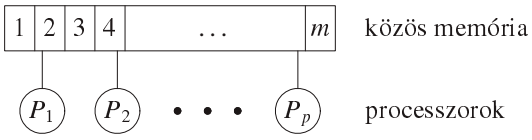
Gyakorlatok

6.3-1. Tervezzünk MPI programot a 6.2-2. gyakorlatban szereplő prímfeladat megoldására. A program kövesse a „mester/szolga” paradigmát. A program az SPMD stílust vagy az MPMD stílust alkalmazza?

6.3-2. Módosítsuk az előző 6.3-1. gyakorlatban tervezett programot úgy, hogy kollektív kommunikációt alkalmazzon.

6.3-3. Hasonlítsuk össze az MPI-t és az OpenMP-t programozhatóság szempontjából, azaz soroljunk fel érveket amellett, hogy az egyik, illetve másik eszközzel könnyebb programot írni.

6.3-4. Tervezzünk egy OpenMP programot, amely megvalósítja a 6.2-3. gyakorlatban leírt stencil kódot.



6.9. ábra. Párhuzamos közvetlen hozzáférésű gép.

6.4. Számítási modellek

PRAM

A legnépszerűbb számítási modell a párhuzamos közvetlen hozzáférésű gép (PRAM), amely a közvetlen hozzáférésű gép (RAM) természetes általánosítása.

A PRAM modell p szinkronizált processzorból $(P_1, P_2, \ldots, P_p)$, az $M[1]$, $M[2]$, ..., $M[m]$ memóriarekeszeket tartalmazó közös memóriából és a processzorok saját memóriájából áll. A 6.9. ábra mutatja a processzorokat és a közös memóriát.

Ennek a modellnek különböző változatai vannak. Ezek a modellek abban különböznek egymástól, hogy a különböző processzorok egyidejűleg hozzáférhetnek-e ugyanahhoz a memóriarekeszhez és hogyan oldódnak meg a konfliktusok. Többek között a következő változatokat különböztetjük meg:

Az ír/olvas műveletek tulajdonságai alapján a következő négy típusuk van:

- EREW (Exclusive-Read Exclusive-Write) PRAM
- ERCW (Exclusive-Read Concurrent-Write) PRAM
- CREW (Concurrent-Read Exclusive-Write) PRAM
- CRCW (Concurrent-Read Concurrent-Write) PRAM

A 6.10(a) ábrán minden memóriarekeszhez legfeljebb egy processzor férhet hozzá (ER), a 6.10(c) ábrán viszont több is. A 6.10(b) ábrán minden memóriarekeszbe legfeljebb egy processzor írhat (EW), a 6.10(d) ábrán viszont több is (CW).

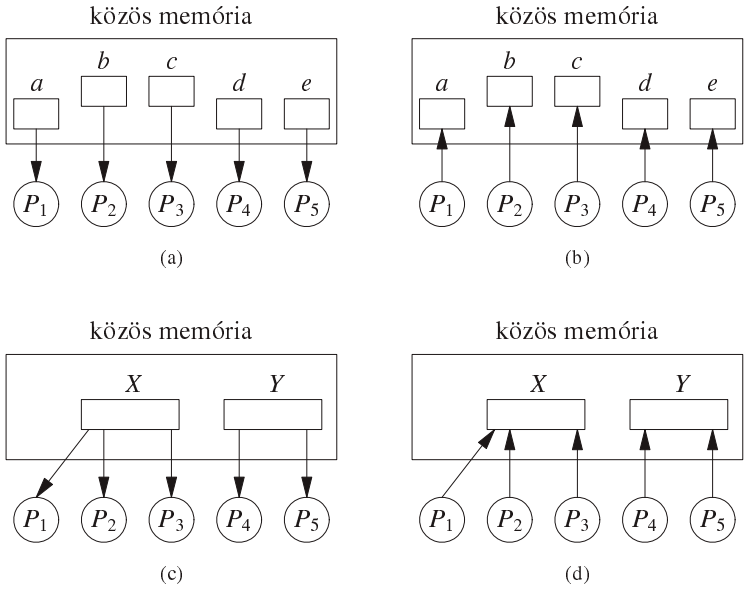
A párhuzamos írás lehetséges típusai: *közös, prioritásos, tetszőleges, kombinált.*

BSP, LogP és QSM

Ebben a pontban a BSP, LogP és QSM modelleket tárgyaljuk.

A szinkronizált párhuzamos modell (**B**ulk-synchronous **P**arallel **M**odel) a számítógépet olyan csomópontok együtteseként írja le, amelyek processzorból és memóriából állnak. A BSP feltételezi, hogy rendelkezésre áll egy útválasztó és egy akadályszinkronizáló képesség. Az útválasztó továbbítja az üzeneteket a csomópontok között, az akadály szinkronizálja a csomópontokat vagy azok egy részét. A BSP a taszkokat úgynevezett *szuperlépésekre* bontja. Egy szuperlépésben minden processzor számításokat végez a saját memóriájában tárolt adatokkal és kommunikációt kezdeményez a többi processzorral. A kommunikáció garantáltan befejeződik a következő szuperlépés megkezdéséig.

g úgy van definiálva, hogy gh az az idő, amelyet egy h-relációs útválasztás igényel a szokásos forgalmi terhelés mellett. A h-reláció egy kommunikációs minta, melyben minden processzor legfeljebb h üzenetet küld és fogad.



6.10. ábra. Párhuzamos közvetlen hozzáférésű gép típusai az írás és olvasás alapján.

Egy szuperlépés költsége $x + gh + l$, ahol x az egy processzor által kezdeményezett kommunikáció maximális száma. Egy program költsége az egyes szuperlépések költségeinek összege.

A BSP tartalmaz egy költség modellt, amely három paramétert tartalmaz: a processzorok számát (p), az akadály szinkronizáció költségét (l) és a rendelkezésre álló sávszélességet (g).

A LogP modell létrejöttét a BSP pontatlanságai és a szuperlépések alkalmazásának igénye motiválta.

Míg a LogP a BSP-nél pontosabb, a QSM egyszerűbb. A BSP-vel szemben a QSM közös-memóriájú modell. A BSP-hez hasonlóan a taszkok a QSM szerint is szuperlépésekre vannak felbontva, és minden processzornak van saját helyi memóriája. A processzor minden szuperlépésben számítási műveleteket végez a helyi memóriában tárolt adatokkal, és olvas/ír műveleteket végez a közös memóriával. A közös memóriához való hozzáférést igénylő műveletek befejeződnek a következő szuperlépés megkezdéséig. A QSM modell megengedi a párhuzamos olvasást és írást. Minden memóriarekeszhez szuperlépésenként legfeljebb k hozzáférés lehetséges. A QSM modellben a költségek értéke $\max(x, gh, k)$, ahol g, h és x a BSP modellnél definiált értékek.

Gyakorlatok

6.4-1. A P és Q párhuzamos algoritmusok megoldják a kiválasztási feladatot. A P algoritmus $n^{0.5}$ processzort igényel és a futási ideje $\Theta(n^{0.5})$. A Q algoritmus n processzort igényel és a futási ideje $\lg n$. Határozzuk meg az algoritmusok munkáját, relatív futási idejét és hatékonyságát. Munkahatékonyak-e ezek az algoritmusok?

6.5. PRAM algoritmusok

Ebben a részben egyszerű feladatokat (mint prefixszámítás, tömbelemek rangsorolása, összefésülés, kiválasztás és rendezés) megoldó párhuzamos algoritmusokat mutatunk. Ezek leírásához kiegészítjük az *Új algoritmusok* második fejezetében leírt pszeudokódot.

P_i **in parallel for** $i \leftarrow 1$ **to** p
⟨1. *utasítás*⟩
⟨2. *utasítás*⟩
.
.
.
⟨*u*. *utasítás*⟩

Egy k dimenziós rács esetében a hasonló utasítás a

$P_{i_1,i_2,\ldots,i_k}$ **in parallel for** $i_1 \leftarrow 1$ **to** $m_1, \ldots, i_k \leftarrow 1$ **to** m_k

sorral kezdődik.

6.5.1. Prefixszámítás

Legyen Σ egy alaphalmaz, melyen definiáltuk a $\oplus$ *bináris asszociatív operátort*. Feltesszük, hogy a művelet egy lépéssel elvégezhető és a Σ halmaz zárt erre a műveletre nézve.

Egy $\oplus$ bináris operátor *asszociatív* a Σ alaphalmazon, ha minden $x, y, z \in \Sigma$ esetében teljesül

$$((x \oplus y) \oplus z) = (x \oplus (y \oplus z)) \; . \tag{6.22}$$

Legyenek az $X = x_1, x_2, \ldots, x_p$ sorozat elemei a Σ alaphalmaz elemei. Ekkor a prefixszámítás bemenő adatai az X sorozat elemei, a *prefixszámítási feladat* pedig az $x_1,\ x_1 \oplus x_2,\ \ldots, x_1 \oplus x_2 \oplus x_3 \oplus \ldots \oplus x_p$ elemek meghatározása. Ezeket a meghatározandó elemeket *prefixeknek* hívjuk.

Érdemes megjegyezni, hogy más területeken inkább az X sorozat $x_1, x_2, \ldots, x_k$ kezdősorozatait hívják prefixeknek.

6.1. példa. *Asszociatív műveletek.* Ha Σ az egészek halmaza, $\oplus$ az összeadás és a bemenő adatok sorozata a 3, -5, 8, 2, 5, 4, akkor a prefixek 3, -2, 6, 8, 13, 17. Ha az ábécé és a bemenő adatok ugyanazok, de a művelet a szorzás, akkor a kimenő adatok (prefixek) $3, -15, -120, -240, -1200, -4800$. Ha a művelet a minimum (ez is asszociatív), akkor a prefixek $3, -5, -5, -5, -5, -5$. Az utolsó prefix megegyezik a bemenő számok legkisebbikével.

A prefixszámítás sorosan megoldható $O(p)$ lépéssel. Bármely $\mathcal{A}$ soros algoritmusnak $N(p, \mathcal{A}) = \Omega(n)$ lépésre szüksége van. Vannak olyan párhuzamos algoritmusok, amelyek különböző párhuzamos modelleken megoldják a munkahatékony prefixszámítást.

Először a CREW-PREFIX CREW PRAM algoritmust mutatjuk be, amely p processzoron $\Theta(\lg p)$ lépéssel számítja ki a prefixeket.

Azután az EREW-PREFIX algoritmus következik, amelynek mennyiségi jellemzői hasonlóak az előző algoritmuséhoz, azonban EREW PRAM számítási modellen is végrehajtható.

Ezekkel az algoritmusokkal a prefixszámítást a soros megoldás $\Theta(p)$ lépésszámánál lényegesen gyorsabban meg tudjuk oldani, de sok az elvégzett munka.

Ezért is érdekes a HATÉKONY-PREFIX CREW PRAM algoritmus, amely $\lceil p/\lg p \rceil$ processzort használ és $O(\lg p)$ lépést tesz. Ennek elvégzett munkája csak $O(p)$, ezért a hatékonysága $\Theta(1)$ és az algoritmus munkahatékony. Ennek az algoritmusnak a gyorsítása $\Theta(n/\lg n)$.

A továbbiakban – a jelölések egyszerűsítése érdekében – a $\lceil p/\lg p \rceil$ típusú kifejezések helyett általában csak $(p/\lg p)$-t írunk.

Az algoritmusok tervezéséhez az oszd-meg-és-uralkodj elvet alkalmazzuk. A bemenő adatok legyenek $X = x_1, x_2, \ldots, x_p$. Az általánosság megszorítása nélkül feltehető, hogy p kettő hatvány.

CREW PRAM algoritmus

Először egy p CREW PRAM processzoros rekurzív algoritmust mutatunk be, melynek lépésszáma $\Theta(\lg p)$.

A bemenő adatok a p processzorszám, az $X[1..p] = x_1, x_2, \ldots, x_p$ tömb, a kimenő adat pedig a prefixeket tartalmazó $Y[1..p] = \langle y_1, y_2, \ldots, y_p \rangle$ tömb.

CREW-PREFIX(p, X)

```
1  if p = 1
2     then y_1 ← x_1
3          return Y
4  if p > 1
5     then P_i in parallel for i ← 1 to p/2
6               az első p/2 processzor rekurzívan számítsa az x_1, x_2, ..., x_{p/2}-höz
                tartozó prefixeket – legyenek ezek y_1, y_2, ..., y_{p/2}
7          P_i in parallel for i ← p/2 to p
                számítsa ki az x_{p/2+1}, x_{p/2+2}, ..., x_p-hez
                tartozó prefixeket – legyenek ezek y_{p/2+1}, y_{p/2+2}, ..., y_p.
8          P_i in parallel for i ← p/2 to p
                olvassák ki a globális memóriából y_{p/2}-t és az
                y_{p/2} ⊕ y_{p/2+1}, y_{p/2} ⊕ y_{p/2+2}, ..., y_{p/2} ⊕ y_p prefixeket számítva
                állítsák elő a végeredmény másik felét.
9          return Y
```

6.2. példa. *8 elem prefixeinek számítása 8 processzoron.* Legyen $n = 8$ és $p = 8$. A prefixszámítás bemenő adatai 12, 3, 6, 8, 11, 4, 5 és 7, az asszociatív művelet az összeadás. Az első szakaszban az első 4 processzor 12, 3, 6, 8 bemenethez a 12, 15, 21, 29 prefixeket számolja ki. A másik 4 processzor pedig a 11, 4, 5, 7 bemenethez számolja ki a 11, 15, 20, 27 prefixeket.

A második szakaszban az első 4 processzor nem dolgozik, a második 4 pedig 29-et ad minden prefixhez és a 40, 44, 49, 56 eredményt kapja.

6.1. tétel. *A* CREW-PREFIX *algoritmus p* CREW PRAM *processzoron $\Theta(\lg p)$ lépésben számítja ki p elem prefixeit.*

Bizonyítás. Az első lépés $W(p/2)$ ideig, a második pedig $O(1)$ ideig tart. Ezért a következő rekurziót kapjuk:

$$W(p) = W\left(\frac{p}{2}\right) + O(1)\ , \tag{6.23}$$

$$W(1) = 1\ . \tag{6.24}$$

Ennek a rekurzív egyenletnek a megoldása $W(p) = O(\lg p)$. ■

Ez az algoritmus nem munkahatékony, mivel $\Theta(p \lg p)$ munkát végez, és ismert olyan soros algoritmus, amely $O(p)$ lépést tesz. Munkahatékony algoritmust kaphatunk például úgy, ha a felhasznált processzorok számát lecsökkentjük $(p/\lg p)$-re, miközben a lépésszám nagyságrendje ugyanaz marad. A processzorszámot úgy csökkentjük, hogy a bemenet méretét csökkentjük $(p/\lg p)$-re, alkalmazzuk az előző algoritmust, majd végül minden prefixet kiszámolunk.

EREW PRAM algoritmus

A következő rekurzív algoritmusban a párhuzamos írás helyett elég a soros írás lehetősége, ezért EREW PRAM modellen is megvalósítható. Bemenő adatai a p processzorszám, az $X[0\,..\,p] = \langle x_0,\ x_1,\ x_2,\ \ldots,\ x_p\rangle$ tömb, kimenő adatai pedig a prefixeket tartalmazó $Y[1\,..\,p] = \langle y_1,\ y_2,\ \ldots,\ y_p\rangle$ tömb.

EREW-PREFIX(p, X)

```
1  P_i in parallel for i ← 1 to p
2        Y[i] ← X[i − 1] ⊕ X[i]
3  k ← 2
4  while k < p
5        do P_i in parallel for i ← 1 to p
6                              do Y[i] ← Y[i − k] ⊕ Y[i]
7        k ← k + k
8  return Y
```

6.2. tétel. *Az* EREW-PREFIX *algoritmus p* EREW PRAM *processzoron* $\Theta(\lg p)$ *lépésben számítja ki p elem prefixeit.*

Bizonyítás. Az 1–3. sorban lévő utasítások $O(1)$ idő alatt végrehajtódnak. A 4–6. sorok annyiszor hajtódnak végre, ahányszor a 7. sorban lévő értékadás, azaz $\Theta(p)$-szer. ■

Munkahatékony algoritmus

Most egy rekurzív munkahatékony eljárás következik, amely $p/\lg p$ CREW PRAM processzort használ. A bemenő adatok: p (a bemenő sorozat hossza) és az $X[1\,..\,p]$ tömb, kimenő adat pedig a kiszámított prefixeket tartalmazó $Y[1\,..\,p]$ tömb.

HATÉKONY-PREFIX(p, X)

1 P_i **in parallel for** $i \leftarrow 1$ **to** $p/\lg p$)
2 **do** rekurzívan számolja a hozzárendelt $\lg p$ darab
$x_{(i-1)\lg p+1}, x_{(i-1)\lg p+2}, \ldots, x_{i\lg p}$ elem prefixeit
Legyen az eredmény $z_{(i-1)\lg p+1}, z_{(i-1)\lg p+2}, \ldots, z_{i\lg p}$.
3 összesen $p/\lg p$ processzor alkalmazza együtt a CREW-PREFIX algoritmust a
$p/\lg p$ darab elem, $z_{\lg p}, z_{2\lg p}, z_{3\lg p}, \ldots, z_p$ prefixeinek számítására
Legyen az eredmény $w_{\lg p}, w_{2\lg p}, w_{3\lg p}, \ldots, w_p$.
4 minden processzor aktualizálja az első lépésben kiszámolt értéket: a P_i processzor
$(i = 2, 3, \ldots, p/\lg p)$ kiszámolja a $w_{(i-1)\lg p} \oplus z_{(i-1)\lg p+1}, w_{(i-1)\lg p} \oplus z_{(i-1)\lg p+2}$,
$\ldots, w_{(i-1)\lg p} \oplus z_{i\lg p}$ prefixeket, majd az első processzor
változtatás nélkül kiadja a $z_1, z_2, \ldots, z_{\lg p}$ prefixeket
5 **return** Y

Az algoritmus lépésszáma logaritmikus. Ennek belátását megkönnyíti a következő két képlet:

$$z_{(i-1)\lg p+k} = \sum_{j=(i-1)\lg p+1}^{i\lg p} x_j \quad (k = 1,\ 2,\ \ldots,\ \lg p) \tag{6.25}$$

és

$$w_{i\lg p} = \sum_{j=1}^{i} z_{j\lg p}\ (i = 1,\ 2,\ \ldots,), \tag{6.26}$$

ahol az összegzés a megfelelő asszociatív művelet segítségével történik.

6.3. tétel (párhuzamos prefixszámítás $\Theta(\lg p)$ lépéssel). *A* HATÉKONY-PREFIX *algoritmus* $p/\lg p$ CREW PRAM *processzoron* $\Theta(\lg p)$ *lépéssel számítja ki* p *elem prefixeit.*

Bizonyítás. Az algoritmus első szakasza $O(\lg p)$ ideig, a második szakasza a 6.1. tétel szerint $\Theta(\lg(p/\lg p)) = \Theta(\lg p)$ lépésig tart. Végül a harmadik szakasz ugyancsak $O(\lg p)$ lépést igényel. ■

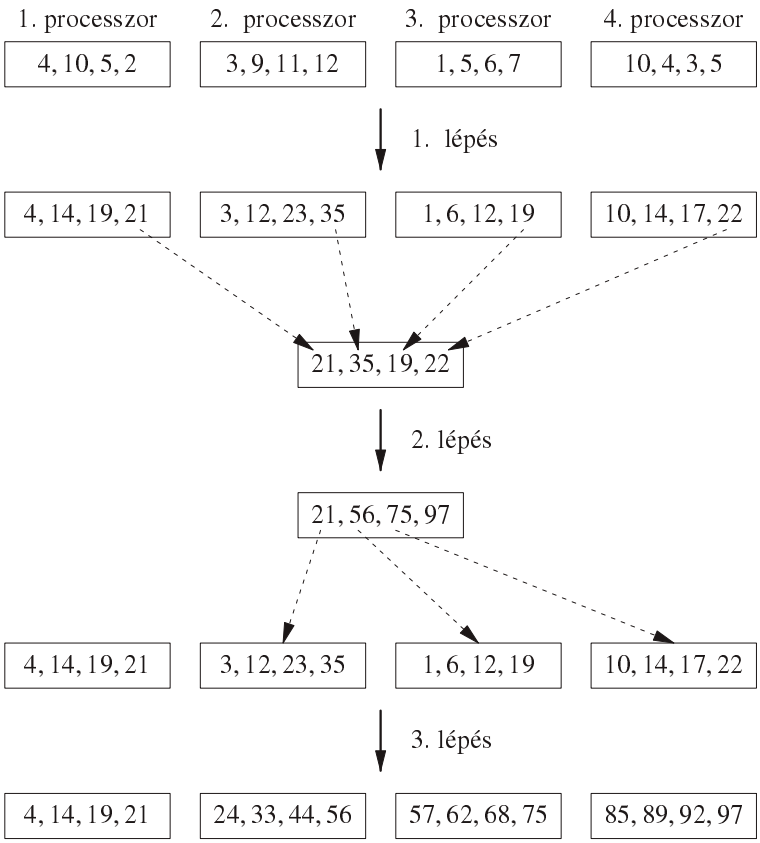
A tételből következik, hogy a HATÉKONY-PREFIX algoritmus munkahatékony.

6.3. példa. *16 elem összeadása.* Legyen 16 elemünk: 4, 10, 5, 2, 3, 9, 11, 12, 1, 5, 6, 7, 10, 4, 3, 5. Az asszociatív művelet az összeadás. Ekkor 16 elem $\lg 16 = 4$ processzort igényel. A processzorok először 4-4 elem prefixeit számolják – sorosan. A második lépésben a helyi összegekből a 4 processzor együtt globális összegeket számol, majd azokkal a harmadik lépésben a processzorok ismét külön sorosan frissítik a helyi eredményeket. A számítás menetét mutatja a 6.11. ábra.

6.5.2. Tömb elemeinek rangsorolása

A *tömbrangsorolási probléma* bemenő adata egy p elemű tömbben ábrázolt lista: minden elem tartalmazza jobboldali szomszédjának az indexét (és esetleges további adatokat). A feladat az elemek *rangjának* (jobboldali szomszédai számának) meghatározása.

Mivel az adatokra nincs szükség a megoldáshoz, feltesszük, hogy az elemek csak a



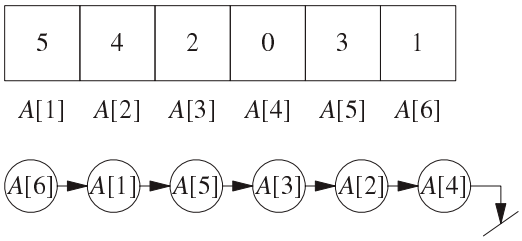
6.11. ábra. 16 elem prefixeinek számítása a HATÉKONY-PREFIX algoritmussal.

szomszéd indexét tartalmazzák. A jobbszélső elem index mezője nulla. Az indexet a továbbiakban mutatónak hívjuk.

6.4. példa. *Tömbrangsorolás bemenő adatai.* Legyen $A[1..6]$ a 6.12. ábra felső sorában bemutatott tömb. Ekkor az $A[1]$ elem jobboldali szomszédja $A[5]$, $A[2]$ jobboldali szomszédja $A[4]$. $A[4]$ az utolsó elem, ezért rangja 0. $A[2]$ rangja 1, mivel csak $A[4]$ van tőle jobbra. $A[5]$ rangja 3, mivel az $A[3]$, $A[2]$ és $A[4]$ elemek vannak tőle jobbra. Az elemek sorrendjét (balról jobbra haladva) mutatja az ábra alsó része.

A tömbrangsorolás sorosan elvégezhető lineáris lépésszámmal. Először meghatározzuk a *tömb fejét* – az egyetlen olyan i értéket ($1 \leq i \leq p$), melyre $A[j] \neq i$ teljesül minden $1 \leq j \leq n$ értékre. Legyen $A[i]$ a tömb feje. A fejtől kiindulva pásztázzuk a tömböt és az elemekhez rendre hozzárendeljük a $p, p-1, \ldots, 1, 0$ rangokat.

Ebben a részben a DET-RANGSOROL algoritmust mutatjuk be, amely egy p processzoros EREW PRAM algoritmus $\Theta(O(\lg p))$ futási idővel. Ennek gyorsítása $\Theta(p/\lg n)$, hatékonysága $\Theta(p)/\Theta(p \lg p) = 1/\lg p$), azaz nem munkahatékony.



6.12. ábra. Tömbrangsorolási probléma bemenő adatai és a megfelelő tömb.

szomsz						*rang*						
5	4	2	0	3	1	1	1	1	0	1	1	(kezdeti állapot)
3	0	4	0	2	5	2	1	2	0	2	2	$q = 1$
4	0	0	0	0	2	4	1	2	0	3	4	$q = 2$
0	0	0	0	0	0	4	1	2	0	3	5	$q = 3$

6.13. ábra. A DET-RENDEZ algoritmus működése a 6.4. példa adataival.

Determinisztikus tömbrangsorolás

Ebben az algoritmusban az egyik alapvető ötlet a *mutatóugrás*. DET-RANGSOROL szerint először mindegyik elem a jobboldali szomszédjának indexét tartalmazza, és ennek megfelelően a rangja – a jobboldali szomszédjához viszonyítva – 1 (kivétel a lista utolsó eleme, melynek rangja 0. Ezt a kezdeti állapotot mutatja a 6.13. ábra első sora.

Ezután módosítjuk a csúcsokat úgy, hogy mindegyik a jobboldali szomszédjának a jobboldali szomszédjára mutasson (ha nincs, akkor a lista végére). Ezt tükrözi a 6.13. ábra második sora.

Ha p processzorunk van, akkor ez $O(1)$ lépéssel elvégezhető.

Most minden csúcs (kivéve az utolsót) olyan csúcsra mutat, amelyik eredetileg 2 távolságra volt. A mutatóugrás következő lépésében a csúcsok olyan csúcsra mutatnak, amelyek eredetileg 4 távolságra voltak tőlük (ha ilyen csúcs nincs, akkor a lista végére) – amint azt az ábra harmadik sora mutatja.

A következő lépésben a csúcsok (pontosabban a mutató részük) a 8 távolságú szomszédra mutatnak (ha van ilyen – ha nincs, akkor a lista végére), a 6.13. ábra utolsó sora szerint.

Minden csúcs minden lépésben információt gyűjt arról, hány csúcs van közte és azon csúcs között, amelyre most mutat. Ehhez kezdetben legyen a csúcsok rang mezőjében 1 – kivéve a jobboldali csúcsot, melyre ez az érték legyen 0. Legyen $R[i]$ és $N[i]$ az i csúcs rang, illetve szomszéd mezője. A mutatóugrás során $R[i]$-t általában $R[i]+R[N[i]]$-re módosítjuk –

kivéve azokat a csúcsokat, melyekre $N[i] = 0$. Ezután $N[i]$-t úgy módosítjuk, hogy $N[N[i]]$-re mutasson.

Az algoritmus bemenő adatai a rangsorolandó elemek p száma, az elemek jobboldali szomszédainak indexeit tartalmazó *szomsz*$[1\,..\,p]$ tömb, kimenő adatai pedig a meghatározott rangokat tartalmazó *rang*$[1\,..\,p]$ tömb.

DET-RANGSOROL pszeudokódja a következő.

DET-RANGSOROL(p, N, R)

```
1  P_i in parallel for i ← 1 to n
2                    if N[i] = 0
3                       then R[i] ← 0
4                       else  R[i] ← 1
5  for i ← 1 to ⌈lg n⌉
6      P_i in parallel for  1 ≤ i ≤ n
7                           if N[i] ≠ 0
8                              then R[i] ← R[i] + R[N[i]]
9                                   N[i] ← N[N[i]]
10 return rang
```

A 6.13. ábra mutatja, hogyan működik DET-RANGSOROL a 2.6. példa adataival.

Kezdetben minden csúcs rangja 1, kivéve a 4. csúcsot. Amikor $q = 1$, akkor például az 1. csúcs R mezőjét kettőre változtatjuk, mert jobboldali szomszédjának (ez az 5. csúcs) rangja 1. Az 1. csúcs N mezőjét az 5. csúcs szomszédjának indexére, azaz 3-ra változtatjuk.

6.4. tétel. *A* DET-RANGSOROL *algoritmus egy p processzoros* EREW PRAM *modellen* $\Theta(\lg p)$ *lépésben határozza meg egy p elemű tömb elemeinek rangját.*

Mivel a A DET-RANGSOROL algoritmus $\Omega(p \lg p)$ munkát végez, ezért nem munkahatékony.

A listarangsorolási probléma megfelel a lista prefix összege számításának, ahol minden csúcs súlya 1, kivéve a jobboldalit, melynek súlya 0. A DET-RENDEZ algoritmus könnyen módosítható úgy, hogy kiszámítsa egy lista prefixeit – a processzorszámra és a lépésszámra vonatkozó hasonló korlátokkal.

6.5.3. Összefésülés

Adott 2 csökkenőleg (vagy növekvőleg) rendezett sorozat, melyek együtt p elemet tartalmaznak. A feladat ennek a sorozatnak egy csökkenő (vagy növekvő) sorozattá való rendezése.

Ez a feladat egy soros processzoron $O(p)$ lépéssel megoldható. Mivel legrosszabb esetben minden elemet meg kell vizsgálni és a helyére kell tenni, ezért a feladat megoldásának lépésszámigénye $\Omega(p)$.

Összefésülés logaritmikus időben

Legyen $X_1 = \langle k_1, k_2, \ldots, k_m \rangle$ és $X_2 = \langle k_{m+1}, k_{m+2}, \ldots, k_{2m} \rangle$ a két bemenő sorozat. Az egyszerűség kedvéért legyen m kettő hatvány és a kulcsok különbözzenek.

Az összefésüléshez elég az összes kulcs rangjának kiszámítása. Ha a rangokat ismerjük, akkor $p = 2m$ processzoron egy lépésben beírhatjuk az i rangú kulcsot az i-edik memóriarekeszbe.

6.5. tétel. *A* LOG-ÖSSZEFÉSÜL *algoritmus két m hosszúságú kulcssorozatot $O(\lg m)$ lépéssel összefésül $2m$* CREW PRAM *processzoron.*

Bizonyítás. A k kulcs rangja legyen r_k^1 (r_k^2) X_1-ben (X_2-ben). Ha $k = k_j \in X_1$, akkor legyen $r_k^1 = j$. Ha egy külön π processzort rendelünk k-hoz, akkor az bináris kiválasztással $O(\lg m)$ lépéssel meghatározza azon X_2-beli elemek q számát, amelyek kisebbek, mint k. Ha q ismert, akkor π kiszámíthatja k ($X_1 \cup X_2$)-beli rangját: ez $j + q$ lesz. Ha k X_2-höz tartozik, hasonlóképpen járhatunk el.

Összegezve: ha elemenként egy, azaz összesen $2m$ processzorunk van, akkor két m hosszúságú rendezett sorozat $O(\lg m)$ lépéssel összefésülhető. Az ezt megoldó algoritmus neve LOG-ÖSSZEFÉSÜL. ■

Ez az algoritmus nem munkahatékony.

Páratlan-páros összefésülő algoritmus

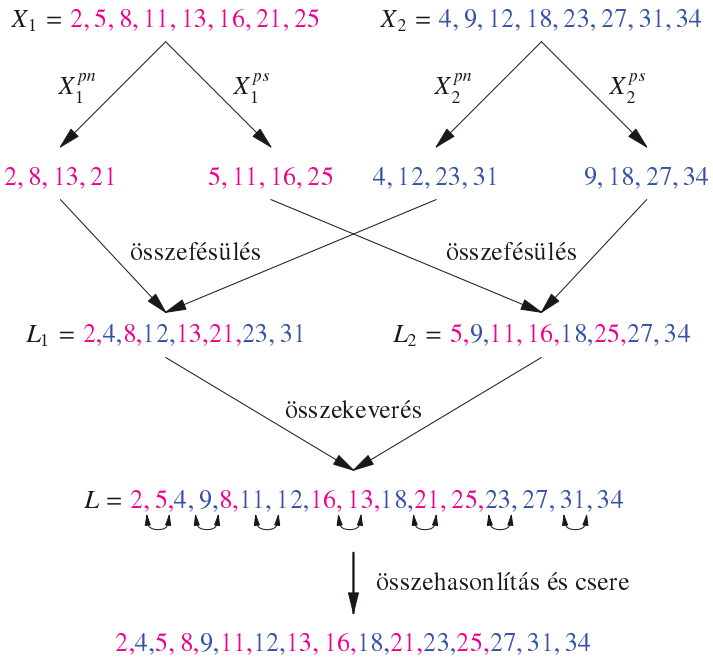
Ez a rekurzív algoritmus a klasszikus oszd-meg-és-uralkodj elvet alkalmazza.

Legyen $X_1 = \langle k_1, k_2, \ldots, k_m \rangle$ és $X_2 = \langle k_{m+1}, k_{m+2}, \ldots, k_{2m} \rangle$ a két bemenő tömb. Az egyszerűség kedvéért legyen m kettő hatvány és a kulcsok legyenek különbözőek. Az algoritmus $2m$ EREW PRAM processzort igényel, a kimenő adat az összefésült elemeket tartalmazó $Y[1\,..\,2m]$ tömb.

PÁRATLAN-PÁROS-ÖSSZEFÉSÜL(X_1, X_2)

1 **if** $m = 1$
2 **then** fésüljük össze a sorozatokat egyetlen összehasonlítással
3 **return** Y
4 **if** $m > 1$
5 **then** bontsuk fel X_1-et és X_2-t páros és páratlan részre, azaz legyen
$X_1^{ptn} = k_1, k_3, \ldots, k_{m-1}$ és $X_1^{prs} = k_2, k_4, \ldots, k_m$
6 hasonlóképpen bontsuk fel X_2-t is X_2^{ptn} és X_2^{prs} részekre
7 rekurzívan fésüljük össze X_1^{ptn}-t és X_2^{ptn}-t m processzoron
legyen az eredmény $L_1 = l_1, l_2, \ldots, l_m$. Ugyanakkor fésüljük össze
X_1^{prs}-t és X_2^{prs}-t másik m processzoron: az eredmény
legyen $L_2 = l_{m+1}, l_{m+2}, \ldots, l_{2m}$.
8 keverjük össze az L_1 és L_2 sorozatokat, azaz legyen
$L = l_1, l_{m+1}, l_2, l_{m+2}, \ldots, l_m, l_{2m}$.
9 hasonlítsuk össze az $(l_{m+i},\ l_{i+1})$ $(i = 1, 2, \ldots, m-1)$ párokat és szükség
esetén cseréljük fel őket. Az eredmény lesz a kimenő sorozat.
10 **return** Y

6.5. példa. *Kétszer nyolc szám összefésülése.* Legyen $X_1 = 2, 5, 8, 11, 13, 16, 21, 25$ és $X_2 = 4, 9, 12, 18, 23, 27, 31, 34$. A 16 szám rendezését mutatja a következő 6.14. ábra.



6.14. ábra. 16 szám rendezése a PÁRATLAN-PÁROS-ÖSSZEFÉSÜL algoritmussal. Az ábra színes változata megtalálható a 810. oldalon.

6.6. tétel (összefésülés $O(\lg m)$ lépéssel). *A* PÁRATLAN-PÁROS-ÖSSZEFÉSÜL *algoritmus két m hosszúságú kulcssorozatot $2m$* EREW PRAM *processzoron $\Theta(\lg m)$ idő alatt fésül össze.*

Bizonyítás. Legyen az algoritmus lépésszáma $W(n)$. Az 1. lépés $O(1)$ ideig tart. A 2. lépés $m/2$ ideig tart. Innen a

$$W(m) = W\left(\frac{m}{2}\right) + O(1) \tag{6.27}$$

rekurzív egyenlőséget kapjuk, melynek megoldása $W(m) = O(\lg m)$. ■

Ennek az algoritmusnak a helyességét a ***nulla-egy elv*** segítségével bizonyítjuk.

Egy összehasonlítás alapú rendező algoritmus ***egyszerű,*** ha az összehasonlítandó elemek sorozata előre meg van határozva (ekkor a következő összehasonlítás elemei nem függnek a mostani eredménytől).

Formálisan ez azt jelenti, hogy adott az összehasonlítandó elempárok indexeinek (i_1, j_1), (i_2, j_2), …, (i_m, j_m) sorozata.

6.7. tétel (nulla-egy elv). *Ha egy egyszerű összehasonlításos rendező algoritmus helyesen rendez egy n hosszúságú nulla-egy sorozatot, akkor tetszőleges kulcsokból álló n hosszúságú sorozatot is helyesen rendez.*

Bizonyítás. Legyen $\mathcal{A}$ egy egyszerű összehasonlításos (növekvőleg) rendező algoritmus és legyen S egy olyan kulcssorozat, melyet az adott algoritmus rosszul rendez. Ekkor a rosszul

rendezett S sorozatban van olyan kulcs, amely az i-edik ($1 \leq i \leq n-1$) helyen van annak ellenére, hogy S-ben legalább i nála kisebb elem van.

Legyen k S legelső (legkisebb indexű) ilyen kulcsa. A bemenő sorozatban írjunk a k-nál kisebb elemek helyére nullát, a többi elem helyére egyest. Ezt a módosított 0–1 sorozatot $\mathcal{A}$ helyesen rendezi, ezért a k helyére írt egyest a rendezett sorozatban legalább i darab nulla megelőzi.

Most kihasználjuk, hogy $\mathcal{A}$ egyszerű. A bemenő sorozatban színezzük pirosra a k-nál kisebb (nulla) elemeket, és kékre a többit (egyeseket). Indukcióval megmutatjuk, hogy az eredeti és a 0-1 sorozatnak megfelelő színes sorozatok minden összehasonlítás után azonosak. A színek szerint háromféle összehasonlítás van: kék, piros vagy különböző színű elemek összehasonlítása. Ha azonos színű elemeket hasonlítunk össze, akkor a színek sorozata egyik esetben sem változik. Ha viszont különböző színű elemeket hasonlítunk össze, akkor mindkét esetben a piros elem kerül a kisebb, és a kék elem a nagyobb indexű helyre. Eszerint k-t legalább i nála kisebb elem megelőzi a rendezett sorozatban. Az ellentmondás az állítás helyességét mutatja. ■

6.6. példa. *Egy nem összehasonlításos rendező algoritmus.* Legyen $k_1, k_2, \ldots, k_n$ egy bitsorozat. Rendezhetjük úgy, hogy megszámoljuk a nullák z számát, majd leírunk előbb z nullát, majd $n-z$ egyest. Erre az elv nem alkalmazható, mert ez nem összehasonlításos rendezés.

Az összefésülés viszont rendezés, és a páros-páratlan összefésülés egyszerű.

6.8. tétel. *A* PÁRATLAN-PÁROS-ÖSSZEFÉSÜL *algoritmus helyesen rendez tetszőleges számokból álló sorozatokat.*

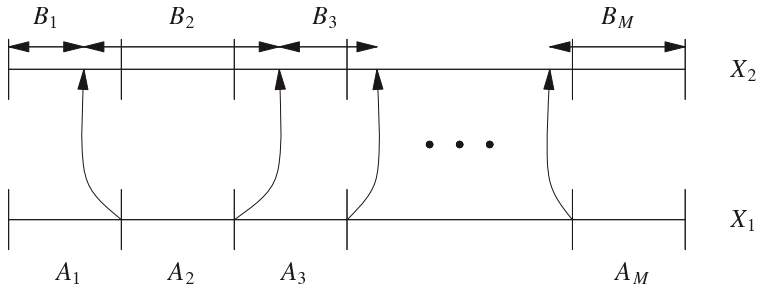
Bizonyítás. Legyenek X_1 és X_2 rendezett 0-1 sorozatok, melyek közös hossza m. Legyen q_1 (q_2) az X_1 (X_2) elején álló nullák száma. Az X_1^{ptn}-ban lévő nullák száma $\lceil q_1/2 \rceil$, és az X_1^{prs}-ban lévő nullák száma $\lfloor q_1/2 \rfloor$. Így az L_1-beli nullák száma $z_1 = \lceil q_1/2 \rceil + \lceil q_2/2 \rceil$ és az L_2-beli nullák száma $z_2 = \lfloor q_1/2 \rfloor + \lfloor q_2/2 \rfloor$.

z_1 és z_2 különbsége legfeljebb 2. Ez a különbség pontosan akkor kettő, ha q_1 és q_2 is páratlan. Egyébként a különbség legfeljebb 1. Tegyük fel, hogy $|z_1 - z_2| = 2$ (a többi eset hasonló). Most L_1-ben kettővel több nulla van. Amikor ezeket a harmadik lépésben összekeverjük, akkor L elején nullák vannak, azután 1,0, majd egyesek. A rendezetlen (*piszkos*) rész csak az 1,0. Amikor a harmadik lépésben az utolsó összehasonlítás és csere megtörténik, az egész sorozat rendezetté válik. ■

Munkahatékony algoritmus

Most $\lceil 2m/\lg m \rceil$ processzoron $O(\lg m)$ lépéssel végezzük az összefésülést. Ez a HATÉKONYAN-ÖSSZEFÉSÜL algoritmus az eredeti problémát $O(m/\lg m)$ részre osztja úgy, hogy mindegyikben $O(\lg m)$ hosszúságú rendezett sorozatokat kell összefésülni. Ezek a részproblémák soros algoritmussal $O(\lg m)$ lépéssel megoldhatók.

Legyen $X_1 = x_1, x_2, \ldots, x_m$ és $X_2 = x_{m+1}, x_{m+2}, \ldots, x_{m+m}$ a két bemenő sorozat. Osszuk X_1-et $\lceil m/\lg m \rceil$ részre: ekkor mindegyikben legfeljebb $\lceil \lg m \rceil$ kulcs lesz. A részek legyenek $A_1, A_2, \ldots, A_M$, ahol $M = m/\lg m$. Az A_1-beli legnagyobb kulcs legyen l_i ($i = 1, 2, \ldots, M$).



6.15. ábra. Munkahatékony összefésülő algoritmus.

Rendeljünk egy-egy processzort ezekhez az l_i elemekhez. Ezek a processzorok bináris kiválasztással meghatározzák l_i X_2-beli (rendezés szerinti) helyét. Ezek a helyek felbontják X_2-t M részre (ezek között üres részek is lehetnek – lásd a következő 6.15. ábrát). Jelöljük ezeket a részeket $B_1, B_2, \ldots, B_M$-mel. B_i-t az A_1-nek X_2-ben megfelelő részhalmaznak nevezzük.

Ekkor X_1 és X_2 összefésülését megkaphatjuk úgy, hogy rendre összefésüljük A_1-et B_1-gyel, A_2-t B_2-vel és így tovább, majd ezeket a sorozatokat egyesítjük.

6.9. tétel. *A* Hatékonyan-összefésül *két m hosszúságú rendezett kulcssorozatot $O(\lg m)$ lépésben összefésül $\lceil 2m/\lg m \rceil$ CREW PRAM processzoron.*

Bizonyítás. Az előző algoritmust alkalmazzuk.

Az A_i részek hossza $\lg m$, a B_i részek hossza azonban nagy is lehet. Ezért még egyszer alkalmazzuk a felbontást. Legyen A_i, B_i tetszőleges pár. Ha $|B_i| = O(\lg m)$, akkor A_i és B_i egy processzoron $O(\lg m)$ lépésben összefésülhető. Ha viszont $|B_i| = \omega(\lg m)$, akkor osszuk B_i-t $|B_i|/\lg m$ részre – ekkor minden rész legfeljebb $\lg m$ egymást követő kulcsot tartalmaz. Mindegyik részhez rendeljünk egy processzort, és az keresse meg az ennek a sorozatnak megfelelő részhalmazt A_i-ben: ehhez $O(\lg \lg m)$ lépés elegendő. Így A_i és B_i összefésülése $|B_i|/\lg m$ részproblémára redukálható, ahol minden részprobléma két $O(\lg m)$ hosszúságú sorozat összefésülése.

A felhasznált processzorok száma $\sum_{i=1}^{M} \lceil |B_i|/\lg m \rceil$, ami legfeljebb $m/\lg m + M$, és ez legfeljebb $2M$. ■

Összefésülés $O(\lg \lg m)$ idő alatt

Ha az előző algoritmust kiegészítjük az oszd-meg-és-uralkodj elvvel, akkor még gyorsabb algoritmust kapunk. A bemenő és kimenő adatok hasonlók az előző algoritmus adataihoz. Feltesszük, hogy m négyzetszám.

Gyorsan-összefésül(X_1, X_2)

1 bontsuk fel X_1-et $\sqrt{m} = b$ egyenlő részre – legyenek ezek $A_1, A_2, \ldots, A_b$.
2 legyen A_i-ben a legnagyobb kulcs l_i $(i = 1, 2, \ldots, b)$. Minden l_i-hez
rendeljünk b processzort.

3 ezek a processzorok végezzenek b-áris keresést X_2-ben,
hogy megtalálják l_i X_2-beli helyét.
4 ezzel X_2 b részre való felbontását kapjuk: legyenek ezek a részek $B_1, B_2, \ldots, B_b$
A B_i részhalmaz az A_i-nek X_2-ben megfelelő részhalmaz.
5 most X_1 és X_2 összefésüléséhez elegendő A_i és B_i $(i = 1, 2, \ldots, b)$
összefésülése. Az A_i-k mérete adott, viszont a B_i-k nagyon
nagyok (és nagyon kicsik) is lehetnek. Ezért újra felbontunk.
6 **return** Y

Legyen A_i és B_i tetszőleges pár. Ha $|B_i| = O(b)$, akkor a két sorozat $O(1)$ lépésben összefésülhető m^ϵ-áris kereséssel (ahol ϵ tetszőleges pozitív szám). Ha viszont $|B_i| = \omega(b)$, akkor B_i-t $\lceil |B_i|/b \rceil$ részre osztjuk, ahol B_i-nek minden részben legfeljebb b egymást követő eleme van. Rendeljünk minden részhez b processzort, hogy megtalálják az ehhez a halmazhoz tartozó részhalmazt A_i-ben: ehhez $O(1)$ lépés elég. Így A_i és B_i összefésülésének problémája $\lceil |B_i|/b \rceil$ részprobléma redukálható, ahol minden részprobléma két $O(b)$ hosszúságú sorozat összefésülése.

A felhasznált processzorok száma $\sum_{i=1}^{b} \lceil b|B_i|/b \rceil$, ami legfeljebb $2m$.

6.10. tétel (összefésülés $O(\lg \lg m)$ lépésben). *Két m hosszúságú rendezett kulcssorozat $O(\lg \lg m)$ lépésben összefésülhető $2m$ CREW PRAM processzoron.*

Bizonyítás. Legyen X_1 és X_2 a két adott sorozat. Legyenek a kulcsok különbözők és legyen $\sqrt{(m)} = b$. Az algoritmus a problémát $N \leq 2b$ részproblémára redukálja, melyek mindegyike két $O(b)$ hosszúságú rendezett sorozat összefésülése. A redukció m processzoron $O(1)$ lépésben elvégezhető. Ha az algoritmus lépésszáma $2m$ processzoron $T(m)$, akkor $T(m)$ kielégíti a

$$T(m) = T(O(b)) + O(1)$$

rekurzív egyenletet, melynek megoldása $O(\lg \lg m)$. ■

Ez az algoritmus nem munkahatékony, bár $\Theta(m)/\Theta(\lg \lg m) = \Theta(m/\lg \lg m)$ relatív sebessége nagyon közel van m-hez. Hatékonysága csak $\Theta(1/\lg \lg m)$.

6.5.4. Munkahatékony algoritmusok elemzése

Ebben az alfejezetben két munkahatékony prefixszámító algoritmust mutatunk be.

1. algoritmus: HATÉKONY-PREFIX1
Tegyük fel, hogy $n/\lg n$ (továbbiakban jelölésben $n/\lg n$) processzorunk van. Az algoritmus három fő szakaszból áll:

1. szakasz. Az X bemenetet $n/\lg n$ részre osztjuk, majd $n/\lg n$ processzor kiszámolja a hozzárendelt $X_{(i-1)\lg n} + 1, X_{(i-1)\lg n+2}, \ldots, X_{(i)\lg n}$ elem prefixeit rekurzívan. Az eredmény legyen $Z_{(i-1)\lg n+1}, Z_{(i-1)\lg n+2}, \ldots, Z_{i\lg n)}$. Ez $O(\lg n)$ lépés, gondoljunk csak a soros algoritmusra.

2. szakasz. $n/\lg n$ processzor együttesen alkalmazza a CREW-PREFIX-et segédalgoritmusként az $n/\lg n$ elem $Z_{\lg n}, Z_{2\lg n}, \ldots, Z_n$ prefixeinek számítására, ami az alábbi:

Ez a szakasz két lépésből áll.

Első lépés: Az első $n/2\lg n$ processzor rekurzívan kiszámítja az első $(Z_{\lg n}, Z_{2\lg n}, \ldots, Z_{n/2})$-hez tartozó prefixet. Az eredmény $Y_{\lg n}, Y_{2\lg n}, \ldots, Y_{n/2}$ lesz. Továbbá a második $n/2\lg n$ processzor rekurzívan kiszámítja az $(Z_{n/2+\lg n}, Z_{(n/2)+2\lg n}, \ldots, Z_n)$-hez tartozó prefixet. Az eredmény $Y_{n/2+\lg n}, Y_{(n/2)+2\lg n}, \ldots, Y_n$ lesz.

Második lépés: A második $n/2\lg n$ processzor párhuzamosan kiolvassa a globális memóriából $Y_{n/2}$-t, és $Y_{n/2}+Y_{n/2+\lg n}, Y_{n/2}+Y_{(n/2)+2\lg n}, \ldots, Y_{n/2}+Y_n$ prefixeket számítva előállítja a végeredmény második felét.

A CREW-PREFIX algoritmus műveletigénye n bemenő adat esetén: $T(n) = O(\lg n)$. Mivel nekünk ebben a részben nem n, hanem csak $n/\lg n$ processzorunk van, és $n/\lg n$ elemünk, ezért a 2. lépésre is igaz az $O(\lg n)$ lépésszám, ha behelyettesítjük n helyére $n/\lg n$-t:

$$O(\lg(n/\lg n)) = O(\lg(n(1/\lg n))) = O(\lg n + \lg(1/\lg n)) = O(\lg n)\ ,$$

és ennyi elég is a munkahatékonyság belátásához.

Ennek a lépésnek az eredménye legyen $Y_{\lg n}, Z_{2\lg n}, \ldots, Y_n$.

3. szakasz. Minden processzor aktualizálja az első lépésben kiszámolt értéket: a $P_i(i = 2, 3, \ldots, n/\lg n)$ processzor kiszámolja az $Y_{(i-1)\lg n} + Z_{(i-1)\lg n+1}, Y_{(i-1)\lg n} + Z_{(i-1)\lg n+2} \ldots, Y_{(i-1)\lg n} + Z_{(i)\lg n}$ prefixeket, majd az első processzor kiadja a $Z_1, Z_2, \ldots, Z_{\lg n}$ prefixeket.

HATÉKONY-PREFIX1(n, X) párhuzamos rekurzív eljárás, amely a CREW PRAM számítási modellt alkalmazza. Bemenete n (a bemenő sorozat hossza) és $X[1\,..\,n] = \langle x_1, x_2, \ldots, x_n\rangle$ (n hosszúságú sorozat), kimenete pedig $Y[1\,..\,n] = \langle y_1, y_2, \ldots, y_n\rangle$ (n hosszúságú sorozat, melynek elemei a prefixek).

HATÉKONY-PREFIX1(X)

```
1  P_i in parallel for i ← 1 to n/lg n)
2                  do Z[(i − 1) lg n + 1] ← X[(i − 1) lg n + 1]
3  for j ← (i − 1) lg n + 2 to i lg n
4      do Z[j] ← Z[j − 1]X[i]
5  P_i in parallel for i ← 1 to n/lg n)
6                  do T[i] ← Z[i lg n]
7  CREW-PREFIX(n/lg n, T, Y)
8  P_i in parallel for i ← 2 to n/lg n)
9                  for j ← (i − 1) lg n + 1 to i lg n
10                     do Z[j] ← Z[j]Y[(i − 1) lg n]
11 return Y
```

6.11. lemma. *A* HATÉKONY-PREFIX1 *algoritmus $n/\lg n$ CREW PRAM processzoron $O(\lg n)$ lépésben munkahatékonyan számítja ki n elem prefixeit.*

Bizonyítás. Az algoritmus összlépésszáma: Az első rész $O(\lg n)$ műveletigényű. A második rész $O(\lg n)$ műveletigényű. A harmadik rész $O(\lg n)$ műveletigényű. Összesen: $O(\lg n) + O(\lg n) + O(\lg n) = O(\lg n)$.

Mind a három rész $n/\lg n$ processzort használ. ■

2. algoritmus: HATÉKONY-PREFIX2

Ez az algoritmus hasonló az előzőhöz, de most csak $\lg n$ processzort használunk. A célunk az, hogy megmutassuk, felcserélhetők egymással a HATÉKONY-PREFIX1 algoritmusban szereplő processzorszám és lépésszám értékek.

1. szakasz. Az X bemenetet $\lg n$ részre osztjuk, majd $\lg n$ processzor kiszámolja a hozzárendelt $X_{(i-1)(n/\lg n)+1}, X_{(i-1)(n/\lg n)+2}, \ldots, X_{(i)(n/\lg n)}$ elem prefixeit rekurzívan. Az eredmény legyen $Z_{(i-1)(n/\lg n)+1}, Z_{(i-1)(n/\lg n)+2}, \ldots, Z_{(i)(n/\lg n)}$. Most $\lg n$ részre osztottuk a bemenetet, így egy rész hossza $n/\lg n$, tehát itt is a soros algoritmusra hivatkozva ennek a résznek a lépésszáma: $O(n/\lg n)$.

2. szakasz. $\lg n$ processzor együttesen alkalmazza a CREW-PREFIX-et segédalgoritmusként a $\lg n$ elem $Z_{n/\lg n}, Z_{2(n/\lg n)}, \ldots, Z_n$ prefixeinek számítására, ami az alábbi:

Ez a szakasz két lépésből áll.

Első lépés: Az első $\lg n/2$ processzor rekurzívan kiszámítja az első $Z_{n/\lg n}, Z_{2(n/\lg n)}, \ldots, Z_{n/2}$-hez tartozó prefixet. Az eredmény $Y_{n/\lg n}, Y_{2(n/\lg n)}, \ldots, Y_{n/2}$ lesz. Továbbá a második $\lg n/2$ processzor rekurzívan kiszámítja a $Z_{n/2+n/\lg n}$, $Z_{(n/2)+2(n/\lg n)}, \ldots, Z_n$-hez tartozó prefixet. Az eredmény $Y_{n/2+n/\lg n}, Y_{(n/2)+2(n/\lg n)}, \ldots, Y_n$ lesz.

Második lépés: A második $(\lg n)/2$ processzor párhuzamosan kiolvassa a globális memóriából $Y_{n/2}$-t, és $Y_{n/2} + Y_{n/2+n/\lg n}, Y_{n/2} + Y_{(n/2)+2(n/\lg n)}, \ldots, Y_{n/2} + Y_n$ prefixeket számítva előállítja a végeredmény második felét.

A CREW-PREFIX eljárást itt is segédalgoritmusként használtuk $\lg n$ méretű bemenetre, így most az alábbiak szerint módosul a lépésszáma:

A 2. lépés $O(\lg n)$ lépésszáma az alábbiak alapján kapható meg: behelyettesítjük n helyére $\lg n$-t: $O(\lg(\lg n)) = O(\lg n)$. Ennek a lépésnek az eredménye legyen $Y\lg n, Z_{2\lg n}, \ldots, Y_n$.

3. szakasz. Minden processzor aktualizálja az első lépésben kiszámolt értéket: a P_i $(i = 2, 3, \ldots, \lg n)$ processzor kiszámolja az $Y_{(i-1)(n/\lg n)} + Z_{(i-1)(n/\lg n)+1}, Y_{(i-1)(n/\lg n)} + Z_{(i-1)(n/\lg n)+2}, \ldots, Y_{(i-1)(n/\lg n)} + Z_{(i)(n/\lg n)}$ prefixeket, majd az első processzor kiadja a $Z_1, Z_2, \ldots, Z_{\lg n}$ prefixeket.

HATÉKONY-PREFIX2(n, X) párhuzamos rekurzív eljárás, amely CREW PRAM számítási modellen alapul. Bemenete n (a bemenő sorozat hossza) és $X[1\,..\,n] = \langle x_1, x_2, \ldots, x_n\rangle$ (n hosszúságú sorozat), kimenete pedig $Y[1\,..\,n] = \langle y_1, y_2, \ldots, y_n\rangle$ (n hosszúságú sorozat, melynek elemei a prefixek).

Az algoritmus pszeudokódja a következő.

HATÉKONY-PREFIX2(X)

```
1  P_i in parallel for i ← 1 to lg n
2                 Z[(i − 1)(n/ lg n) + 1] := X[(i − 1)(n/ lg n) + 1]
3                 for j ← (i − 1)(n/ lg n) + 2 to i lg n
4                 Z[j] ← Z[j − 1]X[i]
5  P_i in parallel for i ← 1 to lg n
6                 T[i] ← Z[i(n/ lg n)]
7                 CREW-PREFIX(lg n, T, Y)
```

```
 8  P_i in parallel for i ← 2 to lg n
 9                    for j ← (i − 1)(n/ lg n) + 1 to in/ lg n
10                    Z[j] ← Z[j]Y[(i − 1)(n/ lg n)]
11  return Y
```

6.12. lemma. *A* Hatékony-prefix2 *algoritmus* $\lg n$ *CREW PRAM processzoron* $O(n/\lg n)$ *lépésben munkahatékonyan számítja ki* n *elem prefixeit.*

Bizonyítás. Az algoritmus összlépésszáma: Az első rész $O(n/\lg n)$ műveletigényű. A második rész $O(\lg n)$ műveletigényű. A harmadik rész $O(n/\lg n)$ műveletigényű. Összesen: $O(n/\lg n) + O(\lg n) + O(n/\lg n) = O(n/\lg n)$.

Mind a három rész $\lg n$ processzort használ. ■

A Hatékony-prefix2 algoritmus nemcsak munkahatékony, hanem aszimptotikusan optimális is, ezt bizonyítja a következő tétel:

6.13. tétel. *Párhuzamos prefixszámításhoz* $\Omega(\lg n)$ *számú* $\oplus$ *műveletet kell végezni.*

Bizonyítás. Vegyünk egy egyszerűbb feladatot, az X elemeinek összeadását. Ez nyilván kevesebb lépést igényel, mivel prefixszámítás esetén is össze kell adni az összes elemet (legutolsó prefix), és még további prefixekett is meg kell határozni.

Megmutatjuk, hogy X elemeinek összeadásához a PRAM modellben legalább $\lg n$ időegységre van szükség. A PRAM számítási modellben egy lépésben egy processzor legfeljebb két elemet adhat össze. Az első lépésben tehát legfeljebb $(n/2)$-re, a másodikban $(n/4)$-re, ... csökkenthető az összeadandók száma, ezért valóban legalább szükség van legalább $\lg n$ lépésre. ■

További munkahatékony algoritmusok

Az előbbi két algoritmus egy korlátot ad a processzorszámra és lépésszámra vonatkozóan. Az előbbi munkahatékony algoritmusokat vizsgálva ennél többet is mondhatunk: tetszőleges $a \in (0,1)$ számhoz meg tudunk adni olyan munkahatékony algoritmust, amely n^a részre osztja a sorozatot.

Ennek belátásához végig kell nézni az algoritmus mindhárom lépését, hogy azokra teljesül-e a munkahatékonyság (az algoritmus felfogható három részalgoritmus egyesítéseként, ahol a futási idők összeadódnak). Az első lépésben a bemenetet n^a részre osztva, minden rész n^{1-a} elemet tartalmaz, így a soros algoritmust figyelembe véve ennek a résznek a lépésszáma n^{1-a} lesz.

A harmadik lépésben szintén az n^a részt aktualizáljuk, ami n^{1-a} lépésből áll. A második lépés okozhat problémát, miszerint a $\lg n^a$ vajon aszimptotikusan nagyobb-e az (n^{1-a})-nál. A l'Hospital-szabály segítségével belátható, hogy ha $a \in (0,1)$, akkor

$$\lim_{n\to\infty} \frac{n^a}{\lg n} = \infty.$$

Innen adódik, hogy ha $a \in (0,1)$, akkor a Hatékony-prefix algoritmust véve alapul a

futási idő tetszőleges n^a alakú polinom lehet úgy, hogy minden esetben munkahatékony algoritmust kapjunk.

Ehhez elég azt észrevenni, hogy az algoritmus első részének műveletigénye $O(n^{1-a})$, a második részének műveletigénye $O(\lg n^a)$ és a harmadik rész műveletigénye ugyancsak $O(n^{1-a})$.

Így az összes műveletigény $O(n^{1-a}) + O(\lg n^a) + O(n^{1-a}) = O(n^{1-a})$.

6.5.5. Kiválasztás

Adott $n \geq 2$ kulcs és egy i $(1 \leq i \leq n)$ egész szám. A feladat az i-edik legkisebb kulcs kiválasztása. Mivel a kiválasztáshoz minden elemet meg kell vizsgálni, ezért $N(n) = \Omega(n)$. Erre a feladatra ismert olyan $\mathcal{A}$ soros algoritmus, amelyikre $W(n, \mathcal{A}) = O(n)$, tehát $\mathcal{A}$ aszimptotikusan optimális.

Ehhez hasonló a ***keresési feladat,*** amelyben azt kell eldönteni, hogy adott elem előfordul-e a vizsgált sorozatban – és ha igen, milyen indexszel. Ennél a feladatnál tagadó válasz is lehetséges és egy elemről tulajdonságai alapján eldönthető, megfelel-e a keresési feladatnak.

Először 3 speciális esetet vizsgálunk, majd egy munkahatékony véletlenített algoritmust ismertetünk.

Kiválasztás konstans időben n^2 processzoron

Legyen $i = n$, azaz a legnagyobb kulcsot keressük. Ez a feladat a következő NÉGYZETES-KIVÁLASZT algoritmussal n^2 CRCW processzoron $O(1)$ lépéssel elvégezhető.

A bemenő adatokat (n különböző kulcsot) a $K = k_1, k_2, \ldots, k_n$ tömb, a megtalált legnagyobb kulcsot pedig az y változó tartalmazza.

NÉGYZETES-KIVÁLASZT(K,Y)

```
1  if n = 1
2     then y ← x1
3          return y
4  Pij in parallel for i ← 1 to n,  j ← 1 to n
5                    számítsa ki a xij = ki < kj értéket
6  osszuk az n^2 processzort n csoportba (G1, ..., Gn) úgy, hogy a Gi csoportba a
         Pi,1, ..., Pi,n processzorok kerüljenek. Mindegyik csoport végezzen logikai VAGY
         műveletet az xi1, ..., xin logikai változókkal
7  ha a Gi csoport számítási eredménye a 4. lépésben HAMIS, akkor a csoport Pi1 processzora
     adja meg (y = ki)-t kimenetként
```

Legyenek a bemenő adatok $k_1, \ldots, k_n$. Az összehasonlításokat párhuzamosan végezzük a P_{ij} $(1 \leq i, j \leq n)$ processzorokon úgy, hogy P_{ij} az $x_{ij} = (k_i < k_j)$ logikai értéket számítja ki. Feltehetjük, hogy a kulcsok különbözőek. Ha mégse, k_i helyett a (k_i, i) párt alkalmazva különbözővé tehetők: ehhez minden kulcshoz egy $(\lg n)$-bites számot kell hozzáadni. Ekkor egyetlen olyan kulcs van, amelyikre minden összehasonlítás eredménye HAMIS. Ez a kulcs egy logikai VAGY művelettel azonosítható.

6.14. tétel (kiválasztás $O(1)$ lépéssel). *n kulcs közül a maximális $O(1)$ lépéssel meghatározható n^2 CRCW közös PRAM processzoron.*

Bizonyítás. Az első és a harmadik szakasz egységnyi ideig tart. A második szakasz $O(1)$ lépéssel elvégezhető. ■

Ennek az algoritmusnak a relatív sebessége $\Theta(n)$. Az elvégzett munka $\Theta(n^2)$. Ezért a hatékonyság $\Theta(n)/\Theta(n^2) = \Theta(1/n)$. Tehát az algoritmus nem munkaoptimális.

Kiválasztás logaritmikus időben n processzoron

Most megmutatjuk, hogy a maximális elem p közös CRCW processzoron $O(\lg\lg p)$ lépéssel meghatározható. A technika az oszd-meg-és-uralkodj. Az egyszerűség kedvéért feltesszük, hogy p négyzetszám.

Legyenek a bemenő adatok $X = k_1, k_2, \ldots, k_p$. Legyen az algoritmusunk lépésszáma $T(p)$. A bemenő adatokat $\sqrt{p} = a$ csoportra osztjuk úgy, hogy minden csoportban a elem legyen. Minden csoporthoz rendeljünk q processzort – ekkor a csoportok maximális eleme párhuzamosan számítható. Mivel csoportonként q elem és ugyanannyi processzor van, a csoport maximális eleme $T(a)$ lépéssel meghatározható. Legyenek $M_1, M_2, \ldots, M_a$ a csoportok maximális elemei. Ezek maximuma lesz az algoritmus kimenete. Mivel most csak q elemünk van, az összes processzort alkalmazhatjuk.

A következő rekurzív CRCW algoritmus $O(\lg\lg p)$ lépést tesz. Bemenő és kimenő adatok az előző algoritmus adataihoz hasonlók.

GYÖKÖS-KIVÁLASZT(X, K, y)

1 **if** $p = 1$
2 **then** $y \leftarrow x_1$
3 **return** Y
4 osszuk a bemenetet a részre $(K_1, K_2, \ldots, K_a)$ úgy, hogy K_i a $k_{(i-1)a+1}, k_{(i-1)a+2}, \ldots, k_{ia}$ elemeket tartalmazza. Hasonlóképpen csoportosítsuk a processzorokat úgy, hogy a P_i $(1 \le i \le a)$ csoportba a $P_{(i-1)a+1}, P_{(i-1)a+2}, \ldots, P_{ia}$ processzorok tartozzanak. A P_i csoport határozza meg rekurzívan a K_i csoport maximális elemét.
5 legyenek a csoportok maximális elemei $M_1, M_2, \ldots, M_a$, és határozzuk meg ezek maximumát a NÉGYZETES-KIVÁLASZT algoritmussal
6 **return** Y

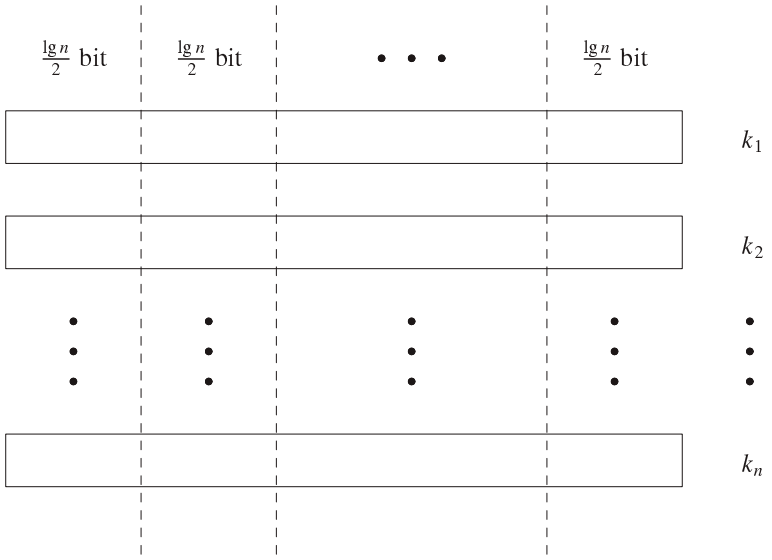
6.15. tétel (kiválasztás $O(\lg\lg p)$ lépéssel). *A* GYÖKÖS-KERES *algoritmus p közös* CRCW PRAM *processzoron $O(\lg\lg p)$ lépéssel meghatározza p kulcs közül a legnagyobbat.*

Bizonyítás. Ennek az algoritmusnak az első lépése $W(\sqrt{p})$, második lépése $O(1)$ ideig tart. Ezért $W(p)$ kielégíti a

$$W(p) = W(\sqrt{p}) + O(1) \tag{6.28}$$

rekurzív egyenletet, melynek megoldása $O(\lg\lg p)$. ■

A GYÖKÖS-KERES algoritmus összes munkája $\Theta(p \lg\lg p)$, ezért hatékonysága $\Theta(p)/\Theta(p \lg\lg p) = \Theta(1/\lg\lg p)$, így ez az algoritmus sem munkahatékony.



6.16. ábra. Maximális egész szám kiválasztása

Kiválasztás egész számok közül

Legyen a feladat ismét n kulcs maximumának meghatározása. Ha a kulcsok egyetlen bitből állnak, akkor a maximum keresése visszavezethető a logikai VAGY műveletre és ezért $O(1)$ lépéssel meghatározható. Ebből adódik a kérdés: mekkora intervallumban lehetnek a kulcsok ahhoz, hogy p processzoron konstans lépéssel meg tudjuk határozni a maximális elemet?

Legyen c adott konstans, a kulcsok pedig legyenek a $[0, n^c]$ intervallumban. Ekkor a kulcsok legfeljebb $c \lg n$ bites bináris számok. Az egyszerűség kedvéért feltesszük, hogy pontosan ennyi bitesek (a számok elejére szükség esetén nullákat írunk).

A következő CRCW algoritmus $O(1)$ lépést tesz.

Az alapötlet az, hogy a számok $b_1, b_2, \ldots, b_{2c}$ bitjeit $(\lg n)/2$ hosszúságú *részekre* bontjuk. Az i-edik rész a $b_{(i-1)+1}, b_{(i-1)+2}, \ldots, b_{(i-1)+b_{(i-1)+(\lg n)/2}}$ biteket tartalmazza, a részek száma $2c$.

Ezt a helyzetet mutatja a 6.16. ábra: először az ábra első oszlopában lévő bitek alapján keressük a maximális kulcsot.

Az algoritmus futási idejét a következő állítással jellemezzük.

6.16. tétel (kiválasztás egész számok közül). *Ha a kulcsok a $[0, n^c]$ intervallumból vett egész számok, akkor p kulcs közül a maximális $O(1)$ lépéssel meghatározható p* CRCW PRAM *processzoron tetszőleges c konstans esetében.*

Bizonyítás. Tegyük fel, hogy a kulcsok maximumát a $(\lg n)/2$ legfontosabb bit alapján határozzuk meg.

Legyen az első részben a maximum M. Ekkor azok a kulcsok, melyek legfontosabb bitjei nem M-et adnak, biztosan nem maximálisak. Ezt az alaplépést megismételjük $2c$-

szer, azaz minden $(\lg p)/2$ bitre pontosan egyszer. Legalább egy kulcs megmarad az utolsó lépés után is – az lesz az eredmény. Az utolsó rész lehet rövidebb, mint $(\lg p)/2$ bit.

Ha egy kulcs legfeljebb $(\lg n)/2$-bites, akkor az értéke legfeljebb $\sqrt{n}-1$. Ezért az EGÉSZET-KIVÁLASZT első lépésében a $[0, \sqrt{n}-1]$ intervallumba eső egész kulcsok maximumát kell meghatározni. Rendeljünk minden kulcshoz egy processzort és használjunk $\sqrt{n}$ közös memóriarekeszt $(M_1, M_2, \ldots, M_{\sqrt{n}-1})$, melyek tartalma kezdetben $-\infty$. Egy párhuzamos lépésben a P_i processzor k_i-t ír az M_{k_i} memóriarekeszbe. Ezután az n kulcs maximuma a $\sqrt{n}$ memóriarekesz tartalmából n processzorral a 2.9. tétel alapján konstans idő alatt meghatározható. ■

Az algoritmus pszeudokódja a következő.

Az EGÉSZET-KIVÁLASZT algoritmus bemenő adatai a p processzorszám és a különböző kapcsolókat tartalmazó $X = k_1, k_2, \ldots, k_p$ tömb, kimenő adata pedig az y változó.

EGÉSZET-KIVÁLASZT(p, X, y)

```
1  for i ← 1 to 2c
2      do határozzuk meg a megmaradt kulcsok maximumát i-edik részük alapján
           legyen a maximum M
3          hagyjuk el azokat a kulcsokat, melyek i-edik része kisebb, mint M
4  y legyen a megmaradt kulcsok egyike
5  return y
```

Általános kiválasztási feladat

Tegyük fel, hogy az $X = \langle k_1, k_2, \ldots, k_n \rangle$ sorozat különböző kulcsokat tartalmaz és az i-edik legkisebb kulcsot akarjuk kiválasztani. Legyen most az x_i kulcs *rangja* eggyel nagyobb, mint a nála kisebb kulcsok száma (ez a definíció eggyel nagyobb értéket ad, mint a korábban használt).

Ezt a rangot a 6.5-5. gyakorlat szerint $n/\lg n$ CREW PRAM processzoron bármely kulcsra $O(\lg n)$ lépésben meg tudjuk határozni.

Ha $n^2/\lg n$ processzorunk van, akkor azokat $C_1, C_2, \ldots, C_n$ csoportokba oszthatjuk úgy, hogy minden csoportban $n/\lg n$ processzor legyen. A C_j $(1 \le j \le n)$ csoport $O(\lg n)$ lépésben meghatározza a k_j kulcs rangját X-ben. Annak a csoportnak egyik processzora, amelyik az i rangot határozta meg, adja a kimenetet. Az így kapott algoritmus neve legyen ÁLT-KIVÁLASZT.

6.17. tétel (általános kiválasztás). *Az* ÁLT-KIVÁLASZT *algoritmus $n^2/\lg n$ processzoron n különböző kulcs közül $\Theta(\lg n)$ lépésben meghatározza az i-edik legkisebbet.*

Nem nehéz belátni, hogy az ÁLT-KIVÁLASZT algoritmus munkája $\Theta(n^2)$, tehát ez az algoritmus sem munkahatékony.

6.5.6. Rendezés

Adott $n \ge 2$ kulcs. A feladat ezek csökkenő vagy növekvő sorrendbe való rendezése.

Ismert, hogy ha a megengedett művelet a szokásos összehasonlítás, akkor minden $\mathcal{A}$ soros algoritmusnak $N(n, \mathcal{A}) = \Omega(n \lg n)$ lépésre van szüksége, másrészt vannak $O(n \lg n)$

lépésszámú összehasonlítás alapú algoritmusok, amelyek tehát aszimptotikusan optimálisak.

Más műveletek vagy a rendezendő kulcsok speciális tulajdonságai esetében a rendezés $O(n)$ lépéssel is megoldható. Ha legrosszabb esetben minden kulcsot meg kell vizsgálni, akkor természetesen a lépésszám $N(n) = \Omega(n)$.

Tehát mind az összehasonlítás alapú, mind pedig a speciális esetben ismert aszimptotikusan optimális soros algoritmus.

Vizsgáljuk meg a következő kérdéseket. Hány rendező algoritmus van? Ezek közül hány egyszerű, hány optimális (aszimptotikusan, szigorúan)?

Ezekre a kérdésekre nem könnyű válaszolni – például először pontosan definiálnunk kell, mi is az a rendező algoritmus.

Szűkítsük a kérdést: hány összehasonlításra van szükség n elem rendezéséhez? Jelöljük ezt a számot $c(n)$-nel. Ismert, hogy

$$\left\lceil \sum_{i=1}^{n} \lg i \right\rceil \leq c(n) \leq \sum_{i=1}^{n} \lceil \lg i \rceil \tag{6.29}$$

és hogy

$$c(n) \leq n \lg n - (n-1)\,. \tag{6.30}$$

Az alsó becslés döntési fákkal vagy információelméleti eszközökkel igazolható (lásd [90]), a felső becslések pedig a bináris beszúró, illetve az összefésüléses rendező jellemző adatai.

4 elemre az alsó és a felső becslések egyaránt ötöt adnak. 5 elemre $5! = 120$ miatt az alsó becslés 7, viszont az előbbi algoritmusoknak 8 összehasonlításra van szüksége.

Rendezés logaritmikus időben n^2 processzoron

n^2 processzoron a kulcsok rangja $O(\lg n)$ lépéssel meghatározható. Ha a rangokat ismerjük, akkor a rendezés egy párhuzamos írással megoldható.

Tehát igaz a következő tétel.

6.18. tétel (rendezés $O(\lg n)$ lépésben). *n kulcs n^2 CREW PRAM processzoron rendezhető $O(\lg n)$ lépéssel.*

Mivel a kulcsok meghatározása $\Omega(\lg n)$ ideig tart, ez a módszer $\Theta(n^2 \lg n)$ munkát igényel, azaz nem munkahatékony.

Páratlan-páros algoritmus – $O(\lg^2 n)$ futási idővel

Ez az algoritmus a klasszikus oszd-meg-és-uralkodj elvet alkalmazza. Az egyszerűség kedvéért legyen n kettő hatvány és a kulcsok legyenek különbözők.

A következő EREW PRAM algoritmus $O(\lg^2 n)$ lépést tesz. Bemenete a kulcsokat tartalmazó $X = \langle x_1, \ldots, x_p \rangle$ tömb, kimenete pedig a rendezett kulcsokat tartalmazó $Y = \langle y_1, \ldots, y_p \rangle$ tömb.

PÁRATLAN-PÁROS-RENDEZ(X)

```
1  if n = 1
2     then Y ← X
3     return Y
4  osszuk az X bemenetet két részre: ezek legyenek
          X'_1 = k_1, k_2, ..., k_{n/2} és X'_2 = k_{n/2+1}, k_{n/2+2}, ..., k_n.
5         rendezze n/2 processzor rekurzívan X'_1-t. Az eredmény legyen X_1. Ugyanakkor
          rendezze n/2 processzor rekurzívan X'_2-t. Az eredmény legyen X_2.
6         fésüljük össze X_1-et és X_2-t 2m processzoron
                  a PÁRATLAN-PÁROS-ÖSSZEFÉSÜL algoritmussal Y-ná.
7  return Y
```

Ez az algoritmus hasonlít a soros összefésüléses algoritmusra. Ott azonban a sorozatok legkisebb elemeit hasonlítjuk össze és a kisebb elem az összehasonlítás eredménye.

6.19. tétel (rendezés $O(\lg^2 n)$ lépéssel). *n kulcs n* EREW PRAM *processzoron rendezhető $O(\lg^2 n)$ lépéssel.*

Bizonyítás. Legyen $W(n)$ az algoritmus lépésszáma. Az 1. lépés $O(1)$ ideig tart, a 2. lépés $W(n/2)$ ideig, a 3. lépés pedig $O(\lg n)$ ideig. Ezért $W(n)$ kielégíti a

$$W(n) = O(1) + W\left(\frac{n}{2}\right) + O(\lg n) \tag{6.31}$$

rekurzív egyenlőséget, melynek megoldása $W(n) = O(\lg^2 n)$. ■

6.7. példa. Rendezés 16 processzorral. Rendezzük 16 processzoron a következő számokat: 25, 21, 8, 5, 2, 13, 11, 16, 23, 31, 9, 4, 18, 12, 27, 34. Az első lépésben a páros és páratlan részeket kapjuk meg, majd a másodikban az első 8 processzor a páratlan részből kapja az $X_1 = 2, 5, 8, 11, 13, 16, 21, 25$-öt, a második 8 processzor pedig az $X_2 = 4, 9, 12, 18, 23, 27, 31, 34$-et. A harmadik lépésben kapjuk az eredményt.

Ez az algoritmus $\Theta(n \lg^2 n)$ munkát végez. Hatékonysága $\Theta(1/\lg n)$, gyorsítása $\Theta(n/\lg n)$.

Preparata algoritmusa – $O(\lg n)$ futási idővel

Több processzorral a lépésszám csökkenthető: Preparata rekurzív algoritmusa $n \lg n$ CREW PRAM processzoron $\lg n$ párhuzamos lépést végez. A bemenő $X[1..p]$ tömb a rendezendő, a kimenő $Y[1..p]$ tömb pedig a rendezett kulcsokat tartalmazza.

PREPARATA(X)

```
1  if n ≤ 20
2     then rendezzük az X bemenetet tetszőleges rendező algoritmussal
3          return Y
```

4 osszuk az adott n kulcsot $\lg n$ részre úgy, hogy mindegyikben $n/\lg n$ kulcs legyen.
Rendezzük a részeket külön, rekurzívan, mindegyik részhez n processzort rendelve. A rendezett sorozatok legyenek $S_1, S_2, \ldots, S_{\lg n}$.
5 fésüljük össze S_i-t és S_j-t $(1 \le i, j \le \lg n)$ párhuzamosan
6 rendeljünk $\lg n$ processzort a kulcsok eredeti sorozatra vonatkozó rangjának meghatározásához
7 legyen Y olyan tömb, amely a kulcsokat tartalmazza a rangok sorrendjében.
8 **return** Y

Ez az algoritmus oszd-meg-és-uralkodj elvű. A kulcssorozatot $\lg n$ részre osztjuk, majd a részeket páronként összefésülve minden kulcsnak minden részre nézve meghatározzuk a rangját. Ezután a kulcsok tényleges rangja az előbbi rangok összege.

Ha a harmadik lépésben minden (i, j) párhoz $n/\lg n$ processzort rendelünk, akkor az összefésülés $O(\lg\lg n)$ lépéssel elvégezhető.

A negyedik lépésben a rang számítása párhuzamosan végezhető a harmadik lépésben kapott $\lg n$ rang összeadásával: ez a prefixet számító algoritmussal $O(\lg\lg n)$ lépéssel megoldható.

6.20. tétel (rendezés $O(\lg n)$ lépéssel). *A* Preparata *algoritmus n elemet $n\lg n$ CREW PRAM processzoron $O(\lg n)$ lépéssel rendez.*

Bizonyítás. Legyen a Preparata-algoritmus lépésszáma $W(n)$. Az első lépés lépésigénye $W(n/\lg n)$, a második és harmadik lépésé együtt $O(\lg\lg n)$. Ezért

$$W(n) = W\left(\frac{n}{\lg n}\right) + O(\lg\lg n), \tag{6.32}$$

melynek megoldása (helyettesítéses módszerrel) $W(n) = O(\lg n)$. ■

A lassulásra vonatkozó tétel segítségével kapjuk a következő állítást.

6.21. következmény (rendezés $(n\lg n)/t$ processzoron). *Tetszőleges $t \ge 1$ egész számra n tetszőleges kulcs rendezhető $O(t\lg n)$ lépésben $(n\lg n)/t$ CREW PRAM processzoron.*

A Preparata-algoritmus munkája ugyanannyi, mint a páros-páratlan rendező algoritmusé, viszont a gyorsítása jobb: $\Theta(n)$. Mindkét algoritmus hatékonysága $\Theta(1/\lg n)$.

Gyakorlatok

6.5-1. A globális memória M_1 rekeszében van bizonyos adat. Másoljuk át ezt az adatot az $M_2, M_3, \ldots, M_n$ rekeszekbe. Mutassuk meg, hogyan lehet ezt megvalósítani $O(\lg n)$ lépéssel n EREW PRAM processzor felhasználásával.

6.5-2. Adjunk meg egy olyan algoritmust, amely $n/\lg n$ PRAM processzor felhasználásával $O(\lg n)$ lépéssel megoldja az előző gyakorlatot.

6.5-3. Legyen $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$. Adjunk $O(1)$ idejű CREW PRAM algoritmust a polinom értékének adott x helyen való kiszámítására. Mennyi processzort igényel a javasolt algoritmus?

6.5-4. Adjunk meg egy $O(\lg\lg n)$ idejű algoritmust, amely $n/\lg\lg n$ közös CRCW PRAM processzoron $O(\lg\lg n)$ lépésben megadja n tetszőleges szám maximumát.
6.5-5. Legyen A egy n kulcsot tartalmazó tömb. Mutassuk meg, hogy $n/\lg n$ CREW PRAM processzoron $O(\lg n)$ lépésben meghatározható tetszőleges $k \in A$ kulcs rangja.
6.5-6. Tervezzünk egy $O(1)$ lépésszámú algoritmust, amely n közös CRCW PRAM processzoron eldönti, hogy adott $A[1\,..\,n]$ tömb elemei között előfordul-e az 5, és ha igen, megadja a legnagyobb olyan i indexet, amelyre $A[i] = 5$.
6.5-7. Tervezzünk algoritmust, amely n^2 CREW PRAM processzoron $O(1)$ lépésben összefésül két n hosszúságú rendezett sorozatot.
6.5-8. Határozzuk meg a fejezetben tárgyalt algoritmusok gyorsítását, összes lépésszámát és hatékonyságát.

Feladatok

6-1. Közös elem
Tervezzünk n^2 CRCW PRAM processzoron $O(1)$ lépésszámú algoritmust annak eldöntésére, hogy adott $A[1\,..\,n]$ és $B[1\,..\,n]$ tömböknek van-e közös eleme.
6-2. Minimális feszítőfa
Párhuzamosítsuk a minimális feszítőfák meghatározására szolgáló Kruskal-algoritmust és Prim-algoritmust. Tervezzünk algoritmust arra a speciális esetre, amikor az élek súlya csak 0 vagy egy lehet.
6-3. Összes csúcspár távolsága
Párhuzamosítsuk a gráfok összes csúcspárjának távolságát meghatározó Bellman–Ford algoritmust.
6-4. Körmentesség
Tervezzünk egy $\mathcal{P}$ párhuzamos algoritmust annak eldöntésére, hogy adott irányítatlan gráf tartalmaz-e kört. Elemezzük a különböző nagyságrendű processzorszám esetében elérhető $W(n, p, \mathcal{P})$ lépésszámokat.

Megjegyzések a fejezethez

A számítógépek felépítését elsősorban Claudia Leopold [289], Leighton [283, 284], valamint Sima, Fountain és Kacsuk [432] monográfiája alapján tárgyaljuk. A párhuzamos programozásról szóló alfejezetek alapja Grama, Gupta, Karypys és Kumar [172], Leopold [289], valamint Roosta [396] könyve.

Az 500 legnagyobb teljesítményű számítógép adatait [467] tartalmazza.

A tárgyalt párhuzamos algoritmusok többsége Berman és Paul [45], Cormen, Leiserson és Rivest [89], Horowitz, Sahni és Rajasekakaran [211], Iványi Antal [221], valamint Jaja [228] tankönyvéből származik.

A számítógépek ismertetett osztályozási rendszereit Flynn [132], illetve Leopold [289] javasolta. Ugyancsak Leopold említett könyvéből származnak a 6.1., a 6.2., a 6.3., a 6.4. és a 6.7. ábrák.

A klasztereket Pfister [369], a grideket pedig Foster és Kesselman könyve [137], valamint hálózaton elérhető anyaga [138] alapján tárgyaljuk. A feladatok kisebb feladatokra való bontásával részletesen foglalkozik például Tanenbaum és van Steen [456].

A közös memória használatával foglalkozik Hwang és Xu[216], Kleiman és társai [255], valamint Tanenbaum és van Steen [458], Részletesen tárgyalják az üzenetküldést a Culler és társai,[97] valamint a Snir és társszerzői [435] által írt könyvek.

Amdahl, Brent és Gustafson eredményei a névadók cikkeiből származnak [16, 57, 183]. A lokalitás javításának módszereit elemzi például Kandemir, Ramanujam és Choudhary [237]. A kódtranszformáció és az adatok kapcsolatával részletesen foglalkozik Wolfe [500]. Sok hasznos ismeretet tartalmaz a kódoptimalizációról Kennedy és Allen könyve [251].

Az MPI programozási modellt Gropp [176], az OpenMP modellt pedig Chandra és társainak műve [72], valamint egy hálózaton elérhető anyag [355] alapján ismertetjük. További programozási modellek – mint a csontvázak, párhuzamos funkcionális programozás, koordinációs nyelvek és párhuzamos mobil ágensek – részletes ismertetése megtalálható Leopold [289] könyvében.

A P-szálakkal például Lewis és Berg [290], a Java-szálakkal pedig Oaks és Wong [351] foglalkoztak részletesen. A nagy teljesítményű FORTRAN leírása megtalálható Koelbel könyvében [262]. A párhuzamosító fordítóprogramokkal például Wolfe [500] foglalkozott.

A PRAM számítási modellt 1978-ban vezette be Fortune és Willye [136].

A BSP modellt Valiant [476], a LogP modellt Culler és társai [96], míg a QSM modellt Gibbons, Matias és Ramachandran javasolták [153].

A munkahatékony prefixszámítási algoritmusok elemzése Hermann és Iványi cikkén [205] alapul. A feladatokban szereplő algoritmusok többsége megtalálható az *Új algoritmusok* című tankönyvben.

7. Szisztolikus rendszerek

A szisztolikus rács – a speciális feladatot ellátó számítógépek legtökéletesebb formája – legegyszerűbb esetben csupán egyetlen számítási művelet ismételt végrehajtására alkalmas. Ennek ellenére számos, gyakorlati szempontból is jelentős alkalmazási területe van, főként az iteratív módszereket alkalmazó numerikus matematika, kombinatorikus optimizálás, lineáris algebra, algoritmikus gráfelmélet, kép- és jelfeldolgozás, nyelv- és szövegfeldolgozás stb. területén.

Egy szisztolikus rács felépítése egészen pontosan megfeleltethető egy végrehajtásra váró algoritmus szerkezetének úgy, hogy minden egyes számítás helye és időpontja egyszer és mindenkorra meghatározott, az egymással kommunikáló cellák közvetlenül és permanens módon egymáshoz vannak kapcsolva, így kommunikációs csatornák létrehozása feleslegessé válik. Az algoritmust közvetlenül hardver valósítja meg. Emiatt a szisztolikus algoritmusokat ebben az összefüggésben „hardver-algoritmusként” is szokták emlegetni.

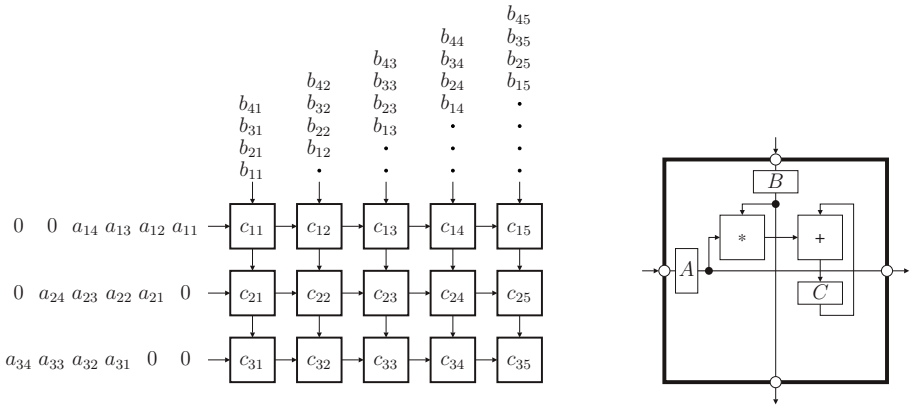
A „szisztolikus algoritmusok” kifejezés tehát nem egy bizonyos, jól körülhatárolt számítási feladatra adott megoldások sokaságát jelenti (mint például a rendezési algoritmusok esetében), hanem sokkal inkább egy sajátos feladatmeghatározási-, programozási-, illetve számítási stílust határoz meg. Igen sokféle, különböző felhasználási területhez tartozó algoritmus lehet „szisztolikus jellegű”, ami nem jelenti feltétlenül azt, hogy ezen területek összes ismert algoritmusa szisztolikus feldolgozásra alkalmas alakra hozható lenne.

Ennek a fejezetnek nem célja, hogy „a szisztolikus algoritmusokat”, vagy akár csak a „legfontosabb szisztolikus algoritmusokat” bemutassa. A cél az, hogy néhány egyszerű, de tipikus példa segítségével megalapozzuk a szisztolikus algoritmusok általános megértését.

A fejezet öt alfejezetből áll. Az alapfogalmak (7.1. alfejezet) után a téridő leképezést (7.2. alfejezet), majd a be/kiviteli sémát (7.3. alfejezet) mutatjuk be. A 7.4. alfejezetben a vezérlési szempontokat, a 7.5. alfejezetben pedig a lineáris szisztolikus rácsokat tárgyaljuk.

7.1. A szisztolika alapfogalmai

Maga a ***szisztolikus*** elnevezés a szisztolikus architektúrák működési elvéből származik. Szisztolikus munkamódon a futószalag-elv, illetve a térbeli párhuzamosság intenzív együttes alkalmazását kell értenünk, melyet egy globális, és teljes mértékben szinkronizált órajel irányít. Ez eredményezi az adatfolyamoknak az összekapcsolt cellák hálózatán keresztül történő ritmikus áramlását, az emberi szervezetben keringő vérhez hasonlóan, melyet a szív a vérereken keresztül a test különböző pontjai felé küld. A futószalag-elv nem csupán egyet-



7.1. ábra. Téglalap alakú szisztolikus rács mátrixszorzáshoz: **(a)** A rács felépítése és a beviteli séma; **(b)** cellaszerkezet.

len irányban érvényesül, hanem a különböző irányba haladó, a szisztolikus rács celláiban egymást keresztező adatfolyamok mindegyikére vonatkozik.

Egy ***szisztolikus rendszer*** általában egy *gazdagépből,* illetve a tulajdonképpeni szisztolikus rácsból áll. A gazdagépnek elvileg alárendelt szerepe van; csupán a szisztolikus rács működésbe hozatalára, továbbá adatokkal való ellátására szolgál. A ***szisztolikus rács*** egy sajátos, cellákból álló hálózatként is felfogható, mely a masszív párhuzamosságnak köszönhetően az adatorientált műveleteket igen nagy sebességgel hajtja végre. Egy szisztolikus rács celláinak egységes működése révén végrehajtott program adja magát a ***szisztolikus algoritmust.***

Bármennyire is különböznek egymástól a szisztolikus rácsok, mégis jó néhány közös tulajdonsággal rendelkeznek: ezek a diszkrét időséma, a szinkron munkamód, a szabályos (általában két dimenziós) mértani felépítés, csak közvetlenül szomszédos cellák között fennálló kommunikáció, valamint a legegyszerűbb vezérlési mechanizmusok alkalmazása.

A fejezet további részében a szisztolikus rácsokkal kapcsolatos alapvető jelenségeket fogjuk egy jellemző példán keresztül bemutatni és megvilágítani. Egy számítási feladatra gyakran többféle megoldás kínálkozik, azaz különböző szisztolikus rácsokat találhatunk, melyek közül a legjobbak általában igen bonyolultak. A továbbiakban ezért nem a legjobb változat bemutatását tűztük ki célul, hanem a legfontosabb elvek tömör és intuitív módon való bemutatását.

7.1.1. Bevezető példa: mátrixok szorzása

A 7.1. ábrán egy 15 *cellából* álló, téglalap alakú *szisztolikus rács* látható, amely egy $3 \times N$ méretű A mátrixnak egy $N \times 5$ méretű B mátrixszal való összeszorzására alkalmas. Az N paraméter a 7.1. ábrán látható szisztolikus rács *felépítése* szempontjából semmiféle szerepet nem játszik, viszont lényeges a *beviteli séma,* valamint az algoritmus *futási ideje* szempontjából.

A beviteli séma az $N = 4$ értéknek felel meg. A 7.1. ábra a következő konkrét feladat megoldását mutatja:

$$A \cdot B = C \ ,$$

ahol

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} ,$$

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \end{pmatrix} ,$$

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \end{pmatrix} ,$$

továbbá

$$c_{ij} = \sum_{k=1}^{4} a_{ik} \cdot b_{kj} \qquad (1 \le i \le 3, 1 \le j \le 5) \ .$$

A szisztolikus rács cellái adatokat cserélhetnek egymás között az őket összekötő ***kapcsolatokon*** keresztül, melyeket a 7.1(a) ábrán a cellákat összekötő nyilakkal jelöltünk. A szisztolikus rács a *peremcelláin* keresztül kommunikál a *külvilággal*. A szisztolikus rács minden egyes cellája ugyanazt a kapcsolódási mintát követi a környezettel való kommunikáció során. A szisztolikus rács teljesen szabályos felépítése (a cellák elhelyezése, illetve a kapcsolatok szerkezete) szabályos adatfolyamokat eredményez a különböző kapcsolódási tengelyek mentén.

A 7.1(b) ábra *egy cella belső szerkezetét* szemlélteti. Egy cella egy *szorzóból,* egy *összeadóból,* három *regiszterből* és négy *csatornából,* illetve az ezeket összekapcsoló vezetékekből áll. Minden cellának ugyanolyan a felépítése.

Az A, B és C regiszterek mindegyike egy szám tárolására képes. A regiszterek jelölését itt értelem szerint választottuk meg, de tulajdonképpen tetszés szerint jelölhetjük őket. Az A, illetve B regiszterek az ún. ***bemeneti csatornáktól*** kapják értékeiket. A 7.1(b) ábrán a cella bal, illetve felső szélén található apró körökkel jelöltük ezeket. Egy ilyen csatorna csatlakozási felületet képez a cellához kívülről illeszkedő kapcsolathoz.

Az A, illetve B regiszterek aktuális értékét a szorzó operandusként fogja felhasználni, ezzel egyidőben az értékek továbbadódnak a cella ***kimeneti csatornáin*** keresztül (lásd az ábra jobb, illetve alsó szélén elhelyezkedő köröket). A szorzás eredményét az összeadó veszi át, mely második paraméterét a C regisztertől kapja. Az összeadás eredménye felül fogja írni C korábbi értékét.

7.1.2. A feladat és a rács paraméterei

A 7.1. ábrán bemutatott szisztolikus rács 15 cellája 3 sorból és 5 oszlopból álló téglalap alakú mintát alkot (akárcsak a C mátrix). Ennek méretei megegyeznek az A mátrix sorainak, illetve a B mátrix oszlopainak számával. Tehát esetünkben a *szisztolikus rács mérete* a megoldásra váró feladatban szereplő *adatszerkezetek méretéhez* igazodik. Általános esetben, ha egy $N_1 \times N_3$ méretű A mátrixot egy $N_3 \times N_2$ méretű B mátrixszal kellene összeszoroznunk, egy N_1 sorral, illetve N_2 oszloppal rendelkező szisztolikus rácsra lenne szükségünk.

A 7.1. ábrán látható felépítés természetesen azt is megengedi, hogy egy több, mint N_1 sorral vagy több, mint N_2 oszloppal rendelkező szisztolikus rácsot használjunk. Ez abban az esetben lényeges, amikor egy *rögzített méretű* szisztolikus rácsot szeretnénk használni különböző méretű mátrixok összeszorzására. Ekkor minden esetben csupán azt az N_1 sorból, illetve N_2 oszlopból álló téglalap alakú részt használnánk, mely a példában a rács bal felső sarkában található. Jóllehet a többi cella is ugyanúgy fog működni, ám a feladat megoldásához nem járulnak hozzá (de hibát sem okoznak).

Az N_1, N_2, N_3 méretek a megoldásra váró feladat paramétereit képezik, mivel a megoldás során végrehajtott műveletek száma mind a három értéktől függ; ezek tehát *a feladat paraméterei*. Ellenben a szisztolikus rács mérete, illetve felépítése csak az N_1 és N_2 méretektől függ, ezek tehát egyúttal a *rács paraméterei* is lesznek.

Megjegyzés. A 7.2. alfejezetben a mátrixok szorzását megvalósító újabb szisztolikus ráccsal ismerkedhetünk meg, melynek méretei mindhárom (N_1, N_2, N_3) paramétertől függnek.

7.1.3. Térbeli koordináták

A továbbiakban a szisztolikus rács minden egyes cellájához szeretnénk egy egyértelmű ***térbeli koordinátát*** hozzárendelni, ennek segítségével jellemezve a cella mértani elhelyezkedését a rács egészéhez képest. Egy téglalap alakú szisztolikus rács esetén legegyszerűbb, ha erre a célra a megfelelő sor-, illetve oszlopszámokat használjuk. A 7.1. ábrán c_{11}-gyel jelölt cella tehát az $(1, 1)$ koordinátákat kapja, a c_{21}-gyel jelölt pedig a $(2, 1)$ koordinátákat, és így tovább. Az így meghatározott térbeli koordinátákat a fejezet hátralevő részében adottnak tekintjük.

Elvileg lényegtelen, hogy hol helyezkedik el a koordinátarendszer kezdőpontja, milyen irányban helyezkednek el a koordinátatengelyek, melyik irány felel meg az első, és melyik a második koordinátának. A példában a mátrix komponenseinek jelölésére a megadott számozást választottuk, ennek alapján az első koordináta a fentről lefelé, 1-gyel kezdődően számozott soroknak felel meg, míg a második komponens a balról jobbra, szintén 1-gyel kezdődően számozott oszlopoknak.

Természetesen másképp is megválaszthattuk volna a koordinátarendszert. A fenti séma azonban kitűnően megfelel az alábbi szisztolikus rácsnak: az egy adott cellában kiszámított c_{ij} mátrixkomponens indexei pontosan megegyeznek a cella koordinátáival. Az A mátrix beviteli adatként megadott sorainak száma pontosan ugyanaz, mint azon cellák első koordinátája, amelyeken ezek az adatok keresztülhaladnak; ugyanez érvényes a második koordinátára, a B mátrix oszlopaival kapcsolatban. Az összes kapcsolat (és ezzel együtt a rajtuk keresztülhaladó összes adatfolyam) tengelypárhuzamos és a koordináták növekvő irányába mutat.

Nem mindig ennyire egyértelmű, hogyan rendelhetünk megfelelő térbeli koordinátákat a cellákhoz; példaként álljon itt a 7.3(a) ábrán látható szisztolikus rács. A koordinátarendszer megválasztásától függetlenül fontos, hogy a cellák koordinátái szembeötlő módon tükrözzék a szisztolikus rács szabályos felépítését. Emiatt használunk tulajdonképpen mindig egész koordinátákat. Ezért jó, ha az egymástól minimális euklideszi távolságra fekvő cellák koordinátái csupán egyetlen komponensben különböznek, és a különbség értéke 1.

7.1.4. Generikus operátorok sorba fejtése

Minden, a 7.1. ábrán látható aktív (i, j) cella pontosan a C eredménymátrix c_{ij} elemét számolja ki. Tehát a cellának az alábbi *skalárszorzatot* kell kiértékelnie:

$$\sum_{k=1}^{4} a_{ik} \cdot b_{kj} \ .$$

Ez iteratív módon történik: minden lépésben egy újabb $a_{ik} \cdot b_{kj}$ szorzatot számítunk ki, ami hozzáadódik az addigi részösszeghez. A részösszeget a számítások elején természetesen nulláznunk kell (vagy egy tetszés szerinti kezdőértéket határozhatunk meg). Az imperatív programozási nyelvek klasszikus jelölésmódját követve a következőképpen írhatjuk le, hogy mi történik.

MÁTRIXSZORZÁS(N_1,N_2,N_3)

```
1  for i ← 1 to N1
2      do for j ← 1 to N2
3             do c(i, j) ← 0
4                for k ← 1 to N3
5                    do c(i, j) ← c(i, j) + a(i, k) · b(k, j)
6  return C
```

Ha $N_1 = N_2 = N_3$, akkor ennek az algoritmusnak a végrehajtása során $\Theta(N^3)$ összeadást, $\Theta(N^3)$ szorzást és $\Theta(N^3)$ értékadást kell végrehajtani, ezért egy soros processzoron a futási ideje $\Theta(N^3)$.

A $\sum$ összegző operátor az úgynevezett ***generikus operátorok*** közé tartozik, melyekhez tetszőleges számú operandus rendelhető. A 7.1. ábrán látható szisztolikus rács esetén minden egyes összeadás, mely ugyanannak az összegnek a kiszámításához tartozik, ugyanabban a cellában hajtódik végre. Ugyanakkor sok olyan példa is van, melyben egy generikus operátor egyes műveletei különböző cellák közt oszlanak meg (lásd például a 7.3. ábrán bemutatott szisztolikus rácsot).

Megjegyzés. Néhány további példa generikus operátorokra: szorzat, minimum, maximum, továbbá az ÉS, VAGY, illetve KIZÁRÓ-VAGY logikai műveletek.

Szükség van tehát a generikus operátorok ***sorba fejtésére,*** mielőtt a végrehajtandó műveleteket hozzárendelhetnénk a szisztolikus rács celláihoz. Ezeket az operátorokat azonban másképp kell kezelnünk, mint a megszokott, rögzített számú operandussal rendelkező operátorokat – ilyen például a két operandusú összeadás – mivel ezek beosztásánál bizonyos fokú szabadsággal rendelkezünk.

7.1.5. Értékadásmentes jelölés

A szisztolikus programok leírására az imperatív forma helyett, akárcsak a MÁTRIXSZORZÁS algoritmus esetében, inkább egy ***értékadásmentes jelölést*** használunk, mely egy *egyenlet megoldásán* alapszik.

Ily módon elkerüljük a mellékhatásokat és közvetlen módon ki tudjuk fejezni a párhuzamosságot. Az alábbi esetben zavaró a $c(i, j)$ változó értékének felülírása. Ezért bevezetjük helyette a $c(i, j, k)$ példányok sorozatát, mely a $c(i, j)$ változónak a MÁTRIXSZORZÁS algoritmus lépéseinek végrehajtása során felvett értékeit jelöli. Ezáltal egy úgynevezett ***rekurzív egyenlet*** jön létre. A MÁTRIXSZORZÁS algoritmusban bemutatott mátrixszorzást például a következő módon lehet értékadásmentes alakban kifejezni:

$$
\begin{aligned}
&\textit{Bemeneti műveletek}\\
&c(i, j, 0) = 0 && 1 \le i \le N_1, 1 \le j \le N_2 \ .\\
&\textit{Számítások}\\
&c(i, j, k) = c(i, j, k-1) + a(i, k) \cdot b(k, j) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 \ .\\
&\textit{Kimeneti műveletek}\\
&c_{ij} = c(i, j, N_3) && 1 \le i \le N_1, 1 \le j \le N_2 \ .
\end{aligned}
\tag{7.1}
$$

A (7.1) rendszer leírja a végrehajtott szisztolikus algoritmus pontos felépítését. A legfelső egyenlet az összes *bemeneti adatot* jellemzi, a legalsó pedig az összes *kimeneti adatot;* a szisztolikus rácsban ezek az egyenletek nem számítási, hanem ki/bemeneti műveleteknek felelnek meg. A középső egyenlet írja le a tulajdonképpeni számításokat.

A rendszer minden egyes egyenletéhez hozzátartozik egy, az egyenlet jobb oldalán látható ***kvantifikálás,*** mely az i és j iterációs változók által felvett értékek halmazát határozza meg (a középső egyenlet esetében k változóét is). Minden ilyen halmazt ***értéktartománynak*** nevezünk. A középső egyenlet i, j, k iterációs változói egy (i, j, k) ***iterációs vektort*** alkotnak. A ki/bemenet esetében az iterációs vektoroknak csupán i és j komponense van. Ahhoz, hogy egy zárt jelölésmódot kapjunk, kiegészíthetjük ezeket a vektorokat egy k komponenssel, melynek rögzített az értéke. A bemeneti adatokat $k = 0$-val jelöljük, a kimeneti adatokat pedig $k = N_3$-mal. A következőket kapjuk:

$$
\begin{aligned}
&\textit{Bemeneti műveletek}\\
&c(i, j, k) = 0 && 1 \le i \le N_1, 1 \le j \le N_2, k = 0 \ .\\
&\textit{Számítások}\\
&c(i, j, k) = c(i, j, k-1) + a(i, k) \cdot b(k, j) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 \ .\\
&\textit{Kimeneti műveletek}\\
&c_{ij} = c(i, j, k) && 1 \le i \le N_1, 1 \le j \le N_2, k = N_3 \ .
\end{aligned}
\tag{7.2}
$$

Figyeljük meg, hogy a be/kimeneti adatokhoz tartozó értéktartományok formálisan szintén háromdimenzióssá váltak ugyan, a megszokott értelemben azonban csak kétdimenziósak.

7.1.6. Elemi számítások

A (7.2) rendszer egyenleteiből közvetlen módon leolvashatók azok az elemi, azaz tovább már nem osztható műveletek, melyeket a szisztolikus rács cellái fognak elvégezni. Ezek pontosan azok – a rendszer valamely egyenletében leírt – műveletek, melyek az egyenlethez rendelt kvantifikálás egy rögzített pontjára vonatkoznak. Amennyiben egy egyenletben több részművelet fordul elő, ezeket együvé tartozónak, azaz egyetlen ***összetett műveletnek*** tekintjük, és ugyanabban a lépésben, ugyanaz a cella fogja őket együttesen végrehajtani.

A (7.2) rendszer középső egyenletében két művelet jelenik meg: az $a(i,k)\cdot b(k,j)$ szorzás, illetve a hozzá tartozó $c(i,j,k)=c(i,j,k-1)+\cdots$ összeadás. Az együvé tartozó egyes műveleteket, vagyis az összeadást és szorzást a 7.1(b) ábrán látható szisztolikus rács cellája egy összetett „szorzás-összeadás” művelet során fogja elvégezni.

Minden egyes ilyen elemi számításhoz hozzárendelhetünk egy jelet. Nyugodtan nevezhetjük ezeket is koordinátáknak. Hogy megfelelő koordinátákat kapjunk, kézenfekvő megoldásnak kínálkozik az adott értéktartományból vett (i,j,k) iterációs vektorok használata.

A fentieket alkalmazva a (7.1) rendszerre, a $c(i,j,k)=c(i,j,k-1)+a(i,k)\cdot b(k,j)$ számításhoz például az (i,j,k) koordinátahármast rendelhetjük hozzá. Ugyanígy a $c(i,j,k)=0$ bemeneti művelethez is ugyanazt az (i,j,k) koordinátahármast rendelhetjük hozzá; persze itt minden esetre érvényes, hogy $k=0$. Egyébként a példában szereplő értéktartományokat úgy választottuk meg, hogy azok diszjunkt halmazok legyenek.

Mivel a számítások, illetve a ki/bemeneti műveletek jelölésére is ugyanúgy iterációs vektorokat használunk, nincs többé szükség arra, hogy ezt a két fogalmat megkülönböztessük. Ez egyúttal azt is jelenti, hogy az értelmezési tartomány valamely pontjához tartozó műveletek ismét egy összetett műveletet képeznek – akkor is, ha ezek különböző egyenletekből származnak, és lehet, hogy semmi közük egymáshoz. A továbbiakban az egyszerűség kedvéért koordinátaként mindig iterációs vektorokat fogunk használni.

7.1.7. Diszkrét időszeletek

A szisztolikus cellák által végzett egyes elemi számítási folyamatok mindig ***diszkrét időszeletekben*** mennek végbe. Egy szisztolikus rács esetében minden ilyen időszelet ugyanolyan hosszú. Ezen felül a szisztolikus rács minden egyes cellája teljes mértékben ***szinkron*** módon működik, azaz mindegyikük egyidőben fejezi be az adott lépéshez tartozó kommunikációs, illetve a számítási műveleteket. Az egyes cellák egyes időszeletei megszakítás nélkül követik egymást.

Megjegyzés. Természetesen Albert Einstein felismerései óta tudjuk, hogy fizikailag nem tudunk igazi egyidejűséget létrehozni. A valóságban itt arról van szó, hogy a szisztolikus rács cellái időben egymáshoz igazodva dolgoznak. Technikailag ez úgy valósítható meg, hogy a cellákat egy közös ütemjel vézérli, mely a rács összes regiszterét összeköti. Az ilyen módon elért egyidejűség keretében a cellák közti kommunikáció eléggé egyidőben megy végbe ahhoz, hogy az egymáshoz kapcsolódó írási, illetve olvasási folyamat során ne vesszenek el adatok. Ennélfogva mindenképpen az a helyénvaló, ha az elméleti fejtegetések során elvi egyidejűséget tételezünk fel.

Emiatt a fizikai időt diszkrét időszeletekre oszthatjuk, folytatólagosan megszámozva az időszeleteket. Nem lényeges, hogy az időtengely kezdőpontja hol helyezkedik el, hiszen az idő múlása minden egyes cella számára szinkron módon történik. Kézenfekvő a $t=0$

időt úgy megválasztani, hogy ez éppen annak az időszeletnek feleljen meg, melynek során a legelső bemeneti adat elérkezik valamely cellához. Ezt az egyezményt használva, a (7.1) rendszer (i, j, k)-val jelölt összetett műveletének elvégzése az $i + j + k - 3$ időpontban történik. Másrészt ugyanúgy megfelelne az is, ha az (i, j, k) koordinátákat az $i + j + k$ időponthoz rendelnénk hozzá; a különbség az eddigiekhez képest csupán annyi, hogy ekkor az idő globálisan három egységgel lenne eltolódva.

Tételezzük tehát fel a továbbiakban, hogy egy bizonyos (i, j, k) számítás az $i + j + k$ időpontban megy végbe. A legelső számításra ekkor a $t = 3$ időpontban kerül sor, a legutolsó pedig a $t = N_1+N_2+N_3$ időpontban. A ***teljes végrehajtási idő*** tehát $N_1+N_2+N_3-2$ időszeletet tesz ki.

7.1.8. Külső és belső kommunikáció

Szisztolikus rácsok esetén a számításokhoz szükséges adatok a számítás kezdetén általában még nincsenek a rács celláiban. Ezeket a rácsba a ***külvilágból*** kell betölteni. A külvilág egy központi *memóriához* való hozzáféréssel rendelkező *gazdagépből* áll, melyet egy vezérlőegység vezérel. A vezérlőegység a megfelelő időpontban kiolvassa a memóriából a szükséges adatokat, megfelelő módon továbbítja azokat a szisztolikus rácsnak, illetve a kiszámolt eredményeket visszaírja a memóriába.

A k-nak megfelelő időszeletben az a_{ik} és b_{kj} operandusoknak minden egyes (i, j) cella rendelkezésére kell állniuk. De a 7.1. ábrán látható szisztolikus rács esetén mindössze az első oszlop cellái kapják az A mátrix elemeit közvetlenül a külvilágtól bemeneti adatként. Az összes többi cella arra van utalva, hogy a szükséges a_{ik} értékeket egy szomszédos cellától kapja meg. Ez, ahogy a 7.1(a) ábrán is látható, a szomszédos cellák között levő vízszintes *kapcsolatokon* keresztül történik. Az a_{ik} érték rendre keresztülhalad az $(i, 1)$, $(i, 2)$, $\ldots$, (i, N_2) cellákon. Ennek megfelelően a b_{kj} érték az $(1, j)$ cellán keresztül kerül a rácsba, onnan pedig a függőleges összeköttetéseken keresztül halad tovább a $(2, j)$, $(3, j)$, $\ldots$ cellákon át, egészen az (N_1, j) celláig. Az ábrán látható nyilak hegye azt mutatja, hogy egy ilyen kapcsolat milyen *irányban* működik.

Az osztott/párhuzamos architektúrák esetén gyakorta gondot okoz, hogy egy értéket egy időegység alatt nagy távolságra továbbítsunk. Ebből következik, hogy a mi esetünkben az a_{ik} érték továbbítása az (i, j) cellától az $(i, j + 1)$ cella felé nem ugyanazon az időszeleten belül történik, melyben az (i, j) cella ezt az értéket az $(i, j - 1)$ cellától vagy a külvilágtól átvette, hanem egy időszelettel később. Ugyanez érvényes a b_{kj} érték továbbítására is. Ez a ***késleltetés*** a részletes cellaszerkezetet bemutató 7.1(b) ábrán világosan látható: minden bemeneti adattól kiinduló útvonal, mely egy cellán vezet keresztül, áthalad egy regiszteren, és minden egyes regiszter pontosan egy időszeletnyi késleltetést eredményez.

Megjegyzés. Szisztolikus architektúrák esetében általában elő van írva, hogy a különböző cellákat összekötő útvonalak mindig legalább egy regiszteren keresztül vezessenek – akkor is, ha csupán szomszédos cellák közti adatátvitelről van szó. A szisztolikus rács globális ütemjele összekapcsolja a cellák regisztereit. Mindez a szisztolikus rács kapcsolatain keresztüláramló, jellemzően ritmikus adatforgalomhoz vezet. A „szisztolé” (magyarul szívösszehúzódás) orvosi kifejezés pontosan emiatt, a lüktető vérerekkel szemben fennálló képi rokonság okán vált e fogalom névadójává.

Hogy az adatoknak ezt a késleltetett továbbadását szemléltessük, tovább bővítjük a (7.1) rendszert úgy, hogy a többszörösen olvasott értékek – mint például az a_{ik} – számára külön-

böző példányokat vezetünk be (ez egy, a szisztolikus algoritmusok tervezésére alkalmas tipikus eljárásmód):

Bemeneti műveletek

$$
\begin{aligned}
&a(i,j,k) = a_{ik} && 1 \le i \le N_1, j = 0, 1 \le k \le N_3 ,\\
&b(i,j,k) = b_{kj} && i = 0, 1 \le j \le N_2, 1 \le k \le N_3 ,\\
&c(i,j,k) = 0 && 1 \le i \le N_1, 1 \le j \le N_2, k = 0 .
\end{aligned}
$$

Számítások

$$
\begin{aligned}
&a(i,j,k) = a(i,j-1,k) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 ,\\
&b(i,j,k) = b(i-1,j,k) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 ,\\
&c(i,j,k) = c(i,j,k-1) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3\\
&\quad + a(i,j-1,k) \cdot b(i-1,j,k) .
\end{aligned}
\tag{7.3}
$$

Kimeneti műveletek

$$
c_{ij} = c(i,j,k) \qquad 1 \le i \le N_1, 1 \le j \le N_2, k = N_3 .
$$

A c_{ij} kiszámítását célzó minden egyes $c(i,j,k)$ részösszeg egy bizonyos időszelet során számítódik ki, és azt csupán egyetlen alkalommal, ti. a következő időszeletben, használjuk. Az (i,j) cellának tehát szüksége van egy regiszterre (a 7.1(b) ábrán ez a C), amely megőrzi a $c(i,j,k)$ értéket egy időszelet erejéig. A $c(i,j,k)$ értékre a továbbiakban nem lesz szükség, ezért a megfelelő regiszter tartalma felülírható a $c(i,j,k+1)$ értékkel. A skalárszorzat kiszámításának végeztével a regiszterben a $c(i,j,N_3)$ érték, azaz a kiszámításra váró c_{ij} eredmény marad meg. A számítások kezdetén a regisztert nulláznunk kell (vagy egy tetszőleges kezdőértéket adhatunk neki).

Az a_{ik} és b_{kj} értékeket ezzel szemben nem szükséges az (i,j) cellában tárolnunk. Amint a 7.1(a) ábrán vázolt adatbeviteli sémán látható, az A mátrix minden sorának bevitele egy időegységgel késleltetve van az előzőhöz képest. Ugyanúgy, a B mátrix minden egyes oszlopa egy időegységgel késleltetve van az előzőhöz képest. Így az $a(i,j-1,k)$ és $b(i-1,j,k)$ értékek pontosan abban az időszeletben érkeznek az (i,j) cellához, amikor ott a $c(i,j,k)$ érték kiszámítása történik, értékük az A, illetve B regiszterbe íródik, innen viszont azonnal felhasználjuk őket a szorzás elvégzésére, illetve értékük továbbadódik a szomszédos celláknak. Amint az (i,j) cellában megtörtént az a_{ik} és b_{kj} értékek összeszorzása, ebben a cellában már nincs szükség az értékükre. Emiatt nem fontos az értéküket a cellában tovább őrizni, így az A és B a következő időszelet során új értéket fog kapni.

Ezekből a fejtegetésekből máris kiderül, hogy ahhoz, hogy egy cella tárolókapacitását a minimálisra csökkentsük, ügyelnünk kell arra, hogy a számítási, illetve kommunikációs folyamatokat úgy irányítsuk térben és időben, hogy az azonnali felhasználás, illetve továbbadás révén az adatokat a lehető legrövidebb ideig kelljen csupán tárolni. A szisztolikus rács általános felépítésén túl ezt lényegében azzal érhetjük el, hogy egy megfelelő ki/beviteli sémát választunk, illetve megfelelő módon állapítjuk meg a cellákon belüli késleltetési időket.

A példában látható ***ki/beviteli séma*** geometriai elrendezése az A és B mátrixok *szétdarabolásának* eredményeként született. A bemeneti adatfolyamokban ezáltal szabaddá vált helyeket null-értékekkel töltjük fel, különben elrontanánk a c_{ij} értékek kiszámítását. A bemeneti adatfolyamok hossza a feladat N_3 paraméterétől függ.

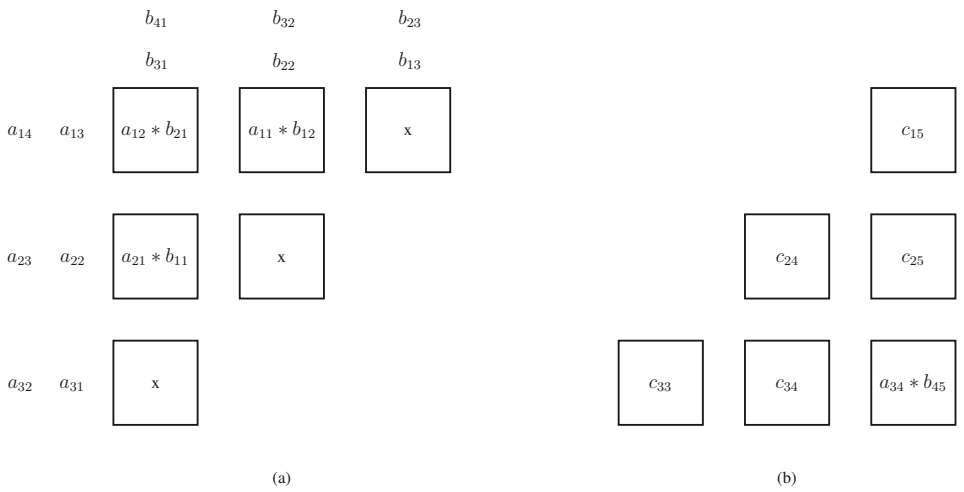
A C mátrix elemeinek kiszámítása a 7.1. ábra alapján ***stacionárius módon*** történik, ami azt jelenti, hogy valamely mátrixelem végleges értéke kiszámításának lépései ugyanabban a cellában mennek végbe. A ***stacionárius változók*** egyáltalán nem mozdulnak a számítások során a szisztolikus rácson belül. Az eredményeket végül továbbítanunk kell a rács peremén levő cellákhoz, mivel csak ezeken keresztül tudjuk átadni őket a külvilágnak. Ráadásul figyelembe kell vennünk azt is, hogy a c_{ij} kiszámítására szolgáló regisztereket kezdőértékkel kell ellátnunk. E két különleges kiegészítő feladat megoldása elég nagy időbeli, illetve anyagi ráfordítást igényel, de ezt a 7.4. alfejezetben még pontosabban megvizsgáljuk majd.

7.1.9. Futószalag-elvű feldolgozás

A jellemző módon diszkrét, egyforma hosszúságú, globálisan szinkronizált időszeletekkel való munkamódnak, illetve a cellák egymástól regiszterek segítségével történő szigorú időbeli elszigeteltségének köszönhetően, a szisztolikus rácsokat a *futószalag-elvű feldolgozás* egy sajátos alkalmazásának tekinthetjük. A cellák regiszterei ez esetben a jól ismert *futószalag-regisztereknek* felelnek meg. A klasszikus értelemben vett futószalag kétségkívül mindig lineáris felépítésű, míg a szisztolikus rácsok szerkezete (amint az a példából is látszik) gyakorta kétdimenziós. Egy *többdimenziós szisztolikus rácsot* felfoghatunk úgy, mint ami több, egymásba fonódó futószalag szövedékéből áll. Természetesen egyértelmű, hogy az egydimenziós futószalagelvű feldolgozásra érvényes alapvető sajátosságok a többdimenziós szisztolikus rácsoknál ugyanúgy fellelhetőek.

A futószalagelvű feldolgozás egyik tipikus velejárója, hogy a ***hatékonyság*** kisebb a számítások kezdetén, illetve végén. Eleinte a futószalag üres, egyetlen fokozata sem dolgozik. Ezt követően csupán az első fokozat rendelkezik adatokkal és ez végez számításokat; az összes többi fokozatnak továbbra sincs semmi tennivalója. A következő időszeletben az első fokozat adatokat ad át a következőnek, ugyanakkor maga is újabb adatokat kap; csak ez a két fokozat dolgozik. Ez így megy tovább mindaddig, amíg végül minden fokozat megtelik adatokkal, és a futószalag dolgozza fel ezeket, vagyis első ízben állíthatjuk a futószalagról, hogy teljes hatékonysággal dolgozik. Néhány ilyen teljes telítettségű lépés után, melynek időtartama az adatfolyamok méretétől függ, egyszer csak elfogy a bemeneti adat, a futószalag első fokozata tehát abbahagyja a működést. A következő időszeletben a második fokozat is abbahagyja a munkát, majd ugyanígy tovább, míg legvégül egyetlen fokozat sem dolgozik már. A kezdő, illetve végső munkaszakasz csökkenti a teljes futószalag átlagteljesítményét, és ennek a teljesítménycsökkenésnek a viszonylagos értéke annál nagyobb, minél több fokozata van a futószalagnek az adatfolyamok hosszához képest.

Ugyanezt a jelenséget a maga sajátos mivoltában kitűnően vizsgálhatjuk a 7.1. ábrán bemutatott szisztolikus rács esetében. Itt is találunk a számítások kezdetén, illetve végén szinte tétlenül álló cellákat. Az első időszeletben csupán az $(1, 1)$ cella végez hasznos munkát; az összes többi cella is dolgozik ugyan, de csak üresjáratban. A második lépésben ehhez hozzáadódnak még az $(1, 2)$ és $(2, 1)$ cellák; ezt az időszeletet szemlélteti a 7.2(a) ábra. Mindez ugyanígy folytatódik, míg végül az (N_1, N_2) cella is hozzá nem járul a számításokhoz. Amint a legutolsó tényleges adat elhagyta az $(1, 1)$ cellát, ez többé nem járul hozzá a számításokhoz, csupán a már kiszámolt c_{11} értéket fogja újra és újra reprodukálni. Lépésről lépésre egyre több cella hagyja abba az aktív tevékenységet, míg legvégül már csak az (N_1, N_2) cella végzi el az utolsó fontos számítást; a 7.2(b) ábra szemlélteti ezt a végső időszeletet.



7.2. ábra. Két kiragadott helyzetkép a 7.1. ábrához (részletek).

Megjegyezzük, hogy a képletekben a szorzás jelölésére pontot használunk, az ábrákon azonban csillagot.

Gyakorlatok

7.1-1. Hogyan kellene módosítani a 7.1(a) ábrán bemutatott beviteli adatsémát, ha ugyanazon szisztolikus rács segítségével egy (2×6)-os és egy (6×3)-as mátrixot szeretnénk összeszorozni? Átszervezhetők-e a számítások oly módon, hogy az eredménymátrix a szisztolikus rács jobb alsó sarkába kerüljön?

7.1-2. Miért fontos a 7.1. ábra szerinti mátrixszorzás szempontjából, hogy a beviteli folyamokban a nem használt helyekre 0 értékeket helyezzünk? Miért nem lényeges ugyanez a B mátrix esetében?

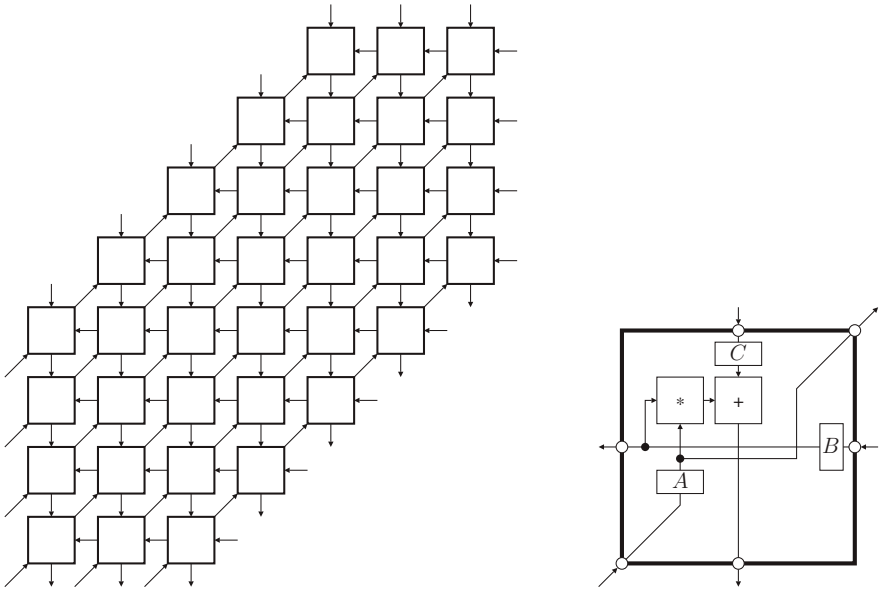
7.1-3. Ha a 7.1. ábrán látható szisztolikus rácsot futószalagként kellene értelmezzük, hány fokozatra lenne szükségünk ahhoz, hogy megfelelőképpen tudjuk leírni ennek viselkedését?

7.2. Tér-idő-leképezés és szisztolikus rács

Az előző alfejezetben leírt bevezető elegendő ugyan egy egyszerű fogalom megalkotásához, de akkor már nem elég, ha a szisztolikus rács tulajdonságait mennyiségileg pontosan szeretnénk megragadni és értékelni. Különösen a *paraméteres feladat-meghatározás* bevezetése igényel matematikai segédeszközöket. Ezért ez az alfejezet az *egyenletes* szisztolikus algoritmusok *lineáris leképezésekre* alapozó elméletének központi kérdéseivel foglalkozik.

7.2.1. Példa: mátrixszorzás stacionárius változók nélkül

A (7.3) rendszer nem csak a 7.1. ábrán bemutatott szisztolikus rács segítségével számolható ki, hanem sok más szisztolikus ráccsal is. A 7.3. ábra példa egy ilyen szisztolikus rácsra.



7.3. ábra. Hexagonális szisztolikus rács mátrixszorzáshoz: **(a)** a rács felépítése és az adatok bevitelének/kivitelének elve; **(b)** cellaszerkezet.

Bár ugyanazt a függvényt értékeli ki, mint a 7.1. ábrán látható rendszer, a képi megjelenése teljesen más:

- a cellák száma itt lényegesen nagyobb, 15 helyett összesen 36;
- a rács körvonala hexagonális, téglalap alakú helyett;
- minden cellának három bemenete és három kimenete van;
- az adatok bevitele lényegesen másképp történik, mint a 7.1(a). ábrán;
- végül: itt a C mátrix elemei is végighaladnak a rácson.

A 7.3(b) ábrán látható cellaszerkezet első látásra nem tűnik lényegesen különbözőnek a 7.1(b) ábrához képest. A különbségek azért mégis jelentősek: az új cellában nincs *ciklikus út*, így *stacionárius változók* itt nem jelennek meg. Ehelyett a cellának három bemenő, illetve három kimenő csatornája van, melyeken keresztül a három mátrix elemei haladnak. A csatornákon keresztül történő kommunikáció iránya a cella jobb és bal oldalán megváltozott, a mátrixok csatornákhoz való hozzárendelése úgyszintén.

7.2.2. A tér-idő leképezés, mint globális szemléletmód

Hogyan függ tehát össze a (7.3) rendszer és a 7.3. ábra? A 7.1. fejezetben bemutatott szisztolikus rács működését minden segítség nélkül jól tudtuk követni. Az alábbi példa esetén ez azonban lényegesen nehezebb – így sokkal inkább indíttatást érzünk egy matematikai segédeszköz használatára.

Ahhoz, hogy le tudjuk írni a szisztolikus rácson belül végzett műveleteket, minden ilyen művelethez két alapvető mértéket rendelhetünk hozzá: az időszeletet, melyben a művelet végrehajtásra kerül, illetve a cellát, amely a műveletet végrehajtja. Amint ez a későbbiekben még inkább nyilvánvalóvá lesz, a ***tér-idő-leképezés*** megválasztásával tulajdonképpen ki is merítettük a tervezéssel kapcsolatos szabadságunkat; szinte minden további elem kényszerű módon következik a tér-idő-leképezésből.

Akárcsak a 7.1. ábra szisztolikus rácsa esetében, a 7.3. ábrán látható szisztolikus rácsban is a $t = i+j+k$ időpontban történik az (i, j, k) elemhez kapcsolódó számítások elvégzése. Ugyanezt egy

$$\pi = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \tag{7.4}$$

idővektor és egy $v = (i, j, k)$ iterációs vektor skalárszorzataként is leírhatjuk,

$$t = \pi \cdot v \,, \tag{7.5}$$

esetünkben tehát

$$t = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \end{pmatrix} = i + j + k \,. \tag{7.6}$$

A 7.1. ábrán látható példában végrehajtott műveletek $z = (x, y)$ térkoordinátáit, a 7.1.3. pontban leírt egyezményünk alapján a $v = (i, j, k)$ iterációs vektorokból következtethetjük ki. Az itt megválasztott leképezés az $\mathbb{R}^3$ tér k tengely menti *projekcióját* végzi el. Ez a lineáris leképezés egy

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{7.7}$$

projekciós mátrix segítségével írható le. A térkoordinátákat úgy kapjuk, hogy a projekciós mátrixot megszorozzuk a v iterációs vektorral, amit az alábbi módon írhatunk le:

$$z = P \cdot v \,. \tag{7.8}$$

A ***projekció irányát*** az a vektor adja, amely ortogonális a projekciós mátrix minden sorára nézve:

$$P \cdot u = \vec{0}. \tag{7.9}$$

A (7.7)-ben szereplő P projekciós mátrix számára például $u = (0, 0, 1)$ egy lehetséges ***projekciós vektor.***

Szisztolikus rácsok tervezésénél gyakran alkalmaznak projekciókat a térkoordináták meghatározására. A 7.3(a) ábrán látható példánkban is az iterációs vektorokra alkalmazott projekció útján kapjuk a térkoordinátákat. Tekintsük adottnak az alábbi projekciós mátrixot:

$$P = \begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \end{pmatrix} . \tag{7.10}$$

Egy hozzá tartozó lehetséges projekciós vektor $u = (1, 1, 1)$.

A projekciós mátrixot és az idővektort összegezhetjük egy T mátrix segítségével, mely magát az úgynevezett ***tér-idő-leképezést*** írja le:

$$\begin{pmatrix} z \\ t \end{pmatrix} = \begin{pmatrix} P \\ \pi \end{pmatrix} \cdot v = T \cdot v \,. \tag{7.11}$$

T első két sorát a P projekciós mátrix adja, a harmadikat pedig a π idővektor.

A 7.1. ábrán szereplő példa esetén a tér-idő-leképezés T mátrixa a következő:

$$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}, \tag{7.12}$$

a 7.3. ábrán szereplő példa esetén pedig

$$T = \begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}. \tag{7.13}$$

A tér-idő-leképezést a szisztolikus rácsra vonatkozó globális szemléletmódként is felfoghatjuk. Amennyiben egy (esetünkben lineáris, T-vel jelölt) tér-idő-leképezést alkalmazunk egy rekurzív egyenletrendszerre, máris láthatóvá válnak a szisztolikus rács külső ismérvei, azaz a felépítése (térkoordináták, kapcsolatrendszer, cellaszerkezet).

Megjegyzés. Tisztán lineáris leképezések helyett affin leképezések is szóba jöhetnek, melyeknek egy vektor-komponensük is van, például $T \cdot v + h$. Affin leképezésekre viszont nincs feltétlenül szükségünk, amíg minden iterációs vektort ugyanazzal a tér-idő-leképezéssel kezelünk.

7.2.3. A térkoordináták szimbolikus meghatározása

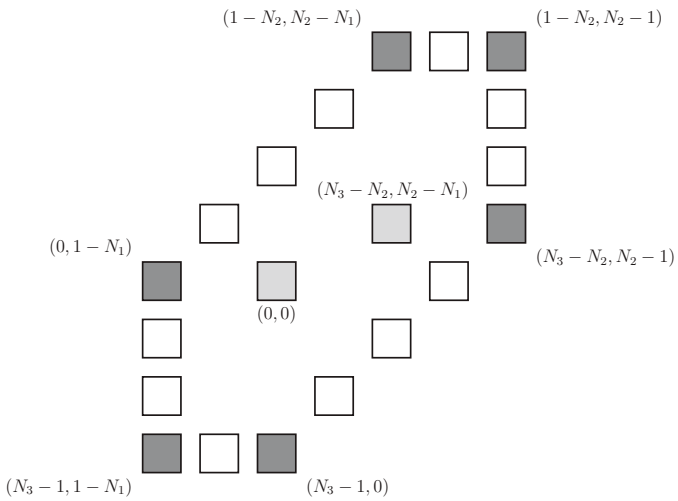
Amikor az értéktartományok számszerűen adottak, és eléggé kicsik, a (7.11) összefüggés segítségével könnyűszerrel kiszámíthatjuk a térkoordináták konkrét halmazát. Amennyiben az értéktartományok paraméteresek, mint a (7.3) rendszer esetében, a cellák helyzetét *szimbolikusan* kell meghatároznunk. Az alábbiakban leírtak e feladat megoldását célozzák.

Minden egyes cellát egy $z = (x, y)$ térkoordinátájú pontnak tekintünk az $\mathbb{R}^2$ kétdimenziós térben. Az S értéktartomány minden egyes iterációs vektorából a (7.8) kifejezés segítségével egy-egy processzor (cella) z térkoordinátáit kapjuk, $z = P \cdot v$, így a v-vel jelölt művelet a z cellára lesz rávetítve. Az így kapott térkoordináták halmaza $P(S) = \{P \cdot v : v \in S\}$ adja meg a szisztolikus rács működése szempontjából fontos összes cella helyzetét.

Általában olyan értéktartományok fordulnak elő, melyek egy konvex terület (itt $\mathbb{R}^3$-ból) összes egész koordinátájú pontjának halmazaként jeleníthetők meg ***(sűrű konvex értéktartományok).*** Egy ilyen (véges számosságú) értéktartomány konvex burka egy *politóp*, melynek csúcsai az értéktartomány pontjai. A politópokat tetszőleges lineáris leképzés újabb politóppá alakítja. Kihasználhatjuk tehát, hogy minden projekció lineáris leképezés. Az újonnan keletkező politóp csúcsai az eredeti politóp csúcsainak projekciói.

Megjegyzés. Nem szükséges, hogy a projekció során az eredeti politóp minden csúcsa az új politópnak is csúcsa legyen. Lásd például a 7.4. ábrát.

A $\mathbb{Z}^3$ rács projekciója egy egész számú P projekciós mátrix által a $\mathbb{Z}^2$ rácshoz vezet, amennyiben a P-t egy π egész számú idővektor megválasztásával egy *unimoduláris T mátrixra* egészítjük ki. Ez egy sűrű konvex értéktartományt (néhány, az alkalmazás szempontjából lényegtelen kivételtől eltekintve) a koordináták sűrű konvex halmazává alakítja, melyet az ezt burkoló politóp csúcsai teljes mértékben jellemeznek.



7.4. ábra. Egy téglalap alakú értéktartomány projekciójának eredménye.

Megjegyzés. Egy mátrixot ***unimodulárisnak*** nevezünk, amennyiben négyzetes, csak egész számú bemenetei vannak és a determinánsa ± 1. Az unimoduláris mátrixok invertálhatók, és inverzük szintén unimoduláris.

Alkalmazzuk ezt a módszert a (7.3) rendszer

$$S = [1, N_1] \times [1, N_2] \times [1, N_3] \tag{7.14}$$

egész számokat tartalmazó értéktartományára. A konvex burok csúcsai esetünkben

$$\begin{aligned} &(1,1,1), (N_1,1,1), (1,N_2,1), (1,1,N_3) , \\ &(1,N_2,N_3), (N_1,1,N_3), (N_1,N_2,1), (N_1,N_2,N_3) . \end{aligned} \tag{7.15}$$

A (7.10)-beli P projekciós mátrix esetén a projekció csúcsainak helyzete

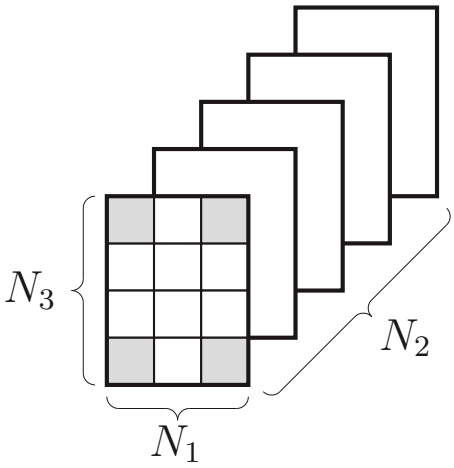
$$\begin{aligned} &(N_3-1,0), (N_3-1,1-N_1), (0,1-N_1) , \\ &(1-N_2,N_2-N_1), (1-N_2,N_2-1), (N_3-N_2,N_2-N_1) . \end{aligned} \tag{7.16}$$

Mivel S-nek nyolc csúcsa van, a $P(S)$ képének viszont csupán hat, világos, hogy az S két csúcsa a terület belső részébe fog vetítődni, tehát a mértékek meghatározásában nem játszik szerepet; ezek az $(1,1,1)$ és az (N_1,N_2,N_3) csúcsok.

Konkrétan $N_1 = 3$, $N_2 = 5$ és $N_3 = 4$ esetében a $(3,0)$, $(3,-2)$, $(0,-2)$, $(-4,2)$, $(-4,4)$ és $(-1,4)$ csúcsok adódnak. Láthatjuk, hogy nem kell minden térkoordinátának feltétlenül pozitívnak lennie. A koordinátarendszer kezdőpontjának megválasztása, mely esetünkben a politóp belsejében helyezkedik el, szintén nem nyilvánvaló.

A projekció eredményeként egy hatszöget kapunk, melynek szembefekvő oldalai párhuzamosak. Ennek peremén mindig N_1, N_2 vagy N_3 egész számú pont helyezkedik el. Az első példával ellentétben itt tehát a feladat minden paramétere egyúttal a rács paramétere is. Ennek a szisztolikus rácsnak a körvonalát ***hexagonálisnak*** nevezzük.

Ennek a tartománynak az F felülete $\Theta(N_1N_2 + N_1N_3 + N_2N_3)$, mely egyformán függ



7.5. ábra. A térkoordináták halmazának részekre bontása.

mindhárom mátrixmérettől. Itt tehát egészen más helyzettel találkozunk, mint a 7.1(a) ábrán, ahol (ugyanennek a feladatnak!) a bonyolultsága csupán $\Theta(N_1N_2)$ volt.

Ezután a hozzávetőleges számolás után most megszámoljuk egész pontosan a cellákat. Ehhez a számoláshoz célszerű az egész tartományt résztartományokra bontani, melyekben a cellák száma könnyűszerrel meghatározható (lásd 7.5. ábra). A $(0,0)$, $(N_3-1,0)$, $(N_3-1,1-N_1)$ és $(0,1-N_1)$ pontok egy N_1N_3 cellát tartalmazó téglalapot határolnak körül. Ha eltoljuk ezt a ponthalmazt N_2-1 cellával feljebb és N_2-1 cellával jobbra, végigjárjuk ezzel a teljes tartományt. Minden alkalommal azonban, amikor a téglalapot egy cellával feljebb és jobbra toljuk, csupán N_1+N_3-1 új cella jön hozzá az eddigiekhez. Ez összesen $N_1N_3+(N_2-1)(N_1+N_3-1)=N_1N_2+N_1N_3+N_2N_3-(N_1+N_2+N_3)+1$ cellát tesz ki.

Például $N_1=3$, $N_2=5$ és $N_3=4$ esetében 36 cellát kapunk, amint az a 7.3(a) ábrán látszik is.

7.2.4. A teljes végrehajtási idő szimbolikus kiszámítása

Egy szisztolikus algoritmus *teljes végrehajtási idejének* szimbolikus kiszámítása a 7.2.3. pontban leírtakhoz hasonló módon történik. Az időleképezés a (7.5) összefüggés alapján szintén lineáris leképezés. Az első, illetve utolsó számításnak megfelelő időszeletet a számítási időpontok $\pi(S)=\{\pi\cdot v : v\in S\}$ halmazának minimuma, illetve maximuma adja. A fentebb leírtak alapján elegendő, ha az S konvex burkának v csúcsait vesszük figyelembe.

A *teljes végrehajtási időt* a következő képlet alapján számíthatjuk ki:

$$t_\Sigma = 1+\max P(S)-\min P(S)\,. \tag{7.17}$$

Az 1 hozzáadása mindenképpen fontos, mert a legelső és a legutolsó számítás időpontja is számít.

A 7.3. ábrán látható példa esetében a (7.6) összefüggést alkalmazva a (7.15)-ben kiszámolt politópcsúcsokra, a következő képek halmazát kapjuk: $\{3, 2+N_1, 2+N_2, 2+N_3, 1+N_1+N_2, 1+N_1+N_3, 1+N_2+N_3, N_1+N_2+N_3\}$. Az alapfeltevésből, mely szerint $N_1, N_2, N_3 \geq 1$ következik, hogy a minimum 3, a maximum pedig $N_1+N_2+N_3$, a teljes végrehajtási idő pedig $N_1+N_2+N_3-2$ időegységet tesz ki (akárcsak a 7.1. ábrán látható szisztolikus rács esetén – elvégre az értéktartomány és az idővektor megegyezik a két példában).

A feladat paramétereinek az $N_1 = 3$, $N_2 = 5$ és $N_3 = 4$ konkrét értékeket adva a kiszámolt teljes végrehajtási idő $12-3+1 = 10$ időszeletet tesz ki.

Ha $N_1 = N_2 = N_3$, akkor a szisztolikus algoritmus egy $\Theta(N^2)$ cellát tartalmazó rendszerrel $\Theta(N)$ idő alatt határozza meg két, $N \times N$ méretű mátrix szorzatát.

7.2.5. A kapcsolatszerkezet levezetése

A szisztolikus rács ***kapcsolatszerkezetét*** úgy kapjuk, hogy a tér-idő-leképezést alkalmazzuk a feladat *adatfüggőségeire*. Minden egyes adatfüggőség abból adódik, hogy egy változó bizonyos példányát közvetlen módon felhasználjuk ugyanazon vagy egy másik változó egy példányának kiszámolására.

Megjegyzés. A szisztolikus rácsoknál – az imperatív programnyelven megírt programok párhuzamos párhuzamos végrehajtásánál szükséges adatfüggőség vizsgálatok helyett, melyeket vagy a párhuzamosan optimalizáló fordító és/vagy a processzor végez el – csupán folyamfüggőségekről van szó. Ez az általunk használt hozzárendelésmentes jelölés következménye.

Az adatfüggőségeket úgy tudjuk leolvasni, hogy az értékadásmentes jelölés kvantifikált egyenletének egyszerre szemléljük a jobb és bal oldalát. Mindenekelőtt a (7.3) rendszer $c(i,j,k) = c(i,j,k-1) + a(i,j-1,k) * b(i-1,j,k)$ egyenletét vizsgáljuk.

A $c(i,j,k)$ érték kiszámítása a $c(i,j,k-1)$, $a(i,j-1,k)$ és $b(i-1,j,k)$ értékek segítségével történik. Van tehát egy ***adatfolyamunk*** $c(i,j,k-1)$-től $c(i,j,k)$ irányában, egy adatfolyam $a(i,j-1,k)$-tól $c(i,j,k)$ irányában és egy adatfolyam $b(i-1,j,k)$-tól $c(i,j,k)$ felé.

Egy ilyen adatfolyam számunkra fontos sajátosságait egy ***függőségi vektor*** segítségével írhatjuk le. Ezt a kiszámolt változópéldány iterációs vektora, illetve az ennek kiszámításához éppen használt változópéldány iterációs vektora közti különbségvektor adja.

A $c(i,j,k)$ iterációs vektora (i,j,k), a $c(i,j,k-1)$-é pedig $(i,j,k-1)$. Az ebből kapott különbségvektor pedig

$$d_C = \begin{pmatrix} i \\ j \\ k \end{pmatrix} - \begin{pmatrix} i \\ j \\ k-1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{7.18}$$

Ugyanúgy, megfelelő módon kapjuk:

$$d_A = \begin{pmatrix} i \\ j \\ k \end{pmatrix} - \begin{pmatrix} i \\ j-1 \\ k \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \tag{7.19}$$

és

$$d_B = \begin{pmatrix} i \\ j \\ k \end{pmatrix} - \begin{pmatrix} i-1 \\ j \\ k \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}. \tag{7.20}$$

A (7.3) rendszer $a(i,j,k) = a(i,j-1,k)$ egyenletében közvetlenül felismerhető, hogy melyik a kiszámolt változópéldány, és melyik a kiszámításához szükséges változópéldány. Itt láthatjuk tehát a különbséget egyenlet és értékadás között. Tekintve, hogy $a(i,j,k)$-t $a(i,j-1,k)$-ból másolás révén kapjuk, ugyanazt a függőségi vektort kapjuk, mint a (7.19) kifejezésben. Ugyanez érvényes a $b(i,j,k) = b(i-1,j,k)$ egyenletre is.

Amennyiben egy változópéldány a v iterációs vektorral rendelkezik, azt a $P \cdot v$ cellában számítjuk ki. Ha ennek kiszámításához egy másik, v' iterációs vektorral rendelkező változópéldányra is szükség van, akkor ez a $P \cdot v'$ cellában számítódott ki és $d = v - v'$ függőségi vektor jellemzi a köztük fennálló adatfüggőséget. Ez a $z' = P \cdot v'$ cellától a $z = P \cdot v$ cella felé történő kommunikációt feltételez. Szisztolikus rácsok esetében ezt a kommunikációt a kommunikáló cellák közti közvetlen, statikus kapcsolat biztosításával oldják meg. A (7.8) leképzés lineáris voltából következik, hogy $z - z' = P \cdot v - P \cdot v' = P \cdot (v - v') = P \cdot d$.

Amennyiben $P \cdot d = \vec{0}$, a kommunikáció a számítást végző cellán belül történik, ami azt jelenti, hogy az csak időben, de nem térben zajlik. Az értékek továbbadása időben a számítást végző cella regiszterein keresztül valósul meg.

Ha viszont $P \cdot d \neq \vec{0}$, akkor két különböző cella közötti kommunikációra lesz szükség. Ekkor a szisztolikus rács összes cellájához hozzá kell rendelni egy $P \cdot d$ irányú kapcsolatot. Meghúzzuk a $P \cdot d$ vektort a számítást végző cella z térkoordinátájától a vektor irányításával ellentétesen haladva, és a z-t ellátó, z' térkoordinátájú cellához jutunk.

Ha több ilyen d függőségi vektorunk van, mindegyikhez tartozik egyegy neki megfelelő kapcsolat. Tekintsük például a (7.18), (7.19), (7.20) és (7.10) összefüggéseket. A következők állnak fenn: $P \cdot d_A = (-1,1)$, $P \cdot d_B = (0,-1)$ és $P \cdot d_C = (1,0)$. A három $P \cdot d$ vektornak megfelelő kapcsolatokat a 7.3(a) ábrán minden egyes cellánál megfigyelhetjük. Ez egy hexagonális kapcsolatszerkezetet eredményez (az eddig ismert ortogonális helyett).

7.2.6. A cellaszerkezet meghatározása

Most átvisszük a 7.2.5. pontban tárgyalt, térrel kapcsolatos fejtegetéseket az időbeli összefüggésekre. Egy v iterációs vektorral rendelkező változópéldány kiszámítására a $\pi \cdot v$ időszeletben kerül sor. Amennyiben ennek kiszámításához egy másik, v' iterációs vektorral rendelkező változópéldány szükséges, akkor ez utóbbi értéke a $\pi \cdot v'$ időszeletben került kiszámításra. A $d = v - v'$ függőségi vektornak megfelelő kommunikáció tehát pontosan $\pi \cdot v - \pi \cdot v'$ időszeletet vesz igénybe.

Mivel a (7.6) leképezés lineáris, fennáll $\pi \cdot v - \pi \cdot v' = \pi \cdot (v - v') = \pi \cdot d$. Mivel a szisztolikus alapelv szerint minden egyes kommunikáció legalább egy regiszteren vezet keresztül, a d függőségi vektorok pedig rögzítettek, az idővektor megválasztását a

$$\pi \cdot d \geq 1 \tag{7.21}$$

feltétel korlátozza.

$P \cdot d = \vec{0}$ esetén az éppen szükséges érték tárolása céljából egy ***regiszterre*** van szükség minden egyes cellában. Mivel egy ilyen regiszter tartalma a következő időszeletben máris felülíródik egy újabb értékkel, a régi értéket egy további regiszterbe kell átmentenünk, amennyiben $\pi \cdot d \geq 2$ fennáll. Mivel mindez $\pi \cdot d$ időszeleten belül megismétlődik minden egyes tárolt érték esetén, a cellának pontosan $\pi \cdot d$ regiszterre van szüksége, melyeken az értékek rendre keresztülhaladnak, mielőtt értékük a következő cellának adódik át. Az előbb vázolt helyzetnek megfelelően, $P \cdot d \neq \vec{0}$ esetében az adatátvitel ugyancsak $\pi \cdot d$ regiszte-

ren keresztül történik, itt azonban nem fontos, hogy ezek mind a számítást végző cellában legyenek elhelyezve.

Minden egyes d függőségi vektor esetén szükség van megfelelő regiszterekre. A 7.3(b) ábrán a cellához rendelt három bemenetet láthatjuk, melyek a d_A, d_B és d_C függőségi vektoroknak felelnek meg. Mind a három vektor esetében fennáll, hogy a $P \cdot d \neq \vec{0}$, illetve (7.6) és (7.4) összefüggések következtében $\pi \cdot d = 1$. Tehát minden egyes d függőségi vektorhoz egyetlen regiszterre van szükség. A (7.3) rendszer szabályos volta miatt a cella három bemenetéhez ugyanakkor három kimenet tartozik, a cella középpontjához képest egymással ellentétes pozícióban.

Mivel a d függőségi vektorok száma egy (7.3)-hoz hasonló rendszer által statikusan korlátozott, és a nekik megfelelő $\pi \cdot d$ érték rögzített és többnyire kicsi, egy cellának általában kevés regiszterre van szüksége.

A cella három be-, illetve kimenete három dinamikus mátrix használatát teszi lehetővé. A 7.1. ábrával ellentétben egy $\sum_{k=1}^{4} a_{ik} \cdot b_{kj}$ skalárszorzat kiszámítása itt nem egyetlen cellában történik, hanem a szisztolikus rács cellái közt felosztva. Itt tehát feltétlenül szükséges, hogy az összeget részösszegek sorozatára bontsuk. Ez tehát példa egy szétosztott generikus operátorra.

A három bemeneten, a hozzátartozó regisztereken és a három kimeneten kívül a 7.3(b) ábrán egy szorzót láthatunk egy sorosan hozzákapcsolt összeadóval. A (7.8) leképezés alkalmazása a (7.3) rendszer $c(i,j,k) = c(i,j,k-1) + a(i,j-1,k) * b(i-1,j,k)$ egyenletének S értéktartományára a fent említett két egység összes cellában való jelenlétét eredményezi. Mivel az egyenlet alapján az összeg felépítése csakis a szorzat sikeres végrehajtása után lehetséges, ebből következik a két operátor 7.3(b) ábrán látható sorrendje.

Hogy a megfelelő operandusok honnan származnak, az a hozzájuk tartozó függőségi vektorok projekciójából olvasható le. Így például $a(i,j-1,k)$-hoz a $d_A = (0,1,0)$ függőségi vektor tartozik. Ennek projekciója, $P \cdot d_A = (-1,1)$, az A mátrix haladási irányát mutatja. A beolvasandó adatot ezért, a számítást végző processzor szemszögéből nézve az ezzel ellentétes $(1,-1)$ irányból kell várni, vagyis a cella bal alsó sarkához kapcsolódó csatornából (de az A regiszteren keresztül). Ugyanígy $b(i-1,j,k)$ jobb oldalról jön (a B regiszteren keresztül), $c(i,j,k-1)$ pedig fentről (a C regiszteren keresztül). A megfelelő $a(i,j,k)$, $b(i,j,k)$ és $c(i,j,k)$ értékek a szemközti irányba csatornákon keresztül továbbítódnak: jobbra, felfelé, bal alsó irányba.

Másrészt, a (7.7)-beli P projekciós mátrixot alkalmazva a d_C-re a $(0,0)$ projekciót kapjuk. Mivel ugyanakkor $\pi \cdot d_C = 1$, következik, hogy pontosan egy C regiszterre van szükség a C mátrix minden egyes eleme számára. Ez a regiszter szolgáltatja a $c(i,j,k)$ érték kiszámítására a $c(i,j,k-1)$ értéket, majd a számítást követően felveszi a $c(i,j,k)$ értékét. Mindez a 7.1(b) ábrán jól követhető. A 7.1(a) ábra ennek megfelelően azt mutatja, hogy a C mátrix számára nincs szükség cellák közti kapcsolatra: a mátrix *stacionárius*.

Gyakorlatok

7.2-1. Egy u projekciós irányhoz mindig több különböző P projekciós mátrixot találunk.

a. Mutassuk meg, hogy a

$$P = \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \end{pmatrix}$$

projekciós mátrix is megfelel az $u = (1,1,1)$ projekciós iránynak.

b. Számítsuk ki ennek a projekciós mátrixnak a segítségével a (7.3) rendszer értéktartományát.

c. Az így kapott térkoordináták különböznek a (7.2.3) pontban adottaktól. Miért mondhatjuk mégis, hogy a két esetben kapott ponthalmazok topológiai szempontból ekvivalensek?

d. Tanulmányozzuk a cellák elhelyezkedését a két esetben, hasonlóságokat és különbségeket keresve.

7.2-2. Végezzük el a 7.2. alfejezetben leírt számításokat a (7.3) rendszerrel és a

$$T = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

tér-idő-mátrixszal kapcsolatban.

7.3. A be/kiviteli séma levezetése

A 7.3(a) ábrán a *be/kiviteli séma* az A, B, C mátrixokhoz tartozó *adatfolyamok irányával* van csupán megadva. Az adatok be/kivitelével kapcsolatos folyamatok megértéséhez szükséges részleteket a 7.6. ábra tartalmazza.

A 7.6. ábrán látható be/kiviteli séma a 7.1(a) ábrához képest egy sor új mozzanatot tartalmaz. Itt is egy bemeneti, illetve kimeneti adatfolyam áramlik keresztül mindegyik közönséges peremcellán, a szisztolikus rács sarkaihoz pedig két-két adatfolyam tartozik. Az egy mátrixhoz tartozó beviteli cellák persze itt már nem egy egyenes vonal mentén helyezkednek el, hanem két, egymással határos perem mentén.

A 7.6. ábrán látható adatszerkezeteknek ugyanakkor más a dőlésszögük, mint a 7.1(a) ábrán. Ezen kívül a 7.6. ábrán az A és B mátrix a 7.1(a) ábrához viszonyítva adatfolyamonként kétharmaddal csökkentett sebességgel érkezik.

Kevés fáradozással alapjában véve itt is lehetséges az, hogy az adott szisztolikus rácshoz elemi szinten próbáljunk találó ki/beviteli sémát építeni.

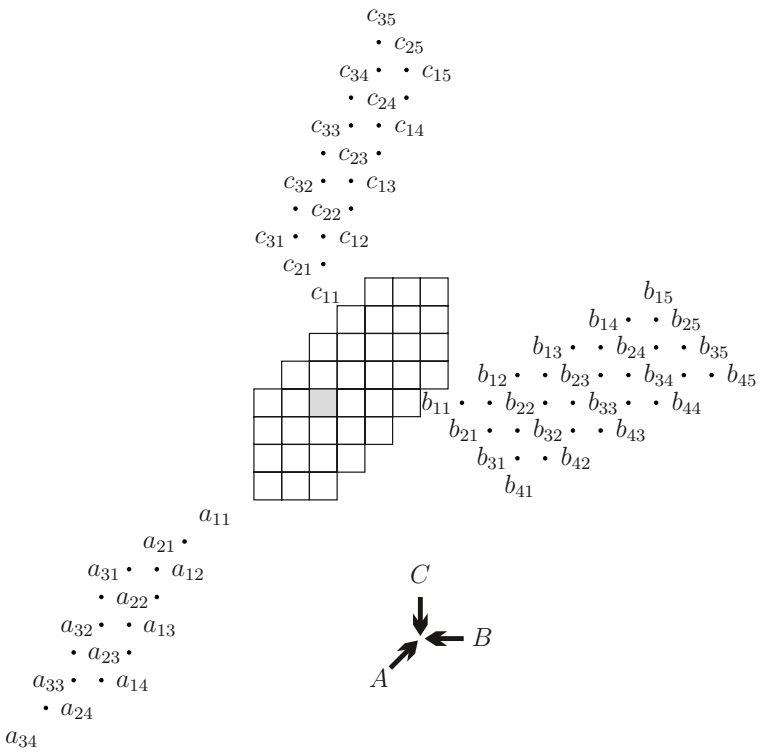
Sokkal biztosabb azonban egy formális levezetésen keresztül vezető út. A következő pontokat az eljárás egyes lépéseinek bemutatásának szenteljük.

7.3.1. Az adatszerkezetek indexeitől az iterációs vektorokig

Mindenekelőtt az absztrakt adatszerkezetek és az értékadásmentes jelölésmód konkrét változópéldányai közti összefüggést kell megvilágítanunk.

Az A mátrix minden eleme egy i sorindexszel és egy k oszlopindexszel van ellátva. Ezt a két indexet egy $w = (i,k)$ ***adatszerkezet-vektorral*** foghatjuk össze. Az a_{ik} elem a (7.3) rendszer $a(i, j, k)$ példányának felel meg, tetszőleges j-vel. E példány koordinátái $\mathbb{R}^3$-nek elemei, és mind egy egyenesen helyezkednek el a $q = (0, 1, 0)$ irány mentén. Az (i, k) adatszerkezet vektortól az (i, j, k) koordinátákra való átmenetet az alábbi leképezés írja le:

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} i \\ k \end{pmatrix} + j \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{7.22}$$



7.6. ábra. Részletes be/kiviteli séma a 7.3(a). ábrához.

A (7.3) rendszerben használt alak esetén minden egyes változópéldány (i, j, k) koordináta vektora pontosan megegyezik az értéktartomány azon iterációs vektorával, melynek kapcsán az illető változópéldány kiszámítása történik. Ezért a (7.22) képletet az adatszerkezet vektorok és az iterációs vektorok között fennálló összefüggésként is felfoghatjuk. Absztrakt módon kifejezve, a megfelelő v iterációs vektorokat megkaphatjuk a

$$v = H \cdot w + \lambda \cdot q + p \tag{7.23}$$

képlet segítségével a w adatszerkezet vektorból. A p affin vektor a mi példánk esetén mindig a nullvektor, általános esetben viszont szükség lesz rá.

Mivel $b(i, j, k) = b_{kj}$, a B mátrixnak megfelelő megjelenítés az alábbi:

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} k \\ j \end{pmatrix} + i \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} . \tag{7.24}$$

Ami a C mátrixot illeti, minden $c(i,j,k)$ változópéldány szükséges egy másik érték kiszámításához. Viszont az összes rögzített (i,j) indexpárral rendelkező $c(i,j,k)$ példányt tekinthetjük úgy, mint ami a c_{ij} mátrixelemhez tartozik, mivel ezek közvetlenül a c_{ij} kiszámítására használt összegző operátor sorba fejtéséből származnak. A (7.23) képletnek megfelelően tehát C-re a következőket kapjuk:

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \end{pmatrix} + k \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{7.25}$$

7.3.2. Adatszerkezetekről készült helyzetképek

Az A, B és C mátrixok mindegyike az adatszerkezet két indexének iránya mentén jött létre: egy sor, illetve egy oszlop mentén. A $(0,1)$ különbségvektor írja le az átmenetet egyik elemről a másikra ugyanazon a soron belül, azaz a következő oszlopbeli elemet adja: $(0,1) = (x, y+1) - (x,y)$. Ugyanígy az $(1,0)$ különbségvektor az ugyanabban az oszlopban, és a következő sorban levő elemhez vezető átmenetet írja le: $(1,0) = (x+1, y) - (x,y)$.

A 7.1(a) és 7.6. ábrákon látható be/kiviteli sémák különböző helyzetképeket mutatnak be: az adatok szisztolikus rácshoz viszonyított helyzete ugyanarra az időpontra vonatkozik.

Amint a 7.6. ábrán látható, az absztrakt adatszerkezetek téglalap alakú mátrixformái ebben a helyzetképben parallelogrammává alakulnak. Ez az alkalmazott tér-idő-leképezés lineáris voltának tulajdonítható. Ezeket a paralelogrammákat is ki lehet fejezni a peremek mentén kiszámolt különbségvektorok segítségével.

Most az adatszerkezetek Δw különbségvektorait Δz térbeli különbségvektorokká szeretnénk alakítani. Ehhez olyan konkrét v, v' iterációs vektor párokat kell meghatároznunk, a (7.23) összefüggésbeli λ paraméterek megválasztása révén, melyeket a megválasztott tér-idő-leképezés ugyanarra az időpontra képez le. Hogy pontosan melyik időpontról is van szó, az itt most nem lényeges. Tehát a $\pi \cdot v = \pi \cdot v'$ egyenlőséget összevetjük azzal, hogy

$$v = H \cdot w + \lambda \cdot q + p \quad \text{és} \quad v' = H \cdot w' + \lambda' \cdot q + p\ . \tag{7.26}$$

Ez azt eredményezi, hogy

$$\pi \cdot H \cdot (w - w') + (\lambda - \lambda') \cdot \pi \cdot q = 0\ , \tag{7.27}$$

azaz

$$\Delta\lambda = (\lambda - \lambda') = \frac{-\pi \cdot H \cdot (w - w')}{\pi \cdot q}\ . \tag{7.28}$$

A keresett Δz térbeli különbségvektor tehát a használt leképezések lineáris voltából adódóan a következőképpen számítható ki az adatszerkezet $\Delta w = w - w'$ különbségvektorából:

$$\Delta z = P \cdot \Delta v = P \cdot H \cdot \Delta w + \Delta\lambda \cdot P \cdot q\ , \tag{7.29}$$

tehát

$$\Delta z = P \cdot H \cdot \Delta w - \frac{\pi \cdot H \cdot \Delta w}{\pi \cdot q} \cdot P \cdot q\ . \tag{7.30}$$

Most meghatározzuk a (7.30) képletből a Δz térbeli különbségvektorokat az A mátrix számára. A fentiek alapján érvényes a következő:

$$H = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix},\ q = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix},\ P = \begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \end{pmatrix},\ \pi = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}.$$

Mivel $\pi \cdot q = 1$, következik

$$\Delta z = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \cdot \Delta w + \Delta\lambda \cdot \begin{pmatrix} -1 \\ 1 \end{pmatrix} \quad \text{ahol} \quad \Delta\lambda = -\begin{pmatrix} 1 & 1 \end{pmatrix} \cdot \Delta w\ .$$

A sorok esetében a $\Delta w = (0, 1)$ különbségvektorunk van, melyből a $\Delta z = (2, -1)$ térbeli különbségvektor következik. Ugyanígy az oszlopok esetében $\Delta w = (1, 0)$-ból következik a $\Delta z = (1, -2)$. Egyeztessük ezt most a 7.6. ábrával, és láthatjuk, hogy az A sorai valóban a $(2, -1)$ vektor mentén helyezkednek el, az oszlopok pedig a $(1, -2)$ vektor mentén.

Ugyanígy kapjuk a B sorai esetében, hogy $\Delta z = (-1, 2)$, illetve $\Delta z = (1, 1)$ a B oszlopaira. Megfelelőképpen $\Delta z = (-2, 1)$ a C sorai esetén, és $\Delta z = (-1, -1)$ a C oszlopai esetében.

Ezzel tehát most már meg tudjuk szerkeszteni a be/kiviteli sémát minden egyes mátrixra külön-külön.

7.3.3. A be/kiviteli séma megszerkesztése

Az A, B, C mátrixok megjelenési formája a helyzetkép számára adott ugyan, de még meg kell határozzuk a szisztolikus rácshoz (és egyúttal egymáshoz) viszonyított elhelyezkedésüket. Ennek megvalósítására létezik egy egyszerű grafikus módszer.

Kiválasztunk egy tetszőleges iterációs vektort, legyen ez $v = (1, 1, 1)$. A P projekciós mátrix segítségével levetítjük ezt arra a cellára, melyben a neki megfelelő számítások történnek.

$$z = \begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Az $(1, 1, 1)$ iterációs vektorhoz az $a(1, 1, 1)$, $b(1, 1, 1)$ és $c(1, 1, 1)$ számítások vannak hozzárendelve, melyek viszont a a_{11}, b_{11} és c_{11} mátrixelemeknek felelnek meg. Helyezzük az A, B, C mátrixok esetén kapott be/kiviteli sémát a szisztolikus rácsra úgy, hogy a megfelelő a_{11}, b_{11} és c_{11} bemenetek mind a $z = (0, 0)$ cellában legyenek.

Ezzel tulajdonképpen már készen is volnánk. Ez a be/kiviteli séma azonban átfedi a szisztolikus rács celláit, és emiatt nem elég világosan felismerhető. Ezért minden mátrix be/kiviteli sémáját egy-egy pozícióval tovább toljuk az adatfolyamok haladásával ellentétes irányba mindaddig, amíg nem áll fenn többé átfedés. Ezzel pontosan a 7.6. ábrán bemutatott be/kiviteli sémát kapjuk.

Ezen elegáns grafikus módszer mellett itt is megvan a lehetőségünk arra is, hogy az átfedésmentes elhelyezkedést formálisan számoljuk ki.

Csak a be/kiviteli séma megadása után tudjuk a szükséges időszeletek számát meghatározni. Az első lényeges időszelet a legelső adat bevitelével kezdődik. Az utolsó lényeges időszelet az utolsó adatkivitellel végződik. A 7.6. ábrán látható példa esetén a számítások

kezdetét a b_{11} elem 0 időszeletben történő bevitelétől számítjuk, a számítások befejeztének pedig a 14. időszeletet tekintjük, miután a c_{35} eredmény kiszámítása megtörtént. Ez összesen 15 időszeletet tesz ki – ami öttel több a tulajdonképpeni számítások elvégzéséhez szükséges időszeletek számánál.

7.3.4. Tér-idő-leképezés által előidézett adatsebesség

Az A és B mátrix 7.1(a) ábrán látható beviteli sémája kompakt felépítésű: ha megrajzoljuk az ábrán a mátrix széleit, ennek belsejében nem találunk kihasználatlan helyet.

Más a helyzet a 7.6. ábrán. Bármelyik adatfolyamot tekintjük, mindegyik elemet két kihasználatlan hely követ. A beviteli mátrixok esetében ez azt jelenti, hogy: a szisztolikus rács peremcellái csak minden harmadik időszeletben kapnak valódi adatot.

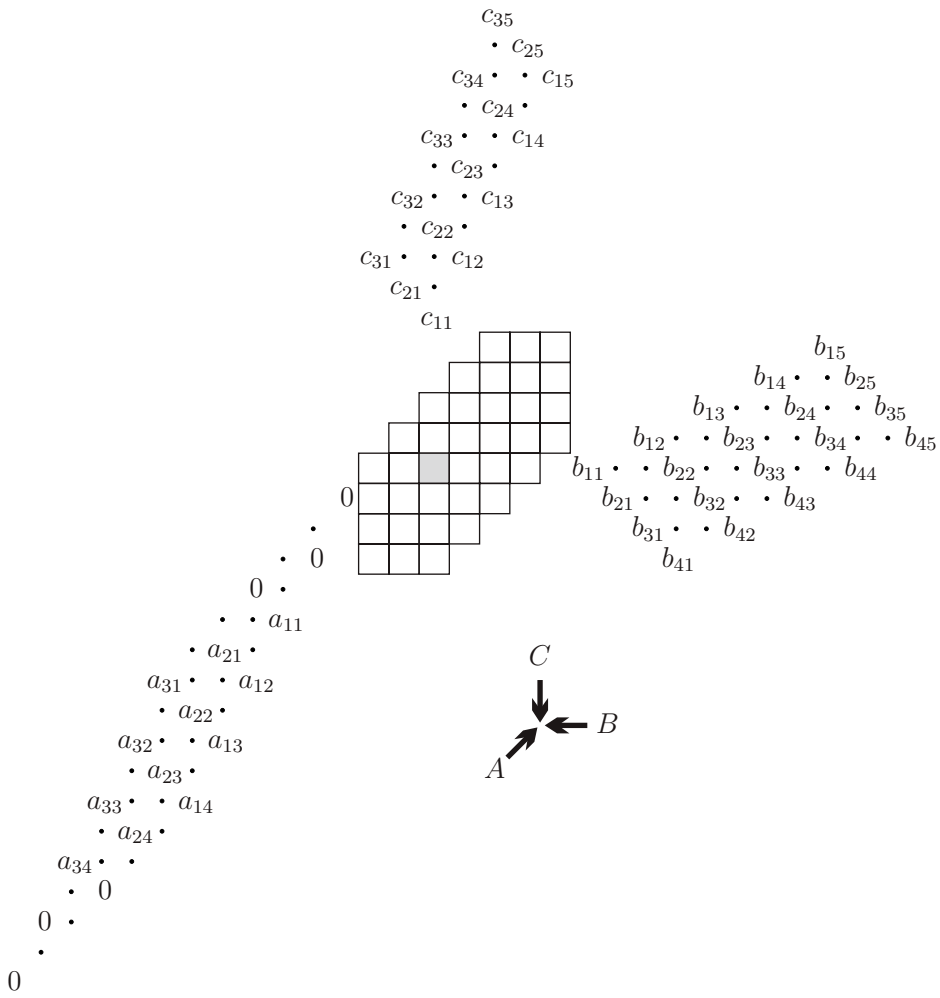
Ez a tulajdonság az éppen alkalmazott tér-idő-leképezés egyenes következménye. Maguk az absztrakt adatszerkezetek mind a két esetben kompaktak. Azonban, hogy milyen sűrűn helyezkednek el az adatok a be/kiviteli sémában, az a T transzformációs mátrix determinánsának abszolút értékétől függ: minden egyes be/kiviteli adatfolyamban a tényleges értékek pontosan $|\det(T)|$ pozíciónyi távolságra követik egymást. Míg a 7.1. ábra esetében $|\det(T)| = 1$ érvényes, addig a 7.6. ábra esetében a $|\det(T)| = 3$ értéket állapíthatjuk meg, melynek most már gyakorlati jelentése is ismert.

Mi történik vajon a ki nem használt helyekkel, mint amilyeneket a 7.6. ábrán is láthatunk? Annak ellenére, hogy a 7.3. ábra szisztolikus rácsának minden cellája tulajdonképpen csak minden harmadik időszeletben végez értelmes munkát, nincs sok értelme annak, hogy minden munkát végző lépés után két időszelet erejéig szüneteltessük őket. Ha pontosabban megfigyeljük, megállapíthatjuk, hogy a 7.6. ábrán pontokkal jelölt helyeken lévő értékeknek semmiféle befolyásuk nincs a c_{ij} elemek kiszámítására, mivel egy $c(i, j, k)$ változó kiszámításának időpontjában soha nincsenek ott a számításnak megfelelő cellában. A nem használt helyeket tehát egyszerűen tetszőleges, akár véletlenszerű értékekkel is feltölthetjük anélkül, hogy a végeredményt elrontanánk ezzel. Ráadásul az is lehetséges, hogy a 7.3. ábra szisztolikus rácsának segítségével egyidőben három különböző mátrixszorzást elvégezzünk, anélkül, hogy ezek zavarnák egymást. A 7.3.7. pontban ezt még pontosabban kifejtjük.

7.3.5. Be/kivitel kiterjesztése és a bővített be/kiviteli séma

A 7.6. ábrát alaposabban tanulmányozva egy újabb kérdés merül fel. Tekintsük példaként a c_{22} útját a szisztolikus rács celláin keresztül. A tér-idő-leképezés alapján a számítások a $(-1, 0)$, $(0, 0)$, $(1, 0)$ és $(2, 0)$ cellákban mennek végbe. A 7.6. ábrán látható be/kiviteli séma szerint természetesen előbb a $(-2, 0)$ cellán, illetve legvégül a $(3, 0)$ cellán is keresztülhalad a c_{22}.

Ezt értelmezhetjük úgy, hogy a választott tér-idő-leképezés révén a (7.3) rendszerbe fiktív számítások kerülnek be, itt például az értéktartomány új $(2, 2, 0)$ és $(2, 2, 5)$ pontjai kapcsán. Ennek a jelenségnek az alapja az a tény, hogy a be/kimeneti műveletek értéktartományai nem a megválasztott u projekciós iránnyal párhuzamosan helyezkednek el. Ennek következtében egyes be/kimeneti műveletek olyan cellákra vetítődnek, amelyek nem a szisztolikus rács peremén vannak. Itt viszont a be/kimeneti műveletre közvetlen módon nem kerülhet sor. Az útvonalat az adatfolyamok iránya mentén meghosszabbítva, illetve ezekkel ellentétes irányban ezektől a belső celláktól a szisztolikus rács pereméig, kiküszöbölhetjük



7.7. ábra. Bővített be/kiviteli séma a 7.6. ábrához.

ezt a problémát. Ezzel viszont új számítások jönnek be, esetleg újabb pontokkal bővül az értéktartomány ***(be/kivitel kiterjesztése).***

Vigyáznunk kell arra, nehogy a $(-2, 0)$ és $(3, 0)$ cellákban végbemenő, mellékes számítások elrontsák a c_{22} tulajdonképpeni értékét. A mátrixszorzás esetében ezt elég könnyű elérni (az általános esetben viszont sokkal nehezebb). A generikus összegzőoperátornak van egy semleges eleme, éspedig a nulla. Tehát ha elérjük, hogy a kiegészítés révén bekerült számításokban mindig csak a nullát adjuk hozzá az eddigiekhez, ez semmiféle kárt nem okoz.

A 7.7. ábra egy alkalmas bővített be/kiviteli sémára ad példát. Az A mátrix elé és mögé hozzá vannak fűzve a szükséges null-elemek. Mivel a bevitt null értékeket adatnak kell tekintenünk, a 7.6. ábra be/kiviteli sémája még egy pozícióval visszatolódik.

A számítások tehát már a -1. időszeletben megkezdődnek, viszont ugyanúgy a 14. időszelettel végződnek, mint korábban. A teljes végrehajtási idő 16 időszelet lesz.

7.3.6. A stacionárius változók kezelése

Térjünk vissza a 7.1(a) ábra példájához. Az A és B elemeinek beviteléhez nincs szükség semmiféle kiterjesztésre, mivel ezekre mindig a peremcellákban van legelőször szükség. Más a helyzet viszont a C mátrixszal. Ennek elemeit stacionárius változókban számítjuk ki, vagyis mindvégig ugyanabban a cellában. A c_{ij} eredmények tehát a szisztolikus rács belsejére esnek, ahonnan a kiszámításukat követően egy további folyamatban a szisztolikus rács peremcelláihoz kell továbbítanunk őket, mert csak ezeken keresztül férhetünk hozzájuk.

Annak ellenére, hogy a megoldandó feladatot felületesen szemlélve úgy tűnik, mintha az a 7.3.5. pontban leírthoz nagyon hasonló lenne, és emiatt igen egyszerűnek látszik, mégis egy teljesen más helyzetről van itt szó. Nem arról van szó ugyanis, hogy a meglévő adatfolyamokat előre vagy visszafelé a szisztolikus tömb pereméig meghosszabbítsuk. Hiszen stacionárius változók esetében a függőségi vektor a nullvektor. Így ez semmiféle térbeli adatfolyamot nem eredményezhet. Egy ilyen adatfolyamot nekünk kell előbb felépítenünk, amiben igen nagy a szabadságunk. Alapjában véve egy vezérlő egységre is szükségünk van a cellákban. Ezt a kérdést a 7.4. alfejezetben tárgyaljuk tovább.

7.3.7. Számítások összekapcsolása

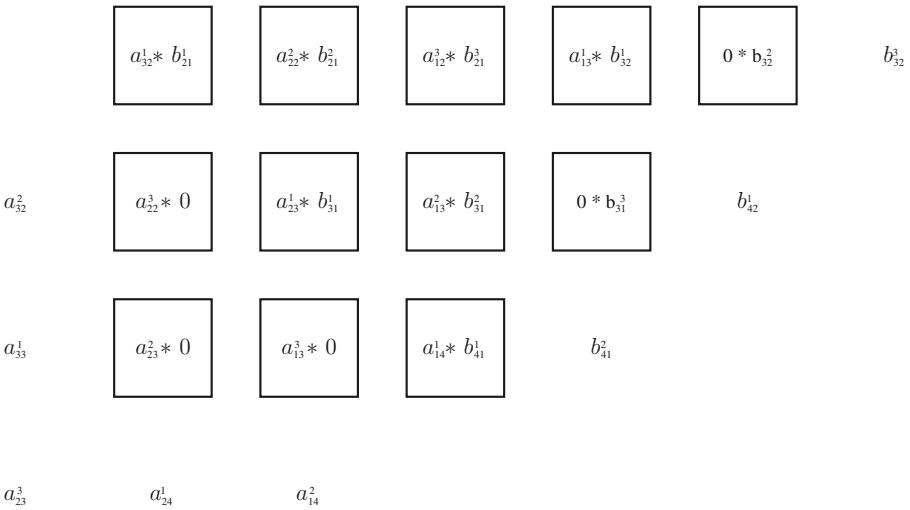
Amint az könnyen belátható, a 7.3. ábra szisztolikus rácsának *kihasználtsága* a 7.7. ábra be/kiviteli sémájával meglehetősen csekély. Anélkül, hogy a betöltési vagy a végső szakaszt közelebbről tanulmányoznánk, máris észrevesszük, hogy a rács átlagos kihasználtsága kevesebb, mint egyharmadot tesz ki – hiszen valódi számítást minden cella legfeljebb minden harmadik időszeletben végez.

Egy egyszerű eszköz ennek a viselkedésnek a javítására a számítások ***összekapcsolása.*** Amennyiben három egymástól független, ugyanazokkal a paraméterekkel rendelkező mátrixszorzást kell elvégeznünk, ezek adatait bevihetjük egymástól csupán egy időszeletnyi távolságban, anélkül, hogy a szisztolikus rácson vagy annak celláin bármit is változtatnánk. A 7.8. ábra helyzetképet mutat a szisztolikus rács egy részletéről, a be/kiviteli séma megfelelő részeivel.

Szeretnénk egyúttal valamilyen formális levezetésen keresztül meggyőződni arról, hogy ez az ötlet valóban működik. Ennek érdekében módosítjuk a (7.3) rendszert úgy, hogy a változókat és az értéktartományt egy negyedik dimenzióval bővítjük ki, mely csupán a három mátrixszorzás megkülönböztetésére szolgál:

Bemeneti műveletek

$$
\begin{aligned}
a(i,j,k,n) &= a^n_{ik} && 1 \le i \le N_1, j = 0, 1 \le k \le N_3, 1 \le n \le 3\ , \\
b(i,j,k,n) &= b^n_{kj} && i = 0, 1 \le j \le N_2, 1 \le k \le N_3, 1 \le n \le 3\ , \\
c(i,j,k,n) &= 0 && 1 \le i \le N_1, 1 \le j \le N_2, k = 0, 1 \le n \le 3\ .
\end{aligned}
$$



7.8. ábra. Három mátrixszorzás összekapcsolt kiszámítása a 7.3. ábra szisztolikus rácsával (részlet).

Számítások

$$
\begin{aligned}
a(i,j,k,n) &= a(i,j-1,k,n) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3, 1 \le n \le 3\ , \\
b(i,j,k,n) &= b(i-1,j,k,n) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3, 1 \le n \le 3\ , \\
c(i,j,k,n) &= c(i,j,k-1,n) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3, 1 \le n \le 3\ . \\
&\quad + a(i,j-1,k,n) \cdot b(i-1,j,k,n)
\end{aligned}
$$

Kimeneti műveletek

$$
c^n_{ij} = c(i,j,k,n) \qquad 1 \le i \le N_1, 1 \le j \le N_2, k = N_3, 1 \le n \le 3\ . \tag{7.31}
$$

Nyilvánvaló, hogy a (7.31) rendszerben a különböző n értékekhez tartozó feladatoknak semmi közük egymáshoz. Ennek tehát a szisztolikus rácsban is így kell lennie. Egy erre vonatkozó, az ábrának megfelelő tér-idő-mátrix a következő

$$
T = \begin{pmatrix} 0 & -1 & 1 & 0 \\ -1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}. \tag{7.32}
$$

A T mátrix itt már nem négyzetes, de ez nem tesz semmit. A térkoordináták kiszámításának szempontjából a negyedik dimenzió teljesen jelentéktelen; ezért a T első két sorának utolsó oszlopában levő null-értékek segítségével egyszerűen eltüntethető.

T utolsó sora újraépíti a π idővektort. A π megfelelő megválasztásával a három megoldásra váró feladat átfedésmentesen ágyazható be a tér-idő-szerkezetbe. A három feladat egymásnak megfelelő iterációs vektorainak példányai egymástól egy időegységnyi távolságban ugyanarra a cellára lesznek vetítődnek, mivel π negyedik bemenete 1.

Kiszámítjuk most az átlagos ***kihasználtságot*** összekapcsolással, illetve e nélkül az $N_1 = 3$, $N_2 = 5$ és $N_3 = 4$ konkrét feladat paraméterei esetében. Egyetlen mátrixszorzás esetén $N_1 \cdot N_2 \cdot N_3 = 60$ számítást kell elvégezni. Ilyenkor egy szorzás és a hozzá tartozó összeadás összetett műveletnek számít, azaz együtt csupán egyetlen számítást jelent; a be/kimeneti műveleteket nem számítjuk hozzá. A szisztolikus rács 36 cellával rendelkezik.

Összekapcsolás nélkül a szisztolikus rácsunknak 16 időszeletre van szüksége egy mátrixszorzás elvégzéséhez. Ez a cellák következő átlagos kihasználtságát eredményezi:

$$60/(16 \cdot 36) \sim 0.104 \text{ számítás időszeletenként és cellánként.}$$

A leírt összekapcsolási technika alkalmazása esetén mindhárom mátrix kiszámítása csupán két időszelettel igényel többet, tehát 18 időszeletet tesz ki. Ám a végrehajtott számítások száma megháromszorozódott, a cellák átlagos kihasználtsága tehát a következő:

$$3 \cdot 60/(18 \cdot 36) \sim 0.278 \text{ számítás időszeletenként és cellánként.}$$

Az összekapcsolással tehát körülbelül 167 százalékkal sikerült növelnünk a kihasználtságot.

Gyakorlatok

7.3-1. Vezessük le formálisan a B és C mátrix térbeli különbségvektorait a 7.6. ábrán látható be/kiviteli séma esetén a (7.30) összefüggés alapján.

7.3-2. Vázoljuk a bővített be/kiviteli sémát a 7.6. ábrához, arra az esetre, amikor a többlet szorzásműveletek mindkét operandusát nullára kell állítanunk.

7.3-3. Alkalmazzuk a 7.3. alfejezetben bemutatott módszereket a 7.1. ábra szisztolikus rácsára.

7.3-4.⋆ Igazoljuk a (7.32) sajátos tér-idő-leképezés 7.3.7. pontban leírt tulajdonságait a (7.31) rendszerre vonatkozóan.

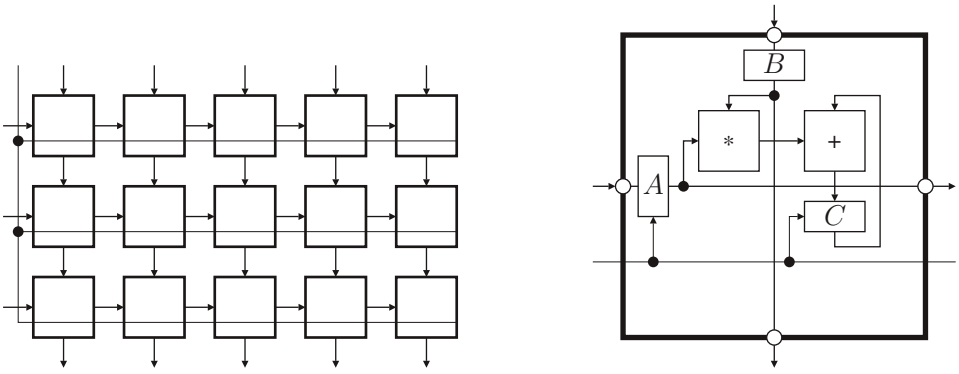
7.4. Vezérlési szempontok

Mindeddig abból indultunk ki, hogy egy szisztolikus rács cellái teljesen egyformán viselkednek minden időszeletben. Érdekes példákat találhatunk ilyen szisztolikus rácsokra. Általában azonban vezérlés segítségével a cellákat különböző *működési módokba* lehet állítani. A következőkben néhány tipikus helyzetet fogunk tanulmányozni.

7.4.1. Vezérlésmentes cellák

A 7.3(b) ábrán látható cella az A, B, C regisztereket tartalmazza, amelyek egy globális órajel aktivál annak érdekében, hogy átvegyék a mindenkori bemenetükön levő jeleket, majd a kimenetükre továbbítsák ezeket. Ettől a globális aktiválástól eltekintve azonban, a sejtek működése minden időszeletben ugyanaz: a három, A, B, C bemenő operandusra egy *szorzás-összeadás* művelet hajtódik végre, melynek az eredménye egy szomszédos cellába kerül; ezzel párhuzamosan az A és B operandusok két másik szomszédos cellának továbbítódnak. Következésképpen ez a cella semmiféle vezérléssel nem rendelkezik.

A generikus összegző operátor végrehajtásához szükséges $c(i, j, 0)$ kezdőértékek, melyeknek itt nem kell feltétlenül nullértékeknek lenniük, a 7.6. ábra alapján bemeneti adatfo-



(a) (b)

7.9. ábra. Regiszterek visszaállítása globális vezérléssel: **(a)** rácsszerkezet; **(b)** cellaszerkezet.

lyamként kerülnek a szisztolikus rácsba, a $c(i, j, N_3)$ eredmények pedig a rács peremén keresztül, ugyanabban az irányban áramlanak kifelé. A 7.3(b) ábra szerint tehát a be/kimenet implicit módon része a cellák működésének. Ennek a nagyon egyszerű, vezérlésmentes sejtműködésnek az ára mindhárom mátrixméret korlátozása: ($M_1 \times M_3$)-as A mátrix csak akkor szorozható össze a 7.3. ábra szisztolikus rácsán keresztül egy ($M_3 \times M_2$)-es B mátrixszal, ha a rács meghatározott N_1, N_2, N_3 paramétereire a következő feltételek érvényesülnek: $M_1 \leq N_1$, $M_2 \leq N_2$ és $M_3 \leq N_3$.

7.4.2. Globális vezérlésű cellák

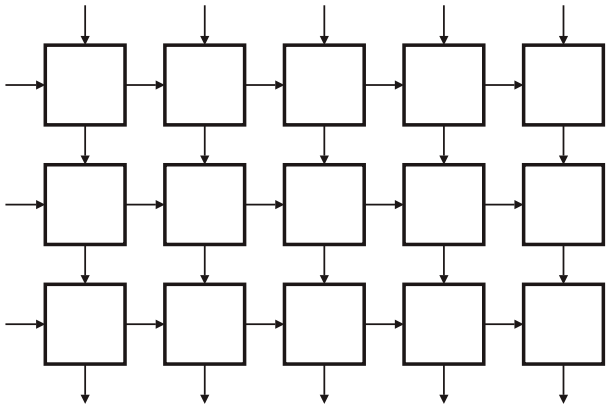
A 7.1. ábrán látható szisztolikus rács esetén az előírások erre vonatkozóan kevésbé szigorúak: bár M_1 és M_2 itt is korlátozva van $M_1 \leq N_1$ és $M_2 \leq N_2$ által, az M_3-ra viszont nincs semmi korlátozás. A feladat azon paraméterei, melyeket a rács előre meghatározott paraméterei nem korlátoznak, csak időben tudnak megnyilvánulni, térben azonban nem. Ezáltal *stacionárius változók* használatára kényszerülünk.

Egy új számítás kezdetén minden egyes stacionárius változóhoz rendelt regisztert a megelőző számításoktól független alapállapotba kell hozni. A 7.3(b) ábrán látható szisztolikus cella esetében ez a regiszter a C. Egy, az órajelhez hasonlítható globális jel segítségével az összes cella C regisztere egyszerre törölhető, azaz lenullázható. A 7.9. ábra egy ezen az ötleten alapuló rácsszerkezetet, illetve cellaszerkezetet mutat be.

7.4.3. Helyi vezérlés

Sajnos a mátrixszorzáshoz nem elegendő csupán a globális vezérlés elve. A 7.1. ábrán bemutatott szisztolikus rácsból ugyanis két lényeges tulajdonság is hiányzik: egyrészt a $c(i, j, 0)$ változóknak nincs kezdőértékük, másrészt a c_{ij} eredmények sem a rács peremén jelennek meg.

Először az eredmények perem felé vezérlése valóban egyszerű feladatnak tűnik: egy c_{ij} elem kiszámításának befejeztével már nincs szükség arra, hogy az (i, j) cellának a szom-



c_{15}
c_{14} c_{25}
c_{13} c_{24} c_{35}
c_{12} c_{23} c_{34}
c_{11} c_{22} c_{33}
c_{21} c_{32}
c_{31}

7.10. ábra. Be/kiviteli séma késleltetett eredménymegadás esetén.

szédos $(i, j+1)$ és $(i+1, j)$ cellák felé vezető kapcsolatait az A és B mátrix elemeinek továbbadására használjuk. Használhatjuk tehát más célokra őket. Például a C összes elemét a lefelé vezető kapcsolatokon keresztül a szisztolikus rács alsó peremével vezethetjük.

Sajnos kiderül azonban, hogy a rács alsó részén még be nem fejezett számítások akadályozzák az eredmények átvezetését a fenti részből. Ha az $i+j+N_3$ időszeletben kiszámított c_{ij} eredményt a következő időszeletben az $(i+1, j)$ cellának továbbítanánk, ez értékütközéshez vezetne: mivel az $(i+1, j)$ cella időszeletenként csak egy értéket bocsáthat ki az alsó csatornán keresztül, vagy a c_{ij}-nek kellene várnia, vagy az $(i+1, j)$ cellában kiszámolt eredménynek. Ugyanez a probléma lefelé az összes további cellát érintené.

Megoldásképpen késleltethetjük a c_{ij} továbbadását. Ha c_{ij}-nek egy cellán való keresztülhaladáshoz nem egy, hanem két időszeletre van szükség, akkor az ütközések elmaradnak. Az eredmények így egymástól egy időszeletnyi eltéréssel haladnak egymást követve, ugyanazon a kapcsolaton keresztül. Egy oszlop alsó peremcelláján legelőször az illető oszlop utolsó sorának eleme jelenik meg, majd az utolsó előtti, legvégül az első. Tehát a 7.10. ábrán látható be/kimeneti sémát kapjuk.

Honnan tudja egy cella, mikor kell az alsó csatornán keresztül továbbítania, ahelyett hogy a B mátrix elemeit a C mátrix elemeivel együtt adná tovább? Ezt a feladatot a cella globális és lokális vezérlésének kombinált alkalmazásával, véges áállapotú automaták segítségével oldhatjuk meg:

Amikor az A és B utolsó értékeit is bevisszük az $(1, 1)$ cellába, egy globális jelet küldünk az összes cellának, ami jelzi, hogy minden cellában elindíthatunk egy számlálót, amely a

még elvégzendő számítási lépések számát adja meg. Az (i, j) cellában még $i + j - 1$ további lépést kell elvégezni, mielőtt a sima továbbvezetésre kapcsolnánk át. A korábban említett globális visszaállító jel később ismét visszakapcsol majd a számítási üzemmódba.

Egy ilyen elv alapján működő szisztolikus rács látható a 7.11. ábrán. A rács felépítése, valamint a kapcsolatszerkezet lényegében változatlanok maradtak. Viszont minden cella kétféle üzemmódban működtethető, melyek között az átkapcsolást egy vezérlőlogika végzi:

1. *Számítási üzemmódban* az összeadás eredménye (akárcsak eddig) az C regiszterbe íródik. Ezzel egy időben a szorzásra használt operandus az B regiszterből a cella alsó csatornáján keresztül átadódik.
2. *Adattovábbító üzemmódban* az B és C regiszterek sorba lesznek kapcsolva. Ebben az üzemmódban a cellák egyetlen feladata az, hogy minden, a felső csatornán kiolvasott értéket két időszeletnyi késleltetéssel továbbítson az alsó csatorna felé.

Adattovábbító üzemmódban az (i, j) cellából kilépő első érték a legutoljára kiszámított, majd az C regiszterben elhelyezett érték lesz, azaz a c_{ij} eredmény. A továbbiakban kilépő összes érték a fentebb elhelyezkedő cellákban kiszámolt, majd innen továbbvezetett eredmény. A 7.11 ábrán megvalósított algoritmus formális leírása a (7.33) értékadásmentes rendszert eredményezi.

Bemeneti műveletek

$$
\begin{aligned}
&a(i,j,k) = a_{ik} && 1 \le i \le N_1, j = 0, 1 \le k \le N_3 ,\\
&b(i,j,k) = b_{kj} && i = 0, 1 \le j \le N_2, 1 \le k \le N_3 ,\\
&c(i,j,k) = 0 && 1 \le i \le N_1, 1 \le j \le N_2, k = 0 .
\end{aligned}
$$

Számítások

$$
\begin{aligned}
&a(i,j,k) = a(i,j-1,k) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 ,\\
&b(i,j,k) = b(i-1,j,k) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 ,\\
&c(i,j,k) = c(i,j,k-1) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3\\
&\quad + a(i,j-1,k) \cdot b(i-1,j,k) .
\end{aligned}
\tag{7.33}
$$

Átvezetés

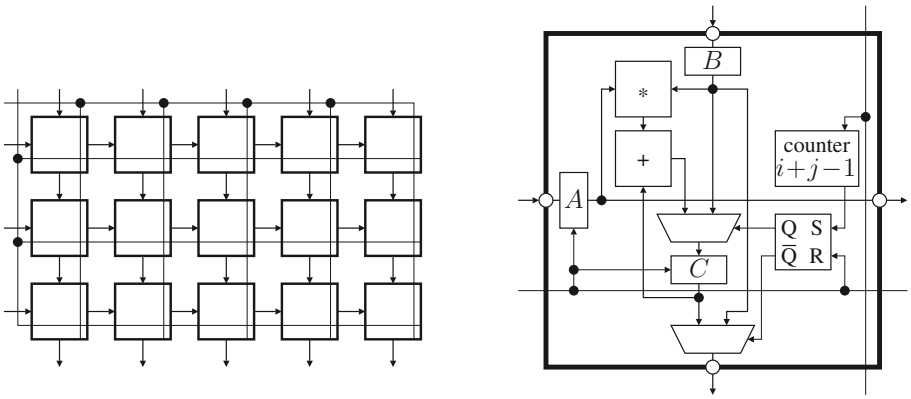
$$
\begin{aligned}
&b(i,j,k) = c(i,j,k-1) && 1 \le i \le N_1, 1 \le j \le N_2, 1 + N_3 \le k \le i + N_3 ,\\
&c(i,j,k) = b(i-1,j,k) && 1 \le i \le N_1, 1 \le j \le N_2, 1 + N_3 \le k \le i - 1 + N_3 ,
\end{aligned}
$$

Kimeneti műveletek

$$
c_{1+N_1+N_3-k,j} = b(i,j,k) \qquad i = N_1, 1 \le j \le N_2, 1 + N_3 \le k \le N_1 + N_3 .
$$

Tisztáznunk kell még, hogy hogyan hozzuk létre egy cella vezérlőjeleit ebben a modellben. A cellának mindenekelőtt egy flipflop állapotkapcsolóval kell rendelkeznie, mely az éppen bekapcsolt üzemmódot adja meg. Ezen flipflop kimenete hozzákapcsolódik a 7.11(b) ábrán látható mindkét multiplexer vezérlőbemenetéhez. A globális visszaállító jel nem csak a cella C regiszterét állítja vissza, hanem a flipflop állapotkapcsolót is: a cella tehát számítási üzemmódban fog dolgozni.

Amikor a globális végjel megérkezik, a cellában egy visszaszámláló indul el, amely minden időszeletben eggyel csökken. A számláló kezdőértéke – cellától függően – $i + j - 1$



7.11. ábra. A helyi és a globális vezérlés kombinált alkalmazása: **(a)** a rács felépítése; **(b)** cellaszerkezet.

értékre lesz állítva. Amikor a számláló eléri a nulla értéket, a flipflop ismét átáll: a cella adattovábbító üzemmódba megy át.

A visszaállítás előtti utolsó, egy cella C regiszterébe továbbított érték felhasználható a cellában kiszámítandó következő skalárszorzat szabadon választható kezdőértékeként, ha az C regiszter közvetlen visszaállításáról lemondunk. Ezután, akárcsak a 7.3. ábrán látható szisztolikus rács esetében, az általános

$$C = A \cdot B + D \,, \tag{7.34}$$

feladatot a következő képletek segítségével oldjuk meg:

Bemeneti műveletek

$$\begin{aligned} &a(i,j,k) = a_{ik} && 1 \le i \le N_1,\ j = 0,\ 1 \le k \le N_3 \,, \\ &b(i,j,k) = b_{kj} && i = 0,\ 1 \le j \le N_2,\ 1 \le k \le N_3 \,, \\ &c(i,j,k) = d_{ij} && 1 \le i \le N_1,\ 1 \le j \le N_2,\ k = 0 \,. \end{aligned}$$

Számítások

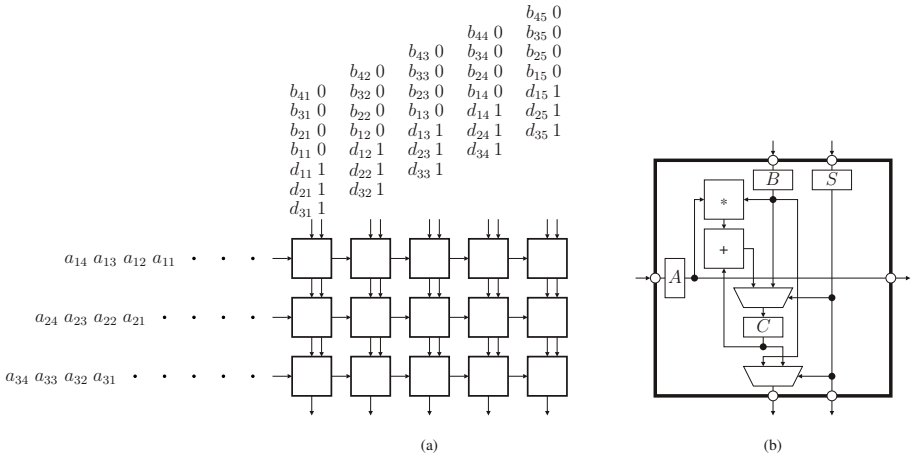
$$\begin{aligned} &a(i,j,k) = a(i,j-1,k) && 1 \le i \le N_1,\ 1 \le j \le N_2,\ 1 \le k \le N_3 \,, \\ &b(i,j,k) = b(i-1,j,k) && 1 \le i \le N_1,\ 1 \le j \le N_2,\ 1 \le k \le N_3 \,, \\ &c(i,j,k) = c(i,j,k-1) && 1 \le i \le N_1,\ 1 \le j \le N_2,\ 1 \le k \le N_3 \\ &\quad + a(i,j-1,k) \cdot b(i-1,j,k). \end{aligned} \tag{7.35}$$

Átvezetés

$$\begin{aligned} &b(i,j,k) = c(i,j,k-1) && 1 \le i \le N_1,\ 1 \le j \le N_2,\ 1 + N_3 \le k \le i + N_3 \,, \\ &c(i,j,k) = b(i-1,j,k) && 1 \le i \le N_1,\ 1 \le j \le N_2,\ 1 + N_3 \le k \le i - 1 + N_3 \,. \end{aligned}$$

Kimeneti műveletek

$$c_{1+N_1+N_3-k,j} = b(i,j,k) \qquad i = N_1,\ 1 \le j \le N_2,\ 1 + N_3 \le k \le N_1 + N_3 \,.$$



7.12. ábra. Mátrixszorzás téglalap alakú szisztolikus ráccsal, az eredmények kivitelével és osztott vezérléssel: **(a)** a rács felépítése; **(b)** cellaszerkezet.

7.4.4. Osztott vezérlés

A 7.11. ábrán bemutatott módszernek a következő hátrányai vannak:

1. Globális vezérlőjeleket használunk. Ez magas fokú technikai pontosságot igényel.
2. Minden cellának egy számlálóra van szüksége, mely igen nagy ráfordítást igényel.
3. A számlálók kezdőértéke nem azonos minden cella esetében. Ezért minden cellát egyénileg kell megtervezni és megvalósítani.
4. Egy új feladat bevitelével meg kell várni, amíg az utolsó eredmény elhagyja a szisztolikus rácsot.

Ezek a hátrányok eltűnnek, ha a vezérlőjeleket az adatokhoz hasonlóan továbbítjuk, tehát osztott vezérlést alkalmazunk. Megtartjuk ugyan a 7.11(b) ábrán látható módon az B és C regiszterek multiplexerekhez való kapcsolását, nem hozunk viszont létre vezérlőjeleket a cellákon belül; a globális visszaállító jeltől is eltekintünk. Ehelyett kívülről vezetjük be a cellákba a szükséges vezérlőjeleket, egy (csupán 1 bit szélességű) S pótregiszterben tároljuk, majd onnan a szomszédos cellákba megfelelő módon továbbítjuk. A tulajdonképpeni vezérlőjel létrehozását a gazdagép veszi át, a betáplálás kizárólag a peremcellákon keresztül történik. A 7.12(a) ábra az ehhez szükséges rácsfelépítést, a 7.12(b) ábra pedig a módosított cellaszerkezetet mutatja be.

Az adattovábbító üzemmódba való átkapcsolás egy oszlop cellái esetén lefele haladva egy-egy időszeletnyi különbséggel következik be. Ezért elegendő csupán az S regiszter okozta késleltetés.

A számítási üzemmódba való visszaállítás ugyanazon vezérfonal mentén következik be, tehát ugyanúgy cellánként egy időszeletnyi késleltetéssel történik. Mivel a c_{ij} eredmények csak fele akkora sebességgel mozognak lefelé, a cellák visszaállításával megfelelő ideig várni kell: ha egy cella a t időszeletben lett számítási üzemmódba állítva, akkor a $t + N_3$ időszeletben kapcsol át adattovábbító üzemmódba, és a $t + N_1 + N_3$ időszeletben ismét

visszakapcsol számítási üzemmódba.

Amint látjuk, a szisztolikus rácsok osztott vezérlése a lokális/globális vezérléstől makroszkopikus időtényezőben különbözik. Míg a 7.12. ábrán látható szisztolikus rács minden $N_1 + N_3$ időszeletben egy új (7.34) feladat megoldásába kezdhet, ugyanez a 7.11. ábrán látható szisztolikus rács esetében csak minden $2N_1 + N_2 + N_3 - 2$ időszeletben lehetséges. Az $N_1 + N_3$, illetve $2N_1 + N_2 + N_3 - 2$ időkülönbséget ***periódusnak*** nevezzük, inverzét pedig ***teljesítménynek***.

A (7.36) rendszer az osztott vezérlés és számítás közötti formális összefüggéseket írja le. Egy tetszőleges hosszúságú, lehető legszorosabban egymást követő mátrixszorzásokból álló sorozatból indulunk ki, a bevezetett n iterációs változó ezért korlátlan.

Vezérlés

$$
\begin{aligned}
&s(i,j,k,n) = 0 && i = 0, 1 \le j \le N_2, 1 \le k \le N_3 \,. \\
&s(i,j,k,n) = 1 && i = 0, 1 \le j \le N_2, 1 + N_3 \le k \le N_1 + N_3 \,. \\
&s(i,j,k,n) = s(i-1,j,k,n) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_1 + N_3 \,.
\end{aligned}
$$

Adatbevitel

$$
\begin{aligned}
&a(i,j,k,n) = a^n_{ik} && 1 \le i \le N_1, j = 0, 1 \le k \le N_3 \,, \\
&b(i,j,k,n) = b^n_{kj} && i = 0, 1 \le j \le N_2, 1 \le k \le N_3 \,, \\
&b(i,j,k,n) = d^{n+1}_{N_1+N_3+1-k,j} && i = 0, 1 \le j \le N_2, 1 + N_3 \le k \le N_1 + N_3 \,.
\end{aligned}
$$

Fedőnévvel rendelkező változók

$$
c(i,j,k,n) = c(i,j,N_1+N_3,n-1) \qquad 1 \le i \le N_1, 1 \le j \le N_2, k = 0 \,.
$$

Adatokkal végzett műveletek

$$
\begin{aligned}
&a(i,j,k,n) = a(i,j-1,k,n) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_1 + N_3 \,, \\
&b(i,j,k,n) = \begin{cases} b(i-1,j,k,n) & : s(i-1,j,k,n) = 0 \\ c(i,j,k-1,n) & : s(i-1,j,k,n) = 1 \end{cases} && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_1 + N_3 \,. \\
&c(i,j,k,n) = \begin{cases} c(i,j,k-1,n) \\ \quad + a(i,j-1,k,n) \\ \quad \cdot b(i-1,j,k,n) & : s(i-1,j,k,n) = 0 \\ b(i-1,j,k,n) & : s(i-1,j,k,n) = 1 \end{cases} && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_1 + N_3 \,,
\end{aligned}
$$

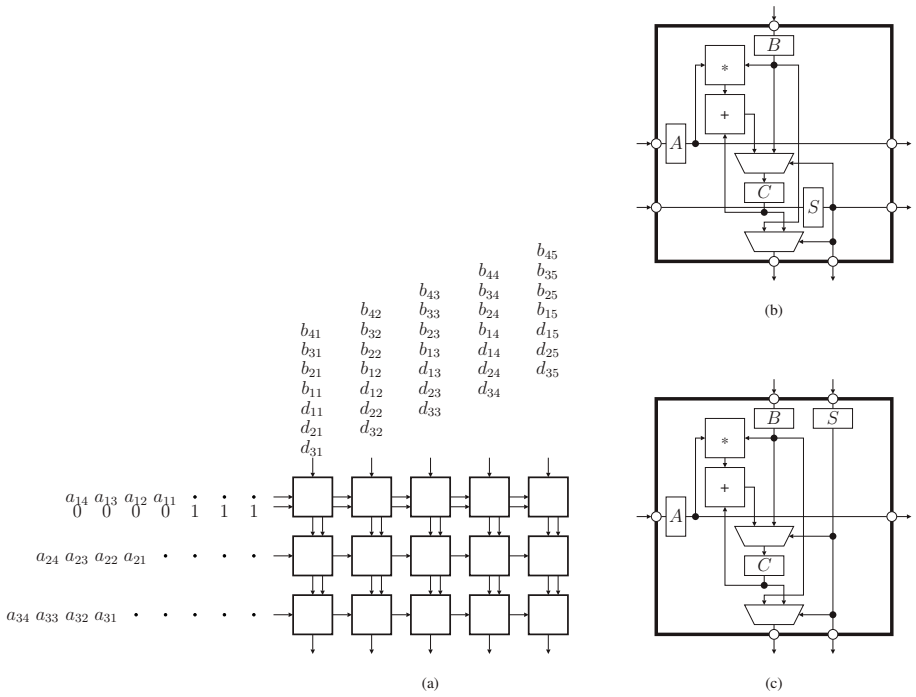
Adatkivitel

$$
c^n_{1+N_1+N_3-k,j} = b(i,j,k,n) \qquad i = N_1, 1 \le j \le N_2, 1 + N_3 \le k \le N_1 + N_3 \,. \tag{7.36}
$$

A (7.37) képlet a hozzátartozó tér-idő-mátrixot mutatja, amelyben az egyik elem nem konstans, hanem a feladat paramétereitől függ.

$$
T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & N_1 + N_3 \end{pmatrix} . \tag{7.37}
$$

Feltűnik, hogy az egy sorhoz tartozó cellák esetében jobbra haladva szintén egy időszeletnyi különbséggel kell átkapcsolni ezeket. A szisztolikus rács teljes szabályosságának követelményét figyelembe véve, ez a körülmény felhasználható arra, hogy a vezérlőjele-



7.13. ábra. Mátrixszorzás téglalap alapú szisztolikus ráccsal, eredménytovábbítással és osztott vezérléssel: **(a)** A rács felépítése; **(b)** a felső perem cellái; **(c)** a fennmaradó területek cellái.

ket csak az $(1,1)$ cellán keresztül tápláljuk be, felszabadítva ezáltal a gazdagépet. Ekkor a vezérlést a következőképpen módosítanánk:

Vezérlés

$$\begin{aligned} &s(i,j,k,n) = 0 && i = 1, j = 0, 1 \le k \le N_3 \ , \\ &s(i,j,k,n) = 1 && i = 1, j = 0, 1 + N_3 \le k \le N_1 + N_3 \ , \\ &s(i,j,k,n) = s(i-1,j,k,n) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_1 + N_3 \ , \\ &\ldots \end{aligned} \tag{7.38}$$

Fedőnévvel rendelkező változók

$$s(i,j,k,n) = s(i+1,j-1,k,n) \qquad i = 0, 1 \le j \le N_2, 1 \le k \le N_1 + N_3 \ .$$

...

A 7.13. ábra e módosítás eredményét mutatja. Két cellatípus létezik tehát: a szisztolikus rács felső szélén található cellák (7.13(b) ábra), és az összes többi cella (7.13(c) ábra). A kapcsolatszerkezet is csupán igen csekély eltérést mutat s szisztolikus rács felső szélén, a szabályos területhez képest.

7.4.5. A cellaprogram, mint lokális szemléletmód

Egy cella működését egy ***cellaprogrammal*** is kifejezhetjük. Ez különösen akkor érdekes, ha rendelkezésünkre áll egy programozható szisztolikus rács, melynek celláit valójában egy ismételt módon végrehajtott program vezérli.

Akárcsak a globális nézetet, azaz a szisztolikus rács architektúráját, a lokális nézetet, azaz a cellaprogramot is a *tér-idő-leképezés* határozza meg. Ez azonban csak implicit alakban adódik, ezért először matematikai transzformációval explicit alakra kell átalakítani, amely aztán alkalmas lesz cellaprogramnak.

A programváltozók példányait általános alakban ***indexkifejezések*** írják le, melyek az iterációs változókra hivatkoznak. Tekintsük például a következő egyenletet a (7.3) rendszerből:

$$c(i,j,k) = c(i,j,k-1) + a(i,j-1,k) \cdot b(i-1,j,k) \quad 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 \ .$$

A c programváltozók $c(i,j,k-1)$ példánya az i, j és $k-1$ indexkifejezésekkel rendelkezik, melyek az i,j,k iterációs változók függvényeiként is felfoghatók.

Amint láthattuk, a (7.11) tér-idő-leképezését alkalmazva a (7.13)-beli T transzformációs mátrixszal, az értéktartomány (i,j,k) iterációs vektorainak halmaza az (x,y,t) tér-idő-koordináták halmazába megy át:

$$\begin{pmatrix} x \\ y \\ t \end{pmatrix} = T \cdot \begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \end{pmatrix} . \tag{7.39}$$

Mivel minden cellát az (x,y) térbeli koordinátája jellemez, és a hozzá tartozó cellaprogramnak a múló t időre is hivatkoznia kell, a programváltozók indexkifejezéseiben előforduló i,j,k iterációs változók nem használhatók többé, és át kell írni őket az új, x,y,t koordinátákra. Ehhez az i,j,k iterációs változókat a (7.39)-beli tér-idő-leképezés inverzének segítségével fejezzük ki, az (x,y,t) tér-idő-koordináták függvényében.

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = T^{-1} \cdot \begin{pmatrix} x \\ y \\ t \end{pmatrix} = \frac{1}{3} \cdot \begin{pmatrix} -1 & -2 & 1 \\ -1 & 1 & 1 \\ 2 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ t \end{pmatrix} . \tag{7.40}$$

Egy ilyen inverz leképezés akkor létezik, ha a tér-idő-leképezés injektív az értéktartományon – és ennek mindig igaznak kell lennie, mert különben ugyanabban az időszeletben egyetlen cellában egyszerre több számítást kellene végrehajtani. A példában az invertálhatóságot a négyzetes, nem szinguláris T mátrix az értéktartományra való hivatkozás nélkül is biztosítja. A π idővektorra és u projekciós irányra vonatkozóan elegendő a következő tulajdonság: $\pi \cdot u \neq 0$.

Az iterációs változók tér-idő-koordinátákkal való lecserélésével, mely az ***értéktartományok transzformációjaként*** is értelmezhető, általában kevésbé szép, új index-kifejezéseket kapunk. A $c(i,j,k-1)$ tehát így alakul:

$$c((-x-2\cdot y+t)/3, (-x+y+t)/3, (2\cdot x+y+t)/3) \ .$$

Az ***indexhalmazok*** egy utólagos ***transzformációjával*** viszont átnevezhetjük a programváltozók példányait úgy, hogy a cellára illetve időre történő hivatkozások tisztábban kivehetők

legyenek. Különösen ajánlatos az egyenletrendszert ismét ***kimeneti normál formába*** hozni, azaz a t időszeletben az (x, y) cellában kiszámított eredményeket a programváltozók (x, y, t) példányaiként jelölni.

Ennek az eljárásnak a megértését leginkább egy absztrakt matematikai formalizmussal érhetjük el, melyet végül a mi speciális helyzetünkre alkalmazunk. Legyen egy kvantifikált egyenlet r és s programváltozókkal, valamint S értéktartománnyal a következőképpen adott:

$$r(\psi_r(v)) = \mathcal{F}(\ldots, s(\psi_s(v)), \ldots), \qquad v \in S \ . \tag{7.41}$$

A ψ_r, valamint ψ_s indexfüggvények a programváltozók példányait indexkifejezéspárokként állítják elő.

Az értéktartomány egy S-en injektív φ függvény segítségével történő transzformációja által a (7.41) a következőképpen alakul át:

$$r(\psi_r(\varphi^{-1}(e))) = \mathcal{F}(\ldots, s(\psi_s(\varphi^{-1}(e))), \ldots), \qquad e \in \varphi(S) \ , \tag{7.42}$$

ahol φ^{-1} egy függvény, amely $\varphi(S)$-en a φ inverz függvényét képezi. Az új indexfüggvények $\psi_r \circ \varphi^{-1}$, valamint $\psi_s \circ \varphi^{-1}$.

Az indexhalmazok transzformációinak nincs közük az értéktartományokhoz és minden egyes programváltozóra külön végrehajthatók, mivel csak ennek az egy programváltozónak a példányait nevezik át következetes módon. A ϑ_r és ϑ_s átnevezésekkel (7.42) a következőképpen alakul:

$$r(\vartheta_r(\psi_r(\varphi^{-1}(e)))) = \mathcal{F}(\ldots, s(\vartheta_s(\psi_s(\varphi^{-1}(e)))), \ldots), \qquad e \in \varphi(S) \ . \tag{7.43}$$

Amennyiben a kimeneti normál formát szeretnénk megkapni, a $\vartheta_r \circ \psi_r \circ \varphi^{-1}$-nek az identitásfüggvénynek kell lennie.

A (példában látható) legegyszerűbb esetben ψ_r mindig az identitásfüggvény és ψ_s egy $\psi_s(v) = v - d$ alakú affin leképezés, konstans d-vel, a már ismert függőségi vektorral. ψ_r ugyanúgy fejezhető ki $d = \vec{0}$-ral. Az értéktartományok transzformációja a $\varphi(v) = T \cdot v$ tér-idő-leképezéssel történik, ahol T invertálható mátrix. Minden indextranszformáció esetében egyértelműen ($\vartheta = \varphi$)-t választjuk. Tehát a (7.43) a következőképpen alakul:

$$r(e) = \mathcal{F}(\ldots, s(e - T \cdot d), \ldots), \qquad e \in T(S) \ . \tag{7.44}$$

Egy *cellaprogram* előállításakor a végrehajtandó műveleteknek, az adatok forrásának, valamint az eredmények rendeltetési helyének (az assembly programokból ismert `opc, src, dst`) minden egyes időszeletben pontosan adottnak kell lennie.

Az éppen végrehajtandó művelet (`opc`) közvetlenül az $\mathcal{F}$ függvényből adódik. A vezérléssel ellátott cellák esetében meg kell még állapítani azt az időtartamot is, mely alatt ez a speciális $\mathcal{F}$ végrehajtódik. Ez meghatározható a térkoordináták függvényében, a $T(S)$ időtengelyre való levetítéssel, egy általános poliéderes S esetében például egy „Fourier-Motzkin elimináció” útján.

A (7.44) rendszerből adódik az új $T \cdot d$ függőségi vektor, amely két komponensből áll, egy térbeli (vektoriális), és egy időbeli (skaláris) részből. A Δz térbeli rész, mint különbségvektor leírja, hogy a szomszédos cellák melyikében számítottuk ki az operandust. Ezt az operandusoknak a z cellába történő bevitelével kapcsolatos információt közvetlenül átalakíthatjuk egy $-\Delta z$ pozíciójú csatorna-jelöléssé (`src`). Ennek megfelelően az operandusokat

kiszámító $z-\Delta z$ cella ezt az értéket egy Δz pozíciójú csatornán keresztül adja át (`dst`).

A $T\cdot d$ időbeli része a Δt időkülönbség által megadja, hogy mikor történt az operandusok kiszámítása. Ez az információ az olvasó z cella számára jelentéktelen, mivel a szomszédos cellákból mindig a közvetlenül megelőző időszelet kimenetét olvassa ki. Azonban ha $\Delta t > 1$, akkor az operandust abban a $z-\Delta z$ cellában, amelyben kiszámították, $\Delta t-1$ időegyszeleten keresztül tárolni kell. Ez megoldható például úgy, hogy a $z-\Delta z$ cella programja $\Delta t-1$ másolási műveletet hajt végre, melynek során az operandusok értékei $\Delta t-1$ regiszteren haladnak keresztül, míg a cella tényleges kimenetéhez érnek.

Ezt a módszert alkalmazva a (7.36)-re, a (7.37)-beli T-t használva a következőket kapjuk:

$$
\begin{aligned}
s(x,y,t) &= s(x-1,y,t-1)\ .\\
a(x,y,t) &= a(x,y-1,t-1)\ .\\
b(x,y,t) &= \begin{cases} b(x-1,y,t-1) & : s(x-1,y,t-1)=0\\ c(x,y,t-1) & : s(x-1,y,t-1)=1\ , \end{cases}\\
c(x,y,t) &= \begin{cases} c(x,y,t-1)\\ \quad + a(x,y-1,t-1)\\ \quad \cdot b(x-1,y,t-1) & : s(x-1,y,t-1)=0\\ b(x-1,y,t-1) & : s(x-1,y,t-1)=1\ . \end{cases}
\end{aligned}
\tag{7.45}
$$

Az n iterációs változónak itt csupán a be/kiviteli séma szempontjából van jelentősége és a transzformáció számára egy rögzített értékre állíthatjuk. A hozzá tartozó cellaprogram, mely minden időszeletben lefut, a következő.

Cellaprogram

```
1  S ← C(−1, 0)(0)
2  A ← C(0, −1)
3  B ← C(−1, 0)(1 : N)
4  C(1, 0)(0) ← S
5  C(0, 1) ← A
6  if S = 1
7     then C(1, 0)(1 : N) ← C
8          C ← B
9     else C(1, 0)(1 : N) ← B
10         C ← C + A · B
```

A csatornajelölések a cella lokális be/kimeneteire vonatkoznak. Alakjukat a csatornának a cella középpontjához viszonyított helyzetéből kapják. $\mathtt{C}(0,-1)$ a cella bal szélén helyezkedik el, $\mathtt{C}(0,1)$ a jobb szélen, $\mathtt{C}(-1,0)$ fent van, $\mathtt{C}(1,0)$ pedig lent. A csatornajelölés után megadható még egy bit-tartomány: $\mathtt{C}(-1,0)(0)$ kizárólag a csatorna 0. bitjét jelenti, $\mathtt{C}(-1,0)(1:N)$ ugyanannak a csatornának az 1-től N-ig terjedő bitjeit. Az `A`, `B`, ... jelölések a cella regisztereit jelentik.

A (7.12)-beli T-t (7.35)-re megfelelően alkalmazva a következőket kapjuk:

$$\begin{aligned}
&a(x,y,t) = a(x,y-1,t-1) && 1+x+y \le t \le x+y+N_3\ ,\\
&b(x,y,t) = b(x-1,y,t-1) && 1+x+y \le t \le x+y+N_3\ ,\\
&c(x,y,t) = c(x,y,t-1) && 1+x+y \le t \le x+y+N_3\\
&\quad + a(x,y-1,t-1)\cdot b(x-1,y,t-1)\ . && \\
&b(x,y,t) = c(x,y,t-1) && x+y+1+N_3 \le t \le 2*x+y+N_3\ ,\\
&c(x,y,t) = b(x-1,y,t-1) && x+y+1+N_3 \le t \le 2*x+y\\
& && \quad -1+N_3\ .
\end{aligned} \tag{7.46}$$

Szépen kidomborodnak tehát az osztott vezérlés előnyei. A (7.45)-ben leírt cellaprogram egy tetszőleges t időszeletre vonatkozik, nem kell tehát globális vezérlőjelekre reagáljon, nincs szüksége számlálóregiszterre, nincsenek számlálóműveletek és a helyi cellakoordinátákat sem kell kódolni.

Gyakorlatok

7.4-1. Adjuk meg a be/kimeneti sémákat két, lehető legszorosabban egymást követő számítás végrehajtására a (7.35) rendszer alapján, a 7.11., illetve 7.12. ábrán bemutatott szisztolikus rácsokra.

7.4-2. Hogyan kellene a 7.12. ábrán látható szisztolikus rácsot módosítani ahhoz, hogy hatékonyan támogassa a mátrixszorzatok kiszámítását $M_1 < N_1$ vagy $M_2 < N_2$ paraméterek esetén?

7.4-3. Hogy néz ki a cellaprogram a 7.3. ábrán látható szisztolikus rács esetében?

7.4-4.⋆ Mekkora teljesítményt ér el a 7.3. ábrán látható szisztolikus rács N_1, N_2, N_3 konkrét értékeire? Hát általános N_1, N_2, N_3-ra?

7.4-5.⋆ Módosítsuk a 7.1. ábrán látható szisztolikus rácsot úgy, hogy a stacionárius változók a számítás befejezése után jobb-alsó irányba (azaz a (i, j) cellától a $(i+1, j+1)$ cella felé) vezető járulékos kapcsolatokon keresztül legyenek kivezetve. Adjunk meg egy, a (7.35)-nek megfelelő értékadásmentes rendszert, mely leírást ad a hozzá tartozó viselkedésről. Hogyan néz ki a be/kimeneti séma? Milyen periódus érhető el?

7.5. Lineáris szisztolikus rácsok

A fenti alfejezetekben leírt megfontolások kétdimenziós szisztolikus rácsokra vonatkoznak. Azonban egydimenziós szisztolikus rácsokra is átültethetjük őket.

A két forma közötti lényeges különbség a szisztolikus rács *peremére* vonatkozik. Az egydimenziós szisztolikus rácsokat egyrészt felfoghatjuk úgy, mint amelyek kizárólag peremcellákból állnak, az adatbevitel a gazdagépről, illetve a kimenetek továbbítása a gazdagépnek minden további intézkedés nélkül lehetséges. Másrészt rendelkeznek egy teljes dimenzióval és egy csupán formális dimenzióval. A lineáris szisztolikus rács működési iránya menti kommunikációjában adott esetben hasonló kérdések merülnek fel, mint a 7.3.5. pontban. Végül, a lineáris szisztolikus rács pereme teljesen másként is definiálható, mégpedig úgy, mint ami csak a két végen elhelyezkedő cellából áll.

7.5.1. Mátrix és vektor szorzása

Ha a 7.1. ábrán az N_1 vagy N_2 feladat-paraméterek egyikének értéke 1, a mátrixszorzás átalakul mátrix és vektor közötti szorzássá: balról $N_1 = 1$-re, illetve jobbról $N_2 = 1$-re. A kétdimenziós szisztolikus rács ekkor egydimenzióssá ***fajul el.*** A szorzandó vektort, mint bemeneti adatfolyamot, a lineáris szisztolikus rács egyik végcelláján keresztül vezetjük be. Ezzel egy időben a mátrix elemei a rács hosszanti oldalán kerülnek be.

Akárcsak a teljes mátrixszorzás esetében, az eredmények stacionárius módon jönnek létre. Ezeket viszont kivezethetjük a rács hosszában az egyik végcelláján keresztül, vagy átadhatjuk közvetlenül a számítást végző cellákból a gazdagépnek. Ez különböző vezérlőmechanizmusokat, idősémákat és teljes futási időket eredményez.

Lehetséges lett volna-e az összes bemeneti adatot a végcellákon keresztül bevezetni? Semmiképp sem, ha a teljes futási időnek $\Theta(N)$-nek kell lennie. A beviteli mátrixnak $\Theta(N^2)$ eleme van, tehát $\Theta(N)$ elemet kell időszeletenként bevinni. Egy időszeleten belül a végcellákba bemenő adatok száma azonban korlátozott. Az esetünkben $\Theta(N)$ nagyságrendű ***be/kimeneti adatsebesség*** előre kizár bizonyos döntéseket.

7.5.2. Rendezés

Rendezéskor a feladat az, hogy egy teljesen rendezett G alaphalmazból vett $\{x_1, \ldots, x_N\}$ elemekből álló halmazt $\{m_i\}_{i=1,\ldots,N}$ növekvő sorrendbe állítsunk, vagyis hogy $m_i \le m_k$ érvényes legyen minden $(i < k)$-ra. A feladatot a következőképpen fogalmazhatjuk meg értékadásmentes jelölést használva, ahol *MAX G* maximumát jelöli:

$$
\begin{array}{ll}
\textit{Bemeneti műveletek} & \\
x(i,j) = x_i & 1 \le i \le N, j = 0\ , \\
m(i,j) = MAX & 1 \le j \le N, i = j - 1\ . \\
\textit{Számítások} & \\
m(i,j) = \min\{x(i,j-1), m(i-1,j)\} & 1 \le i \le N, 1 \le j \le i\ , \\
x(i,j) = \max\{x(i,j-1), m(i-1,j)\} & 1 \le i \le N, 1 \le j \le i\ . \\
\textit{Kimeneti műveletek} & \\
m(i,j) = m_j & 1 \le j \le N, i = N\ .
\end{array}
\tag{7.47}
$$

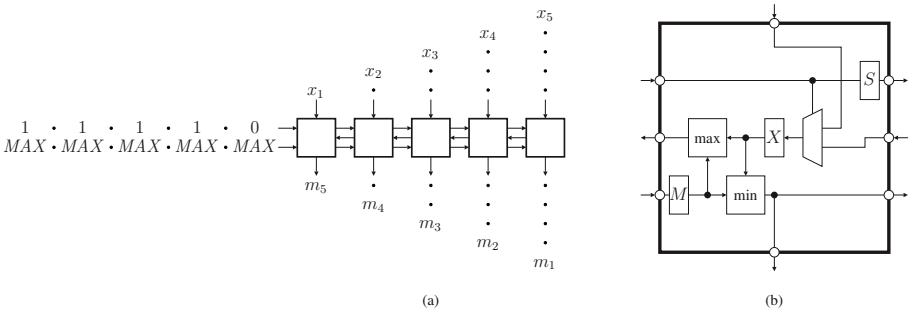
Egy $u = (1, 1)$ irány menti projekció segítségével a tér-idő-leképezésre

$$
\begin{pmatrix} x \\ t \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \end{pmatrix} \tag{7.48}
$$

a 7.14. ábrán látható egydimenziós szisztolikus rácsot kapjuk, mely a buborékrendezést valósítja meg.

Hasonlóképpen, az alábbi tér-idő-mátrix

$$
T = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \tag{7.49}
$$



7.14. ábra. „Buborékos rendezés” lineáris szisztolikus rács segítségével: **(a)** a rács felépítése be/kiviteli sémával; **(b)** Cellaszerkezet.

a beszúró rendezést megvalósító lineáris szisztolikus rácshoz vezetne, míg végül a

$$T = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \tag{7.50}$$

tér-idő mátrix a kiválasztó rendezést valósítaná meg.

A rendezési feladatnál $\Theta(N)$ bemeneti adatunk, $\Theta(N)$ kimeneti adatunk, és $\Theta(N)$ időszeletünk van. Ez $\Theta(1)$ be/kimeneti adatsebességet eredményez. Ellentétben a 7.5.1. pontban bemutatott mátrix-vektor szorzással, a be/kimeneti adatsebesség a kommunikációt elvileg itt még kizárólag csak a lineáris szisztolikus rács végpontjain levő cellákon keresztül engedi meg.

Mindazonáltal a rendezés mindhárom leírt változatában a bemenetek az összes cellán keresztül történnek: a buborékos rendezésnél csak a rendezni való elemek, a kiválasztásos rendezésnél ehhez hozzájönnek még a konstans *MAX* értékek, a beszúrásos rendezésnél csak a konstans értékek. Az utóbbiakat ugyan nem kötelező bemeneti adatként megadni, hanem létrehozhatók közvetlenül a cellákban, vagy kiolvashatók egy csak olvasható (ROM) memóriából is.

Mindhárom változat cellavezérlést igényel: a beszúró-, illetve kiválasztásos rendezés azért, mert stacionárius változói vannak, a buborékos rendezés azért, mert a bemeneti adatok és a kiszámított értékek feldolgozása között átkapcsolásra van szükség.

7.5.3. Lineáris egyenletrendszer alsó háromszögmátrixszal

A (7.51)-beli képletek egy lokalizált algoritmust írnak le az $A \cdot x = b$ lineáris egyenletrendszer megoldására, ahol az $(N \times N)$-es A mátrix egy alsó háromszögmátrix.

Bemeneti műveletek

$$\begin{aligned} &a(i,j) = a_{i,j+1} && 1 \le i \le N, 0 \le j \le i-1\ , \\ &u(i,j) = b_i && 1 \le i \le N, j = 0\ . \end{aligned}$$

Számítások

$$\begin{aligned} &u(i,j) = u(i,j-1) - a(i,j-1) \cdot x(i-1,j) && 2 \le i \le N, 1 \le j \le i-1\ , \\ &x(i,j) = u(i,j-1)/a(i,j-1) && 1 \le i \le N, j = i\ , \\ &x(i,j) = x(i-1,j) && 2 \le i \le N-1, 1 \le j \le i-1\ . \end{aligned} \tag{7.51}$$

Kimeneti műveletek

$$x_i = x(i,j) \qquad 1 \le i \le N, j = i\ .$$

Az összes eddig tanulmányozott példában, a másolási műveletektől eltekintve, az értéktartomány egy hordozópontjára ugyanazok a számítási műveletek vonatkoztak: szorzás és összeadás egymás utáni végrehajtása a szorzás-algoritmusok esetében, valamint minimum és maximum párhuzamos végrehajtása a rendezési algoritmusok esetében. A (7.51) rendszerben ezzel ellentétben egyes hordozópontokra szorzás és kivonás, míg más hordozópontok esetében csak osztás történik.

A (7.51) lineáris szisztolikus rácsra való levetítése során, a választott projekció-iránytól függően ugyanolyan, illetve különböző cellaműködést kapunk. Az $u = (1,1)$ projekció-iránnyal (és csakis ezzel) egyetlenegy osztóval rendelkező cellát kapunk, az összes többi szorzó- és kivonó egységgel rendelkezik. Az $u = (1,0)$ vagy $u = (0,1)$ mentén történő projekció esetén csupa egyforma cellát kapunk, melyek osztóval, szorzóval és kivonóval is rendelkeznek. Az $u = (1,-1)$ projekció-irány egy, három különböző cellatípussal rendelkező, lineáris szisztolikus rácsot eredményez: mindkét végen levő cellában csak osztóra van szükség. Az összes többi cella kap egy szorzó- és kivonó egységet, mindamellett egy osztóval rendelkező és egy osztó nélküli cella váltja egymást. A projekció egy bizonyos módja egy szisztolikus rácsban ***inhomogenitáshoz*** vezethet (ami lehet hasznos – vagy sem).

Gyakorlatok

7.5-1. Adjunk meg a mátrix és vektor szorzásának a 7.5.1. pontban leírt változataira (az eredmények megadása egy végcellán keresztül, illetve az összes cellán keresztül) egy alkalmas rácsfelépítést be/kiviteli sémával, cellaszerkezettel, valamint vezérlési mechanizmussal együtt.

7.5-2. Tanulmányozzuk további projekciós irányok hatását a (7.51) rendszerre.

7.5-3. Adjuk meg a 7.5.2. pontban leírt beszúró, illetve kiválasztásos rendező eljárásokhoz a hozzájuk tartozó szisztolikus rácsokat – beleértve a cellaszerkezetet is.

7.5-4.⋆ Hogyan működtethető a 7.14. ábrán látható, buborékrendezésre használt szisztolikus rács akár vezérlés nélkül is, a bemeneti adatfolyam ésszerű megformálásával?

7.5-5.⋆ Mi a szerepe a *MAX* érték használatának a (7.47) rendszerben? Hogyan lehetne (7.47)-et kifejezni a e konstans érték használata nélkül? Milyen következményei lennének ennek a leírt szisztolikus rácsokra nézve?

Feladatok

7-1. Szalagmátrix algoritmusok
A 7.1. és 7.2. alfejezetekben, valamint a 7.5.1. és 7.5.3. pontokban mindig ***sűrű*** mátrixokból indultunk ki, azaz minden a_{ij} mátrixelem nullától különböző érték (az alsó háromszögmátrix esetében a főátló feletti elemek értéke ugyan mind nulla, de ezek nem képeznek bemenetet az említett algoritmus számára).

Ezzel szemben a gyakorlatban gyakran találkozunk ***szalagmátrixokkal.*** Ezeknél a főátló körüli keskeny sávot kivéve, a legtöbb átló nulla értékekkel van feltöltve. Formálisan fennáll tehát, hogy $a_{ij} = 0$ minden i, j-re, melyre $i - j \geq K$ vagy $j - i \geq L$, ahol K és L pozitív egész számok. Tehát a ***sávszélesség,*** vagyis azon átlók száma, amelyekben a nullától különböző elemek megengedettek, $K + L - 1$.

Feltevődik tehát a kérdés, hogy egy vagy több bemeneti mátrix sávszerkezetét ki lehet-e használni a szisztolikus számítások optimalizálására. Egyrészt fennáll annak a lehetősége, hogy elhagyjuk azokat a cellákat, melyek soha nem végeznek hasznos munkát. Egy másik optimalizálási lehetőség lehetne a be/kimeneti adatfolyam lerövidítése, a teljes futási idő lerövidítése vagy a teljesítmény növelése.

Tanulmányozzuk, hogyan optimalizálhatók a fejezetben bemutatott szisztolikus rácsok ebből a szempontból.

Megjegyzések a fejezethez

A szisztolikus rács fogalmát Kung és Leiserson vezette be, e területen úttörőnek számító cikkükben [268].

Karp, Miller és Winograd az egyenletes rekurzív egyenletek területén végeztek úttörő munkát [242].

Rao doktori értekezése [383] és Quinton munkái [379] is lényeges ötletekkel járultak hozzá a szisztolikus rácsok módszeres tervezéséhez.

Teich és Thiele [463] cikkükben rámutatnak, hogy a cellavezérlés hasonló módszerekkel vezethető le formálisan, mint az általános rácsfelépítés és a normális cellafunkcionalitás.

Darte, Robert és Vivien modern könyve [99] a fordítóprogramok kifinomult módszereit kapcsolja össze a szisztolikus rácsokkal, és alaposan foglalkozik az adatfüggőségek vizsgálatával.

A szalagmátrixokra vonatkozó feladat Kung és Leiserson cikkéből [268] származik.

A szisztolikus rendszerek területén a legátfogóbb áttekintést mind a mai napig a [509] monográfia nyújtja.

Minden szisztolikus rács modellezhető sejtautomataként. Egy cella regiszterei képezik a cella állapotát. Szükség van tehát egy faktorizált állapottér használatára. Amennyiben a szisztolikus rács különböző típusú cellákkal rendelkezik (például változó cellafunkcionalitás vagy pozíciófüggő cellavezérlés révén), ez az állapottér egy újabb komponensével írható le.

Minden szisztolikus algoritmus felfogható olyan PRAM-algoritmusként, melynek viselkedése időben állandó. Ekkor egy szisztolikus cella minden regiszterét egy PRAM-memóriacella valósítja meg, mely kizárólag ezt a célt szolgálja. Mivel minden időszelet-

ben pontosan egy szisztolikus cella olvas ebből a regiszterből, és ugyanakkor pontosan egy szisztolikus cella ír ebbe a regiszterbe, megfelelő az EREW-PRAM-modell.

A szisztolikus rendszerek a Lynch [301] által leírt szinkronizált hálózatok speciális esetei. A futási idő azonos a két rendszerben. A szisztolikus rendszerekben az üzenetek számát nem szokás vizsgálni. A szisztolikus rendszerekben gyakran megkívánt – peremen át történő be/kivitel szinkronizált hálózatokkal jól modellezhető. A hibák fogalma a szisztolikus rendszerek vizsgálatához nem szükséges.

Részletesen foglalkozik a szisztolikus rendszerekkel Sima, Fountain és Kacsuk könyve, amely magyarul is megjelent [432].

III. FOLYTONOS OPTIMALIZÁCIÓ

Előszó

Ebben a részben a folytonos optimalizáció néhány területét tekintjük át.

Az első kötetbe a játékelmélettel foglalkozó *nyolcadik* fejezet került: ebben a fejezetben a véges játékokat, a folytonos játékokat és az oligopol játékokat elemezzük.

A második kötetben fog megjelenni a sorbanállási modellekkel és algoritmusokkal, valamint a belsőpontos lineáris programozási feladatokkal foglalkozó rész.

8. Játékelmélet

A műszaki és a gazdasági életben gyakori az olyan helyzet, melyben több döntéshozó egymással ellentétes érdekeit kell figyelembe venni egyidejűleg, és a helyzet alakulása függ a döntéshozók döntéseitől. Az ilyen helyzetek elemzésére az egyik legnépszerűbb módszer és modell a játékelméleten alapul.

Jelölje N a döntéshozók (a továbbiakban ***játékosok***) számát, és minden $k = 1, 2, \ldots, N$ számra legyen $\mathcal{P}_k$ a k-adik játékos és S_k a $\mathcal{P}_k$ játékos ***megengedett akcióinak halmaza.*** Az $s_k \in S_k$ elemeket $\mathcal{P}_k$ ***stratégiáinak*** nevezzük. S_k a $\mathcal{P}_k$ játékos ***stratégiahalmaza.*** A játék tetszőleges lejátszásában minden játékos választ egy stratégiát, ekkor az $\mathbf{s} = (s_1, s_2, \ldots, s_N)$ vektort a játékosok ***szimultán stratégiavektorának*** hívjuk, ahol $s_k \in S_k$, $k = 1, 2, \ldots, N$. Minden játékos megfeleltet minden $\mathbf{s} \in S = S_1 \times S_2 \times \cdots \times S_N$ szimultán stratégiavektornak egy valós számot. Ez a valós szám minden játékos esetében tekinthető úgy, mint egy hasznossági függvény értéke, mely értéket a játékos a játék adott kimeneteléhez hozzárendeli. Ez a hasznossági függvény tükrözi az adott játékos értékelését a kimenetelekről. Legyen $\mathcal{P}_k$ tetszőleges játékos, ekkor ha $f_k(\mathbf{s})$ jelöli ezt az értéket, akkor az $f_k : S \to \mathbb{R}$ függvényt a $\mathcal{P}_k$ játékos ***kifizetőfüggvényének,*** az $f_k(\mathbf{s})$ értéket $\mathcal{P}_k$ ***kifizetésének,*** az $(f_1(\mathbf{s}), f_2(\mathbf{s}), \ldots, f_N(\mathbf{s}))$ vektort pedig ***kifizetővektornak*** nevezzük. Az N szám, az S_k halmazok, az f_k kifizetőfüggvények ($k = 1, 2, \ldots, N$) összessége teljes körűen meghatároz egy N személyes játékot. A továbbiakban az N személyes játék jelölésére a $G = \{N; S_1, S_2, \ldots, S_N; f_1, f_2, \ldots, f_N\}$ jelölést fogjuk használni.

A G játék megoldása a ***Nash-egyensúly*** (a továbbiakban röviden egyensúly), amely egy olyan $\mathbf{s}^\star = (s_1^\star, \ldots, s_N^\star)$ stratégiavektor, hogy minden k-ra

1. $s_k^\star \in S_k$;
2. Minden $s_k \in S_k$-ra

$$f_k(s_1^\star, s_2^\star, \ldots, s_{k-1}^\star, s_k, s_{k+1}^\star, \ldots, s_N^\star) \leq f_k(s_1^\star, s_2^\star, \ldots, s_{k-1}^\star, s_k^\star, s_{k+1}^\star, \ldots, s_N^\star)\,. \qquad (8.1)$$

Az 1. feltétel szerint az egyensúly k-adik komponense megvalósítható stratégia $\mathcal{P}_k$ számára, míg a 2. feltétel szerint egyik játékos sem tudja növelni a kifizetését az által, hogy egyoldalúan eltér az egyensúlyi stratégiától. Más szavakkal, minden játékosnak az az érdeke, hogy tartsa az egyensúlyt, hiszen ha bármely játékos eltér (egyoldalúan) az egyensúlytól, akkor annak a kifizetése nem nő.

		$\mathcal{P}_2$	
		N	C
$\mathcal{P}_1$	N	$(-2,-2)$	$(-10,-1)$
	C	$(-1,-10)$	$(-5,-5)$

8.1. ábra. Fogoly dilemma.

8.1. Véges játékok

Egy G játékot ***végesnek*** nevezünk, ha minden S_k stratégiahalmaz véges sok stratégiát tartalmaz[1]. A legismertebb kétszemélyes játék a ***fogoly dilemma,*** mely a következő példa tárgya.

8.1. példa. *Fogoly dilemma.* A két játékos két fogoly, akiket egy súlyos bűntett elkövetésének gyanújával vett őrizetbe a rendőrség, de az ügyészségnek még nincs elég bizonyítéka a vádemeléshez. A két fogoly két különböző cellában van fogva tartva, így nem tudnak egymással kommunikálni. Az ügyészség célja, hogy rávegye a foglyokat a hatóságokkal való együttműködésre, abból a célból, hogy a szükséges bizonyítékok meglegyenek a vádemeléshez. Tehát $N = 2$, a stratégiahalmazok mindkét játékos esetén kételeműek: együttműködni (C), vagy nem együttműködni (N). Mindkét játékossal külön-külön közölték, hogy ha csak az egyikük tesz vallomást, akkor a vallomást tevő csak 1 éves, míg a másik 10 éves börtönbüntetést kap. Ha mind a ketten vallomást tesznek, akkor mindketten 5 éves börtönbüntetést kapnak, míg ha mindketten megtagadják a vallomást, akkor csak egy kevésbé súlyos bűncselekmény miatt ítélik el őket, és mindketten 2 éves börtönbüntetést kapnak. Mindkét játékosnak az a célja, hogy minimalizálja a börtönben töltött időt, vagy ami ezzel ekvivalens, maximalizálja a börtönben töltött idő ellentettjét. A kifizetésvektorokat a 8.1. ábra tartalmazza, ahol $\mathcal{P}_1$ stratégiáit a sorok, míg $\mathcal{P}_2$ stratégiáit az oszlopok tartalmazzák, és minden kifizetésvektorban az első érték $\mathcal{P}_1$ kifizetése, míg a második szám $\mathcal{P}_2$ kifizetése.
A kifizetéseket összehasonlítva világos, hogy csak a (C, C) stratégiapáros lehet egyensúly, mivel

$$\begin{aligned} f_2(\mathrm{N,N}) = -2 &< -1 = f_2(\mathrm{N,C})\,, \\ f_1(\mathrm{N,C}) = -10 &< -5 = f_1(\mathrm{C,C})\,, \\ f_2(\mathrm{C,N}) = -10 &< -5 = f_2(C, C)\,. \end{aligned}$$

A (C, C) stratégiapáros tényleg egyensúly, mivel

$$\begin{aligned} f_1(\mathrm{C,C}) = -5 > -10 = f_1(\mathrm{N,C})\,, \\ f_2(\mathrm{C,C}) = -5 > -10 = f_2(\mathrm{C,N})\,. \end{aligned}$$

Ebben a játékban egyetlen egyensúly van.

Az egyensúly létezése általában nem garantált, és ha létezik egyensúly, akkor sem feltétlenül egyetlen.

[1] A játék definíciója szerint a játékosok száma is véges. *A fordító.*

		$\mathcal{P}_2$	
		N	C
$\mathcal{P}_1$	N	(1, 2)	(2, 1)
	C	(2, 4)	(0, 5)

8.2. ábra. Játék, melyben nincs egyensúly.

8.2. példa. *Játék, melyben nincs egyensúly.* Módosítsuk a 8.1. ábra kifizetéseit úgy, ahogy az a 8.2. ábrán látható. Könnyen látható, hogy a módosított játékban nincs egyensúly:

$$\begin{aligned} f_1(\text{N,N}) = 1 &< 2 = f_1(\text{C,N}) , \\ f_2(\text{C,N}) = 4 &< 5 = f_2(\text{C,C}) , \\ f_1(\text{C,C}) = 0 &< 2 = f_1(\text{N,C}) , \\ f_2(\text{N,C}) = 1 &< 2 = f_2(\text{N,N}) . \end{aligned}$$

Ha a kifizetések minden kimenetelre megegyeznek, akkor több egyensúly is van a játékban: minden stratégiapár egyensúly.

8.1.1. Leszámlálás

Jelölje továbbra is N a játékosok számát, és – a kényelmes jelölés kedvéért – jelölje $s_k^{(1)}, \ldots, s_k^{(n_k)}$ a $\mathcal{P}_k$ játékos megengedett stratégiáit. Tehát $S_k = \{s_k^{(1)}, \ldots, s_k^{(n_k)}\}$. Egy $\mathbf{s} = (s_1^{(i_1)}, \ldots, s_N^{(i_N)})$ stratégiavektor pontosan akkor egyensúly, ha minden $k = 1, 2, \ldots, N$ és minden $j \in \{1, 2, \ldots, n_k\} \setminus i_k$ esetén

$$f_k(s_1^{(i_1)}, \ldots, s_{k-1}^{(i_{k-1})}, s_k^{(j)}, s_{k+1}^{(i_{k+1})}, \ldots, s_N^{(i_N)}) \le f_k(s_1^{(i_1)}, \ldots, s_{k-1}^{(i_{k-1})}, s_k^{(i_k)}, s_{k+1}^{(i_{k+1})}, \ldots, s_N^{(i_N)}) . \qquad (8.2)$$

Vegyük észre, hogy véges játékok esetén (8.1) leegyszerűsödik (8.2)-re.

A leszámlálás alkalmazásakor ellenőrizzük a (8.2) egyenlőtlenséget az összes lehetséges $\mathbf{s}^\star = (s_1^{(i_1)}, \ldots, s_N^{(i_N)})$ N-esre úgy, hogy megvizsgáljuk a (8.2) egyenlőtlenség érvényességét minden k-ra, minden j-re. Ha az ellenőrzés sikeres, akkor $\mathbf{s}^\star$ egyensúly, ellenkező esetben $\mathbf{s}^\star$ nem egyensúly. Ha az ellenőrzés alatt egy rögzített $\mathbf{s}^\star$-ra találunk olyan k-t és j-t, hogy (8.2) nem teljesül, akkor $\mathbf{s}^\star$ nem egyensúly, így elhagyhatjuk az ellenőrzést minden további k-ra és j-re. Ez egy nagyon egyszerű algoritmus, mely $N + 2$, egymásba ágyazott, az $i_1, i_2, \ldots, i_N, k$ és j változókat használó ciklusból áll.

A szükséges összehasonlítások száma legfeljebb

$$\left(\prod_{k=1}^{N} n_k\right)\left(\sum_{k=1}^{N}(n_k - 1)\right) .$$

A gyakorlatban azonban az összehasonlítások száma ennél sokkal kisebb lehet, hiszen ha (8.2) nem teljesül valamilyen j-re, akkor az adott stratégiavektor esetén nem kell további összehasonlításokat tenni.

A következőkben a kétszemélyes játékokat (N=2) vizsgáljuk, és bevezetjük az $(n_1 \times n_2)$-es $\mathbf{A}^{(1)}$ és $\mathbf{A}^{(2)}$ mátrixokat, melyek elemei $\mathbf{A}^{(1)}(i, j) = f_1(i, j)$, illetve $\mathbf{A}^{(2)}(i, j) = f_2(i, j)$. Az

$\mathbf{A}^{(1)}, \mathbf{A}^{(2)}$ mátrixokat ***kifizetőmátrixoknak*** nevezzük. Az $\left(s_1^{(i_1)}, s_2^{(i_2)}\right)$ stratégiavektor pontosan akkor egyensúly, ha az (i_1, i_2) elem az $\mathbf{A}^{(1)}$ mátrixban a saját oszlopában, és az $\mathbf{A}^{(2)}$ mátrixban a saját sorában a legnagyobb. Ha $f_1 = -f_2$, akkor a játékot ***zérusösszegű*** játéknak nevezzük, és $\mathbf{A}^{(1)} = -\mathbf{A}^{(2)}$, tehát a játékot teljes körűen leírja $\mathbf{A}^{(1)}$, a $\mathcal{P}_1$ játékos kifizetőmátrixa. Ebben a speciális esetben az $(s_1^{(i_1)}, s_2^{(i_2)})$ stratégiavektor pontosan akkor egyensúly, ha az (i_1, i_2) elem az $\mathbf{A}^{(1)}$ mátrixban a saját oszlopában a legnagyobb, és a saját sorában a legkisebb. A zérusösszegű játékok egyensúlyára a ***nyeregpont*** elnevezés is használatos. Világos, hogy ebben az esetben az egyensúly megtalálása a leszámlálás módszerével leegyszerűsödik, hiszen csak egy mátrixszal kell foglalkozni.

8.1.2. Véges fákkal ábrázolt játékok

Számos véges játék rendelkezik azzal a tulajdonsággal, hogy ábrázolható olyan irányított véges fával, melynek a következő tulajdonságai vannak:

1. a fa gyökeres, és a játék ennél a csúcsnál kezdődik;
2. a fa minden csúcsához tartozik egy játékos, és ha a játék elér egy csúcsot, akkor a csúcshoz tartozó játékos kiválaszt egy élt, mely az adott csúcsból indul ki, így dönt arról, hogy miként folytatódik a játék. Ekkor a játék a kiválasztott él végpontjánál folytatódik;
3. minden levélhez tartozik egy valós, N komponensű vektor, mely vektor tartalmazza az egyes játékosok kifizetéseit, ha a játék ebben a levélben ér véget;
4. minden játékos ismeri a fát, tudja, hogy mely csúcsokhoz van rendelve, és tudja, hogy milyen kifizetések tartoznak az egyes levelekhez.

Például a sakk játék rendelkezik a fenti tulajdonságokkal. A sakkot két játékos játssza ($N = 2$), a csúcsok a lehetséges állások a sakktáblán, egyszer a világossal játszó, egyszer a sötéttel játszó játékos szempontjából. Adott csúcsból kiinduló élek jelentik azokat a lépéseket, melyeket a csúcshoz rendelt játékos (aki lép) megtehet. A levél olyan állás a sakktáblán, mellyel a játék véget ér. A kifizetések az $\{1, 0, -1\}$ halmazból valók, ahol 1 azt jelenti, hogy világos győzelmével, 0 azt jelenti, hogy döntetlennel, -1 azt jelenti, hogy sötét győzelmével ért véget a játék.

8.1. tétel. *Minden, véges fával ábrázolható játéknak van legalább egy egyensúlya.*

Bizonyítás. Abból a célból bizonyítjuk itt be ezt a tételt, hogy egy praktikus algoritmust mutassunk az egyensúly megtalálására. A bizonyítás indukción alapul, melyet annyiszor ismétlünk, amennyi a játék csúcsainak száma. Ha a játéknak csak egyetlen csúcsa van, akkor értelemszerűen ez az egyetlen csúcs egyensúly.

Tegyük fel, hogy a tétel igaz minden olyan játékra, ahol a csúcsok száma kisebb, mint n ($n \geq 2$), és nézzük azt a T_0 játékot, melynek n csúcsa van. Legyen r_0 a T_0 játék gyökere, és legyenek $r_1, r_2, \ldots, r_m$ ($m < n$) azok a csúcsok, melyeket él köt össze r_0-lal (r_0 gyerekei). Jelölje $T_1, T_2, \ldots, T_m$ a T_0 olyan diszjunkt részfáit, melyek gyökerei $r_1, r_2, \ldots, r_m$ a sorrendnek megfelelően (tehát r_2 T_2 gyökere). Ekkor minden részfának kevesebb, mint n csúcsa van, így mindegyiknek van egyensúlya (indukciós feltevés). Tegyük fel, hogy $\mathcal{P}_k$ tartozik az r_0 csúcshoz, legyenek $e_1, e_2, \ldots, e_m$ az egyes részfákhoz tartozó egyensúlyoknak a $\mathcal{P}_k$ játékoshoz tartozó kifizetései (tehát a T_m részfa egy egyensúlyában $\mathcal{P}_k$-nak e_m a kifize-

tése), és legyen $e_j = \max\{e_1, e_2, \ldots, e_m\}$. Ekkor a $\mathcal{P}_k$ játékos az r_j csúcsba lép a gyökérből, és azután folytatódik a játék a T_j részfában létező egyensúllyal[2]. ■

Az előző tétel bizonyítása egy dinamikus programozás típusú algoritmust sugall, mely algoritmust ***visszafelé indukciónak*** nevezünk. Az algoritmus kiterjeszthető általánosabb esetre is, mely esetben a fának véletlen csúcsai vannak, melyekből a játék egy rögzített, diszkrét eloszlásnak megfelelően véletlenszerűen folytatódik.

A fenti algoritmus a következőképpen mutatható be formálisan. Tegyük fel, hogy a csúcsok úgy vannak megszámozva (természetes számokkal), hogy ha a j az i csúcs rákövetkezője, akkor $i < j$. A gyökérnek a legkisebb, az 1-es számot kell kapnia, a legnagyobb szám (n) az egyik levélhez tartozik. Jelölje $J(i)$ azon j csúcsok halmazát, melyekre van olyan él, mely i-ből j-be megy (i gyerekeinek halmaza). Minden i levél esetén $J(i)$ üres halmaz. Jelölje továbbá $\mathbf{p}^{(i)} = (p_1^{(i)}, \ldots, p_N^{(i)})$ a kifizetővektort, mely az i levélhez tartozik. Végül, k_i-vel jelöljük azt a játékost, aki az i csúcshoz tartozik. Az algoritmus az utolsó csúcsnál (n-nél) kezdődik, majd visszafelé lépeget $n, n-1, n-2, \ldots, 2$ és 1 sorrendben. Vegyük észre, hogy n egy levél, és rendeljük hozzá a $\mathbf{p}^{(n)}$ vektort. Ha az algoritmusban a következő csúcs (i) is levél, akkor rendeljük hozzá a $\mathbf{p}^{(i)}$ vektort, ha i nem levél, akkor keressük meg a legnagyobb értékeket a $\mathbf{p}_{k_i}^{(j)}$, $j \in J(i)$ számok közül. Tegyük fel, hogy a legnagyobb érték a j_i csúcsban van, ekkor hozzárendeljük a $\mathbf{p}^{(i)} = \mathbf{p}^{(j_i)}$ vektort az i csúcshoz, és továbblépünk az $i-1$ csúcsba. Miután minden $\mathbf{p}^{(n)}$, $\mathbf{p}^{(n-1)}$, $\ldots$, $\mathbf{p}^{(2)}$ és $\mathbf{p}^{(1)}$ vektort meghatároztunk, a $\mathbf{p}^{(1)}$ vektor tartalmazza a kifizetéseket az egyensúlyban, és az egyensúlyi út a következő csúcsok mentén halad:

$$1 \to i_1 = j_1 \to i_2 = j_{i_1} \to i_3 = j_{i_2} \to \ldots ,$$

amíg egy levélbe el nem értünk. Így megkaptuk az egyensúlyi utat.

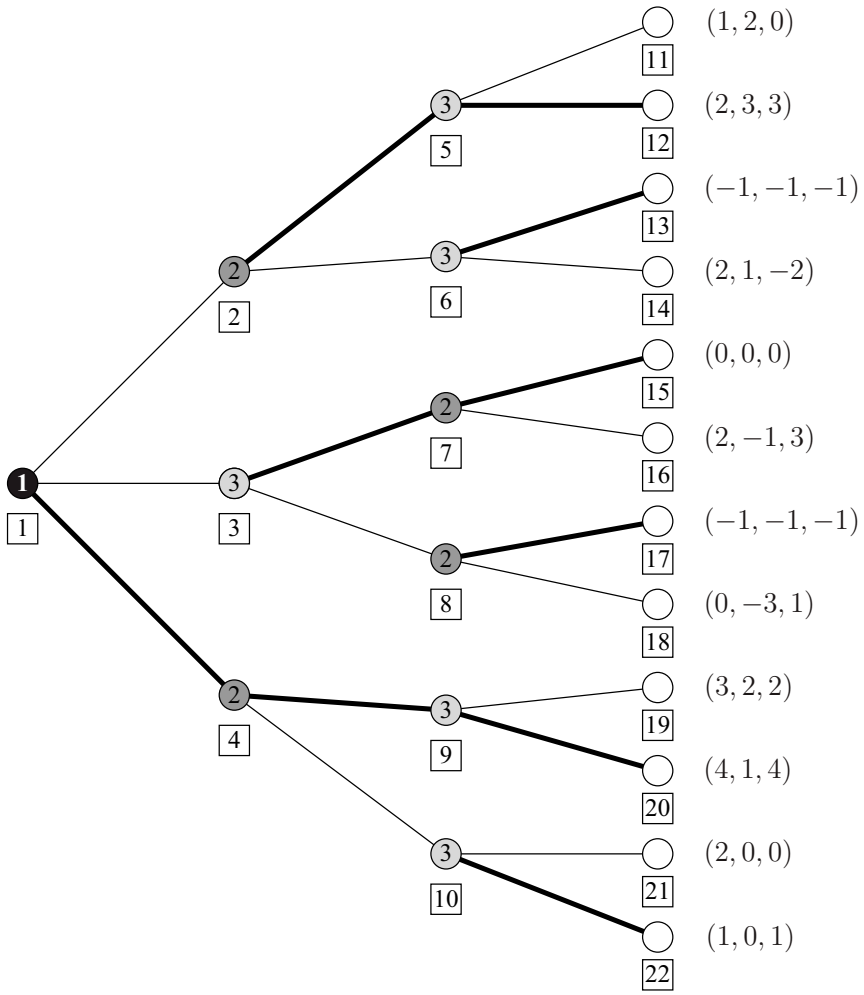
Minden csúcsnál az összehasonlítások száma a csúcsból kiinduló élek száma mínusz 1. Tehát az algoritmusban az összehasonlítások száma az élek száma mínusz n.

8.3. példa. *Véges fa.* A 8.3. ábrán egy véges fa látható. Minden belső csúcsnál egy kis kör látható, mely tartalmazza annak a játékosnak a jelét, aki az adott csúcshoz tartozik. A leveleknél láthatók a kifizetővektorok. Ebben a játékban három játékos van, tehát a kifizetővektoroknak három komponense van. Először megszámozzuk a csúcsokat úgy, hogy minden él kiindulópontjának kisebb legyen a száma, mint a végpontjának. Ezeket a számokat tartalmazzák a csúcsok alatt látható négyzetek. Minden i csúcs esetén igaz, hogy ha $i \geq 11$, akkor i levél, tehát a visszafelé indukciót a 10-es számú csúcsnál kezdjük. Mivel a 10-es csúcshoz $\mathcal{P}_3$ tartozik, így a $(2,0,0)$ és az $(1,0,1)$ kifizetővektorok harmadik komponenseit kell összehasonlítanunk, mivel ezen két kifizetővektor tartozik azokhoz a levelekhez, melyekbe megy él a 10-es csúcsból. Mivel $1 > 0$, ezért $\mathcal{P}_3$ legjobb választása a 22-es csúcs. Tehát $j_{10} = 22$, és $\mathbf{p}^{(10)} = \mathbf{p}^{(22)} = (1,0,1)$. Ezután a 9-es csúcsot vizsgáljuk meg. A $\mathbf{p}^{(19)}$ és a $\mathbf{p}^{(20)}$ vektorok harmadik komponensét összehasonlítva világos, hogy $\mathcal{P}_3$ a 20-as csúcsot választja, így $j_9 = 20$, és $\mathbf{p}^{(9)} = \mathbf{p}^{(20)} = (4,1,4)$. Az ábrán a játékosok választásait a vastagított élek jelzik. Az eljárást a fenti logika szerint folytatva a $8, 7, \ldots, 1$ csúcsokra, végül megkapjuk az 1-es csúcshoz tartozó $\mathbf{p}^{(1)} = (4,1,4)$ kifizetővektort, és az

$$1 \to 4 \to 9 \to 20$$

egyensúlyi utat.

[2]Nem minden egyensúly kapható meg ezzel a módszerrel, de az ezzel a módszerrel kapott egyensúlyok kifizetővektorai megegyeznek egymással. *A fordító.*



8.3. ábra. Egy játékfa.

Gyakorlatok

8.1-1. Egy vállalkozó (E) belép a piacra, amelyet egy áruházlánc (C) tart ellenőrzése alatt. A két szereplő vetélkedése egy kétszemélyes játék. Az áruházlánc stratégiái a megengedés (S), amikor az áruházlánc megengedi, hogy a vállalkozó működjön a piacon, és az elutasítás (T), amikor igyekszik kiszorítani a vállalkozót a piacról. A vállalkozó stratégiái a maradás (I), amikor a vállalkozó a piacon marad, és a kilépés (L), amikor a vállalkozó elhagyja a piacot. A kifizetések a 8.4. ábrán láthatók. Keressük meg az egyensúlyt.

8.1-2. Egy vásárló egy három darabból álló készüléket vásárol a következő feltételekkel: ha minden darab jó, akkor a vevő fizet az eladónak α forintot, egyébként az eladó fizet a vevőnek β forintot. Mielőtt az eladó eladná az árut, ellenőrizheti bármely darabot, de az el-

	I	L
S	2	5
T	0	5

A C játékos kifizetései

	I	L
S	2	1
T	0	1

Az E játékos kifizetései

8.4. ábra. A 8.1-1. gyakorlat adatai.

lenőrzés költsége darabonként γ forint. Tekintsünk egy kétszemélyes játékot, ahol az eladó a $\mathcal{P}_1$ játékos, stratégiái $0,1,2,3$ (az ellenőrzött darabok száma), míg az áru a $\mathcal{P}_2$ játékos, stratégiái $0,1,2,3$ (hány darab hibás). Mutassuk meg, hogy ha feltesszük, hogy minden darab azonos valószínűséggel hibás, akkor a 8.5. ábrán $\mathcal{P}_1$ kifizetőmátrixa látható.

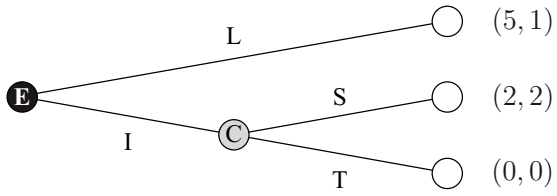
		2-es játékos			
		0	1	2	3
1-es játékos	0	α	$-\beta$	$-\beta$	$-\beta$
	1	$\alpha-\gamma$	$-\frac{2}{3}\beta-\gamma$	$-\frac{1}{3}\beta-\gamma$	$-\gamma$
	2	$\alpha-2\gamma$	$-\frac{1}{3}\beta-\frac{5}{3}\gamma$	$-\frac{4}{3}\gamma$	$-\gamma$
	3	$\alpha-3\gamma$	-2γ	$-\frac{4}{3}\gamma$	$-\gamma$

8.5. ábra. A 8.1-2. gyakorlat adatai.

8.1-3. Tegyük fel, hogy a 8.1-2. gyakorlatban bevezetett játékot úgy módosítjuk, hogy $\mathcal{P}_2$ kifizetései $\mathcal{P}_1$ kifizetéseinek az ellentettjei. Adjuk meg az α,β,γ paraméterek függvényében az egyensúlyok számát. Határozzuk meg az egyensúlyt minden esetre.
8.1-4. Tegyük fel, hogy a 8.1-2. gyakorlatban bevezetett játékban $\mathcal{P}_2$ kifizetőfüggvénye az áru értéke (V, ha minden darab jó, egyébként 0). Van-e egyensúlya ennek a játéknak?
8.1-5. Nézzük a 8.6. ábrán látható fát, mely a 8.1-1. gyakorlatban bevezetett játék fája. Keressük meg a fenti játék egyensúlyát visszafelé indukcióval.
8.1-6. Mutassuk meg, hogy egy játékos esetében a visszafelé indukció a klasszikus dinamikus programozási módszerre egyszerűsödik.
8.1-7. Tegyük fel, hogy egy véges fával ábrázolt játékban néhány csúcs úgynevezett véletlen csúcs, ami azt jelenti, hogy a játék egy ilyen csúcsból egy következő csúcsba valamilyen rögzített valószínűséggel folytatódik. Mutassuk meg, hogy ebben az általánosabb esetben is létezik egyensúly.
8.1-8. Nézzük a 8.3. ábrán adott játékot. Kétszerezzük meg $\mathcal{P}_1$ kifizetéseit, változtassuk ellentettjeire $\mathcal{P}_2$ kifizetéseit, és ne változtassunk $\mathcal{P}_3$ kifizetésein. Keressük meg ennek a módosított játéknak az egyensúlyát.

8.2. Folytonos játékok

Azokat a játékokat, ahol az S_k stratégiahalmazok egy $\mathbb{R}^{n_k}$ euklideszi tér összefüggő részhalmazai, és a kifizetőfüggvények folytonosak, ***folytonos játékoknak*** nevezzük.



8.6. ábra. A 8.1-5. gyakorlat játéka.

8.2.1. A legjobbválaszon alapuló fixpont módszerek

Algoritmikus szemszögből nagyon intuitív és nagyon hasznos a következőkben újrafogalmazni az egyensúly fogalmát. Minden $\mathcal{P}_k$ játékosra és minden $\mathbf{s} = (s_1, s_2, \ldots, s_N) \in S = S_1 \times S_2 \times \cdots \times S_N$ stratégiavektorra definiáljuk a következő leképezést:

$$\begin{aligned} B_k(\mathbf{s}) = \{s_k \in S_k | f_k(s_1, s_2, \ldots, s_{k-1}, s_k, s_{k+1}, \ldots, s_N) \\ = \max_{t_k \in S_k} f_k(s_1, s_2, \ldots, s_{k-1}, t_k, s_{k+1}, \ldots, s_N)\} , \end{aligned} \tag{8.3}$$

amely $\mathcal{P}_k$ legjobb választásainak halmaza, a többi játékos rögzített $(s_1, s_2, \ldots, s_{k-1}, s_{k+1}, \ldots, s_N)$ stratégiái mellett. Vegyük észre, hogy $B_k(\mathbf{s})$ nem függ s_k-tól, $B_k(\mathbf{s})$ csak a többi játékos stratégiáitól, s_l-től $(k \neq l)$ függ. Vegyük észre továbbá, hogy nincs garancia arra, hogy minden $\mathbf{s} \in S_1 \times S_2 \times \cdots \times S_N$ esetén létezik a maximum (8.3)-ban. Legyen $\sum \subseteq S$ olyan részhalmaza S-nek, hogy $B_k(\mathbf{s})$ nemüres halmaz minden k-ra, minden $\mathbf{s} \in \sum$-ra. Az $\mathbf{s}^\star = (s_1^\star, s_2^\star, \ldots, s_N^\star)$ szimultán stratégiavektor pontosan akkor egyensúly, ha $\mathbf{s}^\star \in \sum$, és $s_k^\star \in B_k(\mathbf{s}^\star)$ minden k-ra. Bevezetve a $\mathbf{B}_k(\mathbf{s}) = (B_1(\mathbf{s}), \ldots, B_N(\mathbf{s}))$ ***legjobbválasz-leképezést,*** tovább egyszerűsíthető az egyensúly fogalmának formalizmusa.

8.2. tétel. *Egy* $\mathbf{s}^\star$ *stratégiavektor pontosan akkor egyensúly, ha* $\mathbf{s}^\star \in \sum$ *és* $\mathbf{s}^\star \in \mathbf{B}(\mathbf{s}^\star)$.

Tehát az N személyes játékok egyensúlyi problémája ekvivalens azzal a problémával, hogy megtaláljuk egy halmazértékű leképezés fixpontjait.

A fixpont feladat számítási költsége függ a fixpont feladat típusától, méretétől és a választott számítási módszertől.

Az egyensúlyra vonatkozó – leggyakrabban használt – egzisztencia tételek olyan fixponttételekre támaszkodnak, mint a Brouwer-, a Kakutani-, a Banach-, a Tarski-féle fixponttétel. Bármely fixpontkereső algoritmus sikeresen alkalmazható egyensúlyok meghatározására.

A legnépszerűbb egzisztencia tétel a Kakutani-féle fixponttétel egy nyilvánvaló alkalmazása.

8.3. tétel. *Ha egy N személyes játékra minden k-ra teljesül, hogy*

1. *az* S_k *stratégiahalmazok egy véges dimenziós euklideszi tér nemüres, zárt, korlátos, konvex részhalmazai;*
2. *az* f_k *kifizetőfüggvények folytonosak* S*-en;*
3. *az* f_k *függvény konkáv az* s_k *változó szerint, tehát rögzített* $(s_1, \ldots, s_{k-1}, s_{k+1}, \ldots, s_N)$ *mellett* f_k *konkáv függvény,*

akkor a játéknak van legalább egy egyensúlya.

8.4. példa. *Első kétszemélyes játék.* Tekintsünk egy kétszemélyes játékot ($N = 2$), ahol a stratégiahalmazok $S_1 = S_2 = [0,1]$, a kifizetőfüggvények $f_1(s_1,s_2) = s_1s_2 - 2s_1^2 + 5$, és $f_2(s_1,s_2) = s_1s_2 - 2s_2^2 + s_2 + 3$. Először mindkét játékos legjobbválasz-leképezéseit határozzuk meg. Mindkét kifizetőfüggvény lefelé nyitott parabola, melyek csúcspontjai:

$$s_1 = \frac{s_2}{4} \quad \text{és} \quad s_2 = \frac{s_1+1}{4} .$$

Minden $s_1, s_2 \in [0,1]$ esetén ezek az értékek megvalósítható stratégiák, tehát

$$B_1(\mathbf{s}) = \frac{s_2}{4} \quad \text{és} \quad B_2(\mathbf{s}) = \frac{s_1+1}{4} .$$

Tehát az $(s_1^\star, s_2^\star)$ vektor pontosan akkor egyensúly, ha komponensei kielégítik a következő egyenlőségeket:

$$s_1^\star = \frac{s_2^\star}{4} \quad \text{és} \quad s_2^\star = \frac{s_1^\star+1}{4} .$$

Könnyen látható, hogy az egyenlőségek egyetlen megoldása:

$$s_1^\star = \frac{1}{15} \quad \text{és} \quad s_2^\star = \frac{4}{15} ,$$

tehát $(s_1^\star, s_2^\star)$ a játék egyetlen egyensúlya.

8.5. példa. *Tengeri csatorna.* Tekintsük egy tengeri csatorna egy bizonyos részét a $[0,1]$ intervallumnak. $\mathcal{P}_2$ egy tengeralattjáró, mely az $s_2 \in [0,1]$ helyen rejtőzik. $\mathcal{P}_1$ egy repülőgép, mely bombázhat bármely $s_1 \in [0,1]$ helyet. A bombázó a tengeralattjárónak $\alpha e^{-\beta(s_1-s_2)^2}$ kárt okoz. Így egy speciális kétszemélyes játékot definiáltunk, ahol $S_1 = S_2 = [0,1]$, $f_1(s_1,s_2) = \alpha e^{-\beta(s_1-s_2)^2}$ és $f_2(s_1,s_2) = -f_1(s_1,s_2)$. Ha rögzítjük s_2-t, akkor $f_1(s_1,s_2)$ felveszi maximumát az $s_1 = s_2$ helyen, tehát $\mathcal{P}_1$ legjobbválasz-leképezése: $B_1(\mathbf{s}) = s_2$. $\mathcal{P}_2$ minimalizálni akarja $f_1(s_1,s_2)$-t, mely akkor következik be, ha $|s_1 - s_2|$ a lehető legnagyobb. Ebből következik, hogy

$$B_2(\mathbf{s}) = \begin{cases} 1, & \text{ha } s_1 < 1/2 , \\ 0, & \text{ha } s_1 > 1/2 , \\ \{0,1\}, & \text{ha } s_1 = 1/2 . \end{cases}$$

Világos, hogy nincs olyan $\mathbf{s} = (s_1,s_2) \in [0,1]\times[0,1]$ vektor, hogy $s_1 = B_1(\mathbf{s})$ és $s_2 \in B_2(\mathbf{s})$, tehát nincs egyensúly.

8.2.2. A Fan-egyenlőtlenség alkalmazása

Definiáljuk a $H : S \times S \to \mathbb{R}$ ***összegzőfüggvényt*** a következőképpen:

$$H_{\mathbf{r}}(\mathbf{s},\mathbf{z}) = \sum_{k=1}^{N} r_k f_k(s_1,\dots,s_{k-1},z_k,s_{k+1},\dots,s_N) \tag{8.4}$$

minden $\mathbf{s} = (s_1,\dots,s_N), \mathbf{z}(z_1,\dots,z_N) \in S$-re, ahol $\mathbf{r} = (r_1,r_2,\dots,r_N) > \mathbf{0}$ tetszőleges, rögzített.

8.4. tétel. *Az* $\mathbf{s}^\star \in S$ *vektor pontosan akkor egyensúly, ha*

$$H_{\mathbf{r}}(\mathbf{s}^\star, \mathbf{z}) \leq H_{\mathbf{r}}(\mathbf{s}^\star, \mathbf{s}^\star) \tag{8.5}$$

minden $\mathbf{z} \in S$*-re.*

Bizonyítás. Először tegyük fel, hogy $\mathbf{s}^\star$ egyensúly. Ekkor a (8.1) egyenlőtlenség teljesül minden k-ra és minden $s_k \in S_k$ stratégiára. A (8.1) egyenlőtlenségeinek mindkét oldalát megszorozva az r_k együtthatókkal és összeadva őket a $k = 1, 2, \ldots, N$ értékekre, megkapjuk (8.5)-öt.

Most tegyük fel, hogy a (8.5) egyenlőtlenség teljesül minden $\mathbf{z} \in S$-re. Tetszőleges k-ra és tetszőleges $s_k \in S_k$-ra legyen $\mathbf{z} = (s_1^\star, \ldots, s_{k-1}^\star, s_k, s_{k+1}^\star, \ldots, s_N^\star)$, és alkalmazzuk (8.5)-öt. A k-adik tag kivételével minden tag egyenlő a két oldalon, így törölhetők, míg a megmaradó k-adik tag azt mutatja, hogy a (8.1) egyenlőtlenség teljesül. Tehát $\mathbf{s}^\star$ egyensúly. ■

Vezessük be a következő függvényt: $\phi(\mathbf{s}, \mathbf{z}) = H_{\mathbf{r}}(\mathbf{s}, \mathbf{z}) - H_{\mathbf{r}}(\mathbf{s}, \mathbf{s})$. Világos, hogy $\mathbf{s}^\star$ pontosan akkor egyensúly, ha

$$\phi(\mathbf{s}^\star, \mathbf{z}) \leq 0 \tag{8.6}$$

minden $\mathbf{z} \in S$-re. A (8.6) egyenlőtlenséget ***Fan-egyenlőtlenségnek*** nevezzük. A (8.6) egyenlőtlenség átírható variációs egyenlőtlenséggé (lásd később a 8.2.9. pontban) vagy fixpont feladattá. A második átírási lehetőséget mutatjuk be itt. Minden $\mathbf{s} \in S$-re legyen

$$\Phi(\mathbf{s}) = \{\mathbf{z} | \mathbf{z} \in S,\ \phi(\mathbf{s}, \mathbf{z}) = \max_{\mathbf{t} \in S} \phi(\mathbf{s}, \mathbf{t})\}\ . \tag{8.7}$$

Mivel $\phi(\mathbf{s}, \mathbf{s}) = 0$ minden $\mathbf{s} \in S$-re, így (8.6) egyenlőtlenség pontosan akkor teljesül, ha $\mathbf{s}^\star \in \Phi(\mathbf{s}^\star)$, így $\mathbf{s}^\star$ fixpontja a $\Phi : S \to 2^S$ halmazértékű leképezésnek. Tehát minden fixpontkereső módszer alkalmazható egyensúly számításra.

A fixpont probléma számítási költsége függ a fixpont probléma típusától, méretétől és a választott számítási módszertől.

8.6. példa. *Második kétszemélyes játék.* Tekintsük a 8.4. példát. A mostani esetben:

$$f_1(z_1, s_2) = z_1 s_2 - 2z_1^2 + 5\ ,$$

$$f_2(s_1, z_2) = s_1 z_2 - 2z_2^2 + z_2 + 3\ ,$$

így az összegzőfüggvény formája $r_1 = r_2 = 1$ esetén:

$$H_{\mathbf{r}}(\mathbf{s}, \mathbf{z}) = z_1 s_2 - 2z_1^2 + s_1 z_2 - 2z_2^2 + z_2 + 8\ .$$

Tehát

$$H_{\mathbf{r}}(\mathbf{s}, \mathbf{s}) = 2s_1 s_2 - 2s_1^2 - 2s_2^2 + s_2 + 8\ ,$$

és

$$\phi(\mathbf{s}, \mathbf{z}) = z_1 s_2 - 2z_1^2 + s_1 z_2 - 2z_2^2 + z_2 - 2s_1 s_2 + 2s_1^2 + 2s_2^2 - s_2\ .$$

Vegyük észre, hogy a ϕ függvény szigorúan konkáv mind z_1 szerint, mind z_2 szerint, és ϕ szétválasztható változójú függvény. ϕ stacionárius pontja:

$$\frac{\partial \phi}{\partial z_1} = s_2 - 4z_1 = 0$$

$$\frac{\partial \phi}{\partial z_2} = s_1 - 4z_2 + 1 = 0 \,.$$

Mivel mindkét jobb oldal megvalósítható, így az optimum

$$z_1 = \frac{s_2}{4} \quad \text{és} \quad z_2 = \frac{s_1 + 1}{4} \,.$$

A fixpontban:

$$s_1 = \frac{s_2}{4} \quad \text{és} \quad s_2 = \frac{s_1 + 1}{4} \,,$$

melyből az egyetlen megoldás:

$$s_1 = \frac{1}{15} \quad \text{és} \quad s_2 = \frac{4}{15} \,.$$

8.2.3. A Kuhn–Tucker-feltételek megoldása

Tegyük fel, hogy minden k-ra

$$S_k = \{s_k | \mathbf{g}_k(s_k) \geq \mathbf{0}\} \,,$$

ahol $\mathbf{g}_k : \mathbb{R}^{n_k} \to \mathbb{R}^{m_k}$ az $O_k \supseteq S_k$ nyílt halmazon folytonosan differenciálható, vektor változójú és vektor értékű függvény. Tegyük fel továbbá, hogy az f_k függvény s_k szerint folytonosan parciálisan deriválható O_k-n minden k-ra, tetszőleges rögzített $s_1, \ldots, s_{k-1}, s_{k+1}, \ldots, s_N$ esetén.

Ha $\mathbf{s}^\star = (s_1^\star, \ldots, s_N^\star)$ egyensúly, akkor minden k-ra $s_k^\star$ optimális megoldása a következő feladatnak:

$$\begin{aligned} f_k(s_1^\star, \ldots, s_{k-1}^\star, s_k, s_{k+1}^\star, \ldots, s_N^\star) &\to \max \\ \mathbf{g}_k(s_k) &\geq \mathbf{0} \,. \end{aligned} \tag{8.8}$$

Feltéve, hogy a Kuhn–Tucker regularitási feltételek s_k esetén teljesülnek, a megoldásnak teljesítenie kell a Kuhn–Tucker-féle szükséges feltételeket ($k = 1, 2, \ldots, N$):

$$\begin{aligned} \mathbf{u}_k &\geq \mathbf{0} \\ \mathbf{g}_k(s_k) &\geq \mathbf{0} \\ \nabla_k f_k(\mathbf{s}) + \mathbf{u}_k^T \nabla_k \mathbf{g}_k(s_k) &= \mathbf{0}^T \\ \mathbf{u}_k^T \mathbf{g}_k(s_k) &= 0 \,, \end{aligned} \tag{8.9}$$

ahol $\mathbf{u}_k$ egy m_k komponensű oszlopvektor, $\mathbf{u}_k^T$ jelöli $\mathbf{u}_k$ transzponáltját, $\nabla_k f_k$ az f_k s_k szerinti gradiens függvénye (mint sorvektor), és $\nabla_k \mathbf{g}_k$ a $\mathbf{g}_k$ függvény Jacobi-függvénye.

8.5. tétel. *Ha $\mathbf{s}^\star$ egyensúly, akkor léteznek olyan $\mathbf{u}_k^\star$ vektorok, hogy (8.9) teljesül.*

A (8.9) relációi minden $k = 1, 2, \ldots, N$-re feltételek (általában nagy) rendszerét adja az ismeretlen s_k-ra és $\mathbf{u}_k$-ra. Ha létezik egyensúly, akkor az egyensúlynak teljesítenie kell (8.9)-et. Ha ráadásul minden k-ra $\mathbf{g}_k$ minden komponense szerint konkáv és f_k konkáv s_k szerint, akkor a Kuhn–Tucker-feltételek elégségesek is, tehát (8.9) minden megoldása egyensúly.

A (8.9) megoldásának számítási költsége (8.9) típusától, és a választott módszertől függ. Ha például (8.9) lineáris programozási feladat, melyet szimplex módszerrel oldunk meg, akkor a műveletek száma legrosszabb esetben exponenciális. Egyedi esetekben azonban a megoldás sokkal kevesebb művelettel is meghatározható.

8.7. példa. *Harmadik kétszemélyes játék.* Tekintsük ismét a 8.4. példa kétszemélyes játékát. Világos, hogy

$$S_1 = \{s_1 | s_1 \geq 0,\ 1 - s_1 \geq 0\}\,,$$

$$S_2 = \{s_2 | s_2 \geq 0,\ 1 - s_2 \geq 0\}\,,$$

amiből kapjuk, hogy

$$\mathbf{g}_1(s_1) = \begin{pmatrix} s_1 \\ 1 - s_1 \end{pmatrix} \text{ és } \mathbf{g}_2(s_2) = \begin{pmatrix} s_2 \\ 1 - s_2 \end{pmatrix}.$$

Deriválás után

$$\nabla_1 \mathbf{g}_1(s_1) = \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \quad \nabla_2 \mathbf{g}_2(s_2) = \begin{pmatrix} 1 \\ -1 \end{pmatrix},$$

$$\nabla_1 f_1(s_1, s_2) = s_2 - 4s_1, \quad \nabla_2 f_2(s_1, s_2) = s_1 - 4s_2 + 1\,,$$

tehát a Kuhn–Tucker-feltételek a következő formában írhatók fel:

$$\begin{aligned}
u_1^{(1)}, u_2^{(1)} &\geq 0 \\
s_1 &\geq 0 \\
s_1 &\leq 1 \\
s_2 - 4s_1 + u_1^{(1)} - u_2^{(1)} &= 0 \\
u_1^{(1)} s_1 + u_2^{(1)}(1 - s_1) &= 0 \\
u_1^{(2)}, u_2^{(2)} &\geq 0 \\
s_2 &\geq 0 \\
s_2 &\leq 1 \\
s_1 - 4s_2 + 1 + u_1^{(2)} - u_2^{(2)} &= 0 \\
u_1^{(2)} s_2 + u_2^{(2)}(1 - s_2) &= 0\,.
\end{aligned}$$

Vegyük észre, hogy f_1 konkáv s_1 szerint, f_2 konkáv s_2 szerint, és minden feltétel lineáris, tehát ennek a feltételrendszernek minden megoldása egyensúly. Módszeresen vizsgálva az egyes lehetőségek kombinációit, azt kapjuk, hogy

$$s_1 = 0, \quad 0 < s_1 < 1, \quad s_1 = 1\,,$$

és

$$s_2 = 0, \quad 0 < s_2 < 1, \quad s_2 = 1\,.$$

Könnyen látható, hogy egyetlen megoldás van:

$$u_1^{(1)} = u_1^{(2)} = u_2^{(1)} = u_2^{(2)} = 0\,, \quad s_1 = \frac{1}{15}, \quad s_2 = \frac{4}{15}\,.$$

Túlcsordulás és többlet változókat bevezetve a Kuhn–Tucker-feltételek átírhatók, mint egy nemnegatív rendszer. A nemnegativitási feltételek elhagyhatók, ha a változókat úgy tekintjük, mint valamely új változók négyzeteit, így a végeredmény egy plusz feltételek nélküli, (általában) nemlineáris egyenletrendszer. Számos numerikus módszer áll rendelkezésre az ilyen egyenletrendszerek megoldására.

8.2.4. Visszavezetés optimumszámítási feladatra

Tegyük fel, hogy az előző alfejezet (8.9) feltételei teljesülnek. Tekintsük a következő optimumszámítási feladatot, ahol $k = 1, 2, \ldots, N$:

$$\begin{array}{rcl} \sum_{k=1}^{N} \mathbf{u}_k^T \mathbf{g}_k(s_k) & \rightarrow & \min \\ \mathbf{u}_k & \geq & \mathbf{0} \\ \mathbf{g}_k(s_k) & \geq & \mathbf{0} \\ \nabla_k f_k(\mathbf{s}) + \mathbf{u}_k^T \nabla_k \mathbf{g}_k(s_k) & = & 0\,. \end{array} \tag{8.10}$$

A két első feltétel miatt a célfüggvény nem negatív, így az optimális érték sem negatív. Ebből következik, hogy (8.9)-nek pontosan akkor van megengedett megoldása, ha (8.10)-ben a célfüggvény zéró. Ebben az esetben bármely optimális megoldás teljesíti (8.9)-t.

8.6. tétel. *Egy N személyes játéknak csak akkor van egyensúlya, ha (8.10)-ben a célfüggvény optimális értéke nulla. Ha ráadásul $\mathbf{g}_k$ minden komponense szerint konkáv, és f_k s_k szerint konkáv minden k-ra, akkor (8.10) minden optimális megoldása egyensúly.*

Tehát egy N személyes játék egyensúlyának meghatározása visszavezethető a (8.10) (általában) nemlineáris optimumszámítási feladat megoldására. Bármely nemlineáris programozási módszer használható ennek a problémának a megoldására.

(8.10) megoldásának számítási költsége (8.10) típusától, és a választott módszertől függ. Például, ha (8.10) egy lineáris programozási feladat, melyet a szimplex módszerrel oldunk meg, akkor a műveletek maximális száma exponenciális. Egyedi esetekben azonban a megoldás sokkal kevesebb művelettel is meghatározható.

8.8. példa. *Negyedik kétszemélyes játék.* A 8.7. példa esetén az optimumszámítási feladat a következő formában írható fel:

$$\begin{array}{rcl} u_1^{(1)} s_1 + u_2^{(1)}(1 - s_1) + u_1^{(2)} s_2 + u_2^{(2)}(1 - s_2) & \rightarrow & \min \\ u_1^{(1)}, u_1^{(2)}, u_2^{(1)}, u_2^{(2)} & \geq & 0 \\ s_1 & \geq & 0 \\ s_1 & \leq & 1 \\ s_2 & \geq & 0 \\ s_2 & \leq & 1 \\ s_2 - 4s_1 + u_1^{(1)} - u_2^{(1)} & = & 0 \\ s_1 - 4s_2 + 1 + u_1^{(2)} - u_2^{(2)} & = & 0. \end{array}$$

Vegyük észre, hogy az $u_1^{(1)} = u_1^{(2)} = u_2^{(1)} = u_2^{(2)} = 0$, $s_1 = 1/15$ és $s_2 = 4/15$ megoldás megengedett, a célfüggvény értéke zéró, így egyben optimális megoldás is. Ebből következik, hogy megoldása (8.9)-nek, így a 8.6. tétel miatt egyensúly.

Véges játékok kevert bővítése

Korábban láttuk, hogy egy véges játéknak nem feltétlenül van egyensúlya. Még ha egy véges játéknak van is egyensúlya, és sokszor játsszuk le az adott játékot, akkor is a játékosok szeretnek bevezetni némi véletlenszerűséget az akcióikba, abból a célból, hogy a többi játékost összezavarják, illetve azért, hogy keressenek egy sztochasztikus értelemben vett egyensúlyt.

Ez a gondolat úgy modellezhető, hogy a játékosok stratégiáit valószínűség eloszlásokként vezetjük be, és a várható kifizetések lesznek a kifizetőfüggvények.

A 8.1. alfejezet jelöléseit megtartjuk: N játékosunk van, az $S_k = \{s_k^{(1)}, \ldots, s_k^{(n_k)}\}$ halmaz a $\mathcal{P}_k$ játékos véges stratégiahalmaza. Ennek a véges játéknak a ***kevert bővítésében*** minden játékos egy – a saját stratégiahalmazán értelmezett – diszkrét valószínűségeloszlást vesz, továbbá S_k elemeit a játék minden lejátszásában az adott diszkrét eloszlás szerint választja. Tehát $\mathcal{P}_k$ új stratégiahalmaza:

$$\overline{S}_k = \{\mathbf{x}_k | \mathbf{x}_k = (x_k^{(1)}, \ldots, x_k^{(n_k)}),\ \sum_{i=1}^{n_k} x_k^{(i)} = 1,\ x_k^{(i)} \geq 0 \text{ minden } i\text{-re}\}\ , \tag{8.11}$$

mely halmaz elemei n_k komponensű valószínűségi vektorok. $\mathcal{P}_k$ új kifizető függvénye várható érték függvény:

$$\overline{f}_k(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \ldots \sum_{i_N=1}^{n_N} f_k(s_1^{(i_1)}, s_2^{(i_2)}, \ldots, s_N^{(i_N)}) x_1^{(i_1)} x_2^{(i_2)} \ldots x_N^{(i_N)}\ . \tag{8.12}$$

Vegyük észre, hogy az $\mathbf{x}_k = \mathbf{e}_k$ természetes bázisvektor választással az eredeti „tiszta” stratégiához $(s_k^{(i)})$ tartozó kifizetés kapható meg. A kevert bővítéssel kapott játék folytonos játék, és a 8.2. tétel szerint van legalább egy egyensúlya. Tehát ha adott egy véges játék, melynek nincs egyensúlya, akkor a kevert bővítésének mindig van legalább egy egyensúlya, mely egyensúly az előző alfejezetben ismertetett módszerekkel megkapható.

8.9. példa. *Ötödik kétszemélyes játék.* Tekintsünk egy kétszemélyes játékot ($N = 2$), ahol a 8.1. alfejezetben bevezetett $\mathbf{A}^{(1)}$ és $\mathbf{A}^{(2)}$ mátrixok (i, j) elemei $f_1(s_1^{(i)}, s_2^{(j)})$ és $f_2(s_1^{(i)}, s_2^{(j)})$. Ebben a speciális esetben

$$\overline{f}_k(\mathbf{x}_1, \mathbf{x}_2) = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} a_{ij}^{(k)} x_i^{(1)} x_j^{(2)} = \mathbf{x}_1^T \mathbf{A}^{(k)} \mathbf{x}_2 \ \ (k = 1, 2)\ . \tag{8.13}$$

Az $\overline{S}_k$ feltételei a következő formába írhatók át:

$$\begin{aligned} x_k^{(i)} &\geq 0 \qquad (i = 1, 2, \ldots, n_k)\ , \\ -1 + \sum_{i=1}^{n_k} x_k^{(i)} &\geq 0\ , \\ 1 - \sum_{i=1}^{n_k} x_k^{(i)} &\geq 0\ . \end{aligned}$$

Tehát választhatjuk $\mathbf{g}_k$-t a következőképpen:

$$\mathbf{g}_k(\mathbf{x}_k) = \begin{pmatrix} x_k^{(1)} \\ \vdots \\ x_k^{(n_k)} \\ \sum_{i=1}^{n_k} x_k^{(i)} - 1 \\ -\sum_{i=1}^{n_k} x_k^{(i)} + 1 \end{pmatrix}\ . \tag{8.14}$$

A (8.10)-ben adott optimumszámítási feladat a következő feladatra egyszerűsödik:

$$\begin{aligned}
\textstyle\sum_{k=1}^{2}[\sum_{i=1}^{n_k} u_k^{(i)} x_k^{(i)} + u_k^{(n_k+1)}(\sum_{j=1}^{n_k} x_k^{(j)} - 1) + u_k^{(n_k+2)}(-\sum_{j=1}^{n_k} x_k^{(i)} + 1)] \quad &\rightarrow \quad \min \\
u_k^{(i)} \quad &\geq \quad 0 \quad (1 \leq i \leq n_k + 2) \\
x_k^{(i)} \quad &\geq \quad 0 \quad (1 \leq i \leq n_k) \\
\mathbf{1}^T \mathbf{x}_k \quad &= \quad 1 \\
\mathbf{x}_2^T (\mathbf{A}^{(1)})^T + \mathbf{v}_1^T + (u_1^{(n_1+1)} - u_1^{(n_1+2)})\mathbf{1}_1^T \quad &= \quad \mathbf{0}_1^T \\
\mathbf{x}_1^T (\mathbf{A}^{(2)}) + \mathbf{v}_2^T + (u_2^{(n_2+1)} - u_2^{(n_2+2)})\mathbf{1}_2^T \quad &= \quad \mathbf{0}_2^T \ ,
\end{aligned} \tag{8.15}$$

ahol $\mathbf{v}_k^T = (u_k^{(1)}, \ldots . u_k^{(n_k)})$, $\mathbf{1}_k^T = (1^{(1)}, \ldots, 1^{(n_k)})$ és $\mathbf{0}_k^T = (0^{(1)}, \ldots, 0^{(n_k)})$, $k = 1, 2$.

Vegyük észre, hogy a fenti feladat egy kvadratikus programozási feladat. A számítási költség a választott módszertől függ. Azt is vegyük észre, hogy a fenti probléma általában nem konvex, így lehetséges, hogy az optimum keresése során beragadunk egy lokális optimumba.

Bimátrix-játékok

A kétszemélyes véges játékok kevert kiterjesztéseit ***bimátrix-játékoknak*** nevezzük. A 8.9. példában már vizsgáltunk ilyen játékot. A jelölés egységesítése érdekében a következő egyszerűsítő jelöléseket vezetjük be:

$$\mathbf{A} = \mathbf{A}^{(1)}, \mathbf{B} = \mathbf{A}^{(2)}, \mathbf{x} = \mathbf{x}_1, \mathbf{y} = \mathbf{x}_2, m = n_1 \ \text{ és } \ n = n_2 \ .$$

A következőkben azt mutatjuk meg, hogy a (8.15) feladat átírható olyan kvadratikus programozási feladattá, melyben csak lineáris feltételek vannak.

Tekintsük először a célfüggvényt. Legyenek

$$\alpha = u_1^{(m+2)} - u_1^{(m+1)}, \ \text{ és } \ \beta = u_2^{(n+2)} - u_2^{(n+1)} \ ,$$

ekkor a célfüggvény a következő formába írható át:

$$\mathbf{v}_1^T \mathbf{x} + \mathbf{v}_2^T \mathbf{y} - \alpha(\mathbf{1}_m^T \mathbf{x} - 1) - \beta(\mathbf{1}_n^T \mathbf{y} - 1) \ . \tag{8.16}$$

(8.15)-ben az utolsó két feltétel szintén egyszerűsödik:

$$\begin{aligned}
\mathbf{y}^T \mathbf{A}^T + \mathbf{v}_1^T - \alpha \mathbf{1}_m^T \quad &= \quad \mathbf{0}_m^T \ , \\
\mathbf{x}^T \mathbf{B} + \mathbf{v}_2^T - \beta \mathbf{1}_n^T \quad &= \quad \mathbf{0}_n^T \ ,
\end{aligned}$$

amiből következik:

$$\mathbf{v}_1^T = \alpha \mathbf{1}_m^T - \mathbf{y}^T \mathbf{A}^T \ \text{ és } \ \mathbf{v}_2^T = \beta \mathbf{1}_n^T - \mathbf{x}^T \mathbf{B} \ . \tag{8.17}$$

Mivel

$$\mathbf{1}_m^T \mathbf{x} = \mathbf{1}_n^T \mathbf{y} = 1 \ ,$$

felírhatjuk a célfüggvényt egy újabb formában:

$$(\alpha \mathbf{1}_m^T - \mathbf{y}^T \mathbf{A}^T)\mathbf{x} + (\beta \mathbf{1}_n^T - \mathbf{x}^T \mathbf{B})\mathbf{y} - \alpha(\mathbf{1}_m^T \mathbf{x} - 1) - \beta(\mathbf{1}_n^T \mathbf{y} - 1) = \alpha + \beta - \mathbf{x}^T (\mathbf{A} + \mathbf{B})\mathbf{y} \ .$$

Tehát a következő kvadratikus programozási feladatot kapjuk:

$$\begin{array}{rcl} \mathbf{x}^T(\mathbf{A}+\mathbf{B})\mathbf{y}-\alpha-\beta & \rightarrow & \max \\ \mathbf{x} & \geq & \mathbf{0} \\ \mathbf{y} & \geq & \mathbf{0} \\ \mathbf{1}_m^T\mathbf{x} & = & 1 \\ \mathbf{1}_n^T\mathbf{y} & = & 1 \\ \mathbf{A}\mathbf{y} & \leq & \alpha\mathbf{1}_m \\ \mathbf{B}^T\mathbf{x} & \leq & \beta\mathbf{1}_n\,, \end{array} \tag{8.18}$$

ahol a két utolsó feltétel a $\mathbf{v}_1$, $\mathbf{v}_2$ vektorok nemnegativitásából és (8.17)-ből következik.

8.7. tétel. *Az $\mathbf{x}^\star$, $\mathbf{y}^\star$ vektorpár pontosan akkor egyensúlya az $(\mathbf{A},\mathbf{B})$ bimátrix-játéknak, ha valamilyen $\alpha^\star$-ra és $\beta^\star$-ra $(\mathbf{x}^\star,\mathbf{y}^\star,\alpha^\star,\beta^\star)$ optimális megoldása a (8.18) feladatnak. Ekkor az optimumban a célfüggvény értéke nulla.*

Ez kvadratikus programozási feladat, ahol a számítási költség a választott módszertől függ. Általában nem konvex a feladat, így benneragadhatunk egy lokális optimumban. Mivel tudjuk, hogy a globális optimumban a célfüggvény értéke nulla, így ellenőrizni tudjuk az optimalitást. Ha $\mathbf{A}+\mathbf{B}$ negatív szemidefinit, akkor a feladat konvex, tehát minden lokális optimum globális is.

8.10. példa. *Első bimátrix-játék.* Válasszuk **A**-t és **B**-t a következőképpen:

$$\mathbf{A} = \begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix}$$

és

$$\mathbf{B} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}.$$

Ekkor

$$\mathbf{A}+\mathbf{B} = \begin{pmatrix} 3 & -2 \\ -2 & 3 \end{pmatrix},$$

tehát (8.18) a következő formát ölti:

$$\begin{array}{rcl} 3x_1y_1 - 2x_1y_2 - 2x_2y_1 + 3x_2y_2 - \alpha - \beta & \rightarrow & \max \\ x_1, x_2, y_1, y_2 & \geq & 0 \\ x_1 + x_2 & = & 1 \\ y_1 + y_2 & = & 1 \\ 2y_1 - y_2 & \leq & \alpha \\ -y_1 + y_2 & \leq & \alpha \\ x_1 - x_2 & \leq & \beta \\ -x_1 + 2x_2 & \leq & \beta\,, \end{array}$$

ahol $\mathbf{x} = (x_1, x_2)^T$ és $\mathbf{y} = (y_1, y_2)^T$. A 8.7. tételből tudjuk, hogy az optimális célfüggvényérték nulla, így minden megengedett megoldás, melyre a célfüggvény értéke nulla, szükségszerűen optimális.

Könnyen látható, hogy

$$\mathbf{x} = \begin{pmatrix}1\\0\end{pmatrix},\ \mathbf{y} = \begin{pmatrix}0\\1\end{pmatrix},\ \alpha = 2, \beta = 1\ ,$$
$$\mathbf{x} = \begin{pmatrix}0\\1\end{pmatrix},\ \mathbf{y} = \begin{pmatrix}1\\0\end{pmatrix},\ \alpha = 1,\ \beta = 2\ ,$$
$$\mathbf{x} = \begin{pmatrix}0.6\\0.4\end{pmatrix},\ \mathbf{y} = \begin{pmatrix}0.4\\0.6\end{pmatrix},\ \alpha = 0.2,\ \beta = 0.2$$

mind optimumok, tehát mindegyik meghatároz egy egyensúlyt.

Alkalmazhatjuk (8.9)-et egyensúly keresésre. Ahelyett, hogy megoldjuk a (8.18) optimumszámítási feladatot, megoldjuk az (8.9) feltételrendszert. A bimátrix-játékok esetén a (8.9) feladat a következő formára egyszerűsödik:

$$\begin{aligned}
\mathbf{x}^T\mathbf{A}\mathbf{y} &= \alpha \\
\mathbf{x}^T\mathbf{B}\mathbf{y} &= \beta \\
\mathbf{A}\mathbf{y} &\le \alpha\mathbf{1}_m \\
\mathbf{B}^T\mathbf{x} &\le \beta\mathbf{1}_n \\
\mathbf{x} &\ge \mathbf{0}_m \\
\mathbf{y} &\ge \mathbf{0}_n \\
\mathbf{1}_m^T\mathbf{x} = \mathbf{1}_n^T\mathbf{y} &= 1\ ,
\end{aligned} \tag{8.19}$$

mely feltételrendszer a kvadratikus programozási feladatnál látottakkal analóg módon vezethető le.

A (8.19) feladat számítási költsége a választott módszertől függ.

8.11. példa. *Második bimátrix-játék.* Tekintsük ismét a 8.10. példát. Helyettesítsük a (8.19) feltételrendszer első és a második feltétele alapján α-t és β-t a harmadik és a negyedik feltételbe, ekkor

$$\begin{aligned}
2y_1 - y_2 &\le 2x_1y_1 - x_1y_2 - x_2y_1 + x_2y_2 \\
-y_1 + y_2 &\le 2x_1y_1 - x_1y_2 - x_2y_1 + x_2y_2 \\
x_1 - x_2 &\le x_1y_1 - x_1y_2 - x_2y_1 + 2x_2y_2 \\
-x_1 + 2x_2 &\le x_1y_1 - x_1y_2 - x_2y_1 + 2x_2y_2 \\
x_1, x_2, y_1, y_2 &\ge 0 \\
x_1 + x_2 = y_1 + y_2 &= 1\ .
\end{aligned}$$

Könnyen látható, hogy a 8.10. példában kapott megoldások kielégítik a fenti feltételrendszert, tehát egyensúlyok.

A bimátrix-játék egyensúlyi feladatát átírhatjuk kevert változós feltételrendszerbe is. Tegyük fel, hogy $\mathbf{A}$ és $\mathbf{B}$ minden eleme 0 és 1 között van. Ez a feltevés nem túl erős, hiszen lineáris transzformációk használatával

$$\overline{\mathbf{A}} = a_1\mathbf{A} + b_1\mathbf{1} \quad \text{és} \quad \overline{\mathbf{B}} = a_2\mathbf{B} + b_2\mathbf{1}\ ,$$

ahol $a_1, a_2 > 0$, $\mathbf{1}$ a csupa egyesekből álló $m \times n$-es mátrix. Ekkor az egyensúly nem változik, és – megfelelő a_1, b_1, a_2 és b_2 értékek választásával – az $\overline{\mathbf{A}}$ és $\overline{\mathbf{B}}$ mátrixok minden eleme a $[0, 1]$ intervallumba esik.

8.8. tétel. *Az* $\mathbf{x}$, $\mathbf{y}$ *vektorpár pontosan akkor egyensúly, ha valamilyen* α,β *számokra és* $\mathbf{u}$, $\mathbf{v}$ *nulla-egy vektorokra teljesül, hogy*

$$\begin{array}{ccccccc} 0 & \le & \alpha\mathbf{1}_m - \mathbf{A}\mathbf{y} & \le & \mathbf{1}_m - \mathbf{u} & \le & \mathbf{1}_m - \mathbf{x} \\ 0 & \le & \beta\mathbf{1}_n - \mathbf{B}^T\mathbf{x} & \le & \mathbf{1}_n - \mathbf{v} & \le & \mathbf{1}_n - \mathbf{y} \\ & & & & \mathbf{x} & \ge & \mathbf{0}_m \\ & & & & \mathbf{y} & \ge & \mathbf{0}_n \\ & & \mathbf{1}_m^T\mathbf{x} & = & \mathbf{1}_n^T\mathbf{y} & = & 1\ . \end{array} \tag{8.20}$$

Bizonyítás. Először tegyük fel, hogy $\mathbf{x}$, $\mathbf{y}$ vektorpár egyensúly, ekkor valamilyen α-ra, β-ra (8.19) teljesül. Legyen

$$u_i = \begin{cases} 1, & \text{ha } x_i > 0\ , \\ 0, & \text{ha } x_i = 0\ , \end{cases} \quad \text{és} \quad v_j = \begin{cases} 1, & \text{ha } y_j > 0\ , \\ 0, & \text{ha } y_j = 0\ . \end{cases}$$

Mivel az $\mathbf{A}$ és $\mathbf{B}$ mátrixok elemei $[0,1]$-ből valók, így $\alpha = \mathbf{x}^T\mathbf{A}\mathbf{y}$ és $\beta = \mathbf{x}^T\mathbf{B}\mathbf{y}$ szintén nulla és egy közöttiek. Vegyük észre, hogy

$$0 = \mathbf{x}^T(\alpha\mathbf{1}_m - \mathbf{A}\mathbf{y}) = \mathbf{y}^T(\beta\mathbf{1}_n - \mathbf{B}^T\mathbf{x})\ ,$$

melyből következik, hogy (8.20) teljesül.

Most tegyük fel, hogy (8.20) teljesül. Ekkor

$$\mathbf{0} \le \mathbf{x} \le \mathbf{u} \le \mathbf{1}_m \quad \text{és} \quad \mathbf{0} \le \mathbf{y} \le \mathbf{v} \le \mathbf{1}_n\ .$$

Ha $u_i = 1$, akkor $\alpha - \mathbf{e}_i^T\mathbf{A}\mathbf{y} = 0$, és ha $u_i = 0$, akkor $x_i = 0$. Tehát

$$\mathbf{x}^T(\alpha\mathbf{1}_m - \mathbf{A}\mathbf{y}) = 0\ ,$$

melyből következik, hogy $\alpha = \mathbf{x}^T\mathbf{A}\mathbf{y}$. A $\beta = \mathbf{x}^T\mathbf{B}\mathbf{y}$ egyenlőség érvényessége hasonlóan mutatható meg, így (8.19) teljesül, tehát az $\mathbf{x},\mathbf{y}$ vektorpár egyensúly. ■

A (8.20) feladat számítási költsége a választott módszertől függ.

8.12. példa. *Harmadik bimátrix-játék.* A 8.10. példában bevezetett bimátrix-játék esetén (8.20) a következő formát ölti:

$$\begin{array}{ccccccc} 0 & \le & \alpha - 2y_1 + y_2 & \le & 1 - u_1 & \le & 1 - x_1 \\ 0 & \le & \alpha + y_1 - y_2 & \le & 1 - u_2 & \le & 1 - x_2 \\ 0 & \le & \beta - x_1 + x_2 & \le & 1 - v_1 & \le & 1 - y_1 \\ 0 & \le & \beta + x_1 - 2x_2 & \le & 1 - v_2 & \le & 1 - y_2 \\ & & x_1 + x_2 & = & y_1 + y_2 & = & 1 \\ & & & & x_1, x_2, y_1, y_2 & \ge & 0 \\ & & & & u_1, u_2, v_1, v_2 & \in & \{0,1\}\ . \end{array}$$

Vegyük észre, hogy a 8.10. példában adott három megoldás teljesíti a fenti feltételrendszert az $\mathbf{u} = (1,0)$, $\mathbf{v} = (0,1)$, $\mathbf{u} = (0,1)$, $\mathbf{v} = (1,0)$, és $\mathbf{u} = (1,1)$, $\mathbf{v} = (1,1)$ vektorpárokkal.

Mátrixjátékok

Azokat a bimátrix-játékokat, ahol $\mathbf{B} = -\mathbf{A}$, ***mátrixjátékoknak*** nevezzük, és egy $\mathbf{A}$ mátrixszal jelöljük. Az ilyen játékokra néha $\mathbf{A}$ *mátrixjátékként* fogunk hivatkozni. Mivel $\mathbf{A}+\mathbf{B} = \mathbf{0}$, a (8.18)-bani kvadratikus programozási feladat lineáris:

$$\begin{array}{rcl} \alpha+\beta & \to & \min \\ \mathbf{x} & \geq & \mathbf{0} \\ \mathbf{y} & \geq & \mathbf{0} \\ \mathbf{1}_m\mathbf{x} & = & 1 \\ \mathbf{1}_n\mathbf{y} & = & 1 \\ \mathbf{A}\mathbf{y} & \leq & \alpha\mathbf{1}_m \\ \mathbf{A}^T\mathbf{x} & \geq & -\beta\mathbf{1}_n\ . \end{array} \tag{8.21}$$

A fenti feladatból látható, hogy az egyensúlyok halmaza konvex poliéder. Vegyük észre, hogy $(\mathbf{x},\beta)$ és $(\mathbf{y},\alpha)$ szétválasztható, amiből a következő eredmény vezethető le.

8.9. tétel. *Az $\mathbf{x}^\star$, $\mathbf{y}^\star$ vektorpár pontosan akkor egyensúlya az $\mathbf{A}$ mátrixjátéknak, ha valamilyen $\alpha^\star$-ra, $\beta^\star$-ra $(\mathbf{x}^\star,\beta^\star)$ és $(\mathbf{y}^\star,\alpha^\star)$ optimális megoldásai a következő lineáris programozási feladatpárnak:*

$$\begin{array}{rcl|rcl} \alpha & \to & \min & \beta & \to & \min \\ \mathbf{y} & \geq & \mathbf{0}_n & \mathbf{x} & \geq & \mathbf{0}_m \\ \mathbf{1}_n^T\mathbf{y} & = & 1 & \mathbf{1}_m^T\mathbf{x} & = & 1 \\ \mathbf{A}\mathbf{y} & \leq & \alpha\mathbf{1}_m & \mathbf{A}^T\mathbf{x} & \geq & -\beta\mathbf{1}_n\ . \end{array} \tag{8.22}$$

Vegyük észre, hogy az optimumban $\alpha+\beta = 0$. Az α optimális értékét a mátrixjáték ***értékének*** nevezzük.

Ha a szimplex módszert alkalmazzuk (8.22) megoldására, akkor a műveletek száma exponenciális. Polinomiális algoritmussal (mint amilyen a belső pont módszer) a műveletek száma csak polinomiális.

8.13. példa. *Első mátrixjáték.* Tekintsük a következő mátrixjátékot:

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 0 \\ 2 & 0 & 3 \\ -1 & 3 & 3 \end{pmatrix}.$$

A (8.22)-t erre a feladatra felírva:

$$\begin{array}{rcl|rcl} \alpha & \to & \min & \beta & \to & \min \\ y_1,y_2,y_3 & \geq & 0 & x_1,x_2,x_3 & \geq & 0 \\ y_1+y_2+y_3 & = & 1 & x_1+x_2+x_3 & = & 1 \\ 2y_1+y_2-\alpha & \leq & 0 & 2x_1+2x_2-x_3+\beta & \geq & 0 \\ 2y_1+3y_3-\alpha & \leq & 0 & x_1+3x_3+\beta & \geq & 0 \\ -y_1+3y_2+3y_3-\alpha & \leq & 0 & 3x_2+3x_3+\beta & \geq & 0\ . \end{array}$$

A szimplex módszerrel megkaphatjuk a fenti feladatpár megoldását: $\alpha = 9/7$, $\mathbf{y} = (3/7, 3/7, 1/7)$, $\beta = -9/7$, és $\mathbf{x} = (4/7, 4/21, 5/21)$.

A (8.21) megoldását úgy is megkaphatjuk, hogy lineáris feltételek bizonyos halmazának megengedett megoldását keressük meg. Mivel (8.21)-ben az optimumban $\alpha+\beta = 0$, $\mathbf{x}$, $\mathbf{y}$

vektorok és α, β skalárok pontosan akkor alkotnak optimális megoldást, ha

$$\begin{array}{rcl} \mathbf{x}, \mathbf{y} & \geq & \mathbf{0} \\ \mathbf{1}_m^T \mathbf{x} & = & 1 \\ \mathbf{1}_n^T \mathbf{y} & = & 1 \\ \mathbf{A}\mathbf{y} & \leq & \alpha \mathbf{1}_m \\ \mathbf{A}^T \mathbf{x} & \geq & \alpha \mathbf{1}_n \ . \end{array} \tag{8.23}$$

A (8.23) megoldásához a szimplex módszer első fázisa szükséges, mely a legkedvezőtlenebb esetben exponenciális számú műveletet igényel. A gyakorlatban általában sokkal kevesebb művelet szükséges.

8.14. példa. *Második mátrixjáték.* Nézzük megint a 8.13. példában bevezetett mátrixjátékot. Ha erre a játékra (8.23)-at felírjuk, akkor a következőt kapjuk:

$$\begin{array}{rcl} x_1, x_2, x_3, y_1, y_2, y_3 & \geq & 0 \\ x_1 + x_2 + x_3 = y_1 + y_2 + y_3 & = & 1 \\ 2y_1 + y_2 & \leq & \alpha \\ 2y_1 + 3y_3 & \leq & \alpha \\ -y_1 + 3y_2 + 3y_3 & \leq & \alpha \\ 2x_1 + 2x_2 - x_3 & \geq & \alpha \\ x_1 + 3x_3 & \geq & \alpha \\ 3x_2 + 3x_3 & \geq & \alpha \ . \end{array}$$

Könnyen látható, hogy $\alpha = 9/7$, $\mathbf{x} = (4/7, 4/21, 5/21)^T$, $\mathbf{y} = (3/7, 3/7, 1/7)^T$ kielégíti a (8.9) feltételrendszert, tehát az $\mathbf{x}$, $\mathbf{y}$ vektorpár egyensúly.

8.2.5. A fiktív lejátszás módszere

Tekintsünk most egy $\mathbf{A}$ mátrixjátékot. Az ezen alfejezetben tárgyalt módszer fő gondolata az, hogy lépésenként minden játékos meghatározza a saját legjobbválasz tiszta stratégiáját a másik játékos – az előzőekben választott – stratégiáinak átlaga mellett. Formálisan a módszer a következőképpen írható fel.

Legyen $\mathbf{x}_1$ a $\mathcal{P}_1$ játékos kezdeti (kevert) stratégiája. Válasszuk úgy $\mathbf{y}_1 = \mathbf{e}_{j_1}$-t, hogy teljesüljön

$$\mathbf{x}_1^T \mathbf{A} \mathbf{e}_{j_1} = \min_j \{\mathbf{x}_1^T \mathbf{A} \mathbf{e}_j\} \ . \tag{8.24}$$

Bármely további $k \geq 2$ lépéskor legyen

$$\overline{\mathbf{y}}_{k-1} = \frac{1}{k-1}((k-2)\overline{\mathbf{y}}_{k-2} + \mathbf{y}_{k-1}) \ , \tag{8.25}$$

és válasszuk $\mathbf{x}_k = \mathbf{e}_{i_k}$-t úgy, hogy teljesüljön

$$\mathbf{e}_{i_k}^T \mathbf{A} \overline{\mathbf{y}}_{k-1} = \max_i \{\mathbf{e}_i^T \mathbf{A} \overline{\mathbf{y}}_{k-1}\} \ . \tag{8.26}$$

Ekkor legyen

$$\overline{\mathbf{x}}_k = \frac{1}{k}((k-1)\overline{\mathbf{x}}_{k-1} + \mathbf{x}_k) \ , \tag{8.27}$$

és válasszuk $\mathbf{y}_k = \mathbf{e}_{j_k}$-t úgy, hogy

$$\overline{\mathbf{x}}_k^T \mathbf{A} \mathbf{e}_{j_k} = \min_j \{\overline{\mathbf{x}}_k^T \mathbf{A} \mathbf{e}_j\} \; . \tag{8.28}$$

A fenti általános lépés megismétlése ($k = 2, 3, \ldots$)-ra két sorozatot eredményez: $\{\overline{\mathbf{x}}_k\}$-t, és $\{\overline{\mathbf{y}}_k\}$-t. Ekkor a következő eredményt kapjuk:

8.10. tétel. *Az* $(\{\overline{\mathbf{x}}_k\}, \{\overline{\mathbf{y}}_k\})$ *sorozatpár tetszőleges torlódási pontja az* $\mathbf{A}$ *mátrixjáték egy egyensúlya.*

Bizonyítás. Mivel $\{\overline{\mathbf{x}}_k\}$ és $\{\overline{\mathbf{y}}_k\}$ valószínűségi vektorok, így korlátos, valós sorozatok, tehát van legalább egy torlódási pontjuk[3]. ■

Tegyük fel, hogy az $\mathbf{A}$ mátrix $m \times n$-es. (8.24)-ben mn szorzásra van szükségünk. (8.25)-ben és (8.27)-ben $m + n$ szorzás és osztás van. (8.26)-ban és (8.28)-ban mn szorzás van. Ha L iterációs lépést teszünk, akkor az osztások és szorzások száma:

$$mn + L[2(m + n) + 2mn] = \Theta(Lmn) \; .$$

8.15. példa. *Harmadik mátrixjáték.* A fent tárgyalt módszert alkalmazzuk a 8.14. példában tárgyalt mátrixjátékra. A módszer kezdő állapota: $\mathbf{x}_1 = (1, 0, 0)^T$. 100 lépés után: $\overline{\mathbf{x}}_{101} = (0.446, 0.287, 0.267)^T$ és $\overline{\mathbf{y}}_{101} = (0.386, 0.436, 0.178)^T$. Ha ezeket az értékeket az egyensúlyhoz hasonlítjuk, akkor azt tapasztaljuk, hogy az eltérés kisebb, mint 0.126, tehát a módszer lassan konvergál.

8.2.6. Szimmetrikus mátrixjátékok

Az $\mathbf{A}$ mátrixjátékot, ahol $\mathbf{A}$ ferdén-szimmetrikus, ***szimmetrikus mátrixjátéknak*** nevezzük. Ebben az esetben $\mathbf{A}^T = -\mathbf{A}$ és a két lineáris programozási feladat (8.22)-ben megegyezik. Ebből következik, hogy $\alpha = \beta = 0$ (a játék értéke 0), és tetszőleges egyensúlyban a két játékos stratégiái megegyeznek. Tehát a következő eredmény adódik.

8.11. tétel. *Az* $\mathbf{x}^\star$ *vektor pontosan akkor egyensúlya az* $\mathbf{A}$ *szimmetrikus mátrixjátéknak, ha*

$$\begin{aligned} \mathbf{x} &\geq \mathbf{0} \\ \mathbf{1}^T\mathbf{x} &= 1 \\ \mathbf{A}\mathbf{x} &\leq \mathbf{0} \; . \end{aligned} \tag{8.29}$$

(8.29) megoldásához a szimplex módszer első fázisa szükséges, ahol a legkedvezőtlenebb esetben a műveletek száma exponenciális. A gyakorlatban azonban általában sokkal kisebb számú műveletre van szükség (8.29) megoldásához.

[3] Nem minden egyensúly kapható meg ezzel a módszerrel, illetve van olyan egyensúlyi pont, amit csak akkor talál meg ez a módszer, ha az egyensúlyból indul. *A fordító.*

8.16. példa. *Szimmetrikus mátrixjáték.* Tekintsük az $\mathbf{A} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ szimmetrikus mátrixjátékot. Ekkor (8.29) a következő formát ölti:

$$\begin{aligned} x_1, x_2 &\geq 0 \\ x_1 + x_2 &= 1 \\ x_2 &\leq 0 \\ -x_1 &\leq 0\,. \end{aligned}$$

Könnyen látható, hogy az egyetlen megoldás: $x_1 = 1$ és $x_2 = 0$, ami az első tiszta stratégia.

A következő fejezetben látni fogjuk, hogy egy lineáris programozási feladat ekvivalens egy szimmetrikus mátrixjáték egyensúlyi problémájával, tehát tetszőleges módszer, amely egy szimmetrikus mátrixjáték egyensúlyi problémájának megoldására alkalmas, alkalmas lineáris programozási feladat megoldására is, ezért az ilyen módszerek a szimplex módszer alternatíváiként szolgálnak. A következőkben azt mutatjuk meg, hogy a szimmetria nem túlságosan erős feltétel, mert tetszőleges mátrixjáték megfeleltethető egy vele ekvivalens szimmetrikus mátrixjátéknak.

Tekintsük az $\mathbf{A}$ mátrixjátékot, és szerkesszük meg a következő ferdén-szimmetrikus $\mathbf{P}$ mátrixot:

$$\mathbf{P} = \begin{pmatrix} \mathbf{0}_{m\times m} & \mathbf{A} & -\mathbf{1}_m \\ -\mathbf{A}^T & \mathbf{0}_{n\times n} & \mathbf{1}_n \\ \mathbf{1}_m^T & -\mathbf{1}_n^T & 0 \end{pmatrix}.$$

Az $\mathbf{A}$ és $\mathbf{P}$ mátrixjátékok a következő értelemben ekvivalensek. Tegyük fel, hogy $\mathbf{A} > \mathbf{0}$, ami nem túl erős feltétel, hiszen $\mathbf{A}$ elemeihez egy megfelelő konstanst hozzáadva $\mathbf{A} > \mathbf{0}$ elérhető, és az egyensúlyok nem változnak.

8.12. tétel. *Legyen $\mathbf{P}$ szimmetrikus mátrixjáték, melyet $\mathbf{A}$ mátrixjátékból kaptunk, ekkor igaz a következő két állítás:*

1. *Ha $\mathbf{z} = \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \\ \lambda \end{pmatrix}$ egyensúlyi stratégiája a $\mathbf{P}$ szimmetrikus mátrixjátéknak, akkor az $\mathbf{x} = (1/a)\mathbf{u}$, $\mathbf{y} = (1/a)\mathbf{v}$ vektorpár egyensúlya az $\mathbf{A}$ mátrixjátéknak, és az $\mathbf{A}$ mátrixjáték értéke: $v = \lambda/a$, ahol $a = (1-\lambda)/2$.*
2. *Ha az $\mathbf{x}, \mathbf{y}$ vektorpár egyensúlya az $\mathbf{A}$ mátrixjátéknak, és v az $\mathbf{A}$ mátrixjáték értéke, akkor a*

$$\mathbf{z} = \frac{1}{2+v}\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \\ v \end{pmatrix}$$

vektor egyensúlya a $\mathbf{P}$ szimmetrikus mátrixjátéknak.

Bizonyítás. Először tegyük fel, hogy $\mathbf{z}$ egyensúly a $\mathbf{P}$ szimmetrikus mátrixjátékban (1. pont). Ekkor $\mathbf{u} \geq \mathbf{0}$, $\mathbf{v} \geq \mathbf{0}$, és $\mathbf{Pz} \leq \mathbf{0}$, tehát

$$\begin{aligned} \mathbf{Av} - \lambda\mathbf{1}_m &\leq \mathbf{0} \\ -\mathbf{A}^T\mathbf{u} + \lambda\mathbf{1}_n &\leq \mathbf{0} \\ \mathbf{1}_m^T\mathbf{u} - \mathbf{1}_n^T\mathbf{v} &\leq 0\,. \end{aligned} \tag{8.30}$$

Először megmutatjuk, hogy $\lambda \in (0,1)$, tehát $a \neq 0$. Tegyük fel, hogy $\lambda = 1$, ekkor (mivel $\mathbf{z}$ valószínűségi vektor) $\mathbf{u} = \mathbf{0}_m$ és $\mathbf{v} = \mathbf{0}_n$, ami ellentmond (8.30) második egyenlőtlenségének. Ha $\lambda = 0$, akkor $\mathbf{1}_m^T\mathbf{u}+\mathbf{1}_n^T\mathbf{v} = 1$, és (8.30) harmadik egyenlőtlensége miatt $\mathbf{v}$-nek van legalább egy pozitív komponense, mely ellentmond (8.30) első egyenlőtlenségének.

Most megmutatjuk, hogy $\mathbf{1}_m^T\mathbf{u} = \mathbf{1}_n^T\mathbf{v}$. (8.30)-ból azt kapjuk, hogy

$$\begin{aligned} \mathbf{u}^T\mathbf{A}\mathbf{v} - \lambda\mathbf{u}^T\mathbf{1}_m &\leq 0\,, \\ -\mathbf{v}^T\mathbf{A}^T\mathbf{u} + \lambda\mathbf{v}^T\mathbf{1}_n &\leq 0\,. \end{aligned}$$

A két egyenlőtlenséget összeadva kapjuk, hogy

$$\mathbf{v}^T\mathbf{1}_n - \mathbf{u}^T\mathbf{1}_m \leq 0\,.$$

Ezt (8.30) harmadik egyenlőtlenségével kombinálva kapjuk, hogy $\mathbf{1}_m^T\mathbf{u} - \mathbf{1}_n^T\mathbf{v} = 0$.

Legyen $a = (1-\lambda)/2 \neq 0$, ekkor $\mathbf{1}_m^T\mathbf{u} = \mathbf{1}_n^T\mathbf{v} = a$, így mind $\mathbf{x} = \mathbf{u}/a$, mind $\mathbf{y} = \mathbf{v}/a$ valószínűségi vektor, és (8.30)-ból következik, hogy:

$$\begin{aligned} \mathbf{A}^T\mathbf{x} &= \tfrac{1}{a}\mathbf{A}^T\mathbf{u} \geq \tfrac{\lambda}{a}\mathbf{1}_n\,, \\ \mathbf{A}\mathbf{y} &= \tfrac{1}{a}\mathbf{A}\mathbf{v} \leq \tfrac{\lambda}{a}\mathbf{1}_m\,. \end{aligned}$$

Legyenek $\alpha = \lambda/a$ és $\beta = -\lambda/a$, ekkor $\mathbf{x},\mathbf{y}$ vektorpár megoldása (8.22)-nek és $\alpha+\beta = 0$, tehát $\mathbf{x},\mathbf{y}$ vektorpár egyensúlya az $\mathbf{A}$ mátrixjátéknak.

A 2. pont hasonlóan látható be, itt nem részletezzük. ■

8.2.7. Lineáris programozás és mátrixjátékok

Ebben az alfejezetben megmutatjuk, hogy egy lineáris programozási feladatot meg lehet oldani úgy, hogy egy szimmetrikus mátrixjáték egyensúlyi stratégiáit keressük meg. Tehát tetszőleges olyan módszer, mely alkalmas egy szimmetrikus mátrixjáték egyensúlyainak meghatározására, alkalmas a szimplex módszer kiváltására.

Tekintsük a következő primál-duál lineáris programozási feladatpárt:

$$\begin{array}{rcl|rcl} \mathbf{c}^T\mathbf{x} & \to & \max & \mathbf{b}^T\mathbf{y} & \to & \min \\ \mathbf{x} & \geq & \mathbf{0} & \mathbf{y} & \geq & \mathbf{0} \\ \mathbf{A}\mathbf{x} & \leq & \mathbf{b} & \mathbf{A}^T\mathbf{y} & \geq & \mathbf{c}\,. \end{array} \tag{8.31}$$

Szerkesszük meg a következő ferdén-szimmetrikus mátrixot:

$$\mathbf{P} = \begin{pmatrix} \mathbf{0} & \mathbf{A} & -\mathbf{b} \\ -\mathbf{A}^T & \mathbf{0} & \mathbf{c} \\ \mathbf{b}^T & -\mathbf{c}^T & 0 \end{pmatrix}.$$

8.13. tétel. *Tegyük fel, hogy* $\mathbf{z} = \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \\ \lambda \end{pmatrix}$ *egyensúlya a* $\mathbf{P}$ *szimmetrikus mátrixjátéknak, és* $\lambda > 0$. *Ekkor*

$$\mathbf{x} = \frac{1}{\lambda}\mathbf{v} \quad \text{és} \quad \mathbf{y} = \frac{1}{\lambda}\mathbf{u}$$

optimális megoldásai a (8.31) *primál-duál feladatpárnak* ($\mathbf{x}$ *a primálnak,* $\mathbf{y}$ *a duálnak).*

Bizonyítás. Ha $\mathbf{z}$ egyensúly, akkor $\mathbf{Pz} \le \mathbf{0}$, azaz

$$\begin{aligned} \mathbf{A}\mathbf{v} - \lambda\mathbf{b} &\le \mathbf{0} \\ -\mathbf{A}^T\mathbf{u} + \lambda\mathbf{c} &\le \mathbf{0} \\ \mathbf{b}^T\mathbf{u} - \mathbf{c}^T\mathbf{v} &\le \mathbf{0}\ . \end{aligned} \tag{8.32}$$

Mivel $\mathbf{z} \ge \mathbf{0}$ és $\lambda > 0$, így mind az $\mathbf{x} = (1/\lambda)\mathbf{v}$, mind az $\mathbf{y} = (1/\lambda)\mathbf{u}$ vektor nemnegatív. Osszuk el (8.32) első két egyenlőtlenségét λ-val, ekkor

$$\mathbf{A}\mathbf{x} \le \mathbf{b} \quad \text{és} \quad \mathbf{A}^T\mathbf{y} \ge \mathbf{c}\ ,$$

ahonnan következik, hogy $\mathbf{x}$ megengedett megoldása a primál feladatnak, és $\mathbf{y}$ megengedett megoldása a duál feladatnak. (8.32) harmadik egyenlőtlenségéből azt kapjuk, hogy

$$\mathbf{b}^T\mathbf{y} \le \mathbf{c}^T\mathbf{x}\ .$$

Tudjuk azonban

$$\mathbf{b}^T\mathbf{y} \ge (\mathbf{x}^T\mathbf{A}^T)\mathbf{y} = \mathbf{x}^T(\mathbf{A}^T\mathbf{y}) \ge \mathbf{x}^T\mathbf{c} = \mathbf{c}^T\mathbf{x}\ ,$$

tehát $\mathbf{b}^T\mathbf{y} = \mathbf{c}^T\mathbf{x}$, melyből az következik, hogy a primál feladat célfüggvénye $\mathbf{x}$-ben és a duál feladat célfüggvénye $\mathbf{y}$-ban egyenlő. Ekkor az erős dualitási tétel miatt $\mathbf{x}$ optimális megoldása a primál feladatnak és $\mathbf{y}$ optimális megoldása a duál feladatnak. ■

8.17. példa. *Lineáris programozás.* Tekintsük a következő lineáris programozási feladatot:

$$\begin{aligned} x_1 + 2x_2 &\to \max \\ x_1 &\ge 0 \\ -x_1 + x_2 &\ge 1 \\ 5x_1 + 7x_2 &\le 25\ . \end{aligned}$$

Először fel kell írnunk a feladatot mint primál feladatot. Vezessünk be két új változót:

$$x_2^+ = \begin{cases} x_2, & \text{ha } x_2 \ge 0\ , \\ 0 & \text{különben}\ , \end{cases}$$

$$x_2^- = \begin{cases} -x_2, & \text{ha } x_2 < 0\ , \\ 0 & \text{különben}\ . \end{cases}$$

és szorozzuk meg a második egyenlőtlenséget -1-gyel. Ekkor a következő feladatot kapjuk:

$$\begin{aligned} x_1 + 2x_2^+ - 2x_2^- &\to \max \\ x_1, x_2^+, x_2^- &\ge 0 \\ x_1 - x_2^+ + x_2^- &\le -1 \\ 5x_1 + 7x_2^+ - 7x_2^- &\le 25\ . \end{aligned}$$

Így

$$\mathbf{A} = \begin{pmatrix} 1 & -1 & 1 \\ 5 & 7 & -7 \end{pmatrix},\quad \mathbf{b} = \begin{pmatrix} -1 \\ 25 \end{pmatrix},\quad \mathbf{c}^T = (1, 2, -2)\ .$$

A $\mathbf{P}$ mátrix pedig a következő lesz:

$$\mathbf{P} = \left(\begin{array}{cccccccc} 0 & 0 & \vdots & 1 & -1 & 1 & \vdots & 1 \\ 0 & 0 & \vdots & 5 & 7 & -7 & \vdots & -25 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ -1 & -5 & \vdots & 0 & 0 & 0 & \vdots & 1 \\ 1 & -7 & \vdots & 0 & 0 & 0 & \vdots & 2 \\ -1 & 7 & \vdots & 0 & 0 & 0 & \vdots & -2 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ -1 & 25 & \vdots & -1 & -2 & 2 & \vdots & 0 \end{array}\right).$$

8.2.8. A Neumann-módszer

A fiktív lejátszás módszere egy iterációs algoritmus, ahol a játékosok minden lépésben hozzáigazítják stratégiáikat a többi játékos stratégiáihoz. Ez a módszer tehát úgy tekinthető, mint egy diszkrét rendszer megvalósulása, ahol a játékosok stratégiaválasztásai az állapotváltozók. Neumann János a szimmetrikus játékok esetére bevezetett egy folytonos megközelítést, ahol a játékosok folyamatosan módosítják a stratégiáikat. Ez a módszer alkalmazható tetszőleges mátrixjátékra, hiszen – amint korábban láttuk – bármely mátrixjáték ekvivalens egy szimmetrikus mátrixjátékkal. Ez a módszer szintén használható lineáris programozási feladatok megoldására, hiszen korábban láttuk, hogy minden primál-duál feladatpár visszavezethető egy szimmetrikus mátrixjáték egyensúlyi problémájára.

Legyen a továbbiakban $\mathbf{P}$ n-edrendű ferdén-szimmetrikus mátrix. A $\mathcal{P}_2$ játékos $\mathbf{y}(t)$ stratégiája egy függvény, mely a $t \geq 0$ idő változótól függ. Mielőtt a rendszer dinamikáját felírnánk, bevezetjük a következő jelöléseket:

$$\begin{array}{lccccl} u_i: & \mathbb{R}^n & \to & \mathbb{R}, & u_i(\mathbf{y}(t)) & = \mathbf{e}_i^T \mathbf{P}\mathbf{y}(t) \quad (i = 1, 2, \ldots, n)\,, \\ \phi: & \mathbb{R} & \to & \mathbb{R}, & \phi(u_i) & = \max\{0, u_i\}\,, \\ \Phi: & \mathbb{R}^n & \to & \mathbb{R}, & \Phi(\mathbf{y}(t)) & = \sum_{i=1}^n \phi(u_i(\mathbf{y}(t)))\,. \end{array} \tag{8.33}$$

Oldjuk meg a következő nemlineáris kezdetiérték-feladatot tetszőleges $\mathbf{y}_0$-ra:

$$y_j'(t) = \phi(u_j(\mathbf{y}(t))) - \Phi(\mathbf{y}(t)) y_j(t), \quad y_j(0) = y_{j0} \quad (1 \leq j \leq n)\,. \tag{8.34}$$

Mivel a jobb oldalon lévő kifejezés folytonos, így (8.34)-nek van legalább egy megoldása. A jobb oldalon lévő kifejezés a következőképpen értelmezhető. Tegyük fel, hogy $\phi(u_j(\mathbf{y}(t))) > 0$. Ekkor ha $\mathcal{P}_2$ az $\mathbf{y}(t)$ stratégiát választja, akkor $\mathcal{P}_1$ pozitív kifizetést tud elérni az $\mathbf{e}_j$ stratégia választásával, mely választás negatív kifizetést eredményez a $\mathcal{P}_2$ játékosnak. Ha azonban $\mathcal{P}_2$ egyre úgy növeli $y_j(t)$-t, hogy $\mathbf{e}_j$ stratégiát választ ő is, akkor az $\mathbf{e}_j^T \mathbf{P} \mathbf{e}_j$ kifizetése nullává válik, tehát megnő. Ebből következik, hogy $\mathcal{P}_2$ érdeke $y_j(t)$ növelése. Pontosan ezt fejezi ki a jobb oldali kifejezés első tagja. A második tag azt biztosítja, hogy $\mathbf{y}(t)$ valószínűségi vektor maradjon minden $t \geq 0$-ra.

(8.34) jobb oldalának kiszámításához minden t-re N^2+N szorzásra van szükség. A teljes

számítási költség függ a megoldás intervallumának a hosszától, a választott lépésnagyságtól, és a differenciálegyenletet megoldó módszer megválasztásától.

8.14. tétel. *Tegyük fel, hogy $t_1, t_2, \ldots$ egy szigorúan növekedő nemkorlátos sorozat. Ekkor az $\mathbf{y}(t_n)$ sorozat minden torlódási pontja egyensúlyi stratégia, és létezik egy olyan c konstans, hogy*

$$\mathbf{e}_i^T \mathbf{P}\mathbf{y}(t_k) \leq \frac{\sqrt{n}}{c + t_k} \quad (i = 1, 2, \ldots, n) \; . \tag{8.35}$$

Bizonyítás. Először azt kell megmutatnunk, hogy $\mathbf{y}(t)$ valószínűségi vektor minden $t \geq 0$-ra. Tegyük fel, hogy valamilyen j-re és $t_1 > 0$-ra $y_j(t_1) < 0$. Legyen

$$t_0 = \sup\{t | 0 < t < t_1, y_j(t) \geq 0\} \; .$$

Ekkor $y_j(t)$ folytonossága és $y_j(0) \geq 0$ miatt $y_j(t_0) = 0$, és minden $\tau \in (t_0, t_1)$-re, $y_j(\tau) < 0$. Az előzőekből következik, hogy minden $\tau \in (t_0, t_1]$-re

$$y_j'(\tau) = \phi(u_j(\mathbf{y}(\tau))) - \Phi(\mathbf{y}(\tau)) y_j(\tau) \geq 0 \; .$$

A Lagrange-középérték tétel miatt létezik $\tau \in (t_0, t_1)$, hogy

$$y_j(t_1) = y_j(t_0) + y_j'(\tau)(t_1 - t_0) \geq 0 \; ,$$

ami ellentmondás. Tehát $y_j(t)$ nemnegatív minden $t \geq 0$-ra. A következőkben megmutatjuk, hogy $\sum_{j=1}^n y_j(t) = 1$ minden $t \geq 0$-ra. Legyen $f(t) = 1 - \sum_{j=1}^n y_j(t)$, ekkor

$$f'(t) = -\sum_{j=1}^n y_j'(t) = -\sum_{j=1}^n \phi(u_j(\mathbf{y}(t))) + \Phi(\mathbf{y}(t))(\sum_{j=1}^n y_j(t)) = \Phi(\mathbf{y}(t))(1 - \sum_{j=1}^n y_j(t)) \; ,$$

tehát $f(t)$ megoldása a következő homogén rendszernek:

$$f'(t) = -\Phi(\mathbf{y}(t)) f(t)$$

az $f(0) = 1 - \sum_{j=1}^n y_{j0} = 0$ kezdetiérték mellett. Tehát, minden $t \geq 0$-ra $f(t) = 0$, ami azt jelenti, hogy $\mathbf{y}(t)$ valószínűségi vektor minden $t \geq 0$-ra.

Tegyük fel, hogy valamilyen $t \geq 0$ mellett $y_i(u_i(\mathbf{y}(t))) > 0$. Ekkor

$$\begin{aligned} \frac{d}{dt}\phi(u_i(\mathbf{y}(t))) &= \sum_{j=1}^n p_{ij} y_j'(t) = \sum_{j=1}^n p_{ij}[\phi(u_j(\mathbf{y}(t))) - \Phi(\mathbf{y}(t)) y_j(t)] \\ &= \sum_{j=1}^n p_{ij}\phi(u_j(\mathbf{y}(t))) - \Phi(\mathbf{y}(t))\phi(u_i(\mathbf{y}(t))) \; . \end{aligned} \tag{8.36}$$

Szorozzuk meg mindkét oldalt $\phi(u_i(\mathbf{y}(t)))$-vel, és adjuk össze az így kapott egyenlőségeket $i = 1, 2, \ldots, n$-re:

$$\sum_{i=1}^n \phi(u_i(\mathbf{y}(t))) \frac{d}{dt}\phi(u_i(\mathbf{y}(t))) = \sum_{i=1}^n \sum_{j=1}^n p_{ij}\phi(u_i(\mathbf{y}(t)))\phi(u_j(\mathbf{y}(t))) - \Phi(\mathbf{y}(t))(\sum_{i=1}^n \phi^2(u_i(\mathbf{y}(t)))) \; . \tag{8.37}$$

Mivel $\mathbf{P}$ ferdén-szimmetrikus, így az első tag nulla. Vegyük észre, hogy a fenti egyenlőség a töréspont (ahol $\phi(u_i(\mathbf{y}(t)))$ deriváltja nem létezik) kivételével akkor is érvényes marad, ha $\phi(u_i(\mathbf{y}(t))) = 0$, így (8.36) igaz marad.

Most tegyük fel, hogy valamilyen pozitív t-re $\Phi(\mathbf{y}(t)) = 0$. Ekkor minden i-re $\phi(u_i(\mathbf{y}(t))) = 0$. Mivel (8.37) átírható

$$\frac{1}{2}\frac{d}{dt}\Psi(\mathbf{y}(t)) = -\Phi(\mathbf{y}(t))\Psi(\mathbf{y}(t)) \tag{8.38}$$

formába, ahol

$$\Psi : \mathbb{R}^n \to \mathbb{R} \quad \text{és} \quad \Psi(\mathbf{y}(t)) = \sum_{i=1}^{n} \phi^2(u_i(\mathbf{y}(t))) \ ,$$

így látható, hogy $\Psi(\mathbf{y}(t))$ kielégíti a homogén egyenletet nulla kezdetiérték mellett, tehát a megoldás nulla marad minden $\tau \geq t$-re. Ebből következik, hogy $\phi(u_i(\mathbf{y}(\tau))) = 0$ megoldásra $\mathbf{P}\mathbf{y}(\tau) \leq \mathbf{0}$, azaz $\mathbf{y}(\tau)$ egyensúly.

Ha $\Phi(\mathbf{y}(\mathbf{t})) > 0$ minden $t \geq 0$-ra, akkor $\Psi(\mathbf{y}(t)) > 0$, és könnyen látható, hogy

$$\frac{1}{2}\frac{d}{dt}\Psi(\mathbf{y}(t)) \leq -\sqrt{\Psi(\mathbf{y}(t))}\Psi(\mathbf{y}(t)) \ ,$$

azaz

$$\frac{1}{2}\frac{d}{dt}\Psi(\mathbf{y}(t))(\Psi(\mathbf{y}(t)))^{-\frac{3}{2}} \leq -1 \ .$$

Mindkét oldalt a $[0, t]$ intervallumon integrálva azt kapjuk, hogy

$$-\Psi(\mathbf{y}(t))^{-(1/2)} + c \leq -t \ ,$$

ahol $c = (\Psi(\mathbf{y}(0)))^{-(1/2)}$, melyből következik

$$(\Psi(\mathbf{y}(t)))^{1/2} \leq \frac{1}{c+t} \ . \tag{8.39}$$

A Cauchy–Schwartz-egyenlőtlenség alkalmazásával kapjuk, hogy

$$\mathbf{e}_i^T \mathbf{P}\mathbf{y}(t) = u_i(\mathbf{y}(t)) \leq \phi(u_i(\mathbf{y}(t))) \leq \Phi(\mathbf{y}(t)) \leq \sqrt{n\Psi(\mathbf{y}(t))} \leq \frac{\sqrt{n}}{c+t} \ , \tag{8.40}$$

mely egyenlőtlenség az u_i folytonossága miatt még a töréspontokban is igaz. Végül vegyük a következő sorozatot: $\{\mathbf{y}(t_k)\}$, ahol t_k monoton növekedő nem korlátos sorozat. Mivel $\mathbf{y}(t_k)$-k valószínűségi vektorok, így korlátosak, tehát van legalább egy $\mathbf{y}^\star$ torlódási pontjuk. (8.40)-ből t_k-val a végtelenbe tartva azt kapjuk, hogy $\mathbf{P}\mathbf{y}^\star \leq \mathbf{0}$, tehát $\mathbf{y}^\star$ egyensúly. ■

8.18. példa. *Negyedik mátrixjáték.* Tekintsük a 8.13. példában bevezetett mátrixjátékot. A Neumann-módszer alkalmazásához először fel kell írni az ekvivalens szimmetrikus mátrixjátékot. Az átírás módszere, mely a 8.12. tételben található, megköveteli, hogy a mátrix elemei pozitívak legyenek. Anélkül,

hogy az egyensúlyok megváltoznának, az $\mathbf{A}$ mátrix minden eleméhez hozzáadhatunk 2-t, ekkor a következő mátrixot kapjuk:

$$\mathbf{A} = \begin{pmatrix} 4 & 3 & 2 \\ 4 & 2 & 5 \\ 1 & 5 & 5 \end{pmatrix},$$

és a fent említett módszer segítségével

$$\mathbf{P} = \left(\begin{array}{ccc:ccc:c} 0 & 0 & 0 & 4 & 3 & 2 & -1 \\ 0 & 0 & 0 & 4 & 2 & 5 & -1 \\ 0 & 0 & 0 & 1 & 5 & 5 & -1 \\ \hdashline -4 & -4 & -1 & 0 & 0 & 0 & 1 \\ -3 & -2 & -5 & 0 & 0 & 0 & 1 \\ -2 & -5 & -5 & 0 & 0 & 0 & 1 \\ \hdashline 1 & 1 & 1 & -1 & -1 & -1 & 0 \end{array}\right).$$

A (8.34) differenciálegyenletet a negyedrendű Runge–Kutta-módszerrel oldottuk meg a $[0, 100]$ intervallumon, $h = 0.01$ lépésnagysággal az $\mathbf{y}(0) = (1, 0, \ldots, 0)^T$ kezdetiérték mellett. $\mathbf{y}(100)$-ra kaptuk a következő közelítő értékeket:

$$\mathbf{x} \approx (0.563619, 0.232359, 0.241988),$$

$$\mathbf{y} \approx (0.485258, 0.361633, 0.115144).$$

Ezen közelítés az eredeti játék egy egyensúlyának becslése. Hasonlítsuk össze ezeket az értékeket az egyensúlyi vektorpárral:

$$\mathbf{x} = \left(\frac{4}{7}, \frac{4}{21}, \frac{5}{21}\right) \quad \text{és} \quad \mathbf{y} = \left(\frac{3}{7}, \frac{3}{7}, \frac{1}{7}\right).$$

Látható, hogy a maximális hiba 0.067.

8.2.9. Átlósan szigorúan konkáv játékok

Tekintsünk egy N személyes folytonos játékot, és tegyük fel, hogy a 8.2.3. pont feltevései teljesülnek. Tegyük fel továbbá, hogy minden S_k korlátos minden k-ra, $\mathbf{g}_k$ minden komponense szerint konkáv, és f_k konkáv s_k-ban tetszőlegesen rögzített $s_1, \ldots, s_{k-1}, s_{k+1}, \ldots, N$ mellett. A 8.3. tétel miatt a fenti feltételek mellett a játéknak van legalább egy egyensúlya. Általában az egyensúly unicitása még akkor sem biztosított, ha minden f_k szigorúan kvázi-konkáv s_k szerint. A következő példa ezt a tényt mutatja be.

8.19. példa. *Ellenpélda.* Tekintsük a következő kétszemélyes játékot: $S_1 = S_2 = [0, 1]$ és $f_1(s_1, s_2) = f_2(s_1, s_2) = 1-(s_1-s_2)^2$. Világos, hogy mindkét kifizetőfüggvény szigorúan konkáv, és mégis végtelen sok egyensúly van: $s_1^\star = s_2^\star \in [0, 1]$.

Válasszunk egy nemnegatív $\mathbf{r} \in \mathbb{R}^N$ vektort, és definiáljuk a következő függvényt:

$$\mathbf{h} : \mathbb{R}^M \to \mathbb{R}^M, \quad \mathbf{h}(\mathbf{s}, \mathbf{r}) = \begin{pmatrix} r_1 \nabla_1 f_1(\mathbf{s})^T \\ r_2 \nabla_2 f_2(\mathbf{s})^T \\ \vdots \\ r_N \nabla_N f_N(\mathbf{s})^T \end{pmatrix}, \tag{8.41}$$

ahol $M = \sum_{k=1}^{N} n_k$, és $\nabla_k f_k$ az f_k függvény s_k szerinti gradiens(sor)vektora. Egy játékot ***átlósan szigorúan konkávnak*** hívunk, ha minden $\mathbf{s}^{(1)}, \mathbf{s}^{(2)} \in S$, $\mathbf{s}^{(1)} \neq \mathbf{s}^{(2)}$-re, és valamilyen $\mathbf{r} \geq \mathbf{0}$-ra

$$(\mathbf{s}^{(1)} - \mathbf{s}^{(2)})^T (\mathbf{h}(\mathbf{s}^{(1)}, \mathbf{r}) - \mathbf{h}(\mathbf{s}^{(2)}, \mathbf{r})) < 0 \,. \tag{8.42}$$

8.15. tétel. *Egy átlósan szigorúan konkáv játéknak pontosan egy egyensúlya van.*

Bizonyítás. A 8.3. tétel miatt létezik egyensúly. Az egyértelműség bizonyítása céljából tegyük fel, hogy $\mathbf{s}^{(1)}$ és $\mathbf{s}^{(2)}$ két egyensúly, melyek kielégítik (8.9) feltételeket és $\mathbf{s}^{(1)} \neq \mathbf{s}^{(2)}$. Ekkor $l = 1, 2$-re

$$\begin{aligned} \mathbf{u}_k^{(l)T} \mathbf{g}_k(s_k^{(l)}) &= 0 \\ \nabla_k f_k(\mathbf{s}^{(l)}) + \mathbf{u}_k^{(l)T} \nabla_k \mathbf{g}_k(s_k^{(l)}) &= \mathbf{0}^T \,. \end{aligned}$$

A második egyenlőséget a következő formába lehet átírni:

$$\nabla_k f_k(\mathbf{s}^{(l)}) + \sum_{j=1}^{m_k} u_{kj}^{(l)} \nabla_k g_{kj}(s_k^{(l)}) = 0 \,, \tag{8.43}$$

ahol $u_{kj}^{(l)}$ a $\mathbf{u}_k^{(l)}$-nak, és g_{kj} a $\mathbf{g}_k$ j-edik komponense. Szorozzuk meg (8.43)-at $(r_k(s_k^{(2)} - s_k^{(1)})^T)$-tal $l = 1$ esetén, és $r_k(s_k^{(1)} - s_k^{(2)})^T$-tal $l = 2$ esetén. Adjuk össze az így kapott egyenlőségeket $k = 1, 2, \ldots, N$-re. Ekkor a következőt kapjuk:

$$\begin{aligned} 0 = \{(\mathbf{s}^{(2)} - \mathbf{s}^{(1)})\mathbf{h}(\mathbf{s}^{(1)}, \mathbf{r}) + (\mathbf{s}^{(1)} - \mathbf{s}^{(2)})\mathbf{h}(\mathbf{s}^{(2)}, \mathbf{r})\} \\ + \sum_{k=1}^{N} \sum_{j=1}^{m_k} r_k [u_{kj}^{(1)} (s_k^{(2)} - s_k^{(1)})^T \nabla_k g_{kj}(s_k^{(1)}) + u_{kj}^{(2)} (s_k^{(1)} - s_k^{(2)})^T \nabla_k g_{kj}(s_k^{(2)})] \,. \end{aligned} \tag{8.44}$$

Vegyük észre, hogy az átlósan szigorú konkavitás miatt az első két tag összege pozitív, $\mathbf{g}_k$ komponenseinek a konkavitása miatt pedig

$$(s_k^{(2)} - s_k^{(1)})^T \nabla_k g_{kj}(s_k^{(1)}) \geq g_{kj}(s_k^{(2)}) - g_{kj}(s_k^{(1)})$$

és

$$(s_k^{(1)} - s_k^{(2)})^T \nabla_k g_{kj}(s_k^{(2)}) \geq g_{kj}(s_k^{(1)}) - g_{kj}(s_k^{(2)}) \,.$$

Tudjuk, hogy minden k-ra, l-re

$$0 = \mathbf{u}_k^{(l)T} \mathbf{g}_k(s_k^{(l)}) = \sum_{j=1}^{m_k} u_{kj}^{(l)} g_{kj}(s_k^{(l)}) \,,$$

ekkor (8.44)-ből kapjuk, hogy

$$0 > \sum_{k=1}^{N}\sum_{j=1}^{m_k} r_k[u_{kj}^{(1)}(g_{kj}(s_k^{(2)}) - g_{kj}(s_k^{(1)})) + u_{kj}^{(2)}(g_{kj}(s_k^{(1)}) - g_{kj}(s_k^{(2)}))]$$

$$= \sum_{k=1}^{N}\sum_{j=1}^{m_k} r_k[u_{kj}^{(1)} g_{kj}(s_k^{(2)}) + u_{kj}^{(2)} g_{kj}(s_k^{(1)})] \geq 0 \ ,$$

ami nyilvánvaló ellentmondás, tehát $\mathbf{s}^{(1)} = \mathbf{s}^{(2)}$. ■

Az egyensúly egyértelműségének ellenőrzése

A következő tételben egy olyan, a gyakorlati esetekben nagyon hasznos módszert mutatunk be, melynek segítségével ellenőrizhetjük egy N személyes játék átlósan szigorúan konkavitását.

8.16. tétel. *Tegyük fel, hogy S konvex, f_k kétszer folytonosan differenciálható minden k-ra, és létezik $\mathbf{r} \geq \mathbf{0}$, hogy $\mathbf{J}(\mathbf{s},\mathbf{r}) + \mathbf{J}(\mathbf{s},\mathbf{r})^T$ negatív definit, ahol $\mathbf{J}(\mathbf{s},\mathbf{r})$ a $\mathbf{h}(\mathbf{s},\mathbf{r})$ Jakobi-mátrixa. Ekkor a játék átlósan szigorúan konkáv.*

Bizonyítás. Legyen $\mathbf{s}^{(1)}, \mathbf{s}^{(2)} \in S$ és $\mathbf{s}^{(1)} \neq \mathbf{s}^{(2)}$. Ekkor minden $\alpha \in [0,1]$-re $\mathbf{s}(\alpha) = \alpha\mathbf{s}^{(1)} + (1-\alpha)\mathbf{s}^{(2)} \in S$ és

$$\frac{d}{d\alpha}\mathbf{h}(\mathbf{s}(\alpha),\mathbf{r}) = \mathbf{J}(\mathbf{s}(\alpha),\mathbf{r})(\mathbf{s}^{(1)} - \mathbf{s}^{(2)}) \ .$$

Mindkét oldalt $[0,1]$-en integrálva

$$\mathbf{h}(\mathbf{s}^{(1)},\mathbf{r}) - \mathbf{h}(\mathbf{s}^{(2)},\mathbf{r}) = \int_0^1 \mathbf{J}(\mathbf{s}(\alpha),\mathbf{r})(\mathbf{s}^{(1)} - \mathbf{s}^{(2)})d\alpha \ ,$$

és mindkét oldalt újra megszorozva $(\mathbf{s}^{(1)} - \mathbf{s}^{(2)})^T$-tal

$$(\mathbf{s}^{(1)} - \mathbf{s}^{(2)})^T(\mathbf{h}(\mathbf{s}^{(1)},\mathbf{r}) - \mathbf{h}(\mathbf{s}^{(2)},\mathbf{r})) = \int_0^1 (\mathbf{s}^{(1)} - \mathbf{s}^{(2)})^T\mathbf{J}(\mathbf{s}(\alpha),\mathbf{r})(\mathbf{s}^{(1)} - \mathbf{s}^{(2)})d\alpha$$

$$= \frac{1}{2}\int_0^1 (\mathbf{s}^{(1)} - \mathbf{s}^{(2)})^T(\mathbf{J}(\mathbf{s}(\alpha),\mathbf{r}) + \mathbf{J}(\mathbf{s}(\alpha),\mathbf{r})^T)(\mathbf{s}^{(1)} - \mathbf{s}^{(2)})d\alpha < 0 \ ,$$

tehát a bizonyítást befejeztük. ■

8.20. példa. *Egyszerű kétszemélyes játék.* Tekintsük a következő egyszerű kétszemélyes játékot, ahol $S_1 = S_2 = [0,1]$, és a kifizetőfüggvények:

$$f_1(s_1,s_2) = -s_1^2 + s_1 - s_1s_2$$

és

$$f_2(s_1,s_2) = -s_2^2 + s_2 - s_1s_2 \ .$$

Könnyen látható, hogy – az átlósan szigorú konkavitáson kívül – minden tulajdonság, amit ebben az alfejezetben feltettünk, teljesül. A 8.16. tételt használjuk a hiányzó tulajdonság megmutatására. Ebben az esetben

$$\nabla_1 f_1(s_1, s_2) = -2s_1 + 1 - s_2, \quad \nabla_2 f_2(s_1, s_2) = -2s_2 + 1 - s_1 ,$$

így

$$\mathbf{h}(\mathbf{s}, \mathbf{r}) = \begin{pmatrix} r_1(-2s_1 + 1 - s_2) \\ r_2(-2s_2 + 1 - s_1) \end{pmatrix} .$$

A Jakobi-mátrix:

$$\mathbf{J}(\mathbf{s}, \mathbf{r}) = \begin{pmatrix} -2r_1 & -r_1 \\ -r_2 & -2r_2 \end{pmatrix} .$$

Megmutatjuk, hogy valamilyen $\mathbf{r} \geq \mathbf{0}$-ra a

$$\mathbf{J}(\mathbf{s}, \mathbf{r}) + \mathbf{J}(\mathbf{s}, \mathbf{r})^T = \begin{pmatrix} -4r_1 & -r_1 - r_2 \\ -r_1 - r_2 & -4r_2 \end{pmatrix}$$

mátrix negatív definit. Legyen például $r_1 = r_2 = 1$, ekkor a fenti mátrix

$$\begin{pmatrix} -4 & -2 \\ -2 & -4 \end{pmatrix} ,$$

a karakterisztikus polinom:

$$\phi(\lambda) = \det \begin{pmatrix} -4 - \lambda & -2 \\ -2 & -4 - \lambda \end{pmatrix} = \lambda^2 + 8\lambda + 12 ,$$

mely polinomnak a két gyöke $\lambda_1 = -2$, $\lambda_2 = -6$.

Az egyensúly iteratív kiszámítása

A 8.4. tételben láttuk, hogy $\mathbf{s}^\star \in S$ pontosan akkor egyensúly, ha

$$\mathbf{H}_\mathbf{r}(\mathbf{s}^\star, \mathbf{s}^\star) \geq \mathbf{H}_\mathbf{r}(\mathbf{s}^\star, \mathbf{s}) \tag{8.45}$$

minden $\mathbf{s} \in S$-re, ahol $\mathbf{H}_\mathbf{r}$ a (8.4)-ben definiált összegzőfüggvény. A következőkben feltesszük, hogy az alfejezet elején feltett tulajdonságok érvényesek, és létezik olyan $\mathbf{r} \geq \mathbf{0}$, hogy (8.42) érvényes.

Először a variációs egyenlőtlenség és (8.45) ekvivalenciáját mutatjuk meg.

8.17. tétel. *Egy $\mathbf{s}^\star \in S$ vektor pontosan akkor elégíti ki (8.45)-öt, ha*

$$\mathbf{h}(\mathbf{s}^\star, \mathbf{r})^T (\mathbf{s} - \mathbf{s}^\star) \leq 0 \tag{8.46}$$

minden $\mathbf{s} \in S$ stratégiára, ahol $\mathbf{h}(\mathbf{s}, \mathbf{r})$-et (8.41)-ben definiáltuk.

Bizonyítás. Tegyük fel, hogy $\mathbf{s}^\star$ kielégíti (8.45)-öt. Ekkor $\mathbf{H}_\mathbf{r}(\mathbf{s}^\star, \mathbf{s})$ – mint $\mathbf{s}$ argumentumú függvény – felveszi a maximumát $\mathbf{s} = \mathbf{s}^\star$-ban, tehát

$$\nabla_\mathbf{s} \mathbf{H}_\mathbf{r}(\mathbf{s}^\star, \mathbf{s}^\star)(\mathbf{s} - \mathbf{s}^\star) \leq 0$$

minden $\mathbf{s} \in S$ stratégiára. Mivel $\nabla_\mathbf{s} \mathbf{H}_\mathbf{r}(\mathbf{s}^\star, \mathbf{s}^\star) = \mathbf{h}(\mathbf{s}^\star, \mathbf{r})$, így $\mathbf{s}^\star$ kielégíti (8.46)-t.

Most tegyük fel, hogy $\mathbf{s}^\star$ kielégíti (8.46)-ot. $\mathbf{H}_\mathbf{r}(\mathbf{s}^\star, \mathbf{s}^\star)$ $\mathbf{s}$ szerinti konkavitása, és a játék

átlósan szigorú konkavitása miatt

$$\mathbf{H}_{\mathbf{r}}(\mathbf{s}^\star,\mathbf{s}^\star)-\mathbf{H}_{\mathbf{r}}(\mathbf{s}^\star,\mathbf{s})\geq\mathbf{h}(\mathbf{s},\mathbf{r})^T(\mathbf{s}^\star-\mathbf{s})\geq\mathbf{h}(\mathbf{s},\mathbf{r})^T(\mathbf{s}^\star-\mathbf{s})+\mathbf{h}(\mathbf{s}^\star,\mathbf{r})^T(\mathbf{s}-\mathbf{s}^\star)>0\ ,$$

tehát $\mathbf{s}^\star$ kielégíti (8.45)-öt. ■

Látható, hogy bármely – a variációs egyenlőtlenség megoldására alkalmas – módszer használható a játék egyensúlyi problémájának megoldására.

A következőkben definiálunk egy olyan speciális kétszemélyes, zérusösszegű-játékot, melynek egyensúlyi problémája ekvivalens az eredeti N személyes játék egyensúlyi problémájával.

8.18. tétel. *Az $\mathbf{s}^\star\in S$ vektor pontosan akkor elégíti ki (8.46)-ot, ha $(\mathbf{s}^\star,\mathbf{s}^\star)$ egyensúlya egy olyan kétszemélyes játéknak, ahol mindkét stratégiahalmaz S, a kifizetőfüggvények pedig $f_1(\mathbf{s},\mathbf{z})=f(\mathbf{s},\mathbf{z})$ és $f_2(\mathbf{s},\mathbf{z})=-f(\mathbf{s},\mathbf{z})$, ahol $f(\mathbf{s},\mathbf{z})=\mathbf{h}(\mathbf{z},\mathbf{r})^T(\mathbf{s}-\mathbf{z})$.*

Bizonyítás. Először tegyük fel, hogy $\mathbf{s}^\star\in S$ kielégíti (8.45)-t. Ekkor kielégíti (8.46)-t is, így

$$f(\mathbf{s},\mathbf{s}^\star)\leq 0=f(\mathbf{s}^\star,\mathbf{s}^\star)\ .$$

Még azt kell megmutatnunk, hogy

$$-f(\mathbf{s}^\star,\mathbf{s})\leq 0=-f(\mathbf{s}^\star,\mathbf{s}^\star)\ .$$

Indirekt módon tegyük el, hogy valamilyen $\mathbf{s}$-re $f(\mathbf{s}^\star,\mathbf{s})<0$. Ekkor (8.42) és (8.46) miatt

$$\begin{aligned}0>f(\mathbf{s}^\star,\mathbf{s}) &= \mathbf{h}(\mathbf{s},\mathbf{r})^T(\mathbf{s}^\star-\mathbf{s})>\mathbf{h}(\mathbf{s},\mathbf{r})^T(\mathbf{s}^\star-\mathbf{s})+(\mathbf{s}-\mathbf{s}^\star)^T(\mathbf{h}(\mathbf{s},\mathbf{r})-\mathbf{h}(\mathbf{s}^\star,\mathbf{r}))\\ &= \mathbf{h}(\mathbf{s}^\star,\mathbf{r})^T(\mathbf{s}^\star-\mathbf{s})\geq 0\ ,\end{aligned}$$

ami ellentmondás. Most tegyük fel, hogy $(\mathbf{s}^\star,\mathbf{s}^\star)$ egyensúlya a tételben bevezetett játéknak. Ekkor tetszőleges $\mathbf{s},\mathbf{z}\in S$-re

$$f(\mathbf{s},\mathbf{s}^\star)\leq f(\mathbf{s}^\star,\mathbf{s}^\star)=0\leq f(\mathbf{s},\mathbf{z})\ .$$

Az első egyenlőtlenség átírható a következő formába:

$$\mathbf{h}(\mathbf{s}^\star,\mathbf{r})^T(\mathbf{s}-\mathbf{s}^\star)\leq 0\ ,$$

tehát (8.46) teljesül, így teljesül (8.45) is. ■

Tekintsük a következő iterációs eljárást.

Legyen $\mathbf{s}^{(1)}\in S$ tetszőleges, és oldjuk meg a következő feladatot:

$$\begin{aligned}f(\mathbf{s},\mathbf{s}^{(1)}) &\rightarrow \max\\ \mathbf{s} &\in S\ .\end{aligned}\tag{8.47}$$

Jelöljük $\mathbf{s}^{(2)}$-vel (8.47) feladat megoldását, és legyen $\mu_1=f(\mathbf{s}^{(2)},\mathbf{s}^{(1)})$. Ha $\mu_1=0$, akkor minden $\mathbf{s}\in S$-re

$$f(\mathbf{s},\mathbf{s}^{(1)})=\mathbf{h}(\mathbf{s}^{(1)},\mathbf{r})^T(\mathbf{s}-\mathbf{s}^{(1)})\leq 0\ ,$$

így 8.17. tétel miatt $\mathbf{s}^{(1)}$ egyensúly. Mivel $f(\mathbf{s}^{(1)}, \mathbf{s}^{(1)}) = 0$, így feltesszük, hogy $\mu_1 > 0$. $k \geq 2$ általános lépésben már van k vektorunk $\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \ldots, \mathbf{s}^{(k)}$, és $k-1$ skalárunk $\mu_1, \mu_2, \ldots, \mu_{k-1} > 0$. Ekkor a következő $\mathbf{s}^{(k+1)}$ vektor és μ_k skalár megoldása a következő feladatnak:

$$\begin{array}{rcll} \mu & \to & \max & \\ f(\mathbf{s}, \mathbf{s}^{(i)}) & \geq & \mu & (i = 1, 2, \ldots, k) \\ \mathbf{s} & \in & S\ . & \end{array} \tag{8.48}$$

Vegyük észre, hogy

$$f(\mathbf{s}^{(k)}, \mathbf{s}^{(i)}) \geq \mu_{k-1} \geq 0 \qquad (i = 1, 2, \ldots, k-1)$$

és

$$f(\mathbf{s}^{(k)}, \mathbf{s}^{(k)}) = 0.$$

Ekkor tudjuk, hogy $\mu_k \geq 0$.

A fenti algoritmus konvergencia tételének tárgyalása előtt megjegyezzük, hogy abban a speciális esetben, amikor a stratégiahalmazok lineáris egyenlőtlenségekkel vannak megadva (tehát a $\mathbf{g}_k$ függvények lineárisak), a (8.48) feladat minden feladata lineáris, tehát minden iterációs lépésben egy lineáris programozási feladatot kell megoldanunk.

Ha lineáris esetben a szimplex módszert alkalmazzuk minden iterációs lépésben, akkor a számítási költség exponenciális, tehát az egész eljárás számítási költsége exponenciális (rögzített lépésszám mellett).

8.19. tétel. *Az $\{\mathbf{s}^{(k)}\}$ fenti módszer által generált sorozatnak van olyan $\{\mathbf{s}^{(k_i)}\}$ részsorozata, mely az N személyes játék egyetlen egyensúlyához konvergál.*

Bizonyítás. A bizonyítás több lépésből áll.

Először azt mutatjuk meg, hogy $\lim_{k\to\infty} \mu_k = 0$. Mivel (8.48) minden iterációban egy új feltétellel bővül, tehát $\{\mu_k\}$ nem lehet növekvő. Tudjuk azt is, hogy $\{\mu_k\}$ nemnegatív, tehát konvergens. Az $\{\mathbf{s}^{(k)}\}$ sorozat korlátos, hiszen minden tagja a korlátos S halmazból való, így van egy $\{\mathbf{s}^{(k_i)}\}$ konvergens részsorozata. Vegyük észre, hogy

$$0 \leq \mu_{k_i-1} = \min_{1\leq k\leq k_i-1} \mathbf{h}(\mathbf{s}^{(k)}, \mathbf{r})^T (\mathbf{s}^{(k_i)} - \mathbf{s}^{(k)}) \leq \mathbf{h}(\mathbf{s}^{(k_{i-1})}, \mathbf{r})^T (\mathbf{s}^{(k_i)} - \mathbf{s}^{(k_{i-1})})\ ,$$

ahol az egyenlőtlenség jobb oldala nullához tart. Tehát $\mu_{k_i-1} \to 0$. Mivel $\{\mu_k\}$ monoton, így $\{\mu_k\} \to 0$.

Most legyen $\mathbf{s}^\star$ az N-személyes játék egyensúlya, és legyen

$$\delta(t) = \min\{(\mathbf{h}(\mathbf{s}, \mathbf{r}) - \mathbf{h}(\mathbf{z}, \mathbf{r}))^T (\mathbf{z} - \mathbf{s}) | \|\mathbf{s} - \mathbf{z}\| \geq t,\ \mathbf{z}, \mathbf{s} \in S\}\ . \tag{8.49}$$

(8.42) miatt $\delta(t) > 0$ minden $(t > 0)$-ra. Definiáljuk a k_i indexeket a következőképpen:

$$\delta(\|\mathbf{s}^{(k_i)} - \mathbf{s}^\star\|) = \min_{1\leq k\leq i} \delta(\|\mathbf{s}^{(k)} - \mathbf{s}^\star\|) \quad (i = 1, 2, \ldots)\ .$$

Ekkor minden $k = 1, 2, \ldots, i$-re

$$\begin{array}{rcl} \delta(\|\mathbf{s}^{(k_i)} - \mathbf{s}^\star\|) & \leq & (\mathbf{h}(\mathbf{s}^{(k)}, \mathbf{r}) - \mathbf{h}(\mathbf{s}^\star, \mathbf{r}))^T (\mathbf{s}^\star - \mathbf{s}^{(k)}) \\ & = & \mathbf{h}(\mathbf{s}^{(k)}, \mathbf{r})^T (\mathbf{s}^\star - \mathbf{s}^{(k)}) - \mathbf{h}(\mathbf{s}^\star, \mathbf{r})^T (\mathbf{s}^\star - \mathbf{s}^{(k)}) \\ & \leq & \mathbf{h}(\mathbf{s}^{(k)}, \mathbf{r})^T (\mathbf{s}^\star - \mathbf{s}^{(k)})\ , \end{array}$$

melyből (8.48) miatt következik:

$$\begin{aligned}\delta(\|\mathbf{s}^{(k_i)}-\mathbf{s}^\star\|) &\le \min_{1\le k\le i}\mathbf{h}(\mathbf{s}^{(k)},\mathbf{r})^T(\mathbf{s}^\star-\mathbf{s}^{(k)})\\ &\le \max_{\mathbf{s}\in S}\min_{1\le k\le i}\mathbf{h}(\mathbf{s}^{(k)},\mathbf{r})^T(\mathbf{s}-\mathbf{s}^{(k)})\\ &= \min_{1\le k\le i}\mathbf{h}(\mathbf{s}^{(k)},\mathbf{r})^T(\mathbf{s}^{(i+1)}-\mathbf{s}^{(k)})\\ &= \mu_i\ .\end{aligned}$$

Ebből következik, hogy $\lim_{i\to\infty}\delta(\|\mathbf{s}^{(k_i)}-\mathbf{s}^\star\|)\to 0$. Végül vegyük észre azt, hogy $\delta(t)$ rendelkezik a következő tulajdonságokkal:

1. $\delta(t)$ folytonos;
2. ha $t>0$, akkor $\delta(t)>0$ (ezt mutatja (8.49));
3. ha egy $\{t^{(k)}\}$ konvergens sorozatra $\delta(t^{(k)})\to 0$, akkor szükségszerűen $t^{(k)}\to 0$.

A 3. tulajdonság miatt $\|\mathbf{s}^{(k_i)}-\mathbf{s}^\star\|\to 0$, így $\mathbf{s}^{(k_i)}\to\mathbf{s}^\star$. ■

Gyakorlatok

8.2-1. Tekintsünk egy kétszemélyes játékot, ahol a stratégiahalmazok $S_1=S_2=[0,1]$, a kifizetőfüggvények: $f_1(x_1,x_2)=x_1^2+x_1x_2+2$ és $f_2(x_1,x_2)=x_1+x_2$. Mutassuk meg, hogy a játéknak egyetlen egyensúlya van, és számítsuk ki ezt az egyensúlyt. Mutassuk meg, hogy ebben a játékban a 8.3. tétel nem alkalmazható az egyensúly létezésének bizonyítására.

8.2-2. Tekintsük az „árháború” játékot, melyben két vállalat ármeghatározó. Tegyük fel, hogy p_1 a $\mathcal{P}_1$, p_2 pedig a $\mathcal{P}_2$ játékos egy stratégiája, ahol $p_1,p_2\in[0,p_{max}]$ (p_{max} egy rögzített pozitív valós szám), és a kifizetőfüggvények:

$$f_1(p_1,p_2)=\begin{cases}p_1, & \text{ha } p_1\le p_2\ ,\\ p_1-c, & \text{ha } p_1>p_2\ ,\end{cases}$$

$$f_2(p_1,p_2)=\begin{cases}p_2, & \text{ha } p_2\le p_1\ ,\\ p_2-c, & \text{ha } p_2>p_1\ ,\end{cases}$$

ahol $c<p_{max}$ rögzített. Van-e ennek a játéknak egyensúlya? Ha van egyensúlya, akkor hány van?

8.2-3. Egy tengeralattjáró rejtőzik a tenger egy részében. A tenger ezen része az egységnégyzettel modellezhető. A tengeralattjáró stratégiája az $\mathbf{x}\in[0,1]\times[0,1]$ rejtőzködési helye. Egy repülőgép bombázza az $\mathbf{y}=[0,1]\times[0,1]$-t, a tenger egy bizonyos helyét. A bombázott hely megválasztása ezen játékos stratégiája. A repülőgép kifizetése a tengeralattjárónak okozott kár nagysága: $f_2(\mathbf{x},\mathbf{y})=\alpha e^{-\beta\|\mathbf{x}-\mathbf{y}\|}$, míg a tengeralattjáró kifizetése a neki okozott kár ellentettje: $f_1(\mathbf{x},\mathbf{y})=-f_2(\mathbf{x},\mathbf{y})$. Van-e ennek a kétszemélyes játéknak egyensúlya?

8.2-4. A második-legjobb ár aukcióban egy áru kerül eladásra az N licitálók valamelyikének. A licitálók különbözőképpen értékelik a árut: $v_1<v_2<\cdots<v_N$. A licitálók egyidejűleg ajánlatot tesznek a árura úgy, hogy közben nem ismerik a többiek ajánlatát. A legmagasabb ajánlatot tevő kapja meg a árut, de a áruért csak a második legmagasabb ajánlatot kell fizetnie. Tehát a $\mathcal{P}_k$ játékos stratégiahalmaza $[0,\infty]$, stratégiája $x_k\in[0,\infty]$, és

a kifizetőfüggvénye:

$$f_k(x_1, x_2, \ldots, x_N) = \begin{cases} v_k - \max_{j \neq k} x_j, & \text{ha } x_k = \max_j x_j \,, \\ 0 & \text{különben} \,. \end{cases}$$

Határozzuk meg a $\mathcal{P}_k$ játékos legjobbválasz-leképezését. Van-e a játéknak egyensúlya?

8.2-5. Írjuk fel a Fan-egyenlőtlenséget a 8.2-1. gyakorlatra.

8.2-6. Írjuk fel és oldjuk meg a Fan-egyenlőtlenséget a 8.2-2. gyakorlatra.

8.2-7. Írjuk fel és oldjuk meg a Fan-egyenlőtlenséget a 8.2-4. gyakorlatra.

8.2-8. Tekintsük azt a kétszemélyes játékot, ahol a stratégiahalmazok $S_1 = S_2 = [0, 1]$, a kifizetőfüggvények pedig

$$f_1(x_1, x_2) = -(x_1 - x_2)^2 + 2x_1 - x_2 + 1$$

$$f_2(x_1, x_2) = -(x_1 - 2x_2)^2 - 2x_1 + x_2 - 1$$

Írjuk fel a Fan-egyenlőtlenséget.

8.2-9. Legyenek $N = 2$, $S_1 = S_2 = [0, 10]$, $f_1(x_1, x_2) = f_2(x_1, x_2) = 2x_1 + 2x_2 - (x_1 + x_2)^2$. Írjuk fel a Kuhn–Tucker-feltételeket, és keressük meg az egyensúlyokat. Oldjuk meg az így kapott feltételrendszert.

8.2-10. Tekintsünk egy háromszemélyes játékot, ahol $S_1 = S_2 = S_3 = [0, 1]$, $f_1(x_1, x_2, x_3) = (x_1 - x_2)^2 + x_3$, $f_2(x_1, x_2, x_3) = (x_2 - x_3)^2 + x_1$ és $f_3(x_1, x_2, x_3) = (x_3 - x_1)^2 + x_2$. Írjuk fel a Kuhn–Tucker-feltételeket.

8.2-11. Írjuk fel és oldjuk meg (8.9)-et a 8.2-1. és a 8.2-8. gyakorlatokban bevezetett játékokra.

8.2-12. Írjuk át 8.2-8. gyakorlat Kuhn–Tucker-feltételeit (8.10) formába, és oldjuk meg őket.

8.2-13. Írjuk fel a 8.1-1. gyakorlatban bevezetett véges játék kevert bővítését.

8.2-14. Írjuk fel és oldjuk meg (8.10)-et a 8.2-13. gyakorlatban bevezetett játékra.

8.2-15. Írjuk fel a 8.1-3. gyakorlatban bevezetett játék kevert bővítését. Írjuk fel és oldjuk meg az erre a játékra vonatkozó (8.22)-őt, ha $\alpha = 5$ és $\beta = 3$.

8.2-16. Oldjuk meg a 8.2-15. gyakorlatban bevezetett mátrixjáték egyensúlyi problémáját a fiktív lejátszás módszerével.

8.2-17. Vegyük az $\mathbf{A} = \begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix}$ és $\mathbf{B} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}$ mátrixokkal adott bimátrix-játékot. Oldjuk meg ezt a bimátrix-játékot a 8-1. gyakorlatban meghatározott módszerrel.

8.2-18. Oldjuk meg az $\mathbf{A} = \begin{pmatrix} 0 & 1 & 5 \\ -1 & 0 & -3 \\ -5 & 3 & 0 \end{pmatrix}$ szimmetrikus mátrixjáték egyensúlyi problémáját lineáris programozással.

8.2-19. Oldjuk meg 8.2-18. gyakorlatot a fiktív lejátszás módszerével.

8.2-20. Írjuk fel a (8.9) Kuhn–Tucker-feltételeket a 8.2-18. gyakorlatra.

8.2-21.$\star$ Tekintsük az $\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ -1 & 0 & 1 \end{pmatrix}$ mátrixjátékot. Oldjuk meg az $\mathbf{A}$ mátrixjáték egyensúlyi problémáját lineáris programozással, a fiktív lejátszás módszerével, és írjuk fel a Kuhn–Tucker-feltételeket. *Útmutatás.* Először határozzuk meg az ekvivalens szimmetrikus mátrixjátékot.

8.2-22. Írjuk fel lineáris programozási feladatként az $\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$ mátrixjáték egyensúlyi

problémáját.

8.2-23. Írjuk fel egy lineáris programozási feladat megoldását a fiktív lejátszás módszerével, és oldjuk meg az így kapott módszerrel a következő lineáris programozási feladatot:

$$\begin{aligned} x_1 + x_2 &\rightarrow \max \\ x_1, x_2 &\geq 0 \\ 3x_1 + x_2 &\leq 4 \\ x_1 + 3x_2 &\leq 4\,. \end{aligned}$$

8.2-24. Oldjuk meg a 8.2-21. gyakorlat lineáris programozási feladatát a fiktív lejátszás módszerével.

8.2-25. Oldjuk meg a 8.2-18. gyakorlatot a Neumann-módszerrel.

8.2-26. Oldjuk meg a 8.2-21. gyakorlatot a Neumann-módszerrel.

8.2-27. Oldjuk meg a 8.2-16. gyakorlatot a Neumann-módszerrel.

8.2-28.⋆ Ellenőrizzük a 8.2-25., 8.2-26. és 8.2-27. gyakorlatok eredményeit úgy, hogy a (8.21) lineáris programozási feladat feltételrendszerének érvényességét megvizsgáljuk nulla célfüggvényérték mellett. *Útmutatás.* Minek válasszuk α-t és β-t?

8.2-29. Ismételjük meg a 8.2-23. gyakorlatot a Neumann-módszerrel.

8.2-30. Legyenek $N = 2$, $S_1 = S_2 = [0, 10]$, $f_1(x_1, x_2) = f_2(x_1, x_2) = 2x_1 + 2x_2 - (x_1 + x_2)^2$. Mutassuk meg, hogy mindkét kifizetőfüggvény szigorú konkáv (x_1 szerint f_1, és x_2 szerint f_2). Bizonyítsuk be, hogy ennek a játéknak végtelen sok egyensúlya van, tehát a kifizetőfüggvények szigorú konkavitásából nem következik az egyensúly egyértelműsége.

8.2-31. Lehet-e egy mátrixjáték átlósan szigorúan konkáv?

8.2-32. Tekintsük azt a kétszemélyes játékot, ahol a stratégiahalmazok $S_1 = S_2 = [0, 1]$, a kifizetőfüggvények $f_1(x_1, x_2) = -2x_1^2 + x_1(1 - x_2)$, $f_2(x_1, x_2) = -3x_2^2 + x_2(1 - x_1)$. Mutassuk meg, hogy ez a játék kielégíti a 8.16. tétel feltételeit.

8.2-33. Oldjuk meg a 8.2-32. gyakorlatot a (8.47)–(8.48)-ban adott algoritmussal.

8.3. Az oligopol feladat

Az eddigiekben általános módszereket mutattunk be az egyensúlyi probléma megoldására. Majdnem minden speciális játékosztályra vannak azonban olyan speciális módszerek, melyek az adott játékosztály tagjaira alkalmazhatók. Ezen fejezet további részében egy speciális játékot, az ***oligopol játékot*** vesszük górcső alá. Az oligopol játék egy olyan valós helyzetet ír le, amikor N vállalat azonos terméket gyárt vagy azonos szolgáltatást kínál. Ez a modell a ***klasszikus Cournot-modellként*** ismert. A játékosok stratégiái az x_k gyártási mennyiségek, melyek az $S_k = [0, L_k]$ stratégiahalmazokból valók, ahol L_k az adott játékos (vállalat) gyártási kapacitásának felső határa. Feltesszük, hogy a $p(s)$ piaci ár az összesen gyártott $s = x_1 + x_2 + \cdots + x_N$ mennyiségtől függ, és minden játékos $c_k(x_k)$ gyártási költsége csak a saját gyártási szintjétől függ. A vállalatok profitfüggvényei:

$$f_k(x_1, \ldots, x_N) = x_k p\left(\sum_{l=1}^{N} x_l\right) - c_k(x_k)\,. \tag{8.50}$$

Tehát definiáltuk a $G = \{N; S_1, \ldots, S_N; f_1, \ldots, f_N\}$ játékot.

Fel szokás tenni, hogy a p és c_k $(k = 1, 2, \ldots, N)$ függvények kétszer folytonosan differenciálhatók, továbbá

1. $p'(s) < 0$;
2. $p'(s) + x_k p''(s) \leq 0$;
3. $p'(s) - c_k''(x_k) < 0$;

minden k-ra, $x_k \in [0, L_k]$-ra, és $s \in [0, \sum_{l=1}^{N} L_l]$-re.

Az 1–3. feltételek mellett 8.3. tétel feltételei teljesülnek, tehát G-nek van legalább egy egyensúlya.

Legjobbválasz-leképezések

Vegyük észre, hogy az $s_k = \sum_{l \neq k} x_l$ jelölés mellett a k játékos kifizetőfüggvénye átírható a következő formába:

$$x_k p(x_k + s_k) - c_k(x_k) \ . \tag{8.51}$$

Mivel S_k kompakt halmaz, és a kifizetőfüggvény szigorúan konkáv x_k szerint, így rögzített s_k mellett a $\mathcal{P}_k$ játékos profitmaximalizáló gyártási szintje egyértelmű, mely a k játékos legjobbválasza – jelöljük ezt $B_k(s_k)$-val.

Könnyen látható, hogy három eset lehetséges: $B_k(s_k) = 0$, ha $p(s_k) - c_k'(0) \leq 0$, $B_k(s_k) = L_k$, ha $p(s_k + L_k) + L_k p'(s_k + L_k) - c_k'(L_k) \geq 0$, egyébként $B_k(s_k)$ az egyetlen megoldása a következő monoton egyenletnek:

$$p(s_k + x_k) + x_k p'(s_k + x_k) - c_k'(x_k) = 0 \ .$$

Tegyük fel, hogy $x_k \in (0, L_k)$. Ekkor s_k szerinti implicit deriválással

$$p'(1 + B_k') + B_k' p' + x_k p''(1 + B_k') - c_k'' B_k' = 0 \ ,$$

ahonnan

$$B_k'(s_k) = -\frac{p' + x_k p''}{2p' + x_k p'' - c_k''} \ .$$

Vegyük észre, hogy a 2–3. feltevések miatt

$$-1 < B_k'(s_k) \leq 0 \ , \tag{8.52}$$

mely egyenlőtlenség a töréspontok kivételével a másik két esetben is igaz.

A 8.2.1. pontnak megfelelően vezessük be a legjobbválasz-leképezést:

$$\mathbf{B}(x_1, \ldots, x_N) = \left(B_1\left(\sum_{l \neq 1} x_l \right), \ldots, B_N\left(\sum_{l \neq N} x_l \right) \right) \ . \tag{8.53}$$

A feladat a fenti leképezés fixpontjának megkeresése. Másik lehetőség, hogy bevezetünk egy dinamikus folyamatot, amely konvergál az egyensúlyhoz.

A fiktív lejátszás diszkrét módszeréhez hasonló módszert dolgozunk ki, melyben minden vállalat kiválasztja a legjobb válaszát a versenytársai előző periódusban tett lépéseire:

$$x_k(t+1) = B_k\Big(\sum_{l \neq k} x_l(t)\Big) \qquad (k = 1, 2, \ldots, N) \ . \tag{8.54}$$

(8.52) miatt látható, hogy ($N = 2$)-re a jobb oldalon lévő leképezés $\mathbb{R}^2 \to \mathbb{R}^2$ kontrakció, tehát konvergens. Azonban, ha $N > 2$, akkor a konvergenciát nem lehet garantálni. Tekintsük most a fenti rendszer nyilvánvaló módosítását valamilyen $K_k > 0$-val:

$$x_k(t+1) = x_k(t) + K_k(B_k(\sum_{l \neq k} x_l(t)) - x_k(t)) \tag{8.55}$$

minden $k = 1, 2, \ldots, N$-re. Világos, hogy a fenti rendszer minden stabil állapota egyensúly, és bizonyítható, hogy ha K_k megfelelően kicsi, akkor az $x_k(0), x_k(1), x_k(2), \ldots$ sorozatok konvergensek minden $k = 1, 2, \ldots, N$-ra, és az egyensúlyhoz konvergálnak.

Tekintsük most a (8.55) modell folytonos alkalmazását, ahol (a Neumann-módszerhez hasonlóan) folytonos időskálát tételezünk fel:

$$\dot{x}_k(t) = K_k(B_k(\sum_{l \neq k} x_l(t)) - x_k(t)) \quad (k = 1, 2, \ldots, N)\ . \tag{8.56}$$

A következő eredmény a fenti rendszer konvergenciájáról szól.

8.20. tétel. *Az 1–3. feltevések mellett a (8.56) rendszer aszimptotikusan stabil, azaz ha az $x_k(0)$ kezdetiértéket az egyensúlyhoz elég közelre választjuk, ekkor $x_k(t)$ tart az egyensúlyhoz minden k-ra.*

Bizonyítás. Elég azt megmutatni, hogy a (8.56) rendszer Jakobi-mátrixának sajátértékei negatív valós számok. Látható, hogy a Jakobi-mátrix a következő:

$$\mathbf{J} = \begin{pmatrix} -K_1 & K_1 b_1 & \cdots & K_1 b_1 \\ K_2 b_2 & -K_2 & \cdots & K_2 b_2 \\ \vdots & \vdots & & \vdots \\ K_N b_N & K_N b_N & \cdots & -K_N \end{pmatrix}, \tag{8.57}$$

ahol $b_k = B_k'(\sum_{l \neq k} x_l)$ az egyensúlyban. (8.52)-ből tudjuk, hogy $-1 < b_k \leq 0$ minden k-ra. A $\mathbf{J}$ sajátértékeinek kiszámítása céljából szükségünk van egy nagyon egyszerű, ám annál hasznosabb tényre. Tegyük fel, hogy az $\mathbf{a}$ és a $\mathbf{b}$ N valós komponensű vektorok. Ekkor

$$\det(\mathbf{I} + \mathbf{a}\mathbf{b}^T) = 1 + \mathbf{b}^T\mathbf{a}\ , \tag{8.58}$$

ahol $\mathbf{I}$ az N-edrendű egységmátrix. (8.58) egyenlőtlenség teljes indukcióval könnyen bizonyítható. (8.58)-at felhasználva, a $\mathbf{J}$ mátrix karakterisztikus polinomja:

$$\begin{aligned} \phi(\lambda) &= \det(\mathbf{J} - \lambda\mathbf{I}) = \det(\mathbf{D} + \mathbf{a}\mathbf{b}^T - \lambda\mathbf{I}) \\ &= \det(\mathbf{D} - \lambda\mathbf{I})\det(\mathbf{I} + (\mathbf{D} - \lambda\mathbf{I})^{-1}\mathbf{a}\mathbf{b}^T) \\ &= \det(\mathbf{D} - \lambda\mathbf{I})[1 + \mathbf{b}^T(\mathbf{D} - \lambda\mathbf{I})^{-1}\mathbf{a}] \\ &= \Pi_{k=1}^N(-K_k(1+b_k) - \lambda)[1 + \sum_{k=1}^N \frac{K_k b_k}{-K_k(1+b_k) - \lambda}]\ , \end{aligned}$$

ahol a következő jelöléseket használtuk:

$$\mathbf{a} = \begin{pmatrix} K_1 b_1 \\ K_2 b_2 \\ \vdots \\ K_N b_N \end{pmatrix},\quad \mathbf{b}^T = (1, 1, \ldots, 1),\quad \mathbf{D} = \begin{pmatrix} -K_1(1+b_1) & & \\ & \ddots & \\ & & K_N(1+b_N) \end{pmatrix}.$$

Az első tényező gyökei negatívak: $\lambda = -K_k(1 + b_k)$, a következő egyenlet gyökei adják a többi sajátértéket:

$$1 + \sum_{k=1}^{N} \frac{K_k b_k}{-K_k(1 + b_k) - \lambda} = 0 .$$

Vegyük észre, hogy közös nevezőre hozva és a tagokat összeadva az

$$1 + \sum_{l=1}^{m} \frac{\alpha_k}{\beta_k + \lambda} = 0 \tag{8.59}$$

egyenlőség adódik, ahol $\alpha_k, \beta_k > 0$, és $\beta_1 < \beta_2 < \cdots < \beta_m$. Ha $g(\lambda)$ jelöli a bal oldalt, akkor a $\lambda = -\beta_k$-k a pólushelyek és

$$\lim_{\lambda \to \pm\infty} g(\lambda) = 1, \quad \lim_{\lambda \to -\beta_k \pm 0} g(\lambda) = \pm\infty ,$$

$$g'(\lambda) = \sum_{l=1}^{m} \frac{-\alpha_l}{(\beta_l + \lambda)^2} < 0 ,$$

így $g(\lambda)$ az értelmezési tartományának tetszőleges intervallumán szigorúan monoton fogyó. A függvény grafikonja a 8.7. ábrán látható. Vegyük észre, hogy (8.59) ekvivalens egy m-edfokú polinom egyenletével, tehát m komplex (esetleg) valós gyöke van. A $g(\lambda)$ függvény tulajdonságai miatt egy gyök kisebb mint $-\beta_1$, és egy-egy gyök van $-\beta_k$ és $-\beta_{k+1}$ között minden $k = 1, 2, \ldots, m-1$ esetén. Tehát minden gyök negatív valós szám. ■

(8.55) – az általános diszkrét modell – azonos módon vizsgálható. Ha $K_k = 1$ minden k-ra, akkor (8.55) visszavezethető a (8.54) egyszerű dinamikus rendszerre.

8.21. példa. *Első oligopol játék.* Tekintsünk egy háromszemélyes oligopol játékot, ahol az árfüggvény

$$p(s) = \begin{cases} 2 - 2s - s^2, & \text{ha} \quad 0 \le s \le \sqrt{3} - 1 , \\ 0 & \text{egyébként} , \end{cases}$$

a stratégiahalmazok $S_1 = S_2 = S_3 = [0, 1]$, a költségfüggvények

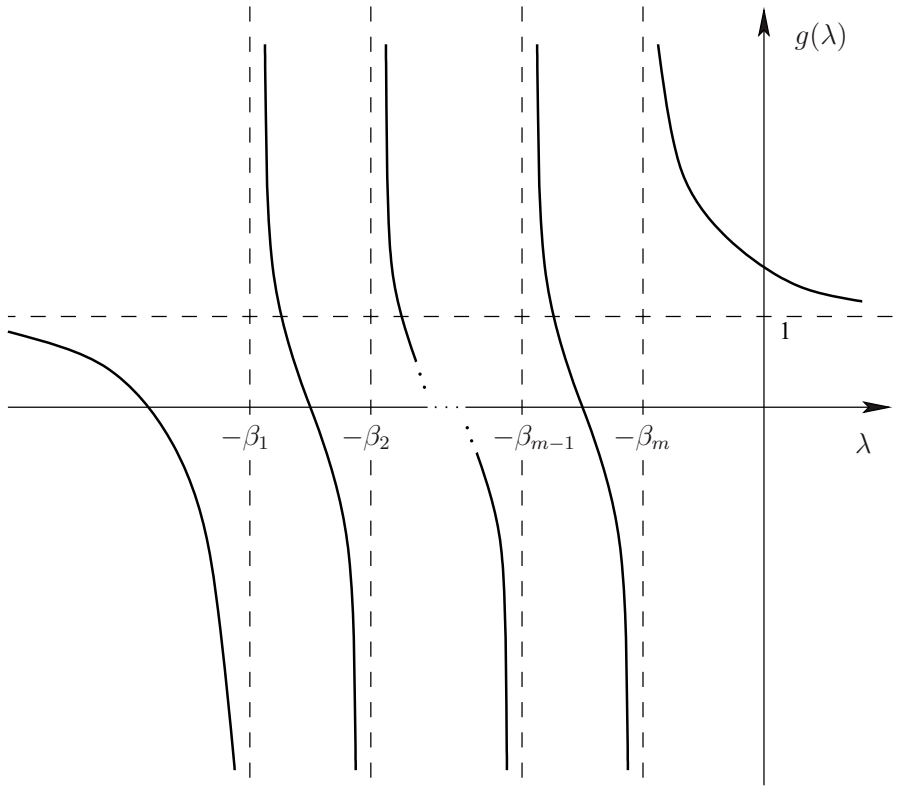
$$c_k(x_k) = k x_k^3 + x_k \qquad (k = 1, 2, 3) .$$

A $\mathcal{P}_k$ vállalat profitja a következő:

$$x_k(2 - 2s - s^2) - (k x_k^3 + x_k) = x_k(2 - 2x_k - 2s_k - x_k^2 - 2x_k s_k - s_k^2) - k x_k^3 - x_k .$$

A $\mathcal{P}_k$ vállalat legjobbválasz-függvényét a következőképpen kapjuk. A 8.3. alfejezet elején vázolt módszert követve, a következő három esetet különböztetjük meg. Ha $1 - 2s_k - s_k^2 \le 0$, akkor $x_k = 0$ a legjobbválasz. Ha $(-6 - 3k) - 6s_k - s_k^2 \ge 0$, akkor $x_k = 1$ a legjobbválasz. Egyébként a legjobbválaszt a következő egyenlet adja meg:

$$\frac{\partial}{\partial x_k}[x_k(2 - 2x_k - 2s_k - s_k^2 - 2s_k x_k - x_k^2) - k x_k^3 - x_k]$$
$$= 2 - 4x_k - 2s_k - s_k^2 - 4s_k x_k - 3x_k^2 - 3k x_k^2 - 1 = 0 ,$$



8.7. ábra. A $g(\lambda)$ függvény görbéje.

ahol az egyetlen pozitív megoldás

$$x_k = \frac{-(4+4s_k) + \sqrt{(4+4s_k)^2 - 12(1+k)(s_k^2+2s_k-1)}}{6(1+k)} .$$

Miután a legjobbválaszokat megtaláltuk, könnyedén megkonstruálhatjuk bármelyik korábban bemutatott módszert.

Visszavezetés egydimenziós fixpont feladatra

Tekintsünk egy N vállalatos (N személyes) oligopol játékot, ahol p az árfüggvény, és a c_k függvények a költségfüggvények $k = 1, 2, \ldots, N$ esetén. Vezessük be a

$$\Psi_k(s, x_k, t_k) = t_k p(s - x_k + t_k) - c_k(t_k) , \tag{8.60}$$

függvényt és definiáljuk az

$$X_k(s) = \{x_k | x_k \in S_k, \;\; \Psi_k(s, x_k, x_k) = \max_{t_k \in S_k} \Psi_k(s, x_k, t_k)\} \tag{8.61}$$

leképezést minden $k = 1, 2, \ldots, N$-re. Legyen továbbá

$$X(s) = \{u | u = \sum_{k=1}^{N} x_k, \quad x_k \in X_k(s), \quad k = 1, 2, \ldots, N\} \; . \tag{8.62}$$

Vegyük észre, hogy ha $s \in [0, \sum_{k=1}^{N} L_k]$, akkor $X(s)$ minden komponense ebbe az intervallumba esik, tehát X egy egydimenziós pont-halmaz leképezés. Világos, hogy $(x_1^\star, \ldots, x_N^\star)$ pontosan akkor egyensúlya az N vállalatos oligopol játéknak, ha $s^\star = \sum_{k=1}^{N} x_k^\star$ fixpontja X-nek, és minden k-ra $x_k^\star \in X_k(s^\star)$. Tehát az egyensúlyi problémát leegyszerűsítettük egy egydimenziós pont-halmaz leképezés fixpont problémájára. Ez jelentős egyszerűsítés, hiszen a legjobbválaszok N dimenziós leképezések.

Ha az 1–3. feltételek teljesülnek, akkor $X_k(s)$ értéke egy egyelemű halmaz minden s-re, k-ra:

$$X_k(s) = \begin{cases} 0, & \text{ha } p(s) - c_k'(0) \le 0 \; , \\ L_k, & \text{ha } p(s) + L_k p_k'(s) - c_k'(L_k) \ge 0 \; , \\ z^\star & \text{egyébként} \; , \end{cases} \tag{8.63}$$

ahol $z^\star$ a következő monoton egyenlet egyetlen megoldása a $[0, L_k]$ intervallumon:

$$p(s) + z p'(s) - c_k'(z) = 0 \; . \tag{8.64}$$

A harmadik esetben a bal oldali kifejezés pozitív $z = 0$-ban, és negatív $z = L_k$-ban, a 2–3. feltételek miatt szigorúan fogyó, tehát egyetlen megoldás van.

Az egész $[0, \sum_{k=1}^{N} L_k]$ intervallumon $X_k(s)$ nemnövekvő konstans az első két esetben, és szigorúan fogyó a harmadik esetben. Tekintsük végül az egydimenziós egyenletet:

$$\sum_{k=1}^{N} X_k(s) - s = 0 \; . \tag{8.65}$$

$s = 0$-ban a bal oldal nemnegatív, $s = \sum_{k=1}^{N} L_k$-ban nempozitív, szigorúan fogyó. Tehát egyetlen megoldás van (az X fixpontja), mely megoldás bármely egydimenziós egyenletmegoldó módszerrel megkapható.

Legyen $[0, S_{max}]$ a (8.65) megoldásának értelmezési tartománya. K felező lépés után a pontosság $s_{max}/2^K$, mely kisebb, mint az $\epsilon > 0$ hibahatár, ha $K > \log_2(S_{max}/\epsilon)$.

8.22. példa. *Második oligopol játék.* Tekintsük megint a 8.21. példában látott háromszemélyes oligopol játékot. (8.63)-ból

$$X(s) = \begin{cases} 0, & \text{ha} \quad 1 - 2s - s^2 \le 0 \; , \\ 1, & \text{ha} \quad -(1 + 3k) - 4s - s^2 \ge 0 \; , \\ z^\star & \text{egyébként} \; , \end{cases}$$

ahol $z^\star$ a

$$3kz^2 + z(2s + 2) + (-1 + 2s + s^2) = 0$$

egyenlet egyetlen megoldása. Az első eset akkor következik be, ha $s \ge \sqrt{2} - 1$, a második eset soha nem következik be, míg a harmadik eset az egyetlen pozitív megoldás:

$$z^\star = \frac{-(2s + 2) + \sqrt{(2s + 2)^2 - 12k(-1 + 2s + s^2)}}{6k} \; . \tag{8.66}$$

Végül (8.65) speciális formája

$$\sum_{k=1}^{3} \frac{-(s+1)+\sqrt{(s+1)^2-3k(-1+2s+s^2)}}{3k} - s = 0\ .$$

Az intervallumfelezéses módszeren alapuló program a következő megoldást adja: $s^\star \approx 0.2982$. (8.66)-ból az egyensúlyi stratégiák: $x_1^\star \approx 0.1077$, $x_2^\star \approx 0.0986$, $x_3^\star \approx 0.0919$.

A Kuhn–Tucker-feltételeken alapuló módszerek

Vegyük észre, hogy az N személyes oligopol játékok esetében $S_k = \{x_k | x_k \geq 0,\ L_k - x_k \geq 0\}$, tehát választhatjuk a

$$\mathbf{g}_k(x_k) = \begin{pmatrix} x_k \\ L_k - x_k \end{pmatrix} \tag{8.67}$$

függvényeket. Mivel a kifizetőfüggvények

$$f_k(x_1, \ldots, x_N) = x_k p(x_k + s_k) - c_k(x_k)\ , \tag{8.68}$$

a (8.9) Kuhn–Tucker-feltételek a következő formát öltik, ahol a két komponensű vektor $\mathbf{u}_k$ komponensei $u_k^{(1)}$ és $u_k^{(2)}$, és minden $k = 1, 2, \ldots, N$ indexre

$$\begin{aligned} u_k^{(1)}, u_k^{(2)} &\geq 0 \\ x_k &\geq 0 \\ L_k - x_k &\geq 0 \\ p(\textstyle\sum_{l=1}^N x_l) + x_k p'(\sum_{l=1}^N x_l) - c_k'(x_k) + (u_k^{(1)}, u_k^{(2)})\begin{pmatrix} 1 \\ -1 \end{pmatrix} &= 0 \\ u_k^{(1)} x_k + u_k^{(2)}(L_k - x_k) &= 0\ . \end{aligned} \tag{8.69}$$

Két lehetőségünk van: vagy megkeressük (8.69) egy megengedett megoldását, vagy átírjuk (8.69)-et (8.10) alakú optimumszámítási feladattá, mely ebben a speciális esetben

$$\begin{aligned} \textstyle\sum_{k=1}^N (u_k^{(1)} x_k + u_k^{(2)}(L_k - x_k)) &\to \min \\ u_k^{(1)}, u_k^{(2)} &\geq 0\ (k = 1, 2, \ldots, N) \\ x_k &\geq 0\ (k = 1, 2, \ldots, N) \\ L_k - x_k &\geq 0\ (k = 1, 2, \ldots, N) \\ p(\textstyle\sum_{l=1}^N x_l) + x_k p'(\sum_{l=1}^N x_l) - c_k'(x_k) + u_k^{(1)} - u_k^{(2)} &= 0\ (k = 1, 2, \ldots, N)\ . \end{aligned} \tag{8.70}$$

A (8.69) vagy (8.70) számítási költsége a p és a c_k függvények típusától függ. Nem adható általános jellemzés.

8.23. példa. *Harmadik oligopol játék.* A 8.21. példában bevezetett játék esetén (8.70) alakja

$$\begin{aligned} \sum_{k=1}^{3} (u_k^{(1)} x_k + u_k^{(2)}(1 - x_k)) &\to \min \\ u_k^{(1)}, u_k^{(2)} &\geq 0 \\ x_k &\geq 0 \\ 1 - x_k &\geq 0 \\ 1 - 2s - s^3 - 2x_k - 2x_k s - 3kx_k^2 + u_k^{(1)} - u_k^{(2)} &= 0 \\ x_1 + x_2 + x_3 &= s\ . \end{aligned}$$

Egy professzionális optimalizációs program használatával a következő megoldást kaptuk:

$$x_1^\star \approx 0.1077, \quad x_2^\star \approx 0.0986, \quad x_3^\star \approx 0.0919 ,$$

és $u_k^{(1)} = u_k^{(2)} = 0$.

Visszavezetés komplementer feladatokra

Ha $(x_1^\star, \ldots, x_N^\star)$ egyensúly egy N személyes oligopol játékban, akkor rögzített $x_1^\star, \ldots, x_{k-1}^\star, x_{k+1}^\star, \ldots, x_N^\star$ esetén $x_k = x_k^\star$ maximumhelye a $\mathcal{P}_k$ játékos f_k kifizetőfüggvényének. Tegyük fel, hogy az 1–3. feltételek teljesülnek, f_k konkáv x_k-ban, így $x_k^\star$ pontosan akkor maximumhelye f_k-nak, ha az egyensúlyban

$$\frac{\partial f_k}{\partial x_k}(\mathbf{x}^\star) = \begin{cases} \leq 0, & \text{ha} \quad x_k^\star = 0 , \\ = 0, & \text{ha} \quad 0 < x_k^\star < L_k , \\ \geq 0, & \text{ha} \quad x_k^\star = L_k . \end{cases}$$

Vezessük be a

$$z_k = \begin{cases} = 0, & \text{ha} \quad x_k > 0 , \\ \geq 0, & \text{ha} \quad x_k = 0 \end{cases}$$

és a

$$v_k = \begin{cases} = 0, & \text{ha} \quad x_k < L_k , \\ \geq 0, & \text{ha} \quad x_k = L_k \end{cases}$$

túlcsordulás változókat és legyen

$$w_k = L_k - x_k . \tag{8.71}$$

(8.71) az egyensúlyban átírható, mint

$$\frac{\partial f_k}{\partial x_k}(\mathbf{x}) - v_k + z_k = 0 . \tag{8.72}$$

A túlcsordulás változók definíciója miatt

$$z_k x_k = 0 , \tag{8.73}$$

$$v_k w_k = 0 . \tag{8.74}$$

A nemnegativitási feltételt hozzávéve

$$x_k, z_k, v_k, w_k \geq 0 , \tag{8.75}$$

mely esetben a (8.71)–(8.75) nemlineáris egyenlőtlenségrendszert kapjuk, mely ekvivalens az egyensúlyi feladattal.

A következőkben megmutatjuk, hogy (8.71)–(8.75) átírható nemlineáris komplementer feladattá. Az ilyen feladatok megoldására vannak közismert módszerek. Vezessük be a következő jelöléseket:

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{pmatrix}, \quad \mathbf{L} = \begin{pmatrix} L_1 \\ L_2 \\ \vdots \\ L_N \end{pmatrix}, \quad \mathbf{h}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_2}(\mathbf{x}) \\ \vdots \\ \frac{\partial f_N}{\partial x_N}(\mathbf{x}) \end{pmatrix},$$

$$\mathbf{t} = \begin{pmatrix} \mathbf{x} \\ \mathbf{v} \end{pmatrix}, \quad \text{és} \quad \mathbf{g}(\mathbf{t}) = \begin{pmatrix} -\mathbf{h}(\mathbf{x}) + \mathbf{v} \\ \mathbf{L} - \mathbf{x} \end{pmatrix}.$$

Ekkor a (8.72)–(8.75) rendszer átírható, mint

$$\begin{aligned} \mathbf{t} &\geq \mathbf{0} \\ \mathbf{g}(\mathbf{t}) &\geq \mathbf{0} \\ \mathbf{t}^T \mathbf{g}(\mathbf{t}) &= 0\,. \end{aligned} \tag{8.76}$$

A fenti feladat a ***nemlineáris komplementer feladatok*** szokásos formája. Vegyük észre, hogy az utolsó feltétel azt követeli meg, hogy komponensenként vagy $\mathbf{t}$, vagy $\mathbf{g}(\mathbf{t})$, vagy mindkettő nulla legyen.

(8.76) számítási költsége a benne lévő függvények típusától, és a választott módszertől függ.

8.24. példa. *Negyedik oligopol játék.* A 8.21. példában bevezetett háromszemélyes oligopol játék elemzéséből kapjuk:

$$\mathbf{t} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} \quad \text{és} \quad \mathbf{g}(\mathbf{t}) = \begin{pmatrix} -1 + 2\sum_{l=1}^3 x_l + (\sum_{l=1}^3 x_l)^2 + 2x_1 + 2x_1 \sum_{l=1}^3 x_l + 3x_1^2 + v_1 \\ -1 + 2\sum_{l=1}^3 x_l + (\sum_{l=1}^3 x_l)^2 + 2x_2 + 2x_2 \sum_{l=1}^3 x_l + 6x_2^2 + v_2 \\ -1 + 2\sum_{l=1}^3 x_l + (\sum_{l=1}^3 x_l)^2 + 2x_3 + 2x_3 \sum_{l=1}^3 x_3 + 9x_3^2 + v_3 \\ 1 - x_1 \\ 1 - x_2 \\ 1 - x_3 \end{pmatrix}.$$

Lineáris oligopol játékok és kvadratikus programozás

Ebben a részben olyan N személyes oligopol játékokat fogunk vizsgálni, ahol mind az ár-függvény, mind a költségfüggvények lineárisak:

$$p(s) = As + B, \qquad c_k(x_k) = b_k x_k + c_k \qquad (k = 1, 2, \ldots, N)\,,$$

ahol $B, b_k, c_k > 0$, de $A < 0$. Tegyük fel megint, hogy a stratégiahalmazok intervallumok: $[0, L_k]$. Ebben az esetben

$$f_k(x_1, \ldots, x_N) = x_k(Ax_1 + \cdots + Ax_N + B) - (b_k x_k + c_k) \tag{8.77}$$

minden k-ra, így

$$\frac{\partial f_k}{\partial x_k}(\mathbf{x}) = 2Ax_k + A\sum_{l \neq k} x_l + B - b_k\,. \tag{8.78}$$

A (8.71)–(8.75) feltételek ebben az esetben (nem az eredeti sorrendben):

$$\begin{aligned} 2Ax_k + A\sum_{l \neq k} x_l + B - b_k - v_k + z_k &= 0 \\ z_k x_k = v_k w_k &= 0 \\ x_k + w_k &= L_k \\ x_k, v_k, z_k, w_k &\geq 0\,. \end{aligned}$$

Vezessük be a következő mátrixot és vektorokat:

$$\mathbf{Q} = \begin{pmatrix} 2A & A & \cdots & A \\ A & 2A & \cdots & A \\ \vdots & \vdots & & \vdots \\ A & A & \cdots & 2A \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} B \\ B \\ \vdots \\ B \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix},$$

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix}, \quad \mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{pmatrix}, \quad \text{és} \quad \mathbf{L} = \begin{pmatrix} L_1 \\ L_2 \\ \vdots \\ L_N \end{pmatrix}.$$

Foglaljuk össze a definiált egyenlőtlenségeket:

$$\begin{aligned} \mathbf{Qx} + \mathbf{B} - \mathbf{b} - \mathbf{v} + \mathbf{z} &= \mathbf{0} \\ \mathbf{x} + \mathbf{w} &= \mathbf{L} \\ \mathbf{x}^T\mathbf{z} = \mathbf{v}^T\mathbf{w} &= 0 \\ \mathbf{x}, \mathbf{v}, \mathbf{z}, \mathbf{w} &\geq \mathbf{0}\,. \end{aligned} \tag{8.79}$$

A következőkben azt látjuk be, hogy $\mathbf{Q}$ negatív definit. Tetszőleges $\mathbf{a} = (a_i)$ nemnulla vektorra

$$\mathbf{a}^T\mathbf{Qa} = 2A\sum_i a_i^2 + A\sum_i\sum_{j\neq i} a_i a_j = A\Big(\sum_i a_i^2 + \big(\sum_i a_i\big)^2\Big) < 0\,,$$

ahonnan következik a feltevésünk.

Vegyük észre, hogy (8.79)-ben a következő, szigorúan konkáv kvadratikus feladat Kuhn–Tucker-feltételei vannak:

$$\begin{aligned} \tfrac{1}{2}\mathbf{x}^T\mathbf{Qx} + (\mathbf{B} - \mathbf{b})\mathbf{x} &\to \max \\ \mathbf{0} \leq \mathbf{x} &\leq \mathbf{L}\,. \end{aligned} \tag{8.80}$$

Mivel a megengedett megoldások halmaza korlátos poliéder, és a célfüggvény szigorúan konkáv, így a Kuhn–Tucker-feltételek szükségesek és elégségesek is egyben. Következésképpen, az $\mathbf{x}^\star$ vektor pontosan akkor egyensúly, ha (8.80) egyetlen optimális megoldása. (8.80) megoldására közismert módszerek találhatók az irodalomban.

Mivel (8.79) egy konvex kvadratikus feladat, így annak megoldására ismeretesek módszerek. A módszerek költsége különböző, tehát (8.79) számítási költsége a választott módszertől függ.

8.25. példa. *Kétszemélyes oligopol játék.* Tekintsünk egy duopol játékot (kétszemélyes oligopol játékot), ahol az árfüggvény $p(s) = 10 - s$, a költségfüggvények $c_1(x_1) = 4x_1 + 1$ és $c_2(x_2) = x_2 + 1$, a kapacitáskorlátok $L_1 = L_2 = 5$. Azaz

$$B = 10,\ \ A = -1,\ \ b_1 = 4,\ \ b_2 = 1,\ \ c_1 = c_2 = 1\,.$$

Tehát,

$$\mathbf{Q} = \begin{pmatrix} -2 & -1 \\ -1 & -2 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 10 \\ 10 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 4 \\ 1 \end{pmatrix}, \quad \mathbf{L} = \begin{pmatrix} 5 \\ 5 \end{pmatrix}.$$

A kvadratikus programozási feladat:

$$\begin{aligned} \frac{1}{2}(-2x_1^2 - 2x_1x_2 - 2x_2^2) + 6x_1 + 9x_2 &\to \max \\ 0 \leq x_1 &\leq 5 \\ 0 \leq x_2 &\leq 5\,. \end{aligned}$$

Egyszerű deriválással látható, hogy a célfüggvény feltételek nélküli globális maximumát az $(x_1^\star, x_2^\star)^T = (1,4)^T$ pontban veszi fel. Mivel ez a pont benne van a megengedett megoldások halmazában, így optimális megoldása a feladatnak, tehát a duopol játék egyetlen egyensúlya.

Gyakorlatok

8.3-1. Tekintsünk egy duopol játékot, ahol $S_1 = S_2 = [0,1]$, $p(s) = 2 - s$ és $c_1(x) = c_2(x) = x^2 + 1$. Vizsgáljuk meg az (8.55)-ben megadott iterációs eljárás konvergenciáját.

8.3-2. Legyen $N = 2$, $S_1 = S_2 = [0, 1.5]$, $c_k(x_k) = 1.5x_k$ $(k = 1, 2)$ és

$$p(s) = \begin{cases} 1.75 - 0.5s, & \text{ha} \quad 0 \le s \le 1.5\ , \\ 2.5 - s, & \text{ha} \quad 1.5 \le s \le 2.5\ , \\ 0, & \text{ha} \quad 2.5 < s\ . \end{cases}$$

Mutassuk meg, hogy végtelen sok egyensúly van:

$$\{(x_1^\star, x_2^\star) | 0.5 \le x_1 \le 1,\ \ 0.5 \le x_2 \le 1,\ \ x_1 + x_2 = 1.5\}\ .$$

8.3-3. Tekintsük a 8.3-1. gyakorlatban bevezetett duopol játékot.
(a) Írjuk fel a legjobbválasz-függvényeket, és határozzuk meg az egyensúlyt.
(b) Tekintsük a (8.62)-ben bevezetett egydimenziós fixpont feladatot, és határozzuk meg segítségével az egyensúlyt.
(c) Írjuk fel a (8.69) Kuhn–Tucker-feltételeket.
(d) Írjuk fel a (8.76) komplementer feladatot.

Feladatok

8-1. Fiktív lejátszás bimátrix-játékokra
Általánosítsuk a fiktív lejátszás módszerét bimátrix-játékokra.

8-2. Fiktív lejátszás véges játékokra
Általánosítsuk a fiktív lejátszás módszerét véges játékokra.

Megjegyzések a fejezethez

A játékelmélet témakörében eddig csak 1994-ben osztottak (közgazdasági) Nobel-díjat. A díjazottak között volt John Nash, aki a róla elnevezett Nash-egyensúly fogalmáért kapta a díjat. Ezt a fogalmat Nash 1951-ben vezette be [342].

Egy másik, szűkebb egyensúlyfogalmat alkalmaz a visszafelé indukció algoritmusa. Ezen algoritmus Kuhn nevéhez köthető és megtalálható Kuhn és Tucker cikkében [267]. Mivel ezen algoritmus a Nash-egyensúlynál szigorúbb feltételű egyensúlyt határoz meg, az ezen módszerrel kapott egyensúly egyben Nash-egyensúly is.

Az egyensúly megtalálásának problémája, illetve magának az egyensúly létezésének problémája matematikailag a fixpontproblémának feleltethető meg. A különböző fixponttételek – mint a Brouwer-féle [60], a Kakutani-féle [236], a Tarski-féle [461] – segítségével

bizonyítható az egyensúly létezése bizonyos játékosztályokban. Az egyensúly létezésének bizonyítására Kakutani-féle fixponttétellel lásd a [350] cikket. Magának a fixpontnak (vagy fixpontoknak) a megkeresésére is ismertek módszerek, lásd például a [447] és [134] könyveket. A legnépszerűbb létezési eredmény Nikaido és Isoda nevéhez fűződik [350].

A Fan-egyenlőtlenséget, mely szerepet játszik a folytonos játékok egyensúlyának jellemzésében, részletesen tárgyalja könyvében Aubin [22]. A Kuhn–Tucker-feltételek leírása megtalálható Martos Béla könyvében [311]. A Kuhn–Tucker-feltételek eltérésváltozókkal átírhatók nemnegatív egyenletrendszerré, mely rendszerek megoldására [447] és [311] tartalmaznak numerikus módszereket.

A bimátrix-játékok egyensúlyi problémájának átírása kevert változós feladattá megtalálható Mills [333] és Shapiro [427] cikkében. A bimátrix-játékok egyensúlyi problémája felírható kvadratikus programozási feladatként is (lásd Mangasarian cikkét [309]).

A fiktív lejátszás módszere megtalálható részletesen Robinson cikkében [394]. A Neumann-módszer alkalmazásakor differenciálegyenletet kell megoldanunk – ehhez a Runge–Kutta-módszert használtuk. Ennek a módszernek a leírása megtalálható a [447] könyvben.

Az átlósan szigorú konkáv játékok leírása Rosen cikkében [397] található meg. Az N személyes játékok egyensúlyának numerikus meghatározására Zuhovitzky, Polyak és Primak [512] javasoltak numerikus módszert.

A klasszikus Cournot-modell általánosítására nézve lásd Okuguchi és Szidarovszky könyveit [353, 354]. A 8.20. tétel bizonyítása a [445] cikkben található meg. A (8.58) lemma bizonyítására lásd a [354] monográfiát. Az intervallumfelezéses módszer leírását [447] tartalmazza. [238] olyan módszereket ír le, melyek alkalmasak nemlineáris komplementaritási feladatok megoldására. A (8.80) feladat megoldása megtalálható Hadley monográfiájában [187]. A nemlineáris programozással foglalkozik magyar nyelven Kovács Margit könyve [265].

A játékelmélet klasszikus tankönyve Neumann János és Oscar Morgenstern műve [346]. Magyar nyelven 1986-ban Szidarovszky Ferenc és Molnár Sándor [446], 1999-ben Kiss Béla és Krebsz Anna [254], 2004-ben pedig Mészáros József [341] adtak közre tankönyvet.

IV. DISZKRÉT OPTIMALIZÁCIÓ

Előszó

Ez a rész a diszkrét optimalizációval foglalkozó fejezeteket tartalmazza. Az első kötetben jelenik meg az *Ütemezéselmélet* című fejezet, amelynek fő témái: egy formális rendszer ütemezési feladatok osztályozására, az ütemezések szemléltetésére szolgáló Gantt-diagramok, egygépes feladatok, párhuzamos berendezésekkel kapcsolatos feladatok, végül az egy- és többutas ütemezési feladat. Ebben a fejezetben végig feltesszük, hogy az ütemezés megkezdésekor minden adat ismert.

A második kötet része lesz az *Online ütemezés* című fejezet, amelyben az ütemező algoritmus folyamatosan kapja az adatokat, és a kapott adatokat – a később beérkező adatok ismerete nélkül – fel kell dolgozni.

9. Ütemezéselmélet

Egy ütemezési feladat megoldása egy üzem, üzemrész, műhely vagy berendezés tevékenységének meghatározását jelenti rögzített időszakon belül. Ez két dolgot takar: egyrészt egy termelési feladat kiválasztását, másrészt a kiválasztott feladat megoldásának időbeli megtervezését. Ehhez számba kell venni az üzemben ható összes tényezőt, és ezeknek megfelelően kialakítani egy modellt, majd azt minden konkrét esetben megoldani. Ebben a fejezetben feltesszük, hogy az üzem működése pontosan jelezhető előre, ezért csak determinisztikus modellekkel foglalkozunk.

A termelést fel kell osztani kisebb részekre. Nem mindegy, hogy a dolog melyik oldalát ragadjuk meg. Ha a termelésnek a tevékenység jellege a fontos, akkor ***feladatokról*** vagy ***tevékenységekről*** beszélünk, ha azonban a termelés anyagi oldalát kívánjuk hangsúlyozni, akkor a ***munkadarab*** vagy a ***sorozat*** fogalmát használjuk. (Az utóbbin egyforma munkadarabokat kell érteni, amelyek együtt haladnak át a termelés minden fázisán.) A munkadarabok és a sorozatok sok tekintetben hasonlók, eltérő tulajdonságaikra külön felhívjuk a figyelmet. Ha a munkadarab vagy a sorozat fogalmával dolgozunk, akkor azt gondoljuk, hogy az adott üzem termékei fizikailag elkülöníthető darabokból állnak, a termelés pedig úgy folyik, hogy a majdani termék valamely kiinduló állapotban bekerül az üzembe, és bár változik az egyes fázisokon, lényegében mégis ugyanaz marad, azaz nem épül össze más termékekkel, illetve nem válik több részre. Ebben az esetben a munka tárgyát tekintjük alapvetőnek, de ekkor is szükség van arra, hogy a munkát, mint tevékenységet felosszuk részekre. Ebből a célból bevezetjük a ***művelet*** fogalmát. Egy művelet az az átalakítás, amit egy gép egyszerre elvégez a munkadarabon. Ha tehát egy munkadarab többször kerül ugyanarra a gépre, akkor több műveletről van szó.

A rendszer harmadik fontos elemét a technológiai körülmények alkotják, melyeken mind az általános, mind a vizsgált időszakra vonatkozó egyedi feltételeket, ezen belül a ***gépek*** kapacitását és a termelés során rendelkezésre álló és igényelt ***erőforrásokat,*** a ***raktározási*** és ***szállítási*** feltételeket értjük. Modellezési szempontból ide soroljuk a munkaerőt is. A különböző gépek általában különböző technológiai feladatokat látnak el, de előfordulhat, hogy egyes gépek helyettesíthetik egymást.

A gépekkel kapcsolatos legfontosabb kérdések a következők:

1. miben mérjük a ***kapacitásukat;***
2. mi a gépeknek a technológiai folyamatban betöltött helye;
3. mi a gépek és a megmunkálások viszonya.

Ezek a kérdések természetesen nem függetlenek egymástól.

Mivel egy vállalatnál a termelés célja nyereség elérése, ezért a modellben tekintettel kell lenni a költségekre, a termékek eladási árára és a terméken keletkező nyereségre.

Mindaz, amit eddig felsoroltunk, még csak egy elvi üzem kereteit szabja meg. Attól válik működő üzemmé, hogy egy több-kevesebb pontossággal meghatározott ***termelési feladatot*** kell elvégeznie, aminek a megszervezése a feladat, ami igen erős megszorítást jelent a lehetőségeknek a technológiai feltételek által megengedett végtelen gazdag tárházával szemben

A továbbiakban egyes esetekben nem kell megkülönböztetni a tevékenységeket, a munkadarabokat és a sorozatokat. Ilyenkor összefoglaló néven ***munkafeladatról*** fogunk beszélni.

9.1. Formális rendszer ütemezési feladatok osztályozására

Ismert egy formális rendszer, amellyel az ütemezési feladatokat nagyon tömören le lehet írni. Ez a formális rendszer nem öleli föl az összes lehetőséget, ezért a következő feltételezések állandóan érvényben vannak:

1. minden gép a teljes vizsgált időszakban rendelkezésre áll;
2. minden munkafeladat elvégzésére egyetlen technológiát határoztunk meg;
3. minden gép egyszerre csak egy munkadarabon dolgozhat;
4. minden gép, az esetlegesen szükséges átszerszámozástól eltekintve, azonnal megkezdheti a következő megmunkálást, mihelyst az előzőt befejezte;
5. a gyártásközi raktár(ak) kapacitása végtelen;
6. az előzés megengedett;
7. a soron következő munkadarab azonnal megmunkálható, mihelyst a gép szabaddá vált, feltéve, hogy a munkadarab előző megmunkálása már befejeződött.

A feladatok leírása három mezőből áll, melyeket így szokás jelölni:

$$\alpha \mid \beta \mid \gamma \,, \tag{9.1}$$

ahol α a gépekre és a legfontosabb technológiai adottságokra, β a munkafeladatokra, illetve ezeknek a gépekhez és egyéb körülményekhez való kapcsolatára vonatkozó feltételeket tartalmazza, végül γ a matematikai feladat célfüggvényét írja le. Az egyes mezők több tényezőre vonatkozó információt is tartalmazhatnak.

A j-edik munkafeladat befejezési időpontja C_j, határideje d_j. Ezen munkafeladat műveleteinek száma m_j, és az i-edik gépen való megmunkálásának ideje p_{ij}.

9.1.1. Az α mező

Itt kell megadni a gépek számát és a technológiai útvonal típusát. A technológiai útvonal azt határozza meg, hogy a munkadarab milyen sorrendben keresi fel a gépeket.

- 1,2,. . .: A gépek száma 1 (vagy 2 stb.), a megadott számtól függően.
- *P*: Azonos párhuzamos gépek vannak, azaz egyetlen homogén gépcsoportról van szó, ahol bármely művelet bármely gépen elvégezhető, és minden művelet bármelyik gépen ugyanannyi ideig tart.
- *Q*: Hasonló párhuzamos gépek – csak a gépek sebességében van különbség, azaz a *j*-edik műveletnek az *i*-edik gépen való elvégzése p_j/s_i ideig tart, ahol p_j csak a művelettől, s_i csak a géptől függő állandó.
- *R*: Általános párhuzamos gépek, ahol az egyes műveletek ideje tetszőleges lehet a különböző gépeken.
- *O*: A technológiai útvonal kötetlen.
- *F*: Egyutas ütemezési probléma, azaz minden munkadarab azonos sorrendben keresi fel a gépeket.
- *J*: Többutas ütemezési probléma, azaz a munkadarabok technológiai útvonala eltérő.

Például a *J*3 többutas ütemezési problémában a gépek száma 3. Minden munkadarab technológiai útvonalának leírását külön meg kell adni.

9.1.2. A β mező

Ez a mező tartalmazza az egyéb technológiai feltételeket.

- $p_{ij} = 1, p_{ij} \in \{1, 2\}$ stb.: A megmunkálási idők speciálisak, például mindegyik 1 (azonos hosszú), illetve mindegyik 1 vagy 2.
- *idle*: Olyan ütemezés is lehetséges, ahol betervezett állásidők vannak, azaz egy berendezés annak ellenére sem dolgozik, hogy van olyan munkadarab, amelyik megmunkálásra vár.
- *pmtn*: A műveletek megszakítása megengedett. Ilyenkor a megszakítás nem okoz veszteséget sem időben, sem költségben.
- *prec, tree*: A munkafeladatok között megelőzési reláció áll fenn, melyet valamely *G* irányított gráf ír le. Ha a mezőben *tree* szerepel, akkor *G* fa.
- r_j: Az egyes munkafeladatoknak különböző lehet a rendelkezésre állási ideje, illetve, ha itt még további feltétel szerepel, akkor a rendelkezésre állási idők ennek eleget tesznek.
- d_j: Az egyes munkafeladatoknak különböző lehet a határideje, illetve, ha itt még további feltétel szerepel, akkor a határidők ennek eleget tesznek. A határidőket ***puha,*** azaz megsérthető feltételeknek tekintjük. Ha a határidők ***tényleges*** feltételeket jelentenek, akkor egy szokásos jelölés $\overline{d}_j$.)
- *no-wait*: A munkadarabok nem várhatnak a gépek előtt.
- *res, res*1 Az erőforrások csak korlátozottan állnak rendelkezésre. Ha a mezőben *res*1 áll, akkor egyetlen erőforrásról van szó.
- $m_j \leq \hat{m}$: Az egyes munkadarabokhoz legfeljebb $\hat{m}$ művelet tartozik.
- *overlap*: átlapolás korlátlanul megengedett, azaz egy újabb gép már dolgozhat a munkadarabon, miközben az előző még nem fejezte be a műveletet. (Ez a sorozatok jellegzetes tulajdonsága.)

A konvenció szerint, ha egy elem nem szerepel, akkor a megfelelő megszorítás nem érvényes. Például, ha nincs a mezőben *pmtn*, akkor ez azt jelenti, hogy a megmunkálások megszakítása nem lehetséges.

9.1.3. A γ mező

A formális rendszer csak kompromisszumos, azaz a késéseket nem kizáró célfüggvényeket enged meg. Ennek oka az, hogy minden más esetben nehezen kezelhető a feladat, és csak lineáris és egészértékű programozási feladatokkal modellezhető a probléma.

Az egyes munkafeladatokhoz a következő értékeket értelmezzük:

- C_j : a befejezési időpont,
- $L_j = C_j - d_j$: a késés,
- $T_j = \max\{0, L_j\}$: a tényleges késés,
- U_j =sgn(T_j) : egységnyi büntetés.

Mindegyikből többféleképpen lehet célfüggvényt formálni, például minimalizálhatjuk a megfelelő értékek maximumát, összegét, súlyozott összegét. Ennek megfelelően tizenkét lehetséges célfüggvény a következő

$$\begin{gathered} C_{\max},\ L_{\max},\ T_{\max},\ U_{\max}\ , \\ \sum C_j,\ \sum L_j,\ \sum T_j,\ \sum U_j\ , \\ \sum w_j C_j,\ \sum w_j L_j,\ \sum w_j T_j, \sum w_j U_j\ , \end{gathered} \tag{9.2}$$

ahol a súlyok nemnegatívak.

Ha a munkafeladatok száma adott, akkor az összeg minimalizálása ekvivalens a megfelelő mennyiség *átlagának* a minimalizálásával, hiszen a két érték csak egy konstans szorzóban tér el egymástól. Azt az esetet, amikor a munkafeladatok folyamatosan érkeznek be, és a helyzet állandóan változik, *on-line* problémának nevezik. Ekkor csak a várható értéket lehet optimalizálni. Vagyis az összeg célfüggvény az on-line eset modellezését is szolgálja.

Látható, hogy a lényegesen különböző célfüggvények száma kevesebb. Ugyanis $\sum w_j C_j$ és $\sum w_j L_j$ csak a konstans $-\sum w_j d_j$ tagban különböznek egymástól. Továbbá $L_{\max}$ minimalizálása egyben minimalizálja a $T_{\max}$ és $U_{\max}$ célfüggvényeket is.

Az utóbbi kettő közül az első minimalizálja a másodikat, hiszen $U_{\max} = 0$ akkor és csak akkor, ha egyetlen munkadarab sem késik, azaz $T_{\max} = 0$. Az $U_{\max}$ célfüggvény szerepe csak annyi, hogy olyan ütemezést keresünk, amely egyetlen határidőt sem sért meg.

A (9.2)-ben megadott függvények a ***reguláris célfüggvények*** közé tartoznak, melyek értéke a befejezési időpontokban monoton növekedő. ***Irregulárisnak*** nevezünk egy célfüggvényt, ha nem rendelkezik ezzel a tulajdonsággal. Ilyen az ú.n. ***sietés*** és a ***súlyozott pontosság*** minimalizálása, ahol

- $E_j = \max\{d_j - C_j, 0\}$: a sietés.
- $\sum(v_j E_j + w_j T_j)$: a súlyozott pontosság.

Ezen célfüggvényeknek akkor van létjogosultsága, ha pontos előrejelzéssel rendelkezünk az eladások várható időpontjáról. Ekkor előre gyártani azért haszontalan, mert raktározási költséggel jár.

Némely esetben nem konkrét célfüggvényről lesz szó, hanem valamely tulajdonságokat kielégítő valamennyi célfüggvényről. Ilyen esetben a γ mezőbe f kerül.

9.1. példa. Az $1 \parallel \sum C_j$ feladat az az egygépes ütemezési feladat, ahol a termékek ütemezésére a vizsgált időszakon belül nincs megszorítás, azaz nincsenek határidők, rendelkezésre állási idők, megelőzési relációk stb.

A befejezési idők összegét kívánjuk minimalizálni. Könnyen látható, hogy a feladat optimális megoldásában a munkadarabokat a megmunkálási idők növekvő sorrendjébe rakjuk. Ezt a gyakran előforduló sorrendet az angol irodalomban SPT ütemezésnek nevezik.

9.2. példa. A klasszikus egyutas ütemezési probléma háromgépes speciális esete $F3 \parallel C_{\max}$.

Gyakorlatok

9.1-1. Írjuk le a formális nyelven azt a feladatot, amelyben a munkák azonos sorrendben keresik fel a gépeket, és nem várhatnak a munkák a gépek előtt.

9.1-2. Van-e az $1|r_j|C_{\max}$ feladatban olyan munka, ami $t = 0$-ban nem végezhető?

munka	1. gép	2. gép	3. gép
1.	3	22	2
2.	22	20	20
3.	20	14	18

9.2. A Gantt-diagramok

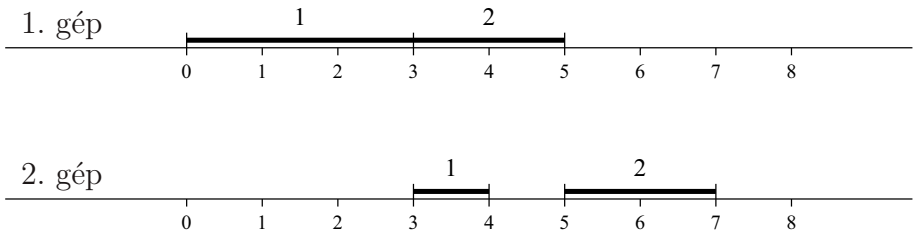
Az ütemezések szemléltetésének fontos eszköze az ú.n. Gantt-diagram. Ebben minden gépet egy-egy időtengellyel szemléltetnek. Az időtengelyeken feltüntetik, hogy az egyes munkadarabok mikor kerülnek megmunkálásra az adott gépen, illetve a gépnek mikor van állásideje.

9.3. példa. Tekintsük azt az $F2 \parallel C_{\max}$ feladatot, ahol $n = 2$, és a megmunkálási időket az alábbi táblázat foglalja össze.

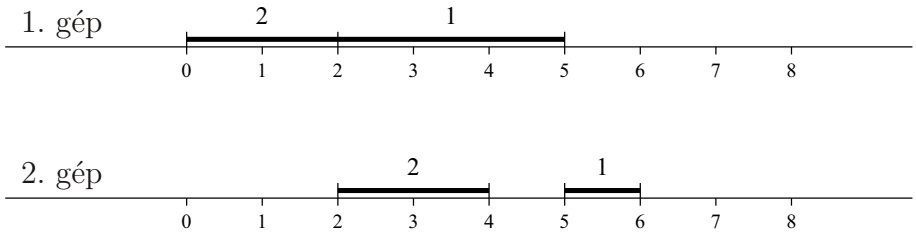
munkadarab	1. gép	2. gép
1.	3	1
2.	2	2

Ha mindkét gépen az (1,2) sorrendet alkalmazzuk, akkor a 9.1. ábrához jutunk.

Ha pedig a fordított sorrend szerinti ütemezést tekintjük, akkor a 9.2. ábrán látható Gantt-diagramot kapjuk.



9.1. ábra. A Gantt-diagram az (1,2) ütemezés esetén.



9.2. ábra. A Gantt-diagram a (2,1) ütemezés esetén.

A Gantt-diagramok fontos eszközei a matematikai bizonyítások szemléltetésének. Segítségükkel könnyen beláthatók az alábbi állítások. Itt végig feltesszük, hogy az egyes gépeken csak a megmunkálások sorrendjét kell megadni, ugyanis nincs értelme az ütemezés szerint soron következő megmunkálással várni. Ez még nem zárja ki a betervezett állásidők lehetőségét, hiszen itt még lehetséges, hogy egy gép előtt egy munkadarab bevár egy másik munkadarabot azért, hogy az megelőzze.

Tekintsünk egy tetszőleges ütemezést és egy rögzített i gépet. A gép működését a Gantt-diagram egybefüggő működő és álló szakaszokra osztja fel, hiszen mihelyst lekerül egy munkadarab a gépről, azonnal rákerül a másik, azaz úgy tekintjük, hogy a gép folyamatosan dolgozik, feltéve persze, hogy van egyáltalán az adott pillanatban végezhető munkája. Tekintsünk egy ilyen összefüggő szakaszt. Jelölje

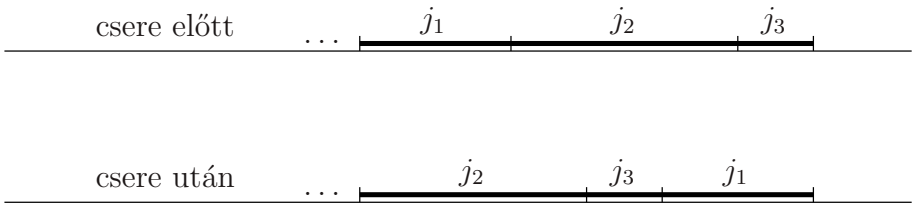
- J az ehhez a szakaszhoz tartozó munkadarabok halmazát,
- t a szakasz kezdetének időpontját,
- r_{ij} ($j \in J$) a j-edik munkadarab megérkezésének időpontját az i-edik gépre adott ütemezés szerint (r_{ij} tulajdonképpen a j-edik munkadarab rendelkezésre állási ideje az i-edik gépen)
- u_{ij} ($j \in J$) a j-edik munkadarab ütemezését, vagyis a j-edik munkadarab megmunkálásának megkezdését, az i-edik gépen.

Ekkor nyilvánvaló, hogy

$$\text{minden } j \in J \text{ esetén } \quad t \leq r_{ij} \ . \tag{9.3}$$

Különben ugyanis közvetlenül a t időpont előtt a gép állna, miközben egy munkadarab várna a megmunkálásra. Hasonlóképpen teljesülnie kell, hogy

$$\text{minden } i \text{ és minden } j \in J \text{ esetén } \quad r_{ij} \leq u_{ij} \ . \tag{9.4}$$



9.3. ábra. A (9.5) feltételt kielégítő cserének nincs hatása a $C_{\max}$ értékre.

Az összefüggő szakaszon belül átrendezhetjük a megmunkálások sorrendjét, de természetesen az így adódó új ütemezésnek is ki kell elégítenie a (9.4) egyenlőtlenségeket. Továbbá, ha a működő szakasz összefüggő marad, akkor teljesülnie kell a

$$t + \sum_{l=1}^{k} p_{i\pi(l)} \geq r_{i\pi(k+1)}, \quad k = 0, \ldots, |J| - 1 , \tag{9.5}$$

egyenlőtlenségeknek, ahol π a J-beli munkadarabok új sorrendjét jelöli.

Végül tegyük fel, hogy az i-edik gép minden technológiai útvonalon az utolsó, azaz itt befejeződik a termelés. Legyen továbbá a vizsgált működő időszak az utolsó. Ekkor bármilyen átrendezés ezen a szakaszon, ami megőrzi a szakasz összefüggését, nem változtatja meg az utolsónak befejeződő munkadarab befejezési időpontját, ami

$$t + \sum_{j \in J} p_{ij} , \tag{9.6}$$

azaz nem változtatja meg a $C_{\max}$ értéket (lásd a 9.3. ábrát).

Gyakorlatok

9.2-1. Ábrázoljuk az alábbi egyutas ütemezési feladat (3,2,1) sorrendű ütemezését Gantt-diagramon.

munka	1. gép	2. gép	3. gép
1.	3	22	2
2.	22	20	20
3.	20	14	18

9.2-2. Hogyan változik a célfüggvény értéke az $F2||C_{\max}$ feladatban, ha úgy rendezzük át a munkákat a második gép utolsó összefüggő szakaszán, hogy az egynél több szakaszra bomlik?

9.3. Ütemezési problémák egyetlen gépen

A legegyszerűbb ütemezési feladatokban csak egyetlen gép van. A valóság általában bonyolultabb. De ez az eset részfeladatként felmerül a bonyolultabb helyzetekben. Továbbá

jól alkalmazható, ha valamelyik berendezés szűk keresztmetszetet jelent.

Az alábbi három tétel azt mondja ki, hogy számos esetben nem érdemes megszakítást és betervezett állásidőt tartalmazó ütemezéssel foglalkozni.

9.1. tétel. *Ha f reguláris célfüggvény, akkor az $1 \mid d_j, prec, idle, pmtn \mid f$ és az $1 \mid d_j, prec, idle \mid f$ feladatoknak van közös optimális megoldása.*

Megjegyzés. Mivel a feladatok leírásában a rendelkezésre állási idők nem szerepelnek, ezért a konvenció szerint minden j esetén $r_j = 0$.

Bizonyítás. Bármely pillanatban minden – még be nem fejeződött – munkadarab megmunkálható, feltéve természetesen, hogy előzményei elkészültek már. Ezt a feltételt azonban minden megengedett ütemezésnek teljesítenie kell. Tegyük fel, hogy valamely megengedett ütemezésben a j_1 munkadarab megmunkálását a j_2 munkadarab kedvéért megszakítottuk. Innen következik, hogy j_1 nem előzménye j_2-nek, és j_2 minden előzménye befejeződött j_1 megmunkálásának ezen, megszakított szakasza előtt. Tekintsük most már azt az ütemezést, amit úgy nyerünk, hogy j_1 megmunkálásának ezen megszakított szakaszát csatoljuk a folytatásához, közvetlenül az elé, és a j_1 ezen két megmunkálása közé eső megmunkálásokat előbbre hozzuk. Ez megtehető, hiszen j_1 ezen megmunkálások után fog befejeződni, így j_1 nem előzménye egyetlen, ebbe a szakaszban befejeződő munkadarabnak sem. Ezen átrendezés mellett egyetlen munkadarab esetén sem növeltük a C_j értéket, így a célfüggvény regularitásából adódik, hogy a célfüggvény értéke nem növekedett. ■

9.2. tétel. *Ha f reguláris célfüggvény, akkor az $1 \mid d_j, prec, idle, pmtn \mid f$ és az $1 \mid d_j, prec, pmtn \mid f$, illetve az $1 \mid d_j, prec, idle \mid f$ és az $1 \mid d_j, prec \mid f$ feladatoknak van közös optimális megoldása.*

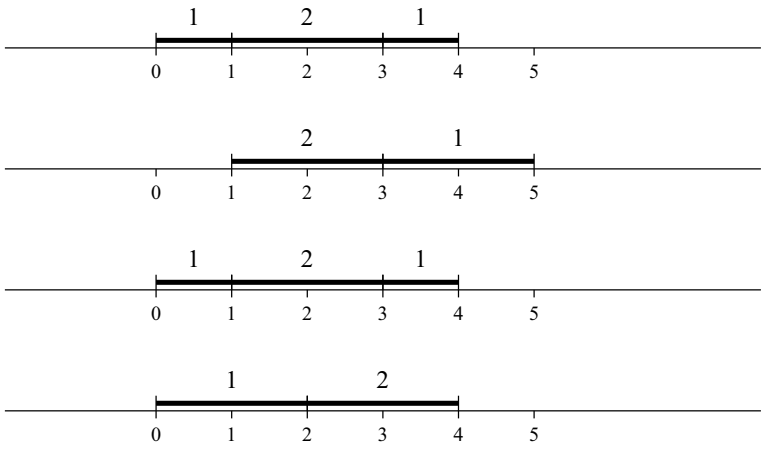
Bizonyítás. Tegyük fel, hogy a $[t_1, t_2]$ időintervallumban betervezett állásidő van. Ekkor a megmunkálások sorrendjét nem változtatva meg, a t_2 időpontban kezdődő szakaszt előrehozhatjuk $t_2 - t_1 > 0$ idővel. Ezáltal nem zavarjuk meg a kötelező megelőzések betartását, míg a C_j értékek nem nőnek, tehát a célfüggvény regularitásából következik ismét, hogy értéke nem nőtt. ■

Ebből a két tételből adódik egy újabb állítás.

9.3. tétel. *Ha f reguláris célfüggvény, akkor az $1 \mid d_j, prec, idle, pmtn \mid f$ és az $1 \mid d_j, prec \mid f$ feladatoknak van közös optimális megoldása.*

Bizonyítás. Az $1 \mid d_j, prec, idle, pmtn \mid f$ és az $1 \mid d_j, prec, idle \mid f$ feladatnak van közös optimális megoldása a 9.1. tétel szerint, az $1 \mid d_j, prec, idle \mid f$ és az $1 \mid d_j, prec \mid f$ feladatnak pedig a 9.2. tétel szerint. Ez a közös megoldás kielégíti az $1 \mid d_j, prec, idle, pmtn \mid f$ feladat feltételeit is, mert csak annyi történt, hogy nem használtuk ki a megszakítás és az ütemezett állásidő lehetőségét. ■

9.4. példa. Fontos rámutatni arra, hogy mennyire lényeges volt minden j esetén az $r_j = 0$ feltétel. Tekintsük azt a feladatot, amelyben csak két munkadarab van a következő adatokkal.



9.4. ábra. Eltérések az optimális megoldásban, ha a rendelkezésre állási idők különbözők.

munkadarab	p_j	r_j	d_j
1.	2	0	4
2.	2	1	2

Észrevehető, hogy a 2. munkadarabot semmiképp sem lehet határidőre befejezni, tehát a maximális késés, azaz $T_{max} \geq 1$. T_{max} reguláris célfüggvény. Az $1 \mid d_j, prec, idle, pmtn \mid T_{max}$, az $1 \mid d_j, prec, idle \mid T_{max}$, az $1 \mid d_j, prec, pmtn \mid T_{max}$ és az $1 \mid d_j, prec \mid T_{max}$ feladatok optimumát rendre a 9.4. ábrán látható ütemezések adják.

Az utolsó esetben csak egyetlen sorrend lehetséges, ami persze így optimális is. Ez az egyetlen olyan megoldás, ahol $T_{max} = 2$, minden más esetben $T_{max} = 1$. A fenti megoldások közül az első és a harmadik azonos, és a második is optimális megoldása az első feladatnak, azonban ennél mindkét munkadarab késik, míg az első megoldás esetében csak a 2. munkadarab.

Az alábbiakban néhány könnyebb feladat optimális megoldását adjuk meg. Az optimális megoldást minden esetben valamely általánosan használható, egyszerű heurisztikus szabály szolgáltatja, így a szükséges számítások mennyisége kicsi. Számos más feladat azonban ú.n. NP-teljes probléma, azaz jelenlegi tudásunk mellett csak leszámlálási módszerekkel oldható meg. Egy optimalizálási feladat megoldó eljárását akkor nevezzük leszámlálási módszernek, ha valamennyi szóba jöhető megoldást explicit módon kipróbál vagy implicit módon kizár azok közül, akik optimálisak lehetnek.

Azonban az ismertetendő heurisztikus módszerek ezekben az esetekben is képesek jó megengedett megoldások előállítására, a célfüggvény optimális értékének becslésére.

Az említett heurisztikus módszerek mindig valamilyen egyértelmű sorrendet jelentenek. Ennek a sorrendnek a jelölése: $[1], [2], \ldots, [n]$, ahol $[1]$ a sorrend első elemét, $[2]$ a sorrend második elemét jelenti stb. Ha egy bizonyításban több sorrendet vizsgálunk, akkor ezeket különböző zárójelekkel jelöljük.

9.4. tétel. *Az $1 \mid d_j \mid L_{max}$ feladat egy optimális megoldását a*

$$d_{[1]} \leq d_{[2]} \leq \cdots \leq d_{[n]} \tag{9.7}$$

sorrend adja meg.

Megjegyzés. Az állításban szereplő sorrend neve a ***legkorábbi határidők sorrendje*** (EDD).
Bizonyítás. Jelöljön $< 1 >, < 2 >, \ldots, < n >$ egy tetszőleges sorrendet. Ha ez nem EDD sorrend, akkor van egy olyan l index, hogy

$$d_{<l>} > d_{<l+1>}$$

Tegyük fel, hogy $< l >$ megmunkálása a T időpontban kezdődik. A két munkadarab késése ekkor

$$L_{<l>} = C_{<l>} - d_{<l>} = T + p_{<l>} - d_{<l>} \ ,$$
$$L_{<l+1>} = C_{<l+1>} - d_{<l+1>} = T + p_{<l>} + p_{<l+1>} - d_{<l+1>} \ .$$

Tekintsük azt az $(1), (2), \ldots, (n)$ sorrendet, amely csak annyiban különbözik ettől, hogy az említett két munkadarabot felcseréljük. Így a többi munkadarab késése nem változik, míg a két felcserélt munkadarab új késése

$$L_{(l)} \ = \ C_{(l)} - d_{(l)} \ = \ C_{(l)} - d_{<l+1>} \ = \ T + p_{<l+1>} - d_{<l+1>} \ ,$$
$$L_{(l+1)} \ = \ C_{(l+1)} - d_{(l+1)} \ = \ C_{(l+1)} - d_{<l>} \ = \ T + p_{<l>} + p_{<l+1>} - d_{<l>} \ .$$

Könnyen látható, hogy

$$L_{<l+1>} \ \geq \ L_{(l)}, L_{(l+1)} \ ,$$

vagyis a célfüggvény értéke nem nőtt. Ezt a gondolatmenetet mindaddig megismételhetjük, amíg a cserékkel egy EDD sorrendet nem kapunk. ■

Az $1 \mid r_j, d_j \mid L_{max}$ feladat esetében már nem ilyen egyszerű a helyzet, mert a probléma teljes általánosságában NP-teljes. Az előbbi heurisztikus módszer következő változata speciális esetekben megoldja a feladatot.

9.5. tétel. *Ha egy, az* $1 \mid r_j, d_j \mid L_{max}$ *feladatosztályhoz tartozó feladat esetében teljesül, hogy nem létezik olyan*

$$j \text{ és } l, \text{ melyekre } r_j < r_l \text{ és } d_l < d_j \ , \tag{9.8}$$

akkor az alábbi rekurzív módszer a feladat egy optimális megoldását adja. Tegyük fel, hogy a keresett sorrend első $j-1$ *tagját már meghatároztuk. Legyen* $N = \{1, \ldots, n\} \backslash \{[1], \ldots, [j-1]\}$, *továbbá*

$$\alpha = \max\{C_{[j-1]}, \min\{r_l : \ l \in N\}\}$$

és

$$S \ = \ \{l \in N : \ r_l \ \leq \ \alpha \ \} \ .$$

Legyen $[j] \in N$ *egy olyan munkadarab, amelyre*

$$d_{[j]} \ = \ \min\{ \ d_l : \ l \in S \ \} \ . \tag{9.9}$$

Megjegyzés. A kiválasztás logikája tehát az, hogy $[j]$ a még nem ütemezett munkadarabok közül egyike azoknak, amelyek megmunkálása a legkorábban megkezdődhet és ezek közül a legkorábbi határidővel rendelkezik.

Bizonyítás. Legyen $< 1 >, < 2 >, \ldots, < n >$ egy tetszőleges sorrend. Erre vonatkozóan legyen $N_j = \{1, \ldots, n\} \backslash \{< 1 >, \ldots, < j-1 >\}$,

$$\alpha_j = \max\{C_{<j-1>}, \min\{r_l : \ l \in N_j\}\}$$

és

$$S_j = \{l \in N_j : \ r_l \leq \alpha_j \} .$$

Nyilvánvaló, hogy az α_j értékek monoton növő sorozatot alkotnak. Ezért, ha $l \in S_j$ teljesül, akkor $l \in S_{j+1}, S_{j+2}, \ldots$ mindaddig, amíg az l munkadarab ütemezésre nem kerül. Ha az $< 1 >, < 2 >, \ldots, < n >$ sorrend nem elégíti ki a rekurzív szabályokat, akkor két eset lehetséges, nevezetesen valamely j indexre

(i) $[j]$ megmunkálása nem a legkorábban megkezdhetők közé tartozik, azaz

$$r_{[j]} > \alpha_j, \text{vagy}$$

(ii) van olyan $l \in S_j$, hogy

$$d_l < d_{[j]}, \quad r_l \leq \alpha_j . \tag{9.10}$$

Tekintsük először az (i) esetet. Vegyük észre, hogy α_j definíciójából következik, hogy mindig van olyan $l \in N_j$ munkadarab, amelyre $\alpha_j \geq r_l$ teljesül. Tegyük fel, hogy a vizsgált sorrendben $[k]$ az első ilyen $[j]$ után, azaz

$$r_{[j]}, r_{[j+1]}, \ldots, r_{[k-1]} > \alpha_j, \quad r_{[k]} \leq \alpha_j .$$

Ekkor (9.8)-ból következik, hogy

$$d_{[j]}, d_{[j+1]}, \ldots, d_{[k-1]} \geq d_{[k]} .$$

Tudjuk, hogy

$$L_{[k]} = C_{[k-1]} + p_{[k]} - d_{[k]} .$$

Innen

$$\begin{aligned} L_{[k]} - L_{[l]} &= C_{[k-1]} + p_{[k]} - d_{[k]} - C_{[l-1]} - p_{[l]} + d_{[l]} \\ &\geq p_{[k]} - d_{[k]} + d_{[l]} \geq p_{[k]}, \quad l = j, \ldots, k-1 . \end{aligned} \tag{9.11}$$

Vagyis $[k]$ késése legalább $p_{[k]}$-val nagyobb, mint a $[j], \ldots, [k-1]$ munkadarabok késése. Tekintsük most azt az $(1), (2), \ldots, (n)$ sorrendet, amelyet a

$$(l) = \begin{cases} [l], & \text{ha } j > l \text{ vagy } j > k , \\ [k], & \text{ha } j = l , \\ [l-1], & \text{ha } j < l \leq k \end{cases}$$

egyenlőség definiál. Ebben a sorrendben C_l értéke $l = 1, \ldots, j-1$ esetén változatlan, $j = k+1, \ldots, n$ esetén nem nőtt, legfeljebb csökkent, mert elmarad az $r_{[j]} - \alpha_j$ állásidő, továbbá $(l) = [l-1] \quad l = j+1, \ldots, k$ esetén pedig legfeljebb $p_{(j)} = p_{[k]}$ értékkel nőtt. Végül

$$L_{(j)} = \alpha_j + p_{(j)} - d_{(j)} ,$$

azaz

$$L_{(j)} - L_{[k]} = \alpha_j - C_{[k-1]} < 0 ,$$

tehát ebben az esetben a késés csökkent. Innen (9.11) alapján adódik, hogy a maximális késés nem nőtt.

Tekintsük most a (ii) esetet. Ekkor válasszuk meg a k indexet úgy, hogy $[k]$ legyen a sorrendben az első, amely (9.9)-et kielégíti. Ezután az előző esetben alkalmazott gondolatmenet megismételhető. ■

9.6. tétel. *A*

$$\frac{p_{[1]}}{w_{[1]}} \leq \frac{p_{[2]}}{w_{[2]}} \leq \cdots \leq \frac{p_{[n]}}{w_{[n]}} \tag{9.12}$$

sorrend az $1 \parallel \sum w_j C_j$ *feladat egy optimális megoldását adja.*

Megjegyzés. Ennek neve ***súlyozott legrövidebb megmunkálási idők sorrendje*** (SWPT).

Bizonyítás. Jelöljön $< 1 >, < 2 >, \ldots, < n >$ egy tetszőleges sorrendet. Ha ez nem SWPT sorrend, akkor van egy olyan l index, hogy

$$\frac{p_{<l>}}{w_{<l>}} > \frac{p_{<l+1>}}{w_{<l+1>}} . \tag{9.13}$$

Ha felcseréljük ezt a két munkadarabot, akkor a többi munkadarab befejezési ideje nem változik, vagyis a célfüggvény értékében bekövetkezett változás pusztán ettől a két munkadarabtól származik. A felcserélés előtt a

$$(C_{<l-1>} + p_{<l>})w_{<l>} + (C_{<l-1>} + p_{<l>} + p_{<l+1>})w_{<l+1>}$$

taggal járultak hozzá a célfüggvény értékéhez, míg utána a

$$(C_{<l-1>} + p_{<l+1>})w_{<l+1>} + (C_{<l-1>} + p_{<l+1>} + p_{<l>})w_{<l>}$$

mennyiséggel. A kettő különbsége

$$p_{<l>}w_{<l+1>} - p_{<l+1>}w_{<l>} ,$$

ami pozitív, mint az azonnal látható (9.13) alapján. ■

Ha valamennyi súly 1, vagyis pusztán az összeget minimalizáljuk, akkor (9.12) az SPT sorrendet adja.

Amikor a tétel állításában a feladatot leírtuk, akkor a rendelkezésre állási idők azért nem szerepeltek, mert mindegyik 0 volt, amit fel is használtunk akkor, amikor egy tetszőleges sorrendben két tetszőleges, egymást követő munkadarabot felcserélhetőnek vettünk. A határidők viszont azért nem szerepeltek, mert létük nem befolyásolja a feladatot, hiszen megsérthetők, így csak a célfüggvényre lehetnek hatással.

A következő tétel elég általános módszert ad arra az esetre, amikor a sorrendet megelőzési relációk korlátozzák.

9.7. tétel. *Minden j ($j = 1, \ldots, n$) esetén legyen f_j monoton növő függvény; legyen továbbá $f_{max}(C_1, \ldots, C_n) = \max\{f_j(C_j) : \quad j = 1, \ldots, n\}$. Jelölje N a munkadarabok halmazát, és egy tetszőleges $S \subset N$ esetén legyen $p(S) = \sum_{j \in S} p_j$ és $f^*_{max}(S)$ azon feladatnak optimális megoldásának értékét, amelyben csak az S-beli munkadarabok szerepelnek. Végül legyen V azon munkadarabok halmaza, amelyeknek nincs rákövetkezője. Ekkor*

$$f^*_{max}(N) \;=\; \min\{\max\{f_j(p(N)),\ f^*_{max}(N\backslash\{j\})\} : \quad j \in V\}\ . \tag{9.14}$$

Megjegyzés. A tétel feltételeiből következik, hogy a célfüggvény reguláris.

Bizonyítás. Nyilvánvaló, hogy bármely megengedett sorrendben a legutolsó munkadarab csak olyan lehet, amelynek nincs rákövetkezője. Tegyük fel, hogy az optimális sorrendben a j munkadarab az utolsó. Ekkor $f^*_{max}(N)$ értékét vagy j adja, és az ekkor $f_j(p(N))$, vagy egy másik munkadarab, és az ekkor $f^*_{max}(N\backslash\{j\})$. ■

A (9.14) képlet segítségével megadható egy $O(n^2)$ futási idejű algoritmus a feladat megoldására.

Mint említettük, a megszakítás lehetőségének akkor láthatjuk előnyét, amikor a rendelkezésre állási idők különbözők. Erre ad egy példát az alábbi tétel az SPT sorrend egy megfelelő általánosításával. Tekintsük az $1 \mid r_j, pmtn \mid \sum C_j$ feladatot. Legyen $T = \{r_1, \ldots, r_n\}$. Tegyük fel, hogy a t időpontig bezárólag elkészült már az ütemezés, amikor a j_t munkadarab mellett döntöttünk.

Jelölje a munkadarabok mindenkori még megmaradt megmunkálási időit $p'_1, \ldots, p'_n$. A t időpont után a következő időpont, amikor dönteni kell,

$$u \;=\; \min\{t + p'_{j_t}, \min\{s \in T : \ s > t\}\}\ .$$

Azt kell tehát eldönteni, hogy az u időpontban melyik j_u munkadarab kerüljön megmunkálásra. Erre a következő kiválasztási szabályt alkalmazzuk:

$$p'_{j_u} \;=\; \min\{p'_j : \ p'_j > 0;\ r_j \le u\}\ . \tag{9.15}$$

A sorrend neve: ***legrövidebb megmaradt megmunkálási idők sorrendje*** (SRPT)

9.8. tétel. *Az $1 \mid r_j, pmtn \mid \sum C_j$ feladatnak egy optimális megoldását az SRPT ütemezés adja meg.*

Bizonyítás. A bizonyítás hasonló a 9.6. tételéhez, ezért azt az Olvasóra bízzuk. ■

A feladat súlyozott változata, azaz $1 \mid r_j, pmtn \mid \sum w_j C_j$, azonban NP-teljes.

Végül egy olyan feladatot mutatunk be, ahol a célfüggvény nem reguláris: a késések és sietések összegét minimalizáljuk, feltéve, hogy a munkadarabok határideje azonos. Formális jelöléssel az $1 \mid idle, d_j = d \geq \sum_{j=1}^{n} p_j \mid \sum(E_j + T_j)$ feladatot tárgyaljuk. Itt tehát arról van szó, hogyan lehet a közös határidő körül szétosztani a munkákat úgy, hogy azok a lehető legközelebb legyenek a határidőhöz. Akkor merülhet fel ilyen vagy hasonló feladat, ha az igény erős szezonális hatást mutat, azaz a szállításoknak rövid időn belül kell megtörténniük.

Minden ütemezést egy kezdeti és egy befejező szakaszra bontunk fel. A kezdeti szakaszhoz tartoznak mindazok a munkadarabok, amelyek megmunkálása befejeződött a közös d határidőig bezárólag, a befejező szakaszhoz tartozik az összes többi munkadarab. A kezdeti szakasz, mint időintervallum, tart a szakaszhoz tartozó első munkadarab megmunkálásának kezdetétől, a szakaszhoz tartozó utolsó munkadarab megmunkálásának befejezéséig, a befejező szakasz pedig a kezdeti szakasz végétől az utolsó munkadarab megmunkálásának befejezéséig.

Most néhány lemmában leírjuk az optimális megoldások legfontosabb tulajdonságait, melyek segítségével aztán egy egyszerű algoritmus adódik azok előállítására.

9.9. lemma. *Az optimális megoldásokban az első és az utolsó megmunkálás között nincs állásidő.*

Bizonyítás. Ha van állásidő, akkor a kezdeti szakasz esetén az állásidő előtti, a befejező szakasz esetén az állásidő utáni megmunkálásokat a határidő felé tolhatjuk el úgy, hogy a sorrend változatlan marad. Ezzel a célfüggvény értéke csökken. ■

9.10. lemma. *Van olyan optimális ütemezés, amelyben egy megmunkálás pontosan a határidőkor fejeződik be.*

Bizonyítás. Tekintsünk egy olyan ütemezést, amelyben az első és az utolsó megmunkálás között nincs állásidő, és amelyre az állítás nem igaz. Ekkor a befejező szakasz első munkadarabjának, j_b-nek, a megmunkálási időintervalluma a belsejében tartalmazza a határidőt. Toljuk el időben az egész ütemezést a következő módon:
(i) ha a kezdeti szakasz tartalmaz kevesebb megmunkálást, akkor j_b megmunkálásának végpontja essen egybe d-vel,
(ii) ha a befejező szakasz tartalmaz kevesebb megmunkálást, akkor j_b megmunkálásának kezdete essen egybe d-vel,
(iii) különben (i) és (ii) valamelyikét alkalmazzuk.
Az (i) és (ii) esetben csökkent a célfüggvény értéke, a (iii) esetben nem változott. ■

Az SPT sorrend fordítottjának neve LPT sorrend, azaz ***leghosszabb megmunkálási idő.***

9.11. lemma. *Bármely optimális megoldásban a megmunkálások a kezdeti szakaszban LPT, a befejező szakaszban SPT sorrendben vannak.*

Bizonyítás. Ha két szomszédos munkadarab ütemezése fordított, akkor ezeket felcserélve a célfüggvény értéke csökken. ■

9.12. lemma. *Legyen* $(1),(2),\ldots,(n)$ *egy olyan ütemezés, amelyben az első és az utolsó megmunkálás között nincs állásidő és amelyben egy megmunkálás pontosan a határidőkor fejeződik be. Ha a kezdeti szakaszban* α*, a befejező szakaszban* β *megmunkálás van* $(\alpha+\beta=n)$*, akkor a célfüggvény értéke*

$$\sum_{j=1}^{\alpha}(j-1)p_{(j)} \;+\; \sum_{j=1}^{\beta}.(\beta-j+1)p_{(\alpha+j)}. \tag{9.16}$$

Bizonyítás. A kezdeti szakaszban (α) sietése 0, $(\alpha-1)$ sietése $p_{(\alpha)}$, $(\alpha-2)$ sietése $p_{(\alpha-1)}+p_{(\alpha)}$ stb. Tehát $p_{(\alpha)}$ pontosan $\alpha-1$ munkadarab sietésében szerepel, $p_{(\alpha-1)}$ $\alpha-2$ munkadarabében stb. Innen kapjuk az első összeget. Hasonlóan járhatunk el a befejező szakasz esetén. $(\alpha+1)$ késése $p_{(\alpha+1)}$, $(\alpha+2)$ késése $p_{(\alpha+1)}+p_{(\alpha+2)}$ stb. Tehát $p_{(\alpha+1)}$ összesen β munkadarab késésében szerepel, $p_{(\alpha+2)}$ $\beta-1$ munkadarabében stb. Ez adja a második összeget. ■

(9.16)-ból még kiolvasható a következő állítás.

9.13. lemma. *Az előző lemma jelöléseit használva van olyan optimális megoldás, amelyben*

$$|\,\alpha-\beta\,|\leq 1\ .$$

Bizonyítás. Legyen $(1),(2),\ldots,(n)$ egy olyan ütemezés, amelyre a lemma állítása nem igaz. Ha $\alpha\geq\beta+2$, akkor toljuk el az ütemezést úgy, hogy (α) legyen a befejező szakasz első munkadarabja. Ekkor (9.16) első összege $(\alpha-1)p_{(\alpha)}$-val csökkent, a második összege $(\beta+1)p_{(\alpha)}$-val nőtt, vagyis a teljes változás $(\beta+2-\alpha)p_{(\alpha)}\ \leq\ 0$. ■

Innen a következő megoldási módszer adódik. Tegyük fel, hogy az indexek szerinti sorrend egyben az LPT sorrend is. Ekkor az utolsó lemma feltételeinek eleget tevő optimális megoldás esetén (9.16)-ban p_1 szorzója 0, p_2 és p_3 szorzója 1, általában p_{2l} és p_{2l+1} szorzója l. Vagyis szabadon dönthetünk afelől, hogy a $2l$ és a $2l+1$ munkadarab közül melyik legyen elölről az $(l+1)$-edik, illetve hátulról az l-edik.

Gyakorlatok

9.3-1. Konkrét példa megadásával bizonyítsuk be, hogy a 9.4. tételben szereplő feltétel nem szükséges, azaz a határidők szerinti növekvő rendezés nem szükséges feltétele az L_{max} optimalizálásának.

9.3-2. Mutassuk meg, hogy a 9.6. tételben szereplő súlyozott legrövidebb megmunkálási idők sorrendje az adott esetben nem csak elégséges, hanem szükséges feltétel is.

9.4. Ütemezési problémák párhuzamos berendezéseken

Az ide tartozó legegyszerűbb feladatok esetében egy m gépből álló homogén gépcsoportról van szó, azaz minden gép egyforma. Így egy munkadarab megmunkálási ideje minden gépen azonos. A feladatok még ebben az esetben is – kevés kivételtől eltekintve – NP-teljesek, ezért nagy jelentősége van a heurisztikus eljárásoknak. Először a polinomiális eredményeket ismertetjük, és utána tárgyaljuk a heurisztikus módszereket.

9.14. tétel. *A következő ütemezés a $P \mid overlap \mid \sum C_j$ feladat egy optimális megoldását adja. Legyen $[1], [2], \ldots, [n]$ egy SPT sorrend. Ekkor minden gépen pontosan egyforma az ütemezés, és minden gépen a j-ediknek ütemezett megmunkálás a $[j]$ megmunkálás $(1/m)$-ed része.*

Bizonyítás. Tekintsünk egy tetszőleges ütemezést és legyen $(1), (2), \ldots, (n)$ a megmunkálások befejezési sorrendje. Legyen $1 \leq j_1 < j_2 \leq n$. Tegyük fel, hogy (j_1) egy h_1 hosszú megmunkálása az m_1 gépen később kezdődik a t_1 időpontban, mint a (j_2) egy h_2 hosszú megmunkálása az m_2 gépen a t_2 időpontban, azaz $t_1 > t_2$. Az m_1 és m_2 gép nem feltétlenül különböző. Ekkor a két megmunkálás egy $\min\{h_1, h_2\}$ szakasza felcserélhető, és így az összes többi, valamint a (j_2) megmunkálás befejezése nem változik, (j_1) befejezése pedig vagy változatlan, vagy előbbre kerül. ■

A következő két tétel a témakör egyik legkorábbi dolgozatából származik.

9.15. tétel. *A $P \mid pmtn \mid$ f feladatban az átfutási idő legalább*

$$C^* = \max\left\{\frac{1}{m}\sum_{j=1}^{n} p_j, \max\{p_j : j = 1, \ldots, n\}\right\}. \tag{9.17}$$

Bizonyítás. Mivel az átlapolás nem megengedett, ezért egy megmunkálás akárhány részre is legyen felosztva, ezek a részek nem fedhetik át egymást, azaz minden j esetén $p_j \leq C_j$. Másfelől a teljes átfutási idő nem lehet rövidebb annál, mint amit úgy kapunk, hogy a megmunkálásokat egyenletesen osztjuk szét a gépek között. ■

9.16. tétel. *A $P \mid pmtn \mid C_{max}$ feladat optimális célfüggvényértéke a (9.17) képletben adott mennyiség.*

Bizonyítás. Egyenként készítünk ütemezést a gépekre. Minden gépet a C^* időpontig terhelünk le munkával, hacsak el nem fogytak a megmunkálások. A megmunkálásokat tetszőleges sorrendben vesszük. Ha a soron következő megmunkálás még befér az éppen ütemezendő gép C^* időkorlátjába, akkor egyetlen egységben ezen a gépen végezzük el a megmunkálást az erre a gépre már ütemezett megmunkálások után közvetlenül. Ha a teljes megmunkálás nem fér be az időkorlátba, akkor a megmunkálás akkora részét ütemezzük erre a gépre, hogy ezzel az időkorlátot elérjük, a maradék rész pedig a következő gép első megmunkálása. Az időkorlát megválasztása miatt ez a két rész nem nyúlhat egymásba. ■

Vegyük észre, hogy az így készített ütemezésben a megszakítások száma legfeljebb $m-1$, és minden megmunkálást legfeljebb egyszer szakítunk meg. A minimálisan szükséges megszakítások számának meghatározása már NP-teljes probléma. Ha egy megmunkálást megszakítottunk, és egy része a következő gépre került, akkor a tényleges termelés során a megmunkálás ezzel az átvitt résszel kezdődik.

A problémák úgy is felfoghatók ebben a körben, mint két részfeladat együttese, először meg kell találni a megmunkálások gépek közötti optimális felosztását, majd az egyes gépeken a legjobb sorrendet. Innen az egy gép esetére mondottak alapján azonnal adódik az alábbi tétel.

9.17. tétel. *Az $R \parallel \sum C_j$, illetve az $R \parallel \sum w_j C_j$ feladat optimális megoldásában a megmunkálások minden gépen SPT, illetve SWPT sorrendben vannak.*

A következő tétel bizonyításában $O(n^3)$ művelet segítségével vezetjük vissza az ütemezési feladatot a polinomiális hozzárendelési feladatra.

9.18. tétel. *Az $R \parallel \sum C_j$ feladat polinomiális idő alatt megoldható.*

Bizonyítás. Tekintsünk egyetlen m_l gépet. Tegyük fel, hogy ezen n_l ($n_l \le n$) megmunkálást akarunk végezni, az (1), (2), ..., (n_l) sorrendben. Ekkor a célfüggvény értéke ezen a gépen

$$\sum_{j=1}^{n_l} (n_l - j + 1) p_{l(j)} \ , \tag{9.18}$$

ahogy ezt például a 9.12. lemmában is láttuk. Vezessük be a következő bináris változókat:

$$x_{(ik)j} = \begin{cases} 1, & \text{ha a } j\text{-edik megmunkálást az } i\text{-edik gépen végezzük,} \\ & \text{sorrendben hátulról a } k\text{-adiknak} \ , \\ 0 & \text{különben} \ . \end{cases} \tag{9.19}$$

Itt k értéke 1 és n közötti egész. Két további feltételcsoport adódik még:
(i) minden megmunkálást el kell végezni,
(ii) minden gép egyszerre csak egy megmunkálással foglalkozhat.

Az elsőt az alábbi egyenletekkel, a másodikat pedig az azokat követő egyenlőtlenségekkel írhatjuk elő:

$$\sum_{i=1}^{m} \sum_{k=1}^{n} x_{(ik)j} = 1, \quad j = 1, \ldots, n \ , \tag{9.20}$$

$$\sum_{j=1}^{n} x_{(ik)j} \le 1, \quad i = 1, \ldots, m, \ \ k = 1, \ldots, n \ . \tag{9.21}$$

Ha $x_{(ik)j} = 1$, akkor a j megmunkálás költsége a célfüggvényben $k p_{ij}$, ezért a teljes költség

$$\sum_{i=1}^{m} \sum_{k=1}^{n} \sum_{j=1}^{n} k p_{ij} x_{(ik)j} \ , \tag{9.22}$$

amit minimalizálni kell. Az így kapott (9.19)–(9.22) feladat egy hozzárendelési probléma, ami polinomiális idő alatt megoldható. ■

Érdemes szemügyre venni a bizonyításban szereplő transzformáció „költségeit”. Az eredeti feladat mérete $m \times n$ volt. A jelenlegié pedig $(mn) \times n$ (az átfogalmazott feladatban az i,k indexpár egyetlen index szerepét játssza). A hozzárendelési feladat megoldásához szükséges műveletek száma $O(n^3)$ marad.

Reguláris célfüggvények mellett bizonyos esetekben a dinamikus programozás természetes módon adódik, mint egy pontos megoldási módszer. A 9.17. tétel azt mondja, hogy az $R \parallel \sum C_j$ és a $R \parallel \sum w_j C_j$ feladatok esetében, ha megtörtént a megmunkálások felosztása az egyes gépek között, akkor az egyes gépeken a sorrend kötött. Hasonló eredmény a következő.

9.19. tétel. *Az $R \parallel \sum T_j$ feladatnak van olyan optimális megoldása, amelyben a megmunkálások minden gépen a határidők szerinti növekvő sorrendben vannak.*

Bizonyítás. Ha a feltétel nem teljesül, akkor van legalább egy gép, amelyen az ütemezés szerint egy későbbi határidejű megmunkálás közvetlenül megelőz egy korábbi határidejű megmunkálást. E két megmunkálást felcserélve a célfüggvény értéke nem növekszik. ■

Hasonló állítás mondható az $U_{\max}$ célfüggvényről, azaz a megengedett ütemezés kereséséről (lásd a 9.4. tételt). Fontos megjegyezni, hogy ebben lényeges szerepet játszik, hogy minden rendelkezésre állási idő azonos. A $C_{\max}$ célfüggvény esetén nemcsak egy, hanem bármely sorrend rendelkezik a tételbeli tulajdonsággal, mert a megmunkálásoknak gépekre való felosztása már egyértelműen meghatározza a teljes átfutási időt. Dinamikus programozással olyan feladatokat lehet viszonylag könnyen megoldani, amelyekre a következő feltétel teljesül:

A megmunkálások indexsorrendje olyan sorrend,
hogy van olyan optimális megoldás, hogy minden (*S*)
gépen a megmunkálások ebben a sorrendben vannak.

Nem jelenti az általánosság megszorítását, hogy az indexsorrendről feltételeztük, hogy a kívánt tulajdonságú.

Vizsgálni fogjuk mind az $f_{\max}$, mind a $\sum f_j$ típusú célfüggvényt. A tárgyalás nagyon hasonló, ezért az alábbi közös jelölést használjuk: $F_j(t_1,\dots,t_m)$ az optimális célfüggvényérték, ha csak az indexsorrendben első j megmunkálást ütemezzük, és a gépek állásidő nélkül dolgozva munkájukat a $t_1,\dots,t_m$ időpontokban fejezik be.

9.20. tétel. *Ha az (S) feltétel teljesül, akkor az $R \parallel \sum f_j$ feladat esetén*

$$F_j(t_1,\dots,t_m) \;=\; \min\{f_j(t_i) + F_{j-1}(t_1,\dots,t_i - p_{ij},\dots,t_m) \;:\; i = 1,\dots,m\}\,. \qquad (9.23)$$

Bizonyítás. Most utoljára a j megmunkálást ütemezzük, és ez az (S) feltétel miatt valamelyik gépen utolsónak fejeződik be az eddig ütemezettek közül. Ha ez történetesen az i gép, akkor ott a j-t közvetlenül megelőző megmunkálás a $t_i - p_{ij}$ időpontban fejeződik be. Így felmerül az előző $j-1$ megmunkáláson $F_{j-1}(t_1,\dots,t_i - p_{ij},\dots,t_m)$ költség, továbbá $f_j(t_i)$ költség a j megmunkáláson. ■

9.21. tétel. *Ha az (S) feltétel teljesül, akkor az $R \,||\, f_{\max}$ feladat esetén*

$$F_j(t_1,\ldots,t_m) = \min\{\max\{f_j(t_i), F_{j-1}(t_1,\ldots,t_i - p_{ij},\ldots,t_m)\} : \; i = 1,\ldots,m\} \; . \qquad (9.24)$$

Bizonyítás. A bizonyítás hasonló az előző tételéhez. ■

A számítási idő $\Theta(mnC^m)$, ahol C a teljes átfutási idő felső korlátja. Jobban korlátozza a módszer alkalmazását, hogy a szükséges tárolási kapacitás $\Theta(mC^m)$. Amennyiben az F függvényt tartalmazó kitöltendő táblákat explicit módon tároljuk, akkor sem a számítás mennyisége, sem a szükséges memória nagyságrendje nem csökkenthető.

A heurisztikus módszerekkel legtöbbet a $P \,||\, C_{\max}$ feladatot vizsgálták, mert ütemezési szempontból egyszerű, de NP-teljes. ***Listás ütemezésnek*** nevezik azt a heurisztikus eljárást, amely a következő két lépésből áll:

1. Meghatározzuk a megmunkálások valamely sorrendjét (lista).
2. Az adott sorrendben véve a megmunkálásokat, az éppen soron következőt az elsőnek megüresedő gépre rakjuk.

Ha L jelöli az adott listát, akkor $C(L)$ a heurisztikus megoldás célfüggvényértéke, míg az optimális megoldásé C^*, és a $C(L)/C^*$ hányadost vizsgáljuk.

9.22. tétel. *Tetszőleges L lista esetén*

$$\frac{C(L)}{C^*} \le 2 - \frac{1}{m} \; . \qquad (9.25)$$

Bizonyítás. Ha sikerül teljesen egyenletesen szétosztani a gépek között a megmunkálásokat, akkor az átfutási időre

$$\frac{1}{m}\sum_{j=1}^{n} p_j$$

adódik. Másfelől legalább egy gépen legalább annyi időt fel kell használnunk, mint a leghosszabb megmunkálási idő, hiszen az átlapolás nem megengedett, azaz

$$\max\{p_j \; : \; j = 1,\ldots,n\}$$

egy másik alsó korlát C^*-ra. Legyen k az utolsónak befejeződő megmunkálás. Ez a $t = C(L) - p_k$ időpontban kezdődött el. Mivel az ütemezésben nincsenek felesleges állásidők, így a t időpontig valamennyi gép folyamatosan dolgozott. Ezért

$$t \; \le \; \frac{1}{m}\sum_{j\ne k} p_j \; .$$

Végül a

$$C(L) \; = \; t \; + \; p_k \; \le \; \frac{1}{m}\sum_{j\ne k} p_j \; + p_k = \frac{1}{m}\sum_{j=1}^{n} p_j \; + \; \frac{m-1}{m} p_k \le \left(2 - \frac{1}{m}\right) C^* \qquad (9.26)$$

egyenlőtlenséget kapjuk. ■

A korlát éles. Erre vonatkozóan lásd a 9-5. feladatot.

Számos heurisztikus eljárás pontosságára ismert a fentinél jobb felső korlát.

9.23. tétel.

$$\frac{C(LPT)}{C^*} \leq \frac{4}{3} - \frac{1}{3m} \,. \tag{9.27}$$

Bizonyítás. Indirekt módon feltesszük, hogy a tétel állítása nem igaz. Feltehető, hogy $m \geq 2$. Az első részállítás, amit bebizonyítunk, hogy feltehető az is, hogy az LPT sorrend szerinti ütemezésben utolsónak az utolsó, azaz a legrövidebb megmunkálási idejű megmunkálás fejeződik be. Tekintsünk valamely rögzített m mellett egy olyan F ütemezési feladatot, amelyre (9.27) nem igaz, és n értéke minimális. Ha az ütemezés szerint egy $p < n$ indexű megmunkálás fejeződik be utolsónak, akkor tekintsük azt az F' ütemezési feladatot, amelyet F-ből úgy kapunk, hogy az LPT sorrendből csak az első p megmunkálást tartalmazza. Itt az LPT sorrend szerinti átfutási idő nem változott, míg az optimális átfutási idő nem nőtt. Ezért a tétel állítása erre az F' feladatra sem lehet igaz, ami ellentmond n minimalitásának.

Innen (9.26) alapján kapjuk, hogy erre a feladatra

$$\frac{4}{3} - \frac{1}{3m} < \frac{C(LPT)}{C^*} \leq \frac{1}{mC^*} \sum_{j=1}^{n} p_j + \frac{(m-1)p_n}{mC^*} \leq 1 + \frac{(m-1)p_n}{mC^*} \,,$$

ahonnan adódik, hogy

$$p_n > \frac{C^*}{3} \,.$$

Ez természetesen az LPT sorrend miatt valamennyi megmunkálási időre vonatkozik, azaz az optimális megoldásban legfeljebb két megmunkálás lehet egy gépen.

Tekintsünk egy tetszőleges olyan U ütemezést, ahol minden gépen legfeljebb két megmunkálás van. Megadunk néhány olyan átrendezést, amely nem növeli meg az átfutási időt. Legyen két – az i gépre ütemezett – megmunkálás ideje t_{i1} és t_{i2}, míg egy másik j gép esetén t_{j1} és t_{j2}, illetve ha ezen csak egy megmunkálás van, akkor t_j.

(i) Ha $t_{i1} > t_{j1}$ és $t_{i2} > t_{j2}$, akkor i_1 és j_1 felcserélésével a két gépen az átfutási idő csökken, hiszen $\max\{t_{i1} + t_{j2}, t_{j1} + t_{i2}\} < t_{i1} + t_{i2}$.

(ii) Ha $t_{i1} > t_j$, akkor az i_2 munkát átrakva a j gépre a két gépen szintén csökken az átfutási idő.

(iii) Az i_1 és i_2 munkák sorrendjének felcserélése nem változtatja meg az i gépen az átfutási időt.

Bármely lehetséges U ütemezésből kiindulva és elvégezve az összes lehetséges fenti átalakítást, egy olyan V ütemezést nyerünk, amelyben az átfutási idő nem hosszabb, mint U-ban, és rendelkezik a következő tulajdonságokkal:

(a) ha bármely i, j gép esetén mindkét gépen két megmunkálás van, akkor (i) alapján

$$t_{i1} > t_{j1} \quad \text{esetén} \quad t_{i2} \leq t_{j2} \,,$$

(b) minden i, j gép esetén, ha az i gépen két megmunkálás van, a j gépen egy, akkor (ii) alapján

$$t_j \geq t_{i1}, t_{i2} ,$$

(c) minden i gép esetén, ha az i gépen két megmunkálás van, akkor (iii) alapján

$$t_{i1} \geq t_{i2} ,$$

(d) mivel a gépek sorrendje nem befolyásolja az átfutási időt, ezért feltehető, hogy a gépek indexsorrendje szerint csökken az első megmunkálási idő,
(e) az egy megmunkálást végző gépek indexei (b) és (d) alapján kisebbek a két megmunkálást végzőkénél,
(f) a két megmunkálást végző gépek indexsorrendje szerint nő a második – azaz a nem nagyobb – megmunkálási idő.

Tehát van olyan optimális megoldás, ami az (a)–(f) tulajdonságokkal rendelkezik, ezek a tulajdonságok azonban egyértelműen meghatározzák az ütemezést, ami nem más, mint az LPT lista szerinti ütemezés. Ez pedig ellentmond a

$$\frac{C(LPT)}{C^*} > \frac{4}{3} - \frac{1}{3m} \geq 1 .$$

egyenlőtlenségnek. ■

A $P \parallel C_{\max}$ problémához nagyon hasonló, mintegy annak duálja a ládapakolási probléma. Egyforma kapacitású ládákba kívánunk belerakni tárgyakat, ahol a tárgyak méretét egyetlen számmal lehet jellemezni. Az egy ládába rakott tárgyak összmérete a méretek összege. Meghatározandó a minimálisan szükséges ládák száma. Visszatérve a párhuzamos berendezésekhez, tegyük fel, hogy a munkákat egy meghatározott idő, például egy műszak alatt akarjuk elvégezni. Ekkor úgy merül fel a kérdés, hogy hány gépre van szükség.

FF(n, **p**, K)

```
1  for i ← 1 to n
2      do z_i ← K
3  m ← 1
4  for i ← 1 to n
5      j ← 1
6      while z_j < p_i
7            do j ← j + 1
8      z_j ← z_j − p_i
9      if j > m
10       then m ← j
11 return m
```

A ládapakolási feladat megoldására egy mohó heurisztikus eljárás a FF. Legyen K a ládák kapacitása. Feltesszük, hogy nincs olyan tárgy, aminek mérete nagyobb lenne, mint K, hiszen ekkor nincs megengedett megoldás. Ha ez a feltétel teljesül, és a tárgyak száma n, akkor legfeljebb n ládát kell használnunk. A FF algoritmus sorra veszi a tárgyakat, és mindegyiket az első (azaz a legkisebb indexű) olyan ládába rakja, ahova belefér. A következő

pszeudokódban a bemenő adat a tárgyak n száma, a tárgyak méretét tartalmazó **p** vektor, valamint a ládák K kapacitása, kimenő adat a felhasznált ládák m száma. A kódban z_i az i-edik láda még fel nem használt kapacitása.

A FF és más ládapakolási algoritmusok elemzése megtalálható a 11.4. alfejezetben.

A FF algoritmust akkor tudjuk felhasználni, ha tudunk felső korlátot adni a maximálisan szükséges kapacitásra, azaz az átfutási időre.

9.24. tétel. *Bármely $P \parallel C_{\max}$ feladat esetén az optimális C^* átfutási időre teljesül, hogy*

$$C^* \le \max\left\{\frac{2}{m}\sum_{j=1}^{n} p_j, \max\{p_j : j = 1,\dots,n\}\right\} . \tag{9.28}$$

Bizonyítás. Legyen C a (9.28) egyenlőtlenség jobboldalán szereplő mennyiség. Indirekt módon tegyük fel, hogy a tétel állítása nem igaz. Alkalmazzuk a FF algoritmust úgy, hogy a ládák kapacitása C és a megmunkálások a megmunkálási idők szerinti, azaz a ládapakolási feladat nyelvén a tárgyak mérete szerinti csökkenő sorrendben vannak. Ha a tétel állítása hamis, akkor az algoritmus kénytelen valamely k tárgyat az $(m+1)$-edik ládába rakni. Nyilvánvalóan $k \ge m+1$. Mivel $p_k \le p_{k-1} \le \cdots \le p_1$, ezért minden láda legalább p_k-ig fel van töltve. Mivel azonban p_k nem fér az első m ládába, ezért ezeket már jobban feltöltöttük, mint $C/2$. Innen adódik, hogy

$$\sum_{j=1}^{n} p_j \;>\; \frac{mC}{2} \;\ge \sum_{j=1}^{n} p_j ,$$

ami ellentmondás. ■

A bizonyításból érezni lehet, hogy a FF algoritmust úgy lesz célszerű alkalmazni, ha előtte a megmunkálási időket csökkenő sorrendbe rakjuk, azaz az LPT sorrendet használjuk. Tegyük fel, hogy megbecsüljük a szükséges átfutási időt, és erre alkalmazzuk a FF algoritmust. Természetesen egy jó megengedett megoldáshoz jutunk, ha az eljárás talál ilyet, azaz nem használ több ládát (gépet), mint amennyit felhasználhat. De ha nem talál megengedett megoldást, akkor lényegében nincs semmi a kezünkben, csak az a durva közelítő megoldás, ami az előző tétel bizonyításából származik. Viszont a 9.15. tételből ismerünk egy alsó korlátot az átfutási időre. Az alsó és felső korlátból kiindulva, logaritmikus kereséssel egy pontosabb korlátot lehet kapni. Ez az FF ismételt alkalmazását jelenti, ahol csökkentjük a felső korlátot, ha találtunk megengedett megoldást, és növeljük az alsó korlátot, ha nem. Ne feledjük, hogy a 9.24. tétel bizonyítása mindenképpen ad egy megengedett megoldást, a most vázolt algoritmus ezt kívánja javítani.

A fentiekben felvázolt algoritmus leírásánál az alábbi jelöléseket alkalmazzuk:

- K_a az aktuális alsó korlát,
- K_f az aktuális felső korlát,
- K a pillanatnyi (azaz a kipróbálás alatt álló) korlát,
- s a megteendő iterációs lépések száma.

Továbbá részeljárásként működik némi módosítással a Монó algoritmus. Mivel a párhuzamos berendezések esetén nem az a kérdés, hogy hány berendezést kell használnunk,

ha K időn belül végezni akarunk, hanem az, hogy K időn belül m géppel végezni tudunk-e, ezért a FF algoritmus módosítását megvalósító MF algoritmus mind a K kapacitáskorlátot, mind a gépek m számát paraméterként kapja meg.

Az algoritmus eredeti angol neve „Multi-Fit”, amit rövidítve megtartunk (magyar jelentése: többszörös beillesztés).

MF(m, K)

```
1  K_a ← max{(1/m) Σ_{j=1}^{n} p_j, max{p_j : j = 1,...,n}}
2  K_f ← max{(2/m) Σ_{j=1}^{n} p_j, max{p_j : j = 1,...,n}}
3  rendezzük a megmunkálásokat a megmunkálási idők szerinti csökkenő sorrendbe
4  for i ← 1 to s
5      do K ← (K_a + K_f)/2
6         Mohó(K, m, elég)
7      if elég = IGAZ
8        then K_f ← K
9        else K_a ← K
10 return K_a és K_f
```

Annak eldöntésére, hogy az adott esetben m darab K kapacitású láda elegendő-e, a Mohó eljárást használjuk. Ennek eredménye az *elég* logikai változó, melynek értéke akkor IGAZ, ha Mohó talált megengedett megoldást.

Mohó(K, m)

```
1  elég ← IGAZ
2  for i ← 1 to m
4      do z_i ← K
5  for i ← 1 to n
6      j ← 1
7      do while z_j < p_i
8                do j ← j + 1
9         if j > m
10          then elég ← HAMIS
11           return elég
12        z_j ← z_j − p_i
```

n tárgy méretének összehasonlításos rendezéséhez $O(n \log n)$ lépés elegendő. Az FF algoritmus beépített változata minden tárgyra legfeljebb m lépést tesz meg, ez $O(nm)$ számítási igény minden korlátra. Tehát az algoritmus műveleti igénye $O(n \log n + nms)$. A számítási igény ennyi is lesz abban az esetben, ha az első $m - 1$ tárgy olyan nagy, hogy egyenként kitöltenek egyet-egyet az első $m - 1$ gép közül úgy, hogy más megmunkálás már nem fér melléjük.

9.5. példa. Tekintsük azt a feladatot, amelyben $n = 5$, $m = 2$ és a megmunkálási idők rendre 7, 7, 5, 5, 5. Ekkor a listás ütemezés a két leghosszabb megmunkálást külön gépre rakja, majd mindkét gépre tesz egy-egy 5 idejűt, végül az utolsó megmunkálást kénytelen az első gépre rakni. Az átfutási idő ekkor 17. Az optimális megoldás viszont az, amikor az azonos megmunkálási idejű munkák ugyanazon

a gépen vannak, mert így az átfutási idő csak 15. A 9.23. tételben megadott felső korlát most 7/6. Ezzel megszorozva az optimális megoldás értékét, vagyis 15-öt, 17.5-et kapunk, vagyis az eljárás lényegében a hibahatáron dolgozik.

Alkalmazzuk most ugyanerre a feladatra az MF algoritmust, $m = 2$ mellett.

A kezdeti alsó korlát 14.5, amit 15-re lehet felkerekíteni.

A kezdeti felső korlát 29, innen az induló korlát 22-nek adódik.

Ekkor az első gépen a megmunkálási idők rendre 7, 7, 5, míg a második gépen 5, 5. Mivel találtunk megoldást, ezért az alsó és a pillanatnyi felső korlát átlagát kell venni, ami kerekítéssel 18. Erre már az optimális megoldást kapjuk.

9.6. példa. Legyen most $n = 6$, $m = 2$ és a megmunkálási idők 10, 7, 7, 6, 6, 4. A listás ütemezés az első gépre a 10, 6, 4 megmunkálási idejű munkákat rakja, míg a másodikra a 7, 7, 6 idejűket. Mivel mindkét gépen 20 az átfutási idő, ezért ez az optimális megoldás. Ha az MF algoritmust futtatjuk, akkor ott a kezdeti alsó korlát éppen ez a 20 lesz. Azonban 20 vagy annál nagyobb korlát esetén az eljárás mindig összerakja a 10 megmunkálási idejű munkát legalább az egyik 7 idejűvel, így sohasem fogja megtalálni az optimális megoldást. A két kezdeti korlát 20 és 40. Ezért az eljárás a 30 korláttal indul. Ehhez van megoldás: első gép 10, 7, 7, 6, a második gép 6, 4. Így a következő korlát 25. Ekkor a 10, 7, 7, illetve 6, 6, 4 megoldást kapjuk. Innen a következő korlát 22. Az ehhez tartozó megoldás 10, 7, 4 és 7, 6, 6. Itt az átfutási idő csak 21. Ezért a megoldás nem változik a következő korlátra sem, ami éppen ennyi. Innen már a 20 korlát következik, de mint azt említettük, ehhez nincs megoldás.

Ahhoz, hogy az algoritmus pontosságát elemezni tudjuk, vissza kell térni a ládapakolási feladathoz.

Ha a ládák méretét minden határon túl növeljük, akkor egyre kevesebb ládára lesz szükség, egészen addig, amíg a ládák mérete el nem éri a tárgyak méreteinek összegét, mert ettől kezdve csak egy ládát kell használnunk. Az ütemezési probléma megoldása szempontjából bennünket rögzített számú láda felhasználása érdekel. Vegyük észre, hogy a 9.15. tételben megadott alsó korlát és a 9.24. tételbeli felső korlát között csak egy legfeljebb 2-es tényező van. Ez az alábbi állítást sugallja: ha megnöveljük a ládák méretét a minimálisan szükségesnek legfeljebb a kétszeresére, akkor a FF algoritmus talál megengedett megoldást, feltéve, hogy a tárgyak méret szerint csökkenően rendezettek. Az alábbiakban azt mutatjuk be, hogy ez az állítás lényegében igaz. Ezen felül az is teljesül, hogy az említett 2 korlátnál, ami rosszabb volna, mint a már elért 4/3 korlát, sokkal jobb az eljárás.

Legyen $\mathbf{p}$ a tárgyak méreteit tartalmazó vektor, továbbá $C_m^*(\mathbf{p})$ a ládák azon legkisebb mérete, amely mellett még van olyan pakolás, amely legfeljebb m ládát használ fel. Ne feledjük, hogy véges sok tárgyat csak véges sok féleképpen lehet szétosztani a ládák közt, ezért $C_m^*(\mathbf{p})$ nem csak infimumként, hanem minimumként is létezik. Mivel itt heurisztikus eljárásról van szó, így egyáltalán nem biztos, hogy a FF algoritmus bármely $\mathbf{p}$ és m esetén talál m ládát használó megoldást a $C_m^*(\mathbf{p})$ korláthoz. Felmerül a kérdés, hogy hányszorosra kell megnövelni a ládák méretét $C_m^*(\mathbf{p})$-hez képest, hogy az algoritmus már találjon ilyent. Legyen $\mathrm{FF}(n, \mathbf{p}, K)$ az FF algoritmus megoldása által használt ládák száma. Jelölje r azt a tényezőt, ahányszorosára a ládák méretét növeltük. Legyen r_m az a legkisebb tényező, amely bármely feladat esetén biztosítja, hogy algoritmusunk legfeljebb m ládát használ, azaz

$$r_m = \inf\{r : \text{minden } \mathbf{p} \text{ esetén } \mathrm{FF}(n, \mathbf{p}, rC_m^*(\mathbf{p})) \le m\} .$$

9.25. tétel. *Bármely* $\mathbf{p}$ *és* $r \ge r_m$ *esetén* $\mathrm{FF}(n, \mathbf{p}, rC_m^*(\mathbf{p})) \le m$.

Bizonyítás. Legyen először $r = r_m$. Ha most a tétel állítása nem igaz, akkor van egy olyan $\mathbf{p}$ méretvektor, hogy $\mathrm{FF}(n, \mathbf{p}, r_m C_m^*(\mathbf{p})) > m$. Ha egy tárgyat nem tudunk betenni egy ládába, annak az az oka, hogy a ládába tett tárgyak összmérete ezen tárggyal együtt meghaladná az $r_m C_m^*(\mathbf{p})$ korlátot. Tekintsük ezeket, az algoritmus végrehajtása során fellépő, az adott korlátnál nagyobb mennyiségeket. Legyen ezek közül a minimális $r_m C_m^*(\mathbf{p}) + \varepsilon$. Mivel csak véges sok mennyiségről van szó, ezért $\varepsilon > 0$. Legyen C olyan korlát, hogy $r_m C_m^*(\mathbf{p}) < C < r_m C_m^*(\mathbf{p}) + \varepsilon$. Ekkor C megválasztásából következik, hogy $\mathrm{FF}(n, \mathbf{p}, C) > m$, ami $r = C/C_m^*(\mathbf{p}) > r_m$ mellett ellentmond r_m megválasztásának.

Legyen most már $r > r_m$. Tegyük fel, hogy van olyan $\mathbf{p}$ méretvektor, hogy $\mathrm{FF}(n, \mathbf{p}, rC_m^*(\mathbf{p})) > m$. Tekintsük azt a $\mathbf{q}$ méretvektort, amelyben a ládák kapacitását $(r/r_m)C_m^*(\mathbf{p})$-ra növeltük, továbbá véges sok, a $\mathbf{p}$ vektor legkisebb eleménél kisebb tárgyat adunk a $\mathbf{p}$ tárgyaihoz úgy, hogy ezekkel a kis tárgyakkal kiegészíthető legyen $\mathbf{p}$ optimális megoldása úgy, hogy a megnagyobbított ládák pontosan fel legyenek töltve. Ez azt jelenti, hogy $C_m^*(Q) = (r/r_m)C_m^*(\mathbf{p})$. Ekkor azonban $r_m C_m^*(Q) = rC_m^*(\mathbf{p})$. Másfelől viszont feltettük, hogy $\mathrm{FF}(n, \mathbf{p}, rC_m^*(\mathbf{p})) = \mathrm{FF}(n, \mathbf{p}, r_m C_m^*(Q)) > m$, ami ellentmond a bizonyítás első felében r_m-re igazoltaknak. ■

Ebből a tételből már azonnal következik annyi, hogy a MF algoritmus legalább a logaritmikus keresés folyamán fellépő legkisebb olyan korlátra fog megengedett megoldást találni, amely korlát nem esik $r_m C_m^*(P)$ alá.

9.26. tétel. *Bármely* $\mathbf{p}$ *méretvektor és* $k \in Z_+$ *egész esetén az* MF *algoritmus által szolgáltatott megoldás* C *értékére igaz, hogy*

$$\frac{C}{C_m^*(\mathbf{p})} \leq r_m + 2^{-k} . \tag{9.29}$$

Bizonyítás. Legyen az algoritmus kezdeti alsó és felső, valamint a befejező alsó és felső korlátja, illetve eredménye rendre A, F, BA, BF és C. Itt C a talált megengedett megoldás átfutási ideje, amire nyilvánvaló, hogy $BF \geq C$. Ha az állítás nem igaz, akkor van olyan $\mathbf{q}$ méretvektor, ahol rosszabb, azaz nagyobb értéket kapunk a (9.29) képletben megadott korlátnál. Mivel azonban az algoritmus által adott korlát legfeljebb C, adódik, hogy

$$BF \geq C > (r_m + 2^{-k})C_m^*(\mathbf{p}). \tag{9.30}$$

A logaritmikus keresés tulajdonságai alapján $A \leq C_m^*(\mathbf{p}) \leq F \leq 2A$, ezért

$$BF - BA = 2^{-k}(F - A) \leq 2^{-k} C_m^*(\mathbf{p}) .$$

Innen (9.30) alapján adódik, hogy

$$BA > r_m C_m^*(P) .$$

Ezért az eljárás részeként szereplő FF algoritmust alkalmazni kellett a BA korlát mellett is, és ekkor FF az előző tétel állításával szemben több, mint m gépet használt. ■

Bizonyítás nélkül említjük meg a következő tételt.

9.27. tétel. *Minden m esetén $r_m \leq 1.22$.*

Gyakorlatok

9.4-1. Bizonyítsuk be, hogy a 9.25–9.27. tételekben szereplő r_m mennyiség minden $m \geq 2$ esetén nagyobb, mint 1.

9.4-2. Oldjuk meg azt a $P2||C_{\max}$ feladatot, melyben a megmunkálási idők: 7, 7, 6, 6, 5, 5, 4, 4, 4.

9.5. Az egyutas ütemezési probléma

Bár az egyutas ütemezési problémát nagyon sokat vizsgálták az irodalomban, a gyakorlatban viszonylag ritkán fordul elő, ezért csak a leglényegesebb eredmények ismertetésére szorítkozunk.

Csak az $F \parallel C_{\max}$ feladatot tárgyaljuk. Tehát minden gépből pontosan egy darab van, a gépek nem helyettesíthetik egymást, és minden munkadarab ugyanazon sorrendben halad végig a gépeken. Az általánosság megszorítása nélkül feltesszük, hogy ez a sorrend a gépek indexsorrendje. Erőforrások és megelőzési feltételek nem korlátozzák az ütemezést. A feladatnak még így is két esete van, az előzéses és az előzés nélküli. Az utóbbi azt jelenti, hogy a munkadarabok minden gépre ugyanolyan sorrendben kerülnek fel, míg az előző esetben a munkadarabok sorrendje a gépek között változhat. Az előzés nélküli esetet fogjuk részletesen tárgyalni, de első célunk az, hogy megmutassuk, hogy 2 és 3 gép esetén a két feladat azonos.

Minden további tárgyaláshoz szükségünk lesz a következő jelölésre. Legyen a egy tetszőleges munkadarab és k $(1 \leq k \leq m)$ egy tetszőleges gép. Ekkor valamely rögzített ütemezés mellett $F(a,k)$ az a munkadarab befejezési ideje a k gépen.

9.28. tétel. *Ha $m > 1$, akkor az előzéses $F \parallel C_{\max}$ problémának van olyan optimális megoldása, ahol az utolsó két gépen a megmunkálások sorrendje azonos.*

Bizonyítás. Tekintsünk egy tetszőleges optimális ütemezést, amelyben az utolsó előtti gépen $(1), (2), \ldots, (n)$, az utolsó gépen $[1], [2], \ldots, [n]$ a munkadarabok sorrendje. Először azt mutatjuk meg, hogy az átfutási idő megnövelése nélkül megváltoztatható úgy a sorrend az utolsó gépen, hogy a két gépen az utolsó munkadarab azonos legyen. Legyen L azon munkadarabok halmaza, melyek az utolsó gép utolsó összefüggő termelési szakaszában vannak, vagyis az utolsó állásidő után. Ebben a szakaszban minden kijelölt megmunkálás azonnal megkezdődik, mihelyst az előtte lévő a gépen befejeződött. $[n]$ mindenképpen eleme L-nek. Ha L nem tartalmaz mást, akkor ez azt jelenti, hogy minden más megmunkálás az m gépen már azelőtt befejeződött, hogy az $m-1$ gép $[n]$ megmunkálásával végzett volna, ezért szükségképpen $(n) = [n]$. Vizsgáljuk azt az esetet, amikor L több elemet is tartalmaz. Legyen $[n] = (k)$. Ha $k = n$, akkor készen vagyunk. Különben rendezzük át az utolsó gépen a sorrendet úgy, hogy az $[n] = (k)$ munkadarab megmunkálását minden (l) munkadarab megmunkálása elé visszük, ahol $k < l \leq n$. Mivel (l) csak az $F((l), m-1) > F([n], m-1)$ időpontban vagy az után kerülhet az m gépre, ezért az $[n]$ munkadarab legfeljebb az $F([n], m-1)$ időpontig kerül előbbre. Így azonban nem keletkezhet újabb állásidő.

Ha ezen átrendezés után a két gépen nem lenne azonos az utolsó munkadarab, akkor

ismételten alkalmazható egy hasonló átrendezés, mindaddig, amíg a kívánt állapotot el nem értük. Ekkor a gondolatmenet megismételhető az utolsó előtti munkadarabokra, és így tovább. ■

Az alábbiakban a következő jelöléseket fogjuk használni, tekintettel arra, hogy először az előzéses feladatot vizsgáljuk. A j gépen a munkadarabok sorrendje $[1]_j, [2]_j, \ldots, [n]_j$. A szokásos feltételezések miatt az ütemezést egyértelműen meghatározza, ha a megmunkálások sorrendje minden gépen adott. Ezért az ütemezés azonosítható a fenti m sorrend együttesével. Továbbá ha a egy tetszőleges munkadarab, akkor $j(a)$ az a munkadarab pozíciója a j gépen, azaz $[j(a)]_j = a$.

9.29. tétel. *Minden i, j $(1 \le i \le n, \quad 1 \le j \le m)$ esetén*

$$F([i]_j, j) \;=\; \max\{F([i]_j, j-1),\; F([i-1]_j, j)\} \;+\; p_{[i]_j j}\;. \tag{9.31}$$

Bizonyítás. Az $[i]_j$ munkadarab megmunkálása a j gépen nem kezdődhet előbb, mint ahogy az $[i]_j$ munkadarab megmunkálása a $j-1$ gépen befejeződik, és mint ahogy az előző megmunkálás, azaz $[i-1]_j$-é, a j gépen befejeződik. Ekkor azonban azonnal meg is kezdődik és $p_{[i]_j j}$ ideig tart. ■

9.30. tétel.

$$F([n]_m, m) = \tag{9.32}$$

$$= \max\left\{ \sum_{j=1}^{m} \sum_{\alpha=j([i_{j-1}]_{j-1})}^{i_j} p_{[\alpha]_j j} :\; \textit{minden } j \textit{ esetén } j([i_{j-1}]_{j-1}) \le i_j;\; 1 = 1([i_0]);\; i_m = n \right\}.$$

Bizonyítás. A 9.29. tételből következik, hogy az $F([n]_m, m)$ teljes átfutási idő vagy $F([n]_m, m-1) + p_{[n]_m m}$ vagy $F([n-1]_m, m) + p_{[n]_m m}$. Tehát az első esetben egy géppel, a második esetben egy munkadarabbal lépünk hátulról visszafelé. Akármelyik kifejezés legyen is a teljes átfutási idő, arra mindaddig alkalmazható a (9.31) rekurzív képlet, míg egy olyan összeget nem kapunk, ami a (9.32) képletben is szerepel.

Tekintsünk most egy tetszőleges

$$\sum_{j=1}^{m} \sum_{\alpha=j([i_{j-1}]_{j-1})}^{i_j} p_{[\alpha]_j j} \tag{9.33}$$

alakú összeget, ahol az indexhatárokra a megfelelő feltételek teljesülnek. Vegyük észre, hogy ebben egyetlen megmunkálás sem kezdődhet előbb, mint az összegzési sorrendben közvetlen előtte lévő. Ugyanis ha $j([i_{j-1}]) < \alpha \le i_j$, akkor az $[\alpha]_j$ munkadarabnak a j gépen való megmunkálásáról van szó, amit $[\alpha-1]_j$ ugyanezen a gépen való megmunkálásának meg kell előznie. Ha $\alpha = j([i_{j-1}])$, akkor $[\alpha]_j = [i_{j-1}]_{j-1}$, és $[\alpha]_j$-nek a j-edik gépen való megmunkálását meg kell előznie a $(j-1)$-edik gépen való megmunkálásnak. Tehát a (9.33) alakú összegek alsó korlátok a teljes átfutási időre. Mivel egy közülük éppen egyenlő vele, ezért a tétel állítása igaz. ■

9.31. definíció. *Egy feladat duáljának nevezzük azt a feladatot, ahol az i munkadarab p^d_{ij} megmunkálási idejét a j gépen a*

$$p^d_{ij} = p_{i,m+1-j}$$

képlet adja meg.

9.32. tétel. *Egy $F \parallel C_{\max}$ feladatnak és duáljának optimális célfüggvényértéke megegyezik.*

Bizonyítás. Tekintsük a feladat egy ütemezését. Vizsgáljuk a duál feladat azon ütemezését, ahol az $(m-j)$-edik gépen a megmunkálások sorrendjét a primál feladat adott ütemezéséből úgy kapjuk, hogy a j-edik gépen való sorrendet megfordítjuk, azaz a duál feladatban a sorrend az $m-j$ gépen $[n]_j, [n-1]_j, \ldots, [1]_j$. Ekkor azonban mind a primál, mind a duál feladat esetén ugyanazok az összegek szerepelnek a (9.32) képletben, csak az összeadandók sorrendje fordított. Így a két átfutási idő azonos, következésképp az optimális átfutási idők is azonosak. ■

9.33. tétel. *Az $F \parallel C_{\max}$ előzéses feladatnál $m \leq 3$ esetén van olyan optimális megoldás, hogy valamennyi gépen ugyanaz a megmunkálások sorrendje.*

Bizonyítás. $m = 1$ esetén nincs előzés. Az állítás $m = 2$ esetén a 9.28. tételből adódik. Ezen utóbbi tételből és az előzőből pedig következik, hogy a feladatnak van olyan optimális megoldása, ahol az első két gépen azonos a sorrend. Viszont $m = 3$ esetén mind az első két, mind az utolsó két gép egyike a második gép, így van olyan optimális megoldás, ahol mindhárom gépen azonos a sorrend. ■

9.7. példa. Az előzés gyengíti a feltételeket, azaz több megoldást enged meg. Ezért az előzéses feladat optimum értéke nem nagyobb, mint az előzés nélkülié. Az olyan feladatnak, ahol az előzéses megoldás határozottan jobb, legalább négy gépet és két munkadarabot kell tartalmaznia. Ilyen minimális méretű ellenpélda létezik, és a megmunkálási időket az alábbi táblázat tartalmazza.

gép	1. munkadarab	2. munkadarab
1.	3	1
2.	1	3
3.	1	3
4.	3	1

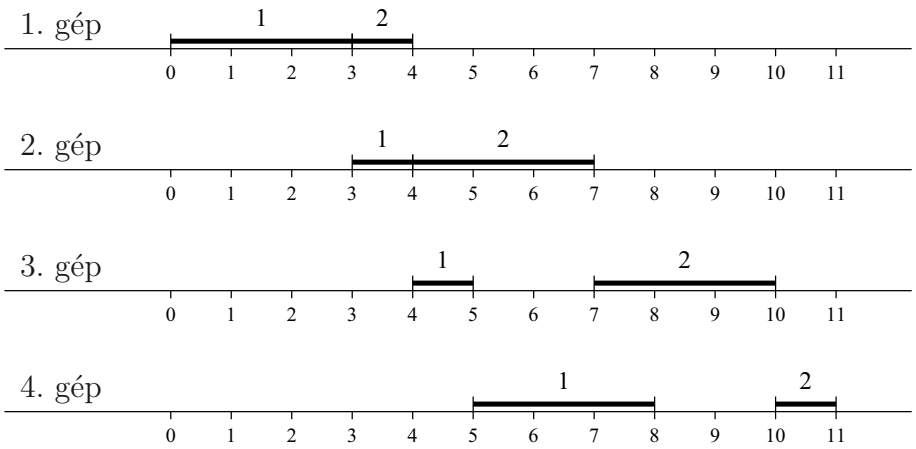
Az előzés nélküli esetben az $(1,2)$ sorrend esetén 11 az átfutási idő (lásd a 9.5. ábrát).

A $(2,1)$ sorrend esetén is 11 az átfutási idő (lásd 9.6. ábra).

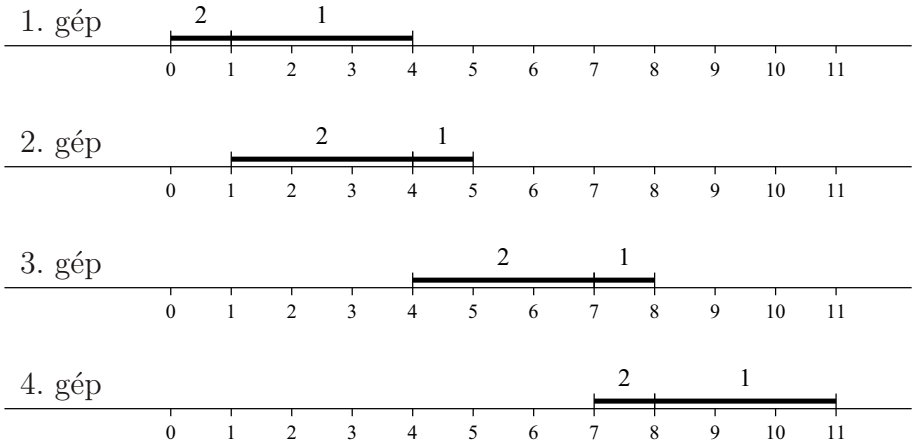
Ha viszont az első két gépen $(2,1)$, a második két gépen $(1,2)$ a sorrend, akkor az átfutási idő csak 10 (lásd 9.7. ábra).

$m \geq 3$ esetén még az előzés nélküli eset is NP-teljes. Két gépre viszont Johnson már 1954-ben megoldotta a problémát.

9.34. tétel. *Az $F2 \parallel C_{\max}$ feladat optimális megoldását a következő algoritmus adja:*



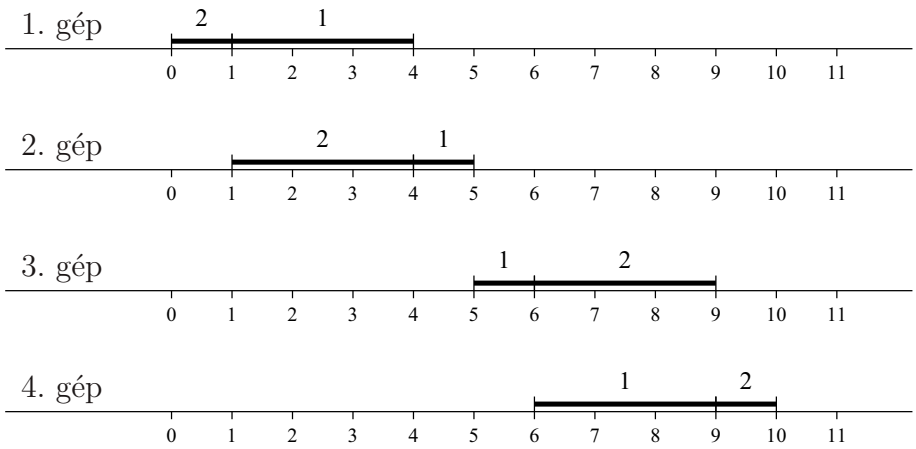
9.5. ábra. Az átfutási idő az (1,2) ütemezés esetén.



9.6. ábra. Az átfutási idő a (2,1) ütemezés esetén.

1. *Rendezzük a munkadarabokat a rövidebb megmunkálási idejük szerinti növekvő sorrendbe.*
2. *Ezután ebben a sorrendben döntünk a munkák sorrendjéről. Tegyük fel, hogy már van egy K kezdeti és egy B befejező sorozat. Ekkor a soron következő megmunkálás aszerint kerül K végére, illetve B elejére, hogy a rövidebb megmunkálási ideje az első, illetve a második gépen volt-e.*

Bizonyítás. Tekintsünk két munkát, indexük legyen j és k, amelyek ebben a sorrendben közvetlenül egymás után következnek. Legyen T_1, illetve T_2 az az időpont, amikor j előtt minden megmunkálás befejeződik az első, illetve a második gépen. Ez azt jelenti, hogy a j munkadarab megmunkálása az első gépen a T_1 időpontban kezdődik meg, míg a második gépen a $\max\{T_1 + p_{1j}, T_2\}$ időpontban, azaz itt a $\max\{T_1 + p_{1j} + p_{2j}, T_2 + p_{2j}\}$ időpontban



9.7. ábra. Az átfutási idő előzéses ütemezés esetén.

fejeződik be. Ezért k megmunkálása a második gépen nem kezdődhet meg ennél, valamint $(T_1 + p_{1j} + p_{1k})$-nál korábban. Összefoglalva azt kapjuk, hogy k befejezésének időpontja a második gépen

$$\max\{T_1 + p_{1j} + p_{2j} + p_{2k}, T_2 + p_{2j} + p_{2k}, T_1 + p_{1j} + p_{1k} + p_{2k}\} \,. \tag{9.34}$$

Ha felcseréljük a két munkadarab sorrendjét, akkor a j befejezésének időpontja a második gépen

$$\max\{T_1 + p_{1k} + p_{2k} + p_{2j}, T_2 + p_{2j} + p_{2k}, T_1 + p_{1k} + p_{1j} + p_{2j}\} \,. \tag{9.35}$$

Vegyük észre, hogy az utolsó két képlet középső tagjai azonosak. Ezért, ha meg tudunk határozni egy olyan sorrendet, amelyben a két szélső tag maximuma mindig (9.34)-ban lesz kisebb, akkor ez lesz az optimális sorrend. Vegyük észre azt is, hogy a két szélső tagban a T_1 additív konstans szerepel, ami nem befolyásolja a maximum helyét, ezért ezt további vizsgálatainkból kihagyjuk. Legyen

$$A = p_{1j} + p_{2j} + p_{2k}, \quad B = p_{1j} + p_{1k} + p_{2k} \tag{9.36}$$

$$C = p_{1k} + p_{2k} + p_{2j}, \quad D = p_{1k} + p_{1j} + p_{2j} \,. \tag{9.37}$$

Arra keresünk feltételt, hogy mikor lesz $\max\{A, B\} \le \max\{C, D\}$. Látható, hogy

$$\begin{aligned}
A \le C &\text{ akkor és csak akkor, ha } p_{1j} \le p_{1k} \\
A \le D &\text{ akkor és csak akkor, ha } p_{2k} \le p_{1k} \\
B \le C &\text{ akkor és csak akkor, ha } p_{1j} \le p_{2j} \\
B \le D &\text{ akkor és csak akkor, ha } p_{2k} \le p_{2j} \,.
\end{aligned}$$

A megmunkálási idők hossza szerint négy esetet különböztetünk meg.

(i) Ha $p_{1j} \le p_{2j}$ és $p_{1k} \ge p_{2k}$, akkor a fentiekből azonnal adódik a kívánt reláció. Ez az az eset, amikor a rövidebb megmunkálási idő az elől álló munka esetében az első, a

mögötte lévőnél pedig a második gépen van.

(ii) Ha $p_{1j} \leq p_{2j}$ és $p_{1k} < p_{2k}$, akkor $B \leq C$ és $A > D$. Ezért még az $A \leq C$ egyenlőtlenségre is szükség van, amiből $p_{1j} \leq p_{1k} < p_{2k}$, tehát mindkét munkának az első gépen rövidebb a megmunkálási ideje, és ezek közül is az elől álló munkáé a rövidebb.

(iii) Ha $p_{1j} > p_{2j}$ és $p_{1k} \geq p_{2k}$, akkor $B > C$ és $A \leq D$, ezért az egyenlőtlenség, amire szükségünk van, $B \leq D$. Ez akkor igaz, ha $p_{2k} \leq p_{2j} < p_{1j}$, vagyis mindkét munkának a második gépen rövidebb a megmunkálási ideje, és ezek közül is a hátul álló munkáé a rövidebb.

(iv) Ha $p_{1j} > p_{2j}$ és $p_{1k} < p_{2k}$, akkor $B > C$ és $A > D$, tehát a kívánt reláció nem teljesülhet, vagyis nem lehetséges, hogy az elől álló munkának a második, a hátul állónak pedig az első gépen legyen határozottan rövidebb a megmunkálási ideje. ■

A jelen szakasz további részében az előzés nélküli feladattal foglalkozunk.

Az egyutas ütemezési probléma pontos megoldási módszerei többnyire vagy korlátozás és szétválasztás, vagy implicit leszámlálás típusú módszerek. A korlátozás és szétválasztás lényege, hogy a megengedett megoldások halmazát egyre kisebb részhalmazokra bontja fel, miközben (a) egyre pontosabban becsüli a részhalmazokba eső legjobb megengedett megoldások célfüggvényértékét, és (b) a becslő eljárás során időnként megengedett megoldásokat állít elő. Ily módon feleslegessé válnak azok a részhalmazok, ahol a becslés rosszabb, mint az addig talált legjobb megoldás értéke. Az implicit leszámlálásnál a feltételek kielégíthetőségéből tudunk levonni következtetéseket. Az ütemezési feladatok esetében ez azt jelenti, hogy bizonyos típusú sorrendek esetén a célfüggvény értéke meghaladna egy meghatározott értéket.

Tanulságosak azok a teljes átfutási időre vonatkozó korlátok, amelyeket a korlátozás és szétválasztás típusú módszereknél kell használni.

Az alsó korlátok mind a 9.30. tételt használják. Két fajtájuk van, a gépre, illetve a munkára alapozott alsó korlát. Mindkettő abból indul ki, hogy az adott objektummal kapcsolatos összes megmunkálást (vagyis az adott gépen vagy az adott munkadarabon végrehajtandó összes megmunkálást) el kell végezni és ez sikerül is állásidő nélkül, és ez előtt és után is a lehető legkevesebb időt használjuk fel. Mindezt az alábbi két tételben fogalmaztuk meg.

9.35. tétel. *Az $F \parallel C_{\max}$ feladat optimális célfüggvényértéke legalább*

$$\max\{\sum_{j\neq l} \min\{p_{1j}, p_{mj}\} + \sum_{i=1}^{m} p_{il} : \quad l = 1, \ldots, n\} . \tag{9.38}$$

Bizonyítás. Tekintsünk egy tetszőleges l munkát. Ennek végig kell mennie valamennyi gépen, vagyis a teljes átfutási időnek tartalmaznia kell ezen munka megmunkálási idejének összegét (ez a második összeg a fenti képletben). Ezen felül minden más j munka vagy előtte van a sorrendben, és akkor ezen j munkának az első gépen való megmunkálási ideje hátrébb tolja l megmunkálásának kezdetét, vagy mögötte van, ekkor j megmunkálása az utolsó gépen csak l befejezése után történik, vagyis a teljes átfutási idő l befejezéséhez képest ennyivel kitolódik. Akkor járunk a legjobban, ha pontosan azok a munkák vannak l előtt, amelyeknek a megmunkálása az első gépen rövidebb, mint az utolsón. Vagyis – a 9.30. tételhez hasonlóan – a két összeg olyan megmunkálásokat tartalmaz, amelyeknek időben egymás után kell következniük. ■

9.36. tétel. *Az $F \parallel C_{\max}$ feladat optimális célfüggvényértéke legalább*

$$\max\{\min_j \sum_{l=1}^{i-1} p_{lj} \; + \; \sum_{j=1}^{n} p_{ij} \; + \; \min_j \sum_{l=i+1}^{m} p_{lj} : \quad i = 1, \ldots, m\} \, . \tag{9.39}$$

Bizonyítás. A hármas összeg tagjainak jelentése a következő. Az első tag azt a legrövidebb időt adja meg, amennyinek mindenképpen el kell telnie ahhoz, hogy az első megmunkálás megkezdődhessék az i-edik gépen, és ez nem más, mint az a legrövidebb idő, ami alatt egy munkadarab az i-edik gépet eléri. A második összeg a megmunkálások ideje az i-edik gépen. Ezek természetesen csak azután kezdődhetnek meg, hogy az első munkadarab elérte a gépet. Végül a harmadik összeg az a legrövidebb idő, ami alatt egy munkadarab elkészítését be lehet fejezni azután, hogy lekerült az i-edik gépről. Vagyis hasonlóan a 9.30. tételhez, a három tag olyan megmunkálásokra tartalmaz alsó becslést, amelyeknek időben egymás után kell következniük. ■

Általános szokás, hogyha egy nehéz feladat egy speciális esetét meg lehet oldani valamely egyszerű eljárással, akkor úgy készítenek heurisztikus eljárást az általános esetre, hogy „visszavezetik" azt a megoldott speciálisra. Mivel az általános egyutas feladat is nehéz, de ugyanakkor Johnson algoritmusa az $m = 2$ esetet megoldja, tehát pontosan a fenti helyzettel állunk szemben, pusztán az a kérdés, hogy hogyan lehet értelmezni a két „gépet". Az idézőjel itt arra vall, hogy ez a két gép csak absztrakt gép lesz, nem az eredeti feladat valamely gépe.

Mielőtt a fenti kérdésre válaszolnánk, érdemes szemügyre venni a Johnson algoritmusa mögött meghúzódó gondolatot. Emlékezzünk rá, hogy azok a munkák kerültek előre, amelyeknek a rövidebb megmunkálási ideje az első gépen, a hosszabb pedig a másodikon volt. A sorrend végére pedig azok kerültek, amelyeknél a helyzet éppen fordított volt. Az ilyen sorrend azt célozza, hogy minél előbb legyen munkája a második gépnek, vagyis ott kevés legyen az állásidő, majd miután az első gép befejezte a munkáját (természetesen állásidő nélkül), akkor minél rövidebb ideig kelljen a második gépnek dolgoznia. Kiterjesztve ezt több gép esetére, olyan sorrendet szeretnénk, ahol az első gépeken kevés időt használó munkák kerülnek előre, a hátsó gépeken keveset használók pedig a sorrend végére.

Úgy lehet két gépet „csinálni" a sokból, hogy bizonyos gépeket „összevonunk", azaz úgy tekintjük, mintha az ezeken végzendő megmunkálások egyetlen nagy megmunkálást adnának ki, ahol a megmunkálási idő az eredeti megmunkálási idők összege. A fentiek szellemében ezt két különböző módon is meg lehet csinálni. Az egyik esetben elölről is és hátulról is ugyanannyi gépet tekintünk, míg a gépek középső részét figyelmen kívül hagyjuk. Ha r a tekintetbe veendő gépek száma mindkét oldalról, akkor a két mesterséges gépen a megmunkálási idők

$$p^r_{1j} = \sum_{i=1}^{r} p_{ij}, \quad p^r_{2j} = \sum_{i=m-r+1}^{m} p_{ij}, \quad j = 1, \ldots, n \tag{9.40}$$

lesznek. Itt $r = 1, \ldots, \lceil m/2 \rceil$. Az így definiált kétgépes feladatokat minden r esetén különkülön megoldjuk Johnson módszerével, majd az így kapott sorrendre kiszámítjuk az eredeti

feladat átfutási idejét. Végül az így kapott sorrendek közül a legjobbat választjuk. Még ha m páratlan is, r felső határának ilyetén megválasztása biztosítja, hogy minden, az eredeti feladatban szereplő gépet legalább egyszer figyelembe veszünk.

A másik lehetőség, hogy a gépeket két részre osztjuk, és eszerint határozzuk meg a mesterséges gépeken a megmunkálási időket. Ha az első csoportban r gép van, akkor a megmunkálási időket

$$p^r_{1j} = \sum_{i=1}^{r} p_{ij},\ p^r_{2j} = \sum_{i=r+1}^{m} p_{ij},\quad j = 1, \ldots, n \tag{9.41}$$

adja meg. Itt $r = 1, \ldots, m-1$. Minden másban úgy járunk el, mint fent.

Befejezésül azt mutatjuk meg, hogy az $F \mid no\text{-}wait \mid C_{\max}$ probléma egy utazóügynök feladatra vezethető vissza. Ebben a feladatban tehát a munkák nem várhatnak a gépek előtt, hanem éppen fordítva, a gépeknek kell várniuk a munkákra.

Ha a j-edik munkának sehol nem kell várnia a gépek előtt, és gyártása a 0 időpontban kezdődik, akkor a megmunkálása az r gépen a

$$\sum_{i=1}^{r} p_{ij}$$

időpontban fejeződik be. Tegyük fel, hogy közvetlenül utána a k munka következik, melynek gyártása a $t \geq 0$ időpontban kezdődik. Tudjuk, hogy k csak akkor kerülhet az r gépre, ha a megmunkálása az $r-1$ gépen befejeződött, ami a

$$t + \sum_{i=1}^{r-1} p_{ik}$$

időpontban következik be. Ekkor azonban az r gépnek szabadnak kell lennie, azaz teljesülnie kell a

$$t + \sum_{i=1}^{r-1} p_{ik} \geq \sum_{i=1}^{r} p_{ij}$$

egyenlőtlenségnek. Vagyis t-nek el kell érnie legalább a

$$\sum_{i=1}^{r} p_{ij} - \sum_{i=1}^{r-1} p_{ik}$$

értéket. Mivel ennek minden gépre teljesülnie kell, ezért a k munka a j munka után

$$\max\left\{\sum_{i=1}^{r} p_{ij} - \sum_{i=1}^{r-1} p_{ik}\ \ r = 1, \ldots, m\right\} \tag{9.42}$$

idővel indítható. (Itt az üres összeg értéke szokás szerint 0, továbbá a megmunkálási időkről feltesszük, hogy nemnegatívak, így a fenti érték is nemnegatív.) Mivel a feltételezés szerint a megmunkálási időket nem tudjuk változtatni, csak úgy tudjuk a teljes átfutási időt a lehető legrövidebbé tenni, hogy minimalizáljuk a munkák indításai között eltelő várakozási idők összegét. Ez azonban nem más, mint az az utazóügynök feladat, ahol a városok a munkák, a közöttük lévő távolságot pedig a (9.42) képlet adja meg.

9.6. A többutas ütemezési probléma

A többutas problémák családja tartalmazza a legbonyolultabb ütemezési feladatokat. Itt semmiféle megkötés nincs a technológiai útvonalakra, még az is megengedett, hogy egy munka ne kerüljön minden gépre, és hogy egyes gépeket többször is felkeressen. Egy megszorítás van, amit általánosan fel szoktak tenni, nevezetesen, hogy nincsenek párhuzamos berendezések, azaz minden gépből csak egy van. Ez annyit jelent, hogy minden megmunkálást egy meghatározott gépen kell elvégezni.

Ahhoz, hogy egy feladatot pontosan meg lehessen fogalmazni és oldani, valamilyen matematikai modell kell. A két leggyakrabban használt modell az ú.n. diszjunktív gráf modell és a diszkrét programozási modell. Először ezeket tárgyaljuk.

Az alábbiakban az angol nyelvű irodalomhoz hasonlóan a megmunkálás szó helyett a művelet kifejezést alkalmazzuk. Ennek oka az, hogy megmunkáláson inkább értjük az egy munkadarabon végzett átalakítást, míg művelet bármi lehet, még két különálló darab összehegesztése vagy egyetlen elem feldarabolása is.

Különösen áll ez az elsőnek tárgyalandó diszjunktív gráf modellre. Ennek mélyebb megértéséhez hasznos a kritikus út módszerének ismerete.

9.6.1. A kritikus út módszere

Ez a módszer (CPM) nem képezi a jelen fejezet tárgyát, ezért itt csak a legfontosabb tudnivalókat foglaljuk össze. Tegyük fel, hogy valamilyen tevékenységek sorozatát – esetünkben valamely termékek gyártásához tartozó műveleteket – akarjuk elvégezni. Ezen tevékenységek között lehetnek bizonyos megelőzési kapcsolatok, amelyek azt fejezik ki, hogy valamely tevékenységnek be kell fejeződnie ahhoz, hogy egy másik elkezdődhessék. A feladat több különböző módon is leírható egy G irányított gráffal. Nekünk arra a változatra lesz szükségünk, ahol a gráf csúcsai a tevékenységek. Ekkor egy α csúcsból (tevékenységből) irányított él vezet minden olyan β csúcsba (tevékenységbe), amely őt közvetlenül követheti. Az él hossza az α tevékenység elvégzéséhez szükséges idő, azaz az α csúcsból kiinduló valamennyi él hossza azonos. Feltételezzük továbbá egy 0 hosszúságú $\circ$ kezdeti és $*$ befejező tevékenység létezését, amelyek a következő tulajdonságokkal rendelkeznek.

(i) A $\circ$ csúcsba nem vezet él.

(ii) A $*$ csúcsból nem indul ki él.

(iii) Legyen α tetszőleges, $\circ$-től és $*$-tól különböző tevékenység. Ekkor $\circ$-ből vezet irányított út α-ba, és α-ból vezet irányított út $*$-ba.

A G gráf egy fontos további tulajdonsága még, hogy

(iv) A G gráf irányított kört nem tartalmaz.

Ha ugyanis irányított kört tartalmazna, akkor a körbe tartozó valamennyi tevékenység megkezdésének az volna a feltétele, hogy a kör többi tevékenysége befejeződjön, azaz egyetlen, a körhöz tartozó tevékenységet sem lehetne elkezdeni.

Feltételezve, hogy az egész tevékenységsorozat a 0 időpontban kezdődik, igaz a következő

9.37. tétel. *(a) Az egész tevékenységsorozat lehetséges legrövidebb átfutási ideje egyenlő a $\circ$-ből $*$-ba vezető leghosszabb út hosszával.*

(b) Semmilyen α tevékenység sem kezdődhet előbb, mint a $\circ$-ből az α-ba vezető leghosszabb út hossza.

(c) Ha a tevékenységsorozat lehetséges legrövidebb átfutási ideje T, akkor egyetlen α tevékenység sem kezdődhet később, mint

$$T - (\alpha\text{-ból } *\text{-ba vezető leghosszabb út hossza})$$

anélkül, hogy a teljes átfutási időt ne tolná ki T-n túlra.

A tétel (a) részében említett leghosszabb úthoz tartozó tevékenységek csak egyetlen jól meghatározott időpontban kezdődhetnek a teljes átfutási idő meghosszabbítása nélkül, ezért ezeket „kritikusnak" hívják, innen ered az egész módszer elnevezése.

Jelölje $p_\alpha, q_\alpha, r_\alpha$ rendre az α tevékenység elvégzéséhez szükséges időt, α lehetséges legkésőbbi befejezési idejét a teljes átfutási idő meghosszabbítása nélkül és végül α lehetséges legkorábbi kezdési idejét. Az utóbbi két mennyiség egy dinamikus programozási algoritmusból is meghatározható. (Ebben fontos szerepet játszik, hogy a gráf nem tartalmaz irányított kört.) A megfelelő Bellman-egyenleteket az alábbi tétel adja meg.

9.38. tétel. *Jelölje N a G gráf csúcsainak, A pedig éleinek halmazát. Ekkor*

$$r_\circ = 0\,, \tag{9.43}$$

$$\textit{minden } \alpha \in N \setminus \{\circ\} \textit{ esetén} \quad r_\alpha = \max\{r_\beta + p_\beta : (\beta,\alpha) \in A\}\,, \tag{9.44}$$

$$q_* = T\,, \tag{9.45}$$

$$\textit{minden } \alpha \in N \setminus \{*\} \textit{ esetén} \quad q_\alpha = \min\{q_\beta - p_\beta : (\alpha,\beta) \in A\}\,. \tag{9.46}$$

9.6.2. A diszjunktív gráf modell

Tekintsünk két tetszőleges α, β műveletet. Ezek az alábbi három viszony valamelyikében állnak egymással:
(i) az egyiknek meg kell előznie a másikat (pl. azért, mert ugyanahhoz a munkadarabhoz tartozó műveletekről van szó, és a technológia előírja a sorrendjüket),
(ii) egyszerre nem végezhetők (pl. azért, mert mindkettő ugyanazt a gépet igényli),
(iii) nincsenek közvetlen hatással egymásra (pl. különböző munkadarabokon különböző gépekkel végzendő műveletek).

A fenti helyzetet a $G = (N, C \cup D)$ ú.n. diszjunktív gráf írja le, ahol N a műveletek halmaza, C az (i) kategóriába eső relációkat leíró, konjunktívnak nevezett éleket tartalmazza, D pedig a (ii) kategóriába tartozó relációkat reprezentáló, diszjunktív élpárokat. Ez azt jelenti, hogyha az α és β művelet nem végezhető egyszerre, akkor mind az (α,β), mind a (β,α) él eleme a D halmaznak. Továbbá minden $\alpha, \beta \in N$ esetén feltesszük, ha $(\alpha,\beta) \in C \cup D$, akkor az (α,β) él hossza p_α, azaz az α művelet elvégzéséhez szükséges idő, ami független az él végpontjától, azaz β-tól. Az élek hosszát, azaz a megmunkálási időket, mindvégig nemnegatívnak tételezzük fel.

9.39. definíció. *Az $A \subset D$ halmazt* ***programnak*** *nevezzük, ha $(\alpha, \beta) \in A$ akkor és csak akkor, ha $(\beta, \alpha) \notin A$.*

Ha tehát A egy program, akkor a $G' = (N, C \cup A)$ egy szokásos irányított gráf. Ennek élei határozzák meg, hogy a műveletek milyen sorrendben végezhetők. Ha minden műveletet a lehetséges legkorábbi időpontra ütemezünk, azaz nem iktatunk be állásidőt olyan esetben, amikor mind a gép rendelkezésre áll, mind a sorrend szerint következő művelet elvégezhető volna, akkor G' egyértelműen meghatároz egy termelési programot.

A diszjunktív gráf modell nyelvén a $J||C_{\max}$ feladat úgy fogalmazható meg, hogy meghatározandó a G gráfhoz egy A program úgy, hogy az így keletkező G' gráfban a kritikus út hossza minimális legyen.

Kritikus útról természetesen csak akkor lehet beszélni, ha G' nem tartalmaz irányított kört. Ezért az alábbiakban mindvégig feltesszük, hogy *csak olyan A programokat vizsgálunk, hogy G' nem tartalmaz irányított kört.*

A diszjunktív gráf modell megoldására szolgáló módszerek kevés kivételtől eltekintve mind a korlátozás és szétválasztás elvén alapulnak. Ehhez természetesen a kritikus út hosszára vonatkozó alsó korlátra van szükség. Az alábbiakban tárgyaljuk a leggyakoribbakat.

Amikor az optimális A programot keressük, az élpárok egy-egy tagját külön-külön választjuk ki. Így az algoritmusok során mindig olyan helyzet áll fenn, hogy a C halmazhoz még hozzáválasztunk néhány további élt. Ezt a halmazt fogjuk B-vel jelölni. Tehát valamely alkalmas A program mellett mindig igaz a $C \subset B \subset C \cup A$ reláció. Az (N, B) gráfot H jelöli. Hangsúlyozzuk, hogy H tartalmazza az összes konjunktív élt, bizonyos diszjunktív élpárokból egyet, míg a többi diszjunktív élpárból egyik élt sem.

9.40. definíció. *Legyen M a műveletek egy részhalmaza. Azt mondjuk, hogy M* ***diszjunktív,*** *ha bármely $\alpha, \beta \in M$ $(\alpha \neq \beta)$ esetén α-t és β-t a H gráfban vagy egy konjunktív élekből álló irányított út köti össze, vagy egy diszjunktív élpár.*

Szükségünk lesz még az alábbi két mesterséges műveletre, melyek bevezetése megkönnyíti a számításokat: $\circ$ egy 0 hosszúságú kezdő művelet, $*$ pedig 0 hosszúságú befejező művelet.

Ezen két műveletről bármely (N, B) gráfra vonatkozóan, tehát még a (N, C) esetén is a következőket tesszük fel:
(1) A $\circ$ csúcsba nem vezet él.
(2) A $*$ csúcsból nem indul ki él.
(3) Legyen α tetszőleges, $\circ$-től és $*$-tól különböző művelet. Ekkor $\circ$-ból vezet irányított út α-ba, és α-ból vezet irányított út $*$-ba.

Továbbá az alábbi jelöléseket alkalmazzuk, amelyek mind a $H = (N, B)$ gráfra vonatkoznak:

- r_α az α művelet megkezdésének lehetséges legkorábbi időpontja,

- q_α az az idő, aminek minimálisan el kell telnie az α művelet befejezése után minden művelet befejezéséig.

Könnyen látható, hogy r_α nem más, mint a $\circ$-ből α-ba vezető leghosszabb út hossza a H gráfban, míg q_α ugyanitt az α csúcsból a $*$ csúcsba vezető leghosszabb út hossza p_α nélkül. Ezért az r_α, q_α mennyiségekre a (9.43)–(9.46) képletekhez hasonlóan az alábbi rekurzív képletek adhatók meg:

$$r_\circ = 0\,, \tag{9.47}$$

$$\text{minden } \alpha \in N \setminus \{\circ\} \text{ esetén } \quad r_\alpha = \max\{r_\beta + p_\beta : (\beta,\alpha) \in B\}\,, \tag{9.48}$$

$$q_* \;=\; 0\,, \tag{9.49}$$

$$\text{minden } \alpha \in N \setminus \{*\} \text{ esetén } \quad q_\alpha = \max\{q_\beta - p_\beta : (\alpha,\beta) \in B\}\,. \tag{9.50}$$

Ha M a műveletek egy diszjunktív részhalmaza, és ebben az α és β művelet olyan, hogy konjunktív élekből álló út vezet α-ból β-ba, akkor az

$$r_\alpha \;\le\; r_\beta, \quad q_\alpha \;\ge\; q_\beta$$

egyenlőtlenségek automatikusan teljesülnek. Ha a B halmazhoz hozzáveszünk egy addig diszjunktív (α,β) élt, akkor az r és q értékeket azonnal úgy kell módosítani, hogy a Bellman-egyenletek továbbra is igazak maradjanak.

9.41. tétel. *Bármely A program esetén, amelyre $B \subset C \cup A$ teljesül, a teljes átfutási idő legalább*

$$\max\{r_\alpha + p_\alpha + q_\alpha : \quad \alpha \in N\}\,. \tag{9.51}$$

Bizonyítás. Az $r_\alpha + p_\alpha + q_\alpha$ összeg a $\circ$-ből α-n keresztül $*$-ba vezető leghosszabb út hossza. Tehát a képlet a kritikus út hosszát adja meg a H gráfban. Ez a hossz nem csökkenhet, ha további nemnegatív hosszú éleket vezetünk be a gráfba. ■

A 9.36. tételnek az volt a logikája, hogy megvizsgáltuk, hogy milyen gyorsan tud egyáltalán eljutni valami egy i gépre, ott optimális esetben állásidő nélkül el lehet végezni az összes megmunkálást, majd az utolsó munkadarabnak még be is kell fejeződnie a további gépeken, ezt is a lehető legrövidebbnek feltételezve. Ehhez hasonló módon számos alsó korlát nyerhető.

9.42. tétel. *A többutas ütemezési feladat átfutási ideje a műveletek bármely diszjunktív M halmaza esetén legalább*

$$\min_{\alpha\in M} r_\alpha \;+\; \sum_{\alpha\in M} p_\alpha \;+\; \min_{\alpha\in M} q_\alpha\,. \tag{9.52}$$

Bizonyítás. Gyengítsük a feladat feltételeit úgy, hogy az M halmaz kivételével az összes többi diszjunktív élpártól eltekintünk. Ekkor is minimálisan el kell annyi időnek telnie, amíg az M-beli műveletek közül az első megkezdődhet, az összes M-beli művelet befejeződik, végül az M-beli műveleteket követő megmunkálások is véget érnek. Ezen három tényezőre ad rendre alsó korlátot a (9.52) kifejezés három tagja. ■

9.43. tétel. *Legyen a műveletek bármely diszjunktív M halmaza esetén az M-beli műveleteknek egy, a q_j értékek szerinti monoton csökkenő sorrendje* $[1], [2], \ldots, [|\ M\ |]$. *A többutas ütemezési feladat átfutási ideje legalább*

$$\min_{\alpha\in M} r_\alpha \ + \ \max\{\sum_{i=1}^{j} p_{[i]} + q_{[j]} : \quad j = 1, \ldots, |\ M\ |\}\ . \tag{9.53}$$

Bizonyítás. Gyengítsük ugyanúgy a feltételeket, mint az előző tételnél. Továbbá tegyük fel, hogy minden művelet rendelkezésre áll a $\min_{\alpha\in M} r_\alpha$ időpontra. Legyen a teljes tevékenység végső határideje T. Innen a gyengített feltételek alapján adódik, hogy az M-beli α művelet határideje $T - q_\alpha$. Az eredeti feladat teljes átfutási idejére T olyan értéke ad alsó becslést, amely mellett a gyengített feladatban a maximális késés nemnegatív. Nyilvánvalóan ezek közül az a legjobb T érték, amely mellett a maximális késés éppen 0. A 9.4. tételből tudjuk, hogy a monoton növő határidők szerinti ütemezés minimalizálja a maximális késést, és esetünkben a q_j értékek szerinti monoton csökkenő sorrend ilyen. Az adott sorrend szerinti $[j]$ művelet befejezési időpontja

$$\min_{\alpha\in M} r_\alpha \ + \ \sum_{i=1}^{j} p_{[i]}\ .$$

Mivel a maximális késés 0, ezért innen a

$$\min_{\alpha\in M} r_\alpha \ + \ \max\{\sum_{i=1}^{j} p_{[i]} \ - \ (T \ - \ q_{[j]}) : \quad j = 1, \ldots, |\ M\ |\} \ \leq \ 0$$

egyenlőtlenség adódik. Ebből T maximális értékére a (9.53)-beli kifejezés kapható. ■

9.44. tétel. *Legyen a műveletek egy diszjunktív M halmaza esetén* $[1], [2], \ldots, [|\ M\ |]$ *az M-beli műveleteknek egy, a rendelkezésre állási idők szerinti monoton csökkenő sorrendje. A többutas ütemezési feladat átfutási ideje legalább*

$$\min_{\alpha\in M} q_\alpha \ + \ \max\{\sum_{i=1}^{j} p_{[i]} + r_{[j]} : \quad j = 1, \ldots, |\ M\ |\}\ . \tag{9.54}$$

Bizonyítás. Alkalmazzuk az előző tételt arra az ütemezési feladatra, amelyben a megmunkálási idők azonosak a vizsgált problémában szereplőkkel, míg a rendelkezésre állási időket a q_j értékek adják meg és r_j időnek kell eltelnie a j művelet befejezése után a műveletek

teljes befejezéséig. ■

A korlátozás és szétválasztási módszerek gyakran keverednek az implicit leszámlálással, s nem csupán az ütemezési problémák megoldásánál. Az utóbbinak az a lényege, hogy a szóba kerülő megoldások egy részét ki akarjuk zárni, anélkül, hogy ténylegesen megvizsgálnánk őket. Itt egy ilyen lehetőséget mutatunk be a 9.42. tételre támaszkodva.

9.45. tétel. *Legyen M műveletek egy halmaza és α egy további úgy, hogy az $M \cup \{\alpha\}$ halmaz diszjunktív. Legyen továbbá $C > 0$ egy tetszőleges szám. Ha*

$$r_\alpha + \sum_{\beta \in M} p_\beta + p_\alpha + \min_{\beta \in M} q_\beta \geq C \tag{9.55}$$

és

$$\min_{\beta \in M} r_\beta + \sum_{\beta \in M} p_\beta + p_\alpha + \min_{\beta \in M} q_\beta \geq C\,, \tag{9.56}$$

akkor minden olyan ütemezésben, amelynek az átfutási ideje rövidebb, mint C, α az összes M-beli művelet mögött áll.

Bizonyítás. A (9.55) egyenlőtlenség baloldala alsó korlát a teljes átfutási időre abban az esetben, ha α megelőzi az összes M-beli műveletet, azaz ebben az esetben nem lehet az átfutási idő C-nél rövidebb. Hasonlóképpen a (9.56) egyenlőtlenség baloldala alsó korlát a teljes átfutási időre abban az esetben, ha α az M-beliek között van, azaz sem nem első, sem nem utolsó. ■

A tételben $|\,M\,|+1$ műveletről esik szó, ezek összesen $(|\,M\,|+1)!$ sorrendet alkothatnak. Ha az állítás feltételei teljesülnek, akkor már csak $|\,M\,|!$ sorrend kerülhet szóba, azaz a megvizsgálandó esetek számát $1/(|M|+1)$-ed részére redukáltuk. Ez az oka annak, hogy az ilyen teszteket akkor is érdemes igen gyakran ellenőrizni, ha számos esetben nem adnak pozitív eredményt.

A korlátozás és szétválasztás eljárásában a részhalmazok felbontásának számos módszere ismert. Ezek közül a legegyszerűbb az, amikor egy halmazt egy még nem vizsgált diszjunktív élpárnak megfelelően bontanak két részhalmazra. Egy finomabb eljárás az ú.n. *kritikus blokkon* alapul. Tegyük fel, hogy már ismert egy ütemezés C átfutási idővel. Tekintsünk a pillanatnyi, esetleg még teljes ütemezést nem adó megoldásban egy olyan α műveletet, ha ilyen egyáltalán van, amelyre

$$r_\alpha + p_\alpha + q_\alpha \geq C \tag{9.57}$$

teljesül. Ha van ilyen művelet, akkor egyetlen olyan teljes ütemezés sem adhat jobbat a már ismert megoldásnál, mely teljes ütemezés a jelenlegi megoldásból keletkezik kiegészítéssel. Ez a helyzet például éppen akkor, amikor egy minden korábbinál jobb ütemezést találtunk, hiszen ettől a pillanattól kezdve ennél jobbat keresünk, és a (9.57) feltétel legalább egy műveletre teljesül. Legyen α egy olyan művelet, amire (9.57) igaz, de egyetlen $\circ$-ből α-ba

vezető útba eső műveletre sem áll fenn a fenti egyenlőtlenség. A $\circ$-ből α-ba vezető útba eső műveletek alkotják az ú.n. kritikus blokkot. Nyilvánvalóan csak akkor tudunk a már ismertnél jobb ütemezést találni, ha sikerül α-t a kritikus blokkon belül előbbre ütemeznünk. Tehát minden, a későbbiekben felbontandó halmaznak elég csak azokat a részhalmazait tekinteni, amelyekbe eső elemek ezt a követelményt teljesítik. Itt persze adott esetben számos lehetőség merülhet fel, melyek kezelésének részleteibe nem megyünk bele. Ha a (9.57) feltétel semmilyen α mellett sem teljesül, akkor a felbontásra a fent említett egyszerű szabály alkalmazható.

9.8. példa. Tekintsük azt a – négy munkát és három gépet tartalmazó – feladatot, melyet az alábbi adatok határoznak meg. A táblázat belsejében az első szám a gép indexe, a második a művelet elvégzéséhez szükséges idő.

munka	1. művelet	2. művelet	3. művelet	4. művelet	5. művelet
1.	1/2	2/1	3/2	1/4	2/3
2.	2/1	3/3	2/4		
3.	1/3	3/2	2/2	1/3	2/4
4.	1/2	2/3	3/4	1/1	3/1

A műveleteket kétjegyű indexekkel azonosítjuk, az első jegy a munka indexe, a második jegy pedig azt mutatja meg, hogy adott munkához tartozó hányadik műveletről van szó. Ha még egyetlen műveletet sem ütemeztünk, vagyis az A halmaz üres, akkor a megfelelő r és q értékek a következők:

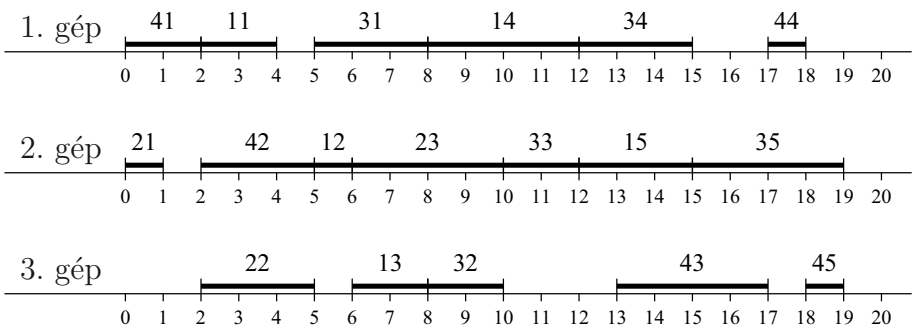
$$
\begin{array}{lllll}
r_{11}=0 & r_{12}=2 & r_{13}=3 & r_{14}=5 & r_{15}=9 \\
r_{21}=0 & r_{22}=1 & r_{23}=4 & & \\
r_{31}=0 & r_{32}=3 & r_{33}=5 & r_{34}=7 & r_{35}=10 \\
r_{41}=0 & r_{42}=2 & r_{43}=5 & r_{44}=9 & r_{45}=10 \\
q_{11}=10 & q_{12}=9 & q_{13}=7 & q_{14}=3 & q_{15}=0 \\
q_{21}=7 & q_{22}=4 & q_{23}=0 & & \\
q_{31}=11 & q_{32}=9 & q_{33}=7 & q_{34}=4 & q_{35}=0 \\
q_{41}=9 & q_{42}=6 & q_{43}=2 & q_{44}=1 & q_{45}=0
\end{array}
$$

Tekintsük a második gépet. Az ezen végzendő műveletek 12, 15, 21, 23, 33, 35, 42. Így a fenti tételek alapján a következő becsléseket kapjuk. Mivel most minden művelet esetén az $r_\alpha + p_\alpha + q_\alpha$ érték nem más, mint az adott munkához tartozó megmunkálási idők összege, ezért a 9.41. tétel alapján mind a 33, mind a 35 műveletre a 14 értéket kapjuk. Lévén $r_{21} = q_{35} = 0$, ezért a 9.42. tétel alapján a második gépen végzendő műveletek összidejét kapjuk, ami 18. Ugyanezt az értéket adja a 9.43. tétel is. A rendelkezésre állási idők szerinti csökkenő sorrend nem egyértelmű, a két lehetséges sorrend 35, 15, 33, 23, 12, 42, 21 és 35, 15, 33, 23, 42, 12, 21. Innen a 9.44. tétel alapján a 19 alsó korlát adódik, nem tekintve ugyanis a 21 műveletet, kapjuk, hogy

$$q_{35} + p_{35} + p_{15} + p_{33} + p_{23} + p_{12} + p_{42} + r_{42} = 19\ .$$

Ez azt sugallja, hogy a második gépen a műveleteket 21, 42, 12, 23, 33, 15, 35 sorrendben kell végezni, és valóban, ez ugyanis az alábbi ütemezéshez vezet, ahol t_j a művelet kezdési, c_j pedig a befejezési időpontja.

	11	12	13	14	15	21	22	23	31	32	33	34	35	41	42	43	44	45
t_j	2	5	6	8	12	0	2	6	5	8	10	12	15	0	2	13	17	18
c_j	4	6	8	12	15	1	5	10	8	10	12	15	19	2	5	17	18	19



9.8. ábra. Egy optimális ütemezés.

Az ütemezés Gantt-diagramja a 9.8. ábrán látható.

9.6.3. Egészértékű programozási modellek

Az első elterjedt modell a $J \parallel f$ feladatra vonatkozik azon egyszerűsítő feltételezés mellett, hogy minden munka legfeljebb egyszer kerül egy gépre. A képletekben az alábbi indexeket és állandókat használjuk:

- i, j: a munkák indexei,
- n: munkák száma,
- m_i: az i munkán végzendő műveletek száma,
- k, l: a műveletek indexei,
- g: a gép indexe,
- d_i: az i munka határideje,
- p_{ig}: az i munka g gépen végzendő műveletének megmunkálási ideje,
- g_{ik}: az a gép, amelyen az i munka k műveletét végezni kell,
- M: egy nagy szám, például $\sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{ig_{ik}}$.

A feladat változói pedig

- $x_{ijg} = \begin{cases} 1, & \text{ha az } i \text{ munka megelőzi a } j \text{ munkát a } g \text{ gépen,} \\ 0 & \text{különben;} \end{cases}$
- t_{ig} – az i-edik munka g gépen végzendő műveletének kezdeti időpontja.

Két feltételcsoportot kell felírnunk. Az első azt biztosítja, hogy ugyanannak a munkának a műveletei a meghatározott sorrendben kövessék egymást, és időben ne legyen átfedés közöttük, míg a második csoport különböző munkák ugyanazon a gépen végzendő műveletei közti átfedést akadályozza meg.

Az első esetben egyszerűen csak azt kell biztosítani, hogy az i munka k és $k+1$ műveletének megkezdése között legalább $p_{ig_{ik}}$ idő elteljen, azaz a

$$t_{ig_{ik}} + p_{ig_{ik}} \leq t_{ig_{i,k+1}}, \quad i = 1, \ldots, n, \; k = 1, \ldots, m_i - 1 \tag{9.58}$$

egyenlőtlenségek teljesülését kell megkövetelni. Tegyük fel, hogy $g = g_{ik} = g_{jl}$, azaz az i munka k és a j munka l műveletét azonos gépen kell végezni. Ekkor a

$$t_{jg} - t_{ig} \geq p_{ig}, \quad t_{ig} - t_{jg} \geq p_{jg} \tag{9.59}$$

egyenlőtlenségek közül egynek teljesülnie kell. Az első azt fejezi ki, hogy a g gépen i megelőzi a j munkát, azaz $x_{ijg} = 1$. Ennek felhasználásával a

$$Mx_{ijg} + t_{ig} - t_{jg} \geq p_{jg} \ , \tag{9.60}$$

$$M(1 - x_{ijg}) + t_{jg} - t_{ig} \geq p_{ig} \tag{9.61}$$

egyenlőtlenségek adódnak. Ha $x_{ijg} = 1$, akkor az első automatikusan teljesül, míg a második kikényszeríti, hogy a j munka megfelelő idővel később kerüljön a g gépre, mint i, és ha $x_{ijg} = 0$, akkor éppen fordítva. A változókra természetes módon adódnak a

$$\text{minden } i, j, g \text{ esetén } \ 0 \leq t_{ig} \leq M, \quad x_{ijg} \in \{0,\ 1\} \tag{9.62}$$

megszorítások. További feltételekre lehet szükség a célfüggvény kezelésére, illetve akkor, ha a munkákra határidők is vannak. Az utóbbi esetben azt kell megkövetelni, hogy az egyes munkák utolsó művelete a határidőnél ne később fejeződjék be, azaz

$$t_{ig_{im_i}} + p_{ig_{im_i}} \leq d_i, \quad i = 1, \ldots, n. \tag{9.63}$$

A célfüggvények közül $\sum w_j C_j$ és így $\sum w_j L_j$ írható fel a legkönnyebben, hiszen az előbbi

$$\sum_{i=1}^{n} w_i (t_{ig_{im_i}} + p_{ig_{im_i}}) = \sum_{i=1}^{n} w_i t_{ig_{im_i}} + \sum_{i=1}^{n} w_i p_{ig_{im_i}} \ .$$

Az utóbbi tag állandó, így el is hagyható. Egy új változóra, legyen ez C, és n további feltételre van szükség $C_{\max}$ kezeléséhez. Az új feltételek

$$C \geq t_{ig_{im_i}} + p_{ig_{im_i}} \quad i = 1, \ldots, n \ .$$

A célfüggvény megfelelő alakja pedig

$$\min C \ .$$

Hasonlóképpen intézhető el $L_{\max}$ is, csak ott C_i értéke helyett L_i értékére kell az egyenlőtlenségeket megkövetelni.

Vegyük számba a modell változóit és feltételeit. Mint láttuk, x_{ijg} és x_{jig} közül elég csak az egyiket bevezetni, így ezek maximális száma $n(n-1)m/2$. Mivel minden munka csak egyszer kerülhet egy gépre, ezért a t_{ig} változók száma legfeljebb nm, összesen pontosan $\sum_{i=1}^{n} m_i$. A (9.58) feltételek száma munkánként $m_i - 1$, azaz összesen $\sum_{i=1}^{n} (m_i - 1) \leq nm - n$. A (9.60)–(9.61) feltételeké pedig legfeljebb $n(n-1)m$.

A következő másik modell általánosabb, és tekintettel tud lenni különböző erőforrásokra is. Ez a modell egy ú.n. *oszlopgeneráló eljárás* alapja.

Matematikai programozásban akkor neveznek így egy módszert, ha az egy olyan modellen (feladaton) alapszik, aminek explicit felírása nem lehetséges, mert az legalább egyenértékű volna a feladat megoldásával, de a modell, illetve annak matematikai tulajdonságai lehetővé teszik, hogy egyre újabb oszlopokat, azaz változókat (a hozzájuk tartozó együtthatókkal együtt), vezessünk be a lineáris feltételeket tartalmazó modell megoldása során, és teszteljük a mindenkori részmegoldás optimalitását. Így végső soron a feladatot annak explicit felírása nélkül tudjuk megoldani.

A modell a $J \mid res \mid \sum f_j$ feladatra vonatkozik, azzal a további könnyítéssel, hogy nem követeli meg, hogy az egyes munkákhoz tartozó műveletekre egyértelműen előírt sorrend legyen, hanem megengedi, hogy a szükséges megelőzési feltételeket munkánként különkülön egy háló írja le.

Viszont feltesszük, hogy minden j esetén az f_j célfüggvény reguláris, továbbá, hogy a megmunkálási idők egészek.

Legyen a j munka műveleteinek halmaza O_j. A j munka ütemezésének nevezzük a t_α számokat, ha minden $\alpha, \beta \in O_j$ esetén $t_\alpha + p_\alpha \leq t_\beta$ teljesül, feltéve, hogy az α műveletnek meg kell előznie a β műveletet. Itt t_α egyben azt is jelenti, hogy minden α művelet esetén a műveletet a $[t_\alpha, t_\alpha + p_\alpha]$ időintervallumban végzik el. Egy további megszorítás, hogy feltesszük, hogy a feladat véges T időkorláton belül megoldható. Az adatok egész voltából következik, hogy feltehető a t_α számok egész volta is. Végül azt a következtetést kapjuk, hogy csak véges sok ütemezés van, ami közül választanunk kell. Ezt a számot a j munka esetén w_j jelöli.

Az erőforrásokra vonatkozó feltételezések a következők. A figyelembe veendő erőforrások száma s. Minden α művelet az elvégzése során állandó intenzitással használja valamennyi erőforrást, ide értve azt az esetet is, amikor a művelet elvégzéséhez nem kell valamely erőforrás. Ezt a mennyiséget, vagyis azt, hogy az α művelet mennyit igényel a k erőforrásból, $b_{\alpha k}$ jelöli. Ilyen módon bármely j munka és annak h ütemezése esetén megmondható, hogy a k erőforrásból a t időpontban az adott munkának mekkora mennyiségre van szüksége, amit a_{jhkt} jelöl. A k erőforrásból a t időpontban elérhető mennyiség e_{kt}.

Hasonlóképpen bármely j munka és h ütemezés esetén egyértelműen meghatározott a j munka során felmerülő költség, ami f_{jh}.

Az alapfeladatban a költségeket akarjuk minimalizálni úgy, hogy semmikor sem lépjük túl az erőforrások szabta korlátokat és minden munkára pontosan egy ütemezést választunk ki. Így a fenti jelölésekkel az alábbi feladathoz jutunk:

$$\min \sum_{j=1}^{n} \sum_{h=1}^{w_j} f_{jh} x_{jh} \tag{9.64}$$

$$\sum_{j=1}^{n} \sum_{h=1}^{w_j} a_{jhkt} x_{jh} \leq e_{kt}, \quad k = 1, \ldots, s, \; t = 0, \ldots, T \,, \tag{9.65}$$

$$\sum_{h=1}^{w_j} x_{jh} = 1 \quad j = 1, \ldots, n \,, \tag{9.66}$$

$$x_{jh} \in \{0, 1\}, \quad j = 1, \ldots, n, \; h = 1, \ldots, w_j \,. \tag{9.67}$$

Tehát x_{jh} az a döntési változó, ami akkor és csak akkor 1, ha a j munka h ütemezését használjuk, különben 0. Itt a (9.65) egyenlőtlenségek jelentik az ú.n. összekötő feltételeket. Ha ezek nem volnának, akkor a feladat igen egyszerű lenne, ugyanis minden munkára a legkisebb költségű ütemezést kellene választani.

Legyenek a λ_{kt} számok tetszőleges rögzített, nemnegatív mennyiségek ($k = 1,\dots,s;\ t = 0,\dots,T$). Könnyen látható, hogy ekkor a (9.64)–(9.67) feladat célfüggvényének értékét minden megengedett megoldás esetén alulról becsüli az alábbi, ú.n. Lagrange-feladat célfüggvényének értéke a változók ugyanazon értéke mellett

$$\min \sum_{j=1}^{n}\sum_{h=1}^{w_j} f_{jh}x_{jh} + \sum_{k=1}^{s}\sum_{t=0}^{T}\sum_{j=1}^{n}\sum_{h=1}^{w_j} a_{jhkt}x_{jh}\lambda_{kt} - \sum_{k=1}^{s}\sum_{t=0}^{T} e_{kt}\lambda_{kt}\ , \tag{9.68}$$

$$\sum_{h=1}^{w_j} x_{jh} = 1, \quad j = 1,\dots,n\ ; \tag{9.69}$$

$$x_{jh} \in \{0,\ 1\} \quad j = 1,\dots,n\ h = 1,\dots,w_j\ . \tag{9.70}$$

A középső tagban az összegzés sorrendjét megváltoztatva a célfüggvény az alábbi alakra hozható

$$\min \sum_{j=1}^{n}\sum_{h=1}^{w_j}\left(f_{jh} + \sum_{k=1}^{s}\sum_{t=0}^{T} a_{jhkt}\lambda_{kt}\right)x_{jh} - \sum_{k=1}^{s}\sum_{t=0}^{T} e_{kt}\lambda_{kt}\ . \tag{9.71}$$

A fentiek csak a modell felső szintjét jelentik, amelyben csak közvetve, az a_{jhkt} számokon keresztül, jelennek meg a műveletek ütemezései és erőforrás-felhasználásai. Az eddig mondottakat ütemezések megkonstruálására akarjuk felhasználni. Ezt a célt szolgálja a célfüggvény (9.71) formája is. Itt ugyanis jól látható egy munka ütemezésének hatása a Lagrange-feladat célfüggvényére. A továbbiakban ezt a tagot akarjuk becsülni. Az α tevékenységhez szükséges erőforrások felhasználása a fenti jelölések mellett

$$u_\alpha(t_\alpha) = \sum_{k=1}^{s} b_{\alpha k} \sum_{t=t_\alpha}^{t_\alpha+p_\alpha-1} \lambda_{kt}$$

értékkel járul hozzá az adott ütemezésnek a célfüggvénybeli együtthatójához, vagyis a

$$\sum_{k=1}^{s}\sum_{t=0}^{T} a_{jhkt}\lambda_{kt}$$

összeghez. Tegyük fel, hogy a j munkából már ütemeztük az $\hat{O}_j \subset O_j$ tevékenységeket. Ekkor az előző modell jelöléseit használva az éppen konstruálandó h ütemezés együtthatója legalább

$$\max\{f_j(t_\alpha + p_\alpha + q_\alpha) : \alpha \in \hat{O}_j\} + \sum_{\alpha\in\hat{O}_j} u_\alpha(t_\alpha)$$

lesz.

A további részleteket elhagyjuk. Az eljárás lényege, amit a fentiekben igyekeztünk demonstrálni, az, hogy ilyen módon az ütemezések konstrukciója során mindvégig becsülni tudjuk a célfüggvény értékét alulról, és így ki lehet szűrni a nem elegendően jó ütemezéseket.

A felsorolt adatokból is látható, hogy bár a kezelhető feladatok mérete állandóan növekszik, azonban számos gyakorlati probléma mérete jóval nagyobb, így érthető, hogy a heurisztikus módszereknek nagy jelentőségük van.

Mielőtt azonban ezekre rátérnénk, a pontos eredmények közül utolsónak Jackson észrevételét adjuk közre, amelyben kiterjesztette Johnson módszerét a kétgépes, többutas probléma megoldására.

9.46. tétel. *Legyen a* $J2 \mid m_j \leq 2 \mid C_{\max}$ *feladat esetén*

- J_1 *azon munkák halmaza, amelyeket csak az első gépen kell megmunkálni,*
- J_2 *azon munkák halmaza, amelyeket csak a második gépen kell megmunkálni,*
- J_{12} *azon munkák halmaza, amelyeket először az első gépen, utána a második gépen kell megmunkálni,*
- J_{21} *azon munkák halmaza, amelyeket először a második gépen, utána az első gépen kell megmunkálni.*

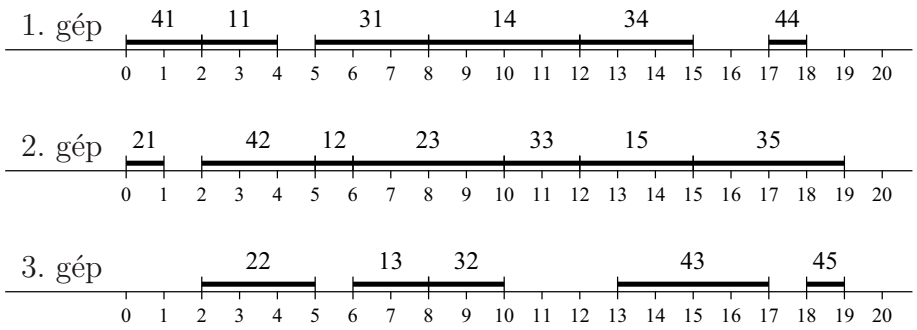
Ekkor a feladat egy optimális ütemezése a következő:

- *az első gépen először a* J_{12}*-beli munkák itteni műveleteit végezzük el a Johnson-algoritmus szerinti sorrendben, utána a* J_1*-beli műveleteket tetszőleges sorrendben, végül a* J_{21}*-beli munkák itteni műveleteit ugyancsak a Johnson-algoritmus szerinti sorrendben.*
- *a második gépen először a* J_{21}*-beli munkák itteni műveleteit végezzük el a Johnson algoritmus szerinti sorrendben, utána a* J_2*-beli műveleteket tetszőleges sorrendben, végül a* J_{12}*-beli munkák itteni műveleteit ugyancsak a Johnson-algoritmus szerinti sorrendben.*

Megjegyzés. Johnson algoritmusát külön-külön alkalmazzuk a J_{12}- és J_{21}-beli munkákra. Azt szokás mondani, hogy a J_1- és J_2-beli munkák esetén csak egy művelet van. Nem jelent megszorítást azonban, ha akárhány műveletet is megengedünk, mint azt látni fogjuk. Annyi azonnal látható, hogy akárhány műveletet is kelljen elvégezni egy munkán, ezek megtehetők egyetlen menetben, hiszen mihelyst véget ért egy művelet, a munkadarab azonnal feltehető ugyanarra a gépre.

Bizonyítás. Tegyük fel, hogy az első gépen a műveletek sorrendje nem azonos az állításbelivel. Ha két szomszédos művelet közül

- az első J_1-beli, a második J_{12}-beli, akkor ezek felcserélése nem változtatja meg az állásidőt az első gépen és nem növeli a második gépen,
- az első J_{21}-beli, a második J_{12}-beli, akkor ezek felcserélése nem növeli az állásidőt egyik gépen sem,
- mindkettő J_{12}-beli, de nem a Johnson algoritmus szerinti sorrendben vannak, akkor ezek felcserélése nem változtatja meg az állásidőt az első gépen és nem növeli a második gépen.



9.9. ábra. Egy optimális megoldás, mely aktív ütemezés.

Tehát az állításban szereplő sorrend az első gépen semmilyen más sorrendnél sem rosszabb. Hasonló átrendezések végezhetők szükség esetén a második gépen is. ■

9.6.4. Heurisztikus módszerek

A továbbiakban rátérünk a heurisztikus módszerek tárgyalására.

Mindvégig feltesszük, hogy a célfüggvény reguláris. Az alább bevezetendő fogalmak ilyenekhez előállítandó közelítő megoldások leírására szolgálnak.

9.47. definíció. *Egy ütemezést* ***aktívnak*** *nevezünk, ha egyetlen műveletet sem lehet időben előbbre tolni anélkül, hogy valamely más művelet ütemezését hátrébb kellene csúsztatni.*

9.9. példa. A 9.8. példában megadott optimális ütemezés nem aktív, mert a 22 műveletet már $t = 1$-ben meg lehet kezdeni. Hasonlóképpen 43 előbbre hozható a 10 időpontig és ezáltal 44 és 45 is előbbre csúsztatható két egységgel. Végül a 31 művelet is megkezdhető már ($t = 4$)-ben. Így a **??**. ábrán látható ütemezéshez jutunk.

Reguláris célfüggvény esetén mindig van aktív optimális megoldás, mert egy kezdési időpont előbbre hozása egy másik kezdési időpont növelése nélkül nem növelheti a célfüggvény értékét. Az aktív ütemezések úgy is felfoghatók, hogy először meghatározzuk a műveletek sorrendjét a gépeken, aztán minden művelet kezdési időpontját a lehető legkorábbinak választjuk úgy, hogy a gépeken előírt sorrend ne sérüljön meg.

Az aktív ütemezések egy fontos alosztálya a következő:

9.48. definíció. *Egy ütemezés* ***állásidő nélküli,*** *ha minden gép dolgozik, ha egyáltalán van elvégezhető művelet.*

9.10. példa. Az előző példában szereplő módosított optimális megoldás nem állásidő nélküli, mert a 43 művelet már $t = 5$-ben rendelkezésre áll. Nem előnyös azonban ekkor megkezdeni, mert akkor mind a 13, mind a 32 műveletet kitolnánk, és így végső soron megnövelnénk a teljes átfutási időt.

A példa mutatja, hogy az állásidő nélküli ütemezés nem ad mindig optimális megoldást. Az irodalom mégis nagy figyelmet szentel neki, mint természetes heurisztikának, amely különösen nagyméretű feladatoknál jó közelítő megoldásokat ad.

Az állásidő nélküli ütemezések előállítása általában egy ú.n. prioritási szabályon alapszik. Ilyeneket láttunk egyetlen gép és párhuzamos berendezések esetében, az utóbbinál lista szerinti ütemezés volt a neve. A leggyakrabban alkalmazott szabályok

- az SPT sorrend,
- az LPT sorrend,
- a legkevesebb megmaradt megmunkálás (LWR), ahol munkánként megvizsgálják a még hátralévő megmunkálások összidejét, és annak a munkának a soron következő megmunkálását választják, ahol az összidő a legrövidebb,
- a leghosszabb megmaradt megmunkálás (rövidítve MWR) az előzőnek a fordítottja,
- a hátralévő műveletek maximális száma (MOPRNR),
- a FIFO sorrend,
- az EDD sorrend.

A második kettő az első kettő általánosításának tekinthető. Általában az a helyzet, hogy ha van egy szabály, akkor annak az ellenkezőjét is használják bizonyos esetekben. Például az említett FIFO sorrend mellett előfordul a LIFO sorrend alkalmazása is.

Egy prioritási szabályon alapuló mohó elindítási módszer az alábbi két követelményt teljesíti:

1. Ha egy munkadarab megérkezik egy éppen nem dolgozó gép elé, akkor a gép azonnal megkezdi a megmunkálását.
2. Ha egy megmunkálás véget ér, akkor a munkadarab azonnal megérkezik a technológiai útvonalban soron következő gép elé. A most felszabaduló gép pedig az előtte várakozó munkadarabok közül a prioritási szabály alapján kiválasztott megfelelő műveletét kezdi meg azonnal.

9.11. példa. Nézzük meg, hogy mit ad a mohó módszer az MWR prioritási szabály alapján a 9.8. példa feladatára. A prioritási szabályhoz felhasználandó értékek műveletenként a következők:

$$
\begin{array}{lllll}
pr_{11} = 12 & pr_{12} = 10 & pr_{13} = 9 & pr_{14} = 7 & pr_{15} = 3 \\
pr_{21} = 8 & pr_{22} = 7 & pr_{23} = 4 & & \\
pr_{31} = 14 & pr_{32} = 11 & pr_{33} = 9 & pr_{34} = 7 & pr_{35} = 4 \\
pr_{41} = 11 & pr_{42} = 9 & pr_{43} = 6 & pr_{44} = 2 & pr_{45} = 1
\end{array}
$$

Az ütemezés elkészítését az alábbi táblázat foglalja össze. Ha ebben a munkák alatt egyetlen szám található, akkor a munka éppen azon a gépen van. Ha pedig például $1v12$ áll egy munka oszlopában, akkor ez azt jelenti, hogy a munka az első gép előtt vár 12 prioritási értékkel.

idő	1. munka	2. munka	3. munka	4. munka	1. gép	2. gép	3. gép
1.	1*v*12	2	1	1*v*11	31	21	–
2.	1*v*12	3	1	1*v*11	31	–	22
3.	1	3	3*v*11	1*v*11	11	–	22
4.	1	2	3	1*v*11	11	23	32
5.	2*v*10	2	3	1	41	23	32
6.	2*v*10	2	2*v*9	1	41	23	–
7.	2*v*10	2	2*v*9	2*v*9	–	23	–
8.	2	–	2*v*9	2*v*9	–	33	13
9.	3	–	2	2*v*9	–	33	13
10.	3	–	2	2*v*9	–	33	13
11.	1	–	1*v*7	2	14	42	–
12.	1	–	1*v*7	2	14	42	–
13.	1	–	1*v*7	2	14	42	–
14.	1	–	1*v*7	3	14	–	43
15.	2	–	1	3	34	15	43
16.	2	–	1	3	34	15	43
17.	2	–	1	3	34	15	43
18.	–	–	2	1	44	35	–
19.	–	–	2	3	–	35	45
20.	–	–	2	–	–	35	–
21.	–	–	2	–	–	35	–

Ha a cél a teljes átfutási idő minimalizálása, akkor ez az eredmény nem olyan jó, mint az optimális megoldásé, de értéke közel van hozzá.

Érdemes összehasonlítani a pontos módszereket a fenti heurisztikus módszerekkel. A pontos módszerek az egész termelő rendszerre, ide értve most a termékeket is, vonatkozó teljes információt követelnek meg, hiszen csak így lehet a megfelelő modellt felállítani. Ezzel szemben a mohó módszer valós idejű eljárásnak tekinthető, hiszen mindig csak akkor dönt, azaz választ ki egy műveletet, ha erre szükség van. Ehhez nem használ mást, mint a rendszer pillanatnyi állapotának ismeretét.

Tehát teljesen figyelmen kívül hagyja azokat a műveleteket, amelyek az adott pillanatban nem végezhetők el, függetlenül attól, hogy ennek mi az oka, azaz, hogy a megelőző műveletek még nem fejeződtek be, vagy a munkadarab még meg sem érkezett a rendszerbe, és függetlenül attól is, hogy ezek későbbi megjelenése milyen hatást okoz. Ennek megvan az az előnye, hogy azonnal reagálni képesek váratlan eseményekre. Például egy munkadarab meghibásodása esetén az ezen végzendő további műveletek egyszerűen nem jelennek meg, és így nincs szükség az előre elkészített sorrendek átdolgozására. Ez a módszer tulajdonképpen nem más, mint a rendszer működésének szimulációja, ahol az egyes gépek irányítását a műveletek szintjén egy nagyon egyszerű szabály határozza meg.

A mohó módszer hátrányaként említhető, hogy egy prioritási szabály bizonyos célokat jól szolgál, ugyanakkor nagyon rossz eredményeket adhat más vonatkozásban. Egy tipikus példa erre, hogy az SPT szabály kiváló az átlagos átfutási idő minél alacsonyabb értéken való tartására, ugyanakkor számos esetben a határidők súlyos megsértéséhez vezet.

A többutas probléma kezelésének fenti két megközelítésében külön-külön rejlő nehézségek leküzdésének egy természetes módja a *kétszintes modell,* ahol a felső szinten a többutas probléma egy diszkrét programozási modellje található. Elképzeléseik szerint ezt a problémát csak ritkán, például naponta vagy hetente oldják meg. Az alatta lévő szinten dön-

tik el gépenként, hogy a gép előtt várakozó műveletek sorából éppen melyiket végezzék el. A felső szint eredményei alapján számolnak költségeket minden művelet minden (diszkrét) időpontban való megkezdésére. Ezek a költségek fejezik ki az alsó szinten az egyes műveletek fontosságát, azaz súlyát. Az éppen elvégzendő műveletnek valamely prioritási szabály szerint első elemét választják. A költségeket pedig ezen feladat optimális Lagrange-szorzói alapján számítják.

Gyakorlatok

9.6-1. Mi a 9.45. tétel párja, ami arra ad feltételt, hogy az α műveletnek az M-beli műveletek előtt kell állnia?

9.6-2. Írjuk fel a $Pm||C_{\max}$ feladat diszjunktív gráf modelljét.

Feladatok

9-1. SPT optimalitása

Bizonyítsuk be, hogy az SPT sorrend optimális megoldása az $1 \parallel \sum L_j$ feladatnak.

9-2. Késésmentes optimális megoldás

Bizonyítsuk be, hogy ha az $1 \mid d_j \mid \sum C_j$ feladatnak van olyan optimális megoldása, amiben nincs késés, akkor ezen megoldásban az α megmunkálás akkor és csak akkor állhat az utolsó helyen, ha $d_\alpha \geq \sum_1^n p_j$ és minden $i : d_i \geq \sum_1^n p_j$ esetén $p_\alpha \geq p_i$.

9-3. Minimális késés maximalizálása

Legyen egy $1 \mid d_j \mid f$ feladat esetén $[1], [2], \ldots, [n]$ olyan sorrend, amelyre teljesül, hogy

$$d_{[1]} - p_{[1]} \leq d_{[2]} - p_{[2]} \leq \cdots \leq d_{[n]} - p_{[n]} .$$

Bizonyítsuk be, hogy ez a sorrend maximalizálja a minimális késést. (*Útmutatás.* Vegyük észre, hogy betervezett állásidő használata nem megengedett. Alkalmazzuk a szokásos „felcseréléses” módszert.)

9-4. Legjobb és legrosszabb lista

Tegyük fel, hogy m azonos gépünk van, amelyeken $2m - 1$ megmunkálást kell elvégezni. A megmunkálási idők közül egy értéke m, további $m - 1$ értéke $m - 1$ és még további $m - 1$ értéke 1. Adjuk meg a legjobb és a legrosszabb értéket adó listát.

9-5. LPT-korlát élessége

Bizonyítsuk be, hogy ha m gép van és a megmunkálási idők $2m - 1, 2m - 1, 2m - 2, 2m - 2, \ldots, m + 1, m + 1, m, m, m$, akkor tetszőleges m esetén az LPT ütemezés pontossága a 9.23. tételben megadott felső korláttal egyezik meg.

9-6. Azonos párhuzamos gépek.

Általánosítsuk a 9.4-2. gyakorlat eredményét két gép és tetszőleges m mellett a $2m-1$, $2m-1$, $2m-2$, $2m-2$, $\ldots$, $m+1$, $m+1$, m, m, m megmunkálási idők esetére.

9-7. Egyutas ütemezési feladat

Oldjuk meg az alábbi egyutas ütemezési feladatot. Ábrázoljuk az optimális megoldást Gantt-diagrammal.

munka	1. gép	2. gép	3. gép
1.	3	22	2
2.	22	20	20
3.	20	14	18

9-8. Kétgépes feladat
Bizonyítsuk be, hogy ha az $F3 \parallel C_{\max}$ probléma esetén a

$$\min_j t_{1j} > \max_j t_{2j}$$

és a

$$\min_j t_{3j} > \max_j t_{2j}$$

feltételek legalább egyike teljesül, akkor a feladat optimális megoldása ugyanaz, mint azé a kétgépes feladaté, ahol a megmunkálási idők munkánként

$$t_{1j} + t_{2j}, \quad t_{1j} + t_{2j}.$$

Útmutatás. Alkalmazzuk a 9.34. tétel bizonyításának módszerét.
9-9. Többutas ütemezés
Tekintsük a többutas ütemezési feladatot. Bizonyítsuk be, hogy ha M páronként diszjunktív műveletek egy tetszőleges halmaza és α egy további művelet, mely diszjunktív valamennyi M-belivel, végül $C > 0$ egy tetszőleges szám, akkor

$$\min_{\beta \in M} r_\beta + \sum_{\beta \in M} p_\beta + p_\alpha + q_\alpha \geq C$$

esetén α egyetlen olyan ütemezésben sem követheti valamennyi M-beli műveletet, amelyben a teljes átfutási idő kevesebb, mint C.
9-10. Diszjunkt gráf modell
Tegyük fel, hogy egy, a diszjunktív gráf modellen alapuló eljárás során a konjunktív élek pillanatnyi halmaza B. Tekintsünk egy diszjunktív $\{(\alpha,\beta),(\beta,\alpha)\}$ élpárt, amelyiknek egyik tagja sincs B-ben. Adjunk egyszerű elegendő feltételt arra, hogy az élpár egyik tagjának hozzá vétele se hozzon létre kört a gráfban.
9-11. Többutas probléma célfüggvényei
A többutas probléma (9.58)–(9.62) modellje esetén írjuk fel a szükséges feltételek és változók bevezetésével a $\sum w_j T_j$ és a $\sum w_j U_j$ célfüggvényeket.

Megjegyzések a fejezethez

Az ütemezési algoritmusok tulajdonságainak jó összefoglalója Bruckner [61] 2002-ben megjelent monográfiája.

Az ütemezési feladatok tömör leírásának rendszere Grahamtől [170] és Lawlertől [282] származik.

A Gantt-diagramok használatát Henry Laurence Gantt javasolta 1903-ban.

Az $1 \mid r_j, d_j \mid L_{max}$ feladat esetében már nem ilyen egyszerű a helyzet, mert a probléma

teljes általánosságában NP-teljes.A 9.5. tétel forrása [214, 420].

Az SPT sorrend fogalmát Schmidt [434] vezette be.

A 9.7. tétel alapjául szolgáló algoritmus megtalálható Lawler cikkében [281].

Az 9.15. és 9.16. tételek a témakör legkorábbi eredményeihez tartoznak és McNaughton cikkében [319] találhatók meg.

Az 9.18. tétel bizonyítása Horntól [210] származik.

A 9.24. tétel, a 9.25. tételt megelőző elemzés, valamint a 9.27. tétel forrása Coffman, Garey és Johnson cikke [83]. A 9.27. tételben szereplő korlátot Friesen [139] 1.20-ra javította. A módszert magát Friesen és Langston [140] finomították. Ennek a változatnak az ismert hibakorlátja már $72/61 + 2^{-k}$. Bár a szükséges műveletek számának nagyságrendje nem változott, az a konstans, amit a nagy ordó tartalmaz, igen nagy. A további munkák közül megemlítjük még Sahni cikkét [404], ahol a korlát $1 + 1/k$, de a műveletek száma $O(n(n^2k)^{m-1})$. Igaz továbbá, hogy minden m esetén $r_m > 1$ (lásd a 9-7. feladatot).

A $P \mid pmtn, d_j \mid U_{\max}$ feladatra, vagyis ahol csak a határidőket betartó, megengedett ütemezés megtalálása a feladat, ha ilyen létezik, illetve annak kimutatása, ha nem létezik, Sahni [405] javasolt $O(n \log nm)$ futási idejű algoritmust, amely olyan megoldást állít elő (ha van megoldás), amelyben legfeljebb $n - 2$ megmunkálást kell megszakítani.

Az $U_{\max}$ célfüggvényre vonatkozó, a 9.19. tételhez hasonló állítás Hujter Mihálytól [214] származik.

Az első, dinamikus programozással elért ütemezési eredmény Rothkopf [401] nevéhez fűződik. Lawler és társai [282] később egyszerűsítették a Rothkopf által használt képleteket.

A $C(L)/C^*$ hányadosra vonatkozó első eredményt – a 9.22. tételt – Graham bizonyította 1966-ban [168]. Ugyancsak Grahamtől származik a 9.23. tétel.

A ládapakolási algoritmusokra vonatkozó eredmények összefoglalása megtalálható Coffman, Csirik János és Woeginger [84] cikkében.

A korlátozás és szétválasztás részletes leírása megtalálható Vizvári Béla jegyzetében [479]. Az egyutas ütemezési problémával részletesen foglalkoznak a [180, 273, 318, 455, 453, 454] dolgozatok.

Az egyutas problémának a (9.42) képlethez kapcsolódó változatát Piehler [370] tanulmányozta.

A diszjunktív gráf modell először a [403] dolgozatban jelent meg.

A (9.47)–(9.50) képletek közvetlenül adódnak [479] 5. fejezete alapján.

Az irodalomban ismert alsó korlátok egy nagyon jó összefoglalását adja [272]. A (9.52) korlátot gyengébb formában, a harmadik tag nélkül közli [73] és [419]. Maga a korlát élesíthető a 9.4. tétel alapján. A (9.53) korlát először a [224] cikkben fordult elő, míg (9.54) megfordítása [59]-ben.

A 9.45. tétel alapja [68].

A kritikus blokk fogalmát Grabowski [167] vezette be. A kritikus blokkal kapcsolatos további részletek találhatók a [32] dolgozatban.

Az egészértékű programozás első modellje [310] alapján terjedt el a szakirodalomban. A másik tárgyalt modell több vonatkozásban általánosabb, és tekintettel tud lenni különböző erőforrásokra is [129], [130]. Ebben a fejezetben csak a modellt tárgyaljuk – a Fisher által javasolt módszer részletesen megtalálható a [479] könyvben.

A (9.64)–(9.67) feladat célfüggvényének elemzése [479] 7. fejezetéből származik.

Az általános egészértékű programozási feladatra vonatkozóan eddig említett módszerek csak szerény méretű feladatokat képesek egzaktul megoldani. [27] még nem számol

be számítástechnikai eredményekről. [129] esetében a legnagyobb vizsgált problémában a munkák száma 5, az erőforrásoké 4. [272] mindössze három feladatot oldott meg, melyek közül a legnagyobb esetén a műveletek száma 36 és mind a munkák, mind a gépek száma 6. [32] már tovább növelte a méretet, a pontosan megoldott legnagyobb feladatokban 8 gép és 64 művelet volt. [68] az, amelyben először oldottak meg egy, az irodalomból jól ismert, és állandó kihívást jelentő 10 munkát és 10 gépet tartalmazó feladatot, továbbá ugyanez a dolgozat beszámol számos, a korábbiaknál nagyobb probléma, például 20 munka, 5 gép és számos művelet megoldásáról is.

Johnson módszerének a kétgépes, többutas problémára való kiterjesztése Jackson [225] érdeme.

A többutas probléma kezelésére Roundy [402] javasolta a kétszintes modellt.

Az on-line algoritmusokra vonatkozó eredmények összefoglalása megtalálható ennek a könyvnek a második kötetében.

Az ütemezéselmélet egyik fontos alkalmazási területe a számítógépes feladatmegoldás. Ezzel kapcsolatban ajánljuk Blazewicz és társai [50], Coffman [82] és Pinedo [371, 372] monográfiáit. Pinedo két ajánlott könyve a determinisztikus ütemezés mellett a sztochasztikus ütemezés módszereit és eredményeit is tárgyalja.

A 9-2. feladat forrása [434], a 9-4. feladat [148]-ból, a 9-5. feladat [169]-ből, a 9-8. feladat pedig [232]-ból származik.

V. ADATBÁZISKEZELÉS

Bevezetés

Ebben a részben három témakört tárgyalunk.

Bár a technikai fejlődés egyre nagyobb kapacitású memóriákat eredményez, ma is aktuális feladata az adatok tömörítése. A *tizedik fejezet* az információelméleti alapok összefoglalása után bemutatja az aritmetikai kódolást, a Ziv–Lempel-tömörítést, a Burrows–Wheeler-transzformációt és végül a képtömörítés témakörét.

A *tizenegyedik fejezet* témája a memóriagazdálkodás, azon belül a particionálás, lapozás, anomáliák és állományok optimális elhelyezése.

A *tizenkettedik fejezet* pedig a relációs adatbázisok tervezéséről szól: az egyes alfejezetek a funkcionális függőségeket, a relációs sémák szétvágását és az általános függőségeket elemzik.

10. Adattömörítés

Az adattömörítő algoritmusok általános működési elve a következő. Egy véges ábécé jeleiből felépülő szöveget alakítanak bitsorozattá, kihasználva azt a tulajdonságot, hogy az ábécé jelei különböző gyakorisággal fordulnak elő a szövegben. Például az „e” betű gyakoribb a „q” betűnél, ezért rövidebb kódszót rendelnek az előbbihez. Ekkor a tömörítés minőségét az átlagos kódszóhosszal jellemzik.

A tömörítés alapja egy statisztikai modell, amelynek része egy véges ábécé és egy valószínűségi eloszlás az ábécé jelein. A valószínűségi eloszlás a jelek (relatív) gyakoriságát adja meg. Egy ilyen ábécé-valószínűségi eloszlás párt nevezünk ***forrásnak.*** Először át kell tekintenünk az *információelmélet* néhány alapvető fogalmát és eredményét. A legfontosabb az *entrópia* fogalma, ugyanis a forrás entrópiája meghatározza a tömöríthetőség mértékét, a tömörített sorozat hosszának minimumát.

Az egyik legegyszerűbb és legérthetőbb modell a *diszkrét, emlékezet nélküli forrás.* Ebben a modellben a jelek egymástól függetlenül fordulnak elő a szövegben. Úgynevezett p prefix kódok segítségével a szöveget a forrás entrópiájával megegyező hosszúságú sorozattá tömöríthetjük. Prefix kód esetén egyetlen kódszó sem kezdőszelete egy másik kódszónak. A következőkben majd részletesen vizsgáljuk ezt a modellt. A tömöríthetőség mértékét a Kraft-egyenlőtlenség adja meg. Az egyenlőtlenség élességét a Huffman-kódolással mutatjuk meg, amelyről belátható, hogy optimális.

A legtöbb gyakorlati alkalmazás nem elégíti ki a diszkrét, emlékezet nélküli forrásra vonatkozó feltételek mindegyikét. Először, ez a forrás modell rendszerint nem valósághű, ugyanis a jelek nem egymástól függetlenül fordulnak elő a szövegben. Másodszor, a valószínűségi eloszlás előre nem ismert, ezért a tömörítő algoritmusoknak univerzálisan kellene működniük a valószínűségi eloszlások teljes osztályára vonatkozóan. Az ilyen univerzális tömörítések elemzése sokkal összetettebb, mint a diszkrét, emlékezet nélküli források elemzése, ezért csak bemutatjuk az algoritmusokat, de nem elemezzük a tömörítési hatékonyságukat. Az univerzális tömörítéseket alapvetően két csoportba sorolhatjuk.

A statisztikai tömörítések a következő jel előfordulási valószínűségét becslik lehetőség szerint minél pontosabban. Ezt a folyamatot nevezzük a ***forrás modellezésének.*** A valószínűségek kellő ismeretében a szöveget tömörítik, rendszerint aritmetikai kódolással. Aritmetikai kódolás esetén a valószínűségeket egy intervallummal ábrázolják, és ezt az intervallumot kódolják.

A szótár alapú tömörítések olyan mintákat tárolnak egy szótárban, amelyek előzőleg már előfordultak a szövegben, és a minta következő előfordulását a szótárban elfoglalt pozíciójával kódolják. Az ilyen típusú eljárások közül kiemelkedik Ziv és Lempel algoritmusa.

Bemutatunk egy harmadik univerzális kódolási módszert, amelyik a fenti osztályok egyikébe sem tartozik. Az algoritmust Burrows és Wheeler tette közzé, és az utóbbi években egyre jobban elterjedt a használata, mivel az erre épülő megvalósítások nagyon hatékonyak a gyakorlatban.

Az összes eddigi algoritmus ***veszteségmentes,*** azaz nem vesztünk információt, amikor a tömörített szöveget dekódoljuk, vagyis pontosan az eredeti szöveget kapjuk vissza hiba nélkül. Ezzel szemben vannak a ***veszteséges tömörítések,*** amelyek esetén a visszakapott szöveg nem teljesen egyezik meg az eredetivel. Veszteséges tömörítéseket alkalmaznak például kép, hang, videó vagy beszéd tömörítése esetén. A veszteség természetesen nem befolyásolhatja lényegesen a minőséget. Például az emberi szem vagy fül által nem érzékelhető tartományok (frekvenciák) elhagyhatók. Ugyanakkor ezen technikák megismeréséhez nélkülözhetetlen a kép-, hang-, beszédfeldolgozás alapos ismerete. Ezek az ismeretek meghaladják a könyv kereteit, ezért a képtömörítő algoritmusok kötzül csak a JPEG alapelemeit tárgyaljuk.

A hangsúlyt a legújabb eredményekre, mint például a Burrows–Wheeler-transzformáció és a környezetfa súlyozó módszer, helyezzük. Pontos bizonyításokat csak a diszkrét, emlékezet nélküli forrás esetén adunk. Ez a legjobban elemzett, de nem túl gyakorlatias modell. Ugyanakkor ez az alapja a bonyolultabb forrásmodelleknek is, amelyekben a számítások során feltételes valószínűségeket kell használni. A tömörítő algoritmusok aszimptotikus számítási bonyolultsága gyakran a szöveg hosszával lineárisan arányos, mert az algoritmusok egyszerűen átolvassák a szöveget. A gyakorlati megvalósítások futási idejét ugyanakkor jelentősen befolyásolják olyan konstansok, mint a szótár mérete Ziv–Lempel-tömörítés, vagy a környezetfa mélysége aritmetikai tömörítés esetén. A tömörítő algoritmusok további, pontos elemzése vagy összehasonlítása gyakran erősen függ a forrás szerkezetétől, a tömörítendő fájl típusától. Ennek megfelelően a tömörítő algoritmusok hatékonyságát szintmérő fájlokon tesztelik. A legismertebb szintmérő fájl gyűjtemények a Calgary Corpus és a Canterbury Corpus.

10.1. Információelméleti eredmények

10.1.1. Diszkrét, emlékezet nélküli forrás

Ebben a részben a ***diszkrét, emlékezet nélküli forrással*** (DMS) foglalkozunk. Egy ilyen forrás egy $(\mathcal{X}, P)$ pár, amelyben $\mathcal{X} = \{1, \ldots, a\}$ egy véges ábécé, és $P = (P(1), \ldots, P(a))$ egy valószínűségi eloszlás $\mathcal{X}$-en. Egy diszkrét, emlékezet nélküli forrás leírható egy X valószínűségi változóval is, ahol $\Pr\{X = x\} = P(x)$ minden $x \in \mathcal{X}$ jelre. Egy $x^n = (x_1 x_2 \ldots x_n) \in \mathcal{X}^n$ szó az $(X_1, X_2, \ldots, X_n)$ valószínűségi változó egy értéke, amelyben minden X_i azonos eloszlású, független változó. Ennek megfelelően a szó valószínűsége a jelek valószínűségeinek szorzata, azaz $P^n(x_1 x_2 \ldots x_n) = P(x_1) \cdot P(x_2) \cdot \cdots \cdot P(x_n)$.

A jelek gyakoriságát természetes nyelvekben statisztikai módszerekkel becsüljük. Például az angol nyelv esetén, ha $\mathcal{X}$ a latin ábécé kiegészítve egy jellel, amely megfelel a szóköznek és egyéb elválasztó jeleknek, a valószínűségi eloszlást megkaphatjuk a 10.1. ábrán látható gyakoriság táblázat alapján. Így $P(A) = 0.064$, $P(B) = 0.014$ stb.

Vegyük észre, hogy ez a forrás modell legtöbbször nem valósághű. Például angol nyelvű szövegekben a ‘th’ pár gyakorisága nagyobb, mint a ‘ht’ páré. Ez nem fordulhat elő, ha az

A	64	H	42	N	56	U	31
B	14	I	63	O	56	V	10
C	27	J	3	P	17	W	10
D	35	K	6	Q	4	X	3
E	100	L	35	R	49	Y	18
F	20	M	20	S	56	Z	2
G	14	T	71				

szóköz/elválasztó jel 166

10.1. ábra. Jelek gyakorisága egy 1000 jelet tartalmazó angol szövegben.

angol szöveget diszkrét, emlékezet nélküli forrással modellezzük, ekkor ugyanis $P(th) = P(t) \cdot P(h) = P(ht)$.

A fejezet bevezetésében rámutattunk arra, hogy a tömörítés során az eredeti adatot egy bináris kód segítségével rövid bitsorozattá alakítjuk. A bináris kód egy $c : \mathcal{X} \longrightarrow \{0,1\}^* = \bigcup_{n=0}^{\infty} \{0,1\}^n$ leképezés, amely minden $x \in \mathcal{X}$ elemhez egy $c(x)$ kódszót rendel. A tömörítés során a kódszavak átlagos hosszát szeretnénk minimalizálni. Belátható, hogy a legjobb tömörítést a P valószínűségi eloszlás $H(P)$ ***entrópiájával*** jellemezhetjük. Az entrópiát a következő képlet adja meg

$$H(P) = -\sum_{x \in \mathcal{X}} P(x) \cdot \lg P(x) \ .$$

A $H(x)$ jelölést is használni fogjuk, annak megfelelően, ahogy a forrást valószínűségi változóként értelmeztük.

10.1.2. Prefix kódok

Egy (változó hosszúságú) ***kód*** egy $c : \mathcal{X} \longrightarrow \{0,1\}^*$ függvény, ahol $\mathcal{X} = \{1, \ldots, a\}$. Ekkor $\{c(1), c(2), \ldots, c(a)\}$ a ***kódszavak*** halmaza, amelyben egy $x = 1, \ldots, a$ jelhez a $c(x) = (c_1(x), c_2(x), \ldots, c_{L(x)}(x))$ kódszó tartozik. $L(x)$ a $c(x)$ kódszó ***hossza***, azaz a $c(x)$-et leíró bitek száma.

A következő példában a latin ábécéhez rendelhető bináris kódokat adunk meg (SP = szóköz).

$\overline{c} : a \longrightarrow 1, b \longrightarrow 10, c \longrightarrow 100, d \longrightarrow 1000, \ldots, z \longrightarrow \underbrace{10\ldots0}_{26}, SP \longrightarrow \underbrace{10\ldots0}_{27}.$

$\hat{c} : a \longrightarrow 00000, b \longrightarrow 00001, c \longrightarrow 00010, \ldots, z \longrightarrow 11001, SP \longrightarrow 11010.$

$\hat{c}(x)$ nem más, mint az x jel ábécébeli indexének bináris alakja.

$\check{c} : a \longrightarrow 0, b \longrightarrow 00, c \longrightarrow 1, \ldots$ (a többi kódszóra nem lesz szükségünk a következő elemzésben).

Az utolsó, $\check{c}$ kód rendelkezik egy nemkívánatos tulajdonsággal. Vegyük észre, hogy 00 megfeleltethető akár b-nek, akár aa-nak. Ezért a $\check{c}$ kóddal tömörített szövegek nem egyértelműen dekódolhatók.

Egy c kód ***egyértelműen dekódolható*** (UDC), ha minden $\{0,1\}^*$-beli sorozat legfeljebb egy kódszó sorozatnak feleltethető meg.

A $\overline{c}$ kód egyértelműen dekódolható, hiszen két 1-es közötti 0-k száma meghatározza a $\overline{c}$ által kódolt következő jelet. A $\hat{c}$ kód is egyértelműen dekódolható, mert minden jelet pontosan öt bittel kódolunk. Ennek megfelelően a bitsorozat első öt bitjéből megkapjuk az eredeti szöveg első jelét, a következő öt bitből a második jelet stb.

Egy c kód ***prefix kód,*** ha bármelyik $c(x)$ és $c(y)$ kódszó párra, amelyre $x \neq y$ és $L(x) \leq L(y)$, $(c_1(x), c_2(x), \ldots, c_{L(x)}(x)) \neq (c_1(y), c_2(y), \ldots, c_{L(x)}(y))$ teljesül. Azaz a $c(x)$ és $c(y)$ kódszavak első $L(x)$ bitjében legalább egy bit eltérő.

A prefix kóddal tömörített szövegek egyértelműen dekódolhatók. A dekódolás során addig olvasunk biteket, amíg egy $c(x)$ kódszóhoz nem jutunk. Miután $c(x)$ nem lehet egy másik kódszó kezdete, csak $x \in \mathcal{X}$ jelnek felelhet meg. Ezután újabb biteket olvashatunk, amíg elő nem áll a következő kódszó. Ezt az eljárást folytathatjuk a bitsorozat végéig. A $c(x)$ kódszó megtalálásának pillanatában a dekódoló tudja, hogy $x \in \mathcal{X}$ az eredeti szöveg következő jele. Ezen tulajdonság miatt szokás a prefix kódot ***azonnali kódnak*** is nevezni. Vegyük észre, hogy $\overline{c}$ kód nem rendelkezik ezzel a tulajdonsággal, mivel minden kódszó a következő jelhez tartozó kódszó kezdőszelete.

Az adattömörítés követelménye a kódszavak átlagos hosszának minimalizálása. Adott $(\mathcal{X}, P)$ forrás esetén, ahol $\mathcal{X} = \{1, \ldots, a\}$ és $P = (P(1), P(2), \ldots, P(a))$ egy valószínűségi eloszlás $\mathcal{X}$-en, az $\overline{L}(c)$ ***átlagos hossz*** definíciója

$$\overline{L}(c) = \sum_{x \in \mathcal{X}} P(x) \cdot L(x) \ .$$

Ha egy angol nyelvű szövegben az összes jel ugyanolyan gyakorisággal fordulna elő, akkor a $\overline{c}$ kód átlagos hossza $(1/27)(1+2+\cdots+27) = (1/27)\cdot(27\cdot 28)/2 = 14$ lenne. Ekkor a $\hat{c}$ kód jobb lenne, mert ennek átlagos hossza 5. Tudjuk, hogy az angol szövegekben előforduló jelek relatív gyakoriságát nem modellezhetjük egyenletes eloszlással (10.1 ábra). Ezért jobb kódot készíthetünk, ha a gyakori jelekhez rövid kódot rendelünk, amint azt a következő c kód szemlélteti, amelynek átlagos hossza $\overline{L}(c) = 3 \cdot 0.266 + 4 \cdot 0.415 + 5 \cdot 0.190 + 6 \cdot 0.101 + 7 \cdot 0.016 + 8 \cdot 0.012 = 4.222$.

$a \longrightarrow 0110$,	$b \longrightarrow 010111$,	$c \longrightarrow 10001$,	$d \longrightarrow 01001$,
$e \longrightarrow 110$,	$f \longrightarrow 11111$,	$g \longrightarrow 111110$,	$h \longrightarrow 00100$,
$i \longrightarrow 0111$,	$j \longrightarrow 11110110$,	$k \longrightarrow 1111010$,	$l \longrightarrow 01010$,
$m \longrightarrow 001010$,	$n \longrightarrow 1010$,	$o \longrightarrow 1001$,	$p \longrightarrow 010011$,
$q \longrightarrow 01011010$,	$r \longrightarrow 1110$,	$s \longrightarrow 1011$,	$t \longrightarrow 0011$,
$u \longrightarrow 10000$,	$v \longrightarrow 0101100$,	$w \longrightarrow 001011$,	$x \longrightarrow 01011011$,
$y \longrightarrow 010010$,	$z \longrightarrow 11110111$,	$SP \longrightarrow 000$.	

Még ennél is hatékonyabban tömöríthetünk, ha nem különálló jeleket kódolunk, hanem n jelből álló blokkokat, valamilyen $n \in N$ értékre. Ekkor az $(\mathcal{X}, P)$ forrást az $(\mathcal{X}^n, P^n)$ forrással helyettesítjük. A forrás emlékezet nélküli, ezért egy $(x_1 x_2 \ldots x_n) \in \mathcal{X}^n$ szóra $P^n(x_1 x_2 \ldots x_n) = P(x_1) \cdot P(x_2) \cdot \cdots \cdot P(x_n)$. Ha például adott a két jelből álló $\mathcal{X} = \{a, b\}$ ábécé, és $P(a) = 0.9$, $P(b) = 0.1$, akkor a $c(a) = 0$, $c(b) = 1$ kódszavakat tartalmazó c kód átlagos hossza $\overline{L}(c) = 0.9 \cdot 1 + 0.1 \cdot 1 = 1$. Ennél jobb kódot nyilvánvalóan nem találhatunk.

A jelpárok valószínűségei a következők:

$$P^2(aa) = 0.81, \quad P^2(ab) = 0.09, \quad P^2(ba) = 0.09, \quad P^2(bb) = 0.01 \ .$$

Tekintsük a következő c^2 prefix kódot:

$$c^2(aa) = 0, \quad c^2(ab) = 10, \quad c^2(ba) = 110, \quad c^2(bb) = 111 \ .$$

Ennek átlagos hossza $\overline{L}(c^2) = 1{\cdot}0.81{+}2{\cdot}0.09{+}3{\cdot}0.09{+}3{\cdot}0.01 = 1.29$. Így $(1/2)\overline{L}(c^2) = 0.645$ tekinthető az átlagos hossznak, amelyet a c^2 kód egy $\mathcal{X}$ ábécébeli jel esetén használ. Amikor n jelből álló blokkokat kódolunk, a következő értéket kell vizsgálnunk:

$$L(n,P) = \min_{cUDC} \frac{1}{n} \sum_{(x_1,\ldots,x_n)\in\mathcal{X}^n} P^n(x_1\ldots x_n)L(x_1\ldots x_n) = \min_{cUDC} \overline{L}(c) \ .$$

A zajmentes kódolás tételéből, amelyet a következő részben mondunk ki, következik, hogy $\lim_{n\longrightarrow\infty} L(n,P) = H(P)$ az $(\mathcal{X},P)$ forrás entrópiája.

Az eddig példaként használt angol nyelvű szövegek esetén $H(P) \approx 4.19$. Ennek megfelelően a bemutatott kód, amelyben csak különálló jeleket kódoltunk, már majdnem optimális az $L(n,P)$ érték alapján. Hatékonyabb tömörítést adhatunk, ha figyelembe vesszük a jelek közötti függőségeket.

10.1.3. Kraft-egyenlőtlenség és a zajmentes kódolás tétele

Megadunk egy szükséges és elégséges feltételt arra vonatkozóan, hogy egy $\mathcal{X} = \{1,\ldots,a\}$ ábécéhez előre adott $L(1),\ldots,L(a)$ kódszóhosszúságú prefix kód létezzen.

10.1. tétel (Kraft-egyenlőtlenség). *Legyen $\mathcal{X} = \{1,\ldots,a\}$. Akkor és csak akkor létezik $c : \mathcal{X} \longrightarrow \{0,1\}^*$ prefix kód, amelyben a kódszavak hossza $L(1),\ldots,L(a)$, ha*

$$\sum_{x\in\mathcal{X}} 2^{-L(x)} \le 1.$$

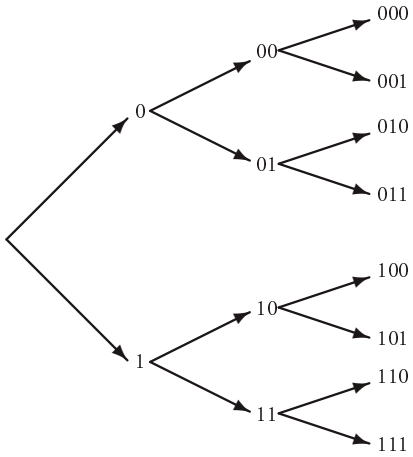
Bizonyítás. A bizonyítás alapötlete, hogy a kódszavakat egy $T = \max_{x\in\mathcal{X}}\{L(x)\}$ mélységű bináris fa csúcsainak tekintjük. A fának teljesnek és regulárisnak kell lennie, azaz a gyökérből levélbe vezető utak összhossza T, és minden belső csúcs fokszáma 3. Ezt szemlélteti a 10.2. ábrán látható fa $T = 3$ esetén.

A fában a gyökértől n távolságra elhelyezkedő csúcsokat az $x^n \in \{0,1\}^n$ kódszavakkal címkézzük meg. Az $x_1x_2\ldots x_n$ címkéjű csúcs felső (bal oldali[1]) gyerekét az $x_1x_2\ldots x_n0$ szóval címkézzük, az alsó (jobb oldali) gyerek címkéje $x_1x_2\ldots x_n1$.

Az $x_1x_2\ldots x_n$ címkéjű csúcs ***árnyéka*** az összes olyan levél halmaza, amelyek (T hosszúságú) címkéje az $x_1x_2\ldots x_n$ szóval kezdődik. Másképpen, az $x_1x_2\ldots x_n$ csúcs árnyéka olyan csúcsokat tartalmaz, amelyek címkéjének prefixe $x_1x_2\ldots x_n$. A 10.2. ábrán látható példában a 0 címkéjű csúcs árnyéka $\{000,001,010,011\}$.

Tegyük fel, hogy adott egy prefix kód, amelyben a kódszavak hossza $L(1),\ldots,L(a)$. Minden kódszó megfelel a T mélységű bináris fa egy csúcsának. Vegyük észre, hogy bármely két kódszó árnyéka diszjunkt halmaz. Ha nem így lenne, találhatnánk egy $x_1x_2\ldots x_T$

[1] Ha a fát más elrendezésben, felülről-lefelé rajzoljuk.



10.2. ábra. Egy lehetséges kódfa.

szót, amelynek két kódszó is a prefixe. A kódszavak hossza legyen s és t, és feltehetjük, hogy $s < t$. Ekkor a két kódszó $x_1x_2\ldots x_s$ és $x_1x_2\ldots x_t$, amelyek közül az első nyilvánvalóan prefixe a másodiknak.

A $c(x)$ kódszó árnyékának elemszáma $2^{T-L(x)}$. A T hosszúságú kódszavak száma 2^T. Az árnyékok elemszámait összegezve kapjuk, hogy

$$\sum_{x\in X} 2^{T-L(x)} \le 2^T \ ,$$

hiszen bármely T hosszúságú kódszó legfeljebb egy árnyék eleme lehet. Mindkét oldalt 2^T-vel osztva a kívánt egyenlőtlenséget kapjuk.

Fordítva, tegyük fel, hogy adottak az $L(1),\ldots,L(a)$ pozitív egészek. Tegyük fel továbbá, hogy $L(1) \le L(2) \le \cdots \le L(a)$. Az első kódszó legyen

$$c(1) = \underbrace{00\ldots0}_{L(1)} \ .$$

Mivel

$$\sum_{x\in X} 2^{T-L(x)} \le 2^T$$

fennáll, $2^{T-L(1)} < 2^T$ is teljesül. (Különben csak egyetlen jelet kellene kódolni.) Így maradt csúcs a T-edik szinten, amelyik nem tartozik $c(1)$ árnyékába. Válasszuk ki ezek közül az elsőt, és menjünk vissza a gyökér felé $T-L(2)$ lépést. $L(2) \ge L(1)$, ezért egy olyan csúcshoz jutunk, amelynek címkéje $L(2)$ bit hosszú, így az nem lehet $c(1)$ prefixe. Ezért ezt a címkét választhatjuk $c(2)$ kódszónak. Ha $a = 2$, akkor kész vagyunk. Ha nem, akkor a feltételezés miatt $2^{T-L(1)} + 2^{T-L(2)} < 2^T$, és találhatunk egy csúcsot a T-edik szinten, amelyik nem tartozik sem $c(1)$, sem $c(2)$ árnyékába. Ezután előállíthatjuk a következő kódszót az eddigieknek megfelelően. Az eljárást addig folytathatjuk, amíg az összes kódszót elő nem állítottuk. ■

A Kraft-egyenlőtlenség egy szükséges és elégséges feltétele egy $L(1), \ldots, L(a)$ kódszó hosszúságú prefix kód létezésének. A következő tétellel megmutatjuk, hogy az egyenlőtlenség teljesülése szükséges feltétel egy egyértelműen dekódolható kód létezéséhez. Ezt úgy is értelmezhetjük, hogy elegendő prefix kódokkal foglalkozni, hiszen nem várhatunk jobb eredményt bármilyen más, egyértelműen dekódolható kódtól sem.

10.2. tétel (Kraft-egyenlőtlenség egyértelműen dekódolható kódokra). *Akkor, és csak akkor létezik $L(1), \ldots, L(a)$ kódszó hosszúságú, egyértelműen dekódolható kód, ha*

$$\sum_{x \in \mathcal{X}} 2^{-L(x)} \leq 1 \ .$$

Bizonyítás. Minden prefix kód egyértelműen dekódolható, ezért az elégségesség következik az előző tételből. Vegyük észre, hogy $\sum\limits_{x \in \mathcal{X}} 2^{-L(x)} = \sum\limits_{j=1}^{T} w_j \, 2^{-j}$, ahol w_j a j hosszúságú kódszavak száma az egyértelműen dekódolható kódban és T a maximális kódszóhossz. A kifejezés jobb oldalának s-edik hatványa

$$\left(\sum_{j=1}^{T} w_j \, 2^{-j} \right)^s = \sum_{k=s}^{T \cdot s} N_k \, 2^{-k} \ .$$

Ebben $N_k = \sum\limits_{i_1 + \cdots + i_s = k} w_{i_1} \ldots w_{i_s}$ az összes olyan szöveg száma, amelynek kódolt formája k bit hosszú. A kód egyértelműen dekódolható, ezért minden k hosszúságú sorozat legfeljebb egy szövegnek felelhet meg.

Ugyanakkor $N_k \leq 2^k$, így $\sum\limits_{k=s}^{T \cdot s} N_k \, 2^{-k} \leq \sum\limits_{k=s}^{T \cdot s} 1 = T \cdot s - s + 1 \leq T \cdot s$. s-edik gyököt vonva kapjuk, hogy $\sum\limits_{j=1}^{T} w_j \, 2^{-j} \leq (T \cdot s)^{1/s}$.

Ez az egyenlőtlenség bármely s értékre fennáll, és $\lim\limits_{s \longrightarrow \infty} (T \cdot s)^{1/s} = 1$, így a

$$\sum_{j=1}^{T} w_j \, 2^{-j} = \sum_{x \in \mathcal{X}} 2^{-L(x)} \leq 1 \ .$$

eredményhez jutunk. ■

10.3. tétel (zajmentes kódolás tétele). *Egy $(\mathcal{X}, P)$, $\mathcal{X} = \{1, \ldots, a\}$ forráshoz mindig található olyan $c : \mathcal{X} \longrightarrow \{0,1\}^*$ egyértelműen dekódolható kód, amelynek átlagos hossza*

$$H(P) \leq L_{\min}(P) < H(P) + 1 \ .$$

Bizonyítás. Jelölje $L(1), \ldots, L(a)$ egy optimális egyértelműen dekódolható kód kódszavainak hosszát. Definiáljuk $\mathcal{X} = \{1, \ldots, a\}$ felett a Q valószínűségi eloszlást. Minden $x \in \mathcal{X}$-re $Q(x) = 2^{-L(x)}/r$, ahol

$$r = \sum_{x=1}^{a} 2^{-L(x)} \ .$$

A Kraft-egyenlőtlenség miatt $r \leq 1$.

Legyen P és Q két valószínűségi eloszlás $\mathcal{X}$ felett. Ezek ***I-divergenciáját,*** melynek jele $D(P\|Q)$, a következőképpen definiáljuk:

$$D(P\|Q) = \sum_{x\in\mathcal{X}} P(x) \lg \frac{P(x)}{Q(x)} \ .$$

Az I-divergencia jól méri két valószínűségi eloszlás távolságát. Az I-divergencia nem lehet negatív, azaz $D(P\|Q) \geq 0$. Bármely P valószínűségi eloszlásra

$$D(P\|Q) = -H(P) - \sum_{x\in\mathcal{X}} P(x) \cdot \lg(2^{-L(x)} \cdot r^{-1}) \geq 0 \ .$$

Ebből következik, hogy

$$\begin{aligned} H(P) &\leq -\sum_{x\in\mathcal{X}} P(x) \cdot \lg(2^{-L(x)} \cdot r^{-1}) \\ &= \sum_{x\in\mathcal{X}} P(x) \cdot L(x) - \sum_{x\in\mathcal{X}} P(x) \cdot \lg r^{-1} = L_{min}(P) + \lg r \ . \end{aligned}$$

Ugyanakkor $r \leq 1$, így $\lg r \leq 0$, ezért $L_{min}(P) \geq H(P)$.

A tétel jobb oldali egyenlőtlenségének bizonyításához vezessük be a következő jelölést: $L'(x) = \lceil -\lg P(x) \rceil \quad (x = 1, \ldots, a)$. Látható, hogy $-\lg P(x) \leq L'(x) < -\lg P(x) + 1$, ezért $P(x) \geq 2^{-L'(x)}$.

Így $1 = \sum\limits_{x\in\mathcal{X}} P(x) \geq \sum\limits_{x\in\mathcal{X}} 2^{-L'(x)}$ és a Kraft-egyenlőtlenség miatt létezik olyan egyértelműen dekódolható kód, amelyben a kódszavak hossza $L'(1), \ldots, L'(a)$. Ennek a kódnak

$$\sum_{x\in\mathcal{X}} P(x) \cdot L'(x) < \sum_{x\in\mathcal{X}} P(x)(-\lg P(x) + 1) = H(P) + 1 \ .$$

■

az átlagos hossza.

10.1.4. A Shannon–Fano–Elias-kód és a Shannon–Fano-algoritmus

A zajmentes kódolás tételének bizonyításában láttuk, hogy adott $P = (P(1), \ldots, P(a))$ valószínűségi eloszláshoz miként rendelhetünk egy c prefix kódot. Minden $x \in \{1, \ldots, a\}$ jelhez egy $L(x) = \lceil \lg(1/P(x)) \rceil$ hosszúságú kódszót rendeltünk úgy, hogy egy megfelelő csúcsot választottunk a megadott fából. Ugyanakkor ez az eljárás nem eredményez minden esetben optimális kódot. Ha például adott az $(1/3, 1/3, 1/3)$ valószínűségi eloszlás, a következő kódokat kapnánk: $1 \longrightarrow 00$, $2 \longrightarrow 01$, $3 \longrightarrow 10$, ezek átlagos hossza 2. Viszont az $1 \longrightarrow 00$, $2 \longrightarrow 01$, $3 \longrightarrow 1$ kód átlagos hossza csak $5/3$.

Shannon megadott egy olyan eljárást, amellyel $\lceil \lg(1/P(x)) \rceil$ kódszó hosszúságú kódot állíthatunk elő. Az eljárás az összegzett valószínűségek bináris ábrázolásán alapul. (Shannon megjegyezte, hogy az eljárást eredetileg Fano dolgozta ki). A forrás elemeit valószínűség szerint csökkenő sorrendbe rendezzük, azaz $P(1) \geq P(2) \geq \cdots \geq P(a)$. A $c_S(x)$ kódszó a $Q(x) = \sum_{j<x} P(j)$ összeg bináris kiterjesztésének első $\lceil \lg(1/P(x)) \rceil$ bitje lesz.

x	$P(x)$	$Q(x)$	$\overline{Q}(x)$	$\lceil \lg(1/P(x)) \rceil$	$c_S(x)$	$c_{SFE}(x)$
1	0.25	0	0.125	2	00	001
2	0.2	0.25	0.35	3	010	0101
3	0.11	0.45	0.505	4	0111	10001
4	0.11	0.56	0.615	4	1000	10100
5	0.11	0.67	0.725	4	1010	10111
6	0.11	0.78	0.835	4	1100	11010
7	0.11	0.89	0.945	4	1110	11110
				$\overline{L}$	3.3	4.3

10.3. ábra. Példa a Shannon- és a Shannon–Fano–Elias-kódra.

x	$P(x)$	$c(x)$	$L(x)$
1	0.25	00	2
2	0.2	01	2
3	0.11	100	3
4	0.11	101	3
5	0.11	110	3
6	0.11	1110	4
7	0.11	1111	4
		$\overline{L}(c)$	2.77

10.4. ábra. Példa a Shannon–Fano-algoritmus működésére.

Ezt az eljárást fejlesztette tovább Elias. A forrás elemeinek sorrendje ebben az esetben tetszőleges. A ***Shannon–Fano–Elias-kód*** $c_{SFE}(x)$ kódszava a $\overline{Q}(x) = \sum_{j<x} P(j) + (1/2)P(x)$ összeg bináris kiterjesztésének első $\lceil \lg(1/P(x)) \rceil + 1$ bitje.

A 10.3. ábrán látható példával szemléltetjük ezeket az eljárásokat.

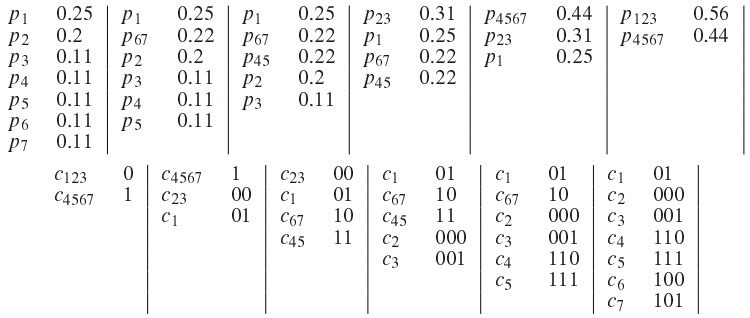
Shannon és Fano közzétett egy hatékonyabb eljárást. A ***Shannon–Fano-algoritmus*** működését szemlélteti az előző példán a 10.4. ábra.

A jeleket valószínűség szerint csökkenő sorrendbe rendezzük. A jeleket egy vágással két diszjunkt csoportba osztjuk úgy, hogy a csoportokba tartozó jelek valószínűségeinek összege a lehető legkisebb mértékben térjen el. A két csoport legyen X_0 és X_1. 0-t rendelünk minden jelhez az első halmazban, és 1-et a többi jelhez. Ezt az eljárást folytatjuk az X_0 és az X_1 halmazokra, azaz az X_0 halmazt X_{00} és X_{01} részhalmazokra bontjuk. Az X_{00} halmazban szereplő jelekhez tartozó kódszó első két jele 00 lesz, az X_{01} halmazbeli jelek kódszavai a 01 sorozattal kezdődnek. Az eljárás véget ér, ha minden részhalmaz egyetlen elemet tartalmaz.

Általában ez az algoritmus sem eredményez optimális kódot, mert például a 10.4. ábrán látható esetben az $1 \longrightarrow 01$, $2 \longrightarrow 000$, $3 \longrightarrow 001$, $4 \longrightarrow 110$, $5 \longrightarrow 111$, $6 \longrightarrow 100$, $7 \longrightarrow 101$ prefix kód átlagos hossza 2.75.

10.1.5. A Huffman-algoritmus

A ***Huffman-algoritmus*** egy rekurzív eljárás, amelynek működését a 10.5. ábrán szemléltetjük a Shannon–Fano-algoritmus kapcsán megismert példán a $p_x = P(x)$ és $c_x = c(x)$ megfeleltetéssel. A forrás elemszámát minden lépésben eggyel csökkentjük úgy, hogy a két legkisebb valószínűségi értéket elhagyjuk, és ezek $P(a-1) + P(a)$ összegét beillesztjük a csökkenően rendezett $P(1) \geq \cdots \geq P(a-2)$ sorozatba. Így egy új P' valószínűségi eloszlást kapunk, amelyben $P'(1) \geq \cdots \geq P'(a-1)$. Az utolsó lépésben a forrás két elemet tartalmaz valószínűség szerint csökkenő sorrendben. Az első elemhez a 0, a másodikhoz az 1 bitet rendeljük. Ezután a forrást lépésenként felbontjuk az utolsó összevonás alapján addig, amíg



10.5. ábra. Példa a Huffman-kódra.

el nem jutunk az eredeti forrásig. Minden lépésben meghatározzuk a $c(a-1)$ és $c(a)$ kódszavakat úgy, hogy a 0, illetve az 1 bitet fűzzük a $P(a-1)+P(a)$ értéknek megfelelő kódszó végéhez.

Helyesség

A következő tételből adódik, hogy a Huffman-algoritmus mindig optimális átlagos kódszó hosszúságú prefix kódot eredményez.

10.4. tétel. *Adott az $(\mathcal{X}, P)$ forrás, amelyben $\mathcal{X} = \{1, \ldots, a\}$ és a valószínűségek monoton csökkenően rendezettek $P(1) \geq P(2) \geq \cdots \geq P(a)$. Legyen P' a következő valószínűségi eloszlás:*

$$P' = \big(P(1), \ldots, P(a-2), P(a-1) + P(a)\big) .$$

Legyen $c' = \big(c'(1), c'(2), \ldots, c'(a-1)\big)$ egy, a P' eloszláshoz tartozó optimális prefix kód. Definiáljuk a P eloszláshoz a c prefix kódot a következőképpen:

$$\begin{aligned} c(x) &= c'(x) \ \ ha \ \ x = 1, \ldots, a-2 , \\ c(a-1) &= c'(a-1)0 , \\ c(a) &= c'(a-1)1 . \end{aligned}$$

Ekkor c egy P eloszláshoz tartozó optimális prefix kód, és $L_{min}(P) - L_{min}(P') = p(a-1) + p(a)$.

Bizonyítás. Az $\mathcal{X} = \{1, \ldots, a\}$ feletti P ($P(1) \geq P(2) \geq \cdots \geq P(a)$) valószínűségi eloszláshoz létezik egy c optimális prefix kód, amelyre
(i) $L(1) \leq L(2) \leq \cdots \leq L(a)$,
(ii) $L(a-1) = L(a)$,
(iii) $c(a-1)$ és $c(a)$ csak az utolsó bitben különbözik.

Ez teljesül, mert:
(i) Tegyük fel, hogy létezik két jel $x, y \in \mathcal{X}$, amelyekre $P(x) \geq P(y)$ és $L(x) > L(y)$. Tekintsük a c' kódot, amelyet úgy kapunk a c kódból, hogy felcseréljük a $c(x)$ és a $c(y)$ kódszavakat. Ekkor c' átlagos hossza $\overline{L}(c') \leq \overline{L}(c)$, mert

$$\overline{L}(c') - \overline{L}(c) = P(x)\cdot L(y) + P(y)\cdot L(x) - P(x)\cdot L(x) - P(y)\cdot L(y) = (P(x)-P(y))\cdot(L(y)-L(x)) \leq 0 \, .$$

(ii) Tegyük fel, hogy adott a c' kód, amelyben $L(1) \leq \cdots \leq L(a-1) < L(a)$. A prefix tulajdonság miatt elhagyhatjuk a $c'(a)$ kódszó utolsó $L(a) - L(a-1)$ bitjét, és így olyan új c kódhoz jutunk, amelyben $L(a) = L(a-1)$.
(iii) Ha van két maximális hosszúságú kódszó, amelyek nem csak az utolsó bitben különböznek, akkor ezekből az utolsó bitet elhagyva egy jobb kódhoz jutunk.

Az előző lemma felhasználásával bebizonyítjuk a tétel állítását. A c és a c' kódok definíciójából adódik, hogy

$$L_{min}(P) \leq \overline{L}(c) = \overline{L}(c') + p(a-1) + p(a) \, .$$

Legyen c'' olyan optimális prefix kód, amelyre fennáll az előző lemma ii) és iii) állítása. Definiáljuk a c''' prefix kódot a

$$P' = (P(1), \ldots, P(a-2), P(a-1) + P(a))$$

valószínűségi eloszlásra a következőképpen: legyen $c'''(x) = c''(x)$ minden $x = 1, \ldots, a-2$ esetén, és $c'''(a-1)$ kódszót származtassuk a $c''(a-1)$ vagy $c''(a)$ kódszóból az utolsó bit elhagyásával.

Ekkor

$$L_{min}(P) = \overline{L}(c'') = \overline{L}(c''') + P(a-1) + P(a) \geq L_{min}(P') + P(a-1) + P(a) \, ,$$

így $L_{min}(P) - L_{min}(P') = P(a-1) + P(a)$, mert $\overline{L}(c') = L_{min}(P')$. ■

Elemzés
Ha a jelöli a forrás ábécé méretét, akkor a Huffman-algoritmus $a-1$ összeadást, és $a-1$ kód változtatást (0 vagy 1 hozzáfűzés) hajt végre. Ezen kívül szükség van $a-1$ beszúrásra, így a teljes műveletigényt az $O(a \lg a)$ értékkel becsülhetjük. Vegyük észre, hogy a zajmentes kódolás tétele miatt a tömörítés mértékét csak k jelből álló blokkok kódolásával javíthatnánk. Ebben az esetben a Huffman-algoritmus forrása $\mathcal{X}^k$ lenne, amelynek mérete a^k. Így a jobb tömörítés ára a műveletigény meglehetősen drasztikus növekedése lenne. Továbbá az összes a^k jelkombinációhoz hozzátartozó kódszót is tárolni kellene. Egy n jel hosszúságú szöveg kódolása $O(n/k) \cdot (a^k \lg a^k))$ lépést igényelne.

Gyakorlatok

10.1-1. Mutassuk meg tetszőleges $n > 0$ értékre, hogy a $c : \{a, b\} \longrightarrow \{0, 1\}^*$ kód, amelyben $c(a) = 0$ és

$$c(b) = \underbrace{0 \ldots 01}_{n}$$

egyértelműen dekódolható, de nem azonnali.

10.1-2. Határozzuk meg az $(\mathcal{X}, P)$ forrás entrópiáját, ha $\mathcal{X} = \{1, 2\}$ és $P = (0.8, 0, 2)$.

10.1-3. Adjuk meg a Huffman- és a Shannon–Fano-kódokat $n = 1, 2, 3$ esetén az $(\mathcal{X}^n, P^n)$ forrásra, ahol $(\mathcal{X}, P)$ az előző gyakorlatban megadott forrás. Határozzuk meg mindkét kód esetén az átlagos kódszó hosszt.

10.1-4. Bizonyítsuk be, hogy $0 \leq H(P) \leq \lg |\mathcal{X}|$.

10.1-5. Mutassuk meg, hogy egy P valószínűségi eloszlású forráshoz tartozó c prefix kód $\rho(c) = \overline{L}(c) - H(P)$ redundanciája kifejezhető, mint speciális I-divergencia.

10.1-6. Lássuk be, hogy egy $\mathcal{X}$ ábécé feletti bármely P és Q valószínűségi eloszlás esetén az I-divergencia nem lehet negatív, azaz $D(P\|Q) \geq 0$, és az egyenlőség csak akkor állhat fenn, ha $P = Q$. Mutassuk meg azt is, hogy az I-divergencia nem metrika.

10.2. Aritmetikai kódolás és modellezés

Statisztikai tömörítések esetén, mint a Shannon–Fano- vagy a Huffman-kódolás, a forrás valószínűségi eloszlását a lehető legpontosabban modellezzük, majd a jeleket kódoljuk úgy, hogy nagyobb valószínűségű jelhez rövidebb kódszót rendelünk.

Tudjuk, hogy a Huffman-algoritmus optimális az átlagos kódszó hossz tekintetében. Ugyanakkor, az entrópiát jobban közelíthetjük, ha a növeljük a blokkhosszt. Viszont hosszú blokkok esetén a Huffman-algoritmus műveletigénye jelentősen nő, hiszen meg kell határozni az összes olyan sorozat valószínűségét, amelynek hossza megegyezik az adott blokkmérettel, és elő kell állítani a megfelelő kódot is.

Statisztikai alapú tömörítések használatakor gyakran az ***aritmetikai kódolást*** részesítik előnyben. Az aritmetikai kódolás természetes kiterjesztése a Shannon–Fano–Elias-kódnak. Az alapötlet, hogy egy valószínűséget egy intervallummal ábrázolunk. Ennek érdekében a valószínűségeket nagyon pontosan kell kiszámítani. Ezt a folyamatot nevezzük a ***forrás modellezésének.*** Ennek megfelelően a statisztikai tömörítések két szakaszra oszthatóak: modellezésre és kódolásra. Amint azt már említettük, a kódolás rendszerint aritmetikai kódolást jelent. A különféle tömörítő algoritmusok, mint például a diszkrét Markov-kódolás (DMC), vagy előrejelzés részleges illeszkedés alapján (PPM), a forrás modellezésében különböznek egymástól. Bemutatjuk a Willems, Shtarkov, és Tjalkens által kifejlesztett környezetfa súlyozó módszert, amelynek lényege a valószínűségek blokkjának becslése. Az algoritmus átláthatósága egy viszonylag egyszerű hatékonysági elemzést tesz lehetővé.

10.2.1. Aritmetikai kódolás

Az aritmetikai kódolás alapgondolata, hogy egy $x^n = (x_1 \ldots x_n)$ szöveget az $I(x^n) = [Q^n(x^n), Q^n(x^n) + P^n(x^n))$ intervallummal ábrázoljuk, ahol $Q^n(x^n) = \sum_{y^n < x^n} P^n(y^n)$ azon sorozatok valószínűségeinek összege, amelyek lexikografikusan kisebbek x^n sorozatnál.

Az x^n szöveghez rendelt $c(x^n)$ kódszó is megfelel egy intervallumnak. Az $L = L(x^n)$ hosszúságú $c = c(x^n)$ kódszót a $J(c) = [bin(c), bin(c)+2^{-L})$ intervallummal adjuk meg, ahol $bin(c)$ a $c/2^L$ tört számlálójának bináris kiterjesztése. A $c(x^n)$ kódszót a következőképpen határozzuk meg a $P^n(x^n)$ és $Q^n(x^n)$ értékek alapján:

$$L(x^n) = \lceil \lg \frac{1}{P^n(x^n)} \rceil + 1, \qquad bin(c) = \lceil Q^n(x^n) \cdot 2^{L(x^n)} \rceil \, .$$

Így az x^n szöveget egy olyan $c(x^n)$ kódszóval kódoljuk, amelyhez tartozó $J(x^n)$ intervallumot tartalmazza az $I(x^n)$ intervallum.

Az aritmetikai kódolást szemlélteti a következő példa. Tekintsünk egy diszkrét, emlékezet nélküli forrást, amelyben $P(0) = 0.9$, $P(1) = 0.1$, és legyen $n = 2$.

x^n	$P^n(x^n)$	$Q^n(x^n)$	$L(x^n)$	$c(x^n)$
00	0.81	0.00	2	00
01	0.09	0.81	5	11010
10	0.09	0.90	5	11101
11	0.01	0.99	8	11111110

Első ránézésre ez a kód sokkal rosszabbnak tűnhet, mint az ugyanezen forráshoz előzőleg meghatározott $(1, 2, 3, 3)$ kódszó hosszúságú Huffman-kód. Viszont bebizonyítható, hogy aritmetikai kódolással mindig $\overline{L}(c) < H(P^n) + 2$ átlagos kódszó hosszúságot lehet elérni, ami csak két bittel tér el a zajmentes kódolás tételében szereplő alsó határtól. A Huffman-algoritmus rendszerint ennél jobb kódot eredményez. Ugyanakkor ezt az „elhanyagolható" tömörítési veszteséget számos előny kompenzálja. A kódszót közvetlenül a forrás sorozatból számoljuk, ezért a Huffman-algoritmussal ellentétben nem kell tárolnunk a kódot. Továbbá, az ilyen esetekben használt forrásmodellek lehetővé teszik, hogy $P^n(x_1x_2 \ldots x_{n-1}x_n)$ és $Q^n(x_1x_2 \ldots x_{n-1}x_n)$ értékeket egyszerűen számítsuk ki $P^{n-1}(x_1x_2 \ldots x_{n-1})$ alapján. Ez rendszerint egy szorzás a $P(x_n)$ értékkel. Ezért a tömörítendő szövegen szekvenciálisan haladhatunk végig, mindig egy jelet olvasva, szemben a Huffman-algoritmussal, amelyben blokkonként kellene kódolnunk.

Kódolás. Az $(x_1, \ldots, x_n)$ sorozat aritmetikai kódolásának alapalgoritmusa a következő. Tegyük fel, hogy $P^n(x_1 \ldots x_n) = P_1(x_1) \cdot P_2(x_2) \cdots P_n(x_n)$. (Ha minden i esetén $P_i = P$, akkor a diszkrét, emlékezet nélküli forrás esetével állunk szemben, de a modellezési részben bonyolultabb esetekkel foglalkozunk.) Ekkor $Q_i(x_i) = \sum_{y<x_i} P_i(x_i)$.

A kiinduló intervallum végpontjai $B_0 = 0$ és $A_0 = 1$. A tömörítendő szöveg első i jele meghatározza a $[B_i, B_i + A_i)$ ***aktuális intervallumot.*** Ezeket az aktuális intervallumokat finomítjuk lépésenként rekurzív módon:

$$B_{i+1} = B_i + A_i \cdot Q_i(x_i), \quad A_{i+1} = A_i \cdot P_i(x_i) \, .$$

A $A_i \cdot P_i(x)$ értéket rendszerint ***kiterjedésnek*** nevezzük. A $[B_n, B_n + A_n) = [Q^n(x^n), Q^n(x^n) + P^n(x^n))$ végső intervallumot kódoljuk a $J(x^n)$ intervallummal, a leírtaknak megfelelően. Így az algoritmus a következő.

ARITMETIKAI-KÓDOLÓ(x)

```
1  B ← 0
2  A ← 1
3  for i ← 1 to n
4      do B ← B + A · Q_i(x[i])
5         A ← A · P_i(x[i])
6  L ← ⌈lg(1/A)⌉ + 1
7  c ← ⌈B · 2^L⌉
8  return c
```

A kódolás menetét az irodalomból származó példával szemléltetjük. Legyen $(\mathcal{X}, P)$ diszkrét, emlékezet nélküli forrás, ahol $\mathcal{X} = \{1, 2, 3\}$ és $P(1) = 0.4$, $P(2) = 0.5$, $P(3) = 0.1$. Az $x^4 = (2, 2, 2, 3)$ sorozatot kell tömörítenünk. Vegyük észre, hogy minden $i = 1, 2, 3, 4$ értékre $P_i = P$ és $Q_i = Q$, továbbá $Q(1) = 0$, $Q(2) = P(1) = 0.4$, és $Q(3) = P(1)+P(2) = 0.9$

A bemutatott algoritmus eredménye a következő.

i	B_i	A_i
0	0	1
1	$B_0 + A_0 \cdot Q(2) = 0.4$	$A_0 \cdot P(2) = 0.5$
2	$B_1 + A_1 \cdot Q(2) = 0.6$	$A_1 \cdot P(2) = 0.25$
3	$B_2 + A_2 \cdot Q(2) = 0.7$	$A_2 \cdot P(2) = 0.125$
4	$B_3 + A_3 \cdot Q(3) = 0.8125$	$A_3 \cdot P(3) = 0.0125$

Így $Q(2, 2, 2, 3) = B_4 = 0.8125$ és $P(2, 2, 2, 3) = A_4 = 0.0125$. Ebből kapjuk, hogy $L = \lceil \lg(1/A) \rceil + 1 = 8$ és $\lceil B \cdot 2^L \rceil = \lceil 0.8125 \cdot 256 \rceil = 208$, amelynek bináris alakja a $c(2, 2, 2, 3) = 11010000$ kódszó.

Dekódolás. A dekódolás nagyon hasonlít a kódolásra. A dekódoló rekurzívan „visszaalakítja" a kódoló rekurzív transzformációját. A $[0, 1)$ intervallumot a Q_i határok alapján részintervallumokra bontjuk. Ezután megkeressük azt az intervallumot, amelyikbe a c kódszó esik. Ez az intervallum megadja a következő jelet. Ezután az intervallumot átalakítjuk a $[0, 1)$ intervallummá úgy, hogy végpontjaiból levonjuk a $Q_i(x_i)$ értéket, és átskálázzuk az $1/P_i(x_i)$ értékkel megszorozva a végpontokat. Ezután az eljárást folytatjuk a következő jelre.

ARITMETIKAI-DEKÓDOLÓ(c, Q, P)

```
1  for i ← 1 to n
2      do j ← 1
3         while c < Q_i(j)
4               do j ← j + 1
5               x[i] ← j − 1
6               c ← (c − Q_i(x[i]))/P_i(x[i])
7  return c
```

Vegyük észre, hogy ha a dekódoló csak a c kódszót kapja meg, akkor nem tudja megállapítani mikor ér véget a dekódolás. Például a $c = 0$ kódszó tartozhat $x^1 = (1)$, $x^2 = (1, 1)$, $x^3 = (1, 1, 1)$ stb. szöveghez. Az előző pszeudokódban feltételeztük, hogy a jelek számát, n-et is megkapja a dekódoló, és ekkor nyilvánvaló, hogy melyik az utolsó dekódolandó jel.

Egy másik lehetőség egy speciális fájl vége jel (EOF) bevezetése. A jel valószínűsége alacsony, és a kódoló és a dekódoló is ismeri. Amikor a dekódoló ehhez a jelhez ér, akkor befejezi a dekódolást. Ebben az esetben az első sort ki kell cserélni, és i értékét növelni kell a ciklus végén.

```
1  while x[i] ≠ EOF
       do ...
6          i ← i + 1
```

A bemutatott példában a dekóder megkapja az 11010000 kódszót, a 0.8125 érték bináris kiterjesztését $L = 8$ bitre. Ez az érték a $[0.4, 0.9)$ intervallumba esik, ami a 2 jelhez tartozik, így $x_1 = 2$ az első jel. Ezután kiszámítja a $(0.8075 - Q(2))(1/P(2)) = (0.815 - 0.4) \cdot 2 = 0.83$ értéket, ami szintén a $[0.4, 0.9)$ intervallumba esik. A második jel tehát $x_2 = 2$. x_3 meghatározásához a $(0.83 - Q(2))(1/P(2)) = (0.83 - 0.4) \cdot 2 = 0.86$ számítást kell elvégezni. $0.86 \in [0.4, 0.9)$, ezért $x_3 = 2$. Végül $(0.86 - Q(2))(1/P(2)) = (0.86 - 0.4) \cdot 2 = 0.92$. Miután $0.92 \in [0.9, 1)$, a sorozat utolsó jele $x_4 = 3$.

Helyesség

Emlékezzünk, hogy az x^n szövegnek megfeleltetett $c(x^n)$ kódszóhoz a $J(x^n)$ intervallum tartozik. Ez az intervallum az $I(x^n)$ intervallum része, mert $\lceil Q^n(x^n) \cdot 2^{L(x^n)} \rceil 2^{-L(x^n)} + 2^{-L(x^n)} < Q^n(x^n) + 2^{1-L(x^n)} = Q^n(x^n) + 2^{-\lceil \lg(1/P^n(x^n)) \rceil} \leq Q^n(x^n) + P^n(x^n)$.

A kapott kód nyilvánvalóan prefix kód, hiszen egy kódszó csak akkor lehet egy másik prefixe, ha a megfelelő intervallumok átfedik egymást, viszont a $J(x^n) \subset I(x^n)$ intervallumok diszjunktak.

Korábban már említettük, hogy az aritmetikai kódolás által előállított bitsorozat hossza legfeljebb két bittel haladja meg a szöveg entrópia értékét. Ennek oka, hogy minden x^n sorozatra $L(x^n) < \lg(1/P^n(x^n)) + 2$. Belátható, hogy a kód kiegészítése a szöveg n hosszával, vagy az EOF jel bevezetése csak elhanyagolható mértékben rontja a tömörítés mértékét.

A bemutatott egyszerű algoritmusok nem alkalmasak hosszabb fájlok tömörítésére, ugyanis a szöveg hosszának növelésével az intervallumok egyre kisebbek lesznek, és ez kerekítési hibákhoz vezet. Bemutatunk egy eljárást, amellyel feloldhatjuk ezt a problémát.

Elemzés

Az aritmetikai kódolás egyszerű algoritmusának futási ideje lineáris függvénye a tömörítendő sorozat n hosszának. Az aritmetikai kódolást rendszerint a Huffman-algoritmussal hasonlítják össze. A Huffman-algoritmussal szemben nem kell tárolnunk a teljes kódot, mert a kódszót a megfelelő intervallumból közvetlenül megkaphatjuk. Ez olyan diszkrét, emlékezet nélküli forrás esetén, amelyben a valószínűségi eloszlás megegyezik az összes jel esetén ($P_i = P$), nem nagy előny. Ekkor ugyanis a Huffman-kód ugyanaz lesz az összes jel (vagy jelek k hosszúságú blokkja) esetén, így csak egyszer kell meghatározni. A Huffman-algoritmusban nincs szükség szorzásra, ami lelassítja az aritmetikai kódolást.

Az adaptív esetben, amikor a P_i értékek változhatnak, különböző x_i kódolandó jelek esetén, egy új Huffman-kódot kell előállítanunk minden egyes új jelre. Ilyenkor rendszerint az aritmetikai kódolást részesítik előnyben. Ezeket az eseteket a modellezésről szóló részben vizsgáljuk majd részletesen.

A gyakorlati megvalósításokban nem használnak lebegőpontos aritmetikát. A $[0, 1)$ intervallum felosztását a $0, \ldots, M$ egész tartomány valószínűségekkel arányos felosztásával helyettesítik. Egész aritmetikával számolnak, ami gyorsabb és pontosabb.

Pontossági probléma
Az aritmetikai kódolás és dekódolás egyszerű algoritmusaiban az intervallumok összehúzódása miatt hosszabb sorozatok esetén nagy pontosságú aritmetikára lenne szükség[2]. Ezen kívül, egyetlen bitet sem állítanánk elő a kódszóból a teljes x^n sorozat beolvasása előtt. Ezt kiküszöbölhetjük, ha minden bitet kiírunk – kiléptetünk – abban a pillanatban, amikor már ismerjük, és az aktuális $[LO, HI)$ intervallum méretét megduplázzuk. A $[LO, HI)$ intervallum a tényleges intervallum ismeretlen részét ábrázolja. Ilyen bit kiléptetésre lehetőségünk van akkor, amikor az intervallum alsó és felső határának vezető bitjei megegyeznek, azaz az intervallum teljes egészében a $[0, \frac{1}{2})$ vagy az $[\frac{1}{2}, 1)$ intervallumba esik. A következő tágítási szabályok garantálják, hogy az aktuális intervallum sose legyen túl kicsi.

1. eset ($[LO, HI) \in [0, 1/2)$): $LO \leftarrow 2 \cdot LO$, $HI \leftarrow 2 \cdot HI$.

2. eset ($[LO, HI) \in [1/2, 1)$): $LO \leftarrow 2 \cdot LO - 1$, $HI \leftarrow 2 \cdot HI - 1$.

3. eset ($1/4 \leq LO < 1/2 \leq HI < 3/4$): $LO \leftarrow 2 \cdot LO - 1/2$, $HI \leftarrow 2 \cdot HI - 1/2$.

Az utolsó eset, amelyet ***alulcsordulásnak*** nevezünk, megakadályozza, hogy az intervallum túlságosan összeszűküljön, amikor a határai közel kerülnek $(1/2)$-hez. Ha az aktuális intervallum az $[1/4, 3/4)$ intervallumba esik, és $LO < 1/2 \leq HI$, akkor ugyan nem tudjuk a következő kiírandó bitet, de tudjuk, hogy bármi lesz is az értéke, a következő bit ennek ellentettje lesz. A többi esettel ellentétben ekkor nem folytathatjuk a kódolást, hanem várnunk kell amíg eldől, hogy az intervallum a $[0, 1/2)$ vagy az $[1/2, 1)$ intervallumba esik-e. A „várakozás közben” az alulcsordulási állapotban maradunk, és az *alulcsordulásszám* számláló értékét állítjuk az egymást követő alulcsordulások számára, azaz az értékét növeljük eggyel. Amikor befejezzük a várakozást, kiírjuk az intervallum vezető bitjét – ez 0 $[0, 1/2)$ esetén és 1 $[1/2, 1)$ esetén –, majd *alulcsordulásszám* darab ellentétes bitet, és a számláló értékét nullázzuk: *alulcsordulásszám* $= 0$. Az eljárás véget ér, amikor az összes jelet beolvastuk és az aktuális intervallumot nem lehet tovább tágítani.

Az algoritmus bemenete az $\mathbf{x} = x[1 \,..\, n]$ tömb.

PONTOS-ARITMETIKAI-KÓDOLÓ($\mathbf{x}$)

```
1  LO ← 0
2  HI ← 1
3  A ← 1
4  alulcsordulásszám ← 0
5  for i ← 1 to n
6       do LO ← LO + Q_i(x[i]) · A
7          A ← P_i(x[i])
8          HI ← LO + A
```

[2]Nincs olyan pontosságú aritmetika, amellyel a probléma gyakorlati esetekben kezelhető lenne. *A fordító.*

Aktuális intervallum	Tevékenység	Részintervallumok 1	2	3	Bemenet
[0.00, 1.00)	felosztás	[0.00, 0.40)	[0.40, 0.90)	[0.90, 1.00)	2
[0.40, 0.90)	felosztás	[0.40, 0.60)	[0.60, 0.85)	[0.85, 0.90)	2
[0.60, 0.85)	1 kódolása [1/2, 1) nyújtása				
[0.20, 0.70)	felosztás	[0.20, 0.40)	[0.40, 0.65)	[0.65, 0.70)	2
[0.40, 0.65)	alulcsordulás [1/4, 3/4) nyújtása				
[0.30, 0.80)	felosztás	[0.30, 0.50)	[0.50, 0.75)	[0.75, 0.80)	3
[0.75, 0.80)	10 kódolása [1/2, 1) nyújtása				
[0.50, 0.60)	1 kódolása [1/2, 1) nyújtása				
[0.00, 0.20)	0 kódolása [0, 1/2) nyújtása				
[0.00, 0.40)	0 kódolása [0, 1/2) nyújtása				
[0.00, 0.80)	0 kódolása				

10.6. ábra. Példa az intervallumnyújtásos aritmetikai kódolás működésére.

```
 9        while HI − LO < 1/2 ∧ ¬(LO < 1/4 ∧ HI ≥ 1/2)
10              do if HI < 1/2
11                    then c ← c||0 és alulcsordulásszám darab 1)
12                    alulcsordulásszám ← 0
13                          LO ← 2 · LO
14                          HI ← 2 · HI
15                 else if LO ≥ 1/2
16                          then c ← c||1 és alulcsordulásszám darab 0)
17                          alulcsordulásszám ← 0
18                          LO ← 2 · LO − 1
19                          HI ← 2 · HI − 1
20                 else if LO ≥ 1/4 ∧ HI ≥ 1/2
21                          then alulcsordulásszám ← alulcsordulásszám + 1
22                          LO ← 2 · LO − 1/2
23                          HI ← 2 · HI − 1/2
24  if alulcsordulásszám > 0
25    then c ← 0 és alulcsordulásszám darab 1)
26  return c
```

A kódoló algoritmus működését a 10.6. ábrán szemléltetjük. A példában a $(2, 2, 2, 3)$ szöveget tömörítjük, ahol az ábécé $\mathcal{X} = \{1, 2, 3\}$, és a valószínűségi eloszlás $P = (0.4, 0.5, 0.1)$. Alulcsordulás lép fel a hatodik sorban. Nyilvántartjuk az alulcsordulási állapotot, és később kódoljuk a következő bit inverzével. Most ez az inverz bit a 0, a kilencedik sorban. A tömörített szöveg 1101000.

A pontos dekódoláshoz az intervallum határok, *LO* és *HI* mellé be kell vezetnünk egy harmadik változót. Ez a változó a kódszóból ugyanazt a bittartományt tartalmazza, mint *LO* és *HI* az intervallum határaiból.

a \ b	0	1	2	3	4	5
0	1	1/2	3/8	5/16	35/128	63/256
1	1/2	1/8	1/16	5/128	7/256	21/1024
2	3/8	1/16	3/128	3/256	7/1024	9/2048
3	5/16	5/128	3/256	5/1024	5/2048	45/32768

10.7. ábra. A Krichevsky–Trofimov-becslés első értékeit tartalmazó táblázat.

10.2.2. Modellezés

Ebben a pontban a Krichevsky–Trofimov-becslést és a környezetfával való becslést tárgyaljuk.

Diszkrét, emlékezet nélküli források modellezése a Krichevsky–Trofimov-becsléssel
Ebben a részben csak olyan esetekkel foglalkozunk, amikor az aritmetikai kódolással $x^n \in \{0,1\}^n$ bináris sorozatokat kell tömörítenünk. Annak érdekében, hogy speciális helyzetek leírásakor szabadon használhassunk alsó és felső indexeket, $P^n(x^n)$ helyett a $P(x^n)$ jelölést használjuk. P_e jelöli a becsült valószínűségeket, P_w a súlyozott valószínűségeket, és P^s egy speciális s környezethez rendelt valószínűségeket.

Az aritmetikai kódolás jól alkalmazható, ha a forrás valószínűségi eloszlása olyan, hogy $P(x_1x_2\ldots x_{n-1}x_n)$ érték könnyen meghatározható $P(x_1x_2\ldots x_{n-1})$ alapján. Ez nyilvánvalóan teljesül diszkrét, emlékezet nélküli források esetén, hiszen ekkor $P(x_1x_2\ldots x_{n-1}x_n) = P(x_1x_2\ldots x_{n-1})\cdot P(x_n)$.

A valószínűségeket még abban az esetben is hatékonyan becsülhetjük a Krichevsky és Trofimov által megadott módon, amikor a bináris, diszkrét, emlékezet nélküli forrás meghatározó paramétere, $\theta = P(1)$ ismeretlen. A becslési formula

$$P(X_n = 1|x^{n-1}) = \frac{b+\frac{1}{2}}{a+b+1}\ ,$$

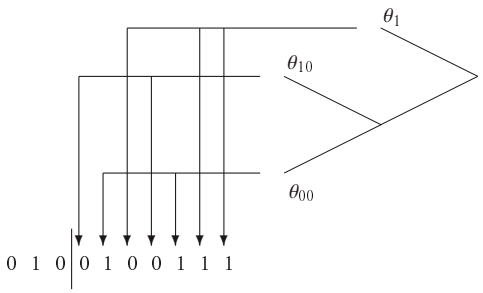
amelyben a és b jelöli a 0 és az 1 bitek számát az $x^{n-1} = (x_1, x_2, \ldots, x_{n-1})$ sorozatban. Tehát egy a darab 0 bitet és b darab 1 bitet tartalmazó x^{n-1} sorozat esetén annak a valószínűsége, hogy a következő x_n jel az 1 bit lesz, a $(b+1/2)/(a+b+1)$ értékkel becsülhető. Egy a nullát és b egyest tartalmazó sorozat becsült valószínűsége

$$P_e(a,b) = \frac{\frac{1}{2}\cdots(a-\frac{1}{2})\frac{1}{2}\cdots(b-\frac{1}{2})}{1\cdot 2\cdots(a+b)}\ ,$$

ahol kezdetben $a = 0$ és $b = 0$. Ezt mutatja a 10.7. ábrán látható táblázat, amelyben a ***Krichevsky–Trofimov-becslés*** $P_e(a,b)$ értékei szerepelnek kis (a,b) értékekre.

Vegyük észre, hogy a számlálóban szereplő 1/2 tag biztosítja, hogy annak a valószínűsége pozitív, hogy a következő jel akkor is 1, ha 1 még nem fordult elő eddig a sorozatban. Aritmetikai kódolás alkalmazásakor ezt a lehetőséget minden lehetséges paraméter-becslés esetén figyelembe kell venni amikor a következő jel valószínűségét becsüljük, hogy elkerüljük a végtelen kódszóhosszúságot.

Modellek ismert környezetfa esetén
A legtöbb esetben a forrás nem emlékezet nélküli, azaz figyelembe kell venni a jelek közötti függőségeket. A szuffix fák alkalmasak ilyen típusú függőségek ábrázolására. A szuffix



10.8. ábra. Példa egy fa forrásra.

fákat a továbbiakban ***környezetfáknak*** nevezzük. Egy x_t jel környezete az x_t jelet megelőző s szuffix. Minden s környezethez (vagy levélhez a szuffixfában) tartozik egy $\theta_s = P(X_t = 1|s)$ paraméter, ami annak a valószínűsége, hogy a következő jel 1, ha a forrás előzőleg beolvasott jeleinek sorozata megegyezik az s környezettel. (Ennek megfelelően $1 - \theta_s$ a 0 jel valószínűsége ebben az esetben.) Itt különbséget teszünk a modell (a szuffixfa) és a paraméterek (θ_s) között.

10.1. példa. Legyen $\mathcal{S} = \{00, 10, 1\}$, továbbá $\theta_{00} = 1/2$, $\theta_{10} = 1/3$ és $\theta_1 = 1/5$. A megfelelő szuffix fát és egy sorozat feldolgozását szemlélteti a 10.8. ábra. A '0100111' sorozat aktuális valószínűsége a '...010' olvasása után $P^s(0100111|\ldots 010) = (1-\theta_{10})\theta_{00}(1-\theta_1)(1-\theta_{10})\theta_{00}\theta_1\theta_1 = 2/3 \cdot 1/2 \cdot 4/5 \cdot 2/3 \cdot 1/2 \cdot 1/5 \cdot 1/5 = 4/1075$, mert az első 0 jelet megelőzi az 10 szuffix, a második 1 jelet megelőzi a 00 szuffix stb.

Tegyük fel, hogy ismerjük az $\mathcal{S}$ modellt, de nem tudunk semmit a θ_s paraméterről. A problémát ekkor egy jó kódolási eloszlás meghatározása jelenti. A fa szerkezet segítségével könnyen eldönthető, hogy milyen környezet előz meg egy adott jelet. Az összes jel, amelynek ugyanaz az $s \in \mathcal{S}$ a környezete (vagy szuffixe), egy olyan emlékezet nélküli forrás részsorozatát alkotja, amelynek valószínűségét az ismeretlen θ_s paraméter határozza meg. A példánkban ezek a részsorozatok '11' θ_{00} esetén, '00' θ_{10} esetén, és '011' θ_1 esetén. Ebben az esetben használható a Krichevsky–Trofimov-becslés. A szuffixfa minden egyes s csúcsára megszámoljuk az s szuffixet követő nullák a_s és egyesek b_s számát. Az s szülő csúcs $0s$ és $1s$ gyerekei esetén $a_{0s} + a_{1s} = a_s$ és $b_{0s} + b_{1s} = b_s$ feltételeknek nyilvánvalóan teljesülniük kell.

Példánkban λ gyökérre $(a_\lambda, b_\lambda) = (3, 4)$, $(a_1, b_1) = (1, 2)$, $(a_0, b_0) = (2, 2)$, és $(a_{10}, b_{10}) = (2, 0)$, $(a_{00}, b_{00}) = (0, 2)$. Továbbá $(a_{11}, b_{11}) = (0, 1)$, $(a_{01}, b_{01}) = (1, 1)$, $(a_{111}, b_{111}) = (0, 0)$, $(a_{011}, b_{011}) = (0, 1)$, $(a_{101}, b_{101}) = (0, 0)$,$(a_{001}, b_{001}) = (1, 1)$, $(a_{110}, b_{110}) = (0, 0)$, $(a_{010}, b_{010}) = (2, 0)$, $(a_{100}, b_{100}) = (0, 2)$ és $(a_{000}, b_{000}) = (0, 0)$. Az utolsó értékek nem játszanak szerepet a speciális $\mathcal{S}$ forrásunk esetén, de a későbbiekben fontosak lesznek, amikor a forrás modell, illetve a megfelelő szuffixfa nem ismert előre.

10.2. példa. Legyen $\mathcal{S} = \{00, 10, 1\}$, mint az előző példában. Egy részsorozatot a megfelelő a_s és b_s számlálók karbantartásával kódolunk. Például a '0100111' sorozat kódolásakor a '...010' rész elolvasása után az előző szuffixfa és a Krichevsky–Trofimov-becslés felhasználásával kapjuk, hogy

$$P_e^s(0100111|\ldots 010) = \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{3}{4}\cdot\frac{1}{4}\cdot\frac{1}{2} = \frac{3}{8}\cdot\frac{3}{8}\cdot\frac{1}{16} = \frac{9}{1024},$$

ahol 3/8, 3/8 és 1/16 az '11', a '00' és a '011' részsorozatok valószínűségei a levelek környezetében. Ezek a részsorozatok emlékezet nélküliek.

A környezetfa súlyozó módszer

Tegyük fel, hogy rendelkezésünkre áll egy jó kódolási eloszlás (P_1), egy forrás, és egy másik (P_2) két forrás esetén. Keresünk egy jó kódolási eloszlást, amelyik jó mindkét forrásra. Egy lehetőség P_1 és P_2 kiszámítása, és ezután szükségünk van 1 bitre a legjobb modell azonosításához, aminek segítségével azután tömörítenénk a sorozatot. Ezt a módszert nevezzük ***választásnak.*** Egy másik lehetőség a ***súlyozott eloszlás*** alkalmazása. A súlyozott eloszlás képlete

$$P_w(x^n) = \frac{P_1(x^n) + P_2(x^n)}{2} \ .$$

Bemutatjuk a ***környezetfa súlyozó algoritmust.*** Feltesszük, hogy a környezetfa egy D mélységű teljes fa, és csak az a_s és a b_s értékeket tartjuk nyilván a fa minden s csúcsában. Ezek az értékek az s környezetet követő bitek részsorozatában előforduló 0, illetve 1 bitek számát adják meg.

Minden s csúcshoz hozzárendelünk egy P_w^s súlyozott valószínűséget, amelyet a következő rekurzív definícióval kapunk:

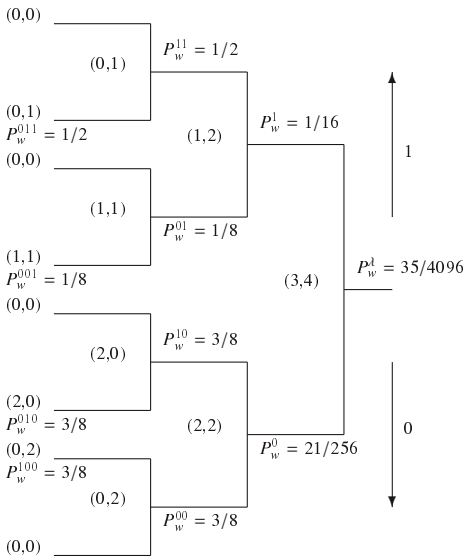
$$P_w^s = \begin{cases} \frac{P_e(a_s,b_s)+P_w^{0s}P_w^{1s}}{2}, & \text{ha } 0 \le L(s) < D \ , \\ P_e(a_s,b_s), & \text{ha } L(s) = D \ . \end{cases}$$

Itt $L(s)$ megadja a (bináris) s sorozat hosszát, és $P_e(a_s, b_s)$ a Krichevsky–Trofimov-becsléssel meghatározott becsült valószínűség.

10.3. példa. Ha kódoljuk a '0100111' sorozatot a '...010' elolvasása után a 10.9. ábrán látható 3 mélységű környezetfát kapjuk eredményül. A λ gyökércsúcs $P_w^\lambda = 35/4096$ súlyozott valószínűsége megadja a a feldolgozott sorozat kódolási valószínűségét.

Aritmetikai kódolás használatakor fontos, hogy a $P(x_1 \dots x_{n-1}0)$ és $P(x_1 \dots x_{n-1}1)$ valószínűségeket hatékonyan lehessen számolni $P(x_1 \dots x_n)$ értékből. Ez fennáll a környezetfa súlyozó módszer esetén, mert csak P_w^s súlyozott valószínűségeket kell karbantartani s változásakor. Ezt csak a $D+1$ környezetre kell végrehajtani. Ezek a környezetfában a gyökértől az x_n jelet megelőző levélig vezető úton helyezkednek el, azaz a λ gyökér és az $x_{n-1} \dots x_{n-D+i}$ $(i = 1, \dots, D)$ környezetek. Az út mentén el kell végezni az $a_s = a_s + 1$ értékadást, ha $x_n = 0$, illetve $b_s = b_s + 1$ értékadást, ha $x_n = 1$. Ezen kívül a megfelelő $P_e(a_s, b_s)$ és P_w^s valószínűségeket kell módosítani.

Ennek alapján a következő algoritmushoz jutunk, amelyik a következő x_n jel olvasásakor módosítja a $CT(x_1 \dots x_{n-1}|x_{-D+1} \dots x_0)$ környezetfát. Emlékezzünk, hogy a fa minden csúcsában az (a_s, b_s), $P_e(a_s, b_s)$ és P_w^s értékeket tároljuk. Ezeket kell módosítani, hogy megkapjuk az új $CT(x_1, \dots, x_n|x_{-D+1}, \dots x_0)$ környezetfát. Feltesszük, hogy az (x_{n-1}, x_n) rendezett pár jelöli a λ gyökeret.



10.9. ábra. A '0100111' forrás sorozathoz tartozó súlyozott környezetfa a ... 010 elolvasása után. Az (a_s, b_s) pár jelöli, hogy a_s nulla és b_s egyes követi a megfelelő s környezetet. Az $s = 111, 101, 110, 000$ környezetek esetén $P_w^s = P_e(0,0) = 1$.

KÖRNYEZET-FA-MÓDOSÍTÁS$(x_n, CT(x_1, \ldots, x_{n-1}|x_{-D+1}, \ldots, x_0))$

```
1  s ← (x_{n-1}, ..., x_{n-D})
2  if x_n = 0
3     then P_w^s ← P_w^s · (a_s + 1/2)/(a_s + b_s + 1)
4          a_s ← a_s + 1
5     else P_w^s ← P_w^s · (b_s + 1/2)/(a_s + b_s + 1)
6          b_s ← b_s + 1
7  for i ← 1 to D
8      do s ← (x_{n-1}, ..., x_{n-D+i})
9         if x_n = 0
10           then P_e(a_s, b_s) ← P_e(a_s, b_s) · (a_s + 1/2)(a_s + b_s + 1)
11                a_s ← a_s + 1
12           else P_e(a_s, b_s) ← P_e(a_s, b_s) · (a_s + 1/2)(a_s + b_s + 1)
13                b_s ← b_s + 1
14        P_w^s ← (1/2) · (P_e(a_s, b_s) + P_w^{0s} · P_w^{1s})
15 return P_w^s
```

Az aritmetikai kódolásban a környezetfa gyökeréhez rendelt P_w^{λ} valószínűséget használjuk az intervallumok egymást követő felosztásához. Kezdetben, x_1 olvasása előtt miden fában szereplő s környezet esetén az $(a_s, b_s) = (0,0)$, $P_e(a_s, b_s) = 1$ és $P_w^s = 1$ értékek találhatók a környezetfában. A példában az $(x_{-2}, x_{-1}, x_0) = (0,1,0)$ olvasása utáni módosítás a következő valószínűségi értékekhez vezet P_w^{λ}: 1/2, ha $x_1 = 0$, 9/32, ha $(x_1, x_2) = (0,1)$, 5/64, ha $(x_1, x_2, x_3) = (0,1,0)$, 13/256, ha $(x_1, x_2, x_3, x_4) = (0,1,0,0)$, 27/1024, ha $(x_1, x_2, x_3, x_4) = (0,1,0,0,1)$, 13/1024, ha $(x_1, x_2, x_3, x_4, x_5) = (0,1,0,0,1,1)$, 13/1024,

ha $(x_1, x_2, x_3, x_4, x_5, x_6) = (0, 1, 0, 0, 1, 1)$, és $35/4096$, ha $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (0, 1, 0, 0, 1, 1, 1)$.

Helyesség

Egy kód minőségét a tömörítési hatékonyság szempontjából a kódszavak átlagos hossza segítségével mérjük. A legjobb kód átlagos kódszóhossza a lehető legjobban megközelíti a forrás entrópiáját. Az átlagos kódszóhossz és az entrópia közötti eltérést nevezzük a kód ***redundanciájának.*** Egy c kód esetén a redundancia jele $\overline{\rho}(c)$, és az eddigiek alapján

$$\overline{\rho}(c) = \overline{L}(c) - H(P) \ .$$

Ez nyilvánvalóan az egyedi redundanciák ($P(x^n)$ súllyal) súlyozott összege

$$\rho(x^n) = L(x^n) - \lg \frac{1}{P(x^n)} \ .$$

Egy adott (és ismert) $\mathcal{S}$ forrás, $|\mathcal{S}| \leq n$, és minden $\theta_s \in [0, 1]$ ($s \in \mathcal{S}$) paraméter esetén az x^n sorozatok $\rho(x^n|\mathcal{S})$ egyedi redundanciája felülről korlátos, és a korlát

$$\rho(x^n|\mathcal{S}) \leq \frac{|\mathcal{S}|}{2} \lg \frac{n}{|\mathcal{S}|} + |\mathcal{S}| + 2 \ .$$

Környezetfa súlyozó algoritmus használata esetén (ez egy teljes fát jelent, amely az összes lehetséges környezetet tartalmazza, mint $\mathcal{S}$ modelljét) az x^n sorozatok $\rho(x^n|\mathcal{S})$ egyedi redundanciájának felső korlátja

$$\rho(x^n|\mathcal{S}) < 2|\mathcal{S}| - 1 + \frac{|\mathcal{S}|}{2} \lg \frac{n}{|\mathcal{S}|} + |\mathcal{S}| + 2 \ .$$

A két korlát összehasonlításából adódik, hogy az egyedi redundanciák különbsége $2|\mathcal{S}| - 1$ bit. Ezt tekinthetjük a modell ismeretlenségéből fakadó költségnek, azaz modell redundanciának. Így a redundancia felbontható paraméter redundanciára, azaz a paraméter ismeretlensége miatt adódó költségre, és modell redundanciára. Belátható, hogy a környezetfa súlyozó módszer várható redundanciája eléri a Rissanen által megállapított aszimptotikus alsó korlátot. Eszerint paraméterenként körülbelül $(1/2) \lg n$ bit a minimálisan várható redundancia, ha $n \to \infty$.

Elemzés

A műveletigény arányos a fa módosítása során érintett csúcsok számával, ami körülbelül $n(D + 1)$. Ezért az n jel feldolgozásához szükséges műveletek száma n-nel arányos. Ugyanakkor ezek a műveletek nagyrészt szorzások, amelyek tényezői nagy pontosságot igényelnek.

Hasonlóan a legtöbb modellező algoritmushoz, a gyakorlati megvalósításokban komoly feladatot jelent az óriási memóriaigény. Egy teljes, D mélységű fát kell tárolni és módosítani. Csak D értékének növelésével érhető el, hogy a valószínűségek becslései pontosabbá váljanak, és így az ezekre épülő aritmetikai kódolás átlagos kódszóhossza kisebb legyen. A felhasználandó memória mérete viszont exponenciálisan függ a fa mélységétől.

A környezetfa súlyozó módszert csak bináris sorozatokra adtuk meg. Ebben az esetben

az $(x_1, \ldots, x_n)$ bináris sorozat összegzett valószínűsége kiszámítható, mint

$$Q^n(x_1 x_2 \ldots x_{n-1} x_n) = \sum_{j=1,\ldots,n; x_j=1} P^j(x_1 x_2 \ldots x_{j-1} 0) \ .$$

Nagyobb ábécét tartalmazó források (például ASCII fájlok) tömörítésének leírását az olvasó megtalálhatja a megadott irodalomban.

Gyakorlatok

10.2-1. Legyen $\mathcal{X} = \{1, 2\}$ és $P = (0.8, 0, 2)$. Adjuk meg az $(\mathcal{X}^n, P^n)$, $n = 1, 2, 3$ források aritmetikai kódját, és hasonlítsuk össze ezeket az előzőleg meghatározott megfelelő Huffman-kódokkal.

10.2-2. Számítsuk ki az előző gyakorlatban meghatározott kódok esetén az egyes kódszavak egyedi redundanciáját és a kódok redundanciáját.

10.2-3. Határozzuk meg a Krichevsky–Trofimov-becslés alapján a 0100110 sorozat és összes részsorozatának $P_e(a, b)$ becsült valószínűségét.

10.2-4. Számítsuk ki az összes (a_s, b_s) értéket és P_e^s becsült valószínűséget a 0100110 sorozatra az 110 olvasása után, az $\mathcal{S} = \{00, 10, 1\}$ környezetfa ismeretében. Mi lesz az aritmetikai kódolás által előállított kódszó?

10.2-5. Határozzuk meg a környezetfa súlyozó algoritmussal az összes (a_s, b_s) értéket és a P_λ becsült valószínűséget a 0100110 sorozatra az 110 olvasása után, ha nem ismerjük a környezetfát.

10.2-6. Az előző gyakorlat eredményeit felhasználva módosítsuk a 01001101 sorozat 110 olvasása utáni becsült valószínűségét.

10.2-7. Lássuk be, hogy az $(x_1 \ldots x_n)$ bináris sorozat összegzett valószínűsége

$$Q^n(x_1 x_2 \ldots x_{n-1} x_n) = \sum_{j=1,\ldots,n; x_j=1} P^j(x_1 x_2 \ldots x_{j-1} 0) \ .$$

10.3. Ziv–Lempel tömörítés

1976–1978-ban Jacob Ziv és Abraham Lempel kifejlesztett két univerzális tömörítő algoritmust, amelyek az eddig tárgyalt statisztikai tömörítésekkel szemben nem használják fel explicit módon a szöveg valószínűségi eloszlását. Az alapötlet, hogy egy előzőleg már előfordult sorozatot egy történeti puffer mutatójával (LZ77), illetve egy szótárbeli hivatkozással (LZ78) helyettesítünk. Az LZ algoritmusokat széles körben használják – a „zip” és ennek különféle változatai az LZ77 algoritmusra épülnek. A több szerző által használt megközelítéssel szemben a Ziv–Lempel tömörítés nem egyetlen algoritmus. Eredetileg Lempel és Ziv sorozatok bonyolultságának mérésére javasolt egy – a Kolmogorov bonyolultsághoz hasonló – módszert. Ebből származik a két különböző, LZ77 és LZ78, algoritmus. Azóta ezeknek több módosítását, variációját fejlesztették ki, de ebben a részben az eredeti algoritmusokat mutatjuk be. Az érdeklődő olvasónak figyelmébe ajánljuk a megadott irodalmat.

10.3.1. LZ77

Az LZ77 algoritmus alapelve, hogy egy ***csúszó ablakot*** tolunk végig a tömörítendő szövegen. Ebben az ablakban keressük a szöveg soron következő részére illeszkedő leghosszabb részsorozatot. Az ablak két részből áll: az l_h hosszúságú történeti ablakból, amelyben a szöveg utolsó l_h darab vizsgált jelét tároljuk, és egy l_f méretű előre néző ablakból, amely a szöveg következő l_f jelét tartalmazza. A legegyszerűbb esetben az l_h és l_f értékek rögzítettek. Rendszerint l_h jóval nagyobb, mint l_f. Ezek felhasználásával előállítjuk az (eltolás, hossz, jel) hármast. Ebben az ***eltolás*** megadja, hogy hány jellel előbb fordul elő a szövegben az illeszkedő részsorozat az aktuális pozícióhoz képest, a hossz az illeszkedő részsorozat hossza, és a jel a részsorozatot követő jel. Egy példával szemléltetjük ezt az eljárást. Tegyük fel, hogy *...abaabbaabbaaabbbaaaabbabbbabbb...* a tömörítendő szöveg, az ablak mérete 15, amelyből $l_h = 10$ jel a történeti, és $l_f = 5$ jel az előre néző rész hossza. Tegyük fel, hogy a csúszó ablak jelenlegi helyzete

$$...aba||abbaabbaaa|bbbaa|| ,$$

azaz a történeti ablak az *abbaabbaaa* jeleket, és az előre néző ablak a *bbbaa* jeleket tartalmazza. A leghosszabb részsorozat, ami illeszkedik az előre néző ablak elejére, a 2 hosszúságú *bb* sorozat. Ez a történeti ablak végétől kilenc jellel előbb fordul elő. A következő jel *b*, így a $(9, 2, b)$ hármast állítjuk elő. (A *bb* részsorozat 5 jellel előbb is megtalálható, de az eredeti LZ77 algoritmus a legnagyobb eltolási értéket választja.) Ezután az ablakot 3 jellel előre toljuk

$$...abaabb||aabbaaabbb|aaaab|| .$$

A következő kódszó $(6, 3, a)$, mert a leghosszabb illeszkedő részsorozat a 3 hosszúságú *aaa* 6 jellel előbb fordul elő, és az előre néző ablakban *a* a részsorozatot követő jel. Az ablak következő helyzete

$$...abaabbaabb||aaabbbaaaa|bbabb|| ,$$

és az előállított kódszó $(6, 3, b)$. Ezután

$$...abaabbaabbaaab||bbaaaabbab|babbb|| .$$

A megfelelő kódszó $(3, 4, b)$. Vegyük észre, hogy az illeszkedő részsorozat átterjedhet az előre néző ablakba.

A kódolás során sok finomságra kell figyelni. Ha egy jel még nem fordult elő a szövegben, az eltolás és a hossz értékét nullára állítjuk. Ha két egyforma hosszúságú illeszkedő részsorozatot találunk, akkor választanunk kell a két lehetséges eltolás között. Mindkét választásnak vannak előnyei. Kezdetben a történeti ablak lehet üres, és az előre néző ablak első jeleit kell tömöríteni – vannak más változatok is.

Az eredeti eljárás egyik gyakori módosítása, hogy csak az (eltolás, hossz) párokat állítjuk elő, és elhagyjuk a szöveg következő jelét. Ekkor külön figyelni kell arra az esetre, amikor a következő jel nem szerepel a történeti ablakban. Ilyenkor rendszerint a jelet tárolják, és így a dekódolónak meg kell különböztetnie a számpárokat az egyszerű jelektől. További változatokban nem feltétlenül a leghosszabb illeszkedő részsorozatot kódolják.

10.3.2. LZ78

Az LZ78 algoritmusban nem egy csúszó ablakot, hanem egy szótárt használunk, amelyet indexeket és bejegyzéseket tartalmazó táblázattal ábrázolunk. Az algoritmus felbontja a tömörítendő szöveget részsorozatok sorozatára úgy, hogy minden részsorozat az addigi részsorozatok közül a leghosszabb illeszkedő α részsorozat és a tömörítendő szöveg ezt követő s jele. Az αs új részsorozattal kiegészítjük a szótárt. Az új bejegyzés formája (i, s), amelyben i a létező α bejegyzés indexe, és s az ehhez fűzött jel.

Példaként tekintsük az *abaabbaabbaaabbbaaaabba* sorozatot. Ezt az LZ78 algoritmus a következő részsorozatokra bontja. Az üres sorozatnak 0 felel meg.

Bemenet	a	b	aa	bb	aab	ba	$aabb$	baa	$aabba$
Index	1	2	3	4	5	6	7	8	9
Eredmény	$(0, a)$	$(0, b)$	$(1, a)$	$(2, b)$	$(3, b)$	$(2, a)$	$(5, b)$	$(6, a)$	$(7, a)$

A részsorozatok elvileg tetszőlegesen távolra hivatkozhatnak, hiszen nem használunk csúszó ablakot. Ugyanakkor a gyakorlatban a szótár mérete korlátozott. Több módon is kezelhető ez a probléma. Például, ha elértük a szótár maximális méretét, több bejegyzést nem adunk a táblázathoz, és a kódolás statikussá válik. Egy másik lehetőség a régi bejegyzések cseréje. A dekódoló tudja, hogy hány bitet kell fenntartani egy szótárbeli index ábrázolásához, így a dekódolás értelemszerű.

Helyesség

A Ziv–Lempel-kódolás aszimptotikusan eléri a lehető legjobb tömörítési arányt, ami a forrás entrópia aránya. A forrásmodell ugyanakkor jóval általánosabb, mint a diszkrét, emlékezet nélküli forrás. A következő jelet előállító sztochasztikus folyamat a feltételezés szerint állandó. (Egy sorozat valószínűsége független az időpillanattól, azaz $P(X_1 = x_1 \dots X_n = x_n) = P(X_{t+1} = x_1 \dots X_{t+n} = x_n)$ minden t és minden $(x_1 \dots x_n)$ sorozat esetén.) Állandó folyamatok esetén a $\lim_{n\to\infty}(1/n)H(X_1 \dots X_n)$ határérték létezik, és ezt nevezzük entrópia aránynak.

Ha $s(n)$ jelöli az LZ78 által létrehozott részsorozatok számát egy állandó forrás által generált szöveg esetén, akkor ez ezen részsorozatok kódolásához szükséges bitek száma $s(n) \cdot (\lg s(n) + 1)$. Belátható, hogy $(s(n) \cdot (\lg s(n) + 1))/n$ a forrás entrópia arányához tart, ha az összes sorozatot tárolhatjuk a szótárban.

Elemzés

Ha rögzítjük a csúszó ablak, illetve a szótár méretét, akkor az n jelből álló sorozat kódolásának futási ideje arányos n értékével. Ugyanakkor az adattömörítések esetén megszokott módon kompromisszumot kell kötni a tömörítési arány és a sebesség között. Jobb tömörítés csak nagyobb memória használatával érhető el. Az ablak, illetve a szótár méretének növelése viszont lassítja a kódolást, mert a leginkább időigényes feladat az illeszkedő részsorozat, illetve szótárbeli index megkeresése.

Akár LZ77, akár LZ78 esetén a dekódolás értelemszerű. LZ77 esetén a dekódolás sokkal gyorsabb a kódolásnál, mert a dekódoló csak felhasználja azt az adatot, hogy a történeti ablak melyik pozícióján kezdődik az előállítandó részsorozat, míg a kódolónak meg kell találnia a leghosszabb illeszkedő részsorozatot a történeti ablakban. Ennélfogva az LZ77 módszeren alapuló algoritmusok hasznosak, ha egy fájlt egyszer kell tömöríteni és gyakran kell kitömöríteni.

A kódolt szöveg nem feltétlen rövidebb az eredeti szövegnél. Különösen a kódolás elején, a kódolt forma jóval hosszabb lehet. Ezt a tulajdonságot figyelembe kell venni.

A megvalósítás során nem optimális, ha a szöveget tömbként ábrázoljuk. Az LZ77 algoritmusban az előre néző ablak esetén egy körbe fűzött sor, a történeti ablak esetén pedig egy bináris keresőfa a megfelelő adatszerkezet. LZ78 alkalmazásakor egy szótárfa használata előnyös.

Gyakorlatok

10.3-1. Alkalmazzuk az LZ77 és LZ78 algoritmusokat az *abracadabra* szövegre.

10.3-2. Milyen típusú fájlokat lehet jól tömöríteni az LZ77, illetve az LZ78 algoritmussal? Milyen típusú fájlok esetén nem előnyös az LZ77, illetve az LZ78 tömörítés?

10.3-3. Elemezzük az első, illetve utolsó eltolás használatának előnyét az LZ77 kódolásban, ha több illeszkedő részsorozat található.

10.4. Burrows–Wheeler-transzformáció

A ***Burrows–Wheeler-transzformációt*** egy példán keresztül mutatjuk be. Tegyük fel, hogy az eredeti szöveg $\vec{X}$ = „WHEELER”. Ezt a szöveget megfeleltetjük egy másik $\vec{L}$ szövegnek és egy I indexnek, a következő szabályok szerint:

1) Előállítjuk az M mátrixot, amelyik az eredeti $\vec{X}$ szöveg összes ciklikus eltolását tartalmazza. Esetünkben

$$M = \begin{pmatrix} W & H & E & E & L & E & R \\ H & E & E & L & E & R & W \\ E & E & L & E & R & W & H \\ E & L & E & R & W & H & E \\ L & E & R & W & H & E & E \\ E & R & W & H & E & E & L \\ R & W & H & E & E & L & E \end{pmatrix}.$$

2) Az M mátrix alapján létrehozzuk az új M' mátrixot, amelyben M sorai lexikografikus sorrendben szerepelnek. A példában ez a mátrix

$$M' = \begin{pmatrix} E & E & L & E & R & W & H \\ E & L & E & R & W & H & E \\ E & R & W & H & E & E & L \\ H & E & E & L & E & R & W \\ L & E & R & W & H & E & E \\ R & W & H & E & E & L & E \\ W & H & E & E & L & E & R \end{pmatrix}.$$

3) A megfeleltetés $\vec{L}$ sorozata az M' mátrix utolsó oszlopa, indexe azon sor I indexe M' mátrixban, amelyik tartalma megegyezik az eredeti sorozattal. A példánkban $\vec{L}$ = „HELWEER” és $I = 6$ – az oszlopok indexelése nullával kezdődik.

Ennek megfelelően a következő algoritmushoz jutunk. Ebben X jelölést alkalmazzuk $\vec{X}$ helyett, és L jelölést $\vec{L}$ helyett, ugyanis a vektor jelölés célja csak az volt, hogy a szövegben megkülönböztessük a vektorokat a betűktől.

Az algoritmus bemenete az $X[0..n-1]$ tömb, kimenete pedig az $L[0..n-1]$ tömb és a I index.

BWT-KÓDOLÓ($X[0..n-1]$)

```
 1  for j ← 0 to n − 1
 2      do M[0, j] ← X[j]
 3  for i ← 0 to n − 1
 4      do for j ← 0 to n − 1
 5             do M[i, j] ← M[i − 1, j + 1 mod n]
 6  for i ← 0 to n − 1
 7      do M′ i-edik sor ← M i-edik sora lexikografikus sorrendben
 8  for i ← 0 to n − 1
 9      do L[i] ← M′[i, n − 1]
10  i ← 0
11  while M′ i-edik sor ≠ X
12      do i ← i + 1
13  I ← i
14  return L és I
```

Belátható, hogy ez a transzformáció invertálható, azaz $\vec{L}$ sorozatból és I indexből megkaphatjuk az eredeti $\vec{X}$ szöveget. Ez a két paraméter – a sorozat és az index – ugyanis elegendő információt tartalmaz ahhoz, hogy megkapjuk a jelek megfelelő permutációját. A helyreállítást az előző példán szemléltetjük. Az átalakított $\vec{L}$ sorozatból a jeleket növekvően rendezve előállítjuk az $\vec{E}$ sorozatot. $\vec{E}$ az M' mátrix első oszlopa. A példában

$$\vec{L} = H \quad E \quad L \quad W \quad E \quad E \quad R$$

$$\vec{E} = E \quad E \quad E \quad H \quad L \quad R \quad W\,.$$

Az eredeti $\vec{X}$ sorozat $\vec{X}(0)$ első jele a rendezett $\vec{E}$ sorozat I-edik pozícióján található jel, esetünkben $\vec{X}(0) = \vec{E}(6) = W$. Ezután megkeressük ezt a jelet az $\vec{L}$ sorozatban – most csak egyetlen W szerepel benne. Ez a 3-adik pozíción fordul elő az $\vec{L}$ sorozatban. Ez az index megadja az eredeti szöveg következő jelének helyét, azaz $\vec{X}(1) = \vec{E}(3) = H$. H jelet a 0. pozíción találtuk meg az $\vec{L}$ sorozatban, így $\vec{X}(2) = \vec{E}(0) = E$. Most három E jel szerepel $\vec{L}$ sorozatban, amelyek közül kiválasztjuk az elsőt, amit még nem használtunk. Esetünkben ez az 1 pozíciót eredményezi, így $\vec{X}(3) = \vec{E}(1) = E$. Ezt az eljárást folytatjuk és azt kapjuk, hogy $\vec{X}(4) = \vec{E}(4) = L$, $\vec{X}(5) = \vec{E}(2) = E$, $\vec{X}(6) = \vec{E}(5) = R$.

Így a következő algoritmushoz jutunk, melynek bemenete az $L[0..n-1]$ tömb és az I index, kimenete pedig az $X[0..n-1]$ tömb.

BWT-DEKÓDOLÓ(L, I)

```
1  E[0..n − 1] ← L[0..n − 1] rendezése
2  pi[−1] ← I
```

```
3  for i ← 0 to n − 1
4      do j ← 0
5         while L[j]) ≠ E[π[i − 1]] ∨ j eleme π-nek
6               do j ← j + 1
7                  π[i] ← j
8                  X[i] ← L[j]
9  return X
```

Az algoritmust ki kell egészítenünk a következő leírással. A dekóder csak az $\vec{L}$ sorozatot ismeri, ezért ennek rendezésével elő kell állítani az $\vec{E}$ sorozatot. Az algoritmusban előállítjuk a π vektort úgy, hogy minden $\vec{L}$ sorozatbeli $\vec{L}(j)$ jelre feljegyezzük azt a $\pi(j)$ pozíciót az $\vec{E}$ sorozatból, ahonnan a leírt eljárással ide ugrottunk. A π vektor egy olyan π permutációt határoz meg, amelyre igaz, hogy az M mátrixban $\vec{L}(j) = \vec{E}(\pi(j))$ minden $j = 0, \ldots, n-1$ értékre. A példában $\pi = (3, 0, 1, 4, 2, 5, 6)$. A permutáció felhasználásával helyreállíthatjuk az n hosszúságú $\vec{X}$ szöveget az $\vec{X}(n-1-j) = \vec{L}(\pi^j(I))$ összefüggés alapján, ahol $\pi^0(x) = x$ és $\pi^j(x) = \pi(\pi^{j-1}(x))$, ha $j = 1, \ldots, n-1$.

Az eddigiek során csak átalakítottuk az eredeti szöveget, de nem tömörítettük, hiszen az $\vec{L}$ sorozat ugyanolyan hosszú, mint az eredeti $\vec{X}$ sorozat. Akkor mi a Burrows–Wheeler-transzformáció előnye? Azt várjuk, hogy az átalakított sorozatot az eredetinél sokkal hatékonyabban lehet kódolni. A jelek közötti függőségek vizsgálata során azt tapasztaljuk, hogy az átalakított $\vec{L}$ sorozat hosszú, azonos jelből álló blokkokat tartalmaz.

Azonos jelből álló gyakori blokkok kihasználására Burrows és Wheeler az ***előremozgató kód*** használatát javasolta. Ezt az eddig tárgyalt példán szemléltetjük.

Elkészítjük a szövegben szereplő jelek ábécé szerint rendezett listáját. A listában a jeleket a pozíciójukkal indexeljük.

E	*H*	*L*	*R*	*W*
0	1	2	3	4 .

Jelenként végigolvassuk az átalakított $\vec{L}$ sorozatot, feljegyezzük a következő jel indexét, és ezt a jelet a lista elejére helyezzük. A példában az első lépésben feljegyezzük 1-et, *H* jel indexét, és a *H* jelet előre visszük. Az új lista:

H	*E*	*L*	*R*	*W*
0	1	2	3	4 .

Ezután feljegyezük 1-et, és előre visszük az *E* jelet:

E	*H*	*L*	*R*	*W*
0	1	2	3	4 .

Feljegyezzük 2-t, és az *L* jelet előre visszük:

L	*E*	*H*	*R*	*W*
0	1	2	3	4 .

Feljegyezzük 4-et, és *W*-t előre visszük,

W	*L*	*E*	*H*	*R*
0	1	2	3	4 .

Feljegyezzük 2-t, és előre visszük E-t:

E	W	L	H	R
0	1	2	3	4 .

Feljegyezzük 0-t, és elől hagyjuk E-t:

E	W	L	H	R
0	1	2	3	4 .

Feljegyezzük 4-et, és előre visszük R-et:

R	E	W	L	H
0	1	2	3	4 .

Az előremozgató kód eredménye az $(1, 1, 2, 4, 2, 0, 4)$ sorozat. Az algoritmus a következő, ahol Q az $\vec{L}$ sorozatban előforduló jelek listája, az algoritmus bemenete az $L[0\,..\,n-1]$ tömb, kimenete pedig a $c[0\,..\,n-1]$ tömb.

ELŐRE-MOZGATÁS(L)

```
1  Q[0..n − 1] ← az L-ben előforduló m jel ábécé szerint rendezve
2  for i ← 0 to n − 1
3      do j ← 0
4          while j ≠ L[i]
5                do j ← j + 1
6      c[i] ← j
7  for l ← 0 to j
8      do Q[l] ← Q[l − 1 mod j + 1]
9  return c
```

Az előremozgató kódolással előállított c kódot tömörítjük, például Huffman-algoritmussal.

Helyesség
A tömörítés az átalakított $\vec{L}$ sorozat előremozgató kódolásának következménye. Látható, hogy az előremozgató kódolás invertálható, azaz az előállított kódból helyreállítható az $\vec{L}$ sorozat.

Megfigyelhető, hogy az előre–mozgató kódolással előállított sorozatban a kis értékek gyakrabban fordulnak elő. Sajnos, ez csak a példánknál jóval hosszabb szövegek esetén válik láthatóvá. Megfigyelések alapján hosszú sorozatokban akár a számok 70 százaléka is 0 lehet. Az eloszlás specialitását kihasználhatjuk, ha az előremozgató kódolással előállított sorozatot tömörítjük, például Huffman- vagy futamhossz-kódolással (RLE).

Az algoritmus teljesítménye a gyakorlatban nagyon jó, akár a tömörítési arányt, akár a sebességet vizsgáljuk. A tömörítés aszimptotikus optimalitását bebizonyították források széles körére.

Elemzés
A Burrows–Wheeler-transzformáció leginkább műveletigényes része az átalakított $\vec{L}$ sorozat előállításhoz szükséges rendezés. A gyors rendezési algoritmusok miatt, amelyek speciálisan a tömörítendő adatok típusához illeszkednek, a Burrows–Wheeler-transzformációt

felhasználó tömörítő algoritmusok rendszerint nagyon gyorsak. Másrészt, ezek az algoritmusok blokkonként tömörítenek. A tömörítendő szöveget megfelelő méretű blokkokra kell osztani. A blokkok méretét megszabja, hogy az M és M' mátrixoknak el kell férniük a memóriában. A dekódolónak meg kell várnia, hogy a teljes következő blokk a rendelkezésére álljon, nem haladhat jelenként, mint például aritmetikai, vagy Ziv–Lempel kódolás esetén.

Gyakorlatok

10.4-1. Alkalmazzuk a Burrows–Wheeler-transzformációt és az előre-mozgató kódolást az *abracadabra* szövegre.

10.4-2. Lássuk be, hogy az átalakított $\vec{L}$ sorozat és az $\vec{E}$ rendezett sorozatban az eredeti szöveg első jelének pozícióját megadó i index alapján valóban helyre lehet állítani az eredeti szöveget.

10.4-3. Szemléltessük, hogy a példánkban a dekóder miként állítja elő az $\vec{L} = HELWEER$ sorozatot az előremozgató kódolásból származó $(1, 1, 2, 4, 2, 0, 4)$ sorozat és a szövegben előforduló E, H, L, W, R jelek alapján. Írjuk le az előremozgató kódot dekódoló általános eljárást.

10.4-4. Ebben a részben a Burrows és Wheeler által megadott kódoló eljárást mutattuk be. Előállítható-e az átalakított $\vec{L}$ sorozat a kódolás során anélkül, hogy létrehoznánk az M és M' mátrixokat?

10.5. Képtömörítés

A képtömörítő algoritmusok alapötlete hasonlít a Burrows–Wheeler-transzformációban alkalmazott módszerre. A tömörítendő szöveget olyan formára hozzák, amit az előző részekben leírt módszerekkel tömöríteni lehet, például Huffman vagy aritmetikai kódolással. Több eljárás is létezik a kép típusának (fekete-fehér, szürke árnyalatos vagy színes), illetve a tömörítés módjának (veszteségmentes vagy veszteséges) megfelelően. A veszteséges képtömörítések, mint például a ***JPEG*** szabvány, alapvető elemeit mutatjuk be: az adatok ábrázolását, a diszkrét koszinusz transzformációt, kvantálást, kódolást.

10.5.1. Adatábrázolás

Egy szürkeárnyalatos képet egy X két dimenziós tömbbel ábrázolnak, amelyben minden $X(i, j)$ elem a kép (i, j) pontjának intenzitását (fényességét) adja meg. $X(i, j)$ előjeles vagy előjel nélküli, k bites egész, azaz $X(i, j) \in \{0, \ldots, 2^k - 1\}$, vagy $X(i, j) \in \{-2^{k-1}, \ldots, 2^{k-1} - 1\}$.

Egy színes kép egy pontját rendszerint három szürkeárnyalatos értékkel ábrázolják. Ezek $R(i, j)$, $G(i, j)$, $B(i, j)$, és megfelelnek az adott pont piros, zöld, kék szín szerinti intenzitásának.

A kép tömörítése során először az R, G, B tömböket (csatornákat) kell elemenként a fényesség/színesség térbe konvertálnunk az $\mathbf{YC_bC_r}$***-transzformációval.***

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.5 \\ 0.5 & -0.419 & -0.0813 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}.$$

$Y = 0.299R + 0.587G + 0.114B$ a csatorna fényessége, vagy intenzitása. A színeket súlyozó együtthatókat tapasztalati úton határozták meg úgy, hogy az emberi szem által érzékelt intenzitást a lehető legjobban közelítsék. A $C_b = 0.564(B - Y)$ és $C_r = 0.713(R - Y)$ színesség csatornák tartalmazzák a piros és kék színekre vonatkozó információt az Y fényességtől való eltérés függvényében. A zöld színre vonatkozó értéket nagyrészt az Y fényességből nyerik ki.

A színes képek tömörítésének első lépése az YC_bC_r-transzformációt követi, amikor a *lényegtelen információt* elhagyjuk. Az emberi szem kevésbé érzékeli a színek gyors változását az intenzitás változásához képest, ezért a C_b és C_r színességi csatornák felbontását vízszintes és függőleges irányban a felére csökkentik, aminek eredményeként az előálló tömbök mérete negyede az eredeti tömbök méretének.

Ezután a tömböket 8×8-as blokkokra bontják, és ezekre alkalmazzák a tényleges (veszteséges) adat tömörítési eljárást.

Vizsgáljuk meg a következő, valós kép alapján előállított példán a tömörítés lépéseit. A következő 8×8 méretű, előjel nélküli 8 bites egészeket tartalmazó blokk felel meg a kép egy részének:

$$f = \begin{pmatrix} 139 & 144 & 149 & 153 & 155 & 155 & 155 & 155 \\ 144 & 151 & 153 & 156 & 159 & 156 & 156 & 155 \\ 150 & 155 & 160 & 163 & 158 & 156 & 156 & 156 \\ 159 & 161 & 162 & 160 & 160 & 159 & 159 & 159 \\ 159 & 160 & 161 & 161 & 160 & 155 & 155 & 155 \\ 161 & 161 & 161 & 161 & 160 & 157 & 157 & 157 \\ 162 & 162 & 161 & 163 & 162 & 157 & 157 & 157 \\ 161 & 162 & 161 & 161 & 163 & 158 & 158 & 158 \end{pmatrix}.$$

10.5.2. A diszkrét koszinusz transzformáció

Minden $(f(i, j))_{i,j=0,\ldots,7}$ 8×8-as blokkot egy új $(F(u, v))_{u,v=0,\ldots,7}$ blokká alakítunk. Több lehetséges átalakítás használható, rendszerint a ***diszkrét koszinusz transzformációt*** használják, amely esetünkben a következő képlettel adható meg:

$$F(u, v) = \frac{1}{4} c_u c_v \left(\sum_{i=0}^{7} \sum_{j=0}^{7} f(i, j) \cdot \cos \frac{(2i + 1)u\pi}{16} \cos \frac{(2j + 1)v\pi}{16} \right).$$

A transzformációt megelőzően az előjel nélküli egészeket előjelessé alakítják, 2^{k-1} értékkel csökkentve azokat.

DCT(f)

1 **for** $u \leftarrow 0$ **to** 7
2 **do for** $v \leftarrow 0$ **to** 7
3 **do** $F(u, v) \leftarrow$ az f mátrix DCT-együtthatója
4 **return** F

Az együtthatókat nem szükséges a megadott képlettel kiszámítani, hanem azokat megkaphatjuk egy megfelelő Fourier-transzformáció (lásd gyakorlatok) segítségével is, így a

számításra gyors Fourier-transzformációt használhatunk. A JPEG szabvány támogatja a hullám transzformációt is, amellyel ebben a lépésben helyettesíthetjük a diszkrét koszinusz transzformációt.

A diszkrét koszinusz transzformáció invertálható az

$$f(i,j) = \frac{1}{4}\left(\sum_{u=0}^{7}\sum_{v=0}^{7} c_u c_v F(u,v) \cdot \cos\frac{(2i+1)u\pi}{16}\cos\frac{(2j+1)v\pi}{16}\right)$$

összefüggés alapján, amelyben

$$c_u = \begin{cases} \frac{1}{\sqrt{2}}, & \text{ha } u = 0\,, \\ 1, & \text{ha } u \neq 0 \end{cases}$$

és

$$c_v = \begin{cases} \frac{1}{\sqrt{2}}, & \text{ha } v = 0\,, \\ 1, & \text{ha } v \neq 0 \end{cases}$$

a normalizáló konstansok.

A példában a kerekített értékeket tartalmazó F transzformált blokk

$$F = \begin{pmatrix} 235.6 & -1.0 & -12.1 & -5.2 & 2.1 & -1.7 & -2.7 & 1.3 \\ -22.6 & -17.5 & -6.2 & -3.2 & -2.9 & -0.1 & 0.4 & -1.2 \\ -10.9 & -9.3 & -1.6 & 1.5 & 0.2 & -0.9 & -0.6 & -0.1 \\ -7.1 & -1.9 & 0.2 & 1.5 & 0.9 & -0.1 & 0.0 & 0.3 \\ -0.6 & -0.8 & 1.5 & 1.6 & -0.1 & -0.7 & 0.6 & 1.3 \\ 1.8 & -0.2 & 1.6 & -0.3 & -0.8 & 1.5 & 1.0 & -1.0 \\ -1.3 & -0.4 & -0.3 & -1.5 & -0.5 & 1.7 & 1.1 & -0.8 \\ -2.6 & 1.6 & -3.8 & -1.8 & 1.9 & 1.2 & -0.6 & -0.4 \end{pmatrix}.$$

A diszkrét koszinusz transzformáció közeli kapcsolatban áll a diszkrét Fourier-transzformációval, és hasonlóan rendel jeleket a frekvenciákhoz. Magasabb frekvenciák eltávolítása kevésbé éles képet eredményez. Ez a hatás elfogadható, így a magasabb frekvenciákat kevésbé pontosan tárolják.

Az $F(0,0)$ érték speciális jelentőséggel bír, ugyanis ez a teljes blokk intenzitás mértékeként értelmezhető.

10.5.3. Kvantálás

A diszkrét koszinusz transzformáció az egészeket valósakká alakítja. A valós értékeket az ábrázolhatóság miatt kerekíteni kell. Természetesen a kerekítés információvesztéshez vezet. Ugyanakkor az átalakított F blokk sokkal jobban kezelhető. ***Kvantálást*** alkalmaznak, amely F értékeit egészekké alakítja úgy, hogy a Q fényességi kvantált mátrix megfelelő elemével osztjuk az értéket. A példában a

$$Q = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}$$

mátrixot használjuk.

A kvantálási mátrixot körültekintően kell megválasztani, hogy a kép minősége a lehető legjobb maradjon. A kvantálás a tömörítés veszteséges része. Az ötlet, hogy eltávolítjuk a „vizuálisan nem jelentős” információt. Ebben a lépésben egyensúlyt kell tartani a tömörítési arány és a dekódolt kép minősége között. Ennélfogva a JPEG esetén a kvantálási táblázat nem része a szabványnak, hanem azt meg kell adni (és így kódolni is).

KVANTÁLÁS(F)

```
1  for i ← 0 to 7
2      do for j ← 0 to 7
3             do T(i, j) ← F(i, j)/Q(i, j)
4  return F
```

A kvantálás az F blokkból előállítja az új T blokkot a $T(i, j) = F(i, j)/Q(i, j)$ összefüggés alapján, ahol $\{x\}$ az x értékhez legközelebb eső egész érték. Végül ezt a blokkot kódolják. Vegyük észre, hogy a transzformált F blokkban az $F(0, 0)$ érték kivételével minden érték viszonylag kicsi. Ennek következményeként a T legtöbb eleme 0 lesz:

$$T = \begin{pmatrix} 15 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

A $T(0, 0)$ együttható, esetünkben 15, megkülönböztetést igényel. ***Egyenáram együtthatónak*** (DC) nevezzük, a többi értéket pedig ***váltóáram együtthatónak*** (AC).

10.5.4. Kódolás

A T mátrixot Huffman-kóddal tömörítjük. Csak vázoljuk az eljárást. Először a DC tagot kódoljuk úgy, hogy az előzőleg kódolt blokk DC tagjától vett eltérést tároljuk. Ha a példában az előző blokk DC tagja 12 volt, akkor $T(0, 0)$ értéket -3 értékkel kódoljuk.

Ezután az AC együtthatókat kódoljuk cikkcakk sorrendben, azaz $T(0, 1)$, $T(1, 0)$, $T(2, 0)$, $T(1, 1)$, $T(0, 2)$, $T(0, 3)$, $T(1, 2)$, stb. sorrendben. A példában ez a $0, -2, -1, -1, -1, 0, 0, -1$ sorozatot, és az ezt követő 55 nullát jelenti. A cikkcakk sorrend

kihasználja az a tulajdonságot, hogy egymás utáni nullákból álló hosszú sorozatok jönnek létre. Ezeket a sorozatokat nagyon hatékonyan ábrázolhatjuk ***futamhossz kódolással***, amelyben megadjuk a következő, nullától különböző elemig tartó nullák számát, a nullától különböző elemet.

Az egészeket úgy tároljuk, hogy a kisebb értékeket rövidebben ábrázoljuk. Ennek érdekében az egészek ábrázolását két részre osztjuk: a méretre (a fenntartott bitek száma) és az amplitúdóra (a tényleges érték). Így a 0 mérete 0, 1 és -1 mérete 1; -3, -2, 2 és 3 mérete 2 stb.

A példában a (2)(3) sorozat felel meg a DC tagnak, amelyet $(1,2)(-2)$, $(0,1)(-1)$, $(0,1)(-1)$, $(0,1)(-1)$, $(2,1)(-1)$, és végül $(0,0)$ követ. Az utolsó elem a blokk vége jel, ami jelöli, hogy ezután már csak nullák következnek. Például $(1,2)(-2)$ azt jelenti, hogy 1 nullát egy 2 méretű elem követ, amelynek amplitúdója -2.

A párokhoz a Huffman-algoritmus alapján rendelünk kódszavakat. Eltérő Huffman-kódok tartoznak a (hossz, méret) párokhoz és az amplitúdókhoz. Ezeket a Huffman-kódokat meg kell határozni, és be kell illeszteni a kép kódjába.

Egy 8×8-as T blokk kódolását adja meg a következő algoritmus. Ebben kód-1, kód-2, kód-3 jelöli az eltérő Huffman-kódokat, || pedig az összeláncolást

FUTAMHOSSZ-KÓD(T)

```
 1  c ← kód-1(méret(DC − T[0,0]))
 2  c ← c|| kód-2(amplitúdó(DC − T[0,0]))
 3  DC ← T[0,0]
 4  for l ← 1 to 14
 5      do for i ← 0 to l
 6             do  if l = 1  mod 2
 7                    then u ← i
 8                    else u ← l − i
 9                 if T[u, l − u] = 0
10                    then futam ← futam + 1
11                    else c ← c|| kód-2(futam, méret(T[u, l − u]))
12                         c ← c|| kód-3(amplitúdó(T[u, l − u])
13                         futam ← 0
14  if futam > 0
15    then kód-2(0,0)
16  return c
```

A dekódolás során a T mátrixot állítjuk helyre. Ezután minden $T(i, j)$ értéket megszorozva a Q kvantálási mátrix megfelelő $Q(i, j)$ elemével az F blokk $\overline{F}$ becsléséhez jutunk. A példában

$$\overline{F} = \begin{pmatrix} 240 & 0 & -10 & 0 & 0 & 0 & 0 & 0 \\ -24 & -12 & 0 & 0 & 0 & 0 & 0 & 0 \\ -14 & -13 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Az inverz koszinusz transzformációt alkalmazzuk $\overline{F}$-re, így létrehozzuk az eredeti kép 8×8-as f blokkját becslő $\overline{f}$ blokkot. A példában

$$\overline{f} = \begin{pmatrix} 144 & 146 & 149 & 152 & 154 & 156 & 156 & 156 \\ 148 & 150 & 152 & 154 & 156 & 156 & 156 & 156 \\ 155 & 156 & 157 & 158 & 158 & 157 & 156 & 155 \\ 160 & 161 & 161 & 162 & 161 & 159 & 157 & 155 \\ 163 & 163 & 164 & 163 & 162 & 160 & 158 & 156 \\ 163 & 164 & 164 & 164 & 162 & 160 & 158 & 157 \\ 160 & 161 & 162 & 162 & 162 & 161 & 159 & 158 \\ 158 & 159 & 161 & 161 & 162 & 161 & 159 & 158 \end{pmatrix}.$$

Gyakorlatok

10.5-1. Adjuk meg az 5, -19 és 32 egészek ábrázolásához használható méret és amplitúdó értékeket.

10.5-2. Írjuk fel a következő mátrix elemeit cikkcakk sorrendben:

$$\begin{pmatrix} 5 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Hogyan kellene ezt a mátrixot kódolni, ha a DC tag és előző DC tag különbsége -2?

10.5-3. A példában a kvantálást követően a (2)(3), $(1,2)(-2)$, $(0,1)(-1)$, $(0,1)(-1)$, $(0,1)(-1)$, $(2,1)(-1)$, $(0,0)$ sorozatot kell kódolni. Tegyük fel, hogy a Huffman-kód a 011 kódszót rendeli az előző blokk DC különbségéből származó 2-höz, 0, 01, illetve 11 kódokat a -1, -2, illetve 3 amplitúdókhoz, és 1010, 00, 11011, illetve 11100 kódokat a (0,0), (0,1), (1,2), illetve (2,1) párokhoz. Mi lesz a példa 8×8-as blokkját ábrázoló bitsorozat? Hány bit szükséges a blokk kódolásához?

10.5-4. Mi lenne a T, $\overline{F}$ és $\overline{f}$ mátrixok értéke, ha a példában a

$$Q = \begin{pmatrix} 8 & 6 & 5 & 8 & 12 & 20 & 26 & 31 \\ 6 & 6 & 7 & 10 & 13 & 29 & 30 & 28 \\ 7 & 7 & 8 & 12 & 20 & 29 & 35 & 28 \\ 7 & 9 & 11 & 15 & 26 & 44 & 40 & 31 \\ 9 & 11 & 19 & 28 & 34 & 55 & 52 & 39 \\ 12 & 18 & 28 & 32 & 41 & 52 & 57 & 46 \\ 25 & 32 & 39 & 44 & 57 & 61 & 60 & 51 \\ 36 & 46 & 48 & 49 & 56 & 50 & 57 & 50 \end{pmatrix}$$

mátrixot használtuk volna kvantálásra a koszinusz transzformáció után?

10.5-5. Mi lenne az előző gyakorlatban a cikkcakk kód (feltéve, hogy az előző DC tagtól vett eltérés -3)?

10.5-6. Bármely $(f(n))_{n=0,\dots,m-1}$ sorozatra definiáljuk az $(\hat{f}(n))_{n=0,\dots,2m-1}$ sorozatot a következő módon

$$\hat{f}(n) = \begin{cases} f(n), & \text{ha } n = 0,\dots,m-1\ , \\ f(2m-1-n), & \text{ha } n = m,\dots,2m-1\ . \end{cases}$$

Ezt a sort kiterjeszthetjük Fourier-sorrá az

$$\hat{f}(n) = \frac{1}{\sqrt{2m}} \sum_{n=0}^{2m-1} \hat{g}(u) e^{i\frac{2\pi}{2m}nu}, \quad \text{ahol } \hat{g}(u) = \frac{1}{\sqrt{2m}} \sum_{n=0}^{2m-1} \hat{f}(u) e^{-i\frac{2\pi}{2m}nu}, \quad i = \sqrt{-1}$$

összefüggéssel.

Mutassuk meg, miként állíthatók elő a diszkrét koszinusz transzformáció

$$F(u) = c_u \sum_{n=0}^{m-1} f(n) \cos(\frac{(2n+1)\pi u}{2m}, \quad c_u = \begin{cases} \frac{1}{\sqrt{m}}, & \text{ha } u = 0\ , \\ \frac{2}{\sqrt{m}}, & \text{ha } u \neq 0 \end{cases}$$

együtthatói a Fourier-sorból.

Feladatok

10-1. Adaptív Huffman-kód

A dinamikus és adaptív Huffman-kódolás a következő tulajdonságon alapul. Egy bináris kódfa rendelkezik a testvér tulajdonsággal, ha minden csúcsnak van testvére, és a csúcsok valószínűség szerint csökkenő sorrendben felsorolhatóak úgy, hogy minden csúcs a testvérével szomszédos. Mutassuk meg, hogy egy bináris prefix kód pontosan akkor Huffman-kód, ha a megfelelő kódfa kielégíti a testvér tulajdonságot.

10-2. A Kraft-egyenlőtlenség általánosításai

A Kraft-egyenlőtlenség bizonyításának lényeges eleme, hogy a hosszakat $L(1) \leq \cdots \leq L(a)$ sorrendbe rendezzük. Lássuk be, hogy 2, 1, 2 hosszúságok esetén rendezés nélkül nem állíthatjuk elő a prefix kódokat. A rendezetlen hosszúságok esete fordul elő a Shannon–Fano–Elias-kódban és az ábécé kódok elméletében, ami speciális keresési problémákkal áll kapcsolatban. Mutassuk meg, hogy ilyen esetben akkor, és csak akkor létezik $L(1) \leq \cdots \leq L(a)$

hosszúságú prefix kód, ha

$$\sum_{x \in \mathcal{X}} 2^{-L(x)} \leq 1/2 \, .$$

Nyitott probléma annak az igazolása, hogy ha az eddigieken kívül még azt is megköveteljük, hogy a prefix kódok szuffixmentesek legyenek, azaz egyetlen kódszó se legyen egy másik vége, a Kraft-egyenlőtlenség fennáll úgy, hogy a jobb oldalon szereplő 1 értéket (3/4)-re cseréljük, azaz

$$\sum_{x \in \mathcal{X}} 2^{-L(x)} \leq 3/4 \, .$$

10-3. A Krichevsky-Trofimov-becslés redundanciája
Lássuk be, hogy ha a Krichevsky-Trofimov-becslést olyan esetben használjuk, amikor egy diszkrét, emlékezet nélküli forrás θ paramétere ismeretlen, akkor minden x^n sorozat és $\theta \in \{0, 1\}$ esetén az x^n sorozat egyedi redundanciája legfeljebb $1/2 \lg n + 3$.

10-4. Az előremozgató kód alternatívái
Keressünk módszereket, amelyek az előremozgató kódhoz hasonlóan előkészítik a szöveget tömörítéshez a Burrows-Wheeler-transzformációt követően.

Megjegyzések a fejezethez

Az angol szövegekben előforduló jelek gyakoriság táblázatát Welsh cikke [491] tartalmazza. A Huffman-algoritmust Huffman tette közzé [212]. Az algoritmus megtalálható a [90] tankönyvben, ahol a Huffman-algoritmus példa egy speciális mohó algoritmusra. Ismertek a Huffman-kódolás adaptív vagy dinamikus változatai olyan esetekre, amikor a forrás valószínűségi eloszlását előre nem ismerjük. Ezek módosítják a Huffman-kódot, ha az aktuális gyakoriság értékek alapján az már nem lenne optimális.

Az aritmetikai kódolást Pasco [360] és Rissanen [388] fejlesztette ki. A megvalósítással kapcsolatos tudnivalókat tárgyalja [278, 387, 499]. A modellezésről szóló részben a Willems, Shtarkov és Tjalkens által alkalmazott megközelítést követtük [494]. A pontos számítások megtalálhatóak az eredeti cikkükben [493], amelyik az *IEEE Information Theory Society* legjobb cikk díját kapta 1996-ban.

A Lempel és Ziv által közzétett eredeti LZ77 és LZ78 [510, 511] algoritmusokat mutattuk be. Azóta több változatot, módosítást és kiterjesztést fejlesztettek ki, például a szótár, a mutatók, illetve a szótár betelési állapotának stb. kezelésére. Ezek leírása megtalálható például a [39] cikkben vagy a [40] könyvben. A legtöbb adattömörítő eszköz alapja a Ziv–Lempel-kódolás valamilyen változata. Például a „zip” és a „gzip” az LZ77 módszeren alapul, és az LZ78 egy változatát használja a „compress” program.

A Burrows–Wheeler-transzformációt a [63] technikai jelentésben írták le. Később népszerűvé vált, különösen, mert a Unix alatt használt „bzip” tömörítő ezen alapult, és számos szintmérő fájlon felülmúlta a legtöbb szótár alapú tömörítő programot. Nem használja az aritmetikai kódolást, amelynek esetén figyelni kell a szabadalmi jogokra. A Burrows–Wheeler-transzformáció további vizsgálatait mutatja be [28, 118, 269].

A veszteséges képtömörítéseknek csak az alapjait vázoltuk, kiemelve az adatok előkészítését, ami különböző módszerekhez, például a Huffman-kódoláshoz, szükséges. Részle-

tes leírás található a [462] könyvben, ahol a JPEG2000 szabvány is megtalálható. A bemutatott példát [483] tartalmazza.

A JPEG szabvány nagyon rugalmas. Például a veszteségmentes adattömörítést is támogatja. A képtömörítéssel foglalkozó részben tárgyalt témák nem egyediek. Léteznek több alapszínt kezelő és az YC_bC_r-transzformáción kívül további transzformációkat alkalmazó modellek. (Az YC_bC_r-transzformációban más értékeket is használnak a színességi csatornák meghatározásakor. Mi a [462] könyvben található adatokat használtuk.) A koszinusz transzformáció helyettesíthető más művelettel, például a hullámtranszformációval. Ezen kívül szabadon választhatjuk meg a kvantálási mátrixot, ami megszabja a tömörített kép és a Huffman-kód minőségét. Ugyanakkor ezeket a paramétereket explicit meg kell adnunk.

A videó- és hangtömörítések alapelve nagyon hasonlít a képtömörítések alapgondolatára. Lényegében ugyanazokból a lépésekből állnak. Ugyanakkor ilyenkor az adatok mennyisége jóval nagyobb. Ekkor is információt veszítünk az emberi szem vagy fül által nem érzékelhető elemek eltávolításával (például pszichoakusztikus modellek), illetve kvantálással, amikor a minőség nem romolhat számottevően. Ilyenkor finomabb kvantálási módszereket alkalmaznak.

Az adattömörítő algoritmusokra vonatkozó lényeges ismeretek megtalálhatók az információelmélettel foglalkozó könyvekben, például [91, 190], mert az elérhető tömörítési arány elemzéséhez szükséges a forráskódolás elméletének ismerete. Nemrégiben több adattömörítéssel foglalkozó könyv is megjelent, például [40, 191, 345, 408, 413], amelyeket az Olvasó figyelmébe ajánlunk. A Calgary Corpus és Canterbury Corpus szintmérő fájlok elérhetők a [66] vagy [67] helyeken.

Információelmélettel (és adattömörítéssel) kapcsolatos magyar nyelvű szakkönyv Bartha Tamás és Pataricza András [33], Csiszár Imre és Körner János [95], Györfi László, Győri Sándor és Vajda István [185], valamint Linder Tamás és Lugosi Gábor [291] munkája.

11. Memóriagazdálkodás

A mai számítógépekre az jellemző, hogy memóriájuk több – különböző kapacitású, hozzáférési idejű és költségű – szintből áll. A processzor számára közvetlenül elérhető szintet fizikai memóriának (vagy röviden memóriának), a további szinteket pedig háttértárnak nevezzük.

Rendszerint több program fut egyidejűleg, amelyeknek együttes tárigénye nagyobb, mint a fizikai memória kapacitása. Ezért a memóriát a folyamatok között el kell osztani.

Ebben a fejezetben a memóriagazdálkodás alapvető algoritmusaival foglakozunk. A 11.1. alfejezetben a statikus és dinamikus partícionálás, majd a 11.2 alfejezetben a lapozás legismertebb módszereit tekintjük át.

A 11.3. alfejezetben az operációs rendszerek történetének legnevezetesebb anomáliáit – a FIFO lapcserélési algoritmus, az átfedéses memória és a listás ütemező algoritmusok meglepő tulajdonságait – elemezzük.

Végül a 11.4. alfejezetben annak az optimalizálási feladatnak a közelítő algoritmusait tekintjük át, melyben adott méretű fájlokat kell minimális számú mágneslemezen elhelyezni.

11.1. Partícionálás

A memória programok közötti elosztásának egyszerű módja, hogy a teljes címtartományt részekre bontjuk, és minden folyamat egy ilyen részt kap. Ezeket a részeket ***partícióknak*** nevezzük. Ehhez a megoldáshoz nincs szükség különösebb hardvertámogatásra, csupán az kell, hogy egy programot különböző címekre lehessen betölteni, azaz a programok ***áthelyezhetők*** legyenek. Erre azért van szükség, mert nem tudjuk garantálni, hogy egy program mindig ugyanabba a partícióba kerüljön, hiszen összességében szinte mindig több futtatható program van, mint amennyi a memóriába befér. Ráadásul azt sem tudjuk meghatározni, hogy mely programok futhatnak egyszerre, és melyek nem, hiszen a folyamatok általában egymástól függetlenek, sokszor különböző felhasználók a tulajdonosaik. Így még az is előfordulhat, hogy egyszerre többen futtatják ugyanazt a programot, és a példányok különböző adatokkal dolgoznak, tehát nem lehetnek ugyanott a memóriában. Az áthelyezés szerencsére könnyen elvégezhető, ha a szerkesztőprogram nem abszolút, hanem relatív címekkel dolgozik, azaz nem a pontos memóriabeli helyeket, hanem egy kezdőcímhez viszonyított eltolásokat használ minden címzésnél. Ezt a módszert ***báziscímzésnek*** nevezzük, ahol a kezdőcímet az úgynevezett *bázisregiszter* tartalmazza. A legtöbb processzor ismeri

ezt a címzési módot, ezért a program nem lesz lassabb, mintha abszolút címeket használna. Báziscímzés használatával az is elkerülhető, hogy a program egy hiba vagy szándékos felhasználói magatartás folytán valamely másik, alacsonyabb memóriabeli címeken elhelyezkedő program adatait kiolvassa, esetleg módosítsa. Ha a módszert kiegészítjük egy másik, úgynevezett *határregiszter* használatával, amely a legnagyobb megengedett eltolást, azaz a partíció méretét adja meg, akkor azt is biztosítani tudjuk, hogy egy program számára a többi, nála magasabb memóriabeli címeken elhelyezkedő program se legyen hozzáférhető.

A partícionálást régebben gyakran alkalmazták nagygépes operációs rendszerekben. A modern operációs rendszerek többsége azonban virtuális memóriakezelést használ, amihez viszont már különleges hardvertámogatásra van szükség.

A partícionálás, mint memóriafelosztási módszer azonban nemcsak operációs rendszerekben alkalmazható. Ha a gépi kódhoz közeli szinten programozunk, akkor előfordulhat, hogy különböző, változó méretű adatszerkezeteket – amelyek dinamikusan jönnek létre, illetve szűnnek meg – kell elhelyeznünk egy folytonos memóriaterületen. Ezek az adatszerkezetek hasonlítanak a folyamatokhoz, azzal a kivétellel, hogy olyan biztonsági problémákkal, mint a saját területen kívülre való címzés, nem kell foglalkoznunk. Ezért az alább felsorolandó algoritmusok nagy része kisebb-nagyobb módosításokkal hasznunkra lehet alkalmazások fejlesztésénél is.

Alapvetően kétféleképpen lehet felosztani egy címtartományt partíciókra. Az egyik módszer, hogy kezdetben a még üres címtartományt osztjuk fel előre meghatározott méretű és számú részre, és ezekbe próbáljuk meg folyamatosan belehelyezni a folyamatokat vagy más adatszerkezeteket, illetve eltávolítjuk belőlük, ha már nincs rájuk szükség. Ezek a rögzített partíciók, hiszen mind helyük, mind méretük előre, az operációs rendszer vagy az alkalmazás indulásakor rögzítve van. A másik módszer, hogy folyamatosan jelölünk ki darabokat a címtartomány szabad részéről az újonnan keletkező folyamatok vagy adatszerkezetek számára, illetve megszűnésükkor újra szabaddá nyilvánítjuk azokat. Ezt a megoldást hívjuk a dinamikus partíciók módszerének, mivel partíciók dinamikusan keletkeznek, illetve szűnnek meg. Mindkét módszernek vannak mind előnyei, mind hátrányai, és megvalósításuk teljesen különböző algoritmusokat igényel. Ezeket tekintjük át a következőkben.

11.1.1. Rögzített partíciók

A ***rögzített partíciók*** módszerénél a címtartomány felosztása előre rögzített, és futás közben nem változtatható. Operációs rendszerek esetén ez úgy történik, hogy a rendszergazda egy táblázatban meghatározza a partíciókat, és a rendszer következő betöltődésekor a felosztás alapja ez a táblázat. Amikor az első felhasználói alkalmazás elindul, a címtartomány már partícionálva van. Alkalmazásoknál a partícionálást akkor kell elvégezni, amikor még egyetlen adatszerkezet sincs a felosztandó memóriaterületen. Ezután a kész partíciókban lehet elhelyezni a különböző méretű adatszerkezeteket.

A továbbiakban csakis az operációs rendszerekbeli megvalósítást vizsgáljuk, a feladat és az algoritmusok átfogalmazását adott alkalmazáshoz az Olvasóra bízzuk, hiszen ezek alkalmazásfajtánként egymástól nagyon eltérőek is lehetnek.

A címtartomány felosztásakor azt kell figyelembe venni, hogy mekkora folyamatok és milyen arányban fognak a rendszerbe érkezni. Nyilvánvalóan van egy maximális méret, amelyet túllépő folyamatok nem futhatnak az adott rendszerben. A legnagyobb partíció mérete ezzel a maximummal egyezik meg. Az optimális partícionáláshoz gyakran statisztikai

felméréseket kell végezni, és a rendszer következő újraindítása előtt a partíciók méretét ezen statisztikák alapján kell módosítani. Ennek miként jével most nem foglalkozunk.

Mivel konstans számú (m) partíciónk van, adataikat tárolhatjuk egy vagy több állandó hosszúságú vektorban. A partíciók konkrét helyével absztrakt szinten nem foglalkozunk, feltételezzük, hogy ezt egy konstans vektor tárolja. Egy folyamatot úgy helyezünk el egy partícióban, hogy nem annak kezdőcímét, hanem sorszámát rögzítjük a folyamat leírójában. Természetesen a konkrét megvalósítás ettől eltérhet. A partíciók méretét a *méret*$[1..m]$ vektor tartalmazza. Folyamatainkat 1-től n-ig számozzuk. Azt, hogy egy partícióban épp mely folyamat fut, a *part*$[1..m]$ vektor tartalmazza, melynek párja a *hely*$[1..n]$ vektor, ami azt adja meg, hogy egy folyamat mely partícióban fut. Egy folyamat vagy fut, vagy pedig várakozik arra, hogy bekerüljön egy partícióba, és ott futhasson. Ezt az információt a *vár*$[1..n]$ vektor tartalmazza: ha az i-edik folyamat vár, akkor *vár*$[i]$ = IGAZ, egyébként pedig *vár*$[i]$ = HAMIS. A folyamatok helyigénye különbözik. A *helyigény*$[1..n]$ azt adja meg, hogy legalább mekkora partíció szükséges a folyamatok futtatásához.

Ha különböző méretű partícióink és különböző helyigényű folyamataink vannak, akkor nyilván nem szeretnénk, hogy a kis folyamatok a nagy partíciókat foglalják el, miközben a kisebb partíciók üresen maradnak, hiszen oda nem férnének be a nagyobb folyamatok. Ezért azt kell elérnünk, hogy minden partícióba az kerüljön betöltésre a várakozó folyamatok közül, amely oda befér, és nincs nála nagyobb, ami szintén beférne. Ezt biztosítja a következő algoritmus:

LEGNAGYOBB-BEFÉRŐ(*hely*, *helyigény*, *méret*, *part*, *vár*)

```
1  for j ← 1 to m
2      do if part[j] = 0
3           then BETÖLT-LEGNAGYOBB(hely, helyigény, méret, p, part, vár)
```

A legnagyobb olyan folyamat megtalálása, amelynek helyigénye nem nagyobb egy adott méretnél, egy egyszerű feltételes maximumkeresés. Amennyiben egyáltalán nem találunk a feltételnek megfelelő folyamatot, kénytelenek vagyunk a partíciót üresen hagyni.

BETÖLT-LEGNAGYOBB(*hely*, *helyigény*, *méret*, *p*, *part*, *vár*)

```
 1  max ← 0
 2  ind ← 0
 3  for i ← 1 to n
 4      do if vár[i] = IGAZ és max < helyigény[i] ≤ méret[p]
 5           then ind ← i
 6                max ← helyigény[i]
 7  if ind > 0
 8    then part[p] ← ind
 9         hely[ind] ← p
10         vár[ind] ← HAMIS
```

A folyamatokat partícióba töltő algoritmusok helyességének alapvető kritériuma, hogy ne töltsenek nagyobb folyamatot egy partícióba, mint amekkora belefér. Ezt a feltételt telje-

síti a fenti algoritmus, hiszen visszavezethető a feltételes maximumkeresés tételére pontosan az említett feltétellel.

Egy másik nagyon fontos kritérium, hogy ne töltsön egy partícióba több folyamatot, illetve ne töltsön egyetlen folyamatot több partícióba. Az első eset azért zárható ki, mert csakis azokra a partíciókra hajtjuk végre a BETÖLT-LEGNAGYOBB algoritmust, amelyekre $part[j] = 0$, és ha betöltünk egy folyamatot a p-edik partícióba, akkor $part[p]$-nek értékül adjuk a betöltött folyamat sorszámát, ami pozitív egész. A második eset bizonyítása is hasonló: a feltételes maximumkeresés feltétele kizárja azokat a folyamatokat, amelyekre $vár[i] =$ HAMIS, és ha az ind-edik folyamatot egynél többször betöltjük valamely partícióba, akkor $vár[ind]$-nek HAMIS értéket adunk.

Az, hogy az algoritmus nem tölt nagyobb folyamatot egyik partícióba sem, mint amekkora belefér, illetve hogy egyetlen partícióba sem tölt egynél több folyamatot és hogy egyetlen folyamatot sem tölt egynél több partícióba, még nem elég. Ezeket az üres algoritmus is teljesíti. Ezért mi többet követelünk: mégpedig azt, hogy ne hagyjon szabadon egyetlen partíciót sem, ha van olyan folyamat, amely beleférne. Ehhez szükségünk van egy invariánsra, amely az egész ciklus során fennáll, és a ciklus végén biztosítja ezt a feltételt is. Legyen az invariánsunk az, hogy j darab partíció megvizsgálása után nem létezik olyan pozitív $k \leq j$, amelyre $part[k] = 0$, és amelyre van olyan pozitív $i \leq n$, hogy $vár[i] =$ IGAZ és $helyigény[i] \leq méret[k]$.

Teljesül: Az algoritmus elején $j = 0$ darab partíciót vizsgáltunk meg, így egyáltalán nem létezik pozitív $k \leq j$.

Megmarad: Ha az invariáns teljesül j-re a ciklusmag elején, akkor először is azt kell látnunk, hogy ugyanerre a j-re a ciklus végén is teljesülni fog. Ez nyilvánvaló, hiszen az első j darab partícióhoz nem nyúlunk a $(j + 1)$-edik vizsgálata során, a bennük levő folyamatokra pedig $vár[i] =$ HAMIS, ami nem felel meg a BETÖLT-LEGNAGYOBB algoritmusban található feltételes maximumkeresés feltételének. A $(j + 1)$-edik partícióra pedig azért fog teljesülni az invariáns a ciklusmag végén, mert ha van a feltételnek megfelelő folyamat, azt a feltételes maximumkeresés biztosan megtalálja, hiszen a feltételes maximumkeresésünk feltétele megegyezik az invariánsunk egyes partíciókon értelmezett követelményével.

Befejeződik: Mivel a ciklus egyesével halad végig egy rögzített intervallumon, biztosan be is fog fejeződni. Mivel a ciklusmag pontosan annyiszor hajtódik végre, mint ahány partíció van, a ciklus befejeződésekor igaz lesz, hogy nem létezik olyan pozitív $k \leq m$, amelyre $part[k] = 0$, és amelyre van olyan pozitív $i \leq n$, hogy $vár[i] =$ IGAZ és $helyigény[i] \leq méret[k]$, azaz nem hagytunk szabadon egyetlen olyan partíciót sem, amelyben lenne folyamat, ami belefér.

A LEGNAGYOBB-BEFÉRŐ algoritmus 1–3. soraiban lévő ciklus mindig teljes egészében lefut, tehát a ciklusmag $\Theta(m)$-szer hajtódik végre. A ciklusmag egy feltételes maximumkeresést hajt végre az üres partíciókon – vagy azokon, amelyekre $part[j] = 0$. Mivel a BETÖLT-LEGNAGYOBB 4. sorában lévő feltételt minden j-re ki kell értékelni, ezért a feltételes maximumkeresés $\Theta(n)$ lépést igényel. Bár azokra a partíciókra, amelyekre $part[j] > 0$, a betöltő eljárás nem hívódik meg, a futási idő szempontjából legrosszabb esetben akár minden partíció üres lehet, ezért az algoritmus összes lépésszáma $\Theta(mn)$.

Az, hogy az algoritmus minden üres partícióba betölt egy várakozó folyamatot, ami befér, sajnos nem mindig elegendő. Gyakran szükség van arra is, hogy egy folyamat vala-

milyen megadott határidőn belül memóriához jusson. Ezt azonban a fenti algoritmus akkor sem biztosítja, ha az egyes folyamatok futási idejére felső korlátot tudunk adni. Amikor ugyanis újra és újra lefuttatjuk az algoritmust, mindig jöhetnek újabb folyamatok, amik kiszoríthatják a régóta várakozókat. Tekintsünk erre egy példát.

11.1. példa. Legyen két partíciónk, 5 kB és 10 kB méretekkel. Kezdetben két folyamatunk van, egy 8 és egy 9 kB helyigényű. Mindkét folyamat futása 2–2 másodpercig tart. Azonban az első másodperc végén megjelenik egy újabb, szintén 9 kB helyigényű és 2 másodperc futási idejű folyamat, és ez megismétlődik 2 másodpercenként, azaz a harmadik, az ötödik stb. másodpercekben. Ha most megvizsgáljuk az algoritmusunkat, akkor láthatjuk, hogy annak mindig két folyamat közül kell választania, és mindig a 9 kB helyigényű lesz a nyertes. A 8 kB-os, noha nincs más partíció, amelybe beférne, soha nem fog memóriához jutni.

Ahhoz, hogy a fent említett követelményt is teljesítsük, módosítani kell az algoritmusunkat: figyelembe kell vennünk, hogy mely folyamatok azok, amelyek már túl hosszú ideje várakoznak, és előnyben kell részesítenünk őket minden más folyamattal szemben akkor is, ha kisebb a helyigényük, mint azoknak. Az új algoritmusunk ugyanúgy megvizsgál minden partíciót, mint az előző.

LEGNAGYOBB-VAGY-RÉGÓTA-VÁRAKOZÓ-BEFÉRŐ(*hely*, *helyigény*, *küszöb*, *méret*, *part*, *vár*)

```
1  for j ← 1 to m
2      do if part[j] = 0
3           then BETÖLT-LEGNAGYOBB-VAGY-RÉGÓTA-VÁRAKOZÓ(hely, helyigény,
                    küszöb, méret, p, part, vár)
```

Azonban a betöltésnél meg kell vizsgálnunk minden folyamatra, hogy az mennyi ideje várakozik. Mivel az algoritmust mindig akkor futtatjuk, amikor egy vagy több partíció felszabadul, nem tudjuk a konkrét időt vizsgálni, csak azt, hogy hányszor nem tettük be egy olyan partícióba, amelybe befért volna. Ehhez módosítanunk kell a feltételes maximumkeresés algoritmusát: azokon az elemeken is műveletet kell végeznünk, amelyek ugyan teljesítik a feltételt (memóriára várakoznak és be is férnének), de nem a legnagyobbak ezek közül. Ez a művelet egy számláló növelése. A számlálóról feltételezzük, hogy a folyamat keletkezésekor a 0 értéket veszi fel. Ezen kívül a feltételt is kicsit módosítanunk kell: ha a számláló értéke egy elemnél túl magas (azaz egy bizonyos *küszöb* feletti), és az eddig talált legnagyobb helyigényű folyamat számlálójánál is nagyobb, akkor lecseréljük azt erre az elemre.

Az, hogy az algoritmus nem helyez több folyamatot ugyanabba a partícióba, ugyanúgy látható, mint az előző algoritmusnál, hiszen a külső ciklust és az abban található elágazás feltételét nem változtattuk meg. A másik két kritérium bizonyításához, azaz hogy nem kerül egy folyamat több partícióba vagy olyan partícióba, amibe nem fér, azt kell látnunk, hogy a feltételes maximumkeresés feltételét úgy alakítottuk át, hogy ez a tulajdonság megmaradt. Látható, hogy a feltételt kettébontottuk, így első része pontosan ez, amit követelünk, és ha ez nem teljesül, az algoritmus biztosan nem helyezi be a folyamatot a partícióba. Az a tulajdonság is megmarad, hogy nem hagyunk üresen partíciót, hiszen a feltételt, amely alapján kiválasztottuk a folyamatot, nem szűkítettük, hanem csak bővítettük, így ha az előző megtalált minden folyamatot, ami megfelel a kritériumnak, akkor az új algoritmus is megtalálja

ezeket. Csupán a kritériumnak megfelelő folyamatok közötti sorrend az, amit változtattunk. A ciklusok lépésszáma sem változott, mint ahogy az sem, hogy a belső ciklust milyen feltétel mellett kell elkezdeni végrehajtani. Tehát az algoritmus lépésszámának nagyságrendje is ugyanannyi, mint az eredeti algoritmusé.

Meg kell még vizsgálnunk, hogy az algoritmus teljesíti-e azt a feltételt, hogy egy folyamat csak meghatározott ideig várakozhat memóriára, amennyiben feltételezzük, hogy ismerünk valamilyen p felső korlátot a folyamatok futási idejére (ellenkező esetben a probléma megoldhatatlan, hiszen minden partíciót elfoglalhat egy-egy végtelen ciklus). Továbbá azt is minden esetben fel kell tételeznünk, hogy a rendszer nincs túlterhelve, azaz tudunk egy q felső becslést adni a várakozó folyamatok számára minden időpillanatban. Ekkor látható, hogy egy folyamatnak egy adott partícióba kerüléshez legrosszabb esetben meg kell várnia az előtte álló folyamatokat, azaz azokat, amelyeknek a számlálója nagyobb, mint az övé (legfeljebb q darab), valamint legfeljebb *küszöb* darab nála nagyobb folyamatot. Így valóban tudunk egy felső korlátot adni arra, hogy egy folyamat legfeljebb mennyi ideig várakozhat memóriára: $(q + küszöb)p$ ideig.

Az algoritmus pszeudokódja a következő.

BETÖLT-LEGNAGYOBB-VAGY-RÉGÓTA-VÁRAKOZÓ(*hely*, *helyigény*, *küszöb*, *méret*, *p*, *part*, *vár*)

```
1  max ← 0
2  ind ← 0
3  for i ← 1 to n
4      do if vár[i] = IGAZ és helyigény[i] ≤ méret[p]
5           then if (pont[i] > küszöb és pont[i] > pont[ind]) vagy helyigény[i] > max
6                  then pont[ind] ← pont[ind] + 1
7                       ind ← i
8                       max ← helyigény[i]
9                  else pont[i] ← pont[i] + 1
10 part[j] ← ind
11 hely[ind] ← p
12 vár[ind] ← HAMIS
```

11.2. példa. Az előző példánkban a 8 kB helyigényű folyamatnak *küszöb* = k darab másikat kell megvárnia, melyek mindegyike 2 másodpercig tart, azaz a 8 kB helyigényű folyamatnak pontosan $2k$ másodpercet kell várnia arra, hogy bekerüljön a 10 kB méretű partícióba.

Eddigi algoritmusainkban a folyamatok abszolút helyigényét vettük a rangsorolás alapjául, azonban ez a módszer nem igazságos: ha egy partícióba két folyamat is beleférne, és egyik sem férne bele kisebb partícióba, nem számít a méretkülönbség, hiszen előbb-utóbb úgyis ugyanabban, vagy egy másik, a vizsgáltnál nem kisebb partícióban kell elhelyezni a kisebbet is. Így az abszolút helyigény helyett annak a legkisebb partíciónak a méretét kellene figyelembe venni rangsoroláskor, amelybe befér az adott folyamat. Sőt, ha a partíciók méretük szerint növekvően rendezve vannak, akkor elég a partíció sorszáma is a rendezett listában. Ezt a sorszámot nevezzük a *folyamat ragjának*. A rangok számítását a következő algoritmus végzi.

RANG-SZÁMÍT(*helyigény*, *méret*, *rang*)

```
 1  rend ← RENDEZ(méret)
 2  for i ← 1 to n
 3      do u ← 1
 4         v ← m
 5         rang[i] ← ⌊(u + v)/2⌋
 6         while rend[rang[i]] < helyigény[i] vagy rend[rang[i] + 1] > helyigény[i]
 7                 do if rend[rang[i]] < helyigény[i]
 8                       then u ← rang[i] + 1
 9                       else  v ← rang[i] − 1
10                 rang[i] ← ⌊(u + v)/2⌋
11  return rang
```

Jól látható, hogy ez az algoritmus előbb rendezi a partíciókat méretük szerint növekvő sorrendbe, majd minden folyamathoz kiszámítja annak rangját. Ezt azonban csak kezdetben kell megtennünk, valamint akkor, ha új folyamat jön. Ez utóbbi esetben azonban már csak az új folyamatokra kell végrehajtanunk a belső ciklust. A rendezést sem kell újra végrehajtanunk, hiszen a partíciók nem változnak a rendszerünk működése során. Az egyetlen, amire szükség van, a folyamat beillesztése a megfelelő két partíció közé, amelyek közül a nagyobba befér, a kisebbe már nem. Ez egy logaritmikus kereséssel megoldható, amiről tudjuk, hogy helyes is. Már csak a rangszámítás lépésszámát kell megmondanunk: az összehasonlításos rendezés $\Theta(m \lg m)$, a logaritmikus keresés pedig $\Theta(\lg m)$, amit n darab folyamatra kell végrehajtanunk. Így az összes lépésszám $\Theta((n + m) \lg m)$.

A rangok kiszámítása után ugyanazt kell tennünk, mint eddig.

RÉGÓTA-VÁRAKOZÓ-VAGY-KISEBBE-NEM-FÉR(*hely*, *helyigény*, *méret*, *part*, *vár*)

```
1  for i ← 1 to m
2      do if part[i] = 0
3            then BETÖLT-RÉGÓTA-VÁRAKOZÓ-VAGY-KISEBBE-NEM(hely, helyigény,
                  méret, p, part, vár)
```

Különbség egyedül a folyamatot egy adott partícióba töltő algoritmusban van: nem a *méret*, hanem a *rang* vektoron kell a feltételes maximumkeresést végrehajtani.

Az algoritmus helyessége az algoritmus előző változatának és a rangszámítás algoritmusának helyességéből következik. Lépésszámának nagyságrendje is az előző változatéval egyezik meg.

11.3. példa. Az előző példát tekintve látszik, hogy a 8 kB helyigényű és a 9 kB helyigényű folyamatok mindegyike csak a 10 kB méretű partícióba fér be, az 5 kB méretűbe nem. Ezért a rangjuk is meg fog egyezni (kettő lesz), így érkezési sorrendben kerülnek elhelyezésre, tehát a 8 kB méretű először vagy másodszor kapja meg az általa igényelt memóriaterületet.

BETÖLT-RÉGÓTA-VÁRAKOZÓ-VAGY-KISEBBE-NEM(*hely*, *helyigény*, *méret*, *p*, *part*, *vár*

```
1  max ← 0
2  ind ← 0
3  for i ← 1 to n
4      do if vár[i] and helyigény[i] ≤ méret[j]
5            then if (pont[i] > küszöb és pont[i] > pont[ind]) vagy rang[i] > max
6                    then pont[ind] ← pont[ind] + 1
7                         ind ← i
8                         max ← rang[i]
9                    else pont[i] ← pont[i] + 1
10 part[j] ← ind
11 hely[ind] ← p
12 vár[ind] ← HAMIS
```

11.1.2. Dinamikus partíciók

A ***dinamikus partícionálás*** egészen máshogy történik, mint a rögzített. Ennél a módszernél nem azt kell eldönteni, hogy egy üres partícióban mely folyamatot helyezzük el, hanem azt, hogy egy folyamatot hol helyezünk el egy adott, több különböző méretű darabból álló memóriaterületen, azaz hol hozunk neki létre dinamikusan egy partíciót. Ebben a részben szintén az operációs rendszerek terminológiáját fogjuk használni, de az algoritmusok természetesen átfogalmazhatók alkalmazások szintjén megadott feladatok megoldására is.

Ha a folyamatok elindulásuk után mind egyszerre érnének véget, nem lenne probléma, hiszen az üres memóriaterületet alulról felfelé folyamatosan tölthetnénk fel. A helyzet azonban szinte mindig bonyolultabb, hiszen a folyamatok egymástól gyakran nagy mértékben különböznek, így a futásidejük sem azonos. Ezáltal viszont az elfoglalt memóriaterület nem feltétlenül lesz folytonos, hanem a foglalt partíciók között szabad partíciók jelennek meg. Mivel a memóriabeli másolás rendkívül költséges művelet, a gyakorlatban nem célravezető a foglalt partíciókat a memória aljára tömöríteni. Gyakran a tömörítés nem is oldható meg a bonyolult relatív címzések miatt. Tehát a szabad terület, amelyen el kell helyezni az új folyamatokat, nem lesz összefüggő. Nyilvánvaló, hogy a folyamatokat egy-egy szabad partíció elején célszerű elhelyezni, az viszont kevésbé, hogy melyik szabad partícióban a sok közül.

Az egyes partíciókat legegyszerűbb egy láncolt listában tárolni. Természetesen sok más, esetleg hatékonyabb tárolási módszer is elképzelhető, az itt felsorolt algoritmusok bemutatásához azonban ez elegendő. A *p* címen található partíciók elejét az *eleje*[*p*], a méretét a *méret*[*p*] tartalmazza, azt pedig, hogy mely folyamat fut az adott partícióban, a *part*[*p*] változóban tároljuk. Ha a folyamat azonosítója 0, akkor üres, különben pedig foglalt partícióról van szó. A láncolt listában következő partíció címe *köv*[*p*].

Ahhoz, hogy dinamikusan létrehozzunk egy megfelelő méretű partíciót, előbb ketté kell vágnunk egy legalább akkora, szabad partíciót. Ezt végzi el a következő algoritmus.

KETTÉVÁG-PARTÍCIÓ(*határ*, *eleje*, *köv*, *méret*, *p*, *q*)

```
1  eleje[q] ← eleje[p] + határ
2  méret[q] ← méret[p] − határ
3  méret[p] ← határ
4  köv[q] ← köv[p]
5  köv[p] ← q
```

A rögzített partíciók módszerénél látott algoritmusokkal szemben, melyek a partíciókhoz választották a folyamatokat, itt fordított szemlélettel dolgozunk. Itt a folyamatok listáján megyünk végig, és minden várakozó folyamathoz keresünk egy olyan szabad partíciót, amelybe befér. Amennyiben befért, a partíció elejéből levágjuk a szükséges darabot, és a folyamathoz rendeljük, a folyamat leírójában pedig jelöljük, hogy mely partícióban fut. Ha nem fért be egyik szabad partícióba sem, akkor az adott körben a folyamatot nem tudtuk elhelyezni.

ELHELYEZ(*vár*)

```
1  for j ← 1 to n
2      do if vár[j] = IGAZ
3           then ∗-FIT(j)
```

A pszeudokódban lévő $*$ a First, Next, Best, Korlátos-best, Worst és Korlátos-worst értékeket veheti fel.

A megfelelő szabad partíció kiválasztására több lehetőség is kínálkozik. Az egyik, hogy a partíciók listáján elindulva az első olyan szabad partícióban helyezzük el a folyamatot, amelyben elfér. Ez lineáris keresés segítségével könnyen megoldható.

FIRST-FIT(*f*, *eleje*, *fej*, *hely*, *helyigény*, *köv*, *méret*, *part*, *vár*)

```
1  p ← fej[P]
2  while vár[f] = IGAZ és p ≠ NIL
3        do if part[p] = 0 és méret[p] ≥ helyigény[f]
4             then KETTÉVÁG-PARTÍCIÓ(helyigény[f], eleje, köv, méret, p, q)
5                  part[p] ← f
6                  hely[f] ← p
7                  vár[f] ← HAMIS
8           p ← köv[p]
```

Az algoritmus helyességéhez itt is több dolgot kell belátnunk. Az első most is az, hogy ne töltsünk egy folyamatot kisebb partícióba, mint amekkorába beférne. Ennek teljesülése következik a lineáris keresés tételéből, mivel annak feltételében ez a kritérium is szerepel.

Ennél a módszernél is fontos, hogy egyetlen folyamat se kerüljön több partícióba, illetve egyetlen partícióba se kerüljön több folyamat. Ezen kritérium teljesülésének bizonyítása szó szerint megegyezik a rögzített partíciónál ismertetettel, azzal a különbséggel, hogy itt

nem a feltételes maximumkeresés, hanem a lineáris keresés tételének helyességére vezetünk vissza.

Természetesen most sem elegendőek ezek a feltételek, hiszen az üres algoritmus is teljesíti mindhármat. Azt fogjuk még bizonyítani, hogy az algoritmus minden olyan folyamatot elhelyez a memóriában, amelyhez tartozik olyan partíció, amelybe befér. Ehhez ismét szükségünk lesz egy invariánsra: a j darab folyamat megvizsgálása után nem létezik olyan pozitív $k \leq j$, amelyre *vár*$[k]$, és amelyre van olyan p partíció, hogy *part*$[p] = 0$ és *méret*$[p] \geq$ *helyigény*$[k]$.

Teljesül: Az algoritmus elején $j = 0$ darab folyamatot vizsgáltunk meg, így egyáltalán nem létezik pozitív $k \leq j$.

Megmarad: Ha az invariáns teljesül j-re a ciklusmag elején, akkor először is azt kell látnunk, hogy ugyanerre a j-re a ciklus végén is teljesülni fog. Ez nyilvánvaló, hiszen az első j darab folyamathoz nem nyúlunk a $(j + 1)$-edik vizsgálata során, az őket tartalmazó partíciókra pedig *part*$[p] > 0$, ami nem felel meg a First-fit algoritmusban található lineáris keresés feltételének. A $(j + 1)$-edik folyamatra pedig azért fog teljesülni az invariáns a ciklusmag végén, mert ha van a feltételnek megfelelő szabad blokk, azt a lineáris keresés biztosan megtalálja, hiszen a lineáris keresésünk feltétele megegyezik az invariánsunk egyes partíciókon értelmezett követelményével.

Befejeződik: Mivel a ciklus egyesével halad végig egy rögzített intervallumon, biztosan be is fog fejeződni. Mivel a ciklusmag pontosan annyiszor hajtódik végre, mint ahány folyamat van, a ciklus befejeződésekor igaz lesz, hogy nem létezik olyan pozitív $k \leq n$, amelyre *vár*$[k]$ és amelyre van olyan p partíció, hogy *part*$[p] = 0$ és *méret*$[p] \geq$ *helyigény*$[i]$, azaz nem hagytunk várakozni egyetlen olyan folyamatot sem, amelyhez lenne partíció, amibe belefér.

Az algoritmus lépésszáma most is könnyen kiszámítható. Mindenképp végignézzük az n darab folyamatot. Ha például minden folyamat vár és a partíciók foglaltak, akkor az algoritmus lépésszáma $\Theta(nm)$.

A lépésszám kiszámításánál azonban nem vettünk figyelembe néhány fontos szempontot. Az egyik az, hogy m nem állandó, hanem az algoritmus többszöri lefuttatása után várhatóan nő, mivel a folyamatok egymástól függetlenek, különböző időpillanatokban indulnak, illetve érnek véget, és méretük is jelentősen eltérhet egymásétól. Ezért gyakrabban vágunk ketté egy partíciót, minthogy kettőt egyesíthetnénk. Ezt a jelenséget a ***memória elaprózódásának*** nevezzük. Tehát a legrosszabb eset lépésszáma az algoritmus többszöri futtatása során folyamatosan nő. Ráadásul a lineáris keresés mindig a legelső megfelelő méretű partíciót vágja ketté, így egy idő után sok kis partíció lesz a memóriaterület elején, amelyekbe nem tölthetjük be a folyamatok többségét. Így az átlagos lépésszám is növekedni fog. Ez utóbbi problémára megoldást jelenthet, ha a keresést nem mindig a partíciók listájának elejéről kezdjük, hanem annak a partíciónak a másik felétől, amelynek első felébe a legutóbbi folyamatot töltöttük, és ha a lista végére érünk, az elejéről folytatjuk mindaddig, amíg a folyamat be nem fért valamelyik partícióba, vagy újra el nem értük a kiindulási elemet, azaz ciklikusan járjuk be a partíciók listáját.

NEXT-FIT($f, fej, helyigény, köv, part, vár$)

```
 1  if utolsó[P] ≠ NIL
 2     then p ← köv[utolsó[P]]
 3     else p ← fej[P]
 4  while vár[f] = IGAZ és p ≠ utolsó[P]
 5        do if p = NIL
 6              then p ← fej[P]
 7           if part[p] = 0 és méret[p] ≥ helyigény[f]
 8              then KETTÉVÁG-PARTÍCIÓ(helyigény[f], eleje, köv, méret, p, q)
 9                   part[p] ← f
10                   hely[f] ← p
11                   vár[f] ← HAMIS
12                   utolsó[P] ← p
13           p ← köv[p]
```

A NEXT-FIT algoritmus helyességének bizonyítása lényegében megegyezik a FIRST-FIT-évél, és lépésszámának nagyságrendje is azonos. Gyakorlatilag itt is lineáris keresésről van szó a belső ciklusban, csupán az intervallumot forgatjuk el mindig a végén. Ez az algoritmus azonban egyenletesen járja be a szabad területek listáját, így nem aprózza el az elejét. Ennek következtében az átlagos lépésszám is várhatóan kisebb lesz, mint a FIRST-FIT algoritmusé.

Ha csak annyit vizsgálunk minden partícióról, hogy elfér-e benne a folyamat, akkor előfordulhat, hogy kis folyamatok számára vágunk el nagy partíciókat, így a később érkező nagyobb folyamatoknak már nem jut megfelelő méretű partíció. A nagy partíciók pazarlását akkor el tudjuk kerülni, ha minden folyamatot a legkisebb olyan partícióban helyezünk el, amelyben elfér.

BEST-FIT($f, fej, helyigény, köv, part, vár$)

```
 1  min ← ∞
 2  ind ← NIL
 3  p ← fej[P]
 4  while p ≠ NIL
 5        do if part[p] = 0 és helyigény[f] ≤ méret[p] < min
 6              then ind ← p
 7                   min ← méret[p]
 8           p ← köv[p]
 9  if ind ≠ NIL
10     then KETTÉVÁG-PARTÍCIÓ(helyigény[f], eleje, köv, méret, ind, q)
11          part[ind] ← f
12          hely[f] ← ind
13          vár[f] ← HAMIS
```

Az algoritmus helyességének összes kritériuma belátható jelen esetben is, ugyanúgy, mint az előzőekben. Az egyetlen különbség a FIRST-FIT-hez képest, hogy most lineáris keresés helyett feltételes minimumkeresést alkalmazunk. Nyilvánvaló az is, hogy ez az algo-

ritmus egyetlen folyamat számára sem fog nagyobb partíciót kettévágni, mint amekkorát a meglévők közül minimálisan szükséges .

Nem mindig jó azonban, ha minden folyamatot a legkisebb szabad helyre teszünk be, ahová befér. Ekkor ugyanis a partíció üresen maradó része gyakran annyira kicsi, hogy az már csak nagyon kevés folyamat számára használható. Ez két okból is hátrányos. Egyrészt ezeket a partíciókat minden folyamat elhelyezésekor újra és újra megvizsgáljuk, hiszen benne vannak a szabad partíciók listájában. Másrészt pedig, hogy sok kis partíció együtt nagy területet foglalhat el, amit azonban nem tudunk hasznosítani, mert nem összefüggő. Tehát valamilyen módon ki kell védenünk azt, hogy az üresen maradó partíciók túlságosan kicsik legyenek. A túlságosan kicsi fogalom jelenthet egy konstanst is, de lehet az elhelyezendő folyamat helyigényének függvénye is. (Például legalább még egyszer akkora legyen a szabad terület, mint az elhelyezendő folyamat.) Mivel a korlátot az egész partícióra, és nem a megmaradó részre vizsgáljuk, ezért ezt mindig egy a folyamattól függő függvényként kezeljük. Természetesen, ha a nincs olyan partíció, amely ennek a kiegészítő kritériumnak is megfelel, akkor helyezzük el a folyamatot a legnagyobb partícióban. Így a következő algoritmust kapjuk.

KORLÁTOS-BEST-FIT(f)

```
 1  min ← ∞
 2  ind ← NIL
 3  p ← fej[P]
 4  while p ≠ NIL
 5        do if part[p] = 0 és méret[p] ≥ helyigény[f] és
                (méret[p] < min és méret[p] ≥ KORLÁT(helyigény[f])) vagy
                ind = NIL vagy (min < KORLÁT(helyigény[f]) és méret[p] > min))
 6                    then ind ← p
 7                         min ← méret[p]
 8              p ← köv[p]
 9  if ind ≠ NIL
10    then KETTÉVÁG-PARTÍCIÓ(helyigény[f], eleje, köv, méret, ind, q)
11         part[ind] ← f
12         hely[f] ← ind
13         vár[f] ← HAMIS
```

Ez az algoritmus bonyolultabb, mint az előzőek. Helyességének bizonyításához azt kell észrevennünk, hogy a belső ciklus egy módosított feltételes minimumkeresés. A feltétel első része, azaz hogy $part[p] = 0$ és $méret[p] \geq helyigény[f]$, továbbra is azt mondja, hogy olyan partíciót keresünk, amely szabad, és belefér a folyamat. A második rész egy diszjunkció, azaz három esetben cseréljük ki a vizsgált elemet az eddig megtalálttal. Az egyik eset, amikor $méret[p] < min$ és $méret[p] \geq$ KORLÁT($helyigény[f]$), vagyis a vizsgált partíció mérete legalább akkora, mint az előírt minimális, de kisebb, mint az eddig talált legkisebb. Ha nem lenne több feltétel, akkor ez a feltételes minimumkeresés lenne, ahol a feltételek közé bevettük azt is, hogy a partíció mérete egy bizonyos korlát fölött legyen. Azonban van még két lehetőség is, amikor kicseréljük az eddig megtalált elemet az éppen vizsgálttal. Ezek közül az első, amikor $ind =$ NIL, azaz az éppen vizsgált partíció az első olyan, amely szabad, és

elfér benne a folyamat. Erre azért van szükség, mert továbbra is megköveteljük, hogy ha van olyan szabad partíció, amelyben elfér a folyamat, akkor az algoritmus helyezze is azt el ezek valamelyikében. Végül a harmadik feltétel alapján akkor cseréljük ki az eddig megvizsgáltak közül legmegfelelőbbnek talált elemet az aktuálissal, ha $min <$ KORLÁT($helyigény[f]$) és $méret[p] > min$, vagyis az eddigi minimum nem érte el az előírt korlátot, és az aktuális nagyobb nála. Ez a feltétel kettős célt szolgál. Egyrészt, ha eddig olyan elemet találtunk csak, amely nem felel meg a kiegészítő feltételnek, az aktuális pedig igen, akkor kicseréljük vele, hiszen ekkor $min <$ KORLÁT($helyigény[f]$) $\leq méret[p]$, tehát az aktuális partíció mérete nyilvánvalóan nagyobb. Másrészt, ha sem az eddig megtalált, sem az aktuális partíció mérete nem éri el az előírt korlátot, de az aktuálisan vizsgált jobban közelíti azt alulról, akkor $min < méret[p] <$ KORLÁT($helyigény[f]$) teljesül, tehát ebben az esetben is kicseréljük az eddig megtaláltat az aktuálisra. Így, ha van olyan partíció, amely legalább akkora, mint az előírt korlát, akkor az algoritmus minden folyamatot ezek közül a legkisebbe fog helyezni, míg ha nincsenek ilyenek, akkor a legnagyobba, amelybe befér.

Bizonyos feladatoknál előfordulhat, hogy kizárólag az a cél, hogy a megmaradó területek mérete minél nagyobb legyen. Ezt úgy érhetjük el, hogy minden folyamatot a legnagyobb szabad partícióban helyezünk el:

WORST-FIT(f,fej,$helyigény$,$köv$,$part$,$vár$)

```
1  max ← 0
2  ind ← NIL
3  p ← fej[P]
4  while p ≠ NIL
5        do if part[p] = 0 és méret[p] ≥ helyigény[f] és méret[p] > max
6              then ind ← p
7                   max ← méret[p]
8           p ← köv[p]
9  if ind ≠ NIL
10   then KETTÉVÁG-PARTÍCIÓ(helyigény[f], eleje, köv, méret, ind, q)
11        part[ind] ← f
12        hely[f] ← ind
13        vár[f] ← HAMIS
```

Az algoritmus helyességének bizonyítása hasonló a BEST-FIT algoritmuséhoz, a különbség csupán annyi, hogy feltételes minimumkeresés helyett maximumkeresést használunk. Az is nyilvánvaló ebből, hogy a fennmaradó helyek mérete maximális lesz.

A WORST-FIT algoritmus garantálja, hogy a legkisebb üresen maradó partíció mérete a lehető legnagyobb lesz, azaz kevés lesz az olyan partíció, amely a legtöbb folyamat számára már túl kicsi. Ezt azáltal éri el, hogy mindig a legnagyobb partícióból vág le. Ennek az a következménye, hogy a nagy helyigényű folyamatok számára sokszor már nem is jut megfelelő méretű partíció, hanem azok várakozásra kényszerülnek a háttértárban. Hogy ez ne így történjen, a BEST-FIT-hez hasonlóan itt is megfogalmazhatunk egy kiegészítő feltételt. Itt azonban nem alsó, hanem felső korlátot adunk. Az algoritmus igyekszik olyan partíciót kettévágni, amelynek mérete egy bizonyos korlát alatt van. A korlát itt is a folyamat helyigényének függvénye. (Például annak kétszerese.) Ha talál ilyen partíciókat, akkor ezek

közül a legnagyobbat választja, hogy elkerülje a túl kis partíciók létrejöttét. Ha csak olyanokat talál, amelyek nagyobbak ennél a korlátnál, akkor viszont a minimálisat vágja ketté közülük, nehogy elvegye a helyet a nagy folyamatok elől.

KORLÁTOS-WORST-FIT(f,*fej*,*helyigény*,*köv*,*part*,*vár*)

```
 1  max ← 0
 2  ind ← NIL
 3  p ← fej[P]
 4  while p ≠ NIL
 5        do if part[p] = 0 és méret[p] ≥ helyigény[f] és
                 (méret[p] > max és méret[p] ≤ KORLÁT(helyigény[f]) vagy
                 ind = NIL vagy (max > KORLÁT(helyigény[f]) és méret[p] < max))
 6              then ind ← p
 7                   max ← méret[p]
 8           p ← köv[p]
 9  if ind ≠ NIL
10    then KETTÉVÁG-PARTÍCIÓ(helyigény[f], eleje, köv, méret, ind, q)
11         part[ind] ← f
12         hely[f] ← ind
13         vár[f] ← HAMIS
```

Látható, hogy az algoritmus nagyon hasonlít a KORLÁTOS-BEST-FIT-hez, csupán a relációs jelek mutatnak az ellenkező irányba. A különbség valóban nem nagy. Mindkét algoritmusban ugyanazt a két feltételt próbáljuk teljesíteni: ne keletkezzenek túl kis üres partíciók, és ne pazaroljuk el a nagy szabad partíciókat kis folyamatokra. Az egyetlen különbség, hogy e két feltétel közül melyiket vesszük figyelembe elsődlegesen, és melyiket másodlagosan. Ezt mindig az adott feladat határozza meg.

Gyakorlatok

11.1-1. Adott egy rögzített partíciókat használó rendszer két darab 100 kB, egy 200 kB és egy 400 kB méretű partícióval. Kezdetben mindegyik üres, majd egy másodperc múlva érkezik egy 80 kB, egy 70 kB, egy 50 kB, egy 120 kB és egy 180 kB helyigényű folyamat, amelyek adatai ebben a sorrendben kerülnek tárolásra a megfelelő vektorokban. A 180 kB méretű a beérkezésétől számított ötödik másodpercben véget ér, ám ekkorra már egy 280 kB helyigényű folyamat is érkezett a memóriába. Mely partíciókban mely folyamatok lesznek a kezdeti folyamatok beérkezésétől számított hatodik másodpercben, ha feltételezzük, hogy más folyamatok nem érnek véget addig, és a LEGNAGYOBB-BEFÉRŐ algoritmust használjuk? Mi a helyzet, ha a LEGNAGYOBB-VAGY-RÉGÓTA-VÁRAKOZÓ-BEFÉRŐ, illetve ha a RÉGÓTA-VÁRAKOZÓ-VAGY-KISEBBE-NEM-FÉR algoritmust használjuk 4 küszöbértékkel?

11.1-2. Egy dinamikus partíciókat használó rendszerben a következő szabad partíciók találhatók meg a partíciók listájában: egy 20 kB méretű, amit egy 100 kB, egy 210 kB, egy 180 kB, egy 50 kB, egy 10 kB, egy 70 kB, egy 130 kB és egy 90 kB méretű követ, pontosan ebben a sorrendben. Legutoljára a 180 kB méretű szabad partíció elé helyeztünk el folyamatot. A rendszerbe érkezik egy 40 kB helyigényű folyamat. Melyik szabad partícióban fogja ezt elhelyezni a FIRST-FIT, a NEXT-FIT, a BEST-FIT, a KORLÁTOS-BEST-FIT, a WORST-FIT, illetve a

KORLÁTOS-WORST-FIT algoritmus?

11.1-3. A WORST-FIT algoritmus egy hatékony megvalósítása, ha a partíciókat nem lineárisan láncolt listában, hanem bináris kupacban tároljuk. Mekkora lesz így az ELHELYEZ algoritmus műveletigénye?

11.2. Lapcserélési algoritmusok

Mint már említettük, a mai számítógépek memóriája több szintből áll. A szinteket egyszintes memóriának látszó ***virtuális memóriává*** szervezik. A felhasználóknak nincs szüksége arra, hogy ezt a többszintes szerkezetet részletesen ismerjék: az operációs rendszerek egységesnek látszó ***virtuális memóriává*** szervezik a szinteket.

Ennek a virtuális memóriának a kezelésére a két legelterjedtebb módszer a lapozás és a szegmentálás: előbbi egységes méretű részekre, úgynevezett ***lapkeretekre*** osztja mindkét memóriaszintet (és ennek megfelelően a programokat is), míg a szegmentálásnál a program ***szegmenseknek*** nevezett, változó méretű részeit mozgatjuk a memóriaszintek között.

Először az egyszerű tárgyalás érdekében tegyük fel, hogy a vizsgált számítógép memóriája két szintből áll: a kisebb és gyorsabb elérésű rész a ***fizikai memória*** (röviden memória), a nagyobb méretű és nagyobb elérési idejű rész pedig a ***háttérmemória.***

Kezdetben a fizikai memória üres, a háttérmemóriában pedig egyetlen program van, amely n részből áll. Feltesszük, hogy a program futása során utasításokat kell végrehajtani, és minden utasítás végrehajtásához egy-egy programrészre van szükségünk.

A hivatkozási sorozat feldolgozása során a következő részfeladatokat kell megoldani.

1. Hol helyezzük el a fizikai memóriában (ha nincs ott) a következő utasítás végrehajtásához szükséges programrészt?
2. Mikor helyezzünk el programrészeket a fizikai memóriában?
3. Hogyan szabadítsunk fel helyet a fizikai memóriában az elhelyezendő programrészek számára?

Az első kérdésre az *elhelyezési algoritmusok* válaszolnak: a lapozásnál egyszerűen azt, hogy akárhol – ugyanis a fizikai memória lapkeretei azonos méretűek és hozzáférési idejűek. A szegmentálás során a a fizikai memóriában programszegmensek és ***lyukaknak*** nevezett üres memóriarészek váltakoznak – és az első kérdésre a szegmenselhelyezési algoritmusok válaszolnak.

A második kérdésre az *átviteli algoritmusok* válaszolnak: a működő rendszerek nagy többségében azt, hogy ***igény szerint,*** azaz akkor kezdődik meg a programrész beolvasása a háttértárból, amikor kiderül, hogy az adott programrészre szükség van. A másik lehetőség az ***előbetöltés*** lenne, a tapasztalatok szerint azonban ez sok felesleges munkával jár, ezért nem terjedt el.

A harmadik kérdésre a ***cserélési algoritmusok*** válaszolnak: lapozásnál a lapcserélési algoritmusok, amelyeket ebben az alfejezetben mutatunk be. A szegmentálásnál alkalmazott szegmenscserélési algoritmusok lényegében a lapcserélési algoritmusok ötleteit hasznosítják – azokat a szegmensek különböző méretének megfelelően kiegészítve.

A lapozott számítógépekben mindkét szintet azonos méretű részekre – úgynevezett ***lapkeretekre*** osztjuk. A fizikai memória mérete m lapkeret, a háttérmemória mérete pedig n

lapkeret. A paraméterek között természetes az $1 \leq m \leq n$ egyenlőtlenség. A gyakorlatban n rendszerint több nagyságrenddel nagyobb, mint m. Kezdetben a fizikai memória üres, a háttérmemóriában pedig egyetlen program van. Feltesszük, hogy a program futása során p utasítást kell végrehajtani, és a t-edik utasítás végrehajtásához az r_t lapkeretben lévő lapra van szükségünk, azaz a program futását az $R = \langle r_1, r_2, \ldots, r_p \rangle$ ***hivatkozási tömbbel*** modellezzük.

A továbbiakban csak az igény szerinti lapozással, azon belül is csak a lapcserélési algoritmusokkal foglalkozunk.

Ebben az egyszerű modellben azt tételezzük fel, hogy az utasítás végrehajtásához az r_t programrészt beolvassuk, és az utasítás végrehajtásának eredményét is az r_t programrészbe írjuk. Ahol szükség van arra, hogy az olvasást és írást megkülönböztessük, ott az R tömb mellett egy $W = \langle w_1, w_2, \ldots, w_p \rangle$ ***írási tömböt*** is megadunk, melynek w_t eleme IGAZ, ha az r_t lapra írunk, egyébként $w_t =$ HAMIS.

Az igény szerinti lapcserélési algoritmusokat szokás ***statikus*** és ***dinamikus*** algoritmusokra osztani. A program futásának elején mindkét típus teletölti a fizikai memória lapkereteit lapokkal, a statikus algoritmusok azonban ezután a futás végéig *pontosan m* lapkeretet tartanak lekötve, míg a dinamikus algoritmusok *legfeljebb m* lapkeretet foglalnak le.

11.2.1. Statikus lapcserélés

A statikus lapcserélési algoritmusok bemenő adatai a fizikai memória mérete lapkeretben (m), a program mérete lapban (n), a program futási ideje utasításban (p) és a hivatkozási sorozat (R), kimenő adata pedig a laphibák száma (*laphiba*).

A statikus algoritmusok működése a laptábla kezelésén alapul. A laptábla egy $n \times 2$ méretű tömb, melynek i-edik sora ($i \in [0..n-1]$) az i-edik lapra vonatkozik. A sor első eleme egy logikai változó (jelzőbit), melynek értéke azt jelzi, hogy a lap az adott időpontban a fizikai memóriában van-e: ha az i-edik lap a fizikai memóriában van, akkor $laptábla[i,1] =$ IGAZ és $laptábla[i,2] = j$, ahol a $j \in [0..m-1]$ azt adja meg, hogy a lap a fizikai memória j-edik lapkeretében van. Ha az i-edik lap nincs benn a fizikai memóriában, akkor $laptábla[i,1] =$ HAMIS és $laptábla[i,2]$ értéke definiálatlan. A *foglalt* munkaváltozó a fizikai memória foglalt lapkereteinek számát tartalmazza.

Ha a lapok mérete z, akkor a v virtuális címből úgy számítjuk ki az f fizikai címet, hogy $j = \lfloor v/z \rfloor$ megadja a ***virtuális lap indexét,*** $v - z\lfloor v/z \rfloor$ pedig megadja a v virtuális címhez tartozó s ***eltolást.*** Ha az adott időpontban a j-edik lap a fizikai memóriában van – amit $laptábla[j,1] =$ IGAZ jelez –, akkor $f = s + z \cdot laptábla[j,2]$. Ha viszont a j-edik lap nincs a fizikai memóriában, ***laphiba*** lép fel. Ekkor a lapcserélési algoritmus segítségével kiválasztjuk a fizikai memória egyik lapkeretét, abba betöltjük a j-edik lapot, frissítjük a *laptábla* j-edik sorát és azután számítjuk ki f-et.

Az igény szerinti statikus lapcserélési algoritmusok működése leírható kezdőállapottal rendelkező Mealy-automatával. Ezek az automaták $(Q, q_0, X, Y, \delta, \lambda)$ alakban adhatók meg, ahol Q a ***vezérlő állapotok halmaza,*** $q_0 \in Q$ a ***kezdeti vezérlő állapot,*** X a ***bemenő jelek halmaza,*** Y a ***kimenő jelek halmaza,*** $\delta : Q \times X \to Q$ az ***állapot-átmenetfüggvény*** és $\lambda : Q \times X \to Y$ a ***kimenetfüggvény.***

Itt nem foglalkozunk az automaták leállásának formalizálásával.

A bemenő jelek $R_p = \langle r_1, r_2, \ldots, r_p \rangle$ (vagy $R_\infty = \langle r_1, r_2, \ldots \rangle$) sorozatát ***hivatkozási sorozatnak*** hívjuk.

Egyszerűsíti az algoritmusok definiálását az S_t ($t = 1, 2, \ldots$) memóriaállapotok bevezetése: ez az állapot a t-edik bemenő jel feldolgozása után az automata memóriájában (a fizikai memóriában) tárolt lapok halmaza. Az igény szerinti statikus lapcserélési algoritmusok esetén $S_0 = \emptyset$. Ha az új memóriaállapot a régitől különbözik (azaz lapbevitelre volt szükség), akkor ***laphiba*** történt. Eszerint mind a lapnak üres lapkeretbe való bevitelét, mind pedig a lapcserét laphibának nevezzük.

A lapcserélési algoritmusok esetén – Denning javaslatára – λ és δ helyett inkább a $g : Q \times M \times X \to Q \times Y$ ***átmenetfüggvényt*** használjuk.

Mivel a lapcserélési algoritmusokra $X = \{0, 1, \ldots, n-1\}$ és $Y = X \cup \emptyset$, ez a két elem a definícióból elhagyható és így a P lapcserélési algoritmus a (Q, q_0, g_P) hármassal adható meg.

Első példánk az egyik legegyszerűbb lapcserélési algoritmus, a FIFO (**F**irst **I**n **F**irst **O**ut), amely a lapokat a betöltés sorrendjében cseréli.

Definíciója a következő: $q_0 = \langle\rangle$ és

$$g_{\text{FIFO}}(S, q, x) = \begin{cases} (S, q, \epsilon), & \text{ha } x \in S\ , \\ (S \cup \{x\}, q', \epsilon), & \text{ha } x \notin S,\ |S| = k < m\ , \\ (S \setminus \{y_1\} \cup \{x\}, q'', y_1), & \text{ha } x \notin S \text{ és } |S| = k = m\ , \end{cases} \tag{11.1}$$

ahol $q = \langle y_1, y_2, \ldots, y_k\rangle$, $q' = \langle y_1, y_2, \ldots, y_k, x\rangle$ és $q'' = \langle y_2, y_3, \ldots, y_m, x\rangle$.

A programok futtatását a következő *-FUTTAT algoritmus végzi. Ebben az alfejezetben az algoritmusok nevében a $*$ helyére mindig az alkalmazandó lapcserélési algoritmus neve kerül (FIFO, LRU OPT, LFU vagy NRU). A pszeudokódokban feltételezzük, hogy a meghívott eljárások ismerik a hívó eljárásban használt változók értékét, és a hívó eljárás hozzáfér az új értékekhez.

```
*-FUTTAT(m, n, p, R, hibaszám, laptábla)
1  hibaszám ← 0
2  foglalt ← 0
3  for i ← 0 to n − 1                          ▷ A laptábla előkészítése.
4      do laptábla[i, 1] ← HAMIS
5  *-ELŐKÉSZÍT(laptábla)
6  for i ← 1 to p                              ▷ A program futtatása.
7      do *-VÉGREHAJT(laptábla, t)
8  return hibaszám
```

Az algoritmus következő megvalósítása egy Q sorban tartja nyilván a lapok betöltési sorrendjét. Az előkészítő algoritmus feladata az üres sor létrehozása, azaz a $Q \leftarrow \emptyset$ utasítás végrehajtása.

A következő pszeudokódban *kidob* a cserélendő lap sorszáma, *behoz* a fizikai memória azon lapjának sorszáma, melybe az új lapot behozzuk.

FIFO-VÉGREHAJT(*laptáblat*),

```
1  if laptábla[r_t, 1] = IGAZ                          ▷ A következő lap benn van.
2     then NIL
3  if laptábla[r_t, 1] = HAMIS                         ▷ A következő lap nincs benn.
4     then laphiba ← laphiba + 1
5          if foglalt < m                              ▷ A fizikai memória nincs tele.
6            then SORBA(Q, r_t)
7                 behoz ← foglalt
8                 foglalt ← foglalt + 1
9          if foglalt = m                              ▷ A fizikai memória tele van.
10           then kidob ← SORBÓL(Q)
11                laptábla[kidob, 1] ← HAMIS
12                behoz ← laptábla[kidob, 2]
13         KIÍR(behoz, kidob)
14    BEOLVAS(r_t, behoz)                              ▷ Beolvasás.
15    laptábla[r_t, 1] ← IGAZ                          ▷ Adatok frissítése.
16    laptábla[r_t, 2] ← behoz
```

A KIÍR eljárás feladata, hogy a cserére kiválasztott lapot kiírja a háttértárba: első paramétere a honnan (a memória melyik lapkeretéből), második paramétere a hová (a háttértár melyik lapkeretébe) kérdésre ad választ. A BEOLVAS eljárás feladata az, hogy a következő utasítás végrehajtásához szükséges lapot a háttértárból a fizikai memória megfelelő lapkeretébe beolvassa: első paramétere a honnan (a háttértár melyik lapkeretéből), második paramétere a hová (a memória melyik lapkeretébe) A két eljárás paramétereinek megadásánál kihasználjuk, hogy a lapkeretek mérete azonos, ezért a j-edik lapkeret kezdőcíme mindkét memóriában a z lapméret j-szerese.

A lapcserélési algoritmusok többségének az r_t hivatkozás feldolgozásához nincs szüksége az R sorozat többi elemének ismeretére, ezért a helyigény elemzésekor a sorozat helyigényével nem kell számolnunk. Kivételt képez például az OPT algoritmus.

A FIFO-FUTTAT algoritmus helyigényét a laptábla mérete határozza meg – ez a helyigény $\Theta(m)$. A FIFO-FUTTAT algoritmus futási idejét a ciklusa határozza meg. Mivel a 6–7. sorokban meghívott eljárás csak konstans számú lépést végez (feltéve, hogy a sorkezelő műveleteket $O(1)$ idő alatt elvégezzük), ezért FIFO-FUTTAT futási ideje $\Theta(p)$.

Érdemes megjegyezni, hogy a lapok egy része a memóriában tartózkodás alatt nem változik meg, ezért ha a memóriában lévő lapokhoz használtsági bitet rendelünk, akkor az esetek egy részében a 12. sorban lévő kiírás megtakarítható.

A következő példánk az egyik legnépszerűbb lapcserélési algoritmus, az LRU (**L**east **R**ecently **U**sed), amely a legrégebben használt lapot cseréli.

Ennek definíciója a következő: $q_0 = ()$ és

$$g_{\mathrm{LRU}}(S,q,x) = \begin{cases} (S,q,\epsilon), & \text{ha } x \in S\ , \\ (S \cup \{x\}, q', \epsilon), & \text{ha } x \notin S,\ |S| = k < m\ , \\ (S \setminus \{y_1\} \cup \{x\}, q", y_1), & \text{ha } x \notin S \text{ és } |S| = k = m\ , \end{cases} \tag{11.2}$$

ahol $q = \langle y_1, y_2, \ldots, y_k\rangle$, $q' = \langle y_1, y_2, \ldots, y_k, x\rangle$ és $q'' = \langle y_2, y_3, \ldots, y_m, x\rangle$.

Az LRU következő megvalósítása nem igényel előkészítést. Az *utolsó-hiv*$[0\,..\,n-1]$ tömbben tartjuk nyilván az egyes lapok utolsó használatának időpontját, és amikor cserélni kell, lineáris kereséssel határozzuk meg a legrégebben használt lapot.

LRU-VÉGREHAJT(*laptábla*, *t*)

```
 1  if laptábla[r_t, 1] = IGAZ                          ▷ A következő lap benn van.
 2    then utolsó-hiv[r_t] ← t
 3  if laptábla[r_t, 1] = HAMIS                         ▷ A következő lap nincs benn.
 4    then laphiba ← laphiba + 1
 5         if foglalt < m                               ▷ A fizikai memória nincs tele.
 6           then behoz ← foglalt
 7                foglalt ← foglalt + 1
 8         if foglalt = m                               ▷ A fizikai memória tele van.
 9           then kidob ← r_{t−1}
10                for i ← 0 to n − 1
11                    do if laptábla[i, 1] = IGAZ és utolsó-hiv[i] < utolsó-hiv[kidob]
12                          then kidob ← utolsó-hiv[i]
13                laptábla[kidob, 2] ← HAMIS
14                behoz ← laptábla[kidob, 2]
15                KIÍR(behoz, kidob)
16         BEOLVAS(r_t, behoz)                          ▷ Beolvasás.
17         laptábla[r_t, 1] ← IGAZ                      ▷ Adatok frissítése.
18         laptábla[r_t, 2] ← behoz
19         utolsó-hiv[r_t] ← t
```

Ha most n és p értékét is változónak tekintjük, akkor az LRU-VÉGREHAJT 10–11. sorában szereplő lineáris keresés miatt az LRU-FUTTAT algoritmus futási ideje $\Theta(np)$.

A következő algoritmus optimális abban az értelemben, hogy az adott feltételek (azaz rögzített m és n) mellett minimális számú laphibát okoz. Ez az algoritmus a bennlévő lapok közül azt a lapot választja a cseréhez, amelyikre a legkésőbb lesz újra szükség (ha több olyan lap is van, amelyre többet nincs szükség, akkor közülük a legkisebb memóriacímen lévő lapot választjuk).

Előkészítésre ennek az algoritmusnak sincs szüksége.

OPT-VÉGREHAJT(*t*, *laptábla*, *R*)

```
 1  if laptábla[r_t, 1] = IGAZ                          ▷ A következő lap benn van.
 2    then NIL
 3  if laptábla[r_t, 1] = HAMIS                         ▷ A következő lap nincs benn.
 4    then laphiba ← laphiba + 1
```

```
5       if foglalt < m                          ▷ A fizikai memória nincs tele.
6         then behoz ← foglalt
7              foglalt ← foglalt + 1
8       if foglalt = m                          ▷ A fizikai memória tele van.
9         then OPT-KIDOB(t, R)
10             laptábla[kidob, 2] ← HAMIS
11             behoz ← laptábla[kidob, 2]
12             KIÍR(behoz, kidob)
13      BEOLVAS(r_t, behoz)                     ▷ Beolvasás.
14      laptábla[r_t, 1] ← IGAZ                 ▷ Adatok frissítése.
15      laptábla[r_t, 2] ← behoz
```

A 9. sorban hívott OPT-KIDOB eljárás feladata, hogy meghatározza a cserélendő lap indexét.

```
OPT-KIDOB(t, R)
1  védve ← 0                                    ▷ Előkészítés.
2  for j ← 0 to m − 1
3      do keret[j] ← HAMIS
4  s ← t + 1                                    ▷ A lapkeretek védettségének meghatározása.
5  while s ≤ p és laptábla[r_s, 1] = HAMIS és keret[laptábla[r_s, 2]] = HAMIS és
           védve < m − 1
6        do védve ← védve + 1
7           keret[r_s] ← IGAZ
8           s ← s + 1
9  kidob ← m − 1                                ▷ A cserélendő lapot tartalmazó keret meghatározása.
10 j ← 0
11 while keret[j] = IGAZ
12       do j ← j + 1
13 kidob ← j
14 return kidob
```

A $keret[0\,..\,m-1]$ tömbben tároljuk a fizikai memóriában lévő lapokra vonatkozó adatokat: $keret[j] =$ IGAZ azt jelzi, hogy a j-edik keretben tárolt lap a közeli felhasználás miatt védve van a cserétől. A *védve* változóban pedig azt tartjuk számon, hány védett lapról tudunk. Ha már találtunk $m-1$ védett lapot, vagy R végére értünk, akkor a fizikai memória legkisebb címén lévő védetlen lapot választjuk cserélendő lapnak.

Mivel az OPT algoritmusnak szüksége van a teljes R tömbre, ezért tárigénye $\Theta(p)$. Mivel az OPT-KIDOB algoritmus 5–8. sorában legfeljebb az R tömb hátralévő részét kell átnézni, ezért az OPT-FUTTAT algoritmus futási ideje $O(p^2)$.

A következő LFU (**L**east **F**requently **U**sed) algoritmus a legritkábban használt lapot cseréli. A lapcsere egyértelműségének érdekében feltesszük, hogy azonos gyakoriság esetén a fizikai memória legkisebb címén tárolt lapot cseréljük.

A *gyakoriság*$[0\,..\,n-0]$ tömb segítségével tartjuk nyilván, hogy a fizikai memóriába való betöltésük óta hányszor hivatkoztunk az egyes lapokra. Előkészítésre ennek az algoritmusnak sincs szüksége,

LFU-VÉGREHAJT($laptábla, t$)

```
 1  if laptábla[r_t, 1] = IGAZ                          ▷ A következő lap benn van.
 2    then gyakoriság[r_t] ← gyakoriság[r_t] + 1
 3  if laptábla[r_t, 1] = HAMIS                         ▷ A következő lap nincs benn.
 4    then laphiba ← laphiba + 1
 5         if foglalt < m                               ▷ A fizikai memória nincs tele.
 6           then behoz ← foglalt
 7                foglalt ← foglalt + 1
 8         if foglalt = m                               ▷ A fizikai memória tele van.
 9           then kidob ← r_{t-1}
10                for i ← n − 1 downto 0
11                    do if laptábla[i, 1] = IGAZ és gyakoriság[i] ≤ gyakoriság[kidob]
12                          then kidob ← utolsó-hiv[i]
13                laptábla[kidob, 2] ← HAMIS
14                behoz ← laptábla[kidob, 2]
15                KIÍR(behoz, kidob)
16         BEOLVAS(r_t, laptábla[kidob, 2])             ▷ Beolvasás.
17         laptábla[r_t, 1] ← IGAZ                      ▷ Adatok frissítése.
18         laptábla[r_t, 2] ← foglalt
19         gyakoriság[r_t] ← 1
```

Az LFU-VÉGREHAJT algoritmus 11–13. sorában lévő ciklus magját legfeljebb n-szer kell végrehajtani, ezért az algoritmus futási ideje $O(np)$.

Bizonyos operációs rendszerekben a fizikai memóriában lévő lapokhoz tartozik két állapotbit. A *hivatkozott* bit minden hivatkozáskor (olvasáskor és íráskor) IGAZ-ra állítódik, a *piszkos* bit pedig minden módosításkor (azaz íráskor) IGAZ lesz. A program indításakor minden lap mindkét állapotbitjét HAMIS-ra állítjuk. Az operációs rendszer bizonyos időközönként (például k utasításonként) HAMIS-ra állítja azoknak a lapoknak a *hivatkozott* bitjét, amelyekre az előző átállítás óta nem volt hivatkozás.

A két állapotbit alapján a lapok négy osztályba sorolhatók: a *nulladik* osztályba a nem hivatkozott és nem módosított, az *első* osztályba a nem hivatkozott, módosított, a *második* osztályba a hivatkozott és nem módosított és végül a *harmadik* osztályba a hivatkozott és módosított lapok kerülnek.

Az NRU (**N**ot **R**ecently **U**sed) algoritmus a legkisebb indexű, nemüres osztályból választ cserélendő lapot. A determinisztikusság érdekében feltesszük, hogy az NRU algoritmus minden osztály elemeit egy sorban tárolja.

Ennek az algoritmusnak az előkészítése a jelzőbiteket tartalmazó *hivatkozott* és *piszkos* tömbök HAMIS értékekkel való feltöltését, az előző nullázás óta végrehajtott utasítások számát megadó *végre* változó nullázását és a négy üres sor létrehozását jelenti.

NRU-ELŐKÉSZÍT(*előkészít*, *piszkos*)

```
1  végre ← 0
2  for i ← 0 to n − 1
3      do hivatkozott[j] ← HAMIS
4         piszkos[j] ← HAMIS
5  Q1 ← ∅
6  Q2 ← ∅
7  Q3 ← ∅
8  Q4 ← ∅
```

NRU-VÉGREHAJT(*hivatkozott*, *piszkos*, *k*, *R*, *W*)

```
 1  if laptábla[r_t, 1] = IGAZ                          ▷ A következő lap benn van.
 2     then if W[r_t] = IGAZ
 3             then piszkos[r_t] ← IGAZ
 4  if laptábla[r_t, 1] = HAMIS                         ▷ A következő lap nincs benn.
 5     then laphiba ← laphiba + 1
 6          if foglalt < m                             ▷ A fizikai memória nincs tele.
 7            then behoz ← foglalt
 8                 foglalt ← foglalt + 1
 9                 hivatkozott[r_t] ← IGAZ
10                 if W[t] = IGAZ
11                   then piszkos[r_t] ← IGAZ
12          if foglalt = m                             ▷ A fizikai memória tele van.
13            then NRU-KIDOB(t, kidob)
14                 laptábla[kidob, 1] ← HAMIS
15                 behoz ← laptábla[kidob, 2]
16                 then if piszkos[kidob] = IGAZ
17                         then KIÍR(behoz, kidob)
18          BEOLVAS(r_t, laptábla[kidob, 2])           ▷ Beolvasás.
19          laptábla[r_t, 1] ← IGAZ                    ▷ Adatok frissítése.
20          laptábla[r_t, 2] ← behoz
21  if t/k = ⌊t/k⌋
22     then for i ← 0 to n − 1
23          if hivatkozott[i] = HAMIS
24            then piszkos[i] ← HAMIS
```

A cserélendő lap kiválasztása azon alapul, hogy a fizikai memóriában lévő lapokat négy sorba (Q_1, Q_2, Q_3, Q_4) soroljuk.

NRU-KIDOB(*idő*)

```
 1  for i ← 0 to n − 1                                  ▷ Lapok osztályozása.
 2      do if hivatkozott[i] = HAMIS
 3            then if piszkos[i] = HAMIS
 4                    then SORBA(Q1,i)
 5                    else SORBA(Q2,i)
 6            else if piszkos[i] = HAMIS
 7                    then SORBA(Q3,i)
 8                    else SORBA(Q4,i)
 9  if Q1 ≠ ∅                                           ▷ Cserélendő lap kiválasztása.
10    then kidob ← SORBÓL(Q1)
11    else if Q2 ≠ ∅
12            then kidob ← SORBÓL(Q3)
13            else if Q2 ≠ ∅
14                    then kidob ← SORBÓL(Q3)
15                    else kidob ← SORBÓL(Q4)
16  return kidob
```

A FUTTAT-NFU algoritmus helyigénye $\Theta(m)$, futási ideje pedig $\Theta(np)$.

A MÁSODIK-ESÉLY algoritmus a FIFO módosítása. Lényege, hogy ha a FIFO szerint cserélendő lapnak a *hivatkozott* változója hamis, akkor kidobjuk. Ha viszont a *hivatkozott* változója IGAZ, akkor azt HAMIS-ra állítjuk és a lapot a sor elejéről a sor végére tesszük, majd ezt ismételjük addig, míg olyan lapot nem találunk a sor elején, amelynek a *hivatkozott* változója HAMIS.

Ennek az ötletnek egy hatékonyabb megvalósítása az ÓRA algoritmus, amely egy ciklikus listában tárolja a memóriában lévő m lap indexét, és egy mutatót használ arra, hogy a következő cserélendő lapot kijelölje.

A LIFO (**L**ast **I**n **F**irst **O**ut) algoritmus lényege, hogy a fizikai memória igény szerinti feltöltése után mindig az utoljára beérkezett lapot cseréljük, azaz a kezdeti szakasz után $m-1$ lap állandóan a memóriában van – és az összes csere a legnagyobb című lapkeretben történik.

11.2.2. Dinamikus lapcsere

A számítógépek többségére az jellemző, hogy egyidejűleg több program hajtódik bennük végre. Ha ezekben a gépekben lapozott virtuális memória van, az kezelhető lokálisan és globálisan is. Előbbi esetben minden program igényét külön kezeljük, utóbbi esetben egy program helyigénye más programok rovására is kielégíthető.

A lokális kezelést megvalósító statikus lapcserélési algoritmusokat az előző pontban tárgyaltuk. Most két dinamikus algoritmust mutatunk be.

A WS (**W**orking **S**et) algoritmus alapja az a tapasztalat, hogy a programok futása során viszonylag rövid időn belül csak a lapjaik kis részére van szükség. Ezek a lapok alkotják az adott időintervallumhoz tartozó ***munkahalmazt.*** Ezt a munkahalmazt definiálhatjuk például úgy, mint az utolsó h utasítás során használt lapok halmazát. Az algoritmus működését elképzelhetjük úgy, hogy egy h hosszúságú „ablakot” csúsztatunk végig az R hivatkozási tömbön, és mindig az ablakban látható lapokat tartjuk a memóriában.

```
WS(laptábla, t, h)
 1  if laptábla[r_t, 1] = HAMIS                          ▷ A következő lap nincs benn.
 2    then WS-KIDOB(t)
 3         KIÍR(laptábla[kidob, 2], kidob)
 4         laptábla[r_t, 1] ← IGAZ
 5         laptábla[kidob, 2] ← kidob
 6  if t > h                                             ▷ Benn marad-e r_{t-h} a memóriában?
 7    then j ← h − 1
 8         while r_j ≠ r_{t-h} és j < t
 9               do j ← j + 1
10                  if j > t
11                    then laptábla[r_{t-h}, 1] ← HAMIS
```

A WS algoritmus elemzésekor az egyszerűség kedvéért feltesszük, hogy $h \leq n$, így akkor is megoldható az ablakban lévő lapok memóriában tárolása, ha mind a h hivatkozás különböző (a gyakorlatban a hivatkozási sorozatban lévő sok ismétlődés miatt h rendszerint lényegesen nagyobb, mint n).

A WS-KIDOB algoritmus lehet például egy olyan statikus lapcserélő algoritmus, amely a memóriában lévő összes lap közül – azaz „globálisan” – választja ki a cserélendő lapot. Ha például erre a célra a $\Theta(p)$ futási idejű FIFO algoritmust használjuk, akkor WS futási ideje – mivel legrosszabb esetben minden utasítással kapcsolatban meg kell vizsgálni az ablakban lévő lapokat – $\Theta(hp)$.

A PFF (**P**age **F**requency **F**ault) algoritmus is használ egy paramétert. Ez az algoritmus számon tartja az utolsó laphiba óta végrehajtott utasítások számát. Ha ez a szám egy újabb laphiba előfordulásakor kisebb, mind az előre megadott d paraméter értéke, akkor a program kap egy új lapkeretet a hibát okozó lap betöltéséhez. Ha azonban a laphiba nélkül végrehajtott utasítások száma eléri a d értéket, akkor előbb elveszünk a programtól minden olyan lapkeretet, amely az utolsó laphiba óta nem használt lapot tartalmaz, és csak ezután kap a program egy lapkeretet a hibát okozó lap tárolására.

```
PFF(laptáblat, d)
 1  számláló ← 0                                         ▷ Előkészítés.
 2  for i ← 1 to n
 3      do laptábla[i, 1] ← HAMIS
 4         hivatkozott[i] ← HAMIS
 5  for j ← 1 to p                                       ▷ Futtatás.
 6      do if laptábla[r_t, 1] = IGAZ
 7           then számláló ← számláló + 1
 8           else PFF-KIDOB(t, d, kidob)
 9                KIÍR(laptábla[kidob, 2], kidob)
10                laptábla[r_t, 1] ← IGAZ
11         for i ← to n
12             do if hivatkozott[i] = HAMIS
13                  then laptábla[i, 1] ← HAMIS
14                       hivatkozott[i] ← HAMIS
```

Gyakorlatok

11.2-1. Tekintsük a következő hivatkozási tömböt: $R = \langle 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6\rangle$. Hány laphiba fordul elő, ha a FIFO, LRU és OPT algoritmust alkalmazzuk k ($1 \leq k \leq 8$) lapkeretes fizikai memóriájú gépen?

11.2-2. Valósítsuk meg a FIFO algoritmust úgy, hogy a Q sor helyett egy mutatót kezelünk, amely mindig a fizikai memória következő, lapbetöltésre váró lapkeretére mutat.

11.2-3. Milyen előnyökkel és hátrányokkal járna, ha a lapcserélési algoritmusok a laptábla mellett egy $m \times 2$ méretű laptérképet is kezelnének, melynek j-edik sora a fizikai memória j-edik ($j \in [0\,..\,m-1]$) lapkeretének foglaltságát jelezné, illetve tartalmát adná meg?

11.2-4. Írjuk meg és elemezzük a Második-esély, az Óra és a LIFO algoritmusok pszeudokódját.

11.2-5. Csökkenthető-e az NFU futási idejének nagyságrendje, ha nem minden laphiba után osztályozzuk a lapokat, hanem folyamatosan karban tartjuk a négy sort?

11.2-6. Ismert az NRU algoritmus olyan NFU' változata is, amely a lapok osztályozására 4 halmazt használ, és a cserélendő lapot a legkisebb indexű nemüres halmazból *véletlenül* választja. Írjuk meg az ehhez szükséges Halmazba és Halmazból műveletek pszeudokódját és elemezzük az NFU' algoritmus erőforrásigényét.

11.2-7.⋆ Terjesszük ki úgy a lapcserélési automata definícióját, hogy a véges hivatkozási sorozat utolsó elemének feldolgozása után megálljon. *Útmutatás.* Egészítsük ki a bemenő jelek halmazát egy „sorozat vége" szimbólummal.

11.3. Anomáliák

Amikor az 1960-as évek elején az IBM Watson Kutató Intézetében az első lapcserélési algoritmusokat tesztelték, nagy meglepetést okozott, hogy bizonyos esetekben a fizikai memória méretének növelése a programok futási idejének *növekedését* okozta.

A számítógépes rendszerekben ***anomáliának*** nevezzük azt a jelenséget, amikor egy feladat megoldásához több erőforrást felhasználva rosszabb eredményt kapunk.

Három konkrét példát említünk. Az első a FIFO lapcserélési algoritmussal, a második a processzorok ütemezésére használt Listásan-ütemez algoritmussal, a harmadik az átfedéses memóriájú számítógépekben folyó párhuzamos programvégrehajtással kapcsolatos.

Érdemes megjegyezni, hogy a három példa közül kettőben is az a ritka eset fordul elő, hogy az anomália mértéke tetszőlegesen nagy lehet.

11.3.1. Lapcsere

Legyenek m, M, n és p pozitív egészek ($1 \leq m \leq M \leq n < \infty$), k nemnegatív egész, $A = \{a_1, a_2, \ldots, a_n\}$ egy véges ábécé. A^k az A feletti, k hosszúságú, A^* pedig az A feletti véges szavak halmaza.

Legyen m egy *kis*, M pedig egy *nagy* számítógép fizikai memóriájában lévő lapkeretek száma, n a háttérmemóriában lévő lapok száma (mindkét számítógépben), A a lapok halmaza.

A FIFO algoritmust már az előző alfejezetben definiáltuk. Mivel ebben a pontban csak

a FIFO lapcserélési algoritmust vizsgáljuk, a jelölésekből elhagyhatjuk a lapcserélési algoritmus jelét.

A laphibák számát $f_P(R,m)$-vel jelöljük. ***Anomáliának*** nevezzük azt a jelenséget, amikor $M > m$ és $f_P(R,M) > f_P(R,m)$. Ekkor az $f_P(R,M)/f_P(R,m)$ hányados az ***anomália mértéke***.

A P algoritmus hatékonyságát az $E_P(R,m)$ ***lapozási sebességgel*** jellemezzük, amit az $R = \rangle r_1, r_2, \ldots, r_p \langle$ véges hivatkozási sorozatra az

$$E_P(R,m) = \frac{f_P(R,m)}{p} , \tag{11.3}$$

$R = \rangle r_1, r_2, \ldots \rangle$ végtelen hivatkozási sorozatra pedig a

$$E_P(R,m) = \liminf_{k\to\infty} \frac{f_P(R_k,m)}{k} \tag{11.4}$$

módon definiálunk, ahol $R_k = (r_1, r_2, \ldots, r_k)$.

Legyen $1 \le m < n$ és $C = (1,2,\ldots,n)^*$ egy végtelen ciklikus hivatkozási sorozat. Ekkor $E_{\mathrm{FIFO}}(C,m) = 1$.

Ha végrehajtjuk az R = (1,2,3,4,1,2,5,1,2,3,4,5) hivatkozási sorozatot, akkor az $m = 3$ esetben 9, az $m = 4$ esetben pedig 10 laphibát kapunk, így $f_{\mathrm{FIFO}}(R,M)/f_{\mathrm{FIFO}}(R,m) = 10/9$.

Bélády, Nelson és Shedler a következő szükséges és elégséges feltételt adták az anomália létezésére.

11.1. tétel. *Akkor és csak akkor létezik olyan R hivatkozási sorozat, amelyre a* FIFO *lapcserélési algoritmus anomáliát okoz, ha $m < M < 2m - 1$.*

Az anomália mértékével kapcsolatban pedig a következőt bizonyították.

11.2. tétel. *Ha $m < M < 2m - 1$, akkor tetszőleges $\epsilon > 0$ számhoz létezik olyan $R = \langle r_1, r_2, \ldots, r_p \rangle$ hivatkozási sorozat, amelyre*

$$\frac{f(R,M)}{f(R,m)} > 2 - \epsilon . \tag{11.5}$$

Bélády, Nelson és Shedler a következőt sejtették.

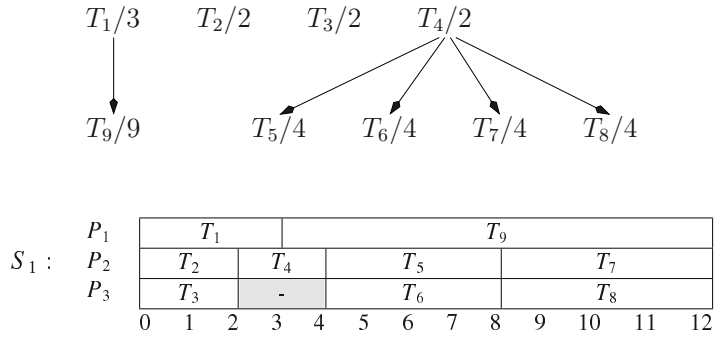
11.3. sejtés. *Tetszőleges R hivatkozási sorozatra és $M > m \ge 1$ memóriaméretekre*

$$\frac{f_{\mathrm{FIFO}}(R,M)}{f_{\mathrm{FIFO}}(R,m)} \le 2 . \tag{11.6}$$

A sejtést például a következő példával cáfolhatjuk.

Legyen $m = 5$, $M = 6$, $n = 7$, $k \ge 1$ és $R = UV^k$, ahol $U = \langle 1,2,3,\ 4,5,6,7,1,2,4,5,6,7,3,1,2,4,5,7,3,6,2,1,4,7,3,6,2,5,7,3,6,2,5\rangle$ és $V = (1,2,3,4,5,6,7)^3$.

Ha az U végrehajtási sorozatot $m = 5$ lapkeretet tartalmazó fizikai memóriával hajtjuk végre, akkor 29 laphiba következik be és a feldolgozás a (7,3,6,2,5) vezérlő állapotot eredményezi. Ezután a V hivatkozási sorozat minden végrehajtása 7 további laphibát okoz és



11.1. ábra. A τ_1 taszkrendszer, és annak optimális ütemezése.

ugyanazt a vezérlő állapotot eredményezi.

Ha az U sorozatot $M = 6$ lapkeretet tartalmazó fizikai memóriával hajtjuk végre, akkor a $\langle 2, 3, 4, 5, 6, 7\rangle$ vezérlő állapotot és 14 laphibát kapunk. Ezután V minden végrehajtása 21 további laphibát és ugyanazt a vezérlő állapotot eredményezi.

A $k = 7$ választással az anomália mértéke $(14 + 7 \times 21)/(29 + 7 \times 7) = 161/78 > 2$. Ha k értékét növeljük, akkor az anomália mértéke tart a háromhoz.

Ennél több is igaz: Fornai Péter és Iványi Antal következő tétele szerint az anomália mértéke tetszőlegesen nagy lehet.

11.4. tétel. *Tetszőlegesen nagy L számhoz megadhatók olyan m, M és R paraméterek, melyekre*

$$\frac{f(R,M)}{f(R,m)} > L\,. \tag{11.7}$$

11.3.2. Listás ütemezés

Tegyük fel, hogy n programot akarunk végrehajtani egy p processzoros láncon. A végrehajtásnak figyelembe kell vennie a programok közötti megelőzési relációt. A processzorok mohók, és a végrehajtás egy adott L lista szerint történik.

E. G. Coffman jr. 1976-ban leírta, hogy a p processzorszám csökkenése, az egyes programok végrehajtásához szükséges lépések t_i számának csökkenése, a megelőzési korlátozások enyhítése és a lista változtatása külön is anomáliát okozhat.

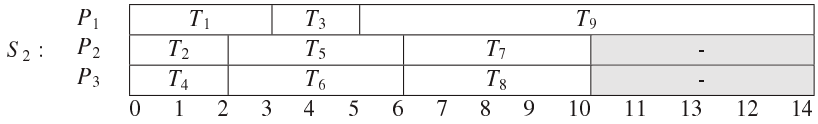
Legyen a programok végrehajtásának lépésszáma τ, a megelőzési reláció $<$, a lista L és a programok közös listás végrehajtásához szükséges lépések száma p azonos processzorból álló láncon $C(p, L, <, \tau)$. Az L' lista, a $<' \subseteq <$ megelőzési reláció, a $\tau' \leq \tau$ lépésszám vektor és a $p' \geq p$ processzorszám esetén a lépésszám legyen $C'(p', L', <', \tau')$.

Az anomália mértékét ezúttal relatív lépésszámmal jellemezzük.

Először négy példát mutatunk az anomália különböző típusaira.

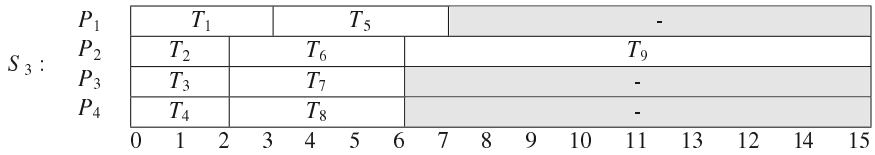
11.4. példa. Tekintsük az alábbi τ_1 taszkrendszert, és annak az $L = (T_1, T_2 \ldots, T_9)$ listával kapott S_1 ütemezését három ($m = 3$) azonos processzoron. Ekkor (lásd 11.1. ábra) $C_{\max}(S_1) = 12$, amiről könnyen belátható, hogy optimális érték.

11.5. példa. Ütemezzük az előbbi τ_1 taszkrendszert $m = 3$ azonos processzorra az $L' = \langle T_1, T_2, T_4, T_5, T_6, T_3, T_9, T_7, T_8 \rangle$ listával. Ekkor a kapott S_2 ütemezésre (lásd 11.2. ábra) $C_{max}(S_2) = 14$.



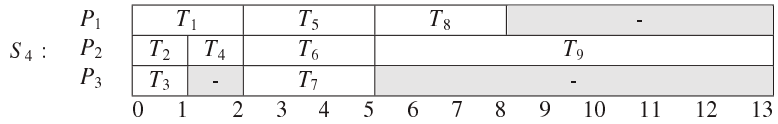
11.2. ábra. A τ_1 taszkrendszer ütemezése az L' lista esetén.

11.6. példa. Ütemezzük a τ_1 taszkrendszert az L listával $m' = 4$ processzorra. Az eredmény $C_{max}(S_3) = 15$ (lásd 11.3. ábra).



11.3. ábra. τ_1 ütemezése az L listával $m' = 4$ processzorra.

11.7. példa. Csökkentsük τ_1-ben a végrehajtási időket eggyel. Ütemezzük az így kapott τ_2 taszkrendszert az L listával $m = 3$ processzorra. Az eredmény: $C_{max}(S_4) = 13$ (ld. 11.4. ábra).

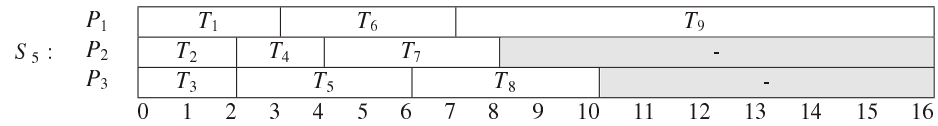


11.4. ábra. τ_2 ütemezése az L listával $m = 3$ processzorra.

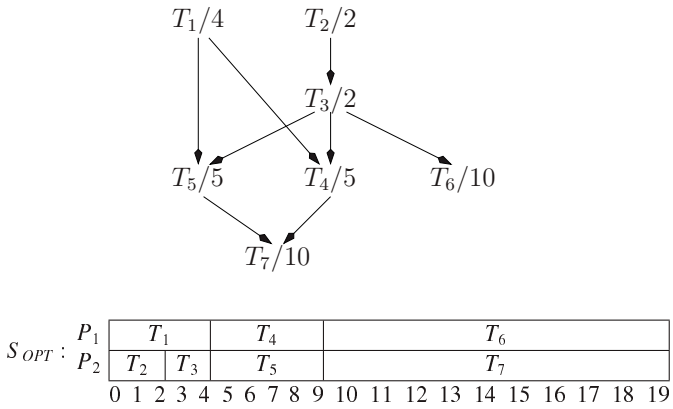
11.8. példa. Enyhítsük a τ_1 taszkrendszerben a precedencia-korlátozásokat: hagyjuk el a (T_4, T_5) és a (T_4, T_6) éleket a gráfból. Az így kapott τ_3 taszkrendszer S_5 ütemezésének eredménye a 11.5. ábrán látható: $C_{\max}(S_5) = 16$.

A következő példa azt mutatja, hogy a maximális befejezési idő növekedése nem csak a lista rossz megválasztása miatt következhet be.

11.9. példa. Legyen a τ taszkrendszer és annak S_{OPT} optimális ütemezése a 11.6. ábra szerinti. Ekkor $C_{max}(S_{\text{OPT}}) = 19$.

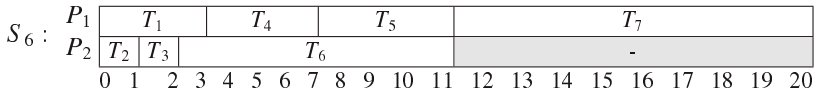


11.5. ábra. A τ_3 taszkrendszer ütemezése $m = 3$ processzorra.



11.6. ábra. A τ taszkrendszer, és annak S_{OPT} ütemezése két processzoron.

Könnyen belátható, hogy ha most a végrehajtási időket 1-gyel csökkentjük, akkor a kapott τ' taszkrendszeren $(S_6) = 20$ értéknél egyetlen listával sem érhetünk el jobbat (lásd a 11.10. ábrát).



11.7. ábra. A τ' taszkrendszer optimális listás ütemezése.

Ezen példák után az ütemezési paraméterek hatását jellemző relatív korlátot adunk meg. Tegyük fel, hogy adott τ és τ' taszkrendszerekre $\mathbf{T}' = \mathbf{T}$, $<' \subseteq <$, $\mathbf{t}' \leq \mathbf{t}$. A τ taszkrendszert egy L, a τ' taszkrendszert pedig egy L' lista alkalmazásával ütemezzük – mégpedig előbbit m, utóbbit pedig m' azonos processzorra. Az így kapott S ill. S' ütemezésekre legyen $C(S) = C$ és $C(S') = C'$.

11.5. tétel (ütemezési korlát). *A fenti feltételek mellett*

$$\frac{C'}{C} \leq 1 + \frac{m-1}{m'} \ . \tag{11.8}$$

Bizonyítás. Tekintsük a vesszős paraméterekhez (S'-höz) tartozó D' ütemezési diagramot.

Legyen a $[0, C')$ intervallum pontjainak két – A és B – részhalmazának definíciója a következő: $A = \{t \in [0, C) \mid$ a t időpontban minden processzor foglalt$\}$, $B = [0, C') \setminus A$. Érdemes megemlíteni, hogy mindkét halmaz diszjunkt, félig nyílt (balról zárt, jobbról nyílt) intervallumok uniója.

Legyen T_{j_1} egy olyan taszk, melynek a végrehajtása D' szerint az C' időpontban fejeződik be (azaz $f_{j_1} = C'$). Ekkor két lehetőség van: a T_{j_1} taszk s_{j_1} kezdőpontja vagy B belső pontja, vagy nem az.

1. Ha s_{j_1} B belső pontja, akkor B definíciója szerint van olyan processzor, amelyre $\varepsilon > 0$ mellett igaz, hogy az $[s_{j_1} - \varepsilon, s_{j_1})$ intervallumban nem dolgozik. Ez azonban csak úgy fordulhat elő, ha van olyan T_{j_2} taszk, amelyre $T_{j_2} < T_{j_1}$ és $f_{j_2} = s_{j_1}$ (*a* eset).
2. Ha s_{j_1} nem belső pontja B-nek, akkor vagy $s_{j_1} = 0$ (*b* eset), vagy $s_{j_1} > 0$. Ha van B-nek s_{j_1}-nél kisebb eleme (*c* eset), akkor legyen $x_1 = \sup\{x \mid x < s_{j_1}$ *és* $x \in B\}$, egyébként pedig legyen $x_1 = 0$ (*d* eset). Ha $x_1 > 0$, akkor A és B konstrukciójából következik, hogy van olyan processzor, amelyhez található olyan T_{j_2} taszk, amelynek ebben az intervallumban még tart a végrehajtása, és amelyre $T_{j_2} < T_{j_1}$.

A két esetet összegezve tehát vagy van olyan $T_{j_2} < T_{j_1}$ taszk, amelyre $y \in [f_{j_2}, s_{j_1})$ esetén $y \in A$ (*a* vagy *c* eset), vagy pedig minden $x < s_{j_1}$ számra $x \in A$ és $x < 0$ egyike fennáll (*a* vagy *d* eset).

Ezt az az eljárást ismételve olyan $T_{j_r}, T_{j_{r-1}}, \ldots, T_{j_1}$ taszklánchoz jutunk, amelyre igaz, hogy $x < s_{j_r}$ esetén vagy $x \in A$ vagy $x < 0$. Ezzel megmutattuk, hogy léteznek olyan taszkok, amelyekre

$$T_{j_r} <' T_{j_{r-1}} <' \cdots <' T_{j_1}, \tag{11.9}$$

továbbá minden $t \in B$ időpontban van olyan processzor, amelyik dolgozik, és éppen a lánc valamelyik elemét hajtja végre. Ebből következik, hogy

$$\sum_{\phi \in S'} t'(\phi) \le (m' - 1) \sum_{k=1}^{r} t'_{j_k} \ , \tag{11.10}$$

ahol ϕ-vel az üres periódusokat jelöltük, és így a bal oldali összeg az S'-ben lévő összes üres periódusra vonatkozik.

(11.9) és $<' \subseteq <$ alapján $T_{j_r} < T_{j_{r-1}} < \cdots < T_{j_1}$, és így

$$C \ge \sum_{k=1}^{r} t_{j_k} \ge \sum_{k=1}^{r} t'_{j_k} \ . \tag{11.11}$$

Mivel

$$mC \ge \sum_{i=1}^{n} t_i \ge \sum_{i=1}^{n} t'_i \ , \tag{11.12}$$

és

$$C' = \frac{1}{m'} \left(\sum_{i=1}^{n} t'_i + \sum_{\phi \in S'} t'(\phi) \right),$$

S_7 :

T_1	T_{m+1}
T_2	T_{m+2}
⋮	⋮
T_{m-1}	T_{2m-1}
T_m	

11.8. ábra. Az $L = (T_1, \ldots, T_n)$ listához tartozó $S_7(\tau_4)$ ütemezés.

S_8 :

T_{m+1}				T_m
T_{m+2}				-
⋮				⋮
T_{2m-1}				-
T_1	T_2	...	T_{m-1}	-

11.9. ábra. Az L' listához tartozó $S_8(\tau_4)$ ütemezés.

így (11.10), (11.11) és (11.12) alapján

$$C' \le \frac{1}{m'}\Big(mC + (m'-1)C\Big)\,,$$

ahonnan $C'/C \le 1 + (m-1)/m'$. ■

A következő példák nemcsak azt mutatják meg, hogy a tételben szereplő korlát a lehető legjobb, hanem azt is, hogy az adott korlát (legalábbis aszimptotikusan) a paraméterek bármelyikének változtatásával elérhető.

11.10. példa. Ebben a példában a lista változik, < üres, m tetszőleges. A végrehajtási idők a következők:

$$t_i = \begin{cases} 1, & ha\ i = 1, \ldots, m-1\,, \\ m, & ha\ i = m\,, \\ m-1, & ha\ i = m+1, \ldots, 2m-1\,. \end{cases}$$

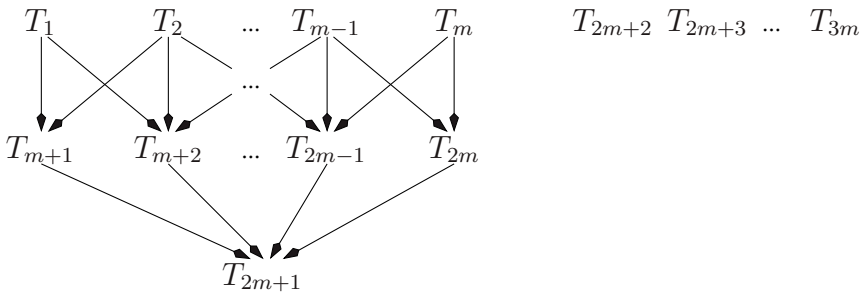
Ha ezt a τ_4 taszkrendszert m processzorra az $L = (T_1, \ldots, T_n)$ listával ütemezzük, akkor 11.8. ábrán lévő $S_7(\tau_4)$ optimális ütemezést kapjuk.

Ha az L lista helyett az $L' = (T_{m+1}, \ldots, T_{2m-1}, T_1, \ldots, T_{m-1}, T_m)$ listát alkalmazzuk, akkor a 11.9. ábrán látható $S_8(\tau_4)$ ütemezést kapjuk.

Ebben az esetben $C = (S_7) = m$, $C' = (S_8) = 2m - 1$, így $C'/C = 2 - 1/m$; tehát a lista változtatásával elértük, hogy a tétel állításában az egyenlőség teljesüljön, vagyis a $\le$ jel jobb oldalán szereplő kifejezés nem csökkenthető.

11.11. példa. Ebben a példában a végrehajtási időket csökkentjük. A lista mindkét esetben az $L = L' = \langle T_1, \ldots, T_n \rangle$. Itt és a fejezet további részében ε tetszőleges kis pozitív számot jelöl. Az eredeti végrehajtási időket a $\mathbf{t} = (t_1, \ldots, t_n)$ vektor tartalmazza, ahol

$$t_i = \begin{cases} 2\varepsilon, & ha\ i = 1, \ldots, m\,, \\ 1, & ha\ i = m+1, \ldots, 2m\,, \\ m-1, & ha\ i = 2m+1, \ldots, 3m\,. \end{cases}$$



11.10. ábra. τ_5 és τ'_5 azonos gráfja.

Az új végrehajtási idők

$$t'_i = \begin{cases} t_i - \varepsilon, & ha\ i = 1, \ldots, m-1\ , \\ t_i, & ha\ i = m, \ldots, 3m\ . \end{cases}$$

A τ_5, illetve a módosított τ'_5 taszkrendszer megelőzési gráfját 11.10. ábra mutatja, az $S_9(\tau_5)$ (optimális) ütemezés és az $S_{10}(\tau'_5)$ ütemezés pedig 11.11. ábrán látható. Itt $C = C_{\max}(S_9(\tau_5)) = m + 2\varepsilon$ és $C' =$

S_9 :

T_1	T_{m+1}	T_{2m+1}
T_2	T_{m+2}	T_{2m+2}
⋮	⋮	⋮
T_{m-1}	T_{2m-1}	T_{3m-1}
T_m	T_{2m-1}	T_{3m}

S_{10} :

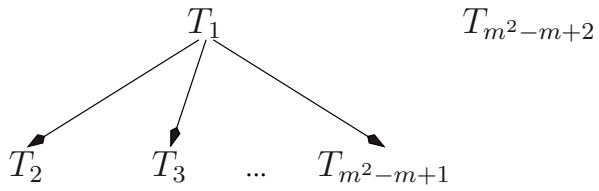
T_1	T_{2m+2}	T_{2m-1}	T_{2m+1}
T_2	T_{2m+3}	-	
⋮	⋮	⋮	
T_{m-1}	T_{3m}	-	
T_m	T_{m+1} ⋯	T_{2m-1}	-

11.11. ábra. Az $S_9(\tau_5)$ és $S_{10}(\tau'_5)$ ütemezések.

$C_{\max}(S_{10}(\tau'_5)) = 2m-1+\varepsilon$, ezért ε csökkenésével C'/C tart a $2-1/m$ értékhez ($\lim_{\varepsilon\to 0} C'/C = 2-1/m$). Tehát a végrehajtási időket változtatva tetszőlegesen megközelíthetjük a tételben szereplő korlátot.

11.12. példa. Ebben a példában a megelőzési korlátozásokat gyengítjük. A τ_6 taszkrendszer megelőzési gráfját 11.12. ábra mutatja. A taszkok végrehajtási ideje pedig: $t_1 = \varepsilon$, $t_i = 1$, ha $i = 1, \ldots, m^2 - m + 1$, és $t_{m^2-m+2} = m$. τ_6-nak az $L = (T_1, \ldots, T_{m^2-m+2})$ listához tartozó, optimális $S_{11}(\tau_6)$ ütemezését a 11.13. ábra tartalmazza. Az összes megelőzési korlátozás elhagyásával kapjuk τ_6-ból a τ'_6 taszkrendszert. A 11.14. ábra mutatja az $S_{12}(\tau'_6)$ ütemezést.

11.13. példa. Ezúttal a processzorok számát növeljük: m-ről m'-re. A τ_7 taszkrendszer gráfját 11.15.



11.12. ábra. A τ_6 taszkrendszer gráfja.

S_{11} :

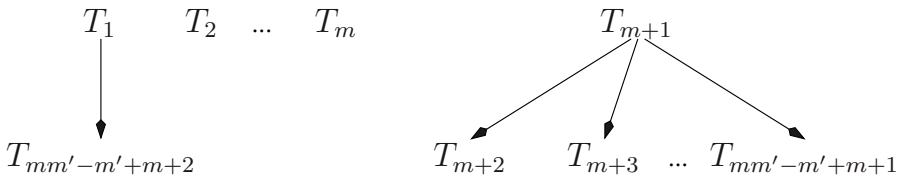
T_1	T_2	T_{m+1}	...	T_{m^2-2m+2}
T_{m^2-m+2}				-
-	T_3	T_{m+2}	...	T_{m^2-2m+3}
⋮	⋮	⋮	⋱	⋮
-	T_m	T_{2m-1}	...	T_{m^2-m+1}

11.13. ábra. Az $S_{11}(\tau_6)$ optimális ütemezés.

S_{12} :

T_1	T_{m+1}	...	T_{m^2-m+1}	-
T_2	T_{m+2}	...	T_{m^2-m+2}	
T_3	T_{m+3}	...	-	
⋮	⋮	⋱	⋮	
T_{m-1}	T_{2m+1}	...	-	
T_m	T_{2m}	...	-	

11.14. ábra. Az $S_{12}(\tau'_6)$ ütemezés.



11.15. ábra. τ_7 rákövetkezési gráfja.

ábra mutatja, a végrehajtási idők pedig

$$t_i = \begin{cases} \varepsilon, & ha\ i = 1, \ldots, m+1, \\ 1, & ha\ i = m+2, \ldots, mm' - m' + m + 1, \\ m', & ha\ i = mm' - m' + m + 2. \end{cases}$$

A taszkrendszer optimális ütemezését m, illetve m' processzoron 11.16., illetve 11.17. ábra mutatja. A maximális befejezési időket összehasonlítva itt is a kívánt aszimptotikus értéket kapjuk: $C = C_{\max}(S_{13}(\tau_7)) = m' + 2\varepsilon$, $C' = C_{\max}(S_{14}(\tau_7)) = m' + m - 1 + \varepsilon$, így $C'/C = 1 + (m-1-\varepsilon)(m'+2\varepsilon)$, valamint $\lim_{\varepsilon \to 0} C'/C = 1 + (m-1)/m'$.

S_{13} :

T_1	T_{m+1}	T_{m+2}	$\cdots$	T_a
T_2	$T_{mm'-m'+2}$			-
T_3	-	T_{m+3}	$\cdots$	T_b
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
T_m	-	T_{2m}	$\cdots$	T_c

11.16. ábra. Az $S_{13}(\tau_7)$ optimális ütemezés ($a = mm' - m' + 3$, $b = a + 1$, $c = mm' - m' + m + 1$).

S_{14} :

T_1	T_{m+2}	$\cdots$	T_a	$T_{mm'-m'+m+2}$
T_2	T_{m+3}	$\cdots$	T_{a+1}	-
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
T_b	T_c	$\cdots$	T_d	-
-	$\vdots$	$\ddots$	$\vdots$	$\vdots$
-	T_e	$\cdots$	T_f	-

11.17. ábra. Az $S_{14}(\tau_7)$ optimális ütemezés ($a = mm' - 2m' + m + 2$, $b = m + 1$, $c = 2m + 2$, $d = mm' - 2m' + 2m + 2$, $e = m + m' + 1$, $f = mm' - m' + m + 1$).

Ezeknek a példáknak a segítségével beláttuk a következő állítás helyességét.

11.6. tétel (ütemezési korlát élessége). *A relatív sebességre adott (11.8) korlát az m, t, $<$ és L paraméterek mindegyikének változására nézve (külön-külön is) aszimptotikusan éles.*

11.3.3. Párhuzamos feldolgozás átfedéses memóriával

Népszerű formában fogalmazzuk meg az átfedéses memóriájú számítógépek működését modellező párhuzamos algoritmust. A gombócsorozatok a feldolgozandó hivatkozási sorozatokat, az óriások a processzorokat, a falatok az egyidejűleg végrehajtott utasításokat modellezik.

A T_0, T_1, … T_r törpék n különböző fajtájú gombócot főznek. Ezeket az egyszerűség kedvéért a 0, 1, …, n számokkal jelöljük. Mindegyik törpe végtelen gombócsorozatot állít elő (a T_i törpe sorozatát G_i jelöli, a gombócsorozatok G mátrixát).

Ezeket a gombócokat O_f óriások eszik meg – ahol az f paraméter azt mutatja, hogy az adott óriás az egyes gombócfajtákból legfeljebb hányat ehet meg egy falatban.

Az O_f óriás a következőképpen eszik. Első falatához a T_0 törpe sorozatának elejéről a lehető legtöbb gombócot választja ki (de egy-egy fajtából legfeljebb f darabot). Ezt a falatot még a T_1, …, T_r törpék sorozatának elejéről kiegészíti – az f korlátot betartva.

A további falatokat hasonlóan állítja össze.

Legyen $h_i(f)$ ($i = 1, 2, \ldots$) az óriás i-edik harapásában lévő gombócok száma. Ekkor az O_f óriás S_f ***gombócevési sebességét*** az

$$S_f(G) = \liminf_{t\to\infty} \frac{\sum_{i=1}^{t} h_i(f)}{t} \tag{11.13}$$

határértékkel definiáljuk.

Könnyen belátható, hogy ha $1 \le f \le g$, akkor a gombócevési sebességekre fennáll

$$f \le S_f(G) \le fn, \quad g \le S_g(G) \le gn\ , \tag{11.14}$$

a két óriás relatív gombócevési sebességére pedig

$$\frac{f}{gn} \le \frac{S_f(G)}{S_g(G)} \le \frac{fn}{g}\ . \tag{11.15}$$

Most megmutatjuk, hogy a *kis óriás* gombócevési sebessége akárhányszor nagyobb lehet, mint a *nagy óriásé.*

11.7. tétel. *Ha* $r \ge 1$, $n \ge 3$, $g > f \ge 1$, *akkor léteznek olyan* G *gombócsorozatok, amelyekre egyrészt*

$$\frac{S_g(G)}{S_f(G)} = \frac{g}{fn}\ , \tag{11.16}$$

másrészt

$$\frac{S_g(G)}{S_f(G)} = \frac{gn}{f}. \tag{11.17}$$

Bizonyítás. A természetes korlátokat megadó (11.15) egyenlőtlenségben szereplő alsó korlát élességének belátásához tekintsük az

$$G_1 = 1^f 2^{2f+1} 1^* \tag{11.18}$$

és

$$G_2 = 1^{f+1}(2\ 3\ \ldots\ n)^* \tag{11.19}$$

sorozatokat.

A felső korlát élességét a

$$G_1 = 1\ 2^{2f+1}\ 1^* \tag{11.20}$$

és

$$G_2 = 1^{f-1}\ 3^{2f}\ 1^f\ (2\ 3\ \ldots\ n)^* \tag{11.21}$$

sorozatok „megevésével” láthatjuk be. ■

Az alsó korlát élessége azt fejezi ki, hogy a kis óriás ehet sokkal kevesebbet – ami természetes. Az n növelésével tetszőlegesen naggyá tehető felső korlát élessége azonban *erős anomália.*

11.3.4. Az anomália elkerülése

Az anomáliát általában igyekszünk elkerülni.

A lapcserélésnél például az elkerülés elégséges feltétele az, ha a cserélési algoritmus rendelkezik a ***veremtulajdonsággal:*** ha ugyanazt a hivatkozási sorozatot m és $m+1$ méretű memóriájú gépen futtatjuk, akkor minden hivatkozás után igaz az, hogy a nagyobb memória mindazokat a lapokat tartalmazza, amelyeket a kisebb tartalmaz.

A vizsgált ütemezési feladatnál elegendő az, ha nem követeljük meg az ütemező algoritmustól a lista alkalmazását.

Gyakorlatok

11.3-1. Adjunk meg olyan m, M, n, p és R paramétereket, amelyekre a FIFO algoritmus az M méretű fizikai memóriával legalább hárommal több laphibát okoz, mint az m méretű memóriával.

11.3-2. Adjunk meg olyan paramétereket, hogy a listás ütemezésnél a processzorszám növeléseker legalább másfélszeresére nőjön a maximális befejezési idő.

11.3-3. Adjunk meg olyan paramétereket, hogy a kis óriás gombócevési sebessége kétszerese legyen a nagy óriásénak.

11.4. Állományok optimális elhelyezése

Ebben az alfejezetben egy olyan memóriakezelési feladatot tárgyalunk, amelyben ismert méretű fájlokat kell elhelyezni adott méretű lemezeken. A cél a felhasznált lemezek számának minimalizálása.

A feladat megegyezik az *Új algoritmusok* című könyv *Közelítő algoritmusok* című fejezetében a feladatok között szereplő ládapakolási feladattal. Az ütemezéselmélet pedig a processzorszám minimalizálásával kapcsolatban használja ugyanezt a modellt.

Adott a fájlok n száma, valamint az elhelyezendő fájlok méretét tartalmazó $\mathbf{t} = (t_1, t_2, \ldots, t_n)$ vektor, melynek elemeire teljesül $0 < t_i \leq 1$ $(i = 1, 2, \ldots, n)$. A fájlokat úgy kell elhelyezni a lemezeken, hogy nem szabad őket részekre bontani és figyelembe kell venni, hogy a lemezek kapacitása egységnyi.

11.4.1. Közelítő algoritmusok

Az adott feladat NP-teljes. A gyakorlatban ezért különböző közelítő algoritmusokat használnak.

Ezeknek az algoritmusoknak a bemenő adatai: a fájlok n száma és a fájlméretek $\mathbf{t} = \langle t_1, t_2, \ldots, t_n \rangle$ vektora. A kimenő adatok pedig a szükséges lemezek száma (*lemezszám*) és a lemezek $\mathbf{h} = (h_1, h_2, \ldots h_n)$ szintvektora.

Lineáris algoritmus (LF)

A lineáris algoritmus (**L**inear **F**it) szerint az F_i fájlt az L_i lemezen helyezzük el. LF pszeudokódja a következő.

LF(n, $\mathbf{t}$, *lemezszám*)

```
1  for i ← 1 to n
2      do h[i] ← t[i]
3  lemezszám ← n
4  return lemezszám
```

Ennek az algoritmusnak mind a helyigénye, mind pedig a futási ideje $\Theta(n)$. Ha azonban a méretek beolvasását és a szintek nyomtatását a 2–3. sorokban lévő ciklusban oldjuk meg,

akkor a helyigény $O(1)$-re csökkenthető.

Egyszerű algoritmus (NF)
Az egyszerű algoritmus (**N**ext **F**it) addig rakja a fájlokat a soron következő lemezre, amíg lehet. Pszeudokódja a következő.

NF(n, **t**, *lemezszám*)

```
1  h[1] ← t[1]
2  lemezszám ← 1
3  for i ← 2 to n
4      do if h[lemezszám] + t[i] ≤ 1
5            then h[lemezszám] ← h[lemezszám] + t[i]
6            else lemezszám ← lemezszám + 1
7                 h[lemezszám] ← t[i]
7  return lemezszám
```

Ennek a algoritmusnak mind a helyfoglalása, mind pedig a futási ideje $\Theta(n)$. Ha a futási idők beolvasását és a szintek kivitelét a 3–6. sorokban lévő ciklusban oldjuk meg, akkor a helyfoglalás $O(1)$-re csökkenthető, a futási idő azonban $\Omega(n)$.

Mohó algoritmus (FF)
A mohó algoritmus (**F**irst **F**it) a fájlokat rendre az első olyan lemezen helyezi el, amelyikre ráférnek.

FF(n, **t**, *lemezszám*)

```
 1  lemezszám ← 1
 2  for i ← 1 to n
 3      do h[i] ← 0
 3  for i ← 1 to n
 4      do k ← 1
 5         while t[i] + h[k] > 1
 6               do k ← k + 1
 7         h[k] ← h[k] + t[i]
 8         if k > lemezszám
 9           then lemezszám ← lemezszám + 1
10  return lemezszám
```

Ennek az algoritmusnak a helyigénye $\Theta(n)$, időigénye pedig $O(n^2)$. Ha például minden fájlméret 1, akkor az algoritmus futási ideje $\Theta(n^2)$.

Gazdaságos algoritmus (BF)
A gazdaságos algoritmus (**B**est **F**it) a fájlokat rendre a legelső olyan lemezre rakja, ahol a lehető legkisebb szabad kapacitás marad.

BF(n, **t**, *lemezszám*)

```
 1  lemezszám ← 1
 2  for i ← 1 to n
 3      do h[i] ← 0
 4  for i ← 1 to n
 5      do szabad ← 1.0
 6         ind ← 0
 7         for k ← 1 to lemezszám
 8             do  if h[k] + t[i] ≤ 1 és 1 − h[k] − t[i] < szabad
 9                 then ind ← k
10                      szabad ← 1 − h[i] − t[i]
11     if ind > 0
12       then h[ind] ← h[ind] + t[i]
13       else lemezszám ← lemezszám + 1
14            h[lemezszám] ← t[i]
15  return lemezszám
```

Ennek az algoritmusnak a helyigénye $\Theta(n)$, futási ideje pedig $O(n^2)$.

Párosító algoritmus (PF)

A párosító algoritmus (**P**airwise **F**it) a méretvektor első és utolsó eleméből rendre párt képez, és a két fájlt – a két méret összegének megfelelően – egy, illetve két lemezre rakja.

A pszeudokódban két segédváltozó van: *bind* az aktuális pár első elemének indexe, *eind* pedig az aktuális pár második elemének indexe.

PF(n, **t**, *lemezszám*)

```
 1  lemezszám ← 0
 2  bind ← 1
 3  eind ← n
 4  while eind ≥ bind
 5        do if eind − bind ≥ 1
 6              then if t[bind] + t[eind] > 1
 7                      then lemezszám ← lemezszám + 2
 8                      h[lemezszám + 1] ← t[bind]
10                      h[lemezszám] ← t[eind]
11              else lemezszám ← lemezszám + 1
12                   h[lemezszám] ← t[bind] + t[eind]
13           if eind = bind
14              then lemezszám ← lemezszám + 1
15                   h[lemezszám] ← t[eind]
16                   bind ← bind + 1
17                   eind ← eind − 1
18  return lemezszám
```

Ennek az algoritmusnak a helyigénye Θn, futási ideje $\Theta(n)$. Ha azonban a fájlméretek beolvasását és a lemezszintek kivitelét online módon megoldjuk, a helyigény csak $\Theta(1)$.

Rendező egyszerű algoritmus (NFD)
A következő öt algoritmus két részből áll: először a végrehajtási idők alapján csökkenő sorrendbe rendezik a taszkokat, majd a második részben a rendezett taszkokat ütemezik.

A rendező egyszerű algoritmus (**N**ext **F**it **D**ecreasing) a rendezés után az egyszerű algoritmus (NF) szerint dolgozik. Így mind helyigénye, mind pedig időigénye az alkalmazott rendező algoritmus és NF megfelelő igényeiből tevődik össze.

Rendező mohó algoritmus (FFD)
A mohó rendező algoritmus (**F**irst **F**it **D**ecreasing) a rendezés után a mohó algoritmus (FF) szerint dolgozik, így helyigénye $\Theta(n)$, időigénye pedig $O(n^2)$.

Rendező gazdaságos algoritmus (BFD)
A gazdaságos rendező algoritmus (**B**est **F**it **D**ecreasing) a rendezés után a gazdaságos algoritmus (BF) szerint dolgozik, így helyigénye $\Theta(n)$, időigénye pedig $O(n^2)$.

Rendező párosító algoritmus (PFD)
A párosító rendező algoritmus (**P**airwise **F**it **D**ecreasing) a rendezés után rendre párokat képez a legelső és legutolsó taszkokból, majd azokat lehetőleg egy processzorra (ha a végrehajtási idők összege nem nagyobb egynél) ütemezi. Ha ez nem lehetséges, akkor az adott pár két processzorra ütemezi.

Rendező gyors algoritmus (QFD)
A gyors rendező algoritmus (**Q**uick **F**it **D**ecreasing) a rendezés után rendre a legelső fájlt a soron következő üres lemezre teszi, majd ehhez a fájlhoz mindaddig hozzáteszi a lehető legnagyobb fájlt (ezeket rendre a rendezett méretsorozat végétől kezdi keresni), amíg ez lehetséges.

A pszeudokódban használt munkaváltozók: *bind* az első vizsgálandó fájl indexe, *eind* pedig az utolsó vizsgálandó fájl indexe.

QFD(n, **t**)

```
 1  bind ← 1
 2  eind ← n
 4  lemezszám ← 0
 5  while eind ← bind
 6        do lemezszám ← lemezszám + 1
 7           h[lemezszám] ← s[bind]
 8           bind ← bind + 1
 9  while eind ≥ bind és h[lemezszám] + s[eind] ≤ 1
10        do ind ← eind
11  while ind > bind és h[lemezszám] + s[ind − 1] ≤ 1
12        do ind ← ind − 1
13  h[lemezszám] ← h[lemezszám] + t[ind]
```

```
14  if eind > ind
15     then for i ← ind to eind − 1
16               do s[i] ← s[i + 1]
17  eind ← eind − 1
18  return lemezszám
```

Ennek a programnak a helyigénye $O(n)$, futásideje kedvezőtlen esetben $\Theta(n^2)$, a gyakorlatban azonban – egyenletes eloszlású végrehajtási idők esetén – körülbelül $\Theta(n \log n)$.

11.4.2. Optimális algoritmusok

Egyszerű hatvány típusú optimális algoritmus (SP)
Ez az algoritmus a fájlokat rendre – egymástól függetlenül – n lemez mindegyikére elhelyezi, így n^n elhelyezést állít elő, majd ezek közül kiválaszt egy optimálisat. Mivel ez az algoritmus minden elhelyezést előállít (feltéve, hogy két elhelyezést azonosnak tekintünk, ha minden lemezhez ugyanazokat a fájlokat rendelik), így biztosan megtalálja az egyik optimális elhelyezést.

Faktoriális típusú optimális algoritmus (FACT)
Ez az algoritmus rendre előállítja a fájlok permutációit (ezek száma $n!$), majd az így kapott listákat helyezi el a NF segítségével.

Az algoritmus optimalitása így látható be. Tekintsünk egy tetszőleges fájlrendszert és annak egy $S_{\mathrm{OPT}(\tau)}$ optimális elhelyezését. $S_{\mathrm{OPT}}(\tau)$ alapján állítsuk elő a fájlok egy P permutációját úgy, hogy rendre felsoroljuk a $P_1, P_2, \ldots, P_{\mathrm{OPT}(\tau)}$ lemezekre elhelyezett fájlokat. Ha a P permutációt az NF algoritmussal helyezzük el, akkor vagy S_{OPT}-hoz jutunk, vagy egy másik optimális elhelyezéshez (bizonyos taszkok esetleg eggyel kisebb indexű processzorra kerülnek).

Gyors hatványtípusú optimális algoritmus (QP)
Ez az algoritmus (**Q**uick **P**ower) azzal próbálja SP időigényét csökkenteni, hogy a „nagy” fájlokat (melyek mérete nagyobb, mint 0.5) eleve külön lemezre helyezi, és csak a többi fájl (a „kicsi” fájlokat) próbálja mind az n lemezre elhelyezni. Így n^n helyett csak n^K elhelyezést állít elő, ahol K a kicsi fájlok száma.

Gazdaságos hatványtípusú optimális algoritmus (EP)
Ez az algoritmus (Economic Power) azt is figyelembe veszi (amellett, hogy két nagy fájl nem fér el egymás mellett), hogy két kis fájl mindig elfér egy lemezen. Ezért a nagy fájlok számát N-nel, a kicsikét K-val jelölve legfeljebb $N + \lfloor (K+1)/2 \rfloor$ lemezre van szüksége. Így először a nagy lemezeket ütemezzük külön lemezekre, majd a kicsiket a fenti számú lemez mindegyikére. Ha például $N = K = n/2$, akkor ezek szerint csak $(0.75n)^{0.5n}$ elhelyezést kell előállítanunk.

11.4.3. Listarövidítés (SL)

Bizonyos feltételek mellett igaz, hogy a $\mathbf{t}$ lista felbontható úgy $\mathbf{t}_1$ és $\mathbf{t}_2$ listákra, hogy $\mathrm{OPT}(\mathbf{t}_1) + \mathrm{OPT}(\mathbf{t}_2) \leq \mathrm{OPT}(\mathbf{t})$ (ilyen esetekben szükségképpen az egyenlőség teljesül). Ennek előnye, hogy a rövidebb listákat rendszerint rövidebb összidő alatt tudjuk optimálisan elhelyezni, mint az eredeti listát.

Tegyük fel például, hogy $t_i + t_j = 1$. Ekkor legyen $\mathbf{t}_1 = (t_i, t_j)$ és $\mathbf{t}_2 = \mathbf{t} \setminus \mathbf{t}_1$. Ekkor $\mathrm{OPT}(\mathbf{t}_1) = 1$ és $\mathrm{OPT}(\mathbf{t}_2) = \mathrm{OPT}(\mathbf{t}) - 1$. Tekintsük ugyanis egy optimális elhelyezés esetén azt a két lemezt, amelyekre a t_1 lista elemei kerültek. Mivel mellettük legfeljebb $1-t_1$, illetve $1-t_2$ összegű fájlok lehetnek, így azok végrehajtási időinek összege legfeljebb $2-(t_1+t_2)$, azaz 1. A listát egyszerre mindkét végén vizsgálva $O(n)$ idő alatt kiválaszthatjuk azokat a fájlpárokat, melyekre a végrehajtási idők összege 1. Ezután rendezzük a $\mathbf{t}$ listát. Legyen a rendezett lista $\mathbf{s}$. Ha például $s_1 + s_n > 1$, akkor az első fájl minden elhelyezésben külön lemezre kerül, és így $\mathbf{t}_1 = (t_1)$ és $\mathbf{t}_2 = (t_2, t_3, \ldots, t_n)$ jó választás.

Ha a rendezett listára $s_1 + s_n < 1$, de $s_1 + s_{n-1} + s_n > 1$, akkor legyen s_j a legnagyobb olyan listaelem, amelyet s_1-hez adva nem lépjük túl az egyet.

Ekkor $\mathbf{t}_1 = (t_1, t_j)$ és $\mathbf{t}_2 = \mathbf{t} \setminus \mathbf{t}_1$ választás mellett a $\mathbf{t}_2$ lista két elemmel rövidebb, mint a $\mathbf{t}$ lista volt.

Az utóbbi két művelettel gyakran lényegesen lerövidíthetőek a listák (szerencsés esetben úgy, hogy mindkét listára könnyen megkaphatjuk az optimális processzorszámot).

A rövidítés után megmaradó listát – például a korábbi algoritmusok valamelyikével – természetesen még fel kell dolgozni.

11.4.4. Becslések (ULE)

A felső és alsó becslésekre (**U**pper and **L**ower **E**stimations) támaszkodó algoritmusok a következőképpen működnek. Valamelyik közelítő algoritmussal előállítják az $\mathrm{OPT}(\mathbf{t})$ egy $\mathrm{A}(\mathbf{t})$ felső becslését, majd alulról is megbecsülik $\mathrm{OPT}(\mathbf{t})$ értékét. Erre alkalmasak például az elhelyezések azon tulajdonságai, hogy két nagy fájl nem helyezhető azonos lemezre, és hogy a méretek összege egyik lemezen sem lehet 1-nél több. Ezért mind a nagy fájlok száma, mind pedig a fájlméretek összege, így ezek $\mathrm{MAX}(\mathbf{t})$ maximuma is alsó becslést ad. Ha $\mathrm{A}(\mathbf{t}) = \mathrm{MAX}(\mathbf{t})$, akkor az A algoritmus optimális ütemezést állított elő. Ellenkező esetben például valamelyik időigényes optimumkereső algoritmussal folytathatjuk.

11.4.5. Algoritmusok páronkénti összehasonlítása

Ha egy ütemezési (vagy más) probléma megoldására több algoritmust ismerünk, akkor az algoritmusok összehasonlításának egyik egyszerű módja annak vizsgálata, hogy megadhatók-e a szereplő paraméterek értékei úgy, hogy a kiválasztott teljesítménymérték értéke az egyik algoritmus esetén kedvezőbb legyen, mint a másik algoritmus esetén.

A most vizsgált elhelyezési probléma esetén a $\mathbf{t}$ méretvektorhoz az A, illetve B algoritmus által rendelt processzorszámot $\mathrm{A}(\mathbf{t})$-vel, illetve $\mathrm{B}(\mathbf{t})$-vel jelölve azt vizsgáljuk, vannak-e olyan $\mathbf{t}_1$, illetve $\mathbf{t}_2$ vektorok, melyekre $\mathrm{A}(\mathbf{t}_1) < \mathrm{B}(\mathbf{t}_1)$, illetve $\mathrm{A}(\mathbf{t}_2) > \mathrm{B}(\mathbf{t}_2)$. Ezt a kérdést most az előbb definiált tíz közelítő és az optimális algoritmusra vizsgáljuk meg.

Az optimális algoritmusok definíciójából adódik, hogy minden $\mathbf{t}$-re és minden A algoritmusra $\mathrm{OPT}(\mathbf{t}) \leq \mathrm{A}(\mathbf{t})$.

A továbbiakban a példavektorok elemei huszadok lesznek.

	LF	NF	FF	BF	PF	NFD	FFD	BFD	PFD	QFD	OPT
$\mathbf{t}_1$	4	3	3	3	3	3	2	2	2	2	2
$\mathbf{t}_2$	6	2	2	2	3	3	3	3	3	3	2
$\mathbf{t}_3$	7	3	2	3	4	3	2	3	4	2	2
$\mathbf{t}_4$	8	3	3	2	4	3	3	2	4	3	2
$\mathbf{t}_5$	5	3	3	3	3	2	2	2	3	2	2
$\mathbf{t}_6$	4	3	2	2	2	3	2	2	2	2	2
$\mathbf{t}_7$	4	3	3	3	3	3	3	3	4	3	3
$\mathbf{t}_8$	4	3	3	3	2	3	2	2	2	2	2

11.18. ábra. Lemezszámok összefoglalása.

Tekintsük a következő hét listát:

$\mathbf{t_1}$ = (12/20, 6/20, 8/20, 14/20),
$\mathbf{t_2}$ = (8/20, 6/20, 6/20, 8/20, 6/20, 6/20),
$\mathbf{t_3}$ = (15/20, 8/20, 8/20, 3/20, 2/20, 2/20, 2/20),
$\mathbf{t_4}$ = (14/20, 8/20, 7/20, 3/20, 2/20, 2/20, 2/20, 2/20),
$\mathbf{t_5}$ = (10/20, 8/20, 10/20, 6/20, 6/20),
$\mathbf{t_6}$ = (12/20, 12/20, 8/20, 8/20),
$\mathbf{t_7}$ = (8/20, 8/20, 12/20, 12/20).

Ezeknek a listáknak az elhelyezési eredményeit a 11.18. ábrán foglaltuk össze.

A 11.18. ábra mutatja, hogy az első listához LF 4, míg a többi algoritmus ennél kevesebb lemezt igényel. Továbbá azt is mutatja $\mathbf{t}_1$ lista sora, hogy FFD, BFD, PFD, QFD és OPT kevesebb lemezt igényel, mint NF, FF, BF, PF és NFD.

Természetes, hogy nincs olyan lista, amelyre bármelyik algoritmus kevesebb lemezt használna fel, mint OPT.

Az is közvetlenül adódik, hogy nincs olyan lista, melyhez az LF kevesebb lemezt használna fel, mint a többi tíz algoritmus közül bármelyik.

Ezeket a megállapításokat mutatja a 11.19. ábra. Az ábrán a főátlóban lévő X szimbólumok azt jelzik, hogy az egyes algoritmusokat önmagukkal nem hasonlítjuk össze. Az első oszlopban lévő nagykötőjelek azt jelzik, hogy a sornak megfelelő algoritmushoz nincs olyan lista, melyet az algoritmus több lemez felhasználásával dolgozna fel, mint az oszlopnak megfelelő algoritmus, azaz LF.

Az utolsó sorban lévő nagykötőjelek pedig azt mutatják, hogy nincs olyan lista, melyhez az optimális algoritmus több lemezt igényelne, mint bármelyik másik vizsgált algoritmus.

Végül az egyesek azt jelzik, hogy a $\mathbf{t}_1$ listához az ábra adott mezője sorának megfelelő algoritmus több lemezt használ fel, mint a mező oszlopának megfelelő algoritmus.

Ha tovább folytatjuk a 11.19. ábra lemezszámainak elemzését, a 11.19. ábrát a 11.20. ábrává egészíthetjük ki.

Mivel az első sort és az első oszlopot már kitöltöttük, az LF algoritmussal nem foglalkozunk többet.

A $\mathbf{t}_2$ listához NF, FF, BF és OPT 2 lemezt használ, míg a többi 6 algoritmus hár-

	LF	NF	FF	BF	PF	NFD	FFD	BFD	PFD	QFD	OPT
LF	X	1	1	1	1	1	1	1	1	1	1
NF	–	X					1	1	1	1	1
FF	–		X				1	1	1	1	1
BF	–			X			1	1	1	1	1
PF	–				X		1	1	1	1	1
NFD	–					X	1	1	1	1	1
FFD	–						X				
BFD	–							X			
PFD	–								X		
QFD	–									X	
OPT	–	–	–	–	–	–	–	–	–	–	X

11.19. ábra. Algoritmusok páronkénti összehasonlítása.

mat. Ezért a „győztesek” oszlopainak és a „vesztesek” sorainak metszéspontjaiba ketteseket írunk (a PF és OPT, valamint az NFD és OPT metszéspontjában azonban a már ott lévő egyest nem írjuk felül, így $4 \times 6 - 2 = 22$ mezőbe kerül kettes). Mivel OTP sorát és oszlopát kitöltöttük, ezért a pont további részében nem foglalkozunk vele.

A harmadik lista hátrányos PF és PFD számára, ezért a soraikban lévő üres mezőkbe hármasokat írunk. Ez a lista arra is példa, hogy NF rosszabb lehet, mint FF, BF rosszabb lehet FF-nél, BFD az FFD-nél és a QFD-nél.

A negyedik listát csak BF és BFD tudja optimálisan – azaz két lemez felhasználásával – feldolgozni. Ezért a két algoritmus oszlopában a még üres mezőkbe négyest írhatunk.

Az ötödik listához NFD, FFD, BFD és QFD csak két, míg NF, FF, BF, PF és PFD három lemezt használ. Ezért a megfelelő mezőkbe ötös kerülhet.

A $\mathbf{t}_6$ lista „vesztesei” NF és NFD – ezért a soraikban még üresen álló mezőkbe hatost írunk.

A $\mathbf{t}_7$ lista feldolgozását PF jobban végzi, mint FF.

További mezők kitöltését segíti a következő

11.8. tétel. *Ha* $\mathbf{t} \in D$, *akkor*

$$\mathrm{FF}(\mathbf{t}) \leq \mathrm{NF}(\mathbf{t}) \ .$$

Bizonyítás. A lista hossza szerinti indukciót alkalmazunk.

Legyen $\mathbf{t} = \langle t_1, t_2, \ldots, t_n \rangle$ és $\mathbf{t_i} = t_1, t_2, \ldots, t_i \rangle$ $(i = 1, 2, \ldots, n)$. Legyen $\mathrm{NF}(\mathbf{t}_i) = \mathrm{N}_i$ és $\mathrm{FF}\mathbf{t}_i = \mathrm{F}_i$, továbbá legyen n_i az *utolsó lemez szintje* NF szerint, azaz a legnagyobb indexű, nem üres lemezre tett állományok hosszainak összege akkor, amikor NF éppen feldolgozta $\mathbf{t}_i$-t. Hasonlóképpen legyen f_i az utolsó lemez szintje FF szerint.

A következő invariáns tulajdonság teljesülését bizonyítjuk minden i-re: vagy $F_i < N_i$, vagy $F_i = N_i$ és $f_i \leq n_i$.

Ha $i = 1$, akkor $F_1 = N_1$ és $f_1 = n_1 = t_1$, azaz az invariáns tulajdonság második része teljesül. Tegyük fel, hogy a tulajdonság teljesül az $1 \leq i < n$ értékre. Ha most a t_{i+1} elhelyezése előtt az invariáns tulajdonság első része teljesült, akkor vagy érvényes marad az $F_i < N_i$ egyenlőtlenség, vagy pedig a lemezszámok egyenlők lesznek és $f_i < n_i$ fog fennállni. Ha most a t_{i+1} elhelyezése előtt a lemezszámok egyenlők voltak, akkor az elhelyezés

	LF	NF	FF	BF	PF	NFD	FFD	BFD	PFD	QFD	OPT
LF	X	1	1	1	1	1	1	1	1	1	1
NF	–	X	3	4	7	5	1	1	1	1	1
FF	–	–	X	4	7	5	1	1	1	1	1
BF	–	–	3	X	8	5	1	1	1	1	1
PF	–	2	2	2	X	3	1	1	1	1	1
NFD	–	2	2	2	6	X	1	1	1	1	1
FFD	–	2	2	2		–	X	4		–	2
BFD	–	2	2	2		–	3	X		3	2
PFD	–	2	2	2	3	3	3	3	X	3	2
QFD	–	2	2	2		–	–	4		X	2
OPT	–	–	–	–	–	–	–	–	–	–	X

11.20. ábra. Algoritmusok páronkénti összehasonlításának eredményei.

után vagy FF lemezszáma kisebb lesz, vagy pedig egyenlő lemezszám mellett FF utolsó lemezén szintje legfeljebb akkora lesz, mint NF utolsó lemezének szintje. ■

Hasonló állítás bizonyítható az NF–BF, NFD–FFD és NFD–BFD algoritmuspárokra. Indukcióval belátható, hogy FFD és QFD minden listára ugyanannyi lemezt igényelnek. Az eddigi megállapításokat a 11.20. ábrán összesítjük.

11.4.6. Közelítő algoritmusok hibája

Két algoritmus (A és B) relatív hatékonyságát gyakran jellemzik a kiválasztott hatékonysági mérték értékeinek hányadosával, jelen esetben az $A(\mathbf{t})/B(\mathbf{t})$ relatív processzorszámmal. Ennek a hányadosnak a felhasználásával különböző jellemzők definiálhatók. Ezeket két csoportba szokás sorolni: egyik csoportba a legrosszabb, míg a másikba az átlagos esetet jellemző mennyiségek kerülnek.

Itt csak a legrosszabb esettel foglalkozunk (az átlagos eset vizsgálata rendszerint lényegesen nehezebb).

Legyen $\mathcal{D}_n$ azon valós listák halmaza, amelyek n elemet tartalmaznak, és legyen $\mathcal{D}$ az összes valós lista halmaza, azaz

$$D = \cup_{i=1}^{\infty} D_i \ .$$

Legyen $\mathcal{A}_{lsz}$ a ***állományelhelyező,*** (azaz a minden $\mathbf{t} \in \mathcal{D}$ listához egy nemnegatív valós számot hozzárendelő, és így a $\mathcal{D} \to \mathbb{R}_0^+$ leképezést megvalósító) algoritmusok halmaza.

Legyen $\mathcal{A}_{opt}$ a minden listához az optimális lemezszámot rendelő algoritmusok halmaza, és OPT ennek a halmaznak egy eleme (azaz egy olyan algoritmus, amely minden $\mathbf{t} \in D$ listához megadja a listához tartozó fájlok elhelyezéséhez szükséges és elégséges lemezek számát).

Legyen $\mathcal{A}_{köz}$ azon $A \in \mathcal{A}_{lsz}$ algoritmusok halmaza, amelyekre $A(\mathbf{t}) \leq OPT(\mathbf{t})$ minden $\mathbf{t} \in D$ listára, és van olyan $\mathbf{t} \in D$ lista, amelyre $A(\mathbf{t}) > OPT(\mathbf{t})$. Legyen A_{becs} azon $E \in \mathbf{A}_{lsz}$ algoritmusok halmaza, amelyekre $E(\mathbf{t}) \leq OPT(\mathbf{t})$ minden $\mathbf{t} \in D$ listára, és van olyan $\mathbf{t} \in D$ lista, amelyre $E(\mathbf{t}) < OPT(\mathbf{t})$.

Legyen F_n azon valós listák halmaza, amelyekre $OPT(\mathbf{t}) = n$, azaz $F_n = \{\mathbf{t} | \mathbf{t} \in D$ és

$\mathrm{OPT}(\mathbf{t}) = n\}$ $(n = 1, 2, \ldots)$. A továbbiakban csak $\mathcal{A}_{lsz}$-beli algoritmusokat fogunk vizsgálni. Az A és B algoritmusok $(\mathrm{A},\ \mathrm{B} \in \mathcal{A})$ $R_{\mathrm{A,B},n}$ hibafüggvényét, $R_{\mathrm{A,B}}$ hibáját (abszolút hibáját) és $R_{A,\infty}$ aszimptotikus hibáját a következőképpen definiáljuk:

$$R_{A,B,n} = \sup_{t \in D_n} \frac{\mathrm{A}(\mathbf{t})}{\mathrm{B}(\mathbf{t})} ,$$

$$R_{A,B} = \sup_{t \in D} \frac{\mathrm{A}(\mathbf{t})}{\mathrm{B}(\mathbf{t})} ,$$

$$R_{A,\infty} = \lim_{n \to \infty} R_{A,B,n} .$$

Ezek a mennyiségek főleg akkor érdekesek, ha $\mathrm{B} \in \mathcal{A}_{opt}$. Ilyenkor az egyszerűség kedvéért a jelölésekből elhagyjuk a B-t, és az $\mathrm{A} \in \mathcal{A}$, illetve az $\mathrm{E} \in \mathcal{A}$ algoritmusok hibafüggvényéről, hibájáról és aszimptotikus hibájáról beszélünk.

A NF fájlelhelyező algoritmus jellemző adatai ismertek.

11.9. tétel. *Ha* $\mathbf{t} \in F_n$, *akkor*

$$n = \mathrm{OPT}(\mathbf{t}) \leq \mathrm{NF}(\mathbf{t}) \leq 2\mathrm{OPT}(\mathbf{t}) - 1 = 2n - 1 . \tag{11.22}$$

Továbbá, ha $k \in \mathbb{Z}$, *akkor léteznek olyan* $\mathbf{u}_k$ *és* $\mathbf{v}_k$ *listák, melyekre*

$$k = \mathrm{OPT}(\mathbf{u}_k) = \mathrm{NF}(\mathbf{u}_k) \tag{11.23}$$

és

$$k = \mathrm{OPT}(\mathbf{v}_k) \text{ és } \mathrm{NF}(\mathbf{v}_k) = 2k - 1 . \tag{11.24}$$

Ebből az állításból adódik a NF fájlelhelyező algoritmus hibafüggvénye, abszolút hibája és aszimptotikus hibája.

11.10. következmény. *Ha* $n \in \mathbb{Z}$, *akkor*

$$R_{\mathrm{NF},n} = 2 - \frac{1}{n} , \tag{11.25}$$

továbbá

$$R_{\mathrm{NF}} = R_{\mathrm{NF},\infty} = 2 . \tag{11.26}$$

A FF és BF fájlelhelyező algoritmus legrosszabb esetére vonatkozik a következő állítás.

11.11. tétel. *Ha* $\mathbf{t} \in F_n$, *akkor*

$$\mathrm{OPT}(\mathbf{t}) \leq \mathrm{FF}(\mathbf{t}),\ \mathrm{BF}(\mathbf{t}) \leq 1.7\mathrm{OPT}(\mathbf{t}) + 1 . \tag{11.27}$$

Továbbá, ha $k \in \mathbb{Z}$, *akkor léteznek olyan* $\mathbf{u}_k$ *és* $\mathbf{v}_k$ *listák, melyekre*

$$k = \mathrm{OPT}(\mathbf{u}_k) = \mathrm{FF}(\mathbf{u}_k) = \mathrm{BF}(\mathbf{u}_k) \tag{11.28}$$

valamint

$$k = \mathrm{OPT}(\mathbf{v}_k) \text{ és } \mathrm{FF}(\mathbf{v}_k) = \mathrm{BF}(\mathbf{v}_k) = \lfloor 1.7k \rfloor . \tag{11.29}$$

Ebből az állításból adódik FF és BF aszimptotikus hibája, valamint hibafüggvényük jó becslése.

11.12. következmény. *Ha* $n \in \mathbb{Z}$, *akkor*

$$\frac{\lfloor 1.7n \rfloor}{n} \leq R_{\mathrm{FF},n}, R_{\mathrm{BF},n} \leq \frac{\lceil 1.7n \rceil}{n} \tag{11.30}$$

továbbá

$$R_{\mathrm{FF},\infty} = R_{\mathrm{BF},\infty} = 1.7\ . \tag{11.31}$$

Ha n osztható tízzel, akkor a (11.30) egyenlőtlenségben az alsó és felső határok megegyeznek, azaz ebben az esetben $1.7 = R_{FF,n} = R_{BF,n}$.

Gyakorlatok

11.4-1. Példával bizonyítsuk, hogy a FF és BF algoritmusok abszolút hibája legalább 1.7.
11.4-2. Valósítsuk meg a FF és BF algoritmusok alapgondolatát úgy, hogy a futási idő $O(n \lg n)$ legyen.
11.4-3. Egészítsük ki a 11.20 ábrát.

Feladatok

11-1. Lapcserélési anomália elkerülése
Osztályozzuk a tárgyalt lapcserélési algoritmusokat aszerint, hogy biztosítják-e az anomália elkerülését.
11-2. Optimális lapcserélési algoritmus
Bizonyítsuk be, hogy minden A igény szerinti lapcserélési algoritmusra, m memóriaméretre és R hivatkozási tömbre teljesül, hogy $f_{\mathrm{A}}(m, R) \geq f_{\mathrm{OPT}}(m, R)$.
11-3. Anomália
Tervezzünk egy algoritmust (és valósítsuk is meg), amelynél előfordulhat, hogy egy adott feladatot $q > p$ processzoron megoldani tovább tart, mint $p > 1$ processzoron.
11-4. Fájlelhelyezési algoritmusok hibája
Adjunk alsó és felső korlátokat a BF, BFD, FF és FFD algoritmusok hibájára.

Megjegyzések a fejezethez

A lapcserélési algoritmusokat Silberschatz, Galvin és Gagne [431], valamint Tanenbaum és Woodhull [459] tankönyvei alapján tárgyaljuk.

A lapcserélési algoritmusok Mealy-automatával való definiálása Denning összefoglaló cikkén [108], Gécseg Ferenc és Peák István [151], Hopcroft, Motwani és Ullman [209], valamint Peák István [365] tankönyvein alapul.

A MIN algoritmus optimalitását Mihnovszkij és Shor 1965-ben [329], majd Mattson, Gecsei, Slutz és Traiger 1970-ben [313] bizonyították.

A FIFO lapcserélési algoritmusnál a gyakorlatban tapasztalt anomáliát először Bélády

László [51] írta le 1966-ban, majd Shedlerrel közös dolgozatában [52] konstruktív módon bizonyította, hogy az anomália mértéke tetszőlegesen megközelítheti a kettőt. Ugyanebben a cikkben fogalmazták meg (1969-ben) azt a sejtést, hogy az anomália mértéke nem érheti el a kettőt. Fornai Péter és Iványi Antal [135] 2002-ben megmutatták, hogy a nagy és kisebb memóriájú számítógépekben szükséges lapcserék számának hányadosa akármilyen nagy lehet.

Ütemezési anomáliára példák szerepelnek Coffman [82], Iványi és Szmeljánszkij [222], valamint Roosta [396] könyvében, továbbá Lai és Sahni [274] cikkében.

Ennek a fejezetnek az anyaga több helyen is kapcsolódik az *Ütemezéselmélet* című fejezethez. Például az 11.5. tétel $m = m'$, $\mathbf{t} = \mathbf{t}'$, $<=<'$ speciális esete ott szerepel a listás ütemezéssel kapcsolatban. Az FF szegmenselhelyező algoritmus pedig ott ládapakoló algoritmusként szerepet játszik az egyik többprocesszoros eredmény bizonyításában.

Az átfedéses memória elemzése a [218] cikkből származik.

Az $\mathrm{NF}(\mathbf{t}) \leq 2\mathrm{OPT}(\mathbf{t}) + 2$ korlát D. S. Johnson PhD értekezésében szerepelt [231], a pontos 11.11. tétel [219]-ből származik. A FF-re és BF-re vonatkozó felső korlát Johnson, Demers, Ullman, Garey és Graham [233] eredménye, a korlát pontosságára vonatkozó bizonyítás pedig Iványi Antalé [219, 220]. Az FFD-re és BFD-re vonatkozó felső korlát forrása [233], az NFD-re vonatkozó korláté pedig [26]. A fájlelhelyezési feladat NP-nehézségének bizonyítása – a részletösszeg feladatra való visszavezetéssel – szerepel az *Új algoritmusok* közelítő algoritmusokkal foglalkozó fejezetében [90].

A témakör magyar nyelvű tankönyvei közül Varga László munkáját [477], Kernighan és Pike [252], Kóczy és Kondorosi [248] Tanenbaum és Woodhull [459] művét, valamint Galambos Gábor [146] 2003-ban megjelent tankönyvét ajánljuk.

12. Relációs adatmodell tervezés

12.1. Bevezetés

A relációs adatmodellt Codd vezette be 1970-ben. Egyszerűsége, kezelhetősége és rugalmassága miatt ma is ez a legelterjedtebb adatbázis szervezési módszer, kiegészítve a World Wide Web lehetőségeivel. A modell lényege, hogy az adatokat relációs táblákban tárolja, ahol a relációk sorai az egyedek rekordjainak felelnek meg, az oszlopok pedig az egyes tulajdonságoknak, más néven ***attribútumoknak.*** A ***relációs séma*** egy vagy több relációból és azok attribútumhalmazaiból áll. Ebben a fejezetben az egyszerűség kedvéért mindig egyetlen relációból álló sémát fogunk tekinteni. A sémán belül a reláció matematikai fogalmától kissé eltérően az attribútumok sorrendje nem lényeges, mindig attribútum ***halmazról*** és nem listáról beszélünk. Minden egyes attribútumhoz hozzátartozik az ***értelmezési tartománya,*** amely azon elemi értékek halmaza, amelyek egy rekord adott attribútumnak megfelelő értékei lehetnek. Példaképpen tekinthetjük az

Alkalmazottak(Név,Anyja neve,TAJ-szám,Beosztás,Fizetés)

sémát. Itt a ***Név*** és ***Anyja neve*** attribútumok értelmezési tartománya a véges karaktersorozatok halmaza (illetve ennek az a részhalmaza, ami az összes lehetséges nevet tartalmazza), a ***TAJ-szám*** értelmezési tartománya bizonyos formai és paritás ellenőrzési feltételeknek eleget tevő egész számok halmaza, a ***Beosztás*** az {igazgató,osztályvezető,szervező,programozó,portás,takarító,mindenes} halmazból vehet fel értékeket. Egy r reláció az R relációs séma egy ***példánya,*** ha oszlopai megfelelnek a séma attribútumainak és a soraiban az egyes attribútumoknak megfelelő helyeken az attribútum értelmezési tartományából vett értékek állnak. Az Alkalmazottak sémához tartozó reláció egy tipikus sora lehet például

(Pató Pál,Szende Ibolya,184-803-151,szervező,244000) .

A reláció egyes adatai közt különböző összefüggések lehetnek, például jelen esetben a TAJ-szám már az összes többi adatot meghatározza. Hasonlóképpen, a (Név, Anyja neve) páros is egyértelmű azonosító. Természetesen az is előfordulhat, hogy bizonyos attribútumhalmazok nem a teljes rekordot, hanem csak egy részét határozzák meg egyértelműen.

A relációs sémához hozzátartoznak különböző integritási feltételek. Ezek legfontosabb típusa a ***funkcionális függőség.*** Ha U és V attribútumhalmazok, akkor a $U \rightarrow V$ jelentése az, hogy ha két rekord megegyezik az U-beli attribútumokban, akkor szükségképpen megegyeznek a V-beli attribútumokban is. Itt, és a továbbiakban is azzal az egyszerűsítéssel

élünk, hogy egy $\{A_1, A_2, \ldots, A_k\}$ attribútumhalmazt $A_1A_2 \ldots A_k$-ként jelölünk.

12.1. példa. *Funkcionális függőségek.* Tekintsük például az

R(**T**anár,**Tá**rgy,**Te**rem,**D**iák,**J**egy,**I**dő)

sémát. Egy rekord jelentése, hogy egy adott diák egy adott tárgyból, melyet egy adott tanár tanít adott teremben és adott időpontban, kapott egy adott osztályzatot. Itt a következő funkcionális függőségek teljesülnek.

Tá→T: Egy tantárgyat egy tanár tanít.
IT→Te: Egy tanár egy időpontban csak egy teremben tanít.
ID→Te: Egy diák egy időpontban csak egy teremben hallgat előadást.
ID→Tá: Egy diák egy időpontban egy tárgy előadását hallgatja.
TáD→J: Egy diák egy tárgyból egy végső osztályzatot kap.

A 12.1. példában az **ID** attribútumhalmaz minden további attribútum értékét egyértelműen meghatározza, és az ilyen tulajdonságú attribútum halmazok közül (tartalmazásra nézve) minimális. ***Kulcsnak*** nevezzük az ilyen attribútumhalmazokat. Ha egy attribútumhalmaztól az összes többi funkcionálisan függ, akkor ***szuperkulcsnak*** nevezzük. Világos, hogy minden szuperkulcs tartalmaz kulcsot, továbbá minden attribútumhalmaz, amelyik szuperkulcsot tartalmaz, maga is szuperkulcs.

12.2. Funkcionális függőségek

Egy adott relációs sémában teljesülő funkcionális függőségek közül némelyek már a tervezési szakaszban ismeretesek, mások pedig ezen függőségek következményei. A 12.1. példában az **ID→Tá** és **Tá→T** függőségek következménye az **ID→T** függőség. Valóban, ha két rekord megegyezik az **ID** attribútumokon, akkor **ID→Tá** alapján a **Tá** attribútumon is. Ekkor azonban a **Tá→T** függést használva kapjuk, hogy **T**-n is megegyeznek. Tehát teljesül **ID→T**.

12.1. definíció. *Legyen R egy relációs séma, F funkcionális függőségek egy halmaza R-en. Azt mondjuk, hogy egy $U \to V$ funkcionális függőség az F függőségi halmaz* ***logikai következménye,*** *ha minden olyan R-hez tartozó példányban, amiben F minden függősége teljesül, $U \to V$ is igaz. Jelölésben: $F \models U \to V$. Az F funkcionális függőség halmaz* ***lezártja*** *az F^+, melyre*

$$F^+ = \{U \to V \colon F \models U \to V\} .$$

12.2.1. Armstrong-axiómák

A kulcsok meghatározásához, illetve hogy a funkcionális függőségek közti logikai következményeket kellően megértsük, szükségünk van arra, hogy egy F funkcionális függőség család F^+ lezárását meghatározzuk, illetve hogy egy adott $X \to Z$ funkcionális függőségről el tudjuk dönteni, F^+-ban van-e. Ehhez következtetési szabályokra van szükségünk, amelyek arról szólnak, hogy néhány funkcionális függőségből milyen más függőségek teljesülésére következtethetünk. Az ***Armstrong-axiómák*** ilyen ***teljes*** és ***ellentmondásmentes*** rendszert alkotnak. Ellentmondásmentes, azaz csak olyan funkcionális függőségek vezethetők

le vele, melyek minden olyan adatbázisban teljesülnek, ahol F igaz. Ezt a tulajdonságot szokás ***helyesnek*** is nevezni. ***Teljes,*** azaz ha egy $X \to Z$ funkcionális függőség logikailag következik F-ból, akkor az Armstrong-axiómák segítségével le is vezethető.

ARMSTRONG-AXIÓMÁK

(A1) *Reflexivitás.* Ha $Y \subseteq X \subseteq R$, akkor $X \to Y$.

(A2) *Bővítés.* Ha $X \to Y$ teljesül, akkor tetszőleges $Z \subseteq R$-ra $XZ \to YZ$ is teljesül.

(A3) *Tranzitivitás.* Ha $X \to Y$ és $Y \to Z$ teljesül, akkor $X \to Z$ is igaz.

12.2. példa. *Levezetés Armstrong-axiómák alapján.* Legyen $R = ABCD$ és $F = \{A \to C, B \to D\}$. Ekkor AB kulcs lesz:

1. $A \to C$ adott.
2. $AB \to ABC$ 1.-et (A2) alapján kiegészítjük AB-vel.
3. $B \to D$ adott.
4. $ABC \to ABCD$ 3.-at (A2) alapján kiegészítjük ABC-vel.
5. $AB \to ABCD$ 2.-re és 4.-re alkalmazzuk a tranzitivitást (A3).

Ezzel beláttuk, hogy AB szuperkulcs. Az, hogy ténylegesen kulcs, következik a LEZÁRÁS algoritmusból.

Természetesen az (A1)–(A3) szabályokon kívül mások is érvényesek. A következő lemmában felsorolunk néhányat, a bizonyítást feladatként az Olvasóra bízva (12.2-3. gyakorlat).

12.2. lemma.

1. *Egyesítési szabály:* $\{X \to Y, X \to Z\} \models X \to YZ$.
2. *Pszeudotranzitivitás:* $\{X \to Y, WY \to Z\} \models XW \to YZ$.
3. *Szétvágási szabály: Ha* $X \to Y$ *teljesül és* $Z \subseteq Y$, *akkor* $X \to Z$ *is teljesül.*

Az (A1)–(A3) rendszer ellentmondásmentessége a levezetés hossza szerinti indukció alapján könnyen látható. A teljességet a következő lemma alapján a LEZÁRÁS algoritmus helyességének bizonyításakor látjuk be. Jelölje X^+ az $X \subseteq R$ attribútumhalmaz F funkcionális függőség család szerinti ***lezárását,*** azaz azon A attribútumok halmazát, melyekre $X \to A$ következik F-ból az Armstrong-axiómák alapján.

12.3. lemma. *Az F funkcionális függőség családból az $X \to Y$ funkcionális függőség következik az Armstrong-axiómák alapján akkor és csak akkor, ha $Y \subseteq X^+$.*

Bizonyítás. Tegyük fel, hogy $Y = A_1A_2 \ldots A_n$, ahol az A_i-k attribútumok, és legyen $Y \subseteq X^+$. X^+ definíciója alapján $X \to A_i$ következik az Armstrong-axiómákból minden $1 \leq i \leq n$-re. A 12.2. lemma egyesítés szabálya alapján $X \to Y$ következik.

Másfelől, tegyük fel, hogy $X \to Y$ következik az axiómákból. A 12.2. lemma szétvágási szabálya alapján $X \to A_i$ következik az Armstrong-axiómákból minden $1 \leq i \leq n$-re. Tehát $Y \subseteq X^+$. ■

12.2.2. Lezárások

A lezárások számítása fontos funkcionális függőség rendszerek ekvivalenciájának, illetve logikai következményének tesztelésekor. A legelső gondolatunk az lehetne, hogy ha adott funkcionális függőségek egy F rendszere, akkor az $F \models \{X \to Y\}$ eldöntéséhez elegendő F^+-t kiszámolni, majd ellenőrizni, hogy vajon $\{X \to Y\} \in F^+$ teljesül-e. Azonban F^+ mérete a bemenet méretében exponenciális is lehet. Tekintsük ugyanis az

$$F = \{A \to B_1, A \to B_2, \ldots, A \to B_n\}$$

funkcionális függőség rendszert. Ekkor F^+ az összes olyan $A \to Y$ funkcionális függőségből fog állni, ahol $Y \subseteq \{B_1, B_2, \ldots, B_n\}$. Ezek száma pedig 2^n. Azonban, egy X attribútumhalmaz F funkcionális függőségi halmaz szerinti X^+ lezártját az összes F-beli funkcionális függőség számának függvényében lineáris időben ki lehet számítani. Egy, az X^+ lezárás kiszámítására szolgáló algoritmus a következő. Bemenetként attribútumok egy véges R halmazát, R-n értelmezett funkcionális függőségek egy F halmazát, valamint egy $X \subseteq R$ attribútumhalmazt adunk meg.

LEZÁRÁS(R,F,X)

```
 1  X^(0) ← X
 2  i ← 0
 3  G ← F                          ▷ A még nem használt funkcionális függőségek.
 4  repeat
 5        X^(i+1) ← X^(i)
 6        for minden G-beli Y → Z-re
 7            do if Y ⊆ X^(i)
 8                 then X^(i+1) ← X^(i+1) ∪ Z
 9                      G ← G − {Y → Z}
10        i ← i + 1
11  until X^(i−1) = X^(i)
12  X^+ ← i
13  return X^+
```

Könnyű látni, hogy azok az attribútumok, amelyeket LEZÁRÁS(R,F,X) valamelyik $X^{(j)}$-be behelyez, valóban X^+-ban vannak. Az algoritmus helyessége bizonyításának nehezebbik iránya azt belátni, hogy minden X^+-ba tartozó attribútum valóban belekerül valamelyik $X^{(j)}$-be.

12.4. tétel. LEZÁRÁS(R,F,X) *helyesen számítja ki* X^+*-t.*

Bizonyítás. Először teljes indukcióval belátjuk, hogy ha egy A attribútum valamikor LEZÁRÁS(R,F,X) során egy $X^{(j)}$-be belekerül, akkor A valóban X^+-ban van.
Kezdő lépés: $j = 0$. Ekkor $A \in X$, azaz a reflexivitás (A1) alapján $A \in X^+$.
Indukciós lépés: Tegyük fel, hogy $j > 0$ és $X^{(j-1)}$ csak X^+-beli attribútumokat tartalmaz. A azért került bele $X^{(j)}$-be, mert F tartalmaz egy $Y \to Z$ funkcionális függőséget, ahol $Y \subseteq X^{(j-1)}$ és $A \in Z$. Az indukciós feltevés alapján $Y \subseteq X^+$. A 12.3. lemma szerint $X \to Y$ teljesül. A tranzitivitás (A3) szerint $X \to Y$ és $Y \to Z$-ből $X \to Z$ következik. A reflexivitás (A1) és $A \in Z$ miatt $Z \to A$ igaz, innen a tranzitivitás újbóli alkalmazásával kapjuk, hogy

$X \to A$, azaz $A \in X^+$.

Másfelől, belátjuk, hogy ha $A \in X^+$, akkor A benne van a LEZÁRÁS(R,F,X) által eredményül adott halmazban. Tegyük fel ellenkezőleg, hogy $A \in X^+$, de $A \notin X^{(i)}$, ahol $X^{(i)}$ a LEZÁRÁS(R,F,X) által eredményül adott halmaz. A 9. sor szerint ekkor $X^{(i)} = X^{(i+1)}$. Készítünk egy r reláció példányt, melyben F összes funkcionális függősége teljesül, azonban $X \to A$ nem áll fenn, ha $A \notin X^{(i)}$. Legyen r a következő két sorból álló reláció:

$X^{(i)}$-beli attribútumok				Többi attribútum			
1	1	...	1	1	1	...	1
1	1	...	1	0	0	...	0

Tegyük fel, hogy az így megadott r megsérti valamelyik $U \to V$ F-beli funkcionális függőséget, azaz $U \subseteq X^{(i)}$ de V nem részhalmaza $X^{(i)}$-nek. Ekkor azonban LEZÁRÁS(R,F,X) nem állhatott még meg, mert $X^{(i)} \neq X^{(i+1)}$.

Mivel $A \in X^+$, a 12.3. lemma alapján $X \to A$ következik F-ből az Armstrong-axiómák alapján. Az axiómák ellentmondásmentesek, ezért minden olyan relációban, ahol F teljesül, ott az $X \to A$ funkcionális függőségnek is igaznak kell lennie. Azonban, az r reláció példányban ez csak úgy lehetséges, ha $A \in X^{(i)}$. ■

Vegyük észre, hogy az előbb megadott r reláció példány az Armstrong-axiómák teljességének bizonyítását is megadja. Ugyanis a LEZÁRÁS által kiszámolt X^+ lezárás azon attribútumok halmaza, amelyekre $X \to A$ következik F-ből az Armstrong-axiómák alapján. Ugyanakkor minden más B attribútumra r-nek van két sora, melyek megegyeznek X-en, de különböznek a B attribútumon, azaz nem teljesül, hogy $F \models X \to B$.

LEZÁRÁS lépésszáma $O(n^2)$, ahol n az input hossza. Ugyanis a 4–11. sorok **repeat-until** ciklusában minden még nem használt funkcionális függőséget egyszer ellenőrzünk, és a ciklust legfeljebb $(|R \setminus X| + 1)$-szer hajtjuk végre, mert csak akkor kezdjük újra, ha $X^{(i-1)} \neq X^{(i)}$, azaz újabb attribútumot vettünk be a lezárásba. Azonban ez a lépésszám megfelelő könyveléssel lineárisra redukálható.

1. Az F minden, még nem használt $W \to Z$ funkcionális függőségéhez feljegyezzük, hogy a W attribútumai közül hány nincs benne még a lezárásban ($i[W, Z]$).

2. Minden A attribútumhoz egy duplán láncolt L_A listában tároljuk azon – még nem használt – funkcionális függőségeket, melyeknek bal oldalán szerepel az A.

3. Egy J láncolt listában tároljuk azokat a – még nem használt – $W \to Z$ funkcionális függőségeket, amelyekre W minden attribútuma már benne van a lezárásban, azaz melyekre $i[W, Z] = 0$.

Feltesszük, hogy F függőség család (W, Z) attribútum párok egy listájaként adott. A LINEÁRIS-LEZÁRÁS eljárás a LEZÁRÁS módosítása a fenti könyvelésekkel, melynek futási ideje lineáris. R a séma, F a rajta értelmezett funkcionális függőségek halmaza, az X attribútumhalmaz F szerinti lezárását keressük.

LINEÁRIS-LEZÁRÁS(R,F,X)

```
 1  ▷ Kezdeti értékek beállítása.
 2  for minden (W,Z) ∈ F
 3      do for minden A ∈ W
 4             do adjuk (W,Z) az L_A listához
 5      i[W,Z] ← 0
 6  for minden A ∈ R \ X
 7      do for minden (W,Z)-re az L_A listán
 8             do i[W,Z] ← i[W,Z] + 1
 9  for minden (W,Z) ∈ F
10      do if i[W,Z] = 0
11           then adjuk (W,Z)-t a J listához
12  X+ ← X
13  ▷ Kezdeti értékek beállításának vége.
14  while J nem üres
15        do (U,W) ← fej(J)
16           töröljük (U,W)-t a J listából
17           for minden A ∈ Z \ X+
18               do for minden (W,Z)-re az L_A listán
19                      do i[W,Z] ← i[W,Z] − 1
20                         if i[W,Z] = 0
21                            then adjuk (W,Z)-t a J listához
22                                 vegyük ki (W,Z)-t az L_A listából
23        X+ ← X+ ∪ Z
24  return X+
```

A LINEÁRIS-LEZÁRÁS(R,F,X) eljárás két részből áll. A kezdeti értékek beállítása (1–13. sorok) részben feltöltjük a listákat kiindulási értékekkel. A 2–5. sorok ciklusai, valamint az 6–8. sorok ciklusai $O(\sum_{(W,Z)\in F} |W|)$ lépést végeznek. A 9–11. sorok ciklusa $O(|F|)$ műveletet jelent. Ha n-nel jelöljük a bemenet hosszát, akkor ez $O(n)$ lépés összesen.

A 14–23. sorokban minden (W,Z) funkcionális függőséget legfeljebb egyszer vizsgálunk meg, amikor a J listából vesszük ki. Tehát a 15–16., illetve 23. sorok legfeljebb $|F|$ lépést jelentenek. A 17–22. sorok ciklusai lépésszámát pedig azzal becsülhetjük meg, hogy minden egyes végrehajtáskor a $\sum i[W,Z]$ eggyel csökken, tehát legfeljebb $O(\sum i_0[W,Z])$ műveletet jelent, ahol $i_0[W,Z]$ az inicializálási szakaszban kapott $i[W,Z]$ érték. Azonban $\sum i_0[W,Z] \leq \sum_{(W,Z)\in F} |W|$, tehát a 14–23. sorok költsége is $O(n)$.

12.2.3. Minimális fedés

A LINEÁRIS-LEZÁRÁS eljárás jól használható funkcionális függőségi rendszerek ekvivalenciájának tesztelésére is. Legyen F és G funkcionális függőségek két családja. Azt mondjuk, hogy F és G ***ekvivalensek,*** ha pontosan ugyanazok a funkcionális függőségek következnek mindkettőből, azaz $F^+ = G^+$. Világos, hogy elegendő eldönteni, hogy igaz-e minden F-beli $X \to Y$ funkcionális függőségre, hogy G^+-hoz tartozik, és fordítva, minden G-beli $W \to Z$-re igaz-e, hogy F^+-ban van. Ugyanis, ha ezek közül valamelyik nem teljesül, mondjuk $X \to Y$ nincs G^+-ban, akkor biztos, hogy $F^+ \neq G^+$. Ha viszont minden $X \to Y \in F$ G^+-

ban van, akkor egy $U \to V$ F^+-beli funkcionális függőség bizonyítását úgy kapjuk meg G-ből, hogy F-beli $X \to Y$ funkcionális függőségek G-ből való bizonyításához hozzátesszük $U \to V$ F-ból történő bizonyítását. Annak eldöntéséhez pedig, hogy F-beli $X \to Y$ G^+-ban van-e, elegendő ellenőriznünk, hogy az X attribútumhalmaz G szerinti $X^+(G)$ lezártja tartalmazza-e Y-t. Egy speciális F-fel ekvivalens funkcionális függőség rendszer hasznos lesz a későbbiekben.

12.5. definíció. *A G funkcionális függőségi halmaz az F funkcionális függőségi halmaz* ***minimális fedése,*** *ha G ekvivalens F-fel, továbbá*

1. *A G-beli funkcionális függőségek $X \to A$ alakúak, ahol A egy attribútum, és $A \notin X$,*
2. *G-ből nem hagyható el funkcionális függőség: $(G - \{X \to A\})^+ \subsetneq G^+$,*
3. *G-beli funkcionális függőségek bal oldalai minimálisak: $X \to A \in G$-re $X \to A \in G$, $Y \subsetneq X \Longrightarrow ((G - \{X \to A\}) \cup \{Y \to A\})^+ \neq G^+$.*

Minden F funkcionális függőségi halmaznak van minimális fedése, a MINIMÁLIS-FEDÉS eljárás létrehoz egyet.

MINIMÁLIS-FEDÉS(R, F)

```
1  G ← ∅
2  for minden X → Y ∈ F
3      do for minden A ∈ Y − X
4             do G ← G ∪ X → A
5                                          ▷ Most már minden jobb oldal egyelemű.
6  for minden X → A ∈ G
7      do while van B ∈ X: A ∈ (X − B)⁺(G)
8             X ← X − B
9                                          ▷ Minden bal oldal minimális.
10 for minden X → A ∈ G
11     do if A ∈ X⁺(G − {X → A})
12        then G ← G − {X → A}
13                                         ▷ Nincs több elhagyható funkcionális függőség.
```

A 2–4. sorok végrehajtása után minden G-beli funkcionális függőség jobb oldala egyelemű. A $G^+ = F^+$ egyenlőség a 12.2. lemma egyesítés szabályából, valamint a reflexivitás axiómából következik. A 6–8. sorokban a bal oldalakat minimalizáljuk. Adott G-beli funkcionális függőségre a 11. sorban ellenőrizzük, hogy elhagyható-e. $X^+(G - \{X \to A\})$ az X halmaz $G - \{X \to A\}$ függőségi halmaz szerinti lezártját jelenti.

12.6. állítás. MINIMÁLIS-FEDÉS *valóban az F egy minimális fedését számítja ki.*

Bizonyítás. Elegendő belátnunk, hogy a 10–12. sorok végrehajtása során nem keletkezhet olyan $X \to A$ funkcionális függőség, melynek bal oldala csökkenthető lenne. Ha létezne olyan $X \to A$, hogy $Y \subsetneq X$-re $Y \to A \in G^+$ lenne, akkor $Y \to A \in G'^+$ is teljesülne, ahol G' a 6–8. sorok végrehajtásnál az $X \to A$ ellenőrzésekor éppen aktuális függőségi halmaz. $G \subseteq G'$, azaz $G^+ \subseteq G'^+$ (lásd a 12.2-1. gyakorlatot). Tehát X-et már a 6–8. sorok végrehajtásakor csökkentenünk kellett volna. ■

12.2.4. Kulcsok

Az adatbázis tervezésekor igen fontos megtalálni azon attribútumhalmazokat, melyek egyértelműen meghatározzák az egyes rekordok adatait.

12.7. definíció. *Legyen (R, F) egy relációs séma, az $X \subseteq R$ attribútumhalmaz* ***szuperkulcs,*** *ha $X \to R \in F^+$. Egy X szuperkulcsot* ***kulcsnak*** *nevezünk, ha tartalmazásra nézve minimális, azaz egyetlen valódi $Y \subsetneq X$ részhalmaza sem szuperkulcs.*

Kérdés az, hogy (R, F) ismeretében hogyan határozhatók meg a kulcsok? A feladat nehézségét az adja, hogy a kulcsok száma (R, F) méretének szuper exponenciális függvénye is lehet.

Pontosabban, Yu és Johnson konstruáltak olyan relációs sémát, melyben $|F| = k$, és a kulcsok száma $k!$. Békéssy és Demetrovics egyszerű és szép bizonyítást adtak arra, hogy k funkcionális függőségből kiindulva legfeljebb $k!$ kulcs kapható. Békéssy és Demetrovics bizonyításának alapja az általuk bevezetett $*$ művelet, ami funkcionális függőségeken van értelmezve.

12.8. definíció. *Legyen $e_1 = U \to V$ és $e_2 = X \to Y$ két funkcionális függőség. A $*$ kétváltozós műveletet az*

$$e_1 * e_2 = U \cup ((R - V) \cap X) \to V \cup Y$$

képlet definiálja.

Felsoroljuk a $*$ művelet néhány tulajdonságát, melyeknek bizonyítását az Olvasóra bízzuk gyakorlatként (12.2-5. gyakorlat). A $*$ művelet asszociatív, valamint idempotens abban az értelemben, hogy ha $e = e_1 * e_2 * \cdots * e_k$ és $e' = e * e_i$ valamely $1 \leq i \leq k$-ra, akkor $e' = e$.

12.9. állítás (Békéssy és Demetrovics). *Legyen (R, F) relációs séma, $F = \{e_1, e_2, \ldots, e_k\}$ a funkcionális függőségek felsorolása. Ha X kulcs, akkor $X \to R = e_{\pi_1} * e_{\pi_2} * \cdots * e_{\pi_s} * d$, ahol $(\pi_1, \pi_2, \ldots, \pi_s)$ az $\{1, 2, \ldots, k\}$ indexhalmaz valamely rendezett részhalmaza, és d egy $D \to D$ alakú triviális függőség.*

Ez az állítás valamennyire behatárolja a kulcsok keresésénél szóba jövő attribútumhalmazokat. A következő állítás egy alsó és egy felső korlátot ad a kulcshalmazokra.

12.10. állítás. *Legyen (R, F) egy relációs séma, $F = \{U_i \to V_i \colon 1 \leq i \leq k\}$. Tegyük fel az általánosság megszorítása nélkül, hogy $U_i \cap V_i = \emptyset$. Legyen $\mathcal{U} = \bigcup_{i=1}^{k} U_i$ és $\mathcal{V} = \bigcup_{i=1}^{k} V_i$. Ha K kulcs az (R, F) sémában, akkor*

$$\mathcal{H}_A = R - \mathcal{V} \subseteq K \subseteq (R - \mathcal{V}) \cup \mathcal{U} = \mathcal{H}_F \ .$$

A bizonyítás nem túlságosan bonyolult, gyakorlatként az Olvasóra bízzuk (12.2-6. gyakorlat). Ebből a becslésből kiindulva adhatjuk a KULCS-KERESŐ algoritmust az (R, F) relációs séma kulcsainak listázására. Az algoritmus lépésszámát $O(n!)$-sal becsülhetjük, de ennél gyorsabbat nem is várhatunk, hiszen pusztán az eredmény kiírásához kell ennyi idő legrosszabb esetben.

KULCS-KERESŐ(R, F)

```
1  ▷ Legyenek 𝒰 és 𝒱 a 12.10. állításban definiált halmazok.
2  if 𝒰 ∩ 𝒱 = ∅
3     then return R − 𝒱
4                                          ▷ R − 𝒱 az egyetlen kulcs.
5  if (R − 𝒱)⁺ = R
6     then return R − 𝒱
7                                          ▷ R − 𝒱 az egyetlen kulcs.
8  for az 𝒰 ∩ 𝒱 attribútumainak minden A₁, A₂, ... Aₕ sorrendjére
9     do K ← (R − 𝒱) ∪ 𝒰
10       for i ← 1 to h
11          do Z ← K − Aᵢ
12             if Z⁺ = R
13                then K ← Z
14       return K
```

Gyakorlatok

12.2-1. Legyen R egy relációs séma, F és G funkcionális függőségi halmazok R felett. Bizonyítsuk be, hogy

a. $F \subseteq F^+$.

b. $(F^+)^+ = F^+$.

c. Ha $F \subseteq G$, akkor $F^+ \subseteq G^+$.

Fogalmazzuk meg és bizonyítsuk az X attribútumhalmaz F szerinti X^+ lezárására érvényes hasonló állításokat.

12.2-2.

12.2-3. Bizonyítsuk be a 12.2. lemma egyesítési, pszeudotranzitivitási és szétvágási szabályait.

12.2-4. Vezessük le az $AB \to F$ funkcionális függőséget a $G = \{AB \to C, A \to D, CD \to EF\}$ függőségi halmazból, az (A1)–(A3) Armstrong axiómák segítségével.

12.2-5. Bizonyítsuk be, hogy a $*$ művelet asszociatív, és ha az $e_1, e_2, \ldots, e_k$ funkcionális függőségekre $e = e_1 * e_2 * \cdots * e_k$ és $e' = e * e_i$ valamely $1 \le i \le k$-ra, akkor $e' = e$.

12.2-6. Bizonyítsuk be a 12.10. állítást.

12.3. Relációs sémák szétvágása

Egy $R = \{A_1, A_2, \ldots, A_n\}$ relációs séma ***szétvágásán*** R részhalmazainak egy olyan $\rho = \{R_1, R_2, \ldots, R_k\}$ családját értjük, amelyikre teljesül, hogy

$$R = R_1 \cup R_2 \cup \cdots \cup R_k.$$

Nem szükséges, hogy az R_i-k diszjunktak legyenek. A szétvágás egyik motivációja a különböző ***anomáliák*** elkerülése.

12.3. példa. Tekintsük a következő sémát:

SZÁLLÍTÓ-INFO(SNÉV,SCÍM,ÁRU,ÁR) .

Ez a séma az alábbi gondokat okozhatja:

1. *Redundancia.* Egy szállító címe minden olyan árunál fel van sorolva, amit szállít.
2. *Inkonzisztencia lehetősége (frissítési anomália).* A redundanciának köszönhetően előfordulhat, hogy egy szállító címét felfrissítjük valamelyik sorban, de marad a régi valamelyik másik sorban. Így nem lenne a szállítónak egyértelmű címe, amit pedig elvárnánk.
3. *Beillesztési anomália.* Nem tudjuk egy szállító címét feljegyezni, ha pillanatnyilag éppen nem szállít semmit. Megpróbálhatnánk ugyan az ÁRU és az ÁR mezőkbe NULL értéket írni, de vajon emlékszünk-e majd arra, hogy a NULL értékeket töröljük, amikor a szállítóhoz feljegyzünk egy szállított terméket? Súlyosabb baj, hogy SNÉV és ÁRU együtt a séma egy kulcsát alkotja, így a NULL értékek megakadályozhatnák a kulcs szerinti index alapján történő keresést.
4. *Törlési anomália.* Az előző ellentettje. Ha egy adott szállító összes áruját töröljük, akkor mellékhatásként a szállító címét is elveszítjük.

A fenti problémák mind megszűnnek, ha a SZÁLLÍTÓ-INFO sémát két másikkal helyettesítjük:

SZÁLLÍTÓK(SNÉV,SCÍM)
SZÁLLÍT(SNÉV,ÁRU,ÁR)

Ekkor minden szállító címét csak egyszer jegyezzük fel, és a cím tárolásához nem szükséges, hogy a szállítónak legyen éppen szállított áruja. Az egyszerűség kedvéért jelöljük az attribútumokat egy-egy betűvel: S (SNÉV), C (SCÍM), U (ÁRU), R (ÁR).

Kérdés, hogy amikor a $SCUR$ sémát az SC és az SUR sémákkal helyettesítjük, akkor nem követünk-e el hibát? Tegyük fel, hogy r az $SCUR$ séma egy példánya. Természetes követelmény, hogy ha az SC és az SUR sémát használjuk, akkor az ezekhez tartozó relációkat az r SC-re, illetve SUR-ra való vetítésével kapjuk, azaz $r_{SC} = \pi_{SC}(r)$ és $r_{SUR} = \pi_{SUR}(r)$. r_{SC} és r_{SUR} ugyanazt az információt tartalmazza, mint r, ha r kiszámítható csak r_{SC} és r_{SUR} használatával. A kiszámítást a ***természetes összekapcsolás*** operációval végezhetjük.

12.11. definíció. *Az R_i relációs sémákhoz tartozó r_i relációk ($i = 1, 2, \ldots, n$)* ***természetes összekapcsolása*** *az $\cup_{i=1}^{n} R_i$ sémához tartozó s reláció, mely mindazon μ sorokból áll, melyekre igaz, hogy minden i-re van az r_i relációnak olyan ν_i sora, hogy $\mu[R_i] = \nu_i[R_i]$. Jelölésben $s = \bowtie_{i=1}^{n} r_i$.*

12.4. példa. Legyen $R_1 = AB$, $R_2 = BC$; $r_1 = \{ab, a'b', ab''\}$ és $r_2\{bc, bc', b'c''\}$. Ekkor az r_1 és r_2 természetes összekapcsolása az $R = ABC$ sémához tartozik, értéke pedig: $r_1 \bowtie r_2 = \{abc, abc', a'b'c''\}$.

Ha s az r_{SC} és r_{SUR} természetes összekapcsolása, azaz $s = r_{SC} \bowtie r_{SUR}$, akkor a 12.12. lemma szerint $\pi_{SC}(s) = r_{SC}$ és $\pi_{SUR}(s) = r_{SUR}$. Ha $r \neq s$, akkor pusztán r_{SC} és r_{SUR} ismeretében nem tudnánk visszanyerni az eredeti relációt.

12.3.1. Veszteségmentes összekapcsolás

Legyen az R séma szétvágása $\rho = \{R_1, R_2, \ldots, R_k\}$, valamint F funkcionális függőségek egy családja. Azt mondjuk, hogy ez a szétvágás ***veszteség mentes összekapcsolású*** (F-re nézve), ha R minden olyan r reláció példánya, melyben F igaz, teljesíti, hogy

$$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \cdots \bowtie \pi_{R_k}(r) \ ,$$

azaz az r reláció megegyezik az R_i attribútum halmazokra való vetítettjeinek természetes összekapcsolásával. A $\rho = \{R_1, R_2, \ldots, R_k\}$ szétvágásra jelölje m_ρ azt a leképezést, ami az r relációhoz az $m_\rho(r) = \bowtie_{i=1}^{k} \pi_{R_i}(r)$ relációt rendeli. Tehát a veszteség mentes összekapcsolás feltétel egy adott F funkcionális függőség családra nézve azt jelenti, hogy $r = m_\rho(r)$ minden olyan r-re, amelyik kielégíti F-t.

12.12. lemma. *Legyen R relációs séma, $\rho = \{R_1, R_2, \ldots, R_k\}$ egy szétvágása, és legyen r tetszőleges, az R sémához tartozó reláció. Legyen továbbá $r_i = \pi_{R_i}(r)$. Ekkor*

1. $r \subseteq m_\rho(r)$.

2. *Ha $s = m_\rho(r)$, akkor $\pi_{R_i}(s) = r_i$.*

3. $m_\rho(m_\rho(r)) = m_\rho(r)$.

A lemma bizonyítását az Olvasóra bízzuk (12.3-7. gyakorlat).

12.3.2. Veszteségmentes összekapcsolás ellenőrzése

Viszonylag egyszerűen ellenőrizhetjük, hogy az R séma $\rho = \{R_1, R_2, \ldots, R_k\}$ szétvágása rendelkezik-e a veszteségmentes összekapcsolás tulajdonságával.

A KAPCSOLÁS-TESZT(R,F,ρ) algoritmus lényege a következő. Létrehozunk egy $k \times n$-es T tömböt, melynek j-ik oszlopa az A_j attribútumnak felel meg, i-ik sora pedig az R_i relációnak. Ha $A_j \in R_i$, akkor $T[i, j] = 0$, egyébként pedig $T[i, j] = i$.

A következő lépést ismételjük, amíg már nem lehetséges több változtatás a tömbben. Tekintünk egy $X \to Y$ funkcionális függőséget F-ből. Megnézzük, hogy mely sorpárok egyeznek meg X minden attribútumában. Ha például az i és j sorok ilyenek, akkor attribútumaikat minden Y beli oszlopban egyenlővé tesszük. Pontosabban, ha valamely oszlopban az egyik ilyen attribútum 0, akkor a másikat is 0-ra változtatjuk. Ha az egyik attribútum i, a másik j, akkor tetszés szerint változtathatjuk mindkettőt i-re vagy j-re. Ha egy szimbólumot megváltoztatunk, akkor annak oszlopában *minden* előfordulását meg kell változtatni. Ha az eljárás végén található T-ben egy csupa 0 sor, akkor a összekapcsolás veszteségmentes, ha nincs ilyen sor, akkor veszteséges.

KAPCSOLÁS-TESZT(R,F,ρ)

```
 1                                                  ▷ Kezdeti értékek beállítása.
 2  for i ← 1 to |ρ|
 3      do for j ← 1 to |R|
 4             do if A_j ∈ R_i
 5                   then T[i, j] ← 0
 6                   else T[i, j] ← i
 7                                                  ▷ Kezdeti értékek beállításának vége.
 8  S ← T
 9  repeat
10          T ← S
11          for minden {X → Y} ∈ F
12              do for i ← 1 to |ρ| − 1
13                     do for j ← i + 1 to |R|
14                            do if minden X-beli A_h-ra S[i, h] = S[j, h]
15                                  then EGYENLŐSÍT(i, j, S, Y)
16  until S = T
17  if van S-ben csupa 0 sor
18    then return „Veszteségmentes”
19    else return „Veszteséges”
```

Az EGYENLŐSÍT(i, j, S, Y) eljárás végzi el a szimbólumok megfelelő egyenlővé tételét.

EGYENLŐSÍT(i, j, S, Y)

```
1  for A_l ∈ Y
2      do if S[i, l] < S[j, l]
3            then u ← S[i, l]
4                 v ← S[j, l]
5            else u ← S[j, l]
6                 v ← S[i, l]
7          for d ← 1 to k
8              do if S[d, l] = v
9                    then S[d, l] ← u
```

12.5. példa. Legyen $R = ABCDE$, $R_1 = AD$, $R_2 = AB$, $R_3 = BE$, $R_4 = CDE$, $R_5 = AE$, a funkcionális függőségek a következők: $\{A \to C, B \to C, C \to D, DE \to C, CE \to A\}$. A kiindulási tömb a 12.1(a) ábrán látható. $A \to C$-t használva a C oszlop 1,2,5 értékeit egyenlősíthetjük 1-re. Ezután $B \to C$-t alkalmazva ugyancsak a C oszlop 3 értékét lehet 1-re változtatni. Az eredmény a 12.1(b) ábrán látható. Most $C \to D$-t használjuk, hogy a D oszlop 2,3,5 értékeit 0-ra változtassuk. Ezek után $DE \to C$ alkalmazásával a C oszlop (egyetlen nem 0) 1 értékét 0-vá tesszük. Végül $CE \to A$ lehetővé teszi, hogy az A oszlopban álló 3, 4-t 0-ra cseréljük: a végeredményt a 12.1(c) ábra mutatja. A harmadik sor csupa 0, így a szétvágás veszteségmentes összekapcsolású.

Világos, hogy a KAPCSOLÁS-TESZT eljárás lépésszáma a bemenet hosszának polinomja. A

A	B	C	D	E
0	1	1	0	1
0	0	2	2	2
3	0	3	3	0
4	4	0	0	0
0	5	5	5	0

(a)

A	B	C	D	E
0	1	1	0	1
0	0	1	2	2
3	0	1	3	0
4	4	0	0	0
0	5	1	5	0

(b)

A	B	C	D	E
0	1	0	0	1
0	0	0	2	2
0	0	0	0	0
0	4	0	0	0
0	5	0	0	0

(c)

12.1. ábra. KAPCSOLÁS-TESZT(R, F, ρ) alkalmazása. **(a)** Kiindulási tömb. **(b)** $A \leftarrow B$ és $B \leftarrow C$ alkalmazásának eredménye. **(c)** A végeredmény.

lényeg, hogy ez az algoritmus csupán a sémát használja, továbbá a funkcionális függőségek halmazát, és nem a ténylegesen tárolt, a sémához tartozó r relációt. Mivel az adatbázis mérete nagyságrendekkel nagyobb, mint a séma mérete, ezért azon algoritmusok futási ideje, amelyek csak a sémát használják, eltörpül az adatokat feldolgozó algoritmusoké mellett.

12.13. tétel. *A* KAPCSOLÁS-TESZT *eljárás helyesen határozza meg, hogy egy adott szétvágás veszteségmentes összekapcsolású-e.*

Bizonyítás. Tegyük fel először, hogy az eredményül kapott T táblázatban nincs csupa 0 sor. Természetesen T maga is tekinthető az R séma feletti relációnak. Ez a reláció minden F-beli funkcionális függőséget kielégít, mivel az algoritmus futása azért fejeződött be, mert a funkcionális függőségek végignézése során már nem történt változás a táblában. A kiindulási táblára igaz, hogy minden R_i-re vonatkozó vetülete tartalmaz egy csupa 0 sort, és ez a tulajdonság az algoritmus futása során nem változik, hiszen egy 0-t soha nem változtatunk nem 0-vá. Ez azt jelenti, hogy az $m_\rho(T)$ összekapcsolás tartalmazza a csupa 0 sort, vagyis $T \neq m_\rho(T)$, azaz a szétvágás veszteséges. A másik irány bizonyítását csak vázoljuk.

Logikát, tartomány kalkulust használunk. A szükséges definíciók megtalálhatók Abiteboul, Hull és Vianu, illetve Ullman könyvében. Képzeljük el, hogy a j-edik oszlopba a 0 helyett a_j változót, i helyett b_{ij} változót írunk, és úgy hajtjuk végre a KAPCSOLÁS-TESZT eljá-

rást, és az eredmény táblában található a csupa 0 sornak megfelelő $a_1 a_2 \dots a_n$ sor. Minden tábla felfogható egyszerűbb jelölésként a következő tartomány kalkulus kifejezésre

$$\{a_1 a_2 \dots a_n \mid (\exists b_{11}) \dots (\exists b_{kn}) \, (R(w_1) \wedge \dots \wedge R(w_k))\} \ , \tag{12.1}$$

ahol w_i a T i-edik sora. Ha T a kiindulási tábla, akkor a (12.1) képlet éppen az m_ρ függvényt definiálja. Indoklásul jegyezzük meg, hogy egy r relációra $m_\rho(r)$ pontosan akkor tartalmazza az $a_1 a_2 \dots a_n$ sort, ha minden i-re r tartalmaz egy sort, melynek a_j a j-edik koordinátája amennyiben A_j az R_i egy attribútuma, és tetszőleges értékeket, melyeket a b_{il} változók ábrázolnak, a többi attribútumban.

Tekintsünk egy tetszőleges, az R sémához tartozó r relációt, amely kielégíti az F funkcionális függőségeket. Azok a változtatások (szimbólumok egyenlővé tétele), amelyeket a KAPCSOLÁS-TESZT eljárás hajt végre a táblán, nem változtatják meg a (12.1) kifejezés által az r-ből eredményül adott sorok halmazát, amennyiben a változtatásokat a formulában is megfelelően végrehajtjuk. Intuitívan ez abból látható, hogy csak olyan szimbólumokat teszünk egyenlővé (12.1)-ben, amelyek olyan relációban vannak, ami az F-beli funkcionális függőségeket kielégíti, tehát amúgy is csak egyenlő értékeket vehetnének fel. A pontos bizonyítás hosszadalmas, így elhagyjuk.

Mivel a KAPCSOLÁS-TESZT eljárás eredmény táblájában szerepel a csupa a sor, ezért az ehhez a táblához tartozó tartomány kalkulus formula a következő alakú:

$$\{a_1 a_2 \dots a_n \mid (\exists b_{11}) \dots (\exists b_{kn}) \, (R(a_1 a_2 \dots a_n) \wedge \dots)\} \ . \tag{12.2}$$

Világos, hogy ha (12.2)-t alkalmazzuk egy R sémához tartozó r relációra, akkor az eredmény az r egy részhalmaza lesz. Azonban, ha r kielégíti az F funkcionális függőségeket, akkor a (12.2) formula $m_\rho(r)$-t számolja ki. A 12.12. lemma 1. része szerint $r \subseteq m_\rho(r)$ teljesül, így ha r kielégíti F-t, akkor a (12.2) pont r-et adja eredményül, tehát $r = m_\rho(r)$, azaz a szétvágás veszteségmentes összekapcsolású. ■

A KAPCSOLÁS-TESZT eljárás használható minden esetben, függetlenül a szétvágásban szereplő részek számától. Ennek az ára a futási idő igényben realizálódik. Azonban, ha csak ***két*** részre akarjuk vágni az R sémát, akkor LEZÁRÁS-t, illetve a LINEÁRIS-LEZÁRÁS-t használva hamarabb célhoz érünk az alábbi tétel alapján.

12.14. tétel. *Legyen $\rho = (R_1, R_2)$ az R reláció szétvágása, továbbá F funkcionális függőségek egy halmaza. A ρ szétvágás akkor és csak akkor veszteségmentes összekapcsolású F-re vonatkozóan, ha*

$$(R_1 \cap R_2) \to (R_1 - R_2) \ \textit{vagy} \ (R_1 \cap R_2) \to (R_2 - R_1) \ .$$

Ezeknek a funkcionális függőségeknek nem kell F-ben lenniük, elegendő, ha F^+-ban vannak.

Bizonyítás. A KAPCSOLÁS-TESZT eljárásban szereplő kiindulási tábla a következő:

	$R_1 \cap R_2$	$R_1 - R_2$	$R_2 - R_1$
R_1 sora	$00\dots0$	$00\dots0$	$11\dots1$
R_2 sora	$00\dots0$	$22\dots2$	$00\dots0$

(12.3)

Indukciót használva a KAPCSOLÁS-TESZT(R,F,ρ) eljárás lépésszámára nem nehéz belátni,

hogy ha az A attribútumra az algoritmus mindkét A oszlopbeli értéket 0-ra változtatja, akkor $A \in (R_1 \cap R_2)^+$. Ez nyilván igaz a kiindulási helyzetben. Ha valamikor az A oszlopban egyenlősíteni kell, az azt jelenti az algoritmus 11–14. sorai szerint, hogy van $\{X \to Y\} \in F$, hogy a tábla két sora X-en megegyezik, és $A \in Y$. Az indukciós feltevés szerint ekkor $X \subseteq (R_1 \cap R_2)^+$, tehát az Armstrong axiómák alapján $A \in (R_1 \cap R_2)^+$ következik.

Másrészről, tegyük fel, hogy $A \in (R_1 \cap R_2)^+$, azaz $(R_1 \cap R_2) \to A$. Ekkor ez a funkcionális függőség F-ből az Armstrong axiómák használatával le is vezethető. Ezen levezetés hosszára vonatkozó indukcióval könnyű látni az előzőekhez hasonlóan, hogy a KAPCSOLÁSTESZT eljárás az A oszlopbeli értékeket egyenlővé, azaz 0-vá fogja tenni. Tehát az R_1 sora pontosan akkor lesz csupa 0, ha $(R_1 \cap R_2) \to (R_2 - R_1)$, és ugyanúgy, az R_2 sora pontosan akkor lesz csupa 0, ha $(R_1 \cap R_2) \to (R_1 - R_2)$. ■

12.3.3. Funkcionális függőségeket megőrző szétbontások

A veszteségmentes összekapcsolás tulajdonság fontos, hogy egy relációt a vetületeiből vissza tudjunk nyerni pontosan. A gyakorlatban rendszeresen nem az R alapsémához tartozó r relációt tároljuk állandóan, hanem egy alkalmas $\rho = (R_1, R_2, \ldots, R_k)$ szétvágáshoz tartozó $r_i = r[R_i]$ relációkat, elkerülendő a különböző anomáliákat. Az R sémához tartozó F funkcionális függőségek az adatbázis integritási feltételeit jelentik, az r reláció konzisztens, ha teljesíti az összes előírt funkcionális függőséget. Ha az adatbázis élete során frissítéseket hajtunk végre, azaz sorokat illesztünk be az egyes vetület relációkba, illetve törlünk, akkor előfordulhat, hogy az új vetületek természetes összekapcsolása már nem elégíti ki F funkcionális függőségeit. Ha minden egyes frissítés után ellenőrzésképpen össze kellene kapcsolnunk a vetületi relációkat, majd újra szétbontanunk, az túlságosan költséges lenne. Definiálhatjuk azonban az F funkcionális függőség család ***vetületét*** a Z attribútumhalmazra: $\pi_Z(F)$ azon $\{X \to Y\} \in F^+$ funkcionális függőségekből áll, amelyekre $XY \subseteq Z$. Frissítéskor ha az r_i relációt változtattuk, akkor a $\pi_{R_i}(F)$ teljesülését könnyen ellenőrizhetjük. Tehát az lenne jó, ha a $\pi_{R_i}(F)$ $i = 1, 2, \ldots, k$ funkcionális függőség családokból F logikailag következne. Legyen $\pi_\rho(F) = \bigcup_{i=1}^{k} \pi_{R_i}(F)$.

12.15. definíció. *Azt mondjuk, hogy a ρ szétvágás **függőségőrző,** ha*

$$\pi_\rho(F)^+ = F^+ \ .$$

Megjegyezzük, hogy $\pi_\rho(F) \subseteq F^+$, következésképpen $\pi_\rho(F)^+ \subseteq F^+$ mindig teljesül. Tekintsük a következő példát.

12.6. példa. Legyen R = (**V**áros,**U**tca,**I**rányítószám) az alap séma, $F = \{VU \to I, I \to V\}$ a funkcionális függőségek. Legyen a ρ felbontás $\rho = (VI, UI)$. Ez az $I \to V$ funkcionális függőség miatt a 12.14. tétel alapján veszteségmentes összekapcsolású. A $\pi_\rho(F)$ a triviális függőségeken kívül az $I \to V$ funkcionális függőséget tartalmazza. Legyen $R_1 = VI$ és $R_2 = UI$. Beszúrunk az R_1 és R_2 sémákhoz tartozó vetületekbe két-két sort úgy, hogy a vetület funkcionális függőségei teljesüljenek, az alábbiak szerint:

R_1	V	I
	Nagykanizsa	8800
	Nagykanizsa	8831

R_2	U	I
	Kossuth	8800
	Kossuth	8831

Ekkor R_1 és R_2 külön-külön kielégítik a rájuk kirótt funkcionális függőségeket, az $R_1 \bowtie R_2$-ben viszont nem teljesül a $VU \to I$ funkcionális függőség.

Igaz az is, hogy ennek a sémának semelyik valódi szétvágása sem őrzi meg a $VU \to I$ funkcionális függőséget. Ugyanis ez az egyetlen olyan funkcionális függőség, melynek a jobb oldalán szerepel az I attribútum, tehát ha meg akarnánk őrizni, akkor kellene a részsémák közt olyannak szerepelnie, ami tartalmazza V-t, U-t és I-t, de az már nem lenne valódi szétvágás. Erre majd még a normálformákra bontáskor visszatérünk.

Megjegyezzük, hogy az az eset is előfordulhat, hogy egy ρ szétvágás megőrzi a funkcionális függőségeket, de nem veszteségmentes összekapcsolású. Legyen ugyanis $R = ABCD$, $F = \{A \to B, C \to D\}$, valamint $\rho = (AB, CD)$.

Elviekben roppant egyszerű ellenőrizni, hogy egy $\rho = (R_1, R_2, \ldots, R_k)$ szétvágás függőségőrző-e az F funkcionális függőségekre nézve. Csak F^+-t kell kiszámolni, majd venni a vetületeiket, végül ellenőrizni, hogy a vetületek uniója ekvivalens-e F-fel. A gond ezzel az, hogy már az F^+ kiszámítása is exponenciálisan sok lépést igényelhet.

Meg lehet azonban oldani a feladatot F^+ tényleges meghatározása nélkül. Legyen $G = \pi_\rho(F)$. G-t nem számítjuk ki, csak azt ellenőrizzük, hogy ekvivalens-e F-el. Ehhez minden $\{X \to Y\} \in F$ funkcionális függőségre el kell tudnunk dönteni, hogy ha X^+-t G-re vonatkozólag vesszük, akkor az tartalmazza-e Y-t. A trükk az, hogy X^+-t G teljes ismerete nélkül határozzuk meg úgy, hogy ismételten vesszük az F egyes R_i-kre való vetületének a hatását a lezárásra. Azaz, bevezetjük az F-re vonatkozó S-operáció fogalmát egy Z attribútumhalmazon, ahol S egy attribútum halmaz: Z-t kicseréljük $Z \cup ((Z \cap S)^+ \cap S)$-re, ahol a lezárás F szerint veendő. Azaz, Z-nek az S-be eső részét lezárjuk F szerint, majd az így kapott attribútumok közül az S-be esőket hozzáadjuk Z-hez.

MEGŐRIZ(ρ, F)

```
 1 for minden (X → Y) ∈ F
 2     do Z ← X
 3        repeat
 4                W ← Z
 5                for i ← 1 to k
 6                    do Z ← Z ∪ (LINEÁRIS-LEZÁRÁS(R, F, Z ∩ R_i) ∩ R_i)
 7        until Z = W
 8     if Y ⊈ Z
 9        then return „Nem őrzi meg”
10 return „Megőrzi”
```

Világos, hogy a MEGŐRIZ eljárás lépésszáma polinomiális a bemenet hosszában. Pontosabban, a legkülső **for** ciklust minden F-beli funkcionális függőségre legfeljebb egyszer kell végrehajtani (előfordulhat, hogy már előbb kiderül, hogy valamelyik függőség nem őrződik meg). A 3–7. sorok **repeat–until** ciklus magja lineáris lépésszámú, a ciklust magát legfeljebb $|R|$-szer kell végrehajtani. Tehát legfeljebb kvadratikus sok lépés kell a **for** ciklus minden egyes ismétlésekor, azaz a futási idő becsülhető a bemenet hosszának köbével.

12.7. példa. Legyen a séma $R = ABCD$, a szétvágás $\rho = \{AB, BC, CD\}$, a funkcionális függőségek családja pedig $F\{A \to B, B \to C, C \to D, D \to A\}$. Azaz, a függőségek látható köre alapján, F^+-ban minden attribútum funkcionálisan meghatározza az összes többit. Mivel a szétvágásban D és

A nem szerepel együtt, azt gondolhatnánk, hogy a $D \to A$ függőség nem őrződik meg, azonban ez az intuíciónk hibás. Ennek oka, hogy amikor például AB-re vetítünk, akkor nem csak az $A \to B$ függőséget kapjuk meg, hanem a $B \to A$-t is, mert nem F-t, hanem ténylegesen F^+-t vetítjük. Hasonlóan megkapjuk $C \to B$-t és $D \to C$-t is, ezekből pedig $D \to A$ logikailag következik az Armstrong-axiómák alapján, használva a tranzitivitást. Tehát ténylegesen azt várjuk, hogy MEGŐRIZ azt mondja nekünk, hogy $D \to A$ megőrződik.

Az $Y = \{D\}$ attribútumhalmazból indulunk ki. Három lehetséges operációnk van, az AB-operáció, a BC-operáció és a CD-operáció. Az első kettő nyilván nem ad semmit $\{D\}^+$-hoz, mivel $\{D\} \cap \{A, B\} = \{D\} \cap \{B, C\} = \emptyset$, tehát az üres halmaz lezárását kellene vennünk, ami üres (ebben a példában). Azonban, ha a CD-operációt használjuk, akkor

$$\begin{aligned} Z &= \{D\} \cup ((\{D\} \cap \{C, D\})^+ \cap \{C, D\}) \\ &= \{D\} \cup (\{D\}^+ \cap \{C, D\}) \\ &= \{D\} \cup (\{A, B, C, D\} \cap \{C, D\}) \\ &= \{C, D\} \,. \end{aligned}$$

A következő menetben a BC-operáció használatával kapjuk, hogy az aktuális $Z = \{C, D\}$-ből $Z = \{B, C, D\}$ lesz, majd erre alkalmazva az AB-operációt kapjuk, hogy $Z = \{A, B, C, D\}$. Ez már nem változhat, úgyhogy a MEGŐRIZ(ρ, F) eljárás futása megáll. Tehát a

$$G = \pi_{AB}(F) \cup \pi_{BC}(F) \cup \pi_{CD}(F)$$

funkcionális függőség családra vonatkozólag $\{D\}^+ = \{A, B, C, D\}$, azaz $G \models D \to A$. Hasonlóan ellenőrizhető, hogy F többi funkcionális függősége is G^+-ban (ténylegesen G-ben) van.

12.16. tétel. *A* MEGŐRIZ *eljárás helyesen dönti el, hogy a ρ szétvágás függőségőrző-e.*

Bizonyítás. Elegendő ellenőrizni, hogy egyetlen $X \to Y$ funkcionális függőségre helyesen határozza-e meg, hogy G^+-ban van-e. A 3–7. sorokban amikor egy attribútumot Z-hez veszünk, akkor G-beli funkcionális függőséget használunk, tehát az Armstrong-axiómák ellentmondásmentessége miatt, ha a MEGŐRIZ eljárás azt találja, hogy $X \to Y \in G^+$, akkor az valóban teljesül.

Megfordítva, ha $X \to Y \in G^+$, akkor a LINEÁRIS-LEZÁRÁS eljárás (G-vel, mint bemenettel futtatva) egyenként hozzáveszi X-hez Y attribútumait. Minden egyes olyan lépésben, amikor egy attribútumot hozzávesz, valamilyen G-beli $U \to V$ függőséget használ. Ez a függőség valamelyik $\pi_{R_i}(F)$-ben van, hiszen G ezek uniója. A LINEÁRIS-LEZÁRÁS(R, F, X) eljárásban használt funkcionális függőségek számára vonatkozó indukcióval könnyen belátható, hogy előbb-utóbb U a Z részhalmazává válik, és akkor az R_i-operáció alkalmazásával V összes attribútumát Z-hez adjuk. ■

12.3.4. Normálformák

A relációs sémák ***normálformára*** hozásának célja, hogy a bevezetőben említett anomáliákat elkerüljük. Több különböző normálformát vezettek be az adatbáziselmélet fejlődése folyamán, ezek közül mi csak a ***Boyce–Codd*** normálformát (BCNF) és a ***harmadik,*** illetve ***negyedik*** normálformát (3NF és 4NF) tárgyaljuk részletesebben, minthogy a gyakorlatban ezek a legfontosabbak.

Boyce–Codd normálforma

12.17. definíció. *Legyen R egy séma, F funkcionális függőségek halmaza. Azt mondjuk, hogy (R,F)* ***Boyce–Codd normálformában*** *van, ha $X \to A \in F^+$ és $A \notin X$ esetén X szuperkulcs.*

A BCNF fontos tulajdonsága, hogy megszünteti a redundanciát. Ez a következő tételen alapszik, melynek bizonyítását gyakorlatként az Olvasóra bízzuk (12.3-8. gyakorlat).

12.18. tétel. *Az (R,F) séma pontosan akkor van BCNF-ben, ha tetszőleges $A \in R$-re és $X \subset R$ kulcsra igaz, hogy nincs olyan $Y \subseteq R$, amire $X \to Y \in F^+$, $Y \to X \notin F^+$, $Y \to A \in F^+$ és $A \notin Y$.*

A 12.18. tétel másképp fogalmazva azt mondja ki, hogy egy (R,F) séma pontosan akkor van BCNF-ben, ha nincs benne kulcstól való tranzitív függés. Tegyük fel, hogy egy adott séma nem lenne BCNF-ben, például $C \to B$; $B \to A$ teljesülne, de $B \to C$ nem, akkor ugyanaz a B érték több C érték mellett is állhatna, de minden példánynál ugyanazt az A értéket is tároljuk vele, ami redundáns. Kicsit más megfogalmazásban BCNF azt jelenti, hogy (pusztán) funkcionális függőségeket használva nem tudjuk megjósolni egy attribútum értékét valamely sorban más attribútum értékekből. Tegyük fel ugyanis, hogy van egy R sémánk, amiben valamely attribútum értékét meg tudjuk határozni egy funkcionális függőséget használva két sor összehasonlításával. Azaz van két sorunk, amelyek megegyeznek egy X attribútumhalmazon, különböznek egy Y attribútumhalmazon, és a maradék (egyetlen) A attribútum egyik sorban felvett értékéből a másik sorbeli értékre következtethetünk.

X	Y	A
x	y_1	a
x	y_2	?

Ha a ? értéket egy funkcionális függőséggel meg tudjuk határozni, akkor az az érték csakis a lehet, a függőség pedig $Z \to A$, ahol Z az X egy alkalmas részhalmaza. Viszont Z nem lehet szuperkulcs, mert akkor a két sornak azonosnak kellene lennie, tehát R nincs BCNF-ben.

3NF

Noha a BCNF segít megszüntetni az anomáliákat, nem mindig igaz, hogy egy sémát szét lehet vágni BCNF alsémákra úgy, hogy a szétvágás megőrizze a függőségeket is. Amint azt a 12.6. példában láttuk, a VUI séma egyetlen valódi szétvágása sem őrzi meg a $VU \to I$ funkcionális függőséget. Ugyanakkor a séma nyilván nincs BCNF-ben, az $I \to V$ függőség miatt.

Minthogy a függőség megőrzése az adatbázis konzisztenciájának ellenőrzése miatt fontos, célszerű egy olyan normálformát is bevezetni, melyre igaz, hogy tetszőleges sémát szét lehet vágni sémákra úgy, hogy a függőségek megőrződnek, továbbá lehetőleg minél kevesebb redundanciát engedjen meg. Egy attribútumot ***prím attribútumnak*** nevezünk, ha szerepel valamely kulcsban.

12.19. definíció. *Az (R,F) séma* ***harmadik normálformában*** *van, ha valahányszor $X \to A \in F^+$, akkor vagy X szuperkulcs, vagy A prím attribútum.*

A 12.3. példa $SCUR$ sémája az $SU \to R$ és $S \to C$ funkcionális függőségekkel nincs 3NF-ben, mert SU az egyetlen kulcs, tehát C nem prím, így az $S \to C$ funkcionális függőség megsérti a 3NF feltételt.

A 3NF nyilvánvalóan gyengébb feltétel, mint a BCNF, hiszen a definícióban ott van a „vagy A egy prím attribútum" kitétel. A 12.6. példa VUI sémája triviálisan 3NF-ben van, hiszen minden attribútuma prím, azonban mint már láttuk nincs BCNF-ben.

Normálformák ellenőrzése

Elvileg minden F^+-beli funkcionális függőséget ellenőrizni kellene, hogy nem sérti-e meg a BCNF, vagy a 3NF feltételt, és azt már láttuk, hogy F^+ lehet exponenciális méretű F-hez viszonyítva. Azonban bebizonyítható, hogy ha F funkcionális függőségei olyan alakúak, hogy minden jobb oldal egyetlen attribútumból áll, akkor mind a BCNF, mind a 3NF feltétel megsértését elegendő csak az F-beli függőségeken ellenőrizni. Legyen ugyanis $X \to A \in F^+$ egy olyan függőség, amelyik megsérti a megfelelő feltételt, azaz nem triviális, X nem szuperkulcs, és a 3NF esetben A nem prím. $X \to A \in F^+ \iff A \in X^+$. Abban a lépésben, amikor a LEZÁRÁS beveszi az A-t X^+-ba (8. sor), akkor egy olyan F-beli $Y \to A$ funkcionális függőséget használ, amelyre $Y \subseteq X^+$ és $A \notin Y$. Ez a függőség tehát nem triviális, A továbbra sem prím, és ha Y szuperkulcs lenne, akkor $R = Y^+ \subseteq (X^+)^+ = X^+$ alapján X is szuperkulcs lenne. Tehát az F-beli $Y \to A$ funkcionális függőség megsérti a normálforma feltételét. Az világos, hogy az F-beli funkcionális függőségek polinomiális időben ellenőrizhetők, hiszen elegendő minden függőség bal oldalának lezárását kiszámítani. Ezzel a BCNF ellenőrzést befejezzük, hiszen ha minden bal oldal lezárása R, akkor BCNF, egyébként meg találtunk egy függőséget, ami megsérti a feltételt. A 3NF-hez azonban szükségünk lehet arra, hogy egy A attribútumról el tudjuk dönteni, prím-e. Ez azonban NP-teljes probléma, lásd a 12-4. feladatot.

Veszteségmentes összekapcsolású felbontás BCNF-be

Legyen (R, F) relációs séma (ahol F a funkcionális függőségek halmaza). Ezt szeretnénk $R_1, R_2, \ldots, R_k$ részsémák uniójára szétvágni úgy, hogy a szétvágás veszteségmentes összekapcsolású legyen, és minden R_i a $\pi_{R_i}(F)$ funkcionális függőségekkel BCNF-ben legyen. A felbontás alapgondolata egyszerű:

- ha (R, F) már BCNF-ben van, akkor kész;
- ha nem, akkor két valódi részre (R_1, R_2) bontjuk, melyek összekapcsolása veszteség mentes;
- ismételjük R_1-re és R_2-re.

Ahhoz, hogy ez így működjön, két dolgot kell megmutatnunk:

- ha (R, F) nincs BCNF-ben, akkor van veszteségmentes összekapcsolású felbontása kisebb részekre;
- ha egy veszteségmentes összekapcsolású felbontás valamely tagját tovább bontjuk, akkor az új felbontás is veszteségmentes összekapcsolású.

12.20. lemma. *Legyen R relációs séma F funkcionális függőségekkel, $\rho = (R_1, R_2, \ldots, R_k)$ az R veszteségmentes összekapcsolású szétvágása. Legyen továbbá $\sigma = (S_1, S_2)$ az R_1 veszteségmentes összekapcsolású szétvágása $\pi_{R_1}(F)$-re nézve. Ekkor az $(S_1, S_2, R_2, \ldots, R_k)$ az R egy veszteségmentes összekapcsolású szétvágása.*

A 12.20. lemma bizonyítása a természetes összekapcsolás asszociativitásán alapszik. A részleteket az Olvasóra bízzuk gyakorlatként (12.3-9. gyakorlat).

Ezt már alkalmazhatjuk egy egyszerű – de sajnos exponenciális futási idejű – algoritmushoz, ami az R sémát BCNF tulajdonságú részsémákra vágja szét. A NAIV-BCNF eljárásban sajnos, a 7–8. sorokban található projekciók exponenciális méretűek is lehetnek a bemenet hosszában. Az R séma felbontásához az eljárást az R, F paraméterekkel kell meghívni. A NAIV-BCNF rekurzív eljárás: S az aktuális séma, G függőségi halmazzal. Feltesszük, hogy a G-beli funkcionális függőségek $X \to A$ alakúak, ahol A egyetlen attribútum.

NAIV-BCNF(S, G)

1 **if** van $\{X \to A\} \in G$, ami megsérti BCNF-t
2 **then** $S_1 \leftarrow \{XA\}$
3 $S_2 \leftarrow S - A$
4 $G_1 \leftarrow \pi_{S_1}(G)$
5 $G_2 \leftarrow \pi_{S_2}(G)$
6 **return** (NAIV-BCNF$(S_1, G_1) \cup$ NAIV-BCNF(S_2, G_2))
7 **else return** S

Ha azonban megengedjük, hogy néha „túllőjünk a célon”, azaz olyan sémát is tovább bontsunk, ami már BCNF-ben van, akkor nem kell a funkcionális függőségek vetítését elvégezni. Az eljárás az alábbi két lemmán alapul.

12.21. lemma.

1. *Minden kétattribútumú séma BCNF-ben van.*
2. *Ha R nincs BCNF-ben, akkor van R-ben két olyan attribútum (A és B), melyekre $(R - AB) \to A$ teljesül.*

Bizonyítás. Ha a séma kétattribútumú, $R = AB$, akkor legfeljebb két nem-triviális függőség lehet, $A \to B$ és $B \to A$. Világos, hogy ha valamelyik teljesül, akkor a függés bal oldala kulcs, azaz nem sérti meg a BCNF-t. Ha egyik sem teljesül, akkor sem sérülhet a BCNF tulajdonság.

Másfelől, tegyük fel, hogy $X \to A$ megsérti a BCNF-t. Ekkor kell legyen $B \in R - (XA)$, mert különben X szuperkulcs lenne. Erre a B-re teljesül, hogy $(R - AB) \to A$. ■

Megjegyezzük, hogy a 12.21. lemma 2. állítása nem fordítható meg, azaz előfordulhat, hogy egy R séma BCNF-ben van, de mégis van hozzá $\{A, B\}$, melyre $(R - AB) \to A$. Legyen ugyanis $R = ABC$, $F = \{C \to A, C \to B\}$. Ez nyilván BCNF-ben van, de mégis $(R - AB) = C \to A$.

A 12.21. lemma nagy előnye, hogy nem kell a függőségek vetítéseit kiszámolni ahhoz, hogy ellenőrizzük, hogy egy, az eljárás során kapott séma BCNF-ben van-e. Elegendő csupán $\{A, B\}$ attribútum párokra kiszámítani $(R - AB)^+$-t, ami a LINEÁRIS-LEZÁRÁS eljárással lineáris időben megy, tehát a teljes ellenőrzés polinomiális (köbös) idejű. Ehhez persze tudnunk kell, hogyan számíthatjuk ki $(R - AB)^+$-t anélkül, hogy a függőségeket levetítenénk. Ebben segít a következő lemma.

12.22. lemma. *Tegyük fel, hogy $R_2 \subset R_1 \subset R$, és az R séma funkcionális függőségeinek*

halmaza F. Ekkor

$$\pi_{R_2}(\pi_{R_1}(F)) = \pi_{R_2}(F) \ .$$

A bizonyítást gyakorlatként az Olvasóra bízzuk (12.3-10. gyakorlat). Ezek után a veszteségmentes összekapcsolású BCNF szétvágás algoritmusának módszere a következő. Az R sémát két részre vágjuk. Az egyik rész attribútum halmaza XA lesz, BCNF tulajdonságú, és az $X \to A$ funkcionális függőség teljesül rá. A másik rész pedig $R - A$, így a 12.14. tétel szerint a szétvágás veszteségmentes összekapcsolású. Utána ezt az eljárást alkalmazzuk rekurzívan $R - A$-ra, amíg egy olyan sémához nem érünk, amelyik teljesíti a 12.21. lemma 2. pontjának feltételét. A 12.20. lemma biztosítja, hogy az így rekurzívan generált szétvágás veszteségmentes összekapcsolású.

POLINOMIÁLIS-BCNF(R, F)

```
 1  Z ← R
 2  ▷ Az eljárás során Z az a séma, ami még nem biztos, hogy BCNF-ben van.
 3  ρ ← ∅
 4  while Z-ben van olyan A, B, hogy A ∈ (Z − AB)^+ AND |Z| > 2
 5        do Legyen A és B egy olyan pár
 6           E ← A
 7           Y ← Z − B
 8           while Y-ban van olyan C, D, hogy C ∈ (Z − CD)^+
 9                 do Y ← Y − D
10                    E ← C
11           ρ ← ρ ∪ {Y}
12           Z ← Z − E
13  ρ ← ρ ∪ {Z}
14  return ρ
```

A POLINOMIÁLIS-BCNF algoritmus futási ideje polinomiális, ténylegesen $O(n^5)$-nel becsülhető. Z mérete a 4–12. sorok ciklusának minden egyes végrehajtásakor csökken, így legfeljebb n-szer hajtjuk végre. A 4. sorban legfeljebb $O(n^2)$ attribútum párra kell $(Z-AB)^+$-t kiszámolni, ami minden egyes alkalommal lineáris időben elvégezhető a LINEÁRIS-LEZÁRÁS algoritmussal, így összesen $O(n^3)$ lépés ciklusonként. A 8–10. sorok ciklusában Y mérete csökken iterációkként, azaz a 3–12. sorok minden egyes végrehajtásakor a 8–10. sorok legfeljebb n iterációt jelentenek. A 8. sor **while** utasításának feltételét $O(n^2)$ attribútum párra kell ellenőrizni, egy ellenőrzés lineáris idejű. Az algoritmus lépésszámában a 8–10. sorok dominálnak, $n \cdot n \cdot O(n^2) \cdot O(n) = O(n^5)$ lépés összesen.

Függőségmegőrző szétvágás 3NF-re

Azt már láttuk, hogy nem mindig lehetséges egy sémát szétvágni BCNF részsémákra úgy, hogy a szétvágás függőségőrző legyen. Ha azonban 3NF-et követelünk csak meg, akkor a MINIMÁLIS-FEDÉS algoritmus segítségével megadható a megfelelő szétvágás. Legyen R relációs séma, F funkcionális függőségekkel. A MINIMÁLIS-FEDÉS algoritmust használva konstruáljuk meg az F valamely minimális fedését, G-t. Legyen $G = \{X_1 \to A_1, X_2 \to A_2, \ldots, X_k \to A_k\}$.

12.23. tétel. *A $\rho = (X_1A_1, X_2A_2, \ldots, X_kA_k)$ szétvágás az R függőségmegőrző szétvágása*

3NF-re.

Bizonyítás. Mivel $G^+ = F^+$, és az $X_i \to A_i$ funkcionális függőség benne van $\pi_{R_i}(F)$-ben, ezért a felbontás megőriz minden funkcionális függőséget. Tegyük fel indirekt, hogy az $R_i = X_iA_i$ séma nem 3NF, azaz van egy olyan $U \to B$ funkcionális függőség, ami megsérti a 3NF feltételt, vagyis nem triviális függés, U nem szuperkulcs R_i-ben és B nem prím R_i-ben. Két eset lehetséges. Ha $B = A_i$, akkor abból, hogy Unem szuperkulcs, az következik, hogy $U \subsetneq X_i$. Ekkor az $U \to A_i$ funkcionális függőség ellentmond annak, hogy $X_i \to A_i$ minimális fedés eleme volt, mert bal oldala csökkenthető lenne. Ha $B \neq A_i$, akkor $B \in X_i$ teljesül. B nem prím R_i-ben, tehát X_i nem lehet kulcs, csak szuperkulcs. Ekkor azonban X_i tartalmazna egy Y kulcsot, $Y \subsetneq X_i$, továbbá $Y \to A_i$ teljesül, ami ellentmond G minimalitásának, mert $X_i \to A_i$ bal oldala csökkenthető lenne. ■

Amennyiben azt szeretnénk, hogy a szétvágás függőségmegőrzés mellett még veszteségmentes összekapcsolású is legyen, akkor a 12.23. tételben megadott ρ szétvágáshoz R egy X kulcsát kell hozzátennünk. Mint azt már láttuk, az ***összes*** kulcsot nem tudjuk megtalálni polinomiális időben, ***egyet*** azonban viszonylag egyszerűen mohó módon megkaphatunk, a részleteket az Olvasóra bízzuk gyakorlatként (12.3-11. gyakorlat).

12.24. tétel. *Legyen (R, F) egy relációs séma, és legyen $G = \{X_1 \to A_1, X_2 \to A_2, \ldots, X_k \to A_k\}$ az F egy minimális fedése. Legyen továbbá X egy kulcs (R, F)-ben. Ekkor $\tau = (X, X_1A_1, X_2A_2, \ldots, X_kA_k)$ szétvágás az R függőségmegőrző veszteségmentes összekapcsolású szétvágása 3NF-re.*

Bizonyítás. Azt már a 12.23. tétel bizonyításánál láttuk, hogy az $R_i = X_iA_i$ sémák 3NF-ben vannak $i = 1, 2, \ldots, k$-ra. Az $R_0 = X$ sémában nem lehet nem triviális függés, mert akkor X nem lenne kulcs, csak szuperkulcs.

Azt, hogy τ veszteségmentes összekapcsolású, a KAPCSOLÁS-TESZT algoritmus használatával mutatjuk meg. Pontosabban, belátjuk, hogy a táblázatban a KAPCSOLÁS-TESZT algoritmus futása után az X-nek megfelelő sor csupa 0 lesz. Legyen $A_1, A_2, \ldots, A_m$ az $R - X$ attribútumainak az a sorrendje, amelyben a LEZÁRÁS veszi be őket X^+-ba. Mivel X kulcs, ezért mindegyik $R - X$-beli attribútum sorra kerül LEZÁRÁS során. i-re vonatkozó indukcióval látjuk be, hogy az X sorában az A_i oszlopban álló elem 0 lesz a KAPCSOLÁS-TESZT algoritmus futása során.

Az $i = 0$ kiindulási eset triviális. Tegyük fel, hogy igaz $(i - 1)$-re, és nézzük meg, hogy A_i mikor és miért kerül be X^+-ba. LEZÁRÁS 6–8. sorában egy olyan $Y \to A_i$ funkcionális függőséget használunk, amelyre $Y \subseteq X \cup \{A_1, A_2, \ldots, A_{i-1}\}$. Ekkor $Y \to A_i \in G$, $YA_i = R_j$ valamely j-re. Az X-nek és $YA_i = R_j$-nek megfelelő sorok megegyeznek X oszlopaiban (csupa 0 az indukciós feltevés szerint), tehát ezen két sor A_i-beli értékeit a KAPCSOLÁS-TESZT algoritmus egyenlővé teszi. Az $YA_i = R_j$-nek megfelelő sorban 0 áll, így az X-nek megfelelő sorban is 0 lesz. ■

Érdekes tény, hogy annak ellenére, hogy tetszőleges séma 3NF szétvágását megtaláljuk polinomiális időben, azt eldönteni, hogy az adott (R, F) séma maga 3NF-ben van-e, az NP-teljes probléma, lásd a 12-4. feladatot. Ugyanakkor a BCNF tulajdonság eldönthető polinomiális időben. A különbség abból adódik, hogy 3NF-hez egy attribútumról el kell tudni

dönteni, hogy prím attribútum-e, ehhez viszont a séma kulcsait kellene megtalálni.

12.3.5. Többértékű függőségek

Tekintsük a következő példát.

12.8. példa. A 12.1. példában a funkcionális függőségek mellett másfajta függőségek is teljesülnek. Egy tárgyból egy héten több különböző időpontban és helyen is van óra. A sémához tartozó reláció részlete lehet a következő:

Tanár	Tárgy	Terem	Diák	Jegy	Idő
Beke Manó	Analízis	K.Aud.Max	Kovács József	3	hétfő 8–10
Beke Manó	Analízis	R522	Kovács József	3	szerda 12–2
Beke Manó	Analízis	K.Aud.Max	Nagy Béla	4	hétfő 8–10
Beke Manó	Analízis	R522	Nagy Béla	4	szerda 12–2

Egy tárgyhoz az idő és terem attribútum értékek egy halmaza tartozik, és a többi attribútum ezek minden lehetséges értékével megismétlődik. Az ITe és DJ attribútumhalmazok függetlenek, azaz minden kombinációban előfordulnak.

Azt mondjuk, hogy az X attribútumhalmaztól ***többértékűen függ*** az Y attribútumhalmaz, jelölésben $X \twoheadrightarrow Y$, ha minden X-en felvett értékhez létezik az Y attribútumhalmazon felvett értékek olyan halmaza, amelyik semmilyen függési kapcsolatban sincsenek az $R - X - Y$ attribútumhalmazon felvett értékekkel. A pontos definíció a következő:

12.25. definíció. *Az R relációs sémában teljesül az $X \twoheadrightarrow Y$ **többértékű függőség,** ha minden az R sémához tartozó r relációra igaz, hogy tetszőleges t_1, t_2 sorokra, melyekre $t_1[X] = t_2[X]$, léteznek $t_3, t_4 \in r$ melyekre*

- $t_3[XY] = t_1[XY]$
- $t_3[R - XY] = t_2[R - XY]$
- $t_4[XY] = t_2[XY]$
- $t_4[R - XY] = t_1[R - XY]$

teljesül.[1]

A 12.8. példában Tá↠IT teljesül.

12.26. megjegyzés. *A funkcionális függőség **egyenlőséggeneráló** függőség, azaz két dolog egyenlőségéből másik két dolog egyenlőségére következtet. A többértékű függőség **sorgeneráló** függőség, azaz két sor létezése, melyek valahol egyenlőek, maga után vonja más sorok létezését.*

A többértékű függőségekhez is létezik egy teljes és ellentmondásmentes axióma rendszer, hasonlóan a funkcionális függőségek Armstrong-axiómáihoz. A logikai következmény, illetve a levezethetőség fogalma is hasonlóan definiálható. Azt mondjuk, hogy egy $X \twoheadrightarrow Y$ ***logikai következménye*** az M többértékű függőség halmaznak, jelölésben $M \models X \twoheadrightarrow Y$, ha minden olyan reláció, amelyikben M teljesül, teljesíti $X \twoheadrightarrow Y$-t is.

[1] Elegendő csak t_3 létezését megkövetelni, mert abból már t_4 létezése következik. Azonban a többértékű függőség szimmetriája így jobban látszik.

Vegyük észre, hogy ha $X \to Y$ teljesül, akkor $X \twoheadrightarrow Y$ is igaz. A 12.25. definícióban szereplő t_3 és t_4 soroknak a $t_3 = t_2$ és $t_4 = t_1$ választás megfelel. Így a funkcionális és a többértékű függőségeket egy közös axiómarendszerrel jellemezhetjük. Az (A1)–(A3) Armstrong-axiómákon kívül a következő öt axiómára van szükségünk. Legyen R a relációs séma.

(A4) *Komplementálás:* $\{X \twoheadrightarrow Y\} \models X \twoheadrightarrow (R - X - Y)$.

(A5) *Bővítés:* Ha $X \twoheadrightarrow Y$ teljesül, és $V \subseteq W$, akkor $WX \twoheadrightarrow VY$.

(A6) *Tranzitivitás:* $\{X \twoheadrightarrow Y, Y \twoheadrightarrow Z\} \models X \twoheadrightarrow (Z - Y)$.

(A7) $\{X \to Y\} \models X \twoheadrightarrow Y$.

(A8) Ha $X \twoheadrightarrow Y$, $Z \subseteq Y$, továbbá valamely Y-tól diszjunkt W-re $W \to Z$ igaz, akkor $X \to Z$ is teljesül.

Beeri, Fagin és Howard bizonyították, hogy az (A1)–(A8) axiómarendszer ellentmondásmentes és teljes a funkcionális és többértékű függőségekre. Az ellentmondásmentesség bizonyítását gyakorlatként az Olvasóra bízzuk (12.3-12. gyakorlat), a teljesség bizonyítása meghaladja e könyv kereteit. Az (A1)–(A8) axiómákon kívül a 12.2. lemma szabályai ugyanúgy érvényesek, mint mikor csak funkcionális függőségeket tekintettünk. További szabályokat ad a következő állítás.

12.27. állítás. *Többértékű függőségekre igazak az alábbiak:*

1. *Egyesítés szabály:* $\{X \twoheadrightarrow Y, X \twoheadrightarrow Z\} \models X \twoheadrightarrow YZ$.
2. *Pszeudotranzitivitás:* $\{X \twoheadrightarrow Y, WY \twoheadrightarrow Z\} \models X \twoheadrightarrow (Z - WY)$.
3. *Vegyes pszeudotranzitivitás:* $\{X \twoheadrightarrow Y, XY \to Z\} \models X \to (Z - Y)$.
4. *Szétvágási szabály: többértékű függőségekre: ha* $X \twoheadrightarrow Y$ *és* $X \twoheadrightarrow Z$, *akkor* $X \twoheadrightarrow (Y \cap Z)$, $X \twoheadrightarrow (Y - Z)$ *és* $X \twoheadrightarrow (Z - Y)$ *is teljesül.*

A 12.27. állítás bizonyítását gyakorlatként az Olvasóra hagyjuk (12.3-13. gyakorlat).

Függőségi bázis

Fontos különbség funkcionális függőségek és többértékű függőségek között, hogy amíg $X \to Y$-ból azonnal következett, hogy $X \to A$ minden Y-beli A attribútumra, addig a többértékű függőségekre vonatkozó szétvágási szabály csak akkor engedi meg az $X \twoheadrightarrow Y$-ból $X \twoheadrightarrow A$-ra való következtetést, ha van egy olyan Z attribútumhalmaz, hogy $X \twoheadrightarrow Z$ és $Z \cap Y = A$, vagy $Y - Z = A$. Azonban a következő tétel igaz.

12.28. tétel. *Legyen R relációs séma, $X \subset R$ attribútumhalmaz. Ekkor létezik az $R - X$ attribútumhalmaznak olyan $Y_1, Y_2, \ldots, Y_k$ partíciója, melyre teljesül, hogy $Z \subseteq R - X$ esetén $X \twoheadrightarrow Z$ akkor és csak akkor igaz, ha $Z = Y_i$ vagy Z több Y_i egyesítése.*

Bizonyítás. Induljunk ki az $W_1 = R - X$ egyelemű partícióból. Ezt fogjuk fokozatosan tovább finomítani, mindvégig megtartva azt a tulajdonságot, hogy $X \twoheadrightarrow W_i$ minden W_i-re, ami az éppen aktuális felbontásban szerepel. Ha $X \twoheadrightarrow Z$ és Z nem áll elő néhány W_i egyesítéseként, akkor minden olyan W_i-t, amelyikre $W_i \cap Z$ és $W_i - Z$ sem üres, helyettesítsünk $W_i \cap Z$ és $(W_i - Z)$-vel. A 12.27. állítás szétvágási szabálya szerint $X \twoheadrightarrow (W_i \cap Z)$ és $X \twoheadrightarrow (W_i - Z)$ teljesül. Mivel $R - X$ véges, ezért ez az eljárás végetér, azaz minden olyan Z, melyre $X \twoheadrightarrow Z$,

Z a partíció néhány blokkjának egyesítése. A bizonyítás befejezéséhez csak annyit kell még megjegyezni, hogy a 12.27. állítás egyesítés szabálya alapján a partíció néhány blokkjának egyesítése többértékűen függ X-től. ■

12.29. definíció. *A 12.28. tételben a D funkcionális és többértékű függőségeket tartalmazó függőségi halmaz alapján konstruált $Y_1, Y_2, \ldots, Y_k$ partíciót az X (D-re vonatkozó) **függőségi bázisának** nevezzük.*

12.9. példa. Tekintsük a 12.1. és a 12.8. példákból már jól ismert

R(**T**anár,**Tá**rgy,**Te**rem,**D**iák,**J**egy,**I**dő)

sémát. A 12.8. példában már megállapítottuk, hogy **Tá**↠**TeI**. A komplementálási szabály alapján **Tá**↠**TDJ** következik. Azt is tudjuk, hogy **Tá**→**T**. (A7) axióma alapján ebből **Tá**↠**T** következik. A szétvágási szabály alapján kapjuk, hogy **Tá**↠**DJ**. Könnyen ellenőrizhető, hogy **Tá** és **T** kivételével más egyelemű attribútumhalmazt **Tá** nem határoz meg többértékű függéssel. Tehát **Tá** függőségi bázisa {**T**,**TeI**,**DJ**}.

Funkcionális és többértékű függőségek adott D halmazára szeretnénk kiszámítani a D logikai következményeiből álló D^+ halmazt. Ennek egy módja lenne, hogy az (A1)–(A8) axiómákat alkalmazva ismételten bővítjük a függőségi halmazt, amíg további bővítés már nem lehetséges. Azonban ez az eljárás exponenciálisan sok lépést igényelhet D méretének függvényében. Jobbat persze nem várhatunk, hiszen azt már láttuk, hogy maga D^+ mérete is lehet exponenciális D méretében. Sok alkalmazásban viszont nem kell a teljes D^+-t kiszámítani, hanem elegendő eldönteni, hogy egy adott $X \to Y$ funkcionális, illetve $X \twoheadrightarrow Y$ többértékű függőség vajon D^+-ban van-e. Egy $X \twoheadrightarrow Y$ többértékű függőség eldöntéséhez elegendő X függőségi bázisát kiszámítani, majd ellenőrizni, hogy $Z - X$ a partíció néhány blokkjának egyesítése-e. Igaz a következő tétel.

12.30. tétel (Beeri). *Ha egy X attribútumhalmaz D függőségi halmaz szerinti függőségi bázisát akarjuk kiszámítani, akkor elegendő csak az alábbi többértékű függőségeket tartalmazó M halmazt tekinteni:*

1. *Az összes D-beli többértékű függőséget és*
2. *Minden D-beli $X \to Y$-re az $X \twoheadrightarrow A_1, X \twoheadrightarrow A_2, \ldots, X \twoheadrightarrow A_k$ többértékű függőségeket, ahol $Y = A_1A_2 \ldots A_k$, és az A_i-k egyedi attribútumok.*

Most már csak funkcionális függőségeket kell tudnunk eldönteni a függőségi bázis alapján. A LEZÁRÁS algoritmus ugyanis csak akkor működik helyesen, ha többértékű függőségek nincsenek. Ebben segít az alábbi tétel.

12.31. tétel (Beeri). *Tegyük fel, hogy $A \notin X$ és X függőségi bázisát a 12.30. tételben kapott M többértékű függőségi halmazra vonatkozólag ismerjük. $X \to A$ akkor és csak akkor teljesül, ha*

1. *A egy egyelemű osztályt alkot az X függőségi bázisát alkotó partícióban, és*
2. *van egy Y attribútumhalmaz, ami A-t nem tartalmazza, $Y \to Z$ az eredetileg adott D függőségi halmaz egy eleme, valamint $A \in Z$.*

Fentiek alapján a következő polinomiális idejű algoritmust adhatjuk az X attribútumhalmaz függőségi bázisának kiszámítására. Adott többértékű függőségek M halmaza, R relációs séma, melyre $X \subseteq R$.

FÜGGŐSÉGI-BÁZIS(R, M, X)

```
1  S ← {R − X}                          ▷ A függőségi bázisban szereplő halmazok együttese S.
2  repeat
3        for minden V ↠ W ∈ M
4            do if van Y ∈ S melyre Y ∩ W ≠ ∅ ∧ Y ∩ V = ∅
5                 then S ← S − {{Y}} ∪ {{Y ∩ W}, {Y − W}}
6  until S nem változik
7  return S
```

Világos, hogy ha a FÜGGŐSÉGI-BÁZIS eljárás 3–5. soraiban $\mathcal{S}$ változik, akkor az algoritmus valamelyik partícióbeli blokkot vágja szét. Ebből látható, hogy a futási idő M és R méretének polinomiális függvénye. Gondos megvalósítással elérhető, hogy ez a polinom $O(|M| \cdot |R|^3)$ legyen (12-5. feladat).

Negyedik normálforma (4NF)

A Boyce–Codd normálforma általánosítható arra az esetre, ha funkcionális függőségek mellett többértékű függőségeket is tekintünk, és az általuk okozott redundanciától is meg akarunk szabadulni.

12.32. definíció. *Legyen R relációs séma, D többértékű és funkcionális függőségek halmaza R-en. R* ***negyedik normálformában (4NF)*** *van, ha tetszőleges $X \twoheadrightarrow Y \in D^+$ többértékű függőségre, melyre $Y \not\subseteq X$ és $R \neq XY$, teljesül, hogy X szuperkulcs R-ben.*

Vegyük észre, hogy 4NF$\Longrightarrow$BCNF. Valóban, ha egy $X \to A$ funkcionális függőség megsértené a BCNF feltételt, akkor $A \notin X$ és XA nem tartalmazhatja R összes attribútumát, mert akkor X szuperkulcs lenne. Viszont, (A8) alapján $(X \to A)$-ból $X \twoheadrightarrow A$ következik, ami a 4NF feltételt sérti meg.

Az R séma D többértékű és funkcionális függőségek halmazával szétvágható $\rho = (R_1, R_2, \ldots, R_k)$ alakba, ahol minden egyes R_i már 4NF-ben van, és a szétvágás veszteségmentes összekapcsolású. A módszer ugyanazt az elvet követi, mint a BCNF felbontás, azaz ha az S séma nincs 4NF-ben, akkor van egy $X \twoheadrightarrow Y$ függőség a D S-re vetített függőségei között, ami megsérti a 4NF feltételt. Azaz, X nem szuperkulcs S-ben, Y nem üres és nem X részhalmaza, valamint X és Y egyesítése nem S. Feltehető, hogy X és Y diszjunktak, hiszen $X \twoheadrightarrow (Y - X)$ következik $X \twoheadrightarrow Y$-ból (A1), (A7), és a szétvágási szabály alkalmazásával. Ekkor S helyettesíthető $S_1 = XY$ és $S_2 = S - Y$ sémákkal, mindkettőnek kevesebb attribútuma van, mint S-nek, azaz az eljárás véges időn belül végetér.

Két dolgot kell csak látnunk, hogy ez helyes eredményt adjon.

- Az S_1, S_2 szétvágás veszteségmentes összekapcsolású.
- Hogyan számíthatjuk ki a $\pi_S(D)$ függőségi halmazt?

Az első problémára válasz az alábbi tétel.

12.33. tétel. *Az R relációs séma $\rho = (R_1, R_2)$ szétvágása akkor és csak akkor veszteségmentes összekapcsolású a D többértékű és funkcionális függőségek halmazára vonatkozólag, ha*

$$(R_1 \cap R_2) \twoheadrightarrow (R_1 - R_2)\ .$$

Bizonyítás. A $\rho = (R_1, R_2)$ szétvágás pontosan akkor veszteségmentes összekapcsolású, ha bármilyen az R sémához tartozó r relációra, amelyik kielégíti D függőségeit, igaz, hogy ha μ és ν két r-beli sor, ha létezik a φ sor, melyre $\varphi[R_1] = \mu[R_1]$ és $\varphi[R_2] = \nu[R_2]$, akkor r-ben megtalálható. Pontosabban, φ a μ R_1-re és a ν R_2-re való vetületének természetes összekapcsolása, ami pontosan akkor létezik, ha $\mu[R_1 \cap R_2] = \nu[R_1 \cap R_2]$. Tehát az, hogy φ mindig r-ben van, ekvivalens azzal, hogy $(R_1 \cap R_2) \twoheadrightarrow (R_1 - R_2)$. ■

A D függőségi halmaz $\pi_S(D)$ vetületének számításához Aho, Beeri és Ullman tételét használhatjuk. $\pi_S(D)$ mindazon többértékű és funkcionális függőségek halmaza, amelyek D logikai következményei, és csak S-be eső attribútumokat használnak.

12.34. tétel (Aho, Beeri és Ullman). *$\pi_S(D)$ függőségei a következők:*

- *Minden $X \to Y \in D^+$-ra, ha $X \subseteq S$, akkor $X \to (Y \cap S) \in \pi_S(D)$.*
- *Minden $X \twoheadrightarrow Y \in D^+$-ra, ha $X \subseteq S$, akkor $X \twoheadrightarrow (Y \cap S) \in \pi_S(D)$.*

Más függőség teljesülése S-ben nem vezethető le abból, hogy R-ben D teljesül.

Sajnos, ez a tétel nem segít abban, hogy a vetület függőségeket polinomiális időben előállítsuk, hiszen maga D^+ kiszámítása exponenciális időbe telhet. Így azonban a 4NF szétvágás algoritmusa sem lesz polinomiális idejű, hiszen az egyes részsémákban a vetület függőségi halmaz szerint kell ellenőrizni, hogy a 4NF feltétel teljesül-e. Ez szöges ellentétben áll a BCNF szétvágás esetével. A különbség oka, hogy a BCNF tulajdonság ellenőrzéséhez nem kell kiszámítani a vetület függőségeket, hanem csupán attribútumhalmazok lezártjait kell vizsgálni a 12.21. lemma szerint.

Gyakorlatok

12.3-1. Igazak-e az alábbi következtetési szabályok?

(a) Ha $XW \to Y$ és $XY \to Z$, akkor $X \to (Z - W)$.

(b) Ha $X \twoheadrightarrow Y$ és $Y \twoheadrightarrow Z$, akkor $X \twoheadrightarrow Z$.

(c) Ha $X \twoheadrightarrow Y$ és $XY \to Z$, akkor $X \to Z$.

12.3-2. Bizonyítsuk be a 12.30. tételt, azaz lássuk be a következőt:

Legyen D funkcionális és többértékű függőségekből álló halmaz, és legyen $m(D) = \{X \twoheadrightarrow Y\colon X \twoheadrightarrow Y \in D\} \cup \{X \twoheadrightarrow A\colon A \in Y \text{ valamely } X \to Y \in D\text{-re}\}$. Ekkor igazak az alábbiak:

(a) $D \models X \to Y \Longrightarrow m(D) \models X \twoheadrightarrow Y$, és

(b) $D \models X \twoheadrightarrow Y \iff m(D) \models X \twoheadrightarrow Y$.

Útmutató. A b. pont bizonyításához alkalmazzunk indukciót a levezetési szabályokon.

12.3-3. Tekintsük egy befektetési cég adatbázisát, melynek attribútumai a következők: *Br* (bróker), *I* (a bróker irodája), *Be* (befektető), *R* (részvény), *M* (a befektető tulajdonában levő részvény mennyisége), *O* (a részvény osztaléka). Érvényes funkcionális függőségek: $R \to O$, $Be \to Br$, $BeR \to M$, $Br \to I$.

(a) Adjuk meg az $S = BrIRMBeO$ séma egy kulcsát.

(b) Hány kulcs van az S sémában?
(c) Adjuk meg S veszteségmentes összekapcsolású felbontását BCNF sémákra.
(d) Bontsuk fel S-t függőségőrzően és veszteségmentesen 3NF sémákra.

12.3-4. A 12.3-3. gyakorlat S sémáját szétvágjuk az *RO*, *BeBr*, *BeRM* és *BrI* sémákra. Veszteségmentes kapcsolású ez a szétvágás?

12.3-5. Tegyük fel most, hogy a 12.3-3. gyakorlat S sémáját a *BeRM*, *BeBr*, *RO* és *BeRI* sémákkal reprezentáljuk. Adjuk meg a 12.3-3. gyakorlatban szereplő függőségek a megadott részsémára való vetületeinek minimális fedőjét. Találjunk minimális fedőt a vetített függőségi halmaz184-803ok egyesítéséhez. Függőségőrző ez a szétvágás?

12.3-6. A 12.3-3. gyakorlat példájában a $R \to O$ funkcionális függőséget kicseréljük a $R \twoheadrightarrow O$ többértékű függőségre. Azaz, O most a részvény osztalék „történetét" reprezentálja.

(a) Számítsuk ki *Be* függőségi bázisát.
(b) Számítsuk ki *BrR* függőségi bázisát.
(c) Adjuk meg R 4NF szétvágását.

12.3-7. Tekintsük az R relációs séma $\rho = \{R_1, R_2, \ldots, R_k\}$ szétvágását. Legyen $r_i = \pi_{R_i}(r)$, valamint $m_\rho(r) = \bowtie_{i=1}^{k} \pi_{R_i}(r)$. Bizonyítsuk be a következő állítást.

(a) $r \subseteq m_\rho(r)$.
(b) Ha $s = m_\rho(r)$, akkor $\pi_{R_i}(s) = r_i$.
(c) $m_\rho(m_\rho(r)) = m_\rho(r)$.

12.3-8. Bizonyítsuk be, hogy az (R, F) séma pontosan akkor van BCNF-ben, ha tetszőleges $A \in R$-re és $X \subset R$ kulcsra igaz, hogy nincs olyan $Y \subseteq R$, amire $X \to Y \in F^+$, $Y \to X \notin F^+$, $Y \to A \in F^+$ és $A \notin Y$.

12.3-9. Bizonyítsuk be a 12.20. lemmát.

12.3-10. Tegyük fel, hogy $R_2 \subset R_1 \subset R$, és az R séma funkcionális függőségeinek halmaza F. Bizonyítsuk be, hogy $\pi_{R_2}(\pi_{R_1}(F)) = \pi_{R_2}(F)$.

12.3-11. Adjunk $O(n^2)$ futási idejű algoritmust az (R, F) relációs séma egy kulcsának megtalálására. *Útmutatás.* Használjuk ki, hogy R szuperkulcs, és minden szuperkulcs tartalmaz kulcsot. Egyenként próbáljuk meg az attribútumokat elhagyni, és teszteljük, hogy az így kapott halmaz még kulcs-e.

12.3-12. Bizonyítsuk be, hogy az (A1)–(A8) axiómák ellentmondásmentesek a funkcionális és többértékű függőségek körében.

12.3-13. Vezessük le az (A1)–(A8) axiómákból a 12.27. állítás négy szabályát.

12.4. Általános függőségek

12.4.1. Összekapcsolási függőségek

Ebben a fejezetben két olyan függőségről lesz szó, melyek az előzőek általánosításai, de nem axiomatizálhatók az (A1)–(A8) axióma rendszerhez hasonló axiómákkal.

A 12.33. tétel azt mondja ki, hogy a többértékű függőség ténylegesen ekvivalens azzal, hogy valamely szétvágás két részre veszteségmentes összekapcsolású. Ennek általánosítása az ***összekapcsolási függőség.***

12.35. definíció. *Legyen R egy relációs séma és legyen $R = \bigcup_{i=1}^{k} X_i$. Azt mondjuk, hogy az*

R sémához tartozó r reláció kielégíti a

$$\bowtie[X_1, X_2, \ldots, X_k]$$

összekapcsolási függőséget, *ha*

$$r = \bowtie_{i=1}^{k} \pi_{X_i}(r) \ .$$

Ebben a megfogalmazásban az r pontosan akkor teljesíti a $X \twoheadrightarrow Y$ többértékű függőséget, ha r teljesíti az $\bowtie[XY, X(R-Y)]$ összekapcsolási függőséget. A $\bowtie[X_1, X_2, \ldots, X_k]$ összekapcsolási függőség azt fejezi ki, hogy a $\rho = (X_1, X_2, \ldots, X_k)$ szétvágás veszteségmentes összekapcsolású. Ennek alapján definiálható az ***ötödik normálforma (5NF).***

12.36. definíció. *Az R séma* ***ötödik normálformában*** *van, ha 4NF, és nincs nem triviális összekapcsolási függősége.*

Az 5NF normál formának elsősorban elméleti jelentősége van. A gyakorlatban alkalmazott sémáknak rendszerint van egyértelmű ***elsődleges kulcsa.*** Ennek használatával szét lehetne vágni a sémát két attribútumos sémákra, ahol az egyik attribútum minden részsémában az elsődleges kulcs.

12.10. példa. Tekintsük egy bank ügyfél adatbázisát: (**Ü**gyfélszám, **N**év, **C**ím, **E**gyenleg). Itt az **Ü** egyértelmű azonosító, tehát az **(ÜN,ÜC,ÜE)** szétvágás veszteségmentes összekapcsolású, de nem éri meg, nem nyerünk vele tárhelyet, és az anomáliákat sem szüntetne meg.

Egy függőségi rendszer ***axiomatizálható,*** ha a létezik véges sok következtetési szabály úgy, hogy a logikai következmény fogalma egybeesik a szabályok alapján történő levezethetőséggel. Például funkcionális függőségek rendszerére az Armstrong-axiómák axiomatizálást adnak, funkcionális és többértékű függőségek axiomatizálását adják az (A1)–(A8) axiómák. Igaz az következő negatív eredmény.

12.37. tétel. *A összekapcsolási függőségek családja nem axiomatizálható.*

Ennek ellenére Abiteboul, Hull és Vianu könyvükben belátják, hogy a logikai következmény feladat összekapcsolási és funkcionális függőségek együttesére algoritmussal eldönthető. A feladat bonyolultságáról a következőt lehet mondani:

12.38. tétel.

- *NP-teljes eldönteni, hogy egy összekapcsolási függőség logikailag következik-e egy adott összekapcsolási függőségből és funkcionális függőségből.*
- *NP-nehéz eldönteni, hogy többértékű függőségek adott halmazából logikailag következik-e egy adott összekapcsolási függőség.*

12.4.2. Elágazó függőségek

A funkcionális függőségek általánosításai az ***elágazó függőségek.*** Tegyük fel, hogy $A, B \subset R$ és nincsen olyan $q+1$ sor az R sémához tartozó r relációban, melyek legfeljebb p különböző értéket tartalmaznak A-beli oszlopokban, de van olyan B-beli oszlop, melyben mind a

$q+1$ érték különböző. Ekkor azt mondjuk, hogy B **(p,q)**-függ A-tól. Jelölésben: $A \xrightarrow{p,q} B$. Speciálisan, $A \xrightarrow{1,1} B$ akkor és csak akkor teljesül, ha B funkcionálisan függ A-tól.

12.11. példa. Tekintsük például a következő adatbázist: egy kamion útjai (érintett országok).

- Egy út: 4 különböző ország.
- Egy országnak legfeljebb 5 szomszédja van.
- 30 ország jöhet szóba.

Legyenek x_1, x_2, x_3, x_4 az érintett országok attribútumai. Ekkor nem igaz, hogy $x_i \xrightarrow{1,1} x_{i+1}$, de egy másfajta függőség fennáll:

$$x_i \xrightarrow{1,5} x_{i+1} .$$

Ezen függőségek használatával az adatbázis tárolási helyigénye csökkenthető jelentősen. Ugyanis felvehetünk egy kis segédtáblázatot, melyben az egyes országok szomszédait tároljuk valamilyen sorrendben, és azután az adatbázisban már az első ország után csak a szomszédok sorszámait kell tárolni. Az országok azonosításához legalább 5 bit kell, míg a szomszédok sorszámai 1 és 5 közé esnek, ezért elég 3 bit a leírásukhoz.

Értelmezhetjük az $X \subset R$ attribútumhalmaz (p, q)-lezártját:

$$C_{p,q}(X) = \{A \in R\colon X \xrightarrow{p,q} A\} .$$

Speciálisan, $C_{1,1}(X) = X^+$. Elágazó függőségekre már olyan alap kérdések is nehezek, mint az ***Armstrong-reláció*** létezése adott függőségi rendszerhez.

12.39. definíció. *Legyen R relációs séma, F pedig az R-en értelmezett valamely $\mathcal{F}$ függőségi családba tartozó függőségek rendszere. Az R sémához tartozó r reláció* ***Armstrong-reláció*** *F-hez, ha az r-ben teljesülő $\mathcal{F}$-beli függőségek halmaza pontosan F.*

Armstrong bizonyította, hogy funkcionális függőségek tetszőleges F halmazára létezik Armstrong-reláció F^+-hoz. A bizonyítás attribútumhalmazok F szerinti lezártjának a 12.2-1. gyakorlatban szereplő három tulajdonságán alapul. Elágazó függőségekre a három közül csak az első kettő teljesül általában.

12.40. lemma. *Legyen $0 < p \le q$, R relációs séma. Minden $X, Y \subseteq R$ attribútum halmazra igaz*

1. $X \subseteq C_{p,q}(X)$ *és*
2. $X \subseteq Y \Longrightarrow C_{p,q}(X) \subseteq C_{p,q}(Y)$.

Létezik olyan $C\colon 2^R \to 2^R$ leképezés, melyhez léteznek p, q természetes számok, hogy a C-hez nem létezik Armstrong-reláció a (p, q)-függőségek családjához.

Grant és Minker az elágazó függőségekhez hasonló ***numerikus függőségeket*** vizsgálták. $X, Y \subseteq R$ attribútumhalmazokra $X \xrightarrow{k} Y$ teljesül az R sémához tartozó r relációban, ha az X attribútumhalmazon felvett minden lehetséges értékhez, legfeljebb k különböző Y-on felvett sor tartozik. Ez a feltétel az $X \xrightarrow{1,k} Y$ feltételnél erősebb, hiszen ez utóbbi csak annyit követel meg, hogy Y minden egyes oszlopában külön-külön legfeljebb k különböző érték álljon olyan sorokban, melyek az X attribútumhalmazon megegyeznek. Így pedig $k^{|Y-X|}$ különböző Y vetület lehetséges. A numerikus függőségeket speciális esetekben sikerült axiomatizálni, ennek alapján Katona belátta, hogy az elágazó függőségek nem axiomatizálhatók.

Az még nyitott kérdés, hogy a logikai következmény feladat algoritmussal eldönthető-e az elágazó függőségek körében.

Gyakorlatok

12.4-1. Bizonyítsuk be a 12.38. tételt.

12.4-2. Bizonyítsuk be a 12.40. lemmát.

12.4-3. Bizonyítsuk be, hogy ha $p = q$, akkor a 12.40. lemmában szereplő két tulajdonságon kívül az is igaz, hogy $C_{p,p}\left(C_{p,p}(X)\right) = C_{p,p}(X)$.

12.4-4. ***Lezárásnak*** nevezünk egy R séma részhalmazaiból a részhalmazaiba képező függvényt, ha a 12.40. lemma két tulajdonságát és a 12.4-3. gyakorlatban szereplő harmadikat teljesíti. Bizonyítsuk be, hogy ha $C\colon 2^R \to 2^R$ lezárás, és F azon függőségek családja, melyre $X \to Y \iff Y \subseteq C(X)$, akkor F-hez létezik Armstrong-reláció az $(1,1)$-függőségek és a $(2,2)$-függőségek családjaiban.

12.4-5. Legyen C az a lezárás, melyre

$$C(X) = \begin{cases} X, & \text{ha } |X| < 2\ , \\ R \text{ egyébként}\ . \end{cases}$$

Bizonyítsuk be, hogy C-hez *nincs* Armstrong-reláció az (n,n)-függőségek családjában, ha $n > 2$.

Feladatok

12-1. Külső attribútum

Maier ***külső attribútumnak*** nevezi az A attribútumot az $X \to Y$ funkcionális függőségben az F függőségi halmazra vonatkozólag az R sémában, ha a következő két feltétel valamelyike teljesül:

1. $(F - \{X \to Y\}) \cup \{X \to (Y - A)\} \models X \to Y$, vagy
2. $(F - \{X \to Y\}) \cup \{(X - A) \to Y\} \models X \to Y$.

Tervezzünk egy $O(n^2)$ futási idejű algoritmust, melynek bemenete az (R, F) séma, kimenete funkcionális függőségek F-fel ekvivalens G halmaza, amelynek már nincsenek külső attribútumai.

12-2. Minimális fedő konstrukciójában az eliminációs lépések sorrendje fontos

A Minimális-fedés eljárás során kétféle módon változtattuk meg a funkcionális függőségek halmazát: egyrészt a redundáns függőség elhagyásával, másrészt a redundáns attribútum elhagyásával a függőségek bal oldaláról. Ha először a második módot alkalmazzuk, amíg csak lehet, majd azután az elsőt, amíg több alkalmazás nem lehet, akkor valóban minimális fedést kapunk a 12.6. állítás szerint. Bizonyítsuk be, hogy ha először az első módot alkalmazzuk, amíg csak lehet, majd azután a másodikat, amíg több alkalmazás nem lehet, akkor nem feltétlenül kapunk minimális fedést.

12-3. BCNF részséma

Bizonyítsuk be, hogy az alábbi probléma coNP-teljes: Adott R relációs séma F funkcionális függőségekkel és $S \subset R$, eldöntendő, hogy $(S, \pi_S(F))$ BCNF-ben van-e.

12-4. 3NF eldöntése nehéz

Adott (R, F) séma, ahol F funkcionális függőségek rendszere.
A ***k-méretű kulcs feladat*** a következő: adott a k természetes szám, döntsük el, hogy létezik-e legfeljebb k méretű kulcs.
A ***prím attribútum feladat*** a következő: adott $A \in R$, döntsük el, hogy prím attribútum-e.

a. Bizonyítsuk be, hogy a k-méretű kulcs probléma NP-teljes. *Útmutatás.* Vezessük vissza rá a csúcs fedés problémát.

b. Bizonyítsuk be, hogy a prím attribútum probléma NP-teljes, azzal, hogy visszavezetjük rá a k-méretű kulcs problémát.

c. Bizonyítsuk be, hogy annak eldöntése, hogy az (R, F) séma, ahol F funkcionális függőségek rendszere 3NF-ben van-e, NP-teljes. Vezessük vissza rá a prím attribútum problémát.

12-5.* FÜGGŐSÉGI-BÁZIS *futási ideje
Adjuk meg a FÜGGŐSÉGI-BÁZIS algoritmus $O(|M| \cdot |R|^3)$ futási idejű megvalósítását.

Megjegyzések a fejezethez

A relációs adatmodellt Codd [80] vezette be 1970-ben. A funkcionális függőségeket 1972-ben megjelent [78] tanulmányában tárgyalta, axiomatizálásuk Armstrong nevéhez fűződik [19]. A logikai következmény feladatot funkcionális függőségekre Beeri és Bernstein [36], valamint Maier [305] tanulmányozta. Maier ebben a cikkében a minimális fedések különböző lehetséges definícióit, azok kapcsolatát, és előállításuk bonyolultsági kérdéseit is tárgyalja. Általános függőségek közti logikai következmény eldöntésére Maier, Mendelzon és Sagiv találtak eljárást [306]. Beeri, Fagin és Howard bizonyították, hogy az (A1)–(A8) axiómarendszer ellentmondásmentes és teljes. A funkcionális és többértékű függőségekre [38] Yu és Johnson [504] konstruáltak olyan relációs sémát, melyben $|F| = k^n$, és a kulcsok száma $k!$. Békéssy és Demetrovics [47] egyszerű és szép bizonyítást adtak arra, hogy k funkcionális függőségből kiindulva legfeljebb $k!$ kulcs kapható. (Tőlük függetlenül Osborne és Tompa is belátta a felső korlátot, azonban konstrukciót a korlát élességére nem adtak, és eredményüket nem publikálták).

Az Armstrong-relációkat Fagin [123, 124], valamint Beeri, Fagin, Dowd és Statman [37] vezették be és tanulmányozták.

A többértékű függőségeket Zaniolo [507], Fagin [122] és Delobel [103] egymástól függetlenül fedezték fel.

A normálformák szükségességét Codd vetette fel a frissítési anomáliák tanulmányozásakor [81, 77]. A Boyce–Codd normálformát [79] vezeti be. A harmadik normálforma általunk használt definícióját Zaniolo [508] adta meg. A negyedik normálformát Fagin [122]vezette be. A normálformákra bontás bonyolultsági kérdéseit Lucchesi és Osborne [298], Beeri és Bernstein [36], valamint Tsou és Fischer [469] vizsgálták.

A 12.30. és 12.31. tételek Beeri eredményei [35]. A 12.34. tétel Aho, Beeri és Ullman cikkéből [5] való. A 12.37. és a 12.38. tételek megtalálhatók Abiteboul, Hull és Vianu könyvében, [1] az összekapcsolási függőségek axiomatizálhatóságáról Petrov [367] mutatta ki, hogy nem lehetséges.

Az elágazó függőségeket Demetrovics, Katona és Sali vezette be, tanulmányozták Armstrong-relációk létezését és a minimális Armstrong-relációk méretét. [104, 105, 106, 406]. Az elágazó függőségek nem axiomatizálhatósága Katona nem publikált eredménye (ICDT'92 Berlin, meghívott előadás).

Numerikus függőségek axiomatizálásának lehetőségét Grant és Minker tárgyalja [173, 174].

A fejezetben szereplő fogalmak jó összefoglalása és bevezetése található Abiteboul, Hull és Vianu [1], Ullman [473] és Thalheim [464] könyveiben.

A fejezet témakörében magyar nyelven Békéssy és Demetrovics [48], Gajdos Sándor [144], Garcia-Molina, Ullman és Widom [147, 474] könyveit, Benczúr András, Kiss Attila és Márkus Tibor cikkét [41], valamint Békéssy András és Demetrovics János megjelenőben lévő tankönyvét [49] ajánljuk.

VI. ALKALMAZÁSOK

Bevezetés

Ez a rész öt fejezetből áll.

A *tizenharmadik* fejezet az informatika egyik legdinamikusabbban fejlődő területének, a bioinformatikának az alapvető algoritmusait foglalja össze.

A *tizennegyedik* fejezet témája az ember-gép kapcsolatok problémákban gazdag területének az a része, ahol ember és gép együttműködéséből lényegesen nagyobb teljesítmény adódik, mint amire külön akár az ember, akár a gép képes lenne.

A *tizenötödik fejezet* a számítógépi grafika algoritmusait foglalja össze.

A *tizenhatodik fejezet* az informatika és a térképészet határterületének, a térinformatikának alapfogalmaival és legfontosabb algoritmusaival ismerteti meg az Olvasót.

A *tizenhetedik fejezet* a numerikus matematika alkalmazásával, a gépi számítások eredményeinek megjelenítésével és a számolásigényes tudományos problémák megoldásával együtt foglalkozó terület első magyar nyelvű összefoglalását nyújtja az Olvasóknak. A fejezet szerzői keresztapaként a *tudományos számítások* nevet javasolják a témakörnek.

13. Bioinformatika

Ebben a fejezetben karaktersorozatok, fák, sztochasztikus nyelvtanok biológiai célú elemzésével foglakozunk.

Az itt bemutatásra kerülő algoritmusok a bioinformatika legfontosabb algoritmusai, számos szoftvercsomag alapját képezik.

13.1. Algoritmusok szekvenciákon

Ebben az alfejezetben olyan dinamikus programozási algoritmusokkal fogunk foglalkozni, amelyek véges karaktersorozatokon – melyeket a biológiában szokásos módon ***szekvenciáknak*** nevezünk – működnek. A dinamikus programozás minden esetben azon alapszik, hogy a szekvenciák prefixeinek a feldolgozásával jutunk el a teljes szekvenciákon szükséges számolások elvégzéséhez.

13.1.1. Két szekvencia távolsága lineáris résbüntetés mellett

Az információt hordozó DNS molekulák a sejt kettéosztódása előtt kettőződnek, az eredeti molekulával megegyező két molekula jön létre. Biokémiai szabályozás hatására mindkét utódsejtbe egy-egy DNS szál kerül, így az utódsejtek mindegyike tartalmazza a teljes genetikai információt. Azonban a DNS replikálódása nem tökéletes, véletlen mutációk hatására a genetikai információ kissé megváltozhat. Így egy egyed utódai között variánsok, mutánsok állnak elő, amelyekből az évmilliók alatt új fajok alakulnak ki.

Legyen adott két szekvencia. A kérdés az, hogy a két szekvencia mennyire rokon – azaz mennyi idő telt el a szétválásuk óta –, illetve milyen mutációk sorozatával lehet leírni a két szekvencia evolúciós történetét.

Tegyük fel, hogy az egyes mutációk egymástól függetlenek, így egy mutációsorozat valószínűsége az egyes mutációk valószínűségének a szorzata. Az egyes mutációkhoz súlyokat rendelünk, a nagyobb valószínűségű mutációk kisebb, a kisebb valószínűségű mutációk nagyobb súlyt kapnak. Egy jó választás a valószínűség logaritmusának a mínusz egyszerese. Ekkor egy mutációsorozat súlya az egyes mutációk súlyainak az összege. Feltételezzük, hogy egy mutációs változás és a megfordítottja ugyanakkora valószínűséggel fordul elő. Így a két szekvencia közös ősből való leszármazása helyett azt kell vizsgálni, hogyan alakulhatott ki az egyik szekvencia a másikból. A ***minimális törzsfejlődés*** elvét feltételezve, azt a minimális súlyú mutációsorozatot keressük, amely az egyik szekvenciát a másikba

alakítja. Fontos kérdés, hogy hogyan lehet egy minimális súlyú mutációsorozatotés végig egybe (lehet, hogy több minimális értékű sorozat van) gyorsan megkeresni. A naiv algoritmus megkeresi az összes lehetséges mutációsorozatot, és kiválasztja közülük a minimális súlyút. Ez nyilvánvalóan nagyon lassú, mert a lehetséges sorozatok száma exponenciálisan nő a szekvenciák hosszával.

Legyen Σ szimbólumok véges halmaza, Σ^* jelölje a Σ feletti véges hosszú szavak halmazát. Egy A szó első n betűjéből álló szót A_n jelöli, az n-edik karaktert pedig a_n. Egy szón a következő transzformációk hajthatók végre:

- Egy a szimbólum beszúrása egy szóba egy adott i pozíció előtt. Ezt a transzformációt $a \leftarrow^i -$ jelöli.
- Egy a szimbólum törlése egy szóból egy adott i pozíciónál. Ezt a transzformációt $- \leftarrow^i a$ jelöli.
- Egy a szimbólum cseréje egy b szimbólumra egy adott i pozícióban. Ezt a transzformációt $b \leftarrow^i a$ jelöli.

A transzformációk ***kompozícióján*** azok egymás utáni végrehajtását értjük, és a $\circ$ szimbólummal fogjuk jelölni. A fenti három transzformációnak, és ezek tetszőleges véges kompozícióinak a halmazát τ-val jelöljük. Azt, hogy egy $T \in \tau$ transzformációsorozat az A szekvenciát B-vé transzformálja, $T(A) = B$-vel jelöljük.

Legyen $w : \tau \to \mathbb{R}^+ \cup \{0\}$ egy olyan ***súlyfüggvény,*** hogy ha bármely T_1, T_2 és S transzformációsorozatra

$$T_1 \circ T_2 = S \tag{13.1}$$

teljesül, akkor

$$w(T_1) + w(T_2) = w(S)\ , \tag{13.2}$$

valamint $w(a \leftarrow^i b)$ független i-től. Két szekvencia, A és B transzformációs távolságán az A-t B-be vivő minimális súlyú transzformációk súlyát értjük, azaz

$$\delta(A, B) = \min\{w(T) | T(A) = B\}\ . \tag{13.3}$$

Ha w-ről feltesszük, hogy

$$w(a \leftarrow b) = w(b \leftarrow a)\ , \tag{13.4}$$

$$w(a \leftarrow a) = 0\ , \tag{13.5}$$

$$w(b \leftarrow a) + w(c \leftarrow b) \geq w(c \leftarrow a) \tag{13.6}$$

bármely $a, b, c \in \Sigma \cup \{-\}$ -re, akkor a $\delta(,)$ transzformációs távolság metrika Σ^*-on, így jogos az elnevezés.

$w(,)$ metrikus tulajdonsága miatt elég olyan transzformációsorozatokkal foglalkozni, amelyek során minden pozíció legfeljebb egyszer transzformálódik. A transzformáció sorozatokat a szekvenciák illesztésével ábrázoljuk. Konvenció alapján a felülre írt szekvencia az ősi szekvencia, az alulra írt a leszármazott. Például, az alábbi illesztés azt mutatja, hogy a harmadik és az ötödik pozícióban egy-egy csere történt, az első pozícióban egy beszúrás, a nyolcadikban pedig egy törlés:

```
- A U C G U A C A G
U A G C A U A - A G
```

Az egyes pozíciókban az egymás alá írt szimbólumokat ***illesztett párnak*** hívjuk. A transzformációsorozat súlya az egyes pozíciókban történt mutációk súlyainak az összege. Látható, hogy minden mutációsorozathoz egyértelműen megadható egy illesztés, amely azt ábrázolja, és egy illesztésből a mutációk sorrendjétől eltekintve egyértelműen megadhatók azok a mutációk, amelyeket az illesztés ábrázol. Mivel az összeadás kommutatív, ezért a teljes súly független a mutációk sorrendjétől.

Most megmutatjuk, hogy a lehetséges illesztések száma is exponenciálisan nő a szekvenciák hosszával. A lehetséges illesztések halmazának egy részhalmazát adják azok az illesztések, amelyekben beszúrás és törlés nem szerepel egymás melletti illesztett oszlopban, azaz nem található az illesztésben az alábbi mintázatok egyike sem:

$$\begin{array}{cc} \# & - \\ - & \# \end{array} \qquad \begin{array}{cc} - & \# \\ \# & - \end{array} ,$$

ahol # tetszőleges karakter Σ-ból. Az ilyen illesztések száma $\binom{|A|+|B|}{|A|}$, mivel bijekció van az ilyen illesztések halmaza és az olyan C szavak között, amelyek pontosan a két szekvencia betűiből állnak, és mind A, mind B összes betűje növekvő sorrendben helyezkedik el C-ben. Ha például $|A| = |B| = n$, akkor $\binom{|A|+|B|}{|A|} = \Theta(2^{2n}/\sqrt{n})$.

Optimális illesztésnek nevezzük azt az illesztést, amelyhez a hozzárendelt mutációsorozat súlya minimális. Jelöljük $\alpha^*(A_i, B_j)$-vel A_i és B_j optimális illesztéseinek a halmazát, $w(\alpha^*(A_i, B_j))$-vel jelöljük az ezen illesztésekhez tartozó mutációsorozatok súlyát.

A gyors algoritmus ötlete az, hogy ha ismerjük $w(\alpha^*(A_{i-1}, B_j))$-t, $w(\alpha^*(A_i, B_{j-1}))$-et, valamint $w(\alpha^*(A_{i-1}, B_{j-1}))$-et, akkor ebből konstans idő alatt kiszámítható $w(\alpha^*(A_i, B_j))$. Tekintsük ugyanis A_i és B_j egy optimális illesztésének az utolsó illesztett párját. Ezt elhagyva vagy A_{i-1} és B_j vagy A_i és B_{j-1} vagy A_{i-1} és B_{j-1} egy optimális illesztését kapjuk, rendre attól függően, hogy az utolsó pár egy törlést, beszúrást vagy cserét ábrázol. Így

$$\begin{aligned} w(\alpha^*(A_i, B_j)) = \quad & \min\{w(\alpha^*(A_{i-1}, B_j)) + w(- \leftarrow a_i) \ ; \\ & w(\alpha^*(A_i, B_{j-1})) + w(b_i \leftarrow -) \ ; \\ & w(\alpha^*(A_{i-1}, B_{j-1})) + w(b_i \leftarrow a_i)\} \ . \end{aligned} \tag{13.7}$$

Az optimális illesztéshez tartozó súlyokat egy úgynevezett dinamikus programozási mátrix, D segítségével lehet megadni. A D mátrix $d_{i,j}$ eleme $w(\alpha^*(A_i, B_j))$-t tartalmazza. Ez egy n és egy m hosszúságú szekvencia összehasonlítása esetén egy $(n+1)(m+1)$-es táblázat, a sorok és oszlopok indexelése 0-tól n-ig, illetve m-ig megy. A kezdeti feltételek a nulladik sorra és oszlopra:

$$d_{0,0} = 0 \ , \tag{13.8}$$

$$d_{i,0} = \sum_{k=1}^{i} w(- \leftarrow a_k) \ , \tag{13.9}$$

$$d_{0,j} = \sum_{l=1}^{j} w(b_l \leftarrow -) \ . \tag{13.10}$$

A táblázat belsejének a kitöltése a (13.7) képlet alapján történik.

A táblázat kitöltésének az ideje $\Theta(nm)$. A táblázat segítségével megkereshetjük az

összes optimális illesztést. Egy optimális illesztés megkereséséhez a jobb alsó sarokból kiindulva mindig a minimális értéket adó előző pozíciót választva – több lehetőség is adódhat – visszafelé haladunk a bal felső sarokig, minden egyes lépés az optimális illesztésnek egy illesztett párját adja meg. A $d_{i,j}$ pozícióból fölfelé lépés az adott pozícióban való törlést, a balra lépés az adott pozícióban való beszúrást jelent, az átlón való haladás pedig vagy az adott pozícióban való szubsztitúciót jelzi, vagy azt mutatja, hogy az adott pozícióban nem történt mutáció, attól függően, hogy a_i nem egyezik meg b_j-vel, vagy megegyezik. Az összes optimális illesztések száma exponenciálisan nőhet a szekvenciák hosszával, viszont azok polinomiális időben ábrázolhatók. Ehhez egy olyan irányított gráfot kell készíteni, amelynek csúcsai a dinamikus programozási táblázat elemei, és egy él megy egy v_1 csúcsból v_2-be, ha v_1 minimális értéket ad v_2-nek Ezen a gráfon minden irányított út $d_{n,m}$-ből $d_{0,0}$-ba egy optimális illesztést határoz meg.

13.1.2. Dinamikus programozás tetszőleges résbüntetés mellett

Mivel egy beszúrást ugyanakkora súllyal értékelünk, mint egy törlést, ezeket közös névvel ***réseknek*** szokás nevezni, a hozzá tartozó súlyt pedig ***résbüntetésnek***. Általában egy súlyt szoktak használni minden karakter beszúrására és törlésére. Az alapalgoritmus úgy tekinti a szekvenciák változását, hogy egy hosszú rés kialakulását rövidebb rések egymásutánjának képzeli. Ez biológiailag helytelen, mert tudjuk, hogy egy hosszabb részszekvencia beszúrása vagy törlése egyetlen mutációval is megtörténhet. Így az alapalgoritmus a hosszú réseket túlzott mértékben bünteti. Ez a felismerés motiválta a különböző résbüntető függvények bevezetését a biológiai szekvenciák elemzésében, melyekben egy k hosszúságú rést g_k értékkel büntetünk. Például az

```
- - A U C G A C G U A C A G
U A G U C - - - A U A G A G
```

illesztés súlya $g_2 + w(G \leftarrow A) + g_3 + w(A \leftarrow G) + w(G \leftarrow C)$.

Továbbra is azt a minimális súlyú mutációsorozatot keressük, amely az egyik szekvenciát a másikba alakítja. A dinamikus programozási algoritmus abban különbözik az előző algoritmustól, hogy az illesztés végén állhat egy hosszú rés, így $w(\alpha^*(A_i, B_j))$ kiszámításához nem csak $w(\alpha^*(A_{i-1}, B_j))$, $w(\alpha^*(A_i, B_{j-1}))$ és $w(\alpha^*(A_{i-1}, B_{j-1}))$ ismeretére van szükség, hanem $w(\alpha^*(A_{i-1}, B_{j-1}))$-en kívül minden $w(\alpha^*(A_k, B_j))$, $0 \le k < i$-re és minden $w(\alpha^*(A_i, B_l))$, $0 \le l < j$-re. Az előbbiekhez hasonlóan belátható, hogy

$$\begin{aligned} w(\alpha^*(A_i, B_j)) = \quad & \min\{w(\alpha^*(A_{i-1}, B_{j-1})) + w(b_j \leftarrow a_i)\ , \\ & \min_{0 \le k < i}\{w(\alpha^*(A_k, B_j)) + g_{i-k}\}\ , \\ & \min_{0 \le l < j}\{w(\alpha^*(A_i, B_l)) + g_{j-l}\}\ . \end{aligned} \tag{13.11}$$

Továbbra is egy $(n+1)(m+1)$-es dinamikus programozási táblázat kitöltésével lehet kiszámítani egy n és egy m hosszúságú szekvencia optimális illesztését. A kezdeti feltételek a nulladik sorra és oszlopra:

$$d_{0,0} = 0\ , \tag{13.12}$$

$$d_{i,0} = g_i\ , \tag{13.13}$$

$$d_{0,j} = g_j\ . \tag{13.14}$$

A táblázat $d_{i,j}$ elemének a kiszámításához $\Theta(i+j)$ időre van szükség, így a táblázat kitöltése – és így egy optimális illesztés súlya – $\Theta(nm(n+m))$ idő alatt van meg. Egy optimális illesztést, hasonlóan az előző algoritmushoz, a $d_{n,m}$-ből a minimális értékeket adó pozíciókon át a $d_{0,0}$-ig vezető út definiál.

Ennek az algoritmusnak a futási ideje tehát a szekvenciák hosszának a köbével arányos, ha kettő, nagyjából egyforma hosszú szekvenciát hasonlítunk össze. Ha a résbüntető függvényre bizonyos megkötéseket teszünk, akkor lehetőség van gyorsabb algoritmusok konstruálására. A következő két pontban ilyen algoritmusokra mutatunk példát.

13.1.3. Gotoh algoritmusa affin résbüntetéssel

Egy résbüntető függvényt ***affin függvénynek*** hívunk, ha

$$g_k = ku + v, \quad u \geq 0, \quad v \geq 0 \,. \tag{13.15}$$

Ilyen résbüntető függvényt alkalmaz a Gotoh-algoritmus, amelynek a futási ideje $\Theta(nm)$. Emlékeztetőül, az előbbi gyors algoritmusban

$$d_{i,j} = \min\{d_{i-1,j-1} + w(b_j \leftarrow a_i);\ p_{i,j};\ q_{i,j}\} \,, \tag{13.16}$$

ahol

$$p_{i,j} = \min_{1 \leq k < i}\{d_{i-k,j} + g_k\} \,, \tag{13.17}$$

$$q_{i,j} = \min_{1 \leq l < j}\{d_{i,j-l} + g_l\} \,. \tag{13.18}$$

Az algoritmus a következő átindexelésen alapul:

$$\begin{aligned} p_{i,j} &= \min\{d_{i-1,j} + g_1,\ \min_{2 \leq k < i}\{d_{i-k,j} + g_k\}\} \\ &= \min\{d_{i-1,j} + g_1,\ \min_{1 \leq k < i-1}\{d_{i-1-k,j} + g_{k+1}\}\} \\ &= \min\{d_{i-1,j} + g_1,\ \min_{1 \leq k < i-1}\{d_{i-1-k,j} + g_k\} + u\} \\ &= \min\{d_{i-1,j} + g_1,\ p_{i-1,j} + u\} \,. \end{aligned} \tag{13.19}$$

És hasonlóan

$$q_{i,j} = \min\{d_{i,j-1} + g_1,\ q_{i,j-1} + u\} \,. \tag{13.20}$$

Így $p_{i,j}$ és $q_{i,j}$ konstans idő alatt kiszámítható, ezekből pedig $d_{i,j}$ is konstans idő alatt kiszámítható, a táblázat minden elemére. Így az algoritmus futási ideje $\Theta(nm)$ marad, és maga az algoritmus csak egy konstans faktorral lesz lassabb, mint az alapalgoritmus, amely nem enged meg hosszú réseket egyetlen mutációs lépésben.

13.1.4. Konkáv résbüntetés

Az affin résbüntető függvény helyességére nincs semmilyen biológiai magyarázat, elterjedt használatát (például Clustal-W) a hozzá tartozó gyors algoritmusnak köszönheti. Meg lehet szabni azonban egy sokkal reálisabb feltételt a résbüntető függvényre, amelyre Gotoh algoritmusánál egy kicsit lassabb, de az általános résbüntető köbös idejű algoritmusnál mégis

lényegesen gyorsabb algoritmus létezik.

Egy résbüntető függvényt ***konkávnak*** nevezünk, ha bármely i-re $g_{i+1} - g_i \leq g_i - g_{i-1}$. Magyarán: a rések növekedését egyre kevésbé büntetjük. Nyilván, ha a függvény egy adott csúcstól csökkenni kezd, akkor elég nagy résekre negatív súlyokat kapunk. Ezt elkerülendő, a függvényről még fel szokták tenni, hogy szigorúan monoton növekszik, bár algoritmikai szempontból ennek semmi jelentősége nincs. Empirikus adatok alapján, ha két szekvencia d PAM egység óta evolválódott, akkor egy q hosszúságú beszúrás vagy törlés súlya

$$35.03 - 6.88 \log d + 17.02 \log q \ , \tag{13.21}$$

ami szintén konkáv függvény. (Egy PAM egység az az időtartam, amely alatt a szekvencia 1%-a változik meg.)

Konkáv résbüntető függvényekre létezik $O(nm(\lg n + \lg m))$ idejű algoritmus. Ez az algoritmus az úgynevezett előretekintő algoritmusok családjába tartozik. Az Előretekintő algoritmus tetszőleges résbüntető függvény esetén a következő módon számítja ki a dinamikus programozási táblázat i-edik sorát.

Előretekintő(i, m, q, d, g, w, a, b)

```
 1  for j ← 1 to m
 2      do q1[i, j] ← d[i, 0] + g[j]
 3         b[i, j] ← 0
 4  for j ← 1 to m
 5      do q[i, j] ← q1[i, j]
 6         d[i, j] ← min[q[i, j], p[i, j], d[i − 1, j − 1] + w(b_j ← a_i)]
 7       ▷ Ennél a lépésnél feltesszük, hogy p[i, j]-t és d[i − 1, j − 1]-et már kiszámoltuk.
 8      for j ← j1 to m                                          ▷Belső ciklus.
 9          do if q1[i, j1] > d[i, j] + g[j1 − j]
10                then q1[i, j1] ← d[i, j] + g[j1 − j]
11                     b[i, j1] ← j
```

A pszeudokódban $g[\,]$ a résbüntető függvény, b pedig egy pointer, amelynek a jelentése a későbbiekben derül ki. A hatodik sorban feltesszük, hogy $p[i, j]$-t és $d[i - 1, j - 1]$-et már kiszámoltuk. Könnyen belátható, hogy az Előretekintő algoritmus pontosan ugyanazokat az összehasonlításokat végzi, mint az eredeti dinamikus programozási algoritmus, csak más sorrendben. Míg az eredeti algoritmus a sor j-edik pozíciójába érkezve megnézi, hogy mi lehet az optimális $q_{i,j}$, addig az Előretekintő algoritmus a j-edik pozícióba érve $q_{i,j}$-t már meghatározta, viszont megnézi, hogy ennek a pozíciónak az értékéhez a konkáv résbüntető értékeket hozzáadva, melyik q_{i,j_1}-re lehet optimális (belső ciklus). Az aktuális legjobb jelölt $q_1[i, j_1]$-ben tárolódik, és mire a j_1-edik cellához érünk, minden szükséges összehasonlítást elvégeztünk. Tehát az Előretekintő algoritmus nem gyorsabb az eredeti algoritmusnál, de a koncepció segít a gyorsításban.

A gyorsítás alapja a következő észrevétel.

13.1. lemma. *Legyen j az aktuális cella. Ha*

$$d_{i,j} + g_{j_1 - j} \geq q_1[i, j_1] \ , \tag{13.22}$$

akkor minden $j_2 > j_1$-re

$$d_{i,j} + g_{j_2 - j} \geq q_1[i, j_2] \ . \tag{13.23}$$

Bizonyítás. A feltételből következik, hogy van olyan $k < j < j_1 < j_2$, melyre

$$d_{i,j} + g_{j_1-j} \geq d_{i,k} + g_{j_1-k} \ . \tag{13.24}$$

Adjunk az egyenlőtlenség mindkét oldalához $(g_{j_2-k} - g_{j_1-k})$-t:

$$d_{i,j} + g_{j_1-j} + g_{j_2-k} - g_{j_1-k} \geq d_{i,k} + g_{j_2-k} \ . \tag{13.25}$$

A konkáv résbüntető függvény tulajdonságából

$$g_{j_2-j} - g_{j_1-j} \geq g_{j_2-k} - g_{j_1-k} \ , \tag{13.26}$$

és ebből átrendezve és a (13.25) egyenletet felhasználva :

$$d_{i,j} + g_{j_2-j} \geq d_{i,j} + g_{j_1-j} + g_{j_2-k} - g_{j_1-k} \geq d_{i,k} + g_{j_2-k} \ , \tag{13.27}$$

amiből már közvetlenül adódik az állítás. ■

Tehát az algoritmus ötlete az, hogy binárisan keressük meg azt a pozíciót, ahonnan felesleges összehasonlításokat végezni, mert az adott pozíció biztosan nem adhat optimális $q_{i,j}$ értéket azon túl. Ez azonban még mindig nem elég az algoritmus kívánt gyorsításához, legrosszabb esetben ugyanis még így is $\Theta(m)$ értéket kellene felülírni minden egyes pozíciónál. Az előző észrevétel egy következménye azonban már elvezet a kívánt gyorsításhoz.

13.2. következmény. *Mielőtt a j-edik cella jelölteket küldene előre az algoritmus belső ciklusában, a j-edik pozíciótól a pozíciók blokkokra oszthatók, minden blokknak a b pointere ugyanakkora, és ezek a pointer értékek blokkonként csökkennek balról jobbra haladva.*

Az algoritmus pszeudokódja a következő.

ELŐRETEKINTŐ-BINÁRISKERESŐ(i, m, q, d, g, w, a, b)

```
 1  (pn[i], p[i, 0], b[i, 0]) ← (0, m, 0)
 2  for j ← 1 to m
 3      do q[i, j] ← q[i, b[i, pn[i]]] + g[j − b[i, pn[i]]]
 4         d[i, j] ← min[q[i, j], p[i, j], d[i − 1, j − 1] + w(b_j ← a_i)]
 5         ▷ Ennél a lépésnél feltesszük, hogy p[i, j]-t és d[i − 1, j − 1]-et már kiszámoltuk.
 6         if p[i, pn[i]] = j
 7           then pn[i] ← pn[i] − 1
 8         if j + 1 < m és d[i, b[i, 0]] + g[m − b[i, 0]] > d[i, j] + g[m − j]
 9           then pn[i] ← 0
10                b[i, 0] ← j
11         else if j + 1 < m
12                then Y ← max_{0≤X≤pn[i]}{X|d[i, b[i, X]] + g[p[i, X] − b[i, X]]
                           ≤ p[i, j] + g[p[i, X] − j]}
13         if d[i, b[i, Y]] + g[p[i, Y] − b[i, Y]] = p[i, j] + g[p[i, X] − j]
14           then (pn[i], b[i, Y]) ← (Y, j)
15           else E ← p[i, Y]
17         if Y < pn[i]
18           then B ← p[i, Y + 1] − 1
19           else B ← j + 1
20                pn[i] ← pn[i] + 1
21                b[i, pn[i]] ← j
22                p[i, pn[i]] ← max_{B≤X≤E}{X|d[i, j] + g[X − j] ≤ d[i, b[i, Y]] + g[x − b[i, Y]]}
```

Az algoritmus a következőképpen működik. minden sorra fenn kell tartani egy változót, amely az aktuális blokkok számát tárolja. Minden blokkra csak egy pointert tartunk fenn, kell készíteni a blokkok végeinek egy csökkenő listáját, valamint ezzel párhuzamosan a blokkok közös pointereinek egy listáját. A lista hossza értelemszerűen a blokkok számával egyezik meg. Ezután bináris kereséssel megkeressük azt az utolsó pozíciót, amelyre az aktuális pozíció még optimális jelöltet ad. Ezt úgy tesszük meg, hogy először kiválasztjuk a blokkot, majd a blokkon belül keressük meg a pozíciót. A blokkok száma eggyel több, mint ahány blokk maradt az új blokkvég után. A bináris keresés módjából adódóan ezt már ismerjük. Végül a listát felülírjuk. Elég egyetlen értéket felülírni mindkét listában, a pozíciók listájának új, utolsó értéke a bináris kereséssel megkeresett pozíció, a pointerek listájának az új utolsó értéke pedig az új blokk pointerértéke, azaz az aktuális pozíció. Így minden egyes pozícióban konstans mennyiségeket írunk felül. A leginkább időigényes rész a bináris keresés, ez $O(\lg m)$ idő minden egyes cellára.

Az oszlopok esetében teljesen hasonlóan járunk el. Egyetlen rész maradt vissza, ami a teljes algoritmus pontos leírásához kell. Ha soronként töltjük fel a dinamikus programozási táblázat értékeit, akkor minden egyes pozícióban más és más oszlop blokkrendszerét változtatjuk meg. De ez természetesen megtehető, ha minden egyes blokkrendszert tárolunk. Így az algoritmus teljes futási ideje valóban $O(nm(\lg n + \lg m))$.

13.1.5. Két szekvencia hasonlósága, Smith–Waterman algoritmus

Két biológiai szekvenciának nem csak a távolságát, de a hasonlóságát is mérni tudjuk. Két karakter hasonlóságára, $S(a,b)$-re, a leggyakrabban használt hasonlósági függvény az úgynevezett ***log-odds***:

$$S(a,b) = \log\left(\frac{\Pr\{a,b\}}{\Pr\{a\}\Pr\{b\}}\right), \tag{13.28}$$

ahol $\Pr\{a,b\}$ a két karakter együttes valószínűsége, $\Pr\{a\}$ és $\Pr\{b\}$ pedig a marginális valószínűségek. A hasonlóság pozitív, ha $\Pr\{a,b\} > \Pr\{a\}\Pr\{b\}$, egyébként meg negatív. A hasonlósági értékeket empirikus adatokból határozzák meg, aminosavak esetében a legismertebbek a PAM és a BLOSUM hasonlósági mátrixok.

Ha a beszúrásokat és törléseket negatív értékekkel büntetjük, akkor az előbbi alfejezetekben megadott algoritmusok hasonlóságokkal ugyanúgy működnek, csak minimalizálás helyett maximalizálni kell.

Hasonlóság alapú értékeléssel azonban lehet egy speciális problémát definiálni, a maximális lokális hasonlóság problémáját, vagy más néven a lokális szekvenciaillesztés problémáját: adott két szekvencia, egy hasonlósági mátrix és egy résértékelési séma, a feladat az, hogy adjuk meg a két szekvencia két olyan részstringjét, amelyek hasonlósága maximális, valamint adjunk meg egy ilyen illesztést is. A ***részstring*** egy szekvencia szomszédos karakterekből álló részszekvenciáját értjük. A probléma biológiai motivációja az, hogy a biológiai szekvenciák egyes részei lassan, míg más részek gyorsabban evolválódnak. A lokális szekvenciaillesztés a legkonzervatívabb részt találja meg, a legelterjedtebb felhasználása az adatbázisokban való keresés. Ugyanis a lokális illesztésből származó hasonlósági érték hatékonyabban tudja elkülöníteni a homológ és nem homológ szekvenciákat, ugyanis a statisztikát nem rontják le a változó szakaszokból adódó negatív értékek.

A Smith-Waterman algoritmus a következőképpen működik: A kezdeti feltételek a nulladik sorra és oszlopra:

$$d_{0,0} = d_{i,0} = d_{0,j} = 0\,. \tag{13.29}$$

A dinamikus programozási táblázat kitöltése lineáris résbüntetés mellett:

$$d_{i,j} = \max\{0;\ d_{i-1,j-1} + S(a_i, b_j), d_{i-1,j} + g;\ d_{i,j-1} + g\}\,. \tag{13.30}$$

Itt a g résbüntetés most negatív érték. A táblázat kitöltése után megkeressük a táblázat legnagyobb értékét, ez lesz a lokálisan legjobb illesztés hasonlósága, majd innen az optimális értékeket adó cellákon át haladunk vissza a 0 értékig, ez a visszakeresés adja meg az illesztést, hasonlóan a távolság alapú módszerekhez.

Könnyű bizonyítani, hogy az így megadott illesztés lokálisan a legjobb lesz: ha az illesztést a vége felől ki tudnánk úgy terjeszteni, hogy az illesztés értéke az aktuális illesztésnél nagyobb legyen, akkor lenne nagyobb érték a dinamikus programozási táblázatban. Ha az illesztés elejét lehetne megtoldani egy pozitív értékű illesztéssel, akkor a dinamikus programozási táblázatban nem 0 szerepelne a lokális illesztés kezdeténél.

13.1.6. Többszörös szekvenciaillesztés

Kettőnél több szekvencia egyszerre történő illesztését először Sankoff ismertette. Az első alkalmazása egy lokális optimalizálás háromszoros illesztéssel volt, mellyel egy bináris fa

belső csúcsaira adtak meg konszenzus szekvenciákat. Mára a bioinformatika egyik kulcskérdésévé vált a gyors és adekvát többszörös szekvencia illesztés, Dan Gusfield a bioinformatika Szent Gráljának nevezi. Ma a többszörös illesztés egyformán elterjedt az adatbázisokban való keresésre, valamint evolúciós leszármazások vizsgálatára. Segítségével meg lehet találni egy szekvencia család konzervatív régióit, azokat a pozíciókat, amelyek az adott fehérjecsalád funkcionális tulajdonságát kialakítják. Arthur Lesk szavaival: *Amit két homológ szekvencia suttog, azt egy többszörös illesztés hangosan kiáltja.*

A többszörös illesztés egymás alá írt k-asait illesztett k-asoknak hívjuk. A többszörös illesztés dinamikus programozási algoritmusa egyszerű általánosítása a páronkénti illesztés algoritmusának: k szekvencia illesztéséhez egy k dimenziós dinamikus programozási táblázatot kell kitölteni. A táblázat minden egyes elemének a kiszámításához ismerni kell azokat az elemeket, amelyeknek valahány indexe eggyel kisebb, ha nem engedünk meg többszörös réseket, és a koordinátatengelyekkel párhuzamos hipersíkok minden kisebb indexű elemét, ha többszörös réseket megengedünk. Így ezen algoritmusok memóriaigénye k darab, egyenként n hosszúságú szekvencia esetén $\Theta(n^k)$, számolásigénye pedig $\Theta(2^k n^k)$, ha lineáris résbüntetést alkalmazunk, és $\Theta(n^{2k-1})$, ha tetszőleges résbüntetést alkalmazunk.

A többszörös szekvencia illesztéssel két alapvető probléma van. Az egyik algoritmuselméleti probléma: a pontos megoldáshoz szükséges idő a szekvenciák számával exponenciálisan nő. Bebizonyították, hogy a többszörös illesztés NP-teljes probléma. A másik metodikai probléma: nem világos, hogyan kell értékelni egy többszörös illesztést, ha több faj leszármazási sorrendjére vagyunk kíváncsiak. Objektív értékelési lehetőség csak akkor adódna, ha ismernénk a leszármazási viszonyokat, ekkor lehetne egy evolúciós fa mentén értékelni egy többszörös illesztést.

Mindkét problémára heurisztikus megoldást ad a fa mentén való iteratív páronkénti illesztés. Ez a módszer először egy úgynevezett ***vezérfát*** állít elő páronkénti távolságokból kiindulva (ilyen fakészítő módszerekkel találkozhatunk például a 13.5 alfejezetben), majd ezt használja fel többszörös illesztésre. Először a fa alapján szomszédos szekvenciákat illeszti, majd a már illesztett szekvencia párokhoz, hármasokhoz stb. illeszti az újabb szekvenciákat, szekvencia párokat, hármasokat, stb. úgy, hogy a már illesztett szekvenciák illesztett k-asait nem lehet megbontani, csak egy csupa rés jelekből álló oszlopot beilleszteni. Így $k-1$ páronkénti illesztéssel kapjuk meg k szekvencia többszörös illesztését. Sokszor a már illesztett szekvenciákat csak egy úgynevezett profillal ábrázolják. Egy profil egy $(|\Sigma|+1)\times l$-es táblázat, ahol l az illesztés hossza. Az egyes oszlopokban az adott pozíciójú illesztett k-asról készült statisztika található. Az egyes értékek azt mutatják, hogy az ábécé adott betűje hány százalékban szerepel az illesztés adott illesztett k-asában. Az oszlop utolsó helyén a rés jel százalékos előfordulása található.

Természetesen a kapott többszörös illesztés felhasználható egy újabb fa készítésére, amiből egy újabb illesztés generálható, és ezt a ciklust addig lehet ismételni, ameddig az újabb iteráció már nem hoz változást az illesztésben. A módszer magyarázata az a feltételezés, hogy a közeli szekvenciák optimális páronkénti illesztése ugyanaz, mint amit az optimális többszörös illesztésből kapunk. A módszer hátránya az, hogy még ha az előbbi feltételezés igaz is, akkor is lehet több egyformán optimális illesztés, és ezek száma is exponenciálisan nőhet a szekvencia hosszával. Például tekintsük az `AUCGGUACAG` és az `AUCAUACAG` szekvenciák alábbi két optimális illesztését:

```
A U C G G U A C A G     A U C G G U A C A G
A U C - A U A C A G     A U C A - U A C A G
```

A páronkénti illesztésben a kettő közül nem tudunk választani, viszont a többszörös illesztésben az egyik már jobb lehet a másiknál. Például ha ezekhez a szekvenciákhoz illesztjük az AUCGAU szekvenciát, akkor a két páronkénti optimális illesztésből kiindulva a következő két, lokálisan optimális illesztést kapjuk:

```
A U C G G U A C A G    A U C G G U A C A G
A U C - A U A C A G    A U C A - U A C A G
A U C G A U - - - -    A U C - G - A U - -
```

melyből a baloldali valóban globálisan optimális, a jobboldali azonban csak lokálisan optimális.

Így az iteratív illesztés csak egy lokális optimumot határoz meg. További hátránya ennek a heurisztikus eljárásnak az, hogy nem képes egy felső becslést adni arra nézve, hogy a kapott illesztés súlya legfeljebb hányszorosa az optimális illesztés súlyának. Mindezek ellenére ez a módszer a leginkább elterjedt a gyakorlatban, mert viszonylag egyszerű és gyors, és általában biológiailag helyes eredményt ad.

Meg kell említeni egy új heurisztikus módszert a többszörös illesztésre, amelyik nem dinamikus programozáson, hanem mohó algoritmuson alapszik. A DiAlign rés-mentes homológ részeket keres szekvenciák páronkénti összehasonlításával. A kapott részstringek résmentes illesztéseit hívjuk kiátlóknak, hiszen a dinamikus programozási táblázatban az ezeknek az illesztéseknek megfelelő utak átlósan helyezkednek el. Innen ered a módszer neve is: Diagonal Alignment. Ezután a homológ párokat mohó módon fűzi össze többszörös illesztéssé. Az átlókat egy heurisztikus módszer szerint értékeljük az alapján, hogy mekkora hasonlósági értékeket kap az adott átló, illetve milyen hasonlósági értékű átlókkal nem kompatibilis az adott átló. Két átló akkor nem kompatibilis, ha nem szerepelhetnek egy közös (többszörös) illesztésben. Az átlókat az értékelésük alapján sorrendbe rakjuk, majd kiválasztjuk a legnagyobb értékűt. Ezután töröljük a listából az összes olyan átlót, amely nem kompatibilis a kiválasztott átlóval, majd kiválasztjuk a megmaradt átlók közül a legnagyobb értékűt, és így tovább, amíg van kiválasztható átló. A többszörös illesztést a kiválasztott átlók uniója adja, amely nem feltétlenül fedi le a megadott szekvenciák összes karakterét. Azon karakterek, amely egyetlen kiválasztott átlóban sem fordulnak elő, „nem illeszthető” minősítést kapnak. Bár a módszer hátránya az, hogy néha indokolatlanul nagy réseket tesz a többszörös illesztésbe, mivel egyáltalán nem bünteti a réseket, a DiAlign a gyakorlatban az egyik legjobb heurisztikus algoritmusnak bizonyult.

13.1.7. Memóriaredukció Hirschberg algoritmusával

Ha két szekvenciának csak a távolságát vagy hasonlóságát akarjuk megadni, de egy optimális illesztésüket nem, akkor lineáris vagy affin résbüntetés esetén ezt nagyon könnyű lineáris memóriaigénnyel megtenni. Vegyük észre ugyanis, hogy a dinamikus programozási táblázatban mindig csak a megelőző sorra van szükségünk, a korábbi sorokat elfelejthetjük. Ha azonban egy optimális illesztést is meg akarunk adni, akkor szükségünk van a teljes táblázatra. Ha a táblázatot újra és újra kitöltve adjuk meg az optimális illesztéshez szükséges információkat, akkor meghatározhatjuk ezt lineáris memóriaigénnyel, de ekkor a számolási idő növekszik meg a szekvenciák hosszával köbös méretűre.

Meg lehet adni azonban egy olyan algoritmust is, amely négyzetes időben és lineáris memóriaigénnyel határozza meg két szekvencia optimális illesztését, ez *Hirschberg algo-*

ritmusa, amelyet lineáris résbüntetés és távolság alapú illesztés esetére mutatunk be.

Az algoritmus bemutatásához bevezetjük egy szekvencia tetszőleges szuffixének a jelölését, A^k jelöli az A szekvencia a_{k+1}-gyel kezdődő szuffixét, azaz a $(k+1)$-edik karaktertől a szekvencia végéig terjedő stringet.

Hirschberg algoritmusa először $A_{[|A|/2]}$-re és B-re hajt végre egy dinamikus programozási algoritmust, a fentebb vázolt lineáris memóriaigénnyel (azaz mindig csak az aktuális és az előző sor értékeit tárolja), valamint egy hasonló dinamikus programozási algoritmust az $A^{[|A|/2]}$ megfordítottján és a B szekvencia megfordítottján.

A két dinamikus programozás alapján tudjuk, hogy mi $A_{[|A|/2]}$ és B tetszőleges prefixe optimális illesztésének az értéke, valamint $A^{[|A|/2]}$ és B tetszőleges szuffix optimális illesztésének az értéke. Ebből már meg tudjuk mondani, hogy mi lesz A és B optimális illesztésének az értéke,

$$\min_j \left\{ w(\alpha^*(A_{[|A|/2]}, B_j)) + w(\alpha^*(A^{[|A|/2]}, B^j)) \right\} , \tag{13.31}$$

és a számolásból adódik, hogy az optimális illesztésben $A_{[|A|/2]}$ B_j azon prefixével van illesztve, melyre

$$w(\alpha^*(A_{[|A|/2]}, B_j)) + w(\alpha^*(A^{[|A|/2]}, B^j)) \tag{13.32}$$

minimális.

Mivel a lineáris memóriaigényű dinamikus programozásban is ismerjük a dinamikus programozási táblázat utolsó előtti sorát, meg tudjuk mondani, hogy $a_{[|A|/2]}$ és $a_{[|A|/2]+1}$ illesztve van-e B szekvencia valamely karakterével, vagy az optimális illesztésben ezen karakterek törlődtek. Az is megállapítható a dinamikus programozási táblázat utolsó két-két sorából, hogy történt-e a B szekvencia valamely karaktereinek a beszúrása $a_{[|A|/2]}$ és $a_{[|A|/2]+1}$ közé.

Így az optimális illesztésnek legalább két oszlopát határoztuk meg. Ezután $A_{[|A|/2]-1}$-re és B fennmaradó prefixére valamint $A^{[|A|/2]+1}$-re és B fennmaradó szuffixére ugyanígy járunk el, azaz mindkét szekvenciapárra két lineáris memóriaigényű dinamikus programozást végzünk el. Ennek eredményeképpen az A szekvencia negyedénél és háromnegyedénél kapunk meg az optimális illesztésből minimum két-két oszlopot. A következő iterációban a negyedelt szekvenciákra végezzük el a fenti eljárást, és ezt folytatjuk addig, amíg az optimális illesztés összes oszlopát meg nem kapjuk.

Nyilvánvaló, hogy a memóriaigény a szekvenciák hosszával csak lineárisan nő. Megmutatjuk, hogy a számolási igény továbbra is $\Theta(nm)$, ahol n és m a két szekvencia hossza. Ez abból adódik, hogy minden egyes iterációban legfeljebb feleannyit számolunk, mint az előző iterációban. Ugyanis egy A és B szekvenciapárra egy lépésben $|A| \times |B|$ mennyiséget számolunk, a következő lépésben viszont csak $(|A|/2) \times j^* + (|A|/2) \times (|B| - j^*)$ mennyiséget, ahol j^* az a pozíció, melyre a (13.31) képletben minimális értéket kapunk. Így a teljes számolási igény

$$nm \times \left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right) = \Theta(nm) . \tag{13.33}$$

13.1.8. Memóriaredukció saroklevágással

A dinamikus programozási algoritmus egyre hosszabb és hosszabb részszekvenciák illesztésével jut el a teljes szekvenciák illesztéséig. Ez az algoritmus gyorsabbá tehető, ha sikerül kiszűrni a részszekvenciák olyan rossz illesztéseit, amelyek biztosan nem vezetnek a teljes

szekvenciák egy optimális illesztéséhez. Az ilyen illesztéseket a dinamikus programozási táblázat jobb felső, valamint a bal alsó sarkában található pozíciókból a minimális értékeket adó pozíciókon át a $d_{0,0}$-ba menő irányított utak adják meg, innen ered a technikának az elnevezése.

A legtöbb ilyen algoritmus egy úgynevezett teszt értéket használ. Ez a teszt érték egy előre megadott felső korlátja a két szekvencia evolúciós távolságának. A teszt értéket használó algoritmusok akkor tudják a két szekvencia távolságát kiszámítani, ha a megadott teszt érték valóban nagyobb a szekvenciák távolságánál. Ellenkező esetben az algoritmus nem jut el a jobb alsó sarkig, vagy ha eljut, az itt kapott érték hibás, nem egyezik meg az optimális illesztés súlyával. Így ezek az algoritmusok adatbázisokban való keresésre alkalmasak, amikor egy adott szekvenciához hasonló szekvenciákat kell kigyűjteni egy adatbázisból. A megadott teszt érték az a felső határ, amelyiknél kisebb távolságot adó illesztéseket kell megkeresni az adatbázisból.

Az alábbiakban két algoritmus leírását közöljük. Spouge algoritmusa általánosítása Fickett, valamint Ukkonen algoritmusainak. A másik algoritmus Gusfieldtől származik, amely példa olyan algoritmusra, amely a szekvenciák távolságánál kisebb teszt értéknél is eljut a bal alsó sarkig, de ekkor a kiszámított távolság nagyobb a megadott teszt értéknél, és ez jelzi, hogy a meghatározott távolság valószínűleg pontatlan.

Spouge algoritmusa csak azokat a $d_{i,j}$ mátrix elemeket számolja ki, amelyekre teljesül, hogy

$$d_{i,j} + |(n-i) - (m-j)|g \leq t \,, \tag{13.34}$$

ahol t a teszt érték, g az rések büntetése (hosszú rések nincsenek megengedve), n és m pedig a szekvenciák hossza. Az ötlete az algoritmusnak az, hogy bármely út $d_{i,j}$-ből $d_{n,m}$-be legalább $|(n-i) - (m-j)| \times g$ értékkel növeli meg az illesztés súlyát. Így, ha t legalább akkora, mint a szekvenciák távolsága, Spouge algoritmusa pontosan határozza meg a szekvenciák távolságát. Ez az algoritmus általánosítása Fickett, ill. Ukkonen algoritmusainak. Azok az algoritmusok szintén teszt értéket tartalmazó egyenlőtlenségeket használtak, de Fickett algoritmusában az egyenlőtlenség

$$d_{i,j} \leq t \,, \tag{13.35}$$

míg Ukkonen algoritmusában az egyenlőtlenség

$$|i-j|g + |(n-i) - (m-j)|g \leq t \tag{13.36}$$

alakú. Mivel mindkét esetben az egyenlőtlenség bal oldala nem nagyobb, mint Spouge egyenlőtlenségének a bal oldala, ezért ezek az algoritmusok legalább akkora részét számolják ki a dinamikus programozási táblázatnak, mint Spouge algoritmusa. Empirikus eredmények igazolják, hogy Spouge algoritmusa valóban gyorsabb. Az algoritmus kiterjeszthető konkáv résbüntető függvényekre is.

A k-eltérésű globális illesztés probléma Gusfieldtől származik, és a következő kérdést teszi fel: Van-e két szekvenciának olyan illesztése, amelynek a súlya nem nagyobb k-nál? A kérdést megválaszoló algoritmus futási ideje $O(kn)$, ahol n a hosszabb szekvencia hossza. Az algoritmus ötlete az az észrevétel, hogy a $d_{n,m}$-ből $d_{0,0}$-ba vezető, legfeljebb k súlyú utak nem tartalmaznak olyan $d_{i,j}$ pozíciót, amelyre $|i-j| > k/g$. Ekképpen az algoritmus csak azokat a $d_{i,j}$ mátrix elemeket számolja ki, amelyekre $(i-j) \leq k/g$, és figyelmen kívül hagyja a határelemek azon $d_{e,f}$ szomszédait, amelyekre $|e-f| > k/g$. Ha van a két szekvenciának

legfeljebb k súlyú illesztése, akkor $d_{n,m} \leq k$, és $d_{n,m}$ valóban a szekvenciák távolsága, ellenkező esetben $d_{n,m} > k$. Ez utóbbi esetben $d_{n,m}$ nem feltétlenül a szekvenciák távolsága: elképzelhető, hogy van olyan illesztés, amelyre az ezt meghatározó út kilép az $|i - j| \leq k/g$ egyenlőtlenség által definiált sávból, és az illesztés súlya mégis kisebb, mint a meghatározott sávban haladó legkisebb súlyú illesztés.

A sarokvágási technikát kiterjesztették olyan többszörös illesztésekre is, ahol egy illesztett k-ast a ***páronkénti összeg*** sémával értékelünk, azaz

$$SP_l = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} d(p_{i,l},\ p_{j,l}) \ , \tag{13.37}$$

ahol SP_l az l-edik illesztett k-as értékelése, $d(,)$ a $\Sigma \cup \{-\}$ halmazon definiált távolságfüggvény, k az illesztett szekvenciák száma, $p_{i,j}$ pedig a profil i-edik sorának j-edik eleme. Egy szekvencia l-szuffixe az $(l+1)$-edik betűtől a szekvencia végéig terjedő részstring. Jelöljük $w_{i,j}(l,m)$-lel az i-edik és a j-edik szekvencia l- illetve m-szuffixének a távolságát. Carillo és Lipman algoritmusa csak azokat a pozíciókat számolja ki, amelyekre

$$d_{i_1,i_2,\ldots i_n} + \sum_{j=1}^{k-1} \sum_{l=j}^{k} w_{j,l}(i_j, i_l) \leq t \ , \tag{13.38}$$

ahol t a teszt érték. Az algoritmus helyességét az bizonyítja, hogy a szekvenciák még nem illesztett szuffixeinek a páronkénti összeg sémával adott optimális illesztésének a súlya nem lehet kisebb, mint a páronkénti illesztésből adódó súlyok összege. A $w_{i,j}(l,m)$-ek a páronkénti illesztésekből számíthatók, így az algoritmus a gyakorlat számára elfogadható idő alatt ki tudja számolni hat darab 200 hosszúságú szekvencia optimális illesztését.

Gyakorlatok

13.1-1. Mutassuk meg, hogy egy n és egy m hosszúságú szekvencia esetén a lehetséges illesztések száma

$$\sum_{i=0}^{\min(n,m)} \frac{(n+m-i)!}{(n-i)!(m-i)!i!} \ .$$

13.1-2. Adjunk meg egy olyan értékelést és szekvenciák olyan sorozatát, amelyekre az optimális illesztések száma exponenciálisan nő a szekvenciák hosszával.

13.1-3. Adjuk meg Hirschberg algoritmusát többszörös szekvenciaillesztésre.

13.1-4. Adjuk meg Hirschberg algoritmusát affin résbüntető függvények esetén.

13.1-5. Adjuk meg a Smith–Waterman algoritmust affin résbüntetésekre.

13.1-6. Adjuk meg a Spouge-féle saroklevágási technikát affin résbüntetésekre.

13.1-7. Két szekvenciának egy többszörös illesztésükből levezetett páronkénti illesztése a következő: kivágjuk a többszörös illesztésből a két szekvenciára vonatkozó sort, ezeket egymás alá írjuk, majd elhagyjuk a csupa rést tartalmazó sorokat. Adjunk példát három szekvencia olyan optimális illesztésére, melyből valamelyik két szekvenciára levezetett páronkénti illesztés nem a két szekvencia optimális páronkénti illesztése.

13.2. Algoritmusok fákon

Az alább ismertetésre kerülő algoritmusok gyökereztetett fákon dolgoznak. A dinamikus programozás a gyökereztetett részfákon történő visszavezetéssel történik. Mint látni fogjuk, nem csak optimális eseteket határozhatunk meg, hanem ugyanakkora futási idő alatt algebrai kifejezéseket is meghatározhatunk.

13.2.1. A takarékossági elv kis problémája

A takarékossági (parszimónia) elv a biológiai szekvenciák változását minél kevesebb számú mutációval akarja leírni. Az alábbiakban csak cserékkel fogunk foglalkozni, azaz adottak egyenlő hosszú biológiai szekvenciák, és a feladat az, hogy adjuk meg a leszármazási kapcsolataikat a takarékossági elv alapján. Definiálhatjuk a takarékossági elv kis és nagy problémáját. A nagy problémában ismeretlen az evolúciós törzsfa topológiája, amely mentén a szekvenciák evolválódtak, a feladat ennek a topológiának a megkeresése, valamint ezen egy legtakarékosabb evolúciós történet megkeresése. Az így kapott megoldás tehát nem csak lokálisan – az adott topológiára nézve – lesz optimális, hanem globálisan is. Megmutatható, hogy a takarékossági elv nagy problémája NP-teljes probléma.

A kis problémában adott egy gyökeres fa topológia, és a feladat az, hogy keressünk egy legtakarékosabb evolúciós történetet ezen fa mentén. Az így kapott megoldás lokálisan optimális lesz, de nincs garancia a globális optimumra nézve. A szekvenciák minden egyes pozíciójához egymástól függetlenül kereshetjük meg a legtakarékosabb történetet, így elég azt az esetet megoldani, amikor a fa minden egyes levelén csupán egy karakter van megadva.

Ekkor egy evolúciós történetet a fa belső csúcsaihoz rendelt karakterekkel jellemezhetünk. Ha két szomszédos csúcson ugyanazt a karaktert látjuk, akkor a takarékossági elv alapján nem történt mutáció a két csúcsot összekötő él mentén, egyébként meg egy mutáció történt. A naiv algoritmus végignézi az összes lehetséges hozzárendelést, ami nyílván lassú, hiszen a lehetséges címkézések száma exponenciálisan növekszik a fa leveleinek számával.

A dinamikus programozás a részfákon történő visszavezetéssel történik (Sankoff algoritmusa). Most részfán csak azokat a részfákat értjük, amelyek egy belső csúcsot, mint gyökeret tartalmaznak, és minden olyan csúcsot, amely az adott gyökér alatt van. Így a részfák száma pontosan a belső csúcsok számával egyezik meg, és ezért egyértelműen beszélhetünk egy adott pont által definiált részfáról. Feltesszük, hogy egy adott r gyökerű részfa esetén ismerjük r minden t gyerekére és ω karakterre azt, hogy a t gyereke által definiált részfán minimum hány mutáció szükséges, ha a t csúcsban ω karakter található. Jelöljük ezt a számot $m_{t,\omega}$-val. Ekkor

$$m_{r,\omega} = \sum_{t \in D(r)} \min_{\sigma \in \Sigma} \{m_{t,\sigma} + \delta_{\omega,\sigma}\} , \tag{13.39}$$

ahol a $D(r)$ r gyerekeinek a halmaza, Σ a lehetséges karakterek halmaza, $\delta_{\omega,\sigma}$ pedig 1 ha $\omega = \sigma$ és 0 egyébként. A (13.39) képlet helyessége abból adódik, hogy a megvizsgáltuk az összes lehetőséget arra vonatkozóan, hogy r gyerekeihez milyen karaktereket rendelhetünk, és azt már ismerjük, hogy ezen karakterek hozzárendelésekor legalább mennyi szubsztitúció szükséges az adott gyerek alatti részfán.

Az adott fa topológiájához szükséges mutációk száma $\min_{\omega \in \Sigma} m_{R,\omega}$, ahol R a fa gyökere. Egy legtakarékosabb evolúciós történetet a gyökérből visszafelé haladva a minimális értékeket adó karakterek beírásával kaphatunk meg. Ehhez természetesen minden r belső

csúcsra és ω karakterre tárolni kell $m_{r,\omega}$-t.

A minimális mutációk számának meghatározásához $\Theta(n|\Sigma|^2)$ időre van szükség, ezután a legtakarékosabb történet megkeresése $\Theta(n|\Sigma|)$ időt vesz igénybe, ahol n a levelek száma. A teljes evolúciós történet meghatározása $\Theta(nl|\Sigma|^2)$ időt vesz igénybe, ahol l a szekvenciák hossza.

13.2.2. Felsenstein algoritmusa

Felsenstein algoritmusában a bemenő adat DNS szekvenciák többszörös illesztése. Csak azokat a pozíciókat tekintjük, ahol az illesztésben nincs rés. Feltesszük, hogy az egyes pozíciók egymástól függetlenül evolválódtak, így egy evolúciós folyamat valószínűsége az egyes csúcsokon történt események valószínűségeinek a szorzata. Legyen adva egy fa topológiája, amely ábrázolja a szekvenciák leszármazási sorrendjét, valamint egy evolúciós modell, amely minden σ-ra, ω-ra és t-re megmondja, hogy mi annak a valószínűsége, hogy σ karakter ω-vá evolválódik t idő alatt. Ezt $f_{\sigma\omega}(t)$ jelöli. Valamint ismerjük a karakterek egyensúlyi eloszlását, amit π jelöl. A kérdés az, hogy mennyi a fa likelihoodja, azaz a fa valószínűsége egy adott paraméterhalmaz mellett. Egy adott paraméterhalmaz mellett a fa likelihoodjának a kiszámítását egy adott fa topológiáján mutatjuk meg (13.1. ábra). Elég azt megmutatni, hogy hogyan kell a likelihoodot egy pozícióra kiszámítani, a fa teljes likelihoodja a pozíciók likelihoodjainak a szorzata. Az adott pozícióra s_i jelöli az i-edik csúcs karakterét, v_j pedig az j-edik él evolúciós ideje, pontosabban a mutációs ráta és az idő szorzata. A belső pontok állapotait persze általában nem ismerjük, ezért minden lehetséges állapotra összegezni kell:

$$\begin{aligned} L \;=\; & \sum_{s_0}\sum_{s_6}\sum_{s_7}\sum_{s_8} \pi_{s_0} \times f_{s_0 s_6}(v_6) \times f_{s_6 s_1}(v_1) \times f_{s_6 s_2}(v_2) \\ & \times f_{s_0 s_8}(v_8) \times f_{s_8 s_3}(v_3) \times f_{s_8 s_7}(v_7) \times f_{s_7 s_4}(v_4) \times f_{s_7 s_5}(v_5)\ . \end{aligned} \tag{13.40}$$

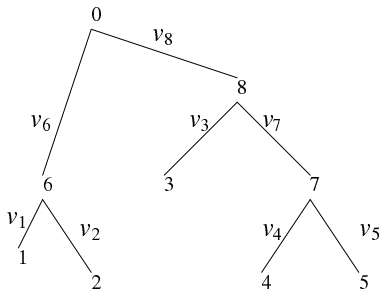
Ha négyelemű ábécét, azaz nukleinsav szekvenciákat tételezünk fel, akkor az összegzés 256 tagból áll, n faj esetén pedig 4^{n-1} tagból, ami könnyen lehet egy túl nagy szám. Szerencsére, ha az adott változótól nem függő szorzótényezőket kihozzuk a szumma jel elé, akkor az összegzés felbomlik egy

$$\begin{aligned} L = & \sum_{s_0} \pi_{s_0} \left\{ \sum_{s_6} f_{s_0 s_6}(v_6)[f_{s_6 s_1}(v_1)][f_{s_6 s_2}(v_2)] \right\} \\ & \times \left\{ \sum_{s_8} f_{s_0 s_8}(v_8)[f_{s_8 s_3}(v_3)] \left(\sum_{s_7} f_{s_8 s_7}(v_7)[f_{s_7 s_4}(v_4)][f_{s_7 s_5}(v_5)] \right) \right\} \end{aligned} \tag{13.41}$$

szorzatra, aminek a számítási igénye jóval kisebb. Vegyük észre, hogy a (13.41) egyenletben a zárójelezések pontosan leírják a fa topológiáját. Minden összegzés külön elvégezhető, és ezeket az összegeket szorozzuk össze, így a számítási idő lecsökken $\Theta(|\Sigma|^2 n)$-re egy pozícióra, a teljes fa likelihoodjának a kiszámítása $\Theta(|\Sigma|^2 nl)$-re, ahol l a pozíciók száma.

Gyakorlatok

13.2-1. Adjunk algoritmust a súlyozott takarékossági elv kis problémájára, azaz amikor az egyes változásokat súlyozzuk, és a változások súlyainak az összegét akarjuk minimalizálni.



13.1. ábra. A fa, amin Felsenstein algoritmusát bemutatjuk. Az élekre írt v-k az éleken eltelt evolúciós időt jelölik.

13.2-2. Az egyes genomok géntartalma különbözik, egy adott faj valamely génje egy másik fajból hiányozhat. A géntartalom változására a legegyszerűbb modell az, amelyben két mutációt különböztetünk meg: egy gén törlődik egy genomból vagy egy gén megjelenik a genomban. Adott néhány faj géntartalma, valamint az ezen fajok leszármazását bemutató törzsfa. Adjuk meg azt az evolúciós történetet, amely minimális számú mutációval írja le az adott fajok evolúcióját.

13.2-3. Adott szekvenciákra és törzsfára adjuk meg a Maximum Likelihood evolúciós történetet, azaz a fa belső csúcsain azokat a szekvenciákat, amelyre a likelihood maximális.

13.2-4. Írjuk fel a takarékossági elv kis problémáját a (13.40) képlethez hasonló alakban (csak az összegzések helyett minimumok szerepeljenek), és mutassuk meg, hogy Sankoff algoritmusa tulajdonképpen Felsenstein algoritmusához hasonló átrendezésen alapszik.

13.2-5. Fitch algoritmusa a következőképen működik. Minden r csúcshoz hozzárendelünk egy karakterhalmazt, C_r-t, a levelekhez egyelemű halmazokat, amelyek az adott levélen található karaktert tartalmazzák, minden r belső csúcshoz pedig

$$\begin{array}{ll} \cap_{t\in D(r)} C_t, & \text{ha} \quad \cap_{t\in D(r)} C_t \neq \emptyset\,, \\ \cup_{t\in D(r)} C_t & \text{egyébként}\ . \end{array}$$

Miután elértünk a gyökérhez, a gyökérhez rendelt halmazból tetszőlegesen kiválasztunk egy karaktert, majd lefelé haladva minden egyes belső csúcsból kiválasztjuk ugyanazt a karaktert, mint amit a felette levő csúcshoz rendeltünk, amennyiben szerepel ez a karakter az adott halmazban, egyébként egy tetszőleges karaktert választunk. Mutassuk meg, hogy így egy legtakarékosabb történethez jutunk. Mennyi lesz ezen algoritmus futási ideje?

13.2-6. Mutassuk meg, hogy a 13.2.1. pontban megadott algoritmussal minden lehetséges legtakarékosabb evolúciós történetet megkaphatunk. Adjunk példát olyan legtakarékosabb történetre, amelyet Fitch algoritmusával nem kaphatunk meg.

13.3. Algoritmusok sztochasztikus nyelvtanokon

Az alábbiakban generatív nyelvtanok sztochasztikus változataival fogunk foglalkozni. A sztochasztikus generatív nyelvtanok központi szerepet játszanak a modern bioinformatikában. Két nyelvtantípus terjedt el széles körben, a rejtett Markov-modellek leggyakoribb

alkalmazási területei fehérjék térszerkezetének jóslása, illetve génkeresés, a sztochasztikus környezetfüggetlen nyelvtanok pedig az RNS molekulák másodlagos szerkezetének jóslásában játszanak fontos szerepet.

13.3.1. Rejtett Markov-modellek: előre, hátra és Viterbi algoritmusa

Az alábbiakban definiáljuk formálisan a ***rejtett Markov-folyamatokat.*** Legyen adva állapotok egy véges X halmaza. A halmazban van két kitüntetett elem, a kezdő és a végállapot. Az állapotokat két részhalmazra bontjuk, emittáló és nem-emittáló állapotokra. Egyelőre feltesszük, hogy csak a kezdő és a végállapot nem emittáló. Később látni fogjuk, hogy ez a feltételezés nem túl szigorú (lásd 13.3-3. gyakorlat).

Megadunk egy M transzformációs mátrixot, melynek egy m_{ij} eleme megadja az i-ből a j állapotba ugrás valószínűségét, így értelemszerűen a mátrix nem-negatív, és minden sor összege 1 (teljes valószínűség tétele). A végállapotra a mátrix nem tartalmaz sort, a kezdőre oszlopot.

Megadunk egy Σ ábécét, és minden emittáló állapotra egy eloszlásfüggvényt az ábécé elemein, amely megmondja, hogy az adott állapot mely valószínűséggel melyik karaktert fogja emittálni amikor a folyamat az adott állapotban van. π^i_ω-vel fogjuk jelölni annak a valószínűségét, hogy az i állapot az ω karaktert emittálja, feltéve persze, hogy az i állapotban van a folyamat. A folyamat a kezdő (*START*) állapotból indul, és a végállapotba (*END*) érkezik. A *START* állapotba ugrani nem lehet. Minden egyes diszkrét időpontban a folyamat továbbhalad, a megadott M mátrix szerint. Minden emittáló állapot emittál egy karaktert, mikor a folyamat az adott állapotban van. A folyamat attól válik rejtetté, hogy mi csak az emittált karakterek láncát látjuk, magát a folyamatot nem. Néha előfordul, hogy nem vezetünk be kezdő és végállapotot, ekkor meg kell adni egy kezdeti eloszlást, hogy a $T = 0$ időpontban melyik állapotban vagyunk (azaz melyik állapot emittál először). Három fontos kérdést tehetünk fel, melyeket dinamikus programozási algoritmussal fogunk megválaszolni.

Az első kérdésünk a következő: adott egy Markov-folyamat, és egy emittált szekvencia. Adjuk meg a Markov-folyamaton azt az utat, amelyik az adott szekvenciát emittálta, és a valószínűsége a legnagyobb.

A kérdéses út megtalálását Viterbi algoritmusával oldjuk meg, ami szintén dinamikus programozási algoritmus. Szokás szerint A_k jelöli az A szekvencia első k karakteréből álló szekvenciát, és a_k a k-adik karaktert. A dinamikus programozás azon alapszik, hogy ha ismerjük minden k-ra és i-re a $\Pr_{max}\{A_k, i\}$ valószínűséget, azaz az A_k szekvenciát emittáló, és aktuálisan az i állapotban végződő utak valószínűségei közül a maximálisat, akkor

$$\Pr_{max}\{A_{k+1},\ j\} = \max_i(\Pr_{max}\{A_k,\ i\}\, m_{i,j}\pi^j_{a_{k+1}})\ . \tag{13.42}$$

A képlet abból adódik, hogy annak a valószínűsége, hogy egy út az adott szekvenciát emittálta, egyszerűen a szorzata az egyes ugrások valószínűségének és az egyes emissziók valószínűségének. Ha adva van ilyen szorzatok halmaza, amelyekben az utolsó két szorzótényező ugyanaz (jelen esetben $m_{i,j}\pi^j_{ak+1}$), akkor ezek közül az a maximális, amelyikben a többi szorzótényező szorzata maximális.

Ha megjegyezzük, hogy az adott $\Pr_{max}\{A_{k+1},\ j\}$ kiszámításához melyik i-t használtuk fel, akkor nyomon tudjuk követni a maximális emittáló útvonalat. Az *END* állapot nem

emittál, így az algoritmus befejeződése

$$\Pr_{max}\{A\} = \Pr_{max}\{A, END\} = \max_i(\Pr_{max}\{A, i\}\, m_{i,END}) \,, \tag{13.43}$$

ahol $\Pr_{max}\{A\}$ a legvalószínűbb út valószínűsége. Ha nincsen végállapot, akkor

$$\Pr_{max}\{A\} = \max_i(\Pr_{max}\{A, i\}) \,. \tag{13.44}$$

Ha van *START* állapot, akkor a folyamat szükségképpen a *START* állapotból indul, úgyhogy $\Pr_{max}\{A_0, START\} = 1$. Ha nincs start állapot, akkor

$$\Pr_{max}\{A_1,\ j\} = p_j \pi^j_{a_1} \,, \tag{13.45}$$

ahol p_j annak a valószínűsége, hogy a folyamat a j állapotból indul.

A második kérdésünk a következő: ha adott egy Markov-folyamat, és egy emittált szekvencia, akkor mi annak a valószínűsége, hogy az adott Markov-folyamat az adott szekvenciát emittálta? Ez a valószínűség egyszerűen az emittáló utak valószínűségeinek az összege. Mivel azonban a lehetséges emittáló utak száma exponenciálisan nőhet a szekvencia hosszával, a naiv módszer, miszerint számoljuk ki minden egyes út valószínűségét, és adjuk ezeket össze, nem járható.

Dinamikus programozással azonban kiszámolható a kérdéses valószínűség. Ezt a dinamikus programozási algoritmust hívják Forward algoritmusnak, és nagyon hasonlít Viterbi algoritmusára, csak maximum helyett összegzések vannak benne. A dinamikus programozás azon alapszik, hogy ha ismerjük minden k-ra és i-re a $\Pr\{A_k, i\}$ valószínűséget, azaz az A_k szekvenciát emittáló, és aktuálisan az i állapotban végződő utak valószínűségeinek összegét, akkor

$$\Pr\{A_{k+1},\ j\} = \sum_i \Pr\{A_k,\ i\}\, m_{i,j} \pi^j_{a_{k+1}} \,. \tag{13.46}$$

Az *END* állapot nem emittál, így az algoritmus $\Pr\{A\}$-t a következőképen számolja ki:

$$\Pr\{A\} = \Pr\{A, END\} = \sum_i \Pr\{A, i\}\, m_{i,END} \,. \tag{13.47}$$

A legvalószínűbb emisszió útját ki tudjuk számolni, és ebből megkaphatjuk azt, hogy a legvalószínűbb úton mely karaktert mely állapot emittálta. Azonban ennek az útvonalnak lesznek jobban és kevésbé megbízható részei. Ezért érdeklődhetünk a $\Pr\{a_k$-t az i állapot emittálta $|$ a folyamat az A szekvenciát emittálta$\}$ valószínűség iránt is, ami a harmadik olyan kérdésünk, melyet dinamikus programozással válaszolunk meg. Ez a valószínűség nem más, mint azon utak valószínűségeinek az összege, melyekre az i állapot bocsátotta ki az a_k karaktert, osztva a teljes kibocsátási valószínűséggel. A kérdéses utak száma exponenciálisan nőhet a szekvencia hosszával, így a naiv algoritmus, amely megkeresi ezen utakat, és egyesével összeadja a valószínűségeiket, megint csak nem járható a gyakorlatban.

Először is ki kell számolni annak a valószínűségét, hogy egy folyamat az A^k szekvenciát emittálta, feltéve, hogy a_k-t a i állapot emittálta, ahol A^k az A szekvencia vége, a $k+1$-edik karaktertől kezdve. Ezt a Forward algoritmushoz hasonló Backward algoritmussal lehet megadni. Jelöljük $\Pr\left\{A^k, i\right\}$-vel annak a valószínűségét, hogy egy folyamat az A^k szekvenciát

emittálta, feltéve, hogy a_k-t a i állapot emittálta. Ekkor

$$\Pr\left\{A^k,\, i\right\} = \sum_j (\Pr\left\{A^{k+1},\, j\right\} m_{i,j}\pi^j_{a_{k+1}}) \ . \tag{13.48}$$

Jelöljük a $\Pr\{a_k$-t az i állapot emittálta $|$ a folyamat az A szekvenciát emittálta$\}$ valószínűséget $\Pr\{a_k = i|A\}$-val.

$$\Pr\{a_k = i|A\}\Pr\{A\} = \Pr\{A \wedge a_k = i\} = \Pr\{A_k,\, i\}\Pr\left\{A^k,\, i\right\} \ , \tag{13.49}$$

amiből:

$$\Pr\{a_k = i|A\} = \frac{\Pr\{A_k,\, i\}\Pr\left\{A^k,\, i\right\}}{\Pr\{A\}} \ , \tag{13.50}$$

ami éppen a keresett valószínűség.

13.3.2. Sztochasztikus környezetfüggetlen nyelvtanok: belülről, kívülről és a CYK algoritmus

Megmutatható, hogy minden környezetfüggetlen nyelvtan átírható úgynevezett ***Chomsky-féle normálformába***. A Chomsky-féle normálformában minden levezetési szabály $W_v \to W_yW_z$ vagy $W_w \to a$ alakú, ahol minden W nemterminális szimbólum, a pedig terminális szimbólum. A sztochasztikus környezetfüggetlen nyelvtanokban a levezetési szabályokhoz valószínűségeket rendelünk és minden nemterminális szimbólum lehetséges levezetéseihez rendelt valószínűségek összege 1.

Legyen adott egy sztochasztikus környezetfüggetlen nyelvtan és egy szekvencia (szó). Három kérdést tehetünk fel, az első: Mi a szekvencia levezetésének a valószínűsége, azaz a lehetséges levezetések valószínűségeinek az összege. A második kérdés az, hogy mi a legvalószínűbb levezetés, a harmadik pedig az, hogy mi annak a valószínűsége, hogy egy részszót egy adott W_x nemterminálisból kiindulva vezettük le, feltéve, hogy az adott szót vezettük le. Hasonlóan a rejtett Markov-folyamatokhoz, az első kérdésre két algoritmust adunk meg, a Kívülről, illetve a Belülről algoritmusokat, melyek analógok az Elölről, illetve Hátulról algoritmusokkal. A második kérdésre a CYK (Cocke–Younger–Kasami) algoritmus adja meg a választ, amely a Viterbi algoritmussal analóg. A harmadik kérdésre pedig a Kívülről és a Belülről algoritmusok közös alkalmazásával kaphatunk választ. A bemutatásra kerülő algoritmusok hasonlóak a rejtett Markov-folyamatok algoritmusaihoz, a futási idők azonban lényegesen nagyobbak.

Jelöljük a $W_v \to W_yW_z$ levezetés valószínűségét $t_v(y,z)$-vel, a $W_v \to a$ levezetés valószínűségét $e_v(a)$-val. A Belülről algoritmus minden $i \leq j$-re és v-re kiszámolja az $\alpha(i,j,v)$ valószínűséget, ami annak a valószínűsége, hogy a W_v nemterminálisból levezetjük az a_i-től a_j-ig terjedő részszót. A dinamikus programozás kezdeti feltételei:

$$\alpha(i,i,v) = e_v(a_i) \ , \tag{13.51}$$

minden i-re és v-re. A fő rekurzió:

$$\alpha(i,j,v) = \sum_{y=1}^{M}\sum_{z=1}^{M}\sum_{k=i}^{j-1}\alpha(i,k,y)t_v(y,z)\alpha(k+1,j,z) \ , \tag{13.52}$$

ahol M a nemterminális szimbólumok száma. A dinamikus programozási táblázat egy felső háromszög mátrix minden nemterminális szimbólumra. A táblázat kitöltése a főátlóval kezdődik, és innen haladunk a jobb felső sarok felé, a főátlóval párhuzamosan töltve ki a mátrixot. A levezetés valószínűségét $\alpha(1, L, 1)$ adja meg, ahol L a szekvencia hossza, W_1 pedig a kezdő nemterminális. Az algoritmus futási ideje $\Theta(L^3M^3)$, a memóriaigény $\Theta(L^2M)$.

A Kívülről algoritmus minden $i \leq j$-re és v-re a $\beta(i, j, v)$ számokat számolja ki, ami azon levezetések valószínűsége, melyben az a_i-től a_j-ig terjedő részszót W_v vezeti le osztva $\alpha(i, j, v)$-vel, illetve 0, ha $\alpha(i, j, v) = 0$. Az algoritmus kezdeti feltételei:

$$\beta(1, L, 1) = 1 \ , \tag{13.53}$$

$$\beta(1, L, v) = 0 \quad \text{ha } v \neq 1 \ . \tag{13.54}$$

A fő rekurzió:

$$\begin{aligned}\beta(i, j, v) \quad = \quad & \sum_{y=1}^{M} \sum_{z=1}^{M} \sum_{k=1}^{i-1} \alpha(k, i-1, z) t_y(z, v) \beta(k, j, y) + \\ & \sum_{y=1}^{M} \sum_{z=1}^{M} \sum_{k=j+1}^{L} \alpha(j+1, k, z) t_y(v, z) \beta(i, k, y) \ .\end{aligned} \tag{13.55}$$

A (13.55) képlet helyessége abból adódik, hogy végignézzük az összes lehetőséget, hogy az a nemterminális szimbólum, amely W_v-t levezette, mely részét vezette le a szekvenciának. Mint látható, a számoláshoz szükségünk van az α-kra, ilyen téren a Kívülről algoritmus eltér a Hátulról algoritmustól, amelyet futtathatunk anélkül, hogy az Elölről algoritmust alkalmaztuk volna, míg a Kívülről algoritmus előtt mindenképpen le kell futtatni a Belülről algoritmust.

A CYK algoritmus kezdeti értékeinek beállítása megegyezik a Belülről kezdeti értékeinek beállításával, a fő rekurzió is nagyon hasonlít a Belülről algoritmus rekurziójához, csak összeadás helyett maximalizálni kell:

$$\alpha_{\max}(i, j, v) = \max_{y} \max_{z} \max_{i \leq k \leq j-1} \alpha_{\max}(i, k, y) t_v(y, z) \alpha_{\max}(k+1, j, z) \ . \tag{13.56}$$

A legvalószínűbb levezetés valószínűségét pedig $\alpha_{\max}(1, L, 1)$ adja meg. A legvalószínűbb levezetést az optimális értékeket adó levezetési szabályokon keresztül kaphatjuk meg.

Végezetül annak a valószínűségét, hogy W_v vezette le az a_i-től a_j-ig terjedő részszót, feltéve, hogy az A szekvenciát vezettük le,

$$\frac{\alpha(i, j, v)\beta(i, j, v)}{\alpha(1, L, 1)} \tag{13.57}$$

adja meg.

Gyakorlatok

13.3-1. A reguláris nyelvtanokban a levezetési szabályok $W_v \to aW_y$ vagy $W_v \to a$ alakúak. Mutassuk meg, hogy minden rejtett Markov-folyamat sztochasztikus reguláris nyelvtan, de nem minden sztochasztikus reguláris nyelvtan rejtett Markov-folyamat.

13.3-2. Adjunk meg dinamikus programozási algoritmust, mely adott sztochasztikus reguláris nyelvtanra és A szekvenciára megadja a

- levezetés valószínűségét,
- a legvalószínűbb levezetést,
- valamint annak a valószínűségét, hogy a szekvencia egy adott a_i karakterét egy adott levezetési szabály vezette le.

13.3-3. Egy rejtett Markov-modellben lehetnek úgynevezett *csendes* vagy nem kibocsátó állapotok, melyek a rejtett Markov-modell ábrázolását elősegíthetik. Mutassuk meg, hogy minden rejtett Markov-modell, mely csendes állapotokat tartalmaz, átírható olyan rejtett Markov-modellé, amely nem tartalmaz csendes állapotokat, és ekvivalens az eredeti modellel, azaz ugyanazon szekvenciákat ugyanakkora valószínűséggel bocsátja ki.

13.3-4. A *páros rejtett Markov-modellek* olyan rejtett Markov-modellek, melyekben az egyes állapotok nem csak egy szekvenciába bocsátanak ki karaktereket, hanem kettőbe. Egyes állapotok csak az egyik szekvenciába, mások csak a másikba, ismét mások pedig mindkét szekvenciába bocsátanak ki karaktereket. Egy állapot egy lépésben mindegyik szekvenciába legfeljebb egy karaktert bocsáthat ki. Adjuk meg a páros rejtett Markov-folyamatok Viterbi, Előre és Hátra algoritmusait.

13.3-5. Viterbi algoritmusában nem használtuk ki, hogy a tranzíciók és a kibocsátási valószínűségek valószínűségek, azaz nem negatívak és eggyé összegződnek. Valamint az algoritmus akkor is működik, ha szorzás helyett összeadások vannak, és maximalizálás helyett akár minimalizálhatunk is. Adjunk meg egy olyan módosított páros rejtett Markov-modellt (amelyben a „valószínűségek” nem feltétlenül nem negatívak, és nem feltétlenül összegződnek eggyé), melyre Viterbi algoritmusa összeadásokkal és minimalizálással ekvivalens Gotoh algoritmusával.

13.3-6. Másodlagos térszerkezetnek nevezzük az RNS-ek olyan bázispárosodását, amelyben a bázispárosodott nukleinsavakat összekötő, a szekvencia fölött haladó ívek nem metszik egymást. A lehetséges bázispárosodások: $A-U$, $U-A$, $C-G$, $G-C$, $G-U$ és $U-G$. Adjunk meg egy dinamikus programozási algoritmust, amely egy adott RNS szekvenciára megkeresi azt a másodlagos térszerkezetet, melyben a párosodott nukleinsavak száma maximális.

13.3-7. A Knudsen–Hein-nyelvtan levezetési szabályai:

$$\begin{aligned} S &\rightarrow LS|L\,, \\ F &\rightarrow dFd|LS\,, \\ L &\rightarrow s|dFd\,, \end{aligned}$$

ahol minden s terminális levezetést még helyettesíteni kell az RNS szekvenciák lehetséges karaktereivel, a dFd kifejezésben pedig a két d terminális szimbólumot helyettesíteni kell a lehetséges bázispárosodásokkal. Mutassuk meg, hogy adott szekvencia és a levezetések adott valószínűségi eloszlása esetén meg lehet adni egy olyan dinamikus programozási algoritmust, amely megadja a szekvencia levezetésének a valószínűségét, anélkül, hogy Chomsky-féle normálformába írnánk át a nyelvtant.

13.4. Szerkezetek összehasonlítása

Az alábbi részben különböző szerkezeteket hasonlítunk össze dinamikus programozási algoritmusok segítségével. Mint meg fogjuk mutatni, a címkézett, gyökeres fák illesztésére használt algoritmus általánosítása a szekvenciák összehasonlítására használt algoritmusnak.

A rejtett Markov-modellek összehasonlítását végző algoritmus egy lineáris egyenletrendszer megoldásával adja meg két rejtett Markov-modell együttes kibocsátásának a valószínűségét, azaz annak a valószínűséget, hogy két rejtett Markov-modell ugyanazt a szekvenciát bocsátja ki.

13.4.1. Címkézett, gyökeres fák illesztése

Legyen Σ egy véges ABC, $\Sigma^- = \Sigma \cup \{-\}$, $\Sigma^2 = \Sigma^- \times \Sigma^- \backslash \{-,-\}$. Egy F fa címkézésén egy olyan függvényt értünk, mely az F fa minden egyes $n \in V_F$ csúcsához hozzárendeli Σ egy karakterét. Ha egy gyökeres fából kitörlünk egy csúcsot, akkor a kitörölt csúcs gyerekei a kitörölt csúcs szülőjének a gyerekei lesznek. Amennyiben a fa gyökerét töröljük ki, a fa erdőre esik szét. Legyen A egy gyökeres fa, melynek csúcsai Σ^2 elemeivel vannak címkézve, az ezt meghatározó függvény legyen $c : V_A \to \Sigma^2$. Azt mondjuk, hogy A illesztése az F és G Σ karaktereivel címkézett, gyökeres fáknak, ha A címkézéseinek első és második koordinátáján vett megszorításával, és az így '$-$' szimbólummal címkézett csúcsok törlésével rendre az F és G fákat kapjuk vissza.

Legyen adva egy hasonlósági függvény, $s : \Sigma^2 \to R$. Erre a hasonlósági függvényre semmilyen megkötést nem teszünk, egy karakter lehet akár kevésbé hasonló önmagához, mint egy másik karakterhez. F és G fák optimális illesztésén egy olyan Σ^2 elemeivel címkézett A fát értünk, melyre

$$\sum_{n \in V_A} s(c(n)) \tag{13.58}$$

maximális. Ezt a fát $A_{F,G}$-vel fogjuk jelölni. Vegyük észre, hogy egy szekvencia ábrázolható olyan faként, melynek egyetlen levele van.

A továbbiakban csak olyan fákkal foglalkozunk, melyekben bármely csúcsnak legfeljebb két gyereke van. Az optimális illesztést megadó dinamikus programozás a gyökeres részfákon történik, ezek azon részfák, melyek a fa egy adott csúcsát, mint gyökeret és ezek leszármazottjait tartalmazzák. Az r gyökér által meghatározott részfát t_r-rel jelöljük. Egy fát egy üres fához csak egyféleképen illeszthetünk. Két, a, ill. b karakterekkel címkézett levél illesztése csak három módon lehetséges: az illesztés vagy egyetlen csúcsot tartalmaz, és (a,b)-vel van címkézve, vagy két csúcsot tartalmaz, ezek közül az egyik $(a,-)$, a másik $(-,b)$ címkézésű, és a két csúcs közül az egyik a gyökér, a másik ennek a gyereke. Ez utóbbi két lehetőség ekvivalens a hasonlósági függvény szempontjából.

Hasonlóan, egy levelet egy gyökeres részfával úgy lehet összeilleszteni, hogy az A illesztésben vagy együtt van címkézve a levél karaktere a részfa valamely karakterével, vagy egy '$-$' szimbólummal van együtt címkézve. Ez utóbbi címkézést tartalmazó csúcsot a részfába sokféleképen lehet beszúrni, de ezek mindegyike ekvivalens.

Ezután az inicializáció után a dinamikus programozásban egyre nagyobb részfákat illesztünk össze. Feltehetjük, hogy t_r, illetve t_s részfák esetében ismerjük már az A_{t_r,t_x}, A_{t_r,t_y}, A_{t_u,t_s}, A_{t_v,t_s}, A_{t_u,t_x}, A_{t_u,t_y}, A_{t_v,t_x} és A_{t_v,t_y} illesztéseket, és ezen illesztések értékeit, ahol u és v csúcsok r gyerekei, x és y csúcsok pedig s gyerekei (amennyiben valamelyik csúcsnak csak

egy gyereke van, akkor természetesen kevesebb részproblémára vezetjük vissza a problémát). Valamint ismerjük a t_u, t_v, t_x és t_y fáknak az üres részfához való illesztésének az értékét is. Legyen r címkézése a, s címkézése b. Ezek után A_{t_r,t_s} meghatározásához konstans sok lehetőséget kell végignézni: vagy az egyik részfa a másik részfában valamely gyerekhez van illesztve, és ekkor a másik gyerek és a gyökér a '$-$' szimbólummal címkéződik az illesztésben, vagy r és s összeillesztődik, vagy bár nem illesztődnek össze, de A_{t_r,t_s}-ben az egyik gyökérnek megfelelő csúcs a gyökér, a másik pedig ennek a gyereke. Ez utóbbi két esetben a gyerekeket vagy összeillesztjük, vagy nem. Zz öt lehetséges eset.

Mivel a lehetséges gyökeres részfák száma egyenlő a fa csúcsainak számával, az optimális illesztés megkereshető $\Theta(|F||G|)$ időben, ahol $|F|$ és $|G|$ F és G csúcsainak száma.

13.4.2. Két rejtett Markov-modell együttes kibocsátási valószínűsége

Legyen adva két Markov-modell, M_1 és M_2. A két modell együttes kibocsátási valószínűsége definíció szerint:

$$C(M_1, M_2) = \sum_s \Pr_{M_1}\{s\} \Pr_{M_2}\{s\}, \tag{13.59}$$

ahol az összegzés az összes lehetséges szekvencián megy, $\Pr_M\{s\}$ pedig annak a valószínűsége, hogy az M modell az s szekvenciát bocsátotta ki. Azt, hogy a p út az s szekvenciát bocsátotta ki, $e(p) = s$-sel jelöljük, egy *START* állapottól x állapotig tartó utat pedig $[x]$-szel. Mivel a kibocsátási valószínűség a lehetséges kibocsátó utak valószínűségeinek az összege,

$$\begin{aligned} C(M_1, M_2) &= \sum_s \left(\sum_{p_1 \in M_1, e(p_1)=s} \Pr_{M_1}\{p_1\} \right) \left(\sum_{p_2 \in M_2, e(p_2)=s} \Pr_{M_2}\{p_2\} \right) \\ &= \sum_{p_1 \in M_1, p_2 \in M_2, e(p_1)=e(p_2)} \Pr_{M_1}\{p_1\} \Pr_{M_2}\{p_2\}. \end{aligned} \tag{13.60}$$

Ez utóbbi képletben figyelembe kell venni, hogy egy útvonalra több lehetséges kibocsátás van, az összegzések a lehetséges útvonalak és kibocsátások együttesein mennek, az útvonal valószínűségébe pedig beleértjük a kibocsátási valószínűségeket is. Jelöljük $\bar{p}_1$-gyel azt az útvonalat, amelyet p_1-ből kapunk a végállapot elhagyásával, valamint p_1-nek az END_1 állapot előtti állapota legyen x_1. ($\bar{p}_2$-t és x_2-t hasonlóan definiáljuk.) Ekkor

$$\begin{aligned} C(M_1, M_2) &= \sum_{p_1 \in M_1, p_2 \in M_2, e(p_1)=e(p_2)} m_{x_1,END_1} m_{x_2,END_2} \Pr_{M_1}\{\bar{p}_1\} \Pr_{M_2}\{\bar{p}_2\} \\ &= \sum_{x_1,x_2} m_{x_1,END_1} m_{x_2,END_2} C(x_1, x_2), \end{aligned} \tag{13.61}$$

ahol $m_{x,END}$ az x-ből az END állapotba ugrás valószínűsége, valamint

$$C(x_1, x_2) = \sum_{[x_1] \in M_1, [x_2] \in M_2, e([x_1])=e([x_2])} \Pr_{M_1}\{[x_1]\} \Pr_{M_2}\{[x_2]\}. \tag{13.62}$$

$C(x_1, x_2)$-t meg lehet adni a következő képlettel is:

$$C(x_1, x_2) = \sum_{y_1,y_2} m_{y_1,x_1} m_{y_2,x_2} C(y_1, y_2) \sum_{\sigma \in \Sigma} \Pr\{\sigma|x_1\} \Pr\{\sigma|x_2\}, \tag{13.63}$$

ahol $\Pr\{\sigma|x_i\}$ annak a valószínűsége, hogy az x_i állapot σ-t bocsátotta ki. A (13.63) képlet egy lineáris egyenletrendszert definiál az összes x_1 és x_2 kibocsátó állapotokra. A kezdeti feltételek:

$$C(START_1, START_2) = 1 , \tag{13.64}$$

$$C(START_1, x_2) = 0, \quad x_2 \neq START_2 , \tag{13.65}$$

$$C(x_1, START_2) = 0, \quad x_1 \neq START_1 . \tag{13.66}$$

Azonban a dinamikus programozás a szokásostól eltérően nem egy táblázat kitöltésével, hanem a (13.63) képlet által meghatározott egyenletrendszer megoldásával történik. Így az együttes kibocsátási valószínűség meghatározható $O\left((n_1 n_2)^3\right)$ időben, ahol n_i az M_i modellben a kibocsátó állapotok száma.

Gyakorlatok

13.4-1. Adjuk meg két fa lokális hasonlóságát, ami a két fa leginkább hasonló részfái illesztésének az értéke. Részfán most a fa tetszőleges összefüggő részét érjük.

13.4-2. ***Rendezett fákon*** olyan gyökeres fákat értünk, melyben minden csúcs gyerekei rendezve vannak. Rendezett fák rendezett illesztése megőrzi a két fa gyerekeinek a rendezését. Adjunk olyan algoritmust, amely két rendezett fának egy optimális rendezett illesztését adja meg, és a számolási igénye mind a fák csúcsszámának, mind a gyerekszám maximális értékének polinomiális függvénye.

13.4-3. Vegyük azt a végtelen dimenziós euklideszi teret, melynek a koordinátái a lehetséges szekvenciák. Minden rejtett Markov-modell egy vektorral adható meg ebben a térben, a vektor j-edik koordinátája megadja a j-edik szekvencia levezetésének a valószínűségét. Határozzuk meg két rejtett Markov-modell által bezárt szöget ebben a térben.

13.4-4. Adjuk meg egy rejtett Markov-modell által kibocsátott szekvenciák hosszainak generátorfüggvényét, azaz a

$$\sum_{i=0}^{\infty} p_i \xi^i$$

függvényt, ahol p_i annak a valószínűsége, hogy a rejtett Markov-modell i hosszúságú szekvenciát bocsájt ki.

13.4-5. Adjuk meg egy páros rejtett Markov-modell által kibocsátott szekvenciák hosszainak generátorfüggvényét, azaz a

$$\sum_{i=0}^{\infty} \sum_{j=0}^{\infty} p_{i,j} \xi^i \eta^j$$

függvényt, ahol $p_{i,j}$ annak a valószínűsége, hogy a rejtett Markov-modell által kibocsátott első szekvencia i, a második pedig j hosszúságú.

13.5. Törzsfakészítés távolságon alapuló algoritmusokkal

Ebben a fejezetben törzsfákon olyan összefüggő, irányítatlan, súlyozott élű, körmentes gráfokat értünk, melyben semelyik csúcsnak nincs kettes fokszáma. A súlyok nem negatívak, és minden olyan élre, mely két belső csúcsot köt össze, a súlyok pozitívak. Olyan törzsfakészítő módszerekkel ismerkedünk meg, amelyek bemenő adatai objektumok halmaza,

valamint mindegyik objektumpárra megadott távolság. Ez a távolság származhat például szekvenciák optimális illesztéséből vett távolságokból, de az itt bemutatásra kerülő algoritmusok tetszőleges távolságokra működnek. A fák levelei a megadott objektumok, a fa topológiáját és a fa éleinek a hosszát pedig a távolságadatokból származtatjuk. Minden fa ábrázol egy, a leveleken, mint objektumokon definiált metrikát: két objektum közötti távolságot az ezen objektumokat összekötő út hosszával definiáljuk. Az algoritmusok jóságát lehet mérni a bemeneti távolságok és a megkonstruált fa által meghatározott távolságok közötti különbséggel.

Két speciális metrikát fogunk definiálni, az ultrametrikát és az additív metrikát. Az osztályozó algoritmusok mindig olyan törzsfát készítenek, amelyek ultrametrikát reprezentálnak. Be fogjuk bizonyítani, hogy amennyiben a bemeneti adatokban szereplő távolságok ultrametrikus tulajdonságúak, akkor az osztályozó algoritmusok által meghatározott fa pontosan ezt fogja reprezentálni.

Hasonlóan a szomszédok egyesítése módszer additív metrikát reprezentáló fát készít, és ha a bemenő távolságokokra teljesül az additív metrika, akkor a szomszédok egyesítése visszaadja ezt a metrikát.

Mindkét bizonyítás esetében szükségünk lesz az alábbi lemmára:

13.3. lemma. *Bármely metrikára legfeljebb egy olyan fa van, amely ezt ábrázolja.*

Bizonyítás. Két objektumra az állítás triviális. A bizonyítás indirekten, teljes indukcióval történik. Az indukciót három objektummal kezdjük. Három objektum esetében egyetlen fa topológia létezik, a csillag alakú. Legyen az i, j és k leveleket a fa belső csúcsával összekötő élek hossza rendre x, y és z. Az élek hosszait az

$$x + y = d_{i,j} , \tag{13.67}$$

$$x + z = d_{i,k} , \tag{13.68}$$

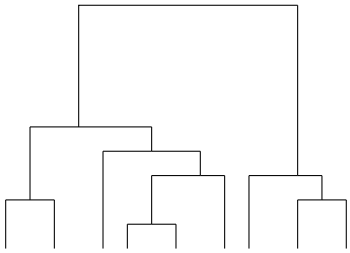
$$y + z = d_{k,l} \tag{13.69}$$

egyenletrendszer adja meg, aminek egyetlen megoldása van, mivel az

$$\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{vmatrix} \tag{13.70}$$

determináns nem 0.

$n > 3$ objektum esetében tegyük fel, hogy van két fa, amely ugyanazt a metrikát reprezentálja. Ekkor az első fán keressünk két olyan levelet, i-t és j-t, melyeket összekötő úton egyetlen egy csúcs van, legyen az a csúcs u. Ilyen i és j csúcsokat minden fában találunk, vegyünk egy olyan utat a fában, melyben a belső csúcsok száma a legnagyobb, az út mindkét végén ilyen levélpár található. Ha a második fában i-t és j-t összekötő úton egyetlen csúcs van, akkor a két fában i-t a belső csúccsal összekötő élek hossza azonos, és úgyszintén a j-t a belső csúccsal összekötő élek hossza azonos, mivel tetszőleges $k(\neq i, j)$ objektumra mindkét fa esetében ugyanazt a részfát kell kapnunk (melyben az u és k közötti utat egyetlen $d_{u,k}$ hosszúságú éllel ábrázoljuk). Legyártunk egy új metrikát, melyben elhagyjuk az i és j objektumokat, bevezetünk egy u' objektumot, melynek bármely k objektumtól vett távolsága $d_{i,k} - d_{i,u}$, ahol $d_{i,u}$, az i-t u-val összekötő él hossza. A két fában elhagyjuk az i és j csúcsokat,



13.2. ábra. Egy dendrogram.

ha u fokszáma 3 volt i és j elhagyása előtt, akkor u levél lesz az új fában, és most ez fogja reprezentálni u'-t, ha u nem levél i és j elhagyása után, akkor u-ba behúzunk egy u' levelet, az új él hossza pedig 0. Így olyan fákat kapunk, melyek ezt a metrikát reprezentálják, és az indukció szerint azonosak.

Ha viszont a második fában i-t j-vel összekötő úton nem egy csúcs van, akkor ellentmondásra jutunk. Ugyanis ebben a fában van az i-t j-vel összekötő úton egy olyan u_1 csúcs, melyre $d_{i,u} \neq d_{i,u_1}$. Vegyünk a második fában egy olyan k csúcsot, melyre az i-t k-val összekötő út áthalad u_1-en. Az első fából számolva

$$d_{i,k} - d_{j,k} = d_{i,u} - d_{j,u} = 2d_{i,u} - d_{i,j} \ , \tag{13.71}$$

míg a második fán

$$d_{i,k} - d_{j,k} = d_{i,u_1} - d_{j,u_1} = 2d_{i,u_1} - d_{i,j} \ , \tag{13.72}$$

ami ellentmond annak, hogy $d_{i,u} \neq d_{i,u_1}$. ■

13.5.1. Osztályozó algoritmusok

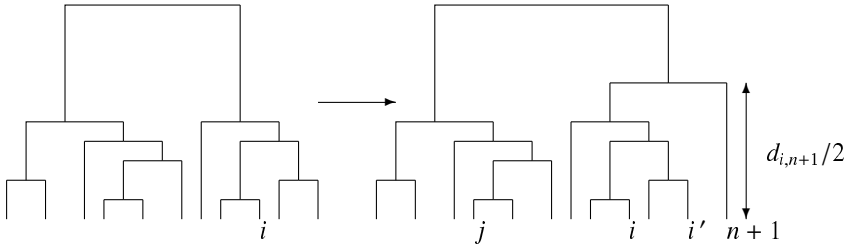
13.4. definíció. *Egy metrikát **ultrametrikának** nevezünk, ha bármely i, j és k csúcsokra*

$$d_{i,j} \leq \max\{d_{i,k},\ d_{j,k}\} \tag{13.73}$$

Könnyen belátható (lásd 13.5-1. gyakorlat), hogy egy ultrametrikában bármely három csúcs között levő három távolság vagy mind azonos, vagy közülük kettő azonos, a harmadik pedig ennél kisebb.

13.5. tétel. *Ha objektumok egy véges halmazán definiált metrika ultrametrika, akkor pontosan egy olyan fa létezik, amely ezt ábrázolja. Továbbá ezt a fát le lehet gyökereztetni úgy, hogy minden levélnek a gyökértől vett távolsága azonos legyen.*

Bizonyítás. A 13.3. lemma alapján legfeljebb egy ilyen fa van, így elég megkonstruálni egy ilyen fát bármely ultrametrikára. Az ultrametrikus fákat dendrogramokként fogjuk ábrázolni, ezekben a reprezentációkban a vízszintesen húzott élek hosszát 0-nak tekintjük (lásd 13.2. ábra). A tétel bizonyítása a levelek, mint objektumok száma szerinti indukcióval történik. Kettő objektum esetében nyílván meg tudjuk konstruálni a dendrogramot. Ha n levélre



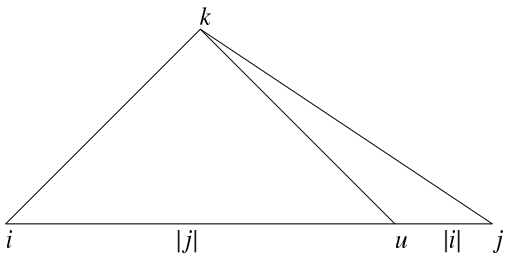
13.3. ábra. Az $(n+1)$-edik levél bekötése a dendrogramba.

megkonstruáltuk már a dendrogramot, az $(n+1)$-edik levéllel a következőképpen járunk el: Keresünk egy olyan i-t, melyre $d_{i,n+1}$ minimális. Ezután i-ből elindulunk a gyökér felé, és $d_{i,n+1}/2$ magasságban kötjük be az $(n+1)$-edik levelet (lásd 13.3. ábra). Ez a dendrogram helyesen ábrázolja az összes csúcsnak az $(n+1)$-edik levéltől vett távolságát. Ugyanis az összes olyan i' csúcsra, amely az $n+1$-ik levél bekötésének helye alatt helyezkedik el, $d_{i,i'} \leq d_{i,n+1}$, és az ultrametrikus tulajdonságból, valamint $d_{i,n+1}$ minimalitásából következik, hogy ekkor $d_{i,n+1} = d_{i',n+1}$. Másrészt az összes többi j csúcsra $d_{i,j} > d_{i,n+1}$, és az ultrametrikus tulajdonságból adódóan ekkor $d_{j,n+1} = d_{i,j}$. ■

Könnyen belátható, hogy a bizonyításban használt megkonstruálás számolási igénye $O(n^2)$, ahol n az objektumok száma. Megadhatunk egy másik algoritmust is, amely megkeresi azt az i és j objektumpárt, amire $d_{i,j}$ minimális. Az ultrametrikus tulajdonságból adódóan minden k-ra $d_{i,k} = d_{j,k} (\geq d_{i,j})$, így az i és j objektumpárt lecserélhetjük egyetlen új objektumra, és ezen új objektumnak az összes többitől vett távolsága jól definiált. Az i és a j objektumot összekötjük $d_{i,j}/2$ magasságban, és az így kapott részdendrogramot egy objektumnak tekintve folytatjuk az iterációt. Ez az algoritmus lassabb, mint az előbbi bizonyításban megadott algoritmus, viszont ez az alapja az úgynevezett osztályozó algoritmusoknak. Az osztályozó algoritmusok mindig dendrogramot adnak, akkor is, ha a bemenő távolságok nem alkotnak ultrametrikát. Viszont ha a bemenő adatok ultrametrikus tulajdonságúak, akkor az osztályozó algoritmusok többsége pontosan visszaadja az ezt reprezentáló dendrogramot.

Mindegyik osztályozó algoritmus azt az i és j objektumokat (illetve az iteráció további lépéseiben objektumok helyett objektumhalmazok is szerepelhetnek) keresi meg, amelyre $d_{i,j}$ minimális. A módszerek közötti különbség abban rejlik, hogy ezután hogyan határozzák meg az új objektumhalmaz és a többi objektum(halmaz) közötti távolságot. Ha az új objektumot u-val jelöljük, akkor az alább ismertetésre kerülő módszerek következőképpen definiálják $d_{u,k}$-t:

- Egyszerű-lánc: $d_{u,k} = \min\{d_{i,k}, d_{j,k}\}$.
- Teljes-lánc: $d_{u,k} = \max\{d_{i,k}, d_{j,k}\}$.
- Csoportátlag: (UPGMA) u és k elemei páronkénti távolságainak számtani közepe, azaz $d_{u,k} = (d_{i,k} \times |i| + d_{j,k} \times |j|)/(|i| + |j|)$, ahol $|i|$ és $|j|$ az i és j objektumhalmazok elemszámai.
- Egyszerű-átlag: Az átlagok átlagát vesszük, azaz $d_{u,k} = (d_{i,k} + d_{j,k})/2$.
- Centroid: Ezt a módszert leggyakrabban akkor alkalmazzák, amikor az objektumok



13.4. ábra. $d_{u,k}$ számolása a CENTROID módszer szerint.

beágyazhatóak Euklideszi térbe. Ekkor a két objektumhalmaz közötti távolságot az objektumhalmazok centrumai közötti távolságként lehet definiálni. A számoláshoz azonban nem feltétlenül kell az Euklideszi tér koordinátáit használni, hiszen a kérdéses $d_{u,k}$ távolság nem más, mint az i, j és k csúcsok által meghatározott háromszögben a k csúcsból kiinduló, az ij szakaszt $|j| : |i|$ arányban osztó szakasz hossza (lásd 13.4. ábra), ez pedig a $d_{i,j}$, $d_{i,k}$ és $d_{j,k}$ adatokból már meghatározható. Ez a számolási mód akkor is alkalmazható, ha az objektumok nem ágyazhatók be Euklideszi térbe, így bár a módszer ötlete geometriai indíttatású, bármely távolságmátrixra alkalmazható.

- MEDIÁN: Az u objektumhalmaz centrumát i és j centrumainak centrumaként definiáljuk. Így ez a módszer úgy viszonyul a centroid módszerhez, mint az egyszerű átlag a csoportátlaghoz. Erre a módszerre is igaz, hogy nem kell $d_{u,k}$ számolásához az Euklideszi koordinátákat ismerni, hiszen a keresett távolság az ijk háromszögben a k-ból induló súlyvonal hossza.

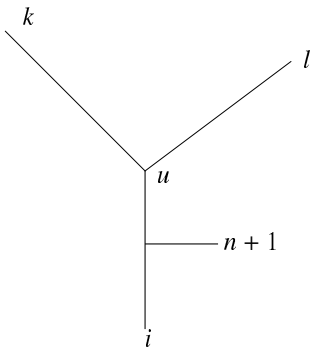
Könnyen belátható, hogy az első négy módszer visszaadja a távolságokat reprezentáló dendrogramot, amennyiben a bemenő adatok ultrametrikus tulajdonságúak, hiszen ekkor $d_{i,k} = d_{j,k}$. A CENTROID és a MEDIÁN módszerek azonban nem adják vissza az ultrametrikát reprezentáló dendrogramot, hiszen $d_{u,k}$ kisebb lesz, mint $d_{i,k}$ (ami egyenlő $d_{j,k}$-val).

Az osztályozó algoritmusok általános problémája, hogy mindig dendrogramot adnak vissza, és ez biológiailag nem feltétlenül helyes. Ugyanis biológiai szekvenciák leszármazási kapcsolatait csak az úgynevezett *molekuláris óra* működésének esetében ábrázolja helyesen egy dendrogram. A molekuláris óra elmélet szerint az egyes szekvenciák a törzsfejlődés során adott időtartam alatt ugyanakkora mennyiségű mutáción mentek át, azonban számos biológiai példa mutatja azt, hogy ez nem mindig teljesül. Ezért szeretnénk egy olyan algoritmust, amely csak akkor ad ultrametrikus fát a bemenő adatsorokra, ha a bemenő távolságok valóban ultrametrikus tulajdonságúak. Ezért mára a SZOMSZÉDOK EGYESÍTÉSE algoritmus sokkal népszerűbbé vált a bioinformatikai alkalmazásokban, mint az osztályozó algoritmusok.

13.5.2. Szomszédok egyesítése

13.6. definíció. *Egy metrikát* ***additív*** *vagy* ***négycsúcs metrikának*** *nevezünk, ha bármely i, j, k és l csúcsára*

$$d_{i,j} + d_{k,l} \leq \max\{d_{i,k} + d_{j,l},\, d_{i,l} + d_{j,k}\}\,. \tag{13.74}$$



13.5. ábra. Az $(n+1)$-edik levél gyökereztetése additív fa megkonstruálásához.

13.7. tétel. *Ha objektumok egy véges halmazán definiált metrika additív, akkor pontosan egy olyan fa létezik, amely ezt reprezentálja.*

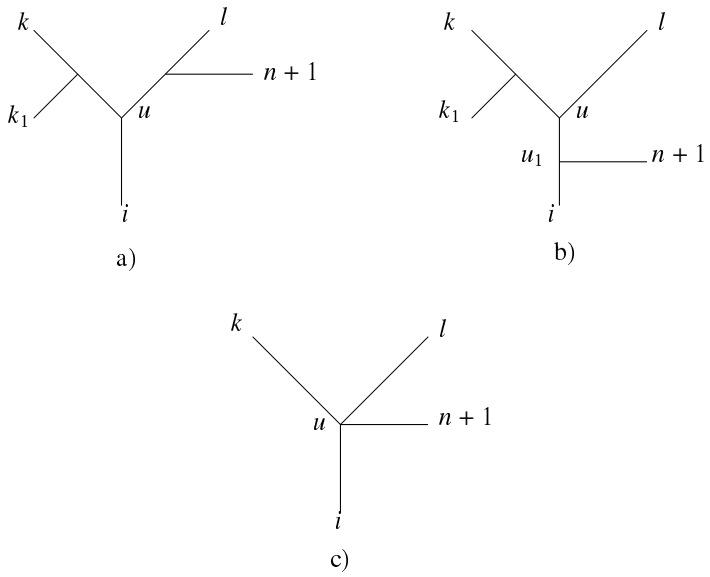
Bizonyítás. A 13.3. lemma alapján legfeljebb egy ilyen fa van, így elég megkonstruálni egy ilyen fát bármely additív metrikára. Először a konstrukciót adjuk meg, ezután bizonyítjuk a konstrukció helyességét:

Három objektumra a (13.67)-(13.69) egyenletek alapján megkonstruálunk egy fát, ezután a konstrukció indukcióval történik, feltesszük, hogy $n \geq 3$ objektumra már elkészítettük a fát, az $(n+1)$-edik objektumot reprezentáló levelet pedig egy éllel valahova bekötjük a már meglévő fába. Ehhez először meghatározzuk az új fa topológiáját, majd az új él hosszát. A topológia meghatározásához egy tetszőleges i levéltől indulunk el. Jelöljük i szomszédját u-val, u-ból még legalább két él indul ki, ezen élekből kiinduló utakon keressünk egy-egy levelet, jelöljük ezeket k-val és l-lel (lásd 13.5. ábra). Az $(n+1)$-edik levél bekötése i-ből nézve az u csúcson innen van, ha

$$d_{i,n+1} + d_{k,l} < d_{i,k} + d_{n+1,l} \; . \tag{13.75}$$

Hasonlóképpen megállapíthatjuk, hogy az $(n+1)$-edik levél bekötése k, illetve az l csúcsból nézve u-n innen van-e. Ha u fokszáma nagyobb, mint három, akkor a további élekből kiinduló utakon is keresünk l' csúcsokat, és az $i, n+1, k$ és l' csúcsnégyesekre hasonlóan járunk el. Az additív metrika tulajdonságából következik, hogy legfeljebb egy esetben állhat fenn az adott irányú egyenlőtlenség. Ha egyetlen esetben áll fenn ilyen irányú egyenlőtlenség, és ez az i levél esete, akkor az $(n+1)$-edik levelet az i-t u-val összekötő élhez kötjük be. Ha egyenlőtlenség más esetben áll fenn, akkor vesszük azt a maximális részfát, amelynek egy levele u, és tartalmazza az $(n+1)$-edik levél bekötését. Definiáljuk $d_{u,n+1}$-et mint $d_{i,n+1} - d_{i,u}$, és ezután u-t i-nek átnevezve folytatjuk a bekötés helyének keresését. Ha minden esetben egyenlőséget kapunk, akkor az $(n+1)$-edik levelet az u csúcshoz kötjük be.

Miután megtörtént a bekötés helyének megkeresése, meghatározzuk a bekötő él hosszát. Ha az $(n+1)$-ik élt az i-t u-val összekötő élen kötjük be, akkor jelöljük a bekötő csúcsot u_1-gyel (13.6/b ábra). Definiáljuk $d_{u,n+1}$-et $(d_{l,n+1} - d_{l,u})$-ként. Ekkor a d_{u,u_1}, d_{i,u_1}, valamint a $d_{u_1,n+1}$ távolságokat az i, u és $n+1$ objektumok távolságait reprezentáló fa adja meg, melyet a (13.67)-(13.69) egyenletek alapján számolunk ki. Ha az $(n+1)$-edik levelet az



13.6. ábra. Néhány fa topológia a 13.7. tétel bizonyításához.

u csúcshoz kötjük be, akkor $d_{u,n+1} = d_{i,n+1} - d_{i,u}$.

Ezután rátérünk a konstrukció helyességének a bizonyítására. Először azt látjuk be, hogy amikor az $(n+1)$-ik levél bekötésének a helyét keressük, és egy új részfán definiáljuk a $d_{u,n+1}$ távolságot, akkor a megadott definíció jól definiált, azaz bármely olyan j csúcsra, mely nem szerepel az újonnan definiált részfában, $d_{j,n+1} - d_{j,u} = d_{i,n+1} - d_{i,u}$. Ha az új részfa tartalmazza l-t, akkor ez azon $j = k$ csúcsra nyilván teljesül, mely k csúcs alapján határoztuk meg az $(n+1)$-edik levél helyzetét (lásd 13.6/a ábra). Az $(n+1)$-edik levél helyzetéből és az additív metrika tulajdonságából adódóan

$$d_{k,n+1} + d_{i,l} = d_{i,n+1} + d_{k,l} \,, \tag{13.76}$$

amiből ha felhasználjuk a $d_{i,l} = d_{i,u} + d_{u,l}$ és a $d_{k,l} = d_{k,u} + d_{u,l}$ egyenlőségeket, adódik, hogy

$$d_{k,n+1} - d_{k,u} = d_{i,n+1} - d_{i,u} \,. \tag{13.77}$$

Ugyanígy minden olyan k_1 levélre, melyet nem választ el k-tól az u csúcs, fennáll a

$$d_{k_1,n+1} + d_{i,l} = d_{i,n+1} + d_{k_1,l} \tag{13.78}$$

egyenlőség. Ez az additív metrikából és a

$$d_{k,k_1} + d_{l,n+1} < d_{k,n+1} + d_{k_1,l} \tag{13.79}$$

egyenlőtlenségből adódik, ez utóbbi pedig levezethető a

$$d_{k,k_1} + d_{i,l} < d_{k_1,l} + d_{k,i} \tag{13.80}$$

és

$$d_{l,n+1} + d_{k,i} < d_{i,l} + d_{k,n+1} \tag{13.81}$$

egyenlőtlenségekből. Hasonlóan, ha u fokszáma háromnál nagyobb, akkor minden olyan csúcsra, melyet u elválaszt az $(n+1)$-edik levéltől, hasonló egyenlőségek és egyenlőtlenségek állnak fenn.

Az új élhosszak kiszámolásából adódik, hogy a $d_{i,n+1}$ távolságot helyesen reprezentálja az új fa, és így a $d_{j,n+1}$ távolságot az összes olyan j csúcsra, melyet i elválaszt $(n+1)$-től. (Ne feledjük el, hogy a beágazás helyének megkereséséből adódóan az ábrán i lehet egy korábbi u.)

Ha az $(n+1)$-edik levél bekötése az u-t az i-vel összekötő szakaszon van (13.6/b ábra), akkor $d_{u,n+1}$ definíciójából adódóan $d_{l,n+1}$-et is helyesen ábrázolja a fa. A

$$d_{k,n+1} + d_{i,l} = d_{k,i} + d_{l,n+1} \tag{13.82}$$

egyenlőségből egyszerűen levezethető, hogy

$$d_{k,n+1} = d_{k,u} + d_{u,n+1} \ , \tag{13.83}$$

így a $d_{k,n+1}$ távolságot is jól reprezentálja a fa. Az előbbi levezetésekkel analóg módon belátható, hogy minden olyan k_1-re, melyet u nem választ el k-tól, a $d_{k_1,n+1}$ távolságot helyesen reprezentálja a fa, és valójában az összes olyan j csúcsra, melyet u elválaszt $n+1$-től, a $d_{j,n+1}$ távolságot helyesen reprezentálja az elkészített fa.

Ha az $(n+1)$-edik levelet az u csúcshoz kötjük be (13.6/c ábra), akkor a

$$d_{i,n+1} + d_{k,l} = d_{k,i} + d_{l,n+1} = d_{k,n+1} + d_{j,i} \tag{13.84}$$

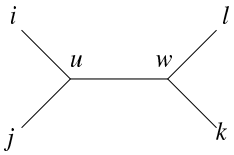
egyenlőségekből már levezethető, hogy mind $d_{k,n+1}$-et, mind $d_{l,n+1}$-et helyesen reprezentálja a fa, és a fentiekhez hasonló okoskodásból következik, hogy ez valójában igaz minden olyan csúcsra, melyet u elválaszt $(n+1)$-től.

Ezzel n csúcs távolságait helyesen reprezentáló fából kiindulva megkonstruáltunk egy olyan fát, mely $n+1$ csúcs távolságát reprezentálja helyesen (feltéve, hogy a csúcsokra teljesül az additív metrika), így bizonyítottuk a 13.7. tételt. ■

Könnyen belátható, hogy a fenti algoritmus, mely megkonstruálja azt a fát, amely egy additív metrikát reprezentál, $O(n^2)$ időt igényel. Az algoritmus azonban csak akkor működik helyesen, ha a bemenő távolságok additív metrikát alkotnak. Ellenkező esetben több esetben is fennállhat a (13.75) egyenlőtlenség, így nem tudnánk eldönteni, hogy hol kössük be az $(n+1)$-edik levelet. Az alábbiakban megadunk egy $\Theta(n^3)$ idejű algoritmust, amely szintén az additív metrikát reprezentáló fát adja vissza, ha a bemenő távolságok additív metrikát alkotnak, de egy additív fát ad vissza egyéb esetekben is.

A Szomszédok-egyesítése algoritmus a következőképpen működik: Adott csúcsok egy halmaza n elemmel, és egy ezen értelmezett metrika, d. Először kiszámítjuk az összes i csúcsra a többi csúcstól vett távolságok összegét:

$$v_i = \sum_{j=1}^{n} d_{i,j} \ . \tag{13.85}$$



13.7. ábra. Az i, j, k és l csúcsok elhelyezkedése, amennyiben i-t és j-t egyetlen u belső csúcs választja el.

Ezután megkeressük azt a csúcspárt, melyre

$$s_{i,j} = (n-2)d_{i,j} - v_i - v_j \tag{13.86}$$

minimális. Az i és a j csúcsokból az új, u belső csúcsig húzott élek hossza

$$e_{i,u} = \frac{d_{i,j}}{2} - \frac{v_i - v_j}{2n-4}\,, \tag{13.87}$$

illetve

$$e_{j,u} = \frac{d_{i,j}}{2} - e_{i,u} \tag{13.88}$$

Ezután következik a távolságmátrix átszámolása. Az i és j csúcsok kiesnek, helyükre kerül be az u csúcs. Az u csúcs és a többi csúcs közötti távolságot az alábbi képlettel határozzuk meg:

$$d_{k,u} = \frac{d_{k,i} + d_{k,j} - d_{i,j}}{2}\,. \tag{13.89}$$

13.8. tétel. *Ha a bemenő csúcsokon megadott d metrika additív metrika, akkor a* Szomszédok-egyesítése *algoritmus visszaadja azt a fát, mely ezt a metrikát reprezentálja.*

Bizonyítás. A 13.7. tételből adódóan pontosan egy fa létezik, amely ezt a metrikát ábrázolja. Ha az algoritmusban az újonnan kiválasztott i és j csúcsokat ezen a fán csak egyetlen belső csúcs választja el, akkor egyszerű számolásból adódik, hogy a Szomszédok-egyesítése algoritmus helyesen jár el. Így elég azt bizonyítani, hogy a kiválasztott i és j csúcsok mindig a megadott módon helyezkednek el.

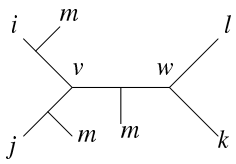
Először azt látjuk be, hogy ha i-t és j-t csak egyetlen egy belső csúcs választja el, akkor bármely k-ra $s_{i,j} < s_{i,k}$ és $s_{i,j} < s_{k,j}$. Valóban, alkalmazva a (13.86) egyenletben szereplő definíciót, az $s_{i,j} < s_{i,k}$ egyenlőtlenséget átrendezve kapjuk, hogy

$$\sum_{l\neq i,j}(d_{i,j} - d_{i,l} - d_{j,l}) - 2d_{i,j} - \sum_{m\neq j,k}(d_{j,k} - d_{j,m} - d_{k,m}) + 2d_{j,k} < 0 \tag{13.90}$$

Amennyiben $l = m \neq i, j, k$, a szummákból kapjuk, hogy

$$(d_{i,j} - d_{i,l} - d_{j,l}) - d_{j,k} + d_{j,l} + d_{k,l} = 2d_{w,l} - 2d_{u,l} < 0 \tag{13.91}$$

(lásd még 13.7. ábra). A szummán kívüli tagok, valamint a szummán belül az $l = k$ és $m = i$ esetek pedig pontosan 0-ra összegződnek, így bizonyítottuk, hogy a (13.90) egyenlőtlenség fennáll.



13.8. ábra. Az m csúcsok lehetséges elhelyezkedése a fán.

Ezután a 13.8. tételt indirekt módon bizonyítjuk. Tegyük fel, hogy az i-t és j-t nem egyetlen belső csúcs választja el, de $s_{i,j}$ minimális. A fentiekből következik, hogy sem i-hez, sem j-hez nem található olyan csúcs, melyet csak egy belső csúcs választ el i-től, illetve j-től. Keressünk olyan k és l párt, melyet csak egyetlen w belső csúcs választ el, i-t w-vel összekötő út és i-t j-vel összekötő út utolsó közös csúcsa pedig legyen v. $s_{i,j}$ minimalitásából

$$s_{k,l} - s_{i,j} > 0 \,. \tag{13.92}$$

Ezt átrendezve kapjuk, hogy

$$\sum_{m_1 \neq k,l} (d_{k,l} - d_{m_1,k} - d_{m_1,l}) - 2d_{k,l} - \sum_{m_2 \neq i,j} (d_{i,j} - d_{m_2,i} - d_{m_2,k}) + 2d_{i,j} > 0 \,. \tag{13.93}$$

A szummákon kívüli tagok, valamint az $m_1 = k$, $m_1 = l$, $m_2 = i$ és $m_2 = j$ esetek pontosan 0-ra összegződnek. A többi $m = m_1 = m_2 \neq i, j, k, l$ esetekben a kérdéses kifejezés

$$d_{k,l} - d_{m,k} - d_{m,l} - d_{i,j} + d_{m,i} + d_{m,k} \,. \tag{13.94}$$

Ha m az i-t j-vel összekötő úton kapcsolódik az i, j, k és l levelek által kifeszített részfához, akkor a (13.94) kifejezés mindig negatív lesz (lásd 13.8. ábra). Nevezzük ezen m csúcsokat I. esetnek. Ha m a v és w közötti úton kötődik be, akkor a (13.94) kifejezés lehet pozitív. Nevezzük ezen eseteket II. esetnek. Mivel a teljes kifejezésnek pozitívnak kell lennie, ezért adódik, hogy a II. esetek száma több kell, hogy legyen, mint az I. esetek száma.

Tudjuk, hogy az i-t v-vel összekötő úton van egy v' csúcs, és ebből kiindulva találunk olyan k' és l' csúcsokat, melyeket csak egyetlen w' belső csúcs választ el. Ezekre megint a II. esetek száma több kell, hogy legyen, mint az I. esetek száma, de ezzel ellentmondásra jutunk a k és l csúcsok esetével. Így i és j szomszédok kell, hogy legyenek, és ezzel bizonyítottuk a 13.8 tételt. ■

Gyakorlatok

13.5-1. Mutassuk meg, hogy ultrametrikában bármely három csúcsból származó három távolság vagy mind azonos, vagy kettő azonos, a harmadik pedig ezeknél kisebb. Bizonyítsuk be az additív metrikák esetében a tetszőleges négy csúcsból származó távolságösszegekre fennálló analóg állítást is.

13.5-2. Mutassuk meg, hogy minden ultrametrika egyben additív metrika is.

13.5-3. Adjunk példát olyan metrikára, amely nem additív.

13.5-4. Mutassuk meg, hogy minden additív metrika euklideszi.

13.5-5. Adjuk meg a pontos képletet, amely a centroid módszer esetében meghatározza $d_{u,k}$-t $d_{i,j}$, $d_{i,k}$ és $d_{j,k}$ segítségével.
13.5-6. Mutassuk meg, hogy a csúcsok számának négyzetével arányos időben eldönthető, hogy egy metrika additív-e, illetve ultrametrikus-e.

13.6. Válogatott témák

Ebben a részben olyan témákkal foglalkozunk, amelyek általában nem szerepelnek bioinformatikai tankönyvekben, vagy csak vázlatosan vannak tárgyalva. Mi is csak a legfontosabb eredményeket említjük meg, és a tételeket nem bizonyítjuk.

13.6.1. Genomok átrendeződése

Egy organizmus genomja különböző génekből áll. A kétszálú DNS-nek csak az egyik szála a kódoló gén, a másik szál ennek a reverz komplementere. Mivel a DNS irányított, így beszélhetünk a gének irányítottságáról is. Ha minden génből egyetlen másolat található a genomban, akkor a gének sorrendje leírható egy előjeles permutációként, ahol az előjel megadja a kódoló szál irányát.

Ha adva van két genom azonos géntartalommal, előjeles permutációként ábrázolva, akkor a feladat az, hogy keressük meg azt a minimális mutációsorozatot, amely az egyik genomot a másikba transzformálja. Három mutációtípust különböztetünk meg:

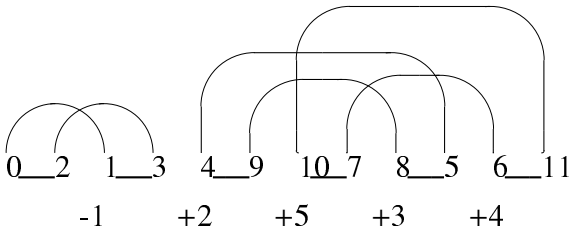
- *Inverzió.* Egy inverzió a genom egy darabját megfordítja. Az adott darabon a gének sorrendje, és a leolvasási irány, azaz az előjel is megváltozik.
- *Transzpozíció.* Egy transzpozíció a genom egy darabját egy másik helyre teszi át, úgy, hogy a gének leolvasási iránya nem változik meg.
- *Invertált transzpozíció.* Ennek hatására nem csak a genom egy darabjának a helyzete változik meg, de az elmozdított darab leolvasási iránya is megváltozik.

Ha feltesszük, hogy csak inverziók történtek, akkor meg tudunk adni egy $\Theta(n^2)$ idejű algoritmust, amely meghatároz egy olyan minimális mutációsorozatot, amely az egyik genomot a másikba transzformálja, sőt, a szükséges mutációk száma $\Theta(n)$ időben eldönthető, ahol n a gének száma.

Ha más, vagy többfajta mutációtípust veszünk figyelembe, akkor a probléma bonyolultsága nem ismert. Transzpozíciókra a legjobb közelítés egy 1.5-közelítés. Ha mind a három fajta mutációt figyelembe vesszük, akkor a legjobb eredmény egy 2-közelítő algoritmus. Ezenkívül súlyozott mutációkra létezik egy $(1 + \epsilon)$-közelítés is, de a súlyok specialitása miatt tudjuk, hogy egy legkisebb súlyú mutációsorozatban nem lesz invertált transzpozíció.

Ha az előjeleket nem ismerjük, és csak inverziókat veszünk figyelembe, akkor a probléma bizonyítottan NP-teljes. Ugyanígy NP-teljes probléma az optimális inverziós medián megtalálása három előjeles permutáció esetében. Az optimális inverziós medián az az előjeles permutáció, melynek a három előjeles permutációtól vett távolságainak az összege minimális.

Az alábbiakban vázoljuk a két genom inverziós távolságának meghatározására szolgáló elméletet, az úgynevezett Hannenhalli–Pevzner-elméletet. Ahelyett, hogy egy π_1 permutációt transzformálnánk a π_2 permutációba, a $\pi_2^{-1}\pi_1$-et transzformáljuk az identikus permu-



13.9. ábra. A $-1, +2, +5, +3, +4$ előjeles permutáció ábrázolása nem előjeles permutációként, és a permutáció töréspontgráfja.

tációba. Egyszerű csoportelméleti okoskodásból következik, hogy a két feladat egymással ekvivalens. Ezért feltesszük, hogy a két genomból a keresett $\pi_2^{-1}\pi_1$ permutációt már meghatároztuk, a továbbiakban ezt π-vel jelöljük.

Egy n elemű előjeles permutációt egy $2n$ hosszú előjel nélküli permutációval ábrázolunk a következőképen. Minden $+i$-t lecserélünk egy $2i-1, 2i$ párra, minden $-i$-t pedig egy $2i, 2i-1$ párra. Ezenfelül az így kapott permutációt 0 és $2n+1$ közé keretezzük. Ezután elkészítjük az úgynevezett töréspont-gráfot. Ennek csúcsai a nem előjeles permutáció elemei, beleértve a 0-t és a $(2n+1)$-et is. A permutáció két elemét egy egyenes vonallal kötjük össze, ha a különbségük abszolútértéke nagyobb, mint egy. Valamint két csúcsot egy ívvel kötünk össze, ha egymás utáni számok, de a permutációban nem egymás után állnak. Egy példát adunk meg a 13.9. ábrán. Könnyen belátható, hogy a töréspont-gráfot egyértelműen fel lehet bontani körökre, a körökben az egyenes élek és ívek felváltva jönnek. Egy kört irányítottnak hívunk, ha egy, a körön megtett séta során legalább egy egyenes élen balról jobbra, és legalább egy egyenes élen jobbról balra is haladtunk. Minden más kör irányítatlan.

Két kör átfed, ha valamely íveik szükségképpen metszik egymást. A permutáció átfedési gráfjainak a csúcsai a töréspont-gráf körei, és két csúcs akkor van összekötve, ha a töréspont-gráfban a két kör metszi egymást. Az átfedési gráf komponensekre bomlik, egy komponens irányított, ha van benne irányított kör, egyébként irányítatlan. Az irányítatlan komponensek közül ***nem-gátaknak*** hívjuk azokat, amelyekre a töréspont-gráfon van két olyan irányítatlan komponens, amelyet az adott komponens elválaszt egymástól. Itt az elválasztáson azt értjük, hogy az egyik irányítatlan komponens valamely ívéből nem tudunk a csúcsokat összekötő vonal felett úgy átmenni a másik irányítatlan komponens valamely ívéhez, hogy ne metszenénk a nem-gát valamely ívét. A többi irányított komponenst ***gátnak*** hívjuk.

A gátak közül szupergátaknak hívjuk azokat, amelyeket ha kitörlünk, akkor valamely nem-gát gáttá válik. Ez olyan esetekben fordul elő, amikor a nem-gát pontosan az adott gátat választja el más nem-irányított komponensektől. Egy permutációt erődnek hívunk, ha páratlan számú gátja van, és ezek mindegyike szupergát.

13.9. tétel. *Legyen adva egy π előjeles permutáció. Inverziók egy optimális sorozata, amely ezt a permutációt rendezi,*

$$b_\pi - c_\pi + h_\pi + f_\pi \tag{13.95}$$

mutációból áll, ahol b_π a töréspont-gráfban az egyenes élek száma, c_π a töréspont-gráfban a körök száma, h_π a gátak száma, és $f_\pi = 1$, ha π erőd, egyébként pedig 0.

A tételt itt nem bizonyítjuk.

A (13.95) képletben szereplő mennyiséget meg lehet $\Theta(n)$ időben határozni, ahol n az előjeles permutáció mérete.

Nyílván b_π és c_π kiszámolható $O(n)$ időben. A nehéz rész h_π és f_π kiszámolása. A problémát az okozza, hogy az átfedési gráfban az élek száma lehet $\Omega(n^2)$. Ezért a gyors algoritmus nem határozza meg a teljes átfedési gráfot, hanem csak minden komponensén egy feszítő részfát.

13.6.2. Sörétes-puska nukleinsavleolvasás

Egy genom DNS-e általában milliós nagyságrendű, vagy még több nukleinsavból áll. Egy biokémiai technikával meghatározható a DNS egyik végén található nukleinsavak sorrendje, de a leolvasási bizonytalanság növekszik, ahogy haladunk a szekvenciában előre, és kb. 500 nukleinsav után a leolvasás teljesen bizonytalanná válik.

Ezt a biokémiai problémát a következőképpen oldják meg. A DNS-ből számos kópiát vesznek, és ezek mindegyikét véletlen módon széttördelik olyan méretű részekre, melyet az előbb leírt technikával aztán már meg lehet határozni. Ezek után az átfedő részletekből kell összerakni az eredeti hosszú szekvenciát. Ezt a technikát hívjuk ***sörétes-puska*** nukleinsav-leolvasásnak, angolul *shotgun sequencing*-nek.

Matematikailag úgy lehet definiálni a feladatot, hogy adott szekvenciáknak keressük a legrövidebb közös szuperszekvenciáját. Egy B szekvencia szuperszekvenciája A-nak, ha A részszekvenciája B-nek (részszekvencián egy szekvencia nem feltétlenül összefüggő részét értjük). Maier bebizonyította, hogy a legrövidebb szuperszekvencia NP-teljes probléma, ha az ábécé mérete legalább 5, és sejtése szerint ugyanez a helyzet a legalább háromelemű ábécék esetén is. Később megmutatták, hogy a feladat minden nem triviális ábécére NP-teljes.

Hasonló a legrövidebb közös szuperstring probléma, ami szintén NP-teljes (részstringen egy szekvencia összefüggő részét értjük). Ez utóbbi probléma az, ami igazán biológiailag érdekes, hiszen átfedő stringeket keresünk. A megoldásra számos közelítő algoritmus született. Egy mohó algoritmus minden stringpárra megkeresi a maximális átfedéseket, majd ezt próbálja mohó módon összefűzni egy legrövidebb szuperstringgé. Az algoritmus futási ideje $O(Nm)$, ahol N a szekvenciák száma, m pedig a szekvenciák összhossza. Az így megtalált szuperstring mérete bizonyítottan kisebb, mint $4n$, ahol n a legrövidebb szuperstring hossza. Egy továbbfejlesztett algoritmus bizonyítottan 3-közelítő, és a sejtés az, hogy valójában sose kapunk $2n$-nél hosszabb szuperstringet.

A sörétes-puska nukleinsav-leolvasás során a nukleinsavak meghatározása nem tökéletes, előfordulhatnak beszúrások, törlések és cserék is a meghatározás közben. Ezért Jiang és Li javasolta a legrövidebb k-közelítő közös szuperstring problémát. Kececioglu és Myers egy programcsomagot dolgozott ki, amelyben számos heurisztikus algoritmust megvalósítottak a probléma megoldására.

Gyakorlatok

13.6-1. Mutassuk meg, hogy ha egy permutáció erőd, akkor legalább három szupergát van benne.

13.6-2. Legalább hány elemből kell egy erődnek állnia?

Feladatok

13-1. Konkáv Smith–Waterman
Adjuk meg a Smith–Waterman algoritmust konkáv résbüntetésekre.
13-2. Konkáv Spouge
Adjuk meg Spouge algoritmusát konkáv résbüntetésekre.
13-3. Kiszolgálás benzinkútnál
Egy benzinkútnál két sorban állnak a kocsik. Mindegyik kocsit vagy gázolajjal vagy benzinnel kell kiszolgálni. Egyszerre legfeljebb két kocsit szolgálhatunk ki, de csak akkor, ha a két kocsi különböző üzemanyagot igényel, és a két sor első kocsijairól van szó, vagy valamelyik sor első két kocsijáról. Akár egy, akár két kocsit szolgálunk ki egyszerre, a két folyamat kiszolgálási ideje ugyanakkora. Adjunk meg egy páros rejtett Markov-folyamatot, amelyre a Viterbi-algoritmus egy legrövidebb kiszolgálási tervet határoz meg.
13-4. Rejtett Markov-modellek momentumai
Adott egy szekvencia és egy rejtett Markov-modell. Számoljuk ki a rejtett Markov-modellben az adott szekvenciát kibocsátó utak valószínűségének a várható értékét, varianciáját, k-adik momentumát.
13-5. Sztochasztikus környezetfüggetlen nyelvtanok momentumai
Adott egy szekvencia és egy sztochasztikus környezetfüggetlen nyelvtan. Számoljuk ki az adott nyelvtanban a szekvenciát levezető levezetések valószínűségeinek a várható értékét, varianciáját, k-adik momentumát.
13-6. Rejtett Markov-modellek együttes kibocsátási valószínűsége
Ki lehet számolni ezt a valószínűséget $O((n_1 n_2)^2)$ időben?
13-7. Rendező inverziók
Egy adott előjeles permutációban rendező inverzióknak hívjuk azokat az inverziókat, amelyek kezdő lépései egy minimális hosszúságú rendező sorozatnak. Hogyan változtathatja meg egy rendező inverzió a töréspont gráfban a körök, töréspontok és a gátak számát?

Megjegyzések a fejezethez

Dinamikus programozási algoritmust biológiai szekvenciák hasonlóságára először Needleman és Wunch adott meg 1970-ben [344]. Tetszőleges résbüntető függvényre Waterman és munkatársai adtak algoritmust [487]. Gotoh algoritmusa affin résbüntető függvényekre 1982-ben jelent meg [165], a konkáv résbüntető függvény ötlete Watermantól származik [486], mellyel később Miller és Myers [332], valamint Galil és Giancarlo [332] foglalkozott. Bár empirikus adatok alapján a konkáv résbüntető függvény biológiailag helyesebb, mégis a leggyakrabban az affin résbüntető függvényt használják, pl. a Clustal-W nevű népszerű szekvenciaillesztő programban is [465].

A többszörös szekvenciaillesztés ötlete Sankofftól származik [411], mely a bioinformatika egyik központi feladata lett [182]. Bebizonyították, hogy a többszörös szekvenciaillesztés NP-teljes probléma [484], így a gyakorlatban heurisztikus módszereket alkalmaznak. A legelterjedtebb heurisztika az iteratív illesztés, ez alapján működik a fent említett Clustal-W is.

Hirschberg algoritmusát először leghosszabb közös részszekvencia megkeresésére

használták [207], de mára számos bioinformatikai algoritmusban szerepel, pl. Doublescan [324]. A szekvenciaillesztés további algoritmikai elemzéséről kimerítően ír Gusfield [182].

A sztochasztikus nyelvtanok és bioinformatikai alkalmazásuk a központi témája Durbin, Eddy, Krogh és Mitchison népszerű könyvének, melyet számos egyetem bioinformatika oktatásában alapkönyvként használnak [116]. Formális nyelvekkel, nyelvtanokkal foglalkozó magyar nyelvű tankönyv Bach Iván [25] és Fülöp Zoltán [131] műve, valamint [90].

Az osztályozó algoritmusok összehasonlításáról részletesen olvashatunk Podani János könyvében [374]. A filogenetika és a filogenetikai algoritmusok iránt érdeklődők figyelmébe ajánljuk Felsenstein [126], valamint Semple és Steel [424] könyveit.

Kevéssé olvasmányos, de sok információt tartalmaz a genomátrendeződési algoritmusokról Pevzner könyve [368].

Stephen „String searching” című könyvében részletesen foglalkozik a legrövidebb szuperstring problémájával. A könyvben számos kiváló referenciát és az algoritmusok részletes leírásait is megtaláljuk [437].

Az alábbi megjegyzések a téma iránt kifejezetten érdeklődőknek szólnak.

Két string edit távolságán a minimális számú beszúrás-törlések számát értjük. Két string edit távolságának a kiszámolására van $\Theta(l^2)$-nél gyorsabb módszer, amely a „négy orosz gyorsítása” néven híresült el, habár a négy egykori szovjetúnió-beli szerző közül csak egy volt orosz nemzetiségű [18]. Az algoritmus futási ideje $O(n^2/\lg(n))$, azonban gyakorlati alkalmazásokban előforduló szekvenciahosszakra lassabb, mint a hagyományos dinamikus programozási algoritmusok.

A leghosszabb közös részszekvenciára használt dinamikus programozási algoritmus a két szekvencia hosszainak szorzatával arányos időben fut, hasonlóan a szekvenciák illesztéseire használt algoritmushoz. Hunt és Szymanski módszere ezzel szemben egy olyan gráfot állít elő, melynek csúcsai a két összehasonlítandó szekvencia, A és B karakterei, és a_i-t pontosan akkor köti össze él b_j-vel, ha $a_i = b_j$. Ezt a gráfot használva létezik olyan algoritmus, melynek futási ideje $\Theta((r+n)\lg(n))$, ahol r a gráf éleinek a száma, n pedig a gráf csúcsainak a száma, azaz a két szekvencia hossza [215]. Habár így az algoritmus futási ideje $O(n^2 \lg n)$, számos alkalmazásban a behúzott élek száma a szekvenciák hosszával egyenesen arányos. Ekkor a futási idő $O(n \lg n)$ lesz, ami lényegesen jobb a kvadratikus idejű algoritmusnál.

A saroklevágási technika egyik fejlett változata az úgynevezett diagonális kiterjesztés. A diagonális kiterjesztés a dinamikus programozási táblázatot átlós irányban tölti ki, és nem igényel tesztértéket. Ilyen algoritmusra példa Wu és munkatársainak az algoritmusa [501]. A Unix `diff` utasításában használt algoritmus szintén diagonális kiterjesztésen alapul [331], melynek az átlagos számolási ideje $O(n + m + d_e^2)$, ahol n és m a két összehasonlítandó szekvencia hossza, d_e pedig a két szekvencia edit távolsága.

A Knuth–Morris–Pratt stringkereső algoritmus egy rövid P stringet keres meg egy hosszú M stringben, és $\Theta(p + m)$ időben fut, ahol p és m a két szekvencia hossza [261]. Landau és Vishkin módosított algoritmusa minden olyan illesztést megtalál M-ben, amely legfeljebb k helyen tér el P-től [277]. Az algoritmus futási ideje $\Theta(k(p \lg(p) + m))$, memóriaigénye pedig $\Theta(k(p + m))$.

Bár a szekvenciaillesztésre legelterjedtebb algoritmusok dinamikus programozással működnek, megadható szekvenciák optimális illesztése egész lineáris programozással is. Az ötlet Kececioglu és munkatársaitól származik [250]. A módszert kiterjesztették tetszőleges résbüntető függvényre is [10]. Az egész értékű programozással kapcsolatos algoritmusokról írt áttekintő cikket Lancia [276]. A DiAlign szintén nem dinamikus programozáson alapul

[337].

A szekvenciák strukturális illesztése csak a fehérjék három dimenziós szerkezetét veszi figyelembe. Olyan szekvenciaillesztést keresünk, melyben a réseket büntetjük, az összeillesztett karaktereket viszont nem hasonlóság vagy távolság alapján értékeljük, hanem az alapján, hogy a három dimenziós térszerkezetben az összeillesztett aminosavak mennyire hozhatóak fedésbe a szerkezetek forgatásával. Számos algoritmust dolgoztak ki strukturális szekvenciaillesztésre, ezek közül az egyik legnépszerűbb a Kombinatorikus Kiterjesztés (CE) algoritmus [430].

Egy adott fa topológiára a Maximum Likelihood címkézés polinomiális időben megadható [378]. Ez az algoritmus az egyik legelterjedtebb filogenetikai elemző csomagba, a PAML-ba is bekerült (http://abacus.gene.ucl.ac.uk/software/paml.html).

Egy adott fa topológia, élhosszak, szubsztitúciós modell és adott szekvenciák esetén lineáris időben ki lehet számolni egy fa likelihoodját Felsenstein algoritmusával. A Maximum Likelihood fa problémája az, hogy adott modell és szekvenciák esetén határozzuk meg azt a fa topológiát és élhosszakat, amelyre a likelihood maximális. Meglepő módon, még senkinek sem sikerült bizonyítania, hogy ez a probléma NP-teljes lenne, bár ez a sejtés. Azt már sikerült bizonyítani, hogy az Ancestral Maximum Likelihood probléma, amikor nem csak a fa topológiáját és élhosszait, hanem a belső csúcsok legvalószínűbb címkézését is keressük, NP-teljes [2].

A két legelterjedtebb, rejtett Markov-modellek alapján működő szekvenciaillesztő programcsomag a SAM [213] és a HMMER (http://hmmer.wustl.edu/). Rejtett Markov-modell alapján működő genomannotáló programot fejlesztett ki Pedersen és Hein [364], páros rejtett Markov-modellt használ szintén genomannotálásra a DoubleScan [324], (http://www.sanger.ac.uk/Software/analysis/doublescan/) és a Projector [325], (http://www.sanger.ac.uk/Software/analysis/projector/).

Olyan rejtett Markov-modellt, amely evolúciós információk alapján határozza meg a kibocsátási valószínűségeket, Goldman, Thorne és Jones publikált először [157], fehérjék másodlagos térszerkezetének jóslására. Ez a rejtett Markov-modell szekvenciák illesztett oszlopait bocsátja ki, egy illesztett oszlop kibocsátási valószínűségét egy evolúciós fa és egy időfolytonos Markov-modell határozza meg. A kibocsátási valószínűséget Felsenstein algoritmusával lehet meghatározni.

A Knudsen-Hein nyelvtant használja a PFold nevű program, amely RNS-ek másodlagos térszerkezetét határozza meg [257]. Ez a sztochasztikus környezetfüggetlen nyelvtan szintén illesztett szekvenciákat vezet le, a terminális szimbólumok ezen szekvenciaillesztés illesztett oszlopai. Az s terminális levezetési valószínűségét egy evolúciós törzsfa és egy időfolytonos Markov-modell határozza meg, a dFd levezetésben a két d terminális helyére írandó két oszlop valószínűségét pedig ugyanezen a törzsfán egy olyan időfolytonos Markov-modell adja meg, melynek állapotai dinukleotidok.

Az Előre algoritmus futási ideje négyzetesen növekszik a rejtett Markov-modell állapotainak a számával. Azonban nem mindig ez a leghatékonyabb algoritmus. Egy biológiailag fontos többszörös rejtett Markov-modellben az Előre algoritmus futási ideje $\Theta(5^n L^n)$, ahol n a szekvenciák száma, L pedig a szekvenciák hosszának geometriai közepe. Meg lehet adni azonban egy olyan algoritmust, amely $\Theta(2^n L^n)$ időben kiszámolja a szekvenciák kibocsátási valószínűségét [299, 300]. Nem ismert azonban, hogy erre a modellre a legvalószínűbb kibocsátás megkeresésére hasonló gyorsítás megadható-e.

Az RNS-ek Zuker-Tinoco [466] modellje térszerkezeti elemeken definiál szabadener-

giát, és egy adott másodlagos térszerkezethez a térszerkezet részeihez rendelt szabadenergiák összegét rendeli. A Zuker–Sankoff-algoritmus [513] $\Theta(l^4)$ időben találja meg a legkisebb energiatartalmú másodlagos térszerkezetet, $\Theta(l^2)$ memóriát használva, ahol l az RNS szekvencia hossza. Ugyanilyen memória- és számolásigényű a térszerkezetek Boltzmann-eloszlásához tartozó partíciófüggvény kiszámolása is [317]. Egy speciális esetben mind az optimális térszerkezet, mind a partíciófüggvény számolása felgyorsítható $\Theta(l^3)$ időre is, továbbra is $\Theta(l^2)$ memóriát használva [303].

Az RNS-ek azon bázispárosodásait, melyben a párosodó nukleinsavakat összekötő, a szekvencia felett haladó ívek metszik egymást, álcsomóknak hívjuk. Az álcsomókat tartalmazó térszerkezetek általános predikciója NP-teljes. Azonban bizonyos speciális álcsomók közül egy optimális álcsomó megkeresésére vannak polinomiális idejű algoritmusok [9, 302, 390, 471].

Rejtett Markov-modellek és sztochasztikus környezetfüggetlen nyelvtanok együttes kibocsátási valószínűségeinek meghatározásával foglalkoztak Jagota és munkatársai [226]. A keresett valószínűséget egy kvadratikus egyenletrendszer megoldásával lehet meghatározni, amelyet csak numerikusan lehet megoldani. Az egyenletrendszerben az ismeretlenek száma Vn^2, ahol V a nemterminálisok száma Chomsky féle normálformában, n pedig a rejtett Markov-modell állapotai. A szerzők megadtak egy iterációt, mely bizonyítottan konvergál a pontos megoldáshoz, egy iterációs lépés számolási igénye $\Theta(V^3n^5)$. Az iteráció konvergenciájának a sebességéről nem szolgálnak analitikus eredménnyel, de azt sejtik, hogy gyors.

Sztochasztikus környezetfüggetlen nyelvtanok generátorfüggvényeit használta RNS térszerkezetek predikcióinak jóságára Markus Nebel. Számos olyan statisztikát sikerült analitikusan kiszámolnia, amelyek korrelálnak a predikció jóságával [343].

Rendezett fákhoz hasonlóan rendezett erdőket is lehet illeszteni. Rendezett erdők illesztésére adtak meg algoritmust Höchsmann és munkatársai. Megmutatták, hogy RNS térszerkezeteket lehet rendezett erdőkkel ábrázolni, és ezen ábrázoláson keresztül az RNS térszerkezetek összehasonlíthatóak [197].

Atteson matematikai definíciót adott a törzsfakészítő módszerek jóságára, és kimutatta, hogy bizonyos definíciók alapján a Szomszédok egyesítése a lehető legjobb módszer [21].

Négy fajra három fa topológiát lehet készíteni, amelyeket quarteteknek hívunk. Ha egy fára ismerjük az összes quartet topológiát, akkor ebből az eredeti fa topológiáját is meg lehet határozni. Sőt bebizonyították, hogy nem szükséges az összes quartetet megnézni, a közeli fajok quartetjei is egyértelműen meghatározzák a fa topológiáját [120].

Egy genom állhat több DNS szekvenciából, az egyes DNS szekvenciákat kromoszómáknak hívjuk. Genomátrendeződés során nem csak kromoszómán belül, hanem a kromoszómák között is keveredhetnek a gének. Ezekre az úgynevezett transzlokációs mutációkra adott meg $\Theta(n^3)$ idejű algoritmust Hannenhalli [192]. Transzlokációs átmérővel és a probléma általánosításával foglalkozik Pisanti és Sagot [373].

A permutációk rendezésének általánosított problémája a minimális generáló szó keresése egy véges csoportban. Ez a probléma bizonyítottan NP-teljes [229]. Az előjeles permutációk inverziókkal való rendezésén és a transzlokációs távolságon kívül csak egy biológiailag kevésbé releváns problémára van polinomiális idejű algoritmus, az úgynevezett blokk átrendeződésekre [76]. Érdekességként megjegyezzük, hogy a Microsoft tulajdonosa, Bill Gates is foglalkozott permutációk rendezésével, speciálisan a prefix inverziókkal [150].

A könyvfejezetben csak a legfontosabb témákról írtunk, számos kevésbé fontos, de nagyon érdekes témakört kénytelenek voltunk kihagyni. Ilyen témakörök például a rekombi-

náció, a pedigré elemzés, a karakteralapú törzsfakészítő módszerek, a részleges emésztés, a fehérjecsavarás, DNS chipek, a tudásábrázolás, a biokémiai hálózatok. Ennek fényében Donald Knuth szavaival [1112] zárjuk könyvfejezetünket: „It is hard for me to say confidently that, after fifty more years of explosive growth of computer science, there will still be a lot of fascinating unsolved problems at peoples' fingertips, that it won't be pretty much working on refinements of well-explored things. Maybe all of the simple stuff and the really great stuff has been discovered. It may not be true, but I can't predict an unending growth. I can't be as confident about computer science as I can about biology. Biology easily has 500 years of exciting problems to work on, it's at that level."

14. Ember-gép kölcsönhatás

Az interneten a következő definíciót találjuk a témával kapcsolatban: „Az ember és számítógép közötti együttműködéssel foglalkozó tudományág az emberi használatra szánt interaktív számítógépes rendszerek tervezésével, megvalósításával és értékelésével, valamint e téma körül felmerülő jelenségek tanulmányozásával foglalkozik. ... E terület legfontosabb vonatkozásai a következők:

- Az emberek és gépek által elvégzett feladatok közös teljesítménye.
- Az ember és gép közötti kommunikáció szerkezete.
- A gépek használatára vonatkozó emberi képességek (például az interfészek tanulhatósága).
- Az interfész programozása és algoritmusai.
- Az interfészek tervezésekor és kialakításakor felmerülő technikai kérdések.
- Az interfészek specifikálásának, tervezésének és megvalósításának folyamata.

... Az ember és számítógép közötti együttműködésnek a fentiek alapján vannak tudományos, technikai és tervezési kérdései."

A fenti témák közül jó néhány csak távolról érinti a klasszikus értelemben vett algoritmusok témakörét. Ezért ebben a fejezetben elsősorban olyan esetekre koncentrálunk majd, ahol a gépeknek nagy mennyiségű számítást kell elvégezniük, az emberek pedig egyfajta intelligens ellenőrző és irányító szerepet töltenek be.

14.1. Több választási lehetőséget kínáló rendszerek

Az emberek képesek gondolkodni, érezni és érzékelni, és gyorsan tudnak alkalmazkodni egy új szituációhoz. Számolni is tudunk, de abban már nem vagyunk olyan jók. A számítógépek ezzel szemben kiválók a számolásban, csak úgy falják a biteket és bájtokat. Ők azonban a számoláson kívül nem nagyon értenek máshoz, például meglehetősen rugalmatlanok. Ha az embernek és a számítógépnek a képességeit és erősségeit megfelelő módon kombináljuk, az nagyon hatásos teljesítményhez vezethet.

Az ilyenfajta csapatmunkának az egyik lehetséges megközelítése az úgynevezett *több választási lehetőséges optimalizálás*. Egy ilyen ***több választási lehetőséget kínáló rendszer*** esetén a számítógép néhány, könnyen áttekinthető alternatívát kínál föl, mondjuk kettőt, hármat, vagy esetleg négyet, de semmiképpen sem sokkal többet. A végső döntést egy szakértő

hozza meg a felkínált lehetőségek közül választva. Ennek a megközelítésnek egyik legfőbb előnye, hogy az ember nincs hatalmas mennyiségű adattal elárasztva.

A több választási lehetőséget kínáló rendszerek különösen hasznosak lehetnek azoknál a valós idejű problémáknál, amikor alapjában véve elegendő idő áll rendelkezésre a teljes megoldás kiszámolására, de a feladat bizonyos paraméterei ismeretlenek vagy fuzzy jellegűek. Ezek konkrét értékét csak egy meglehetősen kései időpontban tudjuk meg, amikor már nincs elegendő idő bonyolult számításokra. Képzeljük el, hogy a ***döntéshozó*** valamilyen több választási lehetőséget adó algoritmus segítségével előre létrehozott néhány megengedett megoldást. Majd amikor a tényleges adatok birtokába jut, valós időben kiválaszt ezek közül egyet.

Nézzünk egy példát, amelyben egy útvonalat kell megkeresnünk. Tegyük fel, hogy egy kamionsofőrnek az A pontból a Z-be kell eljutnia. Az út megkezdése előtt egy PC szoftver segítségével megkeres két-három megfelelő utat, és kinyomtatja őket. Út közben a rádió információkat ad az aktuális forgalmi dugókról, illetve az időjárási problémákról. Ezekben a helyzetekben a kinyomtatott alternatívák segítenek a sofőrnek, hogy valós időben egy másik útvonalat válasszon.

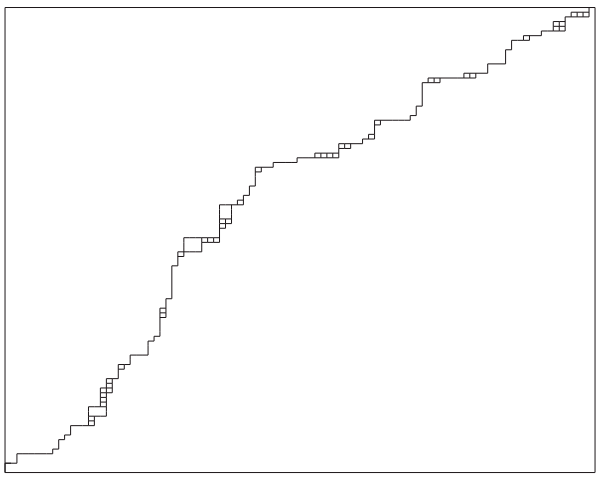
A megfelelő alternatív megoldási lehetőségek megtalálása azonban nem is olyan egyszerű a számítógép segítségével. A legtermészetesebb megoldási módnak az tűnhetne, ha valamilyen k-legjobb algoritmust alkalmaznánk. Ez azt jelenti, hogy adva van egy diszkrét optimalizálási probléma valamilyen célfüggvénnyel, és a k darab legjobb megoldást kell kiszámolni egy előre adott k-ra. Az ilyen k-legjobb megoldások azonban általában egymás *minimális változtatásaiként* jelennek meg, ahelyett, hogy *valódi alternatívák* lennének.

A 14.1. ábra egy tipikus példát mutat erre. Egy 100×100 méretű diagramon az volt a célunk, hogy rövid útvonalakat találjunk a bal alsó sarokból a jobb felsőbe. Az élhosszak véletlen számok, amelyeket nem jelöltünk az ábrán. Az 1000 (!) legrövidebb utat számoltuk ki, és az ábra ezek unióját mutatja. A hasonlóság az egyes utak között feltűnő. Ha az ábrát egy kicsit nagyobb távolságból nézzük, akkor az a benyomásunk támad, mintha azon csak egyetlen útvonal lenne, amit egy bozontos ceruzával rajzoltak. (A 14.2. pontban majd szintén a rövid alternatív útvonalak kiszámítása lesz az egyik legjobb példa.)

Gyakran előfordul, hogy a „több választási lehetőség” fogalmát egy másik értelemben használják, mégpedig a „több választási lehetőséges teszt” értelemben. Ez a dolog teljesen mást jelent. A két fogalom közötti különbség a lehetséges megoldások típusában és számában van:

- A több választási lehetőséges teszt esetén a válaszok közül legalább egy mindig „helyes”, a többi pedig lehet helyes vagy helytelen. A teszt készítője (egy külső szakértő) előre megadja a kérdést, a lehetséges válaszokat, valamint azt, hogy mely válaszok helyesek.

- Az optimalizálási környezetben semmi nem világos előre. Elképzelhető, hogy a lehetséges megoldások valamennyien megfelelőek, de az is előfordulhat, hogy mindannyian rosszak. És általában nincs olyan külső szakértő, aki megmondja az embernek, hogy a választása jó-e vagy sem. Emiatt a bizonytalanság miatt a legtöbb embernek szüksége van valamennyi kezdeti időre, hogy a több választást kínáló rendszeren belül a saját szerepét megismerje és elfogadja.



14.1. ábra. 1000 legrövidebb út egy 100×100-as rács-gráfban, egymásra nyomtatott megjelenítésben.

14.1.1. Példák több választási lehetőséget kínáló rendszerre

1. Rövid útvonalak. Az 1990-es évek elejétől egyre népszerűbbek lettek az útvonalválasztásra megoldást kínáló PC-s programok. 1997-ben a holland AND nevű cég volt az első, amelyik olyan programot árult, amelyik nem csak kiszámolta a „legjobb" (=legrövidebb vagy leggyorsabb) útvonalat, hanem egy vagy két alternatívát is megadott. A felhasználónak lehetősége volt arra, hogy ezeket az alternatívákat egyszerre, illetve egyiket a másik után lássa. A felhasználó további paramétereket is megadhatott, amelyek az útvonalak vizuális tulajdonságait befolyásolták. Ilyen volt például az első, második illetve harmadik legjobb útvonal színe, vastagsága stb. Az ezzel kapcsolatos munkák F. Berger nevéhez fűződnek. Ő fejlesztett ki egy módszert, amelynek segítségével lineáris struktúrák (pl. utak, vasutak, folyók, ...) azonosíthatók szürkeárnyalatos műholdas felvételeken. Általában a lehetséges jelöltek nem egyediek, és Berger algoritmusa néhány további alternatív javaslatot is tartalmaz. Berger módszere a rövid alternatív útvonalakat generáló algoritmusokon alapul.

2. Utazóügynök probléma és lyukak fúrása nyomtatott áramköri lapokba. Az utazóügynök probléma esetén adva van N pont, és ismertek a pontok egymástól mért távolságai. A feladat egy minden pontot érintő, legrövidebb kör megtalálása. Erről a problémáról ismert, hogy NP-teljes. Ennek a feladatnak az egyik fontos alkalmazása az elektronikai iparban a lyukak fúrása nyomtatott áramköri lapokba. Ez esetben a pontoknak azok a helyek felelnek meg, ahová a lyukakat fúrni kell, és a cél a fúrófej mozgásának a minimalizálása. A gyakorlat azt mutatja, hogy ebben a feladatban nem csak a fúrófej mozgási útvonalának hossza az egyetlen feltétele a sikernek. Az útvonaltól függően kisebb-nagyobb feszültségek alakulhatnak ki a nyomtatott áramköri lapkában. A különböző útvonalak különböző feszültségszinteket eredményeznek, amelyeket előre nem nagyon lehet kiszámítani. Ezért célszerűnek tűnik néhány alternatív, kellően rövid útvonal meghatározása, amelyekből ki tudjuk választani azt, amelyik a feszültség minimalizálása szempontjából a legjobb.

3. Internetes kereső motorok. A legtöbb esetben az internetes kereső motorok rengeteg találattal térnek vissza, amelyeket egy átlagos felhasználó nem tud, és nem is akar végig-

nézni. Ezért az ilyen kereső motorok tervezői számára kulcsfontosságú feladat a megfelelő kivonatok készítése. Első számú szabálynak tekinthető, hogy a kapott eredménybeli első tíz találat legyen a leginkább a tárgyhoz tartozó, és legyen kellően szétszórva az eredményhalmazon belül. Ezen a területen, és az e-kereskedelem területén a több választási lehetőséget kínáló rendszereket szokás *tanácsadó rendszereknek* is nevezni.

4. Bolygóközi űrutak röppályái. A távoli bolygókra, kis bolygókra, és üstökösökre való űrutazás a „high-tech" kalandok körébe tartozik. Ezeknél a feladatoknál két kulcsfontosságú tényezőre kell tekintettel lenni. Az egyiket a költségvetési korlátok jelentik, a másik pedig az, hogy az űrszondákat különlegesen nagy sebességre kell felgyorsítani, hogy időben elérjék a céljukat. A rakéták felgyorsításában a gravitáció is segíthet, oly módon, hogy a közbeeső bolygókhoz egészen közel megy el a röppályájuk. Ezzel időt és üzemanyagot is lehet megtakarítani. Az utóbbi években ezek a gravitáció által segített röppályák egyre bonyolultabbak lettek, néha több bolygó-közeli elrepülést is tartalmaztak. A legjelentősebb példák a következők: a Cassini küldetése a Szaturnuszra a Vénusz-Vénusz-Föld-Jupiter sorozatot tartalmazta, a Rosetta küldetése a „67P/Hurjumov-Geraszimenko" üstökösre a Föld-Mars-Föld-Föld sorozatot, a Messenger küldetése a Merkúrra pedig a Föld-Vénusz-Vénusz-Merkúr-Merkúr sorozatot. A röppályák kiszámításának tudománya jelenleg abban tud segíteni, hogy egy korábban meghatározott útvonalat finomítson. Ehhez azonban a mérnököknek megfelelő fantázia és kreativitás segítségével ilyen elsődleges, finomítható útvonalakat kell tervezniük. Ezeknek az elsődleges útvonalaknak a számítógéppel történő generálása még meglehetősen gyerekcipőben jár.

5. Számítógéppel támogatott sakk. Az 1970-es évek végétől kezdtek elterjedni a piacon kapható sakkozó számítógépek. Ezeknek a gépeknek a játékereje fokozatosan növekszik, és ma már a legjobb PC-s programok egy szinten vannak a legjobb sakkozókkal. Az olyan csapatok azonban, amelyekben emberek és számítógépek is részt vesznek, erősebbek a csak emberekből, illetve csak gépekből álló csapatoknál is. E fejezet egyik szerzője (Althöfer) több kísérletet végzett több választási lehetőséget kínáló rendszerekkel. Az egyik ilyen összeállításban, amelyet ***3-agynak*** nevezünk, két különböző sakkprogram fut két független PC-n. Mindkét program javasol egy lépést, amelyek közül egy (emberi) sakkjátékos választ, vagyis ő hozza meg a végső döntést. Néhány kísérlet folyamán a 3-agy kitűnő teljesítményt ért el. A legfontosabb ezek közül egy 1997-es mérkőzés volt, amelyben két, egyenként 2550 Élő pont alatti játékerejű program, és egy amatőr sakkjátékos (1900-as Élővel) 5-3-ra legyőzte az első számú német sakkjátékost, Juszupov nagymestert, akinek Élő pontja 2640 volt. Ezzel a 3-agy átlagos teljesítménye 2700 Élő-pont felettinek felelt meg. Ez után az esemény után a legjobb sakkjátékosok már valahogy nem nagyon akartak a 3-agy csapatok ellen küzdeni. A 3-agy erőssége nagyrészt abban rejlik, hogy két *különböző* sakkbeli tudás kombinálására ad lehetőséget. A számítógépek leginkább a taktikailag helyes lépések megtalálásában jeleskednek, míg az ember erőssége a hosszú távú tervek kiválasztása.

Manapság az összes profi sakkjátékos számítógépes programok segítségével készül a versenyekre, a megnyitásoknak és a partiknak valamilyen több választási lehetőséget kínáló elemzését felhasználva. Még kirívóbb a helyzet a levelezési sakkban, ahol a játékosok hivatalosan is felhasználhatják a számítógép segítségét a játszmáikban.

6. Nyaralási és utazási információk. Amikor valaki a nyaralását tervezi, általában összehasonlít néhány ajánlatot. Mindezt megteheti egy vasútállomáson, egy utazási irodában, vagy otthon az internetet böngészve. A vevők ilyenkor nem vizsgálnak meg több ezer ajánlatot, hanem csak maximum tízet-húszat. Az életben számtalan (elfogadható és kevésbé

elfogadható) stratégiával találkozhatunk, amelyekkel a cégek, szállodák és légitársaságok megpróbálják a termékeiket a legjobb ajánlatok közé pozicionálni. Egy gyakori példa erre, hogy néhány légitársaság hihetetlenül rövid utazási idővel teszi közzé az ajánlatát. Ennek az az egyetlen célja, hogy azokban a szoftverekben, amelyek az A pontból a B pontba történő utazásokat menetidő szerint csökkenő sorrendben listázzák, az ajánlat a legelsők között szerepeljen. Sok esetben nem is egyszerű a vevő számára, hogy az ilyen trükköket észrevegye, amelyeknek a célja a kivonatoló eljárásokban való minél sikeresebbnek tűnő mutatkozás.

7. RNS molekulák másodlagos térszerkezetének meghatározása. Az RNS molekulák másodlagos térszerkezetének meghatározása az egyik központi téma a számítógépes biológia területén. Az erre vonatkozó legjelentősebb algoritmusok a dinamikus programozáson alapulnak. Léteznek on-line adatbázisok, ahonnan valós időben lehetséges megoldások kérhetők le.

Gyakorlatok

14.1-1. Szerezzünk gyakorlatot a több választási lehetőséget kínáló rendszerekben a FreeCell nevű türelemjáték segítségével. Töltsük le a BigBlackCell (BBC) nevű segédprogramot a http://www.minet.uni-jena.de/~BigBlackCell/ címről és ismerkedjünk meg a programmal. Némi gyakorlás után egy átlagos felhasználónak a BBC segítségével óránként legalább 60 FreeCell előfordulást kell megoldania.

14.2. Több lehetséges megoldás előállítása

14.2.1. Lehetséges megoldások előállítása heurisztikák és ismételt heurisztikák segítségével

Nagyon sok optimalizálási probléma valóban nehéznek mondható, ilyenek például az NP-teljes problémák. A pontos, de lassú eljárások, illetve a megbízhatatlan, de gyors heurisztikák két lehetséges megközelítését adják annak, ahogyan pontos vagy közelítő megoldásokat találhatunk. Ha az a feladatunk, hogy néhány alternatív megoldást hozzunk létre, akkor a szükségből erényt kovácsolhatunk. Általában sokkal több jónak mondható megoldás van, mint ahány tökéletes, és a különböző heurisztikák – főleg a véletlen elemeket is tartalmazók – nem mindig ugyanazt a megoldást szolgáltatják.

Ezért egy egyszerű stratégia lehet az, hogy egy vagy több heurisztikát többször alkalmazunk ugyanarra a problémára, és a kapott megoldásokat feljegyezzük. Létrehozhatunk pontosan annyi megoldást, amennyire szükségünk van, de létrehozhatunk többet is, amit aztán majd egy megfelelő kivonatoló módszerrel javítunk. A kivonatok készítésénél alapvető szempont a minőség, és a megoldások megfelelő szóródása. Ami a szóródást illeti, ehhez célszerű a lehetséges megoldások között valamilyen távolsági mértéket bevezetni, valamint megfelelő klaszterező algoritmusokat használni.

Egyetlen heurisztika ismétlődő futtatása

A legtöbb esetben a heurisztika tartalmaz valamilyen mértékű véletlent. Ez esetben nincs más teendőnk, mint, hogy a heurisztikát egymástól függetlenül többször lefuttassuk, amíg kellő számú különböző megoldást kapunk. Az alábbiakban az utazóügynök problémán fogjuk bemutatni ezt a megközelítést. Adunk egy példát a cserélő heurisztikára és a beszúró

heurisztikára, és mindkét esetben rámutatunk a véletlen elemek szerepére.

Amennyiben a pontok közötti $d(i, j)$ távolság szimmetrikus, akkor a lokális keresés kettős cserével egy jól ismert cserélő heurisztika. Az alábbi pszeudokódban $T(p)$ jelöli a T vektor p-edik komponensét.

LOKÁLIS-KERESÉS-AZ-UTAZÓ-ÜGYNÖK-PROBLÉMÁRA(N, d)

1 generáljunk egy kezdeti véletlen útvonalat $T = (i_1, i_2, \ldots, i_N)$
2 **while** létezik olyan p és q index, amelyekre $1 \leq p < q \leq N$ és $q \geq p + 2$, és
$d(T(p), T(p+1)) + d(T(q), T(q+1)) > d(T(p), T(q)) + d(T(p+1), T(q+1))$
(A $q = N$ speciális esetben vegyük a $q + 1 = 1$-et.)
3 **do** $T \leftarrow (i_1, \ldots, i_p, i_q, i_{q-1}, \ldots, i_{p+1}, i_{q+1}, \ldots, i_N)$
4 számoljuk ki a T útvonal hosszát, l-et
5 **return** T, l

Ebben a heurisztikában a véletlen elemeket a kezdeti útvonal kiválasztása, valamint az a sorrend jelenti, ahogyan az élpárokat megvizsgáljuk a 2. lépésben. Különböző beállítások különböző lokális minimumhoz vezetnek. Nagy méretű problémák esetén, például 1000 véletlen pontot véve az egység oldalú négyzetben, az Euklideszi távolságot figyelembe véve, teljesen normálisnak tekinthető, ha 100 független futtatás majdnem 100 különböző lokális minimumhoz vezet.

Az alábbi pszeudokód egy szabványos beszúró heurisztikát mutat be.

BESZÚRÁS-AZ-UTAZÓ-ÜGYNÖK-PROBLÉMÁRA(N, d)

1 generáljuk a $(i_1, i_2, \ldots, i_N)$ véletlen permutációt az $\{1, 2, \ldots, N\}$ elemekből
2 $T \leftarrow (i_1, i_2)$
3 **for** $t \leftarrow 2$ **to** $N - 1$
4 **do** keressük meg a $d(T(r), i_{t+1}) + d(i_{t+1}, T(r+1)) - d(T(r), T(r+1))$ minimumát
ahol $r \in \{1, \ldots, t\}$. ($r = t$ esetén $r + 1 = 1$)
legyen a minimum $r = s$-ben
5 $T \leftarrow (T(1), \ldots, T(s), i_{t+1}, T(s+1), \ldots, T(t))$
6 számoljuk ki a T útvonal hosszát, l-et
7 **return** T, l

Így eljárva az elemeket egyesével szúrjuk be oly módon, hogy a beszúrás után a lehető legkisebb legyen az új útvonalhossz.

Itt a véletlen elem az N pont permutációja. Hasonlóan, mint a kettős cserénél, a különböző beállítások különböző lokális minimumhoz vezetnek. További véletlen elemet jelenthet, ha valamelyik lépésben az optimális beszúrás helye nem egyértelmű.

Néhány modern heurisztika a természettel való hasonlóságon alapul. Ilyen esetekben a felhasználónak még több lehetősége van. A ***szimulált hőkezelés*** esetén néhány köztes megoldást kaphatunk az egyes futtatásokból. Vagy egy ***genetikus algoritmus*** egyes futásaiból is kaphatunk néhány megoldást, amelyek akár különböző generációkat reprezentálhatnak, akár egy kiválasztott generáció többszörös megoldásait.

A cserélő heurisztikák ismételt futtatására egy speciális technikát jelent a lokális optimumok perturbálása. Először lefuttatjuk az eljárást egy lokális optimum megtalálására. Ezután ezen az optimumon véletlenszerű lokális változtatásokat végzünk. Az így kapott

megoldásból kiindulva újra elindítunk egy lokális keresést, ami egy második lokális optimumhoz vezet. Ezen ismét véletlenszerű változtatásokat végzünk, és így tovább. A véletlenszerű változtatások mértéke azt fogja befolyásolni, hogy a lokális optimumok sorozata mennyire lesz különböző egymástól.

Még a determinisztikus heurisztikák alkalmazása esetén is vannak esetek, amikor több lehetséges megoldást kaphatunk. A holtversenyes esetekben például a választástól függően más-más eredményre jutunk, vagy a számolás pontossága (a kerekítési szabályok) is okozóhat ilyesmit. A 14.2.6. pontban tárgyaljuk azokat a büntető módszereket, amelyekben a paramétereket mesterségesen megváltoztatjuk (pl. növeljük az élhosszakat) az ismétlődő futtatások során. Az úgynevezett tetszőleges futási idejű algoritmusokban, mint például a keresési fa iteratív mélyítése, a köztes megoldások használhatók fel alternatív jelöltekként.

A lehetséges megoldások összegyűjtése különböző heurisztikák alkalmazásával

Ha ugyanarra a problémára több heurisztika is ismeretes, akkor mindegyikük szolgáltathat egy vagy több megoldásjelöltet. A 14.1.1. pont 5. részében ismertetett 3-agy összeállítás egy jó példája a több választást kínáló rendszereknek, amelyek több program futását használják fel. Az ott említett két program független egymástól, és különböző gépeken is futnak. (A versenysakkot szigorú időkorlátok keretei között játsszák, ahol 3 perc jut egy lépésre. Ha a két programot egy gépen futtatnánk multitaszk üzemmódban, azzal számítási erőforrásokat veszítenénk, és ez Heinz szerint körülbelül 60-80 Élő-pontba kerülne.) A 3-agy konfigurációnál használt sakkprogramok normális, megvásárolható programok, nem olyanok, amiket speciálisan a több választást kínáló rendszer számára terveztek.

Minden program tartalmazhat hibákat. A független programokat használó, több választási lehetőséget kínáló rendszerek egy nyilvánvaló előnnyel rendelkeznek a katasztrofális hibák tekintetében. Ha két független programot futtatunk, amelyek mindegyikénél p a katasztrofális hiba bekövetkezésének a valószínűsége, akkor az együttes bekövetkezés valószínűsége p^2-re csökken. Egy ellenőrző szerepet betöltő ember általában észre fogja venni, amikor a megoldásjelöltek katasztrofális hibát tartalmaznak. Ezért az az eset, amikor az egyik megoldás normális, a másik pedig katasztrofális (ennek valószínűsége egyébként $2p(p-1)$) nem fog hibához vezetni. Egy további előnyt jelent még, hogy ilyenkor a programoknak nem kell valamiféle k-legjobb vagy k-választást megvalósító módszert tartalmazniuk. A gépek által kínált egybevágó javaslatokat lehet úgy tekinteni, mint annak a jelét, hogy az adott megoldás éppen megfelelő.

A független programokat használó, több választási lehetőséget kínáló rendszereknek azonban vannak gyenge pontjai is:

- Ha a programok között jelentős tudásbeli különbség van, akkor a döntést hozó személy nehezen fogja rászánni magát, hogy a gyengébb gép megoldását válassza.
- Több lépéses műveletek esetén a különböző programok javaslatai egymással inkompatibilisek lehetnek.
- Gyakran előfordul, hogy az operációs rendszertől és a futtatott programoktól függően, egy PC nem elég stabilan működik multitaszk üzemmódban.

És természetesen az sem mindig biztosított, hogy a programok valóban függetlenek egymástól. Az 1990-es évek végén például Németországban számos közúti útvonaltervező program volt kapható különböző nevekkel és interfészekkel. Valójában azonban mindegyik négy független program kernel és adatbázis valamelyikén alapult.

14.2.2. Büntető módszer egzakt algoritmusokkal

Valamivel jobban kézben tartott módot ad a szóba jövő megoldások megtalálására az úgynevezett ***büntető módszer***. Ennek a módszernek az alapötletét az útvonaltervező példán keresztül illusztráljuk. Induljunk ki egy R_1 optimális (vagy megfelelő) útvonalból és keressünk egy R_2 alternatív megoldást, amelyik a lehető legjobban kielégíti az alábbi két feltételt.

(i) R_2-nek megfelelőnek kell lennie a célfüggvény szempontjából. Ellenkező esetben nincs értelme, hogy R_2-t válasszuk. A példánknál maradva, most az útvonal hossza az elsődleges cél.

(ii) R_2-nek nem szabad nagyon hasonlítania az eredeti megoldásra. Ellenkező esetben nem beszélhetünk *valódi* alternatíváról. Az úgynevezett *mikro mutációk* esetén nagy az esélye annak, hogy az összes egymáshoz hasonló megoldásjelölt ugyanazzal a gyenge ponttal rendelkezik. A példánkban a „hasonlóság mérésére" alkalmas lehet az R_1-ben és R_2-ben is megtalálható közös részek hossza.

Ez azt jelenti, hogy R_2-nek rövidnek kell lennie, de emellett R_1-gyel kevés közös részének kell lennie. E cél elérése érdekében két célfüggvény kombinációját fogjuk használni. Az egyik az útvonal hossza, a másik a közös részek hossza. Ezt úgy fogjuk elérni, hogy az R_1-beli szakaszokat büntetni fogjuk, és ennek a módosított legrövidebb útvonal problémának keressük az R_2 megoldását. A büntetés mértékének változtatásával különbözőképpen súlyozhatjuk az (i) és (ii) feltételeket.

Az egyik legegyszerűbb megközelítés az, amikor egy ***relatív büntető tényezőt*** használunk. Ez azt jelenti, hogy az R_1-hez tartozó szakaszok hosszát megszorozzuk $1+\varepsilon$-nal.

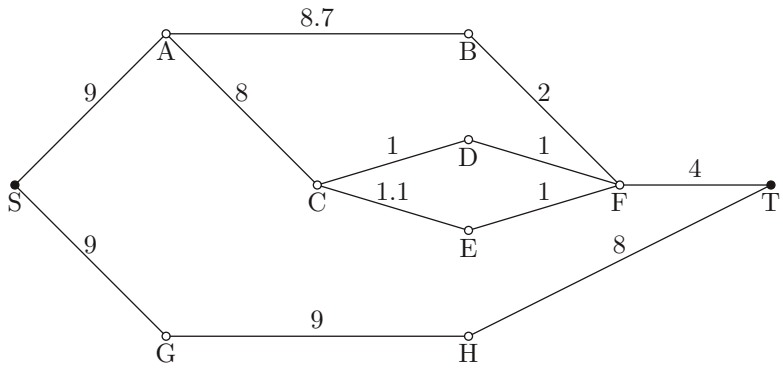
BÜNTETŐ-MÓDSZER-RELATÍV-BÜNTETŐ-TÉNYEZŐKKEL(G, s, t, ε)

```
1  keressük meg az s-ből t-be vezető R_1 legrövidebb utat a G = (V, E, w) súlyozott gráfban
2  for ∀e ∈ E
3      do if e R_1-hez tartozik
4              then ŵ(e) ← w(e) · (1 + ε)
5              else ŵ(e) ← w(e)
6  keressük meg az s-ből t-be vezető R_2 legrövidebb utat a Ĝ = (V, E, ŵ) módosított gráfban
7  számítsuk ki az R_2 módosítás nélküli w(R_2) hosszát
8  return (R_1, R_2) és (w(R_1), w(R_2))
```

Tekintsük az alábbi példát.

14.1. példa. Adott egy $G=(V,E)$ gráf súlyozott élhosszakkal. A 14.2. ábrán az élek hosszát a melléjük írt számok jelzik. Az S-ből T-be vezető legrövidebb út P_D, amelynek hossza 23, és a következő csúcsokat érinti: S - A - C - D - F - T. Ha a P_D éleinek hosszát megszorozzuk 1.1-del, és megoldjuk a kapott legrövidebb út problémát, akkor a P_B útvonalat kapjuk, amelynek módosított hossza 25, eredeti hossza 23.7, és a következő csúcsokat érinti: S - A - B - F - T. P_B és P_D közös részei az S - A és F - T szakaszok, amelyeknek összhossza 13.

ε méretét mindig a körülményeknek megfelelően kell megválasztani. Az AND cég piacon kapható útvonaltervező programjában a legrövidebb útvonal minden szakaszát 1.2-vel szorozták meg, vagyis $\varepsilon = 0.2$. Az alternatív útvonal ezek alapján kerül kiszámításra. Berger munkájában a műholdas felvételeken szereplő lineáris struktúrák (utcák, folyók, reptéri



14.2. ábra. A 14.1., 14.2. és 14.6. példákhoz tartozó gráf.

kifutópályák) felismerése szintén a legrövidebb útvonal módszerrel történik. Itt $\varepsilon = 1.0$ bizonyult megfelelő választásnak, ami érdekes alternatív megoldásokat adott.

A relatív büntető tényező helyett használhatunk ***additív büntető tagot*** is. Ez azt jelenti, hogy minden olyan élhez, amelyet büntetni szeretnénk, hozzáadunk egy konstans ε-t. A fenti algoritmusban ekkor csupán a 4. lépést kell megváltoztatnunk az alábbira.

4^* **then** $\widehat{w}(e) \leftarrow w(e) + \varepsilon$

14.2. példa. Adott a 14.1. példából már ismert $G = (V, E)$ gráf (lásd a 14.2. ábrán). Az S-ből T-be vezető legrövidebb út most is P_D, amelynek hossza 23, és amely a következő csúcsokat érinti: S - A - C - D - F - T. Ha a P_D éleihez hozzáadunk 0.1-et és megoldjuk a kapott legrövidebb út problémát, akkor a P_E útvonalat kapjuk, amelynek módosított hossza 23.4, eredeti hossza 23.1, és a következő csúcsokat érinti: S - A - C - E - F - T. P_D-nek és P_E-nek három közös éle van.

Alapjában véve az additív büntető tag nem rosszabb a relatívnál. Az utóbbinak, a multiplikatívnak azonban megvan az az előnye, hogy nem érzékeny az élek mesterséges kettévágására.

A büntető módszerek általánosításához hasznos a következő definíció.

14.1. definíció (összeg típusú optimalizálási probléma). *Legyen E egy tetszőleges véges halmaz, S pedig egy E részhalmazaiból álló halmaz. E-t* ***alaphalmaznak*** *nevezzük, S elemeit pedig E* ***megengedett részhalmazainak.*** *Legyen $w : E \to \mathbb{R}$ egy valós értékű súlyfüggvény az E-n. Minden $B \in S$-re legyen $w(B) = \sum_{e \in B} w(e)$.*

A $\min_{B \in S} w(B)$ optimalizálási problémát ***összeg típusú optimalizálási problémának,*** *vagy röviden csak* ***$\sum$-típusú problémának*** *nevezzük.*

Megjegyzések.

1. A $B \in S$ elemeket szokás ***megengedett megoldásoknak*** is nevezni.
2. Minden maximalizálási probléma átalakítható minimalizálási problémává ha w-t $-w$-vel helyettesítjük. Ezért a maximalizálási problémákat is $\sum$-típusú problémának fogjuk nevezni.

14.2.3. Példák $\sum$-típusú problémákra

- Legrövidebb út probléma
- Hozzárendelési probléma
- Utazóügynök probléma
- Hátizsák probléma
- Sorozat csoportosítási probléma

14.3. példa. Tekintsük a hátizsák problémát. Adott egy $I = \{I_1, I_2, \ldots, I_n\}$ elemhalmaz, egy $w : I \to \mathbb{R}^+$ súlyfüggvény, egy $v : I \to \mathbb{R}^+$ értékfüggvény és a hátizsák kapacitása C. A feladat az, hogy határozzuk meg azt a legértékesebb elemhalmazt, amelyek összsúlya nem haladja meg a hátizsák kapacitását.

Ha I-t tekintjük alaphalmaznak, S pedig azon részhalmazok összessége, amelyek összsúlya kisebb vagy egyenlő, mint C, akkor egy $\sum$-típusú problémához jutunk. Maximalizálnunk kell $v(B)$-t $B \in S$-re.

14.2.4. A büntető módszer absztrakt megfogalmazása $\sum$-típusú problémákra

14.2. definíció (büntető módszer). *Legyen E egy tetszőleges halmaz, S pedig álljon E megengedett részhalmazaiból. Legyen $w : E \to \mathbb{R}$ egy valós értékű, $p : E \to \mathbb{R}^{\geq 0}$ pedig egy nem negatív valós értékű függvény az E-n.*

Minden $\varepsilon > 0$-ra legyen B_ε egy optimális megoldása a

$$\min_{B \in S} f_\varepsilon(B),$$

problémának, ahol

$$f_\varepsilon(B) := w(B) + \varepsilon \cdot p(B) \ .$$

*Egy olyan algoritmussal, amelyik képes a büntetés nélküli $\min_{B \in S} w(B)$ problémát megoldani, megtalálhatjuk B_ε megoldásait is. Ehhez csak a w függvényt kell módosítanunk, oly módon, hogy minden $e \in E$-re $w(e)$-t helyettesítjük $w(e) + \varepsilon \cdot p(e)$ -vel. B_ε-t **ε-büntetés melletti megoldásnak** vagy **ε-alternatívának** nevezzük. .*

Definiáljuk ezen kívül B_∞-t, mint a következő probléma megoldását:

$$\operatorname{lex}\min_{B \in S} (p(B), w(B)) \quad \textit{(minimalizálás a lexikografikus sorrendnek megfelelően)},$$

amely a minimális $p(B)$ értékkel rendelkezik, és az ilyen megoldások között minimális $w(B)$ értékkel.

Megjegyzés. Amennyiben w és p is pozitív, valós értékű függvények, ekkor egyfajta szimmetria áll fenn az optimális megoldások körében: B^* pontosan akkor lesz ε-büntetés melletti megoldás $(0 < \varepsilon < \infty)$ a (w, p) függvénypárra nézve, ha B^* $(1/\varepsilon)$ büntetés melletti megoldás a (p, w) függvénypárra nézve.

A szimmetria megőrzése miatt van értelme definiálni B_0-t, ami optimális megoldása a következő problémának:

$$\operatorname{lex}\min_{B\in S}(w(B), p(B)) \ .$$

Ez azt jelenti, hogy B_0 nem csak optimális megoldás a w célfüggvényre nézve, hanem az ilyen megoldások között a minimális p értékkel rendelkezik.

14.4. példa. Adjuk meg a formális definícióját a 14.1. példának ebben az absztrakt $\sum$-típusú megfogalmazásban. Ismerjük az S-ből T-be vezető P_D legrövidebb utat, és keresünk egy alternatív, jó megoldást. A p büntető függvényt a következőképpen definiáljuk:

$$p(e) = \begin{cases} w(e), & \text{ha } e \text{ egyik éle a } P_D \text{ legrövidebb útnak} \ , \\ 0 & \text{egyébként} \ . \end{cases}$$

Büntetés melletti megoldások keresése az összes $\varepsilon \geq 0$ paraméterre

Gyakran előre nem látható, hogy mely ε paraméter mellett kapunk használható alternatív megoldásokat. Egy „oszd meg és uralkodj” jellegű algoritmussal megtalálhatjuk az *összes* olyan megoldást, amelyik *valamely* ε-ra előállna.

Véges S halmazokra megadunk egy hatékony algoritmust, amelyik egy viszonylag kicsi $\mathcal{B} \subseteq S$ megoldásokból álló, a következő tulajdonságokkal rendelkező halmazt állít elő:

- minden $B \in \mathcal{B}$ elemre létezik olyan $\varepsilon \in \mathbb{R}^+ \cup \{\infty\}$, hogy B optimális megoldás az ε büntető paraméter mellett;
- minden $\varepsilon \in \mathbb{R}^+ \cup \{\infty\}$-re létezik olyan $B \in \mathcal{B}$ elem, hogy B optimális megoldás az ε büntető paraméter mellett;
- $\mathcal{B}$ a fenti két tulajdonsággal rendelkező összes halmazrendszer közül a minimális elemszámmal rendelkezik.

Egy olyan B megoldást, amelyik legalább egy büntető paraméter mellett optimális, ***büntetés-optimálisnak*** nevezünk. A következő algoritmus büntetés-optimális megoldásoknak egy olyan halmazát keresi meg, amelyek minden $\varepsilon \in \mathbb{R}^+ \cup \{\infty\}$-t lefednek.

Az egyszerűbb azonosíthatóság kedvéért a $\mathcal{B}$ halmaz elemeit rögzített sorrendben adjuk meg $(B_{\varepsilon(1)}, B_{\varepsilon(2)}, \ldots, B_{\varepsilon(k)})$, ahol $0 = \varepsilon(1) < \varepsilon(2) < \cdots < \varepsilon(k) = \infty$. Az algoritmusnak ellenőriznie kell, hogy $\varepsilon(i) < \varepsilon(i+1)$ esetén ne létezzen olyan köztes ε, $\varepsilon(i) < \varepsilon < \varepsilon(i+1)$, hogy erre a büntető paraméterre sem $B_{\varepsilon(i)}$ sem $B_{\varepsilon(i+1)}$ nem optimális. Ellenkező esetben az algoritmusnak azonosítania kell legalább egy ilyen ε-t, és keresnie kell egy ε-büntetés melletti B_ε megoldást. Az alábbi pszeudokód 11. lépésében a $Border[i]$ változót akkor állítjuk 1-re, ha kiderül, hogy nem létezik ilyen köztes ε.

Az alábbiakban látható a pszeudokód, amelyhez néhány megjegyzést is fűztünk.

Algoritmus büntetés-optimális megoldások olyan $\mathcal{B}$ sorozatának megtalálására, amelyek minden $\varepsilon \geq 0$-ra lefedik a következő problémát:

$$\min_{B\in S} f_\varepsilon(B)$$

ahol $f_\varepsilon(B) = w(B) + \varepsilon \cdot p(B)$.

OSZD-MEG-ÉS-FEDD-LE(w, p)

```
1  számítsuk ki azt a B_0-at, amelyik minimalizálja w(B)-t és p(B)-értéke a lehető legkisebb
2  számítsuk ki azt a B_∞-t, amelyik minimalizálja p(B)-t és w(B)-értéke a lehető legkisebb
3  if (p(B_0) = p(B_∞))
4     then do B ← {B_0}
5             E ← (0)
6             Border ← ∅
                       ▷ B_0 minimalizálja a w és p függvényeket és minden ε-ra optimális
7     else do k ← 2
8             E = (ε(1), ε(2)) ← (0, ∞)
9             Border[1] ← 0
10            B ← (B_0, B_∞)
11            while van olyan i ∈ {1, 2, ..., k − 1} hogy Border[i] = 0.
12                  do ε̄ ← (w(B_ε(i+1)) − w(B_ε(i)))/(p(B_ε(i)) − p(B_ε(i+1)))
13                     keressünk egy optimális B_ε̄ megoldást a ε̄ paraméterhez
14                     if f_ε̄(B_ε̄) = f_ε̄(B_ε(i)) = f_ε̄(B_ε(i+1))
15                        then Border[i] ← 1
16                        else do B ← (B_ε(1), ..., B_ε(i), B_ε̄, B_ε(i+1), ..., B_ε(k))
17                                E ← (ε(1), ..., ε(i), ε̄, ε(i + 1), ..., ε(k))
18                                Border ← (Border[1], ..., Border[i], 0, Border[i + 1], ...,
                                            Border[k − 1]
19                                k ← k + 1
20 return B, E, Border
```

Az algoritmus végén $\mathcal{B}$ különböző büntetés-optimális megoldások sorozata lesz, az $\mathcal{E}$ vektor pedig egymás utáni epszilonokat fog tartalmazni.

A fenti algoritmus a következő tulajdonságokon alapul:

(1) Ha B egy ε-optimális megoldás, akkor létezik olyan $I_B = [\varepsilon_l, \varepsilon_h]$ intervallum ($\varepsilon_l, \varepsilon_h \in \mathbb{R} \cup \{\infty\}$), hogy B optimális minden $\varepsilon \in I_B$ paraméterre, más paraméterre viszont nem optimális.

(2) Két különböző B és B' megoldásra, és a hozzájuk tartozó nem üres I_B és $I_{B'}$ intervallumokra csak a következő három eset valamelyike fordulhat elő.

- $I_B = I_{B'}$. Ez pontosan akkor igaz ha $w(B) = w(B')$ és $p(B) = p(B')$.
- I_B és $I_{B'}$ diszjunktak.
- $I_B \cap I_{B'} = \{\overline{\varepsilon}\}$, vagyis a metszet egyetlen epszilont tartalmaz. Ez az eset akkor áll fenn, ha I_B és $I_{B'}$ szomszédos intervallumok.

Az E halmaz végessége miatt csak véges sok $B \in S$ megengedett megoldás létezik. Ezért csak véges sok optimalitási intervallum lehet. Így (1)-ből és (2)-ből következik, hogy a $[0, \infty]$ intervallumot fel tudjuk osztani intervallumoknak a következő halmazára: $\{[0 = \varepsilon_1, \varepsilon_2], [\varepsilon_2, \varepsilon_3], \ldots, [\varepsilon_k, \varepsilon_{k+1} = \infty]\}$. Minden intervallumra vonatkozóan különböző B megoldásokat kapunk, amelyek optimálisak az intervallumbeli összes ε-ra. Az ilyen megoldást az ***intervallum reprezentánsának*** nevezzük.

(3) Az algoritmus célja, hogy ezeknek az optimalitási intervallumoknak a *határait* megtalálja, és minden intervallumra találjon egy reprezentáns megoldást. Az iteráció minden

lépésében vagy egy új intervallum reprezentánsát, vagy két intervallum között egy új határt találunk meg (7-13 lépések). Ha k darab optimalitási intervallumunk van, ahol $k \geq 2$, akkor elegendő $2k-1$ darab $\min_{B \in S} w(B) + \varepsilon \cdot p(B)$ típusú problémát megoldani, hogy valamennyit megvizsgáljuk, és megtaláljuk a reprezentáns megoldásokat.

Az ε-alternatívák unimodalitási tulajdonsága

Amennyiben csak egy ε-alternatívát számolunk ki, felmerül a kérdés, hogy milyen büntető paramétert használjunk, hogy a „lehető legjobb" alternatív megoldáshoz jussunk. Ha a büntető paraméter túl kicsi, az optimális és az alternatív megoldás túlságosan hasonló egymáshoz, és ez nem ad valódi választási lehetőséget. Ha a paraméter túl nagy, az alternatív megoldás túlságosan gyenge lesz. A legjobb választásnak az tűnik, ha „közepes" ε-t választunk. Ezt fogjuk illusztrálni a következő, útvonaltervező példában.

14.5. példa. Tegyük fel, hogy egy adott kezdő és végpont közötti útvonalat kell megterveznünk. Ismerjük az átlagos utazási időket minden szakaszra vonatkozóan, és *két* útvonalat tervezhetünk. Az utolsó pillanatban ismerjük meg a tényleges utazási időket, és ekkor választhatjuk ki a gyorsabbat a két jelöltünk közül.

Legyen az első útvonal az, amelyik az átlagos utazási idők alapján a leggyorsabb, a második pedig egy olyan, amit a büntető módszer szerint találtunk. A kérdés az, hogy milyen büntető paramétert használjunk, hogy a gyorsabb út tényleges utazási idejét minimalizálni tudjuk

Konkrétan, vegyünk véletlenszerűen generált példákat a legrövidebb útvonal problémára egy 25×25-ös méretű súlyozott, irányított, rácsos $G = (V, E, w)$ gráfban. Az élek súlyainak eloszlása legyen egyenletes a $[0, 1]$ intervallumban. Kiszámoljuk a bal alsó sarokból a jobb felsőbe vezető, minimális súlyú P_0 útvonalat. Ezután oly módon büntetjük a P_0 éleit, hogy megszorozzuk azokat $1 + \varepsilon$-nal, és kiszámolunk egy sor ε-büntetés melletti megoldást P_{ε_1}-et, P_{ε_2}-őt, ..., $P_{\varepsilon_{30}}$-et, $\varepsilon = 0.025,\ 0.050,\ \ldots,\ 0.750$-re. Így 30 megoldás párt kapunk, $\{S_0, S_{\varepsilon_1}\}, \{S_0, S_{\varepsilon_2}\}, \ldots, \{S_0, S_{\varepsilon_{30}}\}$-at, ezeket tudjuk összehasonlítani.

Az élek $w(e)$ súlya a *késedelem nélküli, átlagos utazási időt jelöli*, vagyis azt a minimális időt, amire forgalmi dugó nélkül az adott útszakaszon szükség van. Az élre vonatkozó $\hat{w}(e)$ *tényleges* utazási idő ettől a következőképpen térhet el:

$$\hat{w}(e) = \begin{cases} \lambda_c(e) \cdot w(e): & p \text{ valószínűséggel}, \\ w(e): & 1-p \text{ valószínűséggel} \end{cases},$$

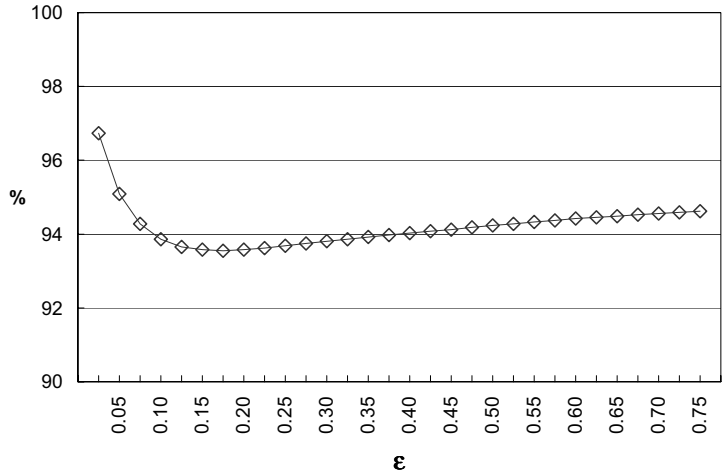
egymástól függetlenül minden élre. Itt a $\lambda_c(e)$ számok egymástól független véletlen számok, amelyek egyenletesen oszlanak el az $[1, c]$ intervallumban. A $0 \leq p \leq 1$ paramétert ***hiba valószínűségnek***, a $c \geq 1$ paramétert pedig ***hiba szélességnek*** nevezzük.

Minden $\{S_0, S_{\varepsilon_i}\}$ párra kiszámoljuk a $\hat{w}(S_0)$ és $\hat{w}(S_{\varepsilon_i})$ függvények minimumát. Azért, hogy jobban érzékeljük annak előnyét, hogy két választási lehetőségünk van egy helyett, képezzük az előbbi értéknek az S_0 optimális megoldás értékével vett hányadosát.

$$\phi_{\varepsilon_i} = \frac{\min\{\hat{w}(S_0), \hat{w}(S_{\varepsilon_i})\}}{\hat{w}(S_0)} \quad (i = 1, \ldots, 30).$$

Kiszámoltuk a ϕ_{ε_i} értékeket 100,000 véletlenszerűen generált 25×25-ös rácsos gráfra, ahol a hiba valószínűség $p = 0.1$ volt, a hibaszélesség pedig $c = 8$. A 14.3. ábrán a $\overline{\phi}_{\varepsilon_i}$ átlagos értékeit láthatjuk $\varepsilon_1 = 0.025,\ \varepsilon_2 = 0.050,\ \ldots,\ \varepsilon_{30} = 0.750$-re.

Amint az a 14.3. ábrán is látható, a megoldás párok ϕ_ε várható minősége unimodális ε-ra nézve. Ez azt jelenti, hogy ϕ_ε először csökken, majd növekszik növekvő ε-ra. Ebben a példában $\varepsilon^* \approx 0.175$ az optimális büntető paraméter.



14.3. ábra. $\overline{\phi}_{\varepsilon_i}$ értékei $\varepsilon_1 = 0.025$, $\varepsilon_2 = 0.050$, ..., $\varepsilon_{30} = 0.750$-re 25×25-ös rácson.

További kísérletek azt is kimutatták, hogy az optimális paraméter ε^* a probléma méretének növekedésével csökken. (Például $\varepsilon^* \approx 0.6$ volt 5×5-ös rácsokra, $\varepsilon^* \approx 0.175$ 25×25-ös rácsokra, és $\varepsilon^* \approx 0.065$ 100×100-as rács gráfokra.)

A büntetéses megoldások monotonitási tulajdonságai

Függetlenül attól, hogy egyszerre az összes ε-büntetés melletti megoldást generáljuk-e, vagy csak egyetlen egyet, a következő strukturális tulajdonságok bizonyíthatók:

Az ε büntető tényező fokozatos növekedésével olyan B_ε megoldásokat kapunk, amelyekre

- a célfüggvény p büntető része monoton módon egyre jobban illeszkedik (a megoldás egyre kevesebb büntetett részt tartalmaz),
- az eredeti w célfüggvény monoton módon egyre rosszabbá válik, ami kompenzálja a büntető részben bekövetkező javulást.

A fenti állításokat a következő tétel mondja ki pontosan.

14.3. tétel. *Legyen $w : E \to \mathbb{R}$ egy valós értékű függvény, $p : E \to \mathbb{R}^+$ pedig egy pozitív valós értékű függvény az E-n. Legyen B_ε a 14.2. definíciónak megfelelően definiálva minden $\varepsilon \in \mathbb{R}^+$-ra. Ekkor a következő négy állítás igaz.*

1. *$p(B_\varepsilon)$ gyengén monoton csökkenő ε-ra nézve.*
2. *$w(B_\varepsilon)$ gyengén monoton növekvő ε-ra nézve.*
3. *A $w(B_\varepsilon) - p(B_\varepsilon)$ különbség gyengén monoton növekvő ε-ra nézve.*
4. *$w(B_\varepsilon) + \varepsilon \cdot p(B_\varepsilon)$ gyengén monoton növekvő ε-ra nézve.*

Bizonyítás. Legyen δ és ε két tetszőleges nem negatív valós szám, amelyekre $0 \le \delta < \varepsilon$.

B_δ és B_ε definíciójából adódóan a következő egyenlőtlenségek teljesülnek.

1. $\varepsilon < \infty$ esetén

$$w(B_\varepsilon) + \varepsilon \cdot p(B_\varepsilon) \le w(B_\delta) + \varepsilon \cdot p(B_\delta) \,, \tag{14.1}$$

$$w(B_\varepsilon) + \delta \cdot p(B_\varepsilon) \ge w(B_\delta) + \delta \cdot p(B_\delta) \,. \tag{14.2}$$

(14.2)-t kivonva (14.1)-ből a következőt kapjuk:

$$(\varepsilon - \delta) \cdot p(B_\varepsilon) \le (\varepsilon - \delta) \cdot p(B_\delta) \qquad | : (\varepsilon - \delta) > 0$$

$$\Leftrightarrow \quad p(B_\varepsilon) \le p(B_\delta) \,. \tag{14.3}$$

$\varepsilon = \infty$ esetén (14.3)) egyenlőtlenség közvetlenül következik a B_∞ definíciójából.

2. Vonjuk ki (14.3)-at (14.2)-ből megszorozva δ-val, ekkor a következőt kapjuk:

$$w(B_\varepsilon) \ge w(B_\delta) \,. \tag{14.4}$$

3. Vonjuk ki (14.3)-at (14.4)-ből, ekkor a következőt kapjuk:

$$w(B_\varepsilon) - p(B_\varepsilon) \ge w(B_\delta) - p(B_\delta) \,.$$

4. (14.2)-ből az $\varepsilon > \delta \ge 0$ egyenlőtlenség felhasználásával a

$$w(B_\delta) + \delta \cdot p(B_\delta) \le w(B_\varepsilon) + \delta \cdot p(B_\varepsilon) \le w(B_\varepsilon) + \varepsilon \cdot p(B_\varepsilon)$$

$$\Rightarrow \quad w(B_\varepsilon) + \varepsilon \cdot p(B_\varepsilon) \ge w(B_\delta) + \delta \cdot p(B_\delta) \tag{14.5}$$

egyenlőtlenséget kapjuk. ■

Több alternatív megoldás létrehozása ugyanarra az ε büntető paraméterre

Ha adva van egy S_0 megoldás és további alternatív megoldásokra van szükségünk, akkor alkalmazhatjuk a büntető módszert többször egymás után, különböző $\varepsilon_1 < \cdots < \varepsilon_m$ paraméterekkel büntetve az S_0-t. Az így kapott megoldások rendre $S_{\varepsilon_1}, S_{\varepsilon_2}, \ldots, S_{\varepsilon_m}$. Ennek a módszernek az a nagy hátránya, hogy csak az eredeti S_0 megoldásnak és az egyes alternatív megoldásoknak a közös részére van hatása az ε_i értékeknek, de két különböző alternatív megoldás közös részére nincsen hatása. Ezért az S_{ε_i} és S_{ε_j} nagyon hasonló is lehet különböző i-re és j-re ($i \ne j$).

Ezt elkerülhetjük, ha a büntető módszert *iteratívan* alkalmazzuk ugyanarra az ε-ra.

ITERATÍV-BÜNTETŐ-MÓDSZER(w, p, k, ε)

1 oldjuk meg az eredeti $\min w(B)$ problémát és keressük meg az optimális S_0 megoldást
2 definiáljuk a $p_1(B) \leftarrow \varepsilon \cdot w(B \cap S_0)$ büntető függvényt
3 oldjuk meg a módosított $\min w(B) + \varepsilon \cdot p_1(B)$ problémát és keressük meg az S_1 megoldást
4 **for** $j \leftarrow 2$ **to** k
5 **do** $p_j(B) \leftarrow \varepsilon \cdot w(B \cap S_0) + \varepsilon \cdot w(B \cap S_1) + \cdots + \varepsilon \cdot w(B \cap S_{j-1})$
6 oldjuk meg a módosított $\min w(B) + \varepsilon \cdot p_j(B)$ problémát és
keressük meg az S_j megoldást
7 **return** $(S_0, S_1, \ldots, S_k)$

Az 5. lépést a következő változattal is helyettesíthetjük:

5* **do** $p_j(B) \leftarrow \varepsilon \cdot w(B \cap (S_0 \cup S_1 \cup \cdots \cup S_{j-1}))$

Az első esetben (5) a j számú $S_0, S_1, \ldots S_{j-1}$ megoldás közül r-hez tartozó megoldásrészt $r{\cdot}\varepsilon$ tényezővel bünteti. A második esetben (5*) az $S_0, S_1, \ldots$ or S_{j-1} megoldások közül legalább egyhez tartozó megoldásrészt egyszeres multiplicitással bünteti. A teljesítménybeli különbség a két eset között jelentős lehet. A legrövidebb útvonal problémára azonban három (S_0, S_1 és S_2) megoldás esetén az (5) változat valamivel jobb eredményt adott.

14.6. példa. Vegyük ismét a 14.2. ábrán látható $G = (V, E)$ gráfot. Az $\varepsilon = 0.1$ büntető paraméterre vonatkozóan keressünk három megoldást. Az S-ből T-be vezető legrövidebb út P_D, amelynek hossza 23, és a következő csúcsokat érinti: S - A - C - D - F -T. Ha a P_D éleinek hosszát megszorozzuk 1.1-del, és megoldjuk a kapott legrövidebb út problémát, akkor a P_B útvonalat kapjuk, amely a következő csúcsokon megy keresztül: S - A - B - F - T.

Ha az (5) lépésben megadott módszert követjük, akkor az (A, C), (C, D), (D, F), (A, B) és (B, F) élek hosszait kell 1.1 büntető tényezővel megszoroznunk. Az (S, A) és (F, T) éleket 1.2-vel kell megszoroznunk (dupla büntetés). Az ily módon kapott optimális megoldás P_H lesz, ami az S - G - H - T csúcsokon megy keresztül.

14.2.5. Lineáris programozás – büntető módszer

Jól ismert tény, hogy a legrövidebb útvonal probléma, hasonlóan sok más áramlási problémához, lineáris programozással is megoldható. A lineáris programozás segítségével alternatív megoldások is létrehozhatók. Először az eredeti legrövidebb útvonal problémára mutatjuk be a lineáris programozást.

A legrövidebb útvonal probléma lineáris programként megfogalmazva

Vegyünk egy $G = (V, E)$ irányított gráfot, és egy $w : E \to \mathbb{R}^+$ függvényt, amelyik a gráf minden éléhez egy hosszúságot rendel. Legyen s és t a gráf két megkülönböztetett pontja.

Melyik a legrövidebb egyszerű útvonal s-ből t-be?

Minden $e = (i, j) \in E$ élre bevezetünk egy x_e változót. x_e-nek 1 értéket kell kapnia ha az e él része a legrövidebb útvonalnak, egyébként pedig 0-t. Jelöljük $S(i) = \{j \in V : (i, j) \in E\} \subseteq V$-vel az i csúcsra rákövetkező csúcsok halmazát, $P(i) = \{j \in V : (j, i) \in E\} \subseteq V$-vel pedig az i csúcsot megelőző csúcsok halmazát. Az $LP_{\text{legrövidebb-út}}$ lineáris program a következőképpen formalizálható:

$$\min \sum_{e \in E} w(e) \cdot x_e$$

feltéve, hogy

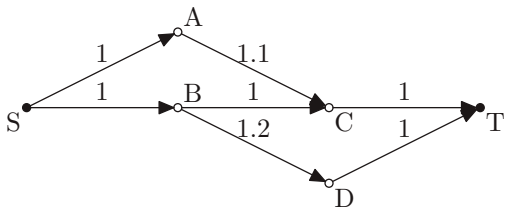
$$\sum_{j \in S(s)} x_{(s,j)} - \sum_{j \in P(s)} x_{(j,s)} = 1 \qquad \text{kimenő feltétel az } s \text{ kezdőpontra vonatkozóan,}$$

$$\sum_{j \in S(t)} x_{(t,j)} - \sum_{j \in P(t)} x_{(j,t)} = -1 \qquad \text{bemenő feltétel a } t \text{ végpontra vonatkozóan,}$$

$$\sum_{j \in S(i)} x_{(i,j)} - \sum_{j \in P(i)} x_{(j,i)} = 0 \qquad \text{minden további } i \in V \backslash \{s, t\} \text{pontra}$$

Kirchhoff-feltételek a belső pontokra

$0 \le x_e \le 1$ minden $e \in E$-re .



14.4. ábra. Példa gráf az LP-büntető módszerhez.

A kezdő és végpontra vonatkozó feltételek miatt s egy forrás, t pedig egy nyelő. A Kirchhoff-feltételek miatt nincs több forrás, sem pedig nyelő. Ezért kell, hogy legyen egy s-ből t-be vezető „kapcsolat”.

Nem nyilvánvaló, hogy ez a kapcsolat egy egyszerű út. Az x_e változóknak lehetne nem egész értéke is, vagy körök is előfordulhatnának bárhol. Van azonban egy alapvető áramlástani tétel, amelyik azt mondja ki, hogy az $LP_{legrvidebb-t}$ lineáris programnak van olyan optimális megoldása, amelyre minden $x_e > 0$ értéke egyenlő 1-el. Az $x_e = 1$-nek megfelelő élek egy egyszerű útvonalat adnak s-ből t-be.

14.7. példa. Vegyük a 14.4 ábrán látható gráfot. A legrövidebb útvonal problémához tartozó lineáris programozási feladat most hat egyenlőség feltételt tartalmaz (minden csomópontra egyet), és hét egyenlőtlenség párt (minden élre egy párt).

$$\begin{aligned}
&\min(x_{SA} + x_{SB} + x_{BC} + x_{CT} + x_{DT}) \cdot 1 + x_{AC} \cdot 1.1 + x_{BD} \cdot 1.2 \\
&\text{feltéve, hogy } x_{SA} + x_{SB} = 1, \\
&\qquad x_{CT} + x_{DT} = 1\ , \\
&\qquad x_{SA} - x_{AC} = 0, \\
&\qquad x_{SB} - x_{BC} - x_{BD} = 0\ , \\
&\qquad x_{AC} + x_{BC} - x_{CT} = 0, \\
&\qquad x_{BD} - x_{DT} = 0\ , \\
&\qquad 0 \le x_{SA}, x_{SB}, x_{AC}, x_{BC}, x_{BD}, x_{CT}, x_{DT} \le 1\ .
\end{aligned}$$

Az optimális megoldásra $x_{SB} = x_{BC} = x_{CT} = 1$.

Egy lineáris programozási feladat, amelyik két alternatív útvonalat ad meg s-ből t-be
Az alábbiakban megadjuk annak a feladatnak a lineáris programozásbeli reprezentációját, amelyik *két* alternatív útvonalat keres meg s-ből t-be.

Minden $e = (i, j) \in E$ élre bevezetünk két változót, x_e-t és y_e-t. Ha az e él mindkét útvonalnak része, akkor x_e és y_e is 1 értéket fog kapni. Ha az e él csak egy útvonalnak része, akkor x_e értéke 1 lesz, y_e értéke pedig 0. Egyébként mind x_e, mind y_e 0 értéket kap. $\varepsilon > 0$ egy büntető paraméter, amellyel a mindkét útvonalban szereplő éleket büntetjük.

A fentiek figyelembe vételével a következőképpen formalizálhatjuk az $LP_{2\text{-rövid-út}}$ lineáris programozási feladatot:

$$\min f(x,y) := \sum_{e\in E} w(e)\cdot x_e + (1+\varepsilon)\cdot w(e)\cdot y_e$$

feltéve, hogy
$$\sum_{j\in S(s)} x_{(s,j)} + y_{(s,j)} - \sum_{j\in P(s)} x_{(j,s)} + y_{(j,s)} = 2 \quad \text{feltétel az } s \text{ kezdőpontra vonatkozóan}$$
$$\sum_{j\in S(t)} x_{(t,j)} + y_{(t,j)} - \sum_{j\in P(t)} x_{(j,t)} + y_{(j,t)} = -2 \quad \text{feltétel a } t \text{ végpontra vonatkozóan}$$
$$\sum_{j\in S(i)} x_{(i,j)} + y_{(i,j)} - \sum_{j\in P(i)} x_{(j,i)} + y_{(j,i)} = 0 \quad \text{Kirchhoff-feltételek}$$
minden további pontra $i \in V\backslash\{s,t\}$

$$0 \le x_e, y_e \le 1 \text{ minden } e\in E\text{-re .}$$

14.8. példa. Tekintsük ismét a 14.4. ábrán szereplő gráfot. A két alternatív útvonal problémához tartozó lineáris programozási feladat most hat egyenlőség feltételt tartalmaz (minden csúcspontra egyet), és $2\cdot 7 = 14$ egyenlőtlenség párt.

$$\begin{aligned}
\min\quad & (x_{SA}+x_{SB}+x_{BC}+x_{CT}+x_{DT})\cdot 1 + x_{AC}\cdot 1.1 + x_{BD}\cdot 1.2 \\
& +[(y_{SA}+y_{SB}+y_{BC}+y_{CT}+y_{DT})\cdot 1 + y_{AC}\cdot 1.1 + y_{BD}\cdot 1.2]\cdot(1+\varepsilon) \\
\text{feltéve, hogy}\quad & x_{SA}+y_{SA}+x_{SB}+y_{SB} = 2, \\
& x_{CT}+y_{CT}+x_{DT}+y_{DT} = 2\ , \\
& x_{SA}+y_{SA}-x_{AC}-y_{AC} = 0, \\
& x_{SB}+y_{SB}-x_{BC}-y_{BC}-x_{BD}-y_{BD} = 0\ , \\
& x_{AC}+y_{AC}+x_{BC}+y_{BC}-x_{CT}-y_{CT} = 0, \\
& x_{BD}+y_{BD}-x_{DT}-y_{DT} = 0\ , \\
& 0 \le x_{SA}, x_{SB}, x_{AC}, x_{BC}, x_{BD}, x_{CT}, x_{DT}, y_{SA}, y_{SB}, y_{AC}, y_{BC}, y_{BD}, y_{CT}, y_{DT} \le 1\ .
\end{aligned}$$

A lineáris programozási feladatot úgy értelmezhetjük, mint egy minimális költségű áramlási problémát. De hol van vajon a kapcsolat a lineáris programozási feladat, és a között a probléma között, hogy keresnünk kell két útvonalat s-ből t-be?

14.4. tétel. *Ha az $LP_{2\text{-rövid-út}}$ lineáris programozási feladatnak van optimális megoldása, akkor van olyan (x,y) optimális megoldása is, amelyik a következő tulajdonságokkal rendelkezik.*

Léteznek olyan $E_1, E_2, E_3 \subseteq E$ diszjunkt halmazok, amelyekre

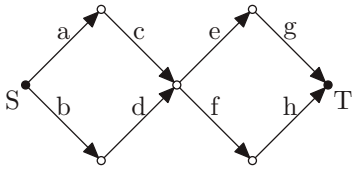
(i) $E_1 \cap E_2 = \emptyset$, $E_1 \cap E_3 = \emptyset$ *és* $E_2 \cap E_3 = \emptyset$,

(ii) $x_e = 1$, $y_e = 0$ *minden* $e \in E_1 \cup E_2$,

(iii) $x_e = 1$, $y_e = 1$ *minden* $e \in E_3$,

(iv) $x_e = 0$, $y_e = 0$ *minden* $e \notin E_1 \cup E_2 \cup E_3$.

(v) $E_1 \cup E_3$ *egy* P_1 *s-ből t-be vezető utat reprezentál,* $E_2 \cup E_3$ *egy* P_2 *s-ből t-be vezető utat ábrázol.* E_3 *pedig azon élek halmaza, amelyek mindkét útvonalban szerepelnek.*



14.5. ábra. Példa két útvonal nem egyértelmű dekompozíciójára.

(vi) *Nem létezik olyan* (Q_1, Q_2) *útvonal pár, amelyik jobb lenne* (P_1, P_2)*-nél, azaz*

$$w(P_1) + w(P_2) + \varepsilon \cdot w(P_1 \cap P_2) \leq w(Q_1) + w(Q_2) + \varepsilon \cdot w(Q_1 \cap Q_2),$$
$$\text{minden } (Q_1, Q_2) \text{ párra.}$$

Ez éppen azt jelenti, hogy a P_1 *és* P_2*-beli élhosszak összege plusz a kétszer használt élekre vonatkozó büntetés minimális.*

A fentiekhez még az alábbi megjegyzéseket fűzhetjük.

- Minden élhez két változó tartozik, x_e és y_e. Ezt értelmezhetjük úgy is, mint egy olyan utcát, amelyiken van egy ***normális sáv,*** és egy további ***extra sáv.*** Az extra sáv használata drágább, mint a normális sávé. Ha egy megoldás egy élt csak egyszer használ, akkor az olcsóbb, normális sávot használja. Ha azonban a megoldás kétszer használja az élt, akkor a normális sávot és az extra sávot is használja.

- Az (x, y) megoldásnak a kezdő csúcspontból a végső csúcspontba vezető útvonalának a felbontása a legtöbb esetben nem egyértelmű. A 14.5. ábrán S-ből T-be két útvonal párt is előállíthatunk, $(a-c-e-g,\ b-d-f-h)$-t és $(a-c-f-h,\ b-d-e-g)$-t. Mindkét pár egyformán optimális a 14.4. tétel értelmében. Így a felhasználónak kell választania közülük más, további szempontok alapján.

- A büntető módszer és az LP-büntető módszer általában különböző megoldásokhoz vezet. A büntető módszer kiszámolja az egyetlen legjobb megoldást, és egy megfelelő alternatívát. Az LP-büntető módszer két jónak mondható megoldást számol ki, amelyek között kicsi az átfedés. A 14.4. ábrán láthatjuk, hogy ez a pár nem feltétlenül tartalmazza a legjobb megoldást. Az ábrán az S -ből T-be vezető legrövidebb útvonal $P_1 = S$–B–C–T, amelynek hossza 3. Minden $\varepsilon > 0.1$-re az ε büntetés melletti megoldás $P_2 = S$–A–C–T. A (P_1, P_2) útvonal pár összhossza 6.1, a közös szakaszok hossza 1.0. $\varepsilon > 0.2$-re azonban az LP-büntető módszer a $(P_2, P_3) = (S$–A–C–T, S–B–D–$T)$ útvonalakat állítja elő, amelyek össz hossza 6.3, a közös szakaszaik hossza pedig 0.

- Lehetséges lenne k darab megoldásjelölt útvonal megkeresése is valamely $k > 2$-re, ha bevezetünk k darab $x_e^0, x_e^1, \ldots, x_e^{k-1}$ változót minden e élre, és beállítjuk s kínálatát és t

keresletét k-ra. Célfüggvényként használhatjuk például a következőt:

$$\min f(x^0,\dots,x^{k-1}) := \sum_{e\in E}\sum_{j=0}^{k-1}(1+j\cdot\varepsilon)\cdot w(e)\cdot x_e^j$$

vagy

$$\min f(x^0,\dots,x^{k-1}) := \sum_{e\in E}\sum_{j=0}^{k-1}(1+\varepsilon)^j\cdot w(e)\cdot x_e^j\ .$$

- Az LP büntető módszer nem csak a legrövidebb útvonal problémára működik. Általánosíthatjuk azt bármilyen, lineáris programozással megoldható problémára.
- Egy hasonló módszert, az *egész értékű lineáris programozásos büntető módszert* alkalmazhatunk egész értékű lineáris programozási feladatokra.

14.2.6. Büntető módszer heurisztikák alkalmazásával

A 14.2.2. pontban a büntető módszernek egzakt algoritmusokkal együtt való alkalmazását tárgyaltuk. Ilyen volt például a Dijkstra-algoritmus, vagy a dinamikus programozás a legrövidebb útvonal problémára. A büntető módszert azonban (egzakt megoldások helyett) heurisztikák esetén is alkalmazhatjuk több megoldás jelölt megkeresésére.

14.9. példa. Egy jól ismert heurisztika az utazóügynök problémára a lokális keresés kettős cserével (lásd a 14.2.1. pontot).

BÜNTETÉS-AZ-UTAZÓ-ÜGYNÖK-PROBLÉMÁRA-KETTŐS-CSERÉVEL

1 alkalmazzuk a kettős csere heurisztikát a büntetés nélküli problémára,
  az így kapott lokálisan optimális megoldás (ami nem feltétlenül globálisan optimális) legyen T
2 büntessük meg a T-hez tartozó éleket úgy, hogy megszorozzuk a hosszukat $(1+\varepsilon)$-nal
3 alkalmazzuk a kettős csere heurisztikát a büntetés melletti problémára,
  az így kapott alternatív megoldás legyen T_ε
4 számoljuk ki a T_ε módosítás nélküli hosszát
5 **return** (T, T_ε)

Kérdés: Milyen $\varepsilon \geq 0$ paramétert használjunk, hogy a leggyorsabb útvonal utazási idejét minimalizálni tudjuk?

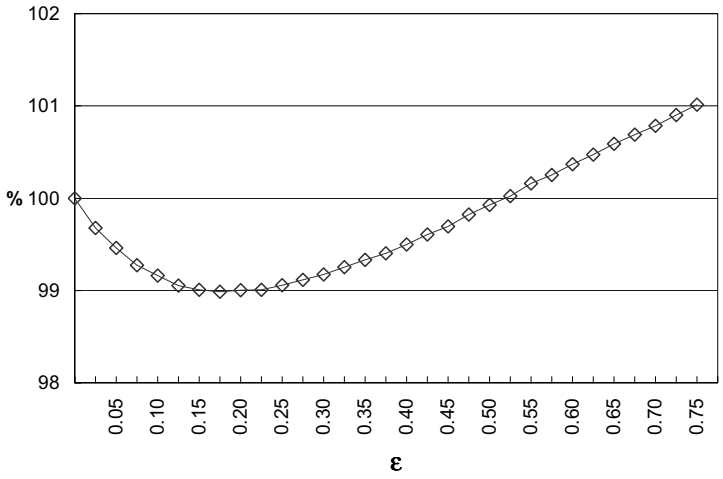
A 14.5. példában már ismertetetthez hasonló kísérletet végeztek el az utazóügynök problémára 25 véletlenül kiválasztott ponttal az egységnégyzetben. A 14.6. ábra az arányosított átlagokat mutatja az $\varepsilon_0 = 0.000$, $\varepsilon_1 = 0.025$, $\dots$, $\varepsilon_{30} = 0.750$ értékekre.

A megoldás párok ϕ_ε várható minősége (most is) unimodális az ε büntetési tényezőre nézve. Ez azt jelenti, hogy ϕ_ε először csökken, majd növekszik növekvő ε-ra. Ebben a példában $\varepsilon^* \approx 0.175$ az optimális büntető paraméter.

További kísérletek azt is kimutatták, hogy az ε^* optimális paraméter a probléma méretének növekedésével csökken.

Gyakorlatok

14.2-1. A következő, utazóügynök problémára vonatkozó programozási gyakorlat segít abban, hogy jobban átérezzük a lokális optimumok nagy változatosságát. Generáljunk vélet-



14.6. ábra. $\bar{\phi}_{\varepsilon_i}$ az $\varepsilon_0 = 0$, $\varepsilon_1 = 0.025$, ..., $\varepsilon_{30} = 0.750$ értékekre 25×25-ös rácson.

lenszerűen 200 pontot a 2-dimenziós egységnégyzetben. Számoljuk ki a távolságokat az Euklideszi metrika szerint. Futtassuk le százszor a lokális keresést kettős cserével, véletlenül választott kezdő útvonalból kiindulva. Számoljuk meg, hogy hány különböző lokális minimumot találtunk.

14.2-2. Adjuk meg ugyanazokat a kulcsszavakat különböző internetes keresőmotoroknak. Hasonlítsuk össze a találati listákat, és azok változatosságát.

14.2-3. Formalizáljuk az utazóügynök problémát egy $\sum$-típusú problémaként.

14.2-4. Bizonyítsuk be a 589. oldalon található megjegyzésben foglalt állítást.

14.2-5. Hogyan néz ki a $p(e)$ büntető függvény additív büntetés esetén, mint például a 14.2. példában?

14.2-6. Bizonyítsuk be a 591. oldalon levő (1) és (2) tulajdonságokat.

14.2-7. Alkalmazzuk az OSZD MEG ÉS FEDD LE algoritmust a 14.2 ábrán látható legrövidebb útvonal problémára S kezdőponttal és T végponttal. Legyen $w(e)$ az él hossza minden e-re, $p(e)$ pedig legyen azokra az élekre, amelyek hozzátartoznak P_D-hez (S - A - C - D - F - T útvonal) az él hossza, a többi élre pedig $p(e) = 0$. Vagyis egy útvonalra vonatkozó büntetés értéke egyenlő lesz azon szakaszainak hosszával, amelyek közösek P_D-vel.

14.2-8. Keressünk olyan $\varepsilon > 0$ büntető paramétert a 595. oldalon levő 14.6. példában, hogy $k = 3$-ra az ott leírt első módszer (a pszeudokód 5. sora) három különböző útvonalat hozzon létre, a második módszer (a pszeudokód 5* sora) viszont csak kettőt.

14.3. További interaktív problémamegoldó algoritmusok

Van néhány további terület, ahol egy embernek kell a számítógép által generált megoldásjelöltek közül választania. Ebben a szakaszban négy fontos esetet mutatunk be ezek közül, majd különböző vegyes témákkal zárjuk e fejezetet.

14.3.1. Tetszőleges futási idejű algoritmusok

Egy ilyen tetszőleges futási idejű algoritmusban a számítógép elkezd dolgozni egy problémán, és szinte az első pillanattól kezdve folyamatosan jelennek meg a megoldásjelöltek (mindig az addig talált legjobbak) a képernyőn. Természetesen egy ilyen folyamat során a kezdeti outputok gyakran csak előzetes, vagy közelítő megoldások, amelyeknek az optimalitása nem garantált, és ezek még messze vannak a tökéletes megoldástól.

Nézzünk egy példát. Az iteratív mélyítés többszörös, mélységében korlátozott keresést végez, és minden lépésben fokozatosan növeli a keresés mélységi korlátját. Tegyük fel, hogy a feladatunk jó megoldások keresése egy nagyméretű $T = (V, E)$ fában. Legyen $f : V \to \mathbb{R}$ a maximalizálandó függvény, V_d pedig a fa azon csúcspontjainak halmaza, amelyek d távolságra vannak a gyökértől.

FÁBAN-VALÓ-KERESÉS-ITERATÍV-MÉLYÍTÉSSEL(T, f)

```
1  Opt ← f(root)
2  d ← 1
3  while d < ∞
4      do határozzuk meg f Max_d maximumát V_d-n
5          if Max_d > Opt
6              then Opt ← Max_d
7          d ← d + 1
```

Minden időpillanatban az éppen aktuális legjobb megoldás (Opt) jelenik meg a monitoron. Az operátor bármelyik pillanatban megállíthatja a folyamatot.

Az iteratív mélyítés nem csak a számítógép és ember kölcsönhatásával foglalkozó terület számára érdekes, hanem van számos alkalmazása a teljesen automatizált számításokban is. Jó példa erre a különböző játékok fájában való keresés. A versenysakkban a programnak rögzített idő áll rendelkezésére 40 lépés megtételére. Itt az iteratív mélyítés kulcsfontosságú eszköz abban, hogy megtaláljuk az egyensúlyt az időfelhasználás és az alfa-béta keresés között.

Egy másik gyakori példa tetszőleges futási idejű algoritmusokra egy heurisztika ismételt alkalmazása. Legyen $f : A \to \mathbb{R}$ valamilyen bonyolult függvény, és keressük a nagy függvényértékkel rendelkező elemeket. Legyen H egy valószínűség alapú heurisztika, amely egy megoldásjelöltet ad meg erre az (A, f) maximalizálási problémára. H lehet például valamilyen lokális keresés, vagy más gradiens módszeren alapuló eljárás. H-t alkalmazhatjuk újra és újra, egymástól független menetekben, és mindig az eddig talált legjobb megoldást kínáljuk fel kimenetként.

A tetszőleges futási idejű algoritmusok harmadik alkalmazási területe a Monte Carlo szimulációk, például a Monte Carlo integráció. Egy statikus megközelítés előre meghatározott számú (pl. 1000) véletlen pont alapján működne, és ezek alapján adná meg az átlagot az outputban. Azonban már a menet közbeni átlag értékek (1, 2, 3 pont után, vagy minden 10-es, 50-es blokk után) adhatnának előrejelzést arra vonatkozóan, hogy melyik régióban várható a végső eredmény, illetve, hogy van-e értelme az összes lépés végrehajtásának. A varianciáknak és a kilógó értékek gyakoriságának a megjelenítése további információt szolgáltat arra vonatkozóan, hogy mikor a leginkább érdemes megállítani a Monte Carlo eljárás futását.

Az ember és számítógép együttműködését feltételező rendszerekben a tetszőleges futási idejű algoritmusok még egy további előnnyel rendelkeznek, mégpedig azzal, hogy a számítások ideje alatt az ember már értékelheti és összehasonlíthatja az előzetes megoldásjelölteket.

14.3.2. Interaktív evolúció és generatív tervezés

A *genetikus algoritmusok* olyan kereső algoritmusok, amelyek a természetes kiválasztódáson és a természetes genetikán alapulnak. Egyetlen megoldás helyett megoldások egész populációjával foglalkoznak. A genetikus algoritmusokat gyakran alkalmazzák olyan nagy és bonyolult problémákra, amelyeknél a hagyományos optimalizálás csődöt mond.

Az *interaktív evolúció* olyan evolúciós algoritmus, amely emberi közreműködést igényel. Az interaktív evolúció során a felhasználó választ ki egy vagy több egyedet az aktuális populációból, amelyek túlélve és önmagukat (mutációval) reprodukálva az új generációt fogják alkotni. Így az interaktív evolúcióban a felhasználó játssza a célfüggvény szerepét, és ezért meglehetősen aktív szerepe van a kereső folyamatban.

Az olyan területeken mint a művészet, építészet, fényképfeldolgozás (beleértve a fantomképek tervezését), az interaktív evolúciónak egy speciális formáját, az úgynevezett *generatív tervezést* használják. A generatív tervezés során az aktuális generáció *összes* megoldását egyidejűleg láthatjuk a képernyőn. Itt az „összes” alatt általában egy 4 és 20 közötti számra kell gondolni. Gondoljunk a fényképfeldolgozás példára, ahol a felhasználó kiválaszthatja a módosított kontrasztot, fényerőt, szín intenzitást és élességet. A felhasználó megvizsgálja az aktuális jelölteket, és egyetlen egérkattintással bejelöli azt, amelyik a legjobban tetszik neki. Az összes többi megoldás törölve lesz, és a megjelöltnek N darab újabb mutánsa generálódik. A folyamat addig folytatható amíg a felhasználó meg lesz elégedve az eredménnyel. A generatív tervezésben járatlan ember számára talán hihetetlenül hangzik, de gyakran még egy gyenge minőségű kiinduló megoldásból is néhány iteráció alatt elfogadható eredmények születnek.

14.3.3. Egymást követő rögzítések

Számos probléma sokdimenziós, és így sok paraméter beállítását igényli. Ha egy ilyen probléma esetén jó megoldásoknak egy halmazát állítjuk elő heurisztikák ismételt alkalmazásával, akkor a következő többlépéses, interaktív folyamatot használhatjuk. Először néhány heurisztikus megoldást generálunk, amiket egy szakértő ember megvizsgál. A szakértő elsősorban „tipikus” mintákat keres a megoldásokban és rögzíti ezeket. Ezután további heurisztikus megoldásokat generálunk azzal a mellékfeltétellel, hogy valamennyien tartalmazzák a korábban rögzített részeket. A szakértő ismét megvizsgálja a megoldásokat, és újabb részeket rögzít. A folyamat addig folytatódik, amíg minden rész rögzített lesz, és így egyetlen (és remélhetőleg jó) megoldást kapunk.

14.3.4. Interaktív több feltételes döntéshozatal

A több feltételes döntéshozatal esetén nem egy, hanem kettő vagy több célfüggvényünk van. A feladat olyan elfogadható megoldások keresése, amelyek az összes célfüggvényt figyelembe véve a lehető legjobbak. Általában a célfüggvények többé-kevésbé ellentmondanak egymásnak, és így kizárják az egyértelmű optimum létezését. Hasznos lehet ilyenkor a

„hatékony megoldás” fogalma, amit a következőképpen definiálhatunk: egy hatékony megoldás esetén nem létezik olyan másik megoldás, amelyik legalább egy célfüggvény szempontjából jobb nála, az összes többi szempontjából pedig nem rosszabb.

A több feltételes döntéshozatalnál az szokott az első lépés lenni, hogy kiszámoljuk a hatékony megoldásokat. A két feltételes esetben a „hatékony” határt vizuálisan is megjeleníthetjük egy kétdimenziós diagramon, ami az emberi döntéshozónak jó áttekintést ad a lehetőségekről.

14.3.5. Különböző további témák

- *Számítógépes megoldások grafikus megjelenítése.* Az még nem elegendő, hogy a számítógép megfelelő megoldásjelölteket generál, az eredményeket megfelelő módon meg is kell jeleníteni. Egyetlen megoldás esetén a fontos részeket és tulajdonságokat kell kiemelni, míg több, egymással versengő megoldás esetén a különbségeket és a specialitásokat kell hangsúlyozni.
- *Folyamatos számítógépes futás rövid emberi közbeavatkozásokkal.* Ezt a módszert szokás „1 + 23 óra mód”-nak is nevezni a következő hasonlat miatt. Az ember minden nap 1 órát ül a számítógép előtt. Ez alatt az idő alatt megnézi az elmúlt 23 órában a számítógép által előállított eredményeket, különböző interakciókat végez a géppel, valamint megmondja neki, hogy mit csináljon a következő 23 órában. Így az ember az idejének csak egy kis részét fekteti be a munkába, a gép viszont folyamatosan fut.

  Egy jó példa a fentiekre a levelezési sakk, ahol a számítógép segítségének igénybevétele hivatalosan is megengedett. A vezető játékosok legtöbbje egy vagy több számítógépet futtat egész nap, amelyek a kritikus állásokat és folytatásokat elemzik. A sakkozók csupán összegyűjtik ezeket az eredményeket, és naponta csak egy rövid időt töltenek az elemzésükkel.
- *Váratlan hibák és numerikus instabilitások.* „Minden programban van hiba!” ezt az alapszabályt gyakran elfelejtik. Az emberek túlságosan gyakran minden kritika nélkül elhiszik, amit a monitoron látnak, vagy amit a szoftvertermék leírásában olvasnak. Mégis meglepően gyakran előfordul, hogy ugyanarra a feladatra (aminek egyetlen optimális megoldása van) több független programot futtatva különböző eredményeket kapunk. A numerikus stabilitás sincs ingyen. Ugyanarra a problémára különböző programok különböző eredményt adhatnak a kerekítési hibák miatt. Ezeket a hibalehetőségeket úgy fedezhetjük fel, ha egymástól független programokat futtatunk.

  Természetesen a hardverben is lehetnek hibák, főleg a folyamatos miniatürizálás korában. Ezért kritikus helyzetekben az lehet a jó stratégia, ha ugyanazt a programot teljesen független gépeken futtatjuk le, lehetőleg egymástól független operátor személyzet segítségével.

Gyakorlatok

14.3-1. Tekintsük az utazóügynök problémát 200 véletlenszerű (x_i, y_i) ponttal a $[0, 1]\times[0, 1]$ egységnégyzetben, az Euklideszi távolsággal. Generáljunk 100 lokálisan optimális megoldást (a kettős cserével, lásd a 14.2.1. pontban) és számoljuk össze, hogy melyik él hányszor fordul elő ebben a száz megoldásban. Definiáljunk egy K küszöböt (például $K = 30$) és rög-

zítsük azokat az éleket, amelyek legalább K megoldásban előfordulnak. Generáljunk újabb 100 megoldást, úgy, hogy a rögzített élek cseréjét ne engedjük meg. Ezt ismételjük addig, amíg a folyamat nem konvergál, majd hasonlítsuk össze a végeredményt az első sorozatok jellemző lokális optimumaival.

Megjegyzések a fejezethez

Az „ember-gép kapcsolat” bevezetésben idézett definíciójának forrása a „HCI Bibliography” [366].

Sameith [409] technikai jelentésében számos kísérletet, leírást és elemzést találunk a büntető módszerre, különböző összeg típusú problémák, dimenziók és hibaszélességek esetére. A 14.3. tétel bizonyítása először [14]-ben jelent meg. Az e-kereskedelemben a több választási lehetőséget kínáló rendszereket gyakran „Tanácsadó rendszerek”-nek nevezik – mint például Resnick és Varian cikkében [384] – szem előtt tartva a vevőket, akik számára az őket érdeklő termékeket ki kell listázni. Érthető okokból a kereskedelmi kereső motorok és az e-cégek titokban tartják a kivonatoló algoritmusaikat.

A 14.2.5. pontban említett áramlástani tétel megtalálható Ahuja, Magnanti és Orlin könyvében [7].

Egy útvonaltervező programot forgalmaz az AND cég [514]. Műholdas felvételeken alapuló útvonaltervezéssel foglalkozik Berger [44] diplomamunkája.

A BigBlackCell nevű szoftver Grosse és Schwarz munkája [178].

A genetikus algoritmusokról például Goldberg [156] tekinthető jó tankönyvnek. Az interaktív evolúciót és a generatív tervezést Banzhaf [30] tárgyalja. A több feltételes döntéshozatallal több cikk is részletesen foglalkozik, az egyik alapmű Gal, Stewart és Hanne könyve [145].

Althöfer könyvében [11] a 3-agy történeti hátteréről és a versenysakkban elért sikereiről olvashatunk. A 3-agy és Juszupov nagymester közötti mérkőzésről [12] számol be. [13] általános tájékoztatást ad arról az esetről, amikor több számítógép javaslatát használva javítjuk a játékerőt. [15] néhány k-legjobb megvalósítást mutat be játékok fájában való keresésre iteratív mélyítéssel. Ezen megvalósítások képernyőmentéseit a következő web címen tekinthetjük meg: http://www.minet.uni-jena.de/www/fakultaet/iam/personen/k-best.html. [200] a sakkprogramok és más bonyolultabb játékok technikai hátterét mutatja be.

M. Zuker és D. H. Turner által írt programok értékes on-line gyűjteménye található a http://www.bioinfo.rpi.edu/applications/mfold címen. A felhasználó bevihet például RNS-láncokat, és a rendszer valós időben előállítja ezen láncok lehetséges másodlagos térszerkezetét. Többek között olyan paraméterek adhatók meg, mint a „kiszámított gyűrődések száma” (alapértelmezés = 50), vagy például az optimálistól való „százalékos eltérés mértéke” (alapértelmezés = 5 %).

A genetikai algoritmusokkal magyarul foglalkozik Álmos Attila, Győri Sándor, Horváth Gábor és Várkonyiné Kóczy Annamária könyve [293], a kapcsolódó biológiai fogalmak megismeréséhez pedig Podani János könyvét [374] ajánljuk.

15. Számítógépes grafika

A számítógépes grafika egy ***virtuális világot*** épít fel a memória adatszerkezeteiben, amit egy virtuális kamerával fényképez le. A virtuális világ ***alakzatokat*** (pontokat, szakaszokat, felületeket, testeket stb.) tartalmaz, amit az algoritmikus feldolgozáshoz számokkal írunk le. A ***képszintézis*** a virtuális világ és a kamera tulajdonságai alapján számítja ki a képet. A kép feltételezésünk szerint azonos méretű kicsiny téglalapokból, úgynevezett képelemekből, ***pixelekből*** áll. Egy képelemhez egyetlen szín rendelhető, így elegendő a képszintézist pixelenként egyetlen pontban, például a középpontban végrehajtani. A fényképezés megkeresi a ponton keresztül látható alakzatot, és annak színét írja a kép pixelébe. Ebben a könyvben csak a látható alakzatok meghatározásával foglalkozunk, az alakzatok színét ismertnek tételezzük fel.

A következő fejezetekben először áttekintjük, hogyan írhatjuk le az alakzatokat számokkal, majd megismerkedünk a fényképezési folyamat algoritmusaival.

15.1. Analitikus geometriai alapok

Vizsgálatunk alaphalmaza általában az euklideszi ***tér.*** A számítógépes algoritmusokban a tér elemeit számokkal kell leírni. A geometria számokkal dolgozó ága az ***analitikus geometria,*** melynek alapvető eszköze a vektor és a koordinátarendszer.

15.1. definíció. *A **vektor** egy irányított szakasz, vagy másképpen egy **eltolás,** aminek iránya és hossza van, és amely a tér egy pontjához azt a másik pontot rendeli hozzá, amelyik tőle a megadott irányban és a vektor hosszának megfelelő távolságban van. A vektorokra a $\vec{v}$ jelölést fogjuk alkalmazni.*

A vektor hosszát gyakran a vektor ***abszolút értékének*** is mondjuk és $|\vec{v}|$-vel jelöljük. A vektorokon értelmezzük az ***összeadás*** műveletet, amelynek eredménye egy újabb vektor, amely az összeadandó eltolások egymás utáni végrehajtását jelenti. A továbbiakban az összeadásra a $\vec{v}_1 + \vec{v}_2 = \vec{v}$ jelölést alkalmazzuk. Beszélhetünk egy vektor és egy szám szorzatáról, amely ugyancsak vektor lesz ($\lambda \cdot \vec{v}_1 = \vec{v}$), és ugyanabba az irányba tol el, mint a $\vec{v}_1$ szorzandó, de a megadott λ szám arányában kisebb vagy nagyobb távolságra.

Két vektor ***skaláris szorzata*** egy szám, amely egyenlő a két vektor hosszának és a bezárt szögük koszinuszának a szorzatával:

$$\vec{v}_1 \cdot \vec{v}_2 = |\vec{v}_1| \cdot |\vec{v}_2| \cdot \cos\alpha, \quad \text{ahol } \alpha \text{ a } \vec{v}_1 \text{ és } \vec{v}_2 \text{ vektorok által bezárt szög .}$$

Két vektor ***merőleges,*** ha skaláris szorzatuk zérus.

Másrészt, két vektor ***vektoriális szorzata*** egy vektor, amely merőleges a két vektor síkjára, a hossza pedig a két vektor hosszának és az általuk bezárt szög szinuszának a szorzata:

$$\vec{v}_1 \times \vec{v}_2 = \vec{v}, \quad \text{ahol } \vec{v} \text{ merőleges } \vec{v}_1, \vec{v}_2\text{-re, és } |\vec{v}| = |\vec{v}_1| \cdot |\vec{v}_2| \cdot \sin\alpha \ .$$

A két lehetséges merőleges közül azt az irányt tekintjük a vektoriális szorzat irányának, amerre a jobb kezünk középső ujja mutatna, ha a hüvelykujjunkat az első vektor irányába, a mutatóujjunkat pedig a második vektor irányába fordítanánk ***(jobbkéz szabály).*** Két vektor ***párhuzamos,*** ha vektoriális szorzatuk nulla.

15.1.1. A Descartes-koordinátarendszer

A sík bármely $\vec{v}$ vektora egyértelműen kifejezhető két, nem párhuzamos $\vec{i}$, $\vec{j}$ vektor lineáris kombinációjaként, azaz

$$\vec{v} = x \cdot \vec{i} + y \cdot \vec{j}$$

alakban. Hasonlóan a tér bármely $\vec{v}$ vektora egyértelműen megadható három, nem egy síkba eső vektor lineáris kombinációjaként:

$$\vec{v} = x \cdot \vec{i} + y \cdot \vec{j} + z \cdot \vec{k} \ .$$

Az $\vec{i}$, $\vec{j}$, $\vec{k}$ vektorokat ***bázisvektornak,*** az x, y, z skalárokat ***koordinátáknak*** nevezzük. A továbbiakban feltételezzük, hogy a bázisvektorok egységnyi hosszúak, és egymásra páronként merőlegesek. A bázisvektorok ismeretében bármely vektor egyértelműen megadható számokkal, mégpedig a koordinátáival.

Egy ***pontot*** egy vektorral adunk meg úgy, hogy megmondjuk, hogy az a tér egy kitüntetett pontjához, az ***origóhoz*** képest milyen irányban és távolságra van. Ezt a vektort a pont ***helyvektorának*** nevezzük.

Az origó és a bázisvektorok együttese a ***Descartes-koordinátarendszer,*** amellyel az euklideszi sík, illetve a tér pontjai egyértelműen számszerűsíthetők.

A Descartes-koordinátarendszer az euklideszi geometria algebrai megalapozása, amin azt értjük, hogy a ***Descartes-koordináta*** hármasok a tér pontjainak megfeleltethetők, és a geometriai, illetve algebrai fogalmak párba állítása után az euklideszi geometria axiómái (és így a tételei is) algebrai eszközökkel igazolhatók.

Gyakorlatok

15.1-1. Igazoljuk, hogy a Descartes-koordináták és a pontok egy-egyértelmű kapcsolatban állnak.

15.1-2. Bizonyítsuk be, hogy ha a bázisvektorok egységnyi hosszúak és egymásra páronként merőlegesek, akkor $(x_1, y_1, z_1) \cdot (x_2, y_2, z_2) = x_1x_2 + y_1y_2 + z_1z_2$.

15.1-3. Igazoljuk, hogy a skaláris szorzás az összeadásra nézve disztributív.

15.2. Ponthalmazok leírása egyenletekkel

A koordinátarendszerek lehetőséget adtak pontok számokkal történő megadására. Egy koordinátákra megfogalmazott feltételrendszerrel pedig egy teljes ponthalmazt azonosíthatunk.

test	$f(x,y,z)$ implicit függvény
R sugarú ***gömb***	$R^2 - x^2 - y^2 - z^2$
$2a, 2b, 2c$ élű ***téglatest***	$\min\{a - \lvert x\rvert, b - \lvert y\rvert, c - \lvert z\rvert\}$
z tengelyű, r (hurka) és R (lyuk) sugarú ***tórusz***	$r^2 - z^2 - (R - \sqrt{x^2 + y^2})^2$

15.1. ábra. Néhány – origó középpontú – testet definiáló implicit függvény.

A feltételrendszer általában egyenlet vagy egyenlőtlenség, amelyet kielégítő koordinátahármasokhoz tartozó pontokat mondjuk a ponthalmazhoz tartozónak.

15.2.1. Testek

A ***test*** a háromdimenziós tér egy részhalmaza. A részhalmaz kijelöléséhez egy folytonos f függvényt hívunk segítségül, amely a tér pontjait a valós számokra képezi le, és azokat a pontokat tekintjük a testhez tartozónak, amelyek kielégítik az alábbi *implicit* egyenlőtlenséget:

$$f(x,y,z) \geq 0 \,.$$

Az $f(x,y,z) > 0$ egyenlőtlenséget teljesítő pontok a test ***belső pontjai,*** az $f(x,y,z) < 0$ egyenlőtlenséget kielégítők a ***külső pontok.*** Az f függvény folytonossága miatt a külső és belső pontok „között” találjuk az $f(x,y,z) = 0$ egyenletet kielégítő pontokat, amelyek a test ***határfelületét*** alkotják. Szemléletesen a külső és belső pontokra az f függvény a határfelülettől mért előjeles távolságot jellemzi.

Megjegyezzük, hogy szigorú értelemben nem tekintjük a tér pontjainak bármilyen részhalmazát testnek, csak azokat, amelyek nem tartalmaznak háromnál alacsonyabb dimenziós elfajulásokat (például a testből kilógó görbéket, és felületeket), azaz megköveteljük, hogy a határfelület minden pontjának tetszőlegesen kicsiny környezetében belső pont is legyen. Nem kell azonban betartanunk ezt a feltételt, ha a megjelenítő algoritmus figyelmen kívül hagyja az elfajuló részeket.

A 15.1. ábra bemutatja a gömböt, téglatestet és tóruszt definiáló implicit függvényt.

15.2.2. Felületek

Az $f(x,y,z) = 0$ egyenletet kielégítő pontok a test határpontjai, amelyek egy ***felületet*** alkotnak. A felületeket tehát leírhatjuk ezzel az ***implicit egyenlettel.*** Mivel a pontokat azok helyvektoraival is definiálhatjuk, az implicit egyenletet megfogalmazhatjuk magukra a helyvektorokra is:

$$f(\vec{r}) = 0 \,.$$

Egy felületet sokféleképpen adhatunk meg egyenlettel, például az $f(x,y,z) = 0$ helyett az $f^2(x,y,z) = 0$ és a $2 \cdot f^3(x,y,z) = 0$ nyilván ugyanazon pontokat jelöli ki.

Egy $\vec{n}$ normálvektorú és $\vec{r}_0$ helyvektorú ***sík*** azon $\vec{r}$ pontokat tartalmazza, amelyekre az $\vec{r} - \vec{r}_0$ vektor a sík normálvektorára merőleges, tehát skalárszorzatuk zérus, így a sík pontjainak implicit vektor és skalár egyenlete:

$$(\vec{r} - \vec{r}_0) \cdot \vec{n} = 0, \qquad n_x \cdot x + n_y \cdot y + n_z \cdot z + d = 0 \,, \tag{15.1}$$

test	$x(u,v)$	$y(u,v)$	$z(u,v)$
R sugarú ***gömb***	$R \cdot \cos 2\pi u \cdot \sin \pi v$	$R \cdot \sin 2\pi u \cdot \sin \pi v$	$R \cdot \cos \pi v$
R sugarú, z tengelyű, h magasságú ***henger***	$R \cdot \cos 2\pi u$	$R \cdot \sin 2\pi u$	$h \cdot v$
R sugarú, z tengelyű, h magasságú ***kúp***	$R \cdot (1-v) \cdot \cos 2\pi u$	$R \cdot (1-v) \cdot \sin 2\pi u$	$h \cdot v$

15.2. ábra. Néhány – felületet definiáló – paraméteres forma, ahol $u, v \in [0,1]$.

ahol n_x, n_y, n_z a normálvektor koordinátái és $d = -\vec{r}_0 \cdot \vec{n}$. Amennyiben a normálvektor hossza egységnyi, a d a síknak az origótól vett előjeles távolsága. Két síkot ***párhuzamosnak*** nevezünk, ha normálvektoraik párhuzamosak.

Az implicit egyenlet helyett a másik lehetőség a ***paraméteres forma*** alkalmazása, amikor a felületnél két szabad paraméter alapján állítjuk be a koordinátákat. Például egy felület paraméteres egyenlete az u, v szabad paraméterekkel:

$$x = x(u,v), \qquad y = y(u,v), \qquad z = z(u,v), \qquad u \in [u_{\min}, u_{\max}], \quad v \in [v_{\min}, v_{\max}] \ .$$

A felület paraméteres egyenleteiből az implicit egyenletet úgy származtathatjuk, hogy a három egyenletből kiküszöböljük az u, v szabad paramétereket.

A 15.2. ábra bemutatja a gömböt, hengert és kúpot definiáló paraméteres formát.

Az implicit egyenlethez hasonlóan, a paraméteres egyenleteket megfogalmazhatjuk magukra a helyvektorokra is:

$$\vec{r} = \vec{r}(u,v) \ .$$

A ***háromszög*** a $\vec{p}_1, \vec{p}_2$ és $\vec{p}_3$ pontok ***konvex-kombinációinak*** halmaza, azaz

$$\vec{r}(\alpha,\beta,\gamma) = \alpha \cdot \vec{p}_1 + \beta \cdot \vec{p}_2 + \gamma \cdot \vec{p}_3, \quad \text{ahol } \alpha,\beta,\gamma \geq 0 \text{ és } \alpha+\beta+\gamma = 1 \ .$$

A szokásos, kétparaméteres egyenlethez úgy jutunk, hogy az α-t u-val, a β-t v-vel, a γ-t pedig $(1-u-v)$-vel helyettesítjük:

$$\vec{r}(u,v) = u \cdot \vec{p}_1 + v \cdot \vec{p}_2 + (1-u-v) \cdot \vec{p}_3, \quad \text{ahol } u,v \geq 0 \text{ és } u+v \leq 1 \ .$$

15.2.3. Görbék

Két felület metszeteként ***görbét*** kapunk, amelyet megadhatunk a két metsző felület

$$f_1(x,y,z) = f_2(x,y,z) = 0$$

alakú implicit egyenleteivel is, de ez általában feleslegesen körülményes. Tekintsük inkább a két felület $\vec{r}_1(u_1,v_1)$, illetve $\vec{r}_2(u_2,v_2)$ paraméteres formáit. A metszet pontjai kielégítik az $\vec{r}_1(u_1,v_1) = \vec{r}_2(u_2,v_2)$ vektoregyenletet, amely a háromdimenziós térben három skalár-egyenletnek felel meg. A négy ismeretlen (u_1,v_1,u_2,v_2) paraméterből tehát hármat kiküszöbölhetünk, így a görbét az egyváltozós

$$x = x(t), \qquad y = y(t), \qquad z = z(t), \qquad t \in [t_{\min}, t_{\max}]$$

függvényekkel, vagy az

$$\vec{r} = \vec{r}(t), \qquad t \in [t_{\min}, t_{\max}]$$

vektoriális alakkal írhatjuk le.

test	$x(u,v)$	$y(u,v)$	$z(u,v)$
$2a$, $2b$ főtengelyű, z síkon lévő ***ellipszis***	$a \cdot \cos 2\pi t$	$b \cdot \sin 2\pi t$	0
R sugarú, z irányban h emelkedésű ***csavarvonal***	$R \cdot \cos 2\pi t$	$R \cdot \sin 2\pi t$	$h \cdot t$
(x_1, y_1, z_1) és (x_2, y_2, z_2) közötti ***szakasz***	$x_1(1-t) + x_2 t$	$y_1(1-t) + y_2 t$	$z_1(1-t) + z_2 t$

15.3. ábra. Néhány – görbét definiáló – paraméteres forma, ahol $t \in [0,1]$.

A 15.3. ábra bemutatja az ellipszist, csavarvonalat és szakaszt definiáló paraméteres formát.

Figyeljük meg, hogy a felületeken úgy is felvehetünk görbéket, hogy az u, v szabad paraméterek közül az egyiket rögzítjük. Például a v paraméter rögzítésével kapott görbe paraméteres alakja $\vec{r}_v(u) = \vec{r}(u,v)$. Ezeket a görbéket ***izoparametrikus görbéknek*** nevezzük.

Az ***egyenes*** pontjai közül jelöljünk ki egyet, és a kijelölt pont helyvektorát nevezzük az ***egyenes helyvektorának.*** Az egyenes egy tetszőleges pontját a kijelölt pont egy kitüntetett iránnyal párhuzamos eltolásával kaphatjuk meg. A helyvektort $\vec{r}_0$ vektorral, a kitüntetett irányt pedig $\vec{v}$ ***irányvektorral*** jelölve, az ***egyenes egyenlete:***

$$\vec{r}(t) = r_0 + \vec{v} \cdot t, \qquad t \in (-\infty, \infty)\ . \tag{15.2}$$

Két egyenest ***párhuzamosnak*** mondunk, ha irányvektoraik párhuzamosak.

A teljes egyenes helyett egy szakasz pontjait is megadhatjuk, ha a t paramétert egy véges intervallumra korlátozzuk. Például az $\vec{r}_1, \vec{r}_2$ pontok közötti ***szakasz egyenlete:***

$$\vec{r}(t) = \vec{r}_1 + (\vec{r}_2 - \vec{r}_1) \cdot t = \vec{r}_1 \cdot (1-t) + \vec{r}_2 \cdot t, \qquad t \in [0,1]\ . \tag{15.3}$$

Ezen definíció szerint a szakasz pontjai a végpontjainak ***konvex-kombinációi.***

15.2.4. Normálvektorok

A számítógépes grafikában a felülethez tartozó pontok mellett gyakran szükség van az egyes pontokban a felület normálvektorára is (azaz a felületet érintő sík normálvektorára). Vegyünk egy példát. A tükör a fényt úgy veri vissza, hogy a megvilágítási irány, a felületi normális és a visszaverődési irány egy síkban van, és a beesési szög megegyezik a visszaverődési szöggel. Ezen (és hasonló) számítások elvégzéséhez tehát a normálvektort is elő kell állítani.

Az implicit egyenlet alapján az érintősík egyenletét a felület (x_0, y_0, z_0) pont körüli elsőfokú Taylor-soros közelítésével kaphatjuk:

$$f(x,y,z) = f(x_0 + (x-x_0), y_0 + (y-y_0), z_0 + (z-z_0)) \approx$$

$$f(x_0,y_0,z_0) + \frac{\partial f}{\partial x} \cdot (x-x_0) + \frac{\partial f}{\partial y} \cdot (y-y_0) + \frac{\partial f}{\partial z} \cdot (z-z_0)\ .$$

Az (x_0, y_0, z_0) és (x,y,z) pontok a felületen vannak, így $f(x_0,y_0,z_0) = 0$ és $f(x,y,z) = 0$, ezért az ***érintősík egyenlete:***

$$\frac{\partial f}{\partial x} \cdot (x-x_0) + \frac{\partial f}{\partial y} \cdot (y-y_0) + \frac{\partial f}{\partial z} \cdot (z-z_0) = 0\ .$$

Az egyenletet a (15.1) egyenlettel összevetve megállapíthatjuk, hogy a felület Taylor-soros

közelítésével kapott sík normálvektora éppen

$$\vec{n} = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right) = \mathrm{grad} f \ . \tag{15.4}$$

A paraméteres felület normálvektorához az izoparametrikus vonalak tanulmányozásával juthatunk. A v paraméter rögzítésével kapott $\vec{r}_v(u)$ görbe érintőjét elsőfokú Taylor-soros közelítéssel kapjuk:

$$\vec{r}_v(u) = \vec{r}_v(u_0 + (u - u_0)) \approx \vec{r}_v(u_0) + \frac{d\vec{r}_v}{du} \cdot (u - u_0) = \vec{r}_v(u_0) + \frac{\partial \vec{r}}{\partial u} \cdot (u - u_0) \ .$$

A közelítést az egyenes (15.2) egyenletével összevetve megállapíthatjuk, hogy az érintő irányvektora $\partial\vec{r}/\partial u$. A felületre illeszkedő görbék érintői a felületet érintő síkban vannak, így a normálvektor a felületre illeszkedő görbék érintőinek az irányvektoraira merőleges. A normálvektor egyértelmű előállításához ki kell számítanunk mind az $\vec{r}_v(u)$ érintőjét, mind pedig az $\vec{r}_u(v)$ érintőjét, és a két érintő irányvektorainak a vektoriális szorzatát kell képezni, mert a vektoriális szorzat mindkét tényezőre merőleges eredményt ad. Az $\vec{r}(u, v)$ felület normálvektora tehát

$$\vec{n} = \frac{\partial \vec{r}}{\partial u} \times \frac{\partial \vec{r}}{\partial v} \ . \tag{15.5}$$

15.2.5. Görbemodellezés

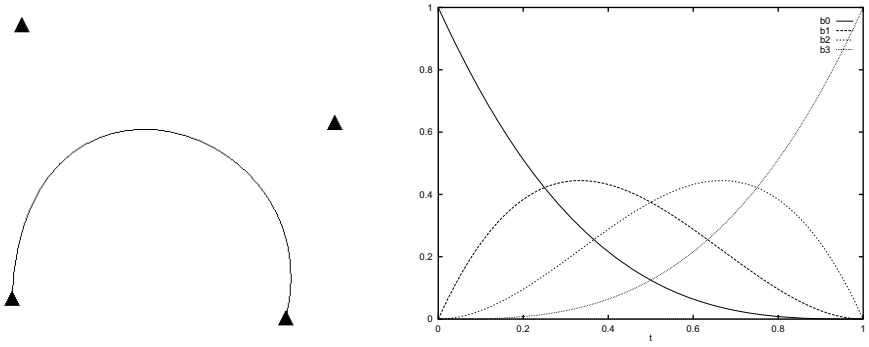
A ponthalmazok paraméteres, illetve implicit egyenletekkel történő leírásával a virtuális világ geometriai tervezését egyenletek megadására, a képszintézis feladatot pedig egyenletek megoldására vezethetjük vissza. Az egyenletek azonban nagyon kevéssé szemléletesek, így a geometriai tervezésnél közvetlenül nem alkalmazhatók. Nyilván nem várhatjuk, hogy a tervező egy emberi arc vagy egy sportkocsi felépítése során közvetlenül ezen alakzatok egyenleteit adja meg. Indirekt módszerekre van szükségünk, amelyekben a program a tervezőtől szemléletes információkat kér, amiből az egyenletet automatikusan építi fel. Az indirekt megközelítés egyik irányzata vezérlőpontokból indul ki, a másik pedig elemi építőelemekkel (kocka, gömb, kúp stb.) és halmazműveletekkel dolgozik.

Tekintsük először a pontalapú eljárás alkalmazását görbék definiálására. Legyen a tervező által kijelölt ***vezérlőpont-sorozat*** $\vec{r}_0, \vec{r}_1, \ldots, \vec{r}_m$, és keressünk egy $\vec{r} = \vec{r}(t)$ $(t_0 \le t \le t_m)$ paraméteres egyenletű görbét, amely „követi" a kijelölt pontokat. Egyelőre nem követeljük meg, hogy a görbe át is menjen a kijelölt vezérlőpontokon.

A görbe felépítéséhez a mechanikai rendszerek súlypontjának analógiáját használjuk. Tételezzük fel, hogy egységnyi tömegű homokunk van, amelyet szétosztunk a vezérlőpontok között. Ha egy vezérlőpontban van a homok nagy része, akkor a rendszer súlypontja is ennek közelébe kerül. Ha a t paraméter függvényében a homok eloszlását folytonosan úgy változtatjuk, hogy mindig más vezérlőpont jusson domináns szerephez, akkor a súlypont egy görbét fut be, egymás után megközelítve a vezérlőpontokat.

Legyenek a t paraméter mellett a vezérlőpontokba helyezett súlyok $B_0(t), B_1(t), \ldots, B_m(t)$, amelyeket a görbe ***bázisfüggvényeinek*** nevezünk. Mivel egységnyi súlyt osztunk szét, megkívánjuk, hogy minden t-re fennálljon a következő azonosság:

$$\sum_{i=0}^{m} B_i(t) = 1 \ .$$



15.4. ábra. Négy vezérlőpont által definiált Bézier-görbe és bázisfüggvényei ($m = 3$).

Adott t-re a görbe pontját ezen mechanikai rendszer súlypontjaként értelmezzük:

$$\vec{r}(t) = \frac{\sum_{i=0}^{m} B_i(t) \cdot \vec{r}_i}{\sum_{i=0}^{m} B_i(t)} = \sum_{i=0}^{m} B_i(t) \cdot \vec{r}_i \ .$$

Figyeljük meg, hogy azért érdemes mindig egységnyi tömegű homokot szétosztani, mert ekkor a tört nevezője egységnyi lesz. A vezérlőpontokat kis mágnesekként is elképzelhetjük, amelyek a bázisfüggvényeknek megfelelő erővel vonzzák a görbét. Ha a bázisfüggvények nem negatívak, akkor ez az erő sohasem lehet taszító, és a súlypont analógia is teljes (a súly sem lehet negatív). Egy pontrendszer súlypontja a pontok ***konvex burkában***[1] van, így nemnegatív bázisfüggvények esetén a görbénk is a vezérlőpontok konvex burkában marad.

A görbék tulajdonságait a bázisfüggvények határozzák meg. Most két közkedvelt bázisfüggvény rendszert ismertetünk: a Bézier- és a B-spline-görbék bázisfüggvényeit.

Bézier-görbe

A Renault gyár Pierre Bézier nevű konstruktőre a ***Bernstein-polinomokat*** javasolta bázisfüggvényeknek, amelyeket az $1^m = (t+(1-t))^m$ binomiális tétel szerinti kifejtésével kapunk:

$$(t + (1 - t))^m = \sum_{i=0}^{m} \binom{m}{i} \cdot t^i \cdot (1 - t)^{m-i} \ .$$

A ***Bézier-görbe*** bázisfüggvényei ezen összeg tagjai ($i = 0, 1, \ldots, m$):

$$B_{i,m}^{\text{Bezier}}(t) = \binom{m}{i} \cdot t^i \cdot (1 - t)^{m-i} \ . \tag{15.6}$$

A Bernstein-polinomok bevezetéséből nyilvánvaló, hogy a bázisfüggvények valóban teljesítik a $\sum_{i=0}^{m} B_i(t) = 1$ és $B_i(t) \geq 0$ feltételeket a $t \in [0, 1]$ tartományban, így a görbe mindig a vezérlőpontok konvex burkában marad. A görbe bázisfüggvényeit és alakját a 15.4.

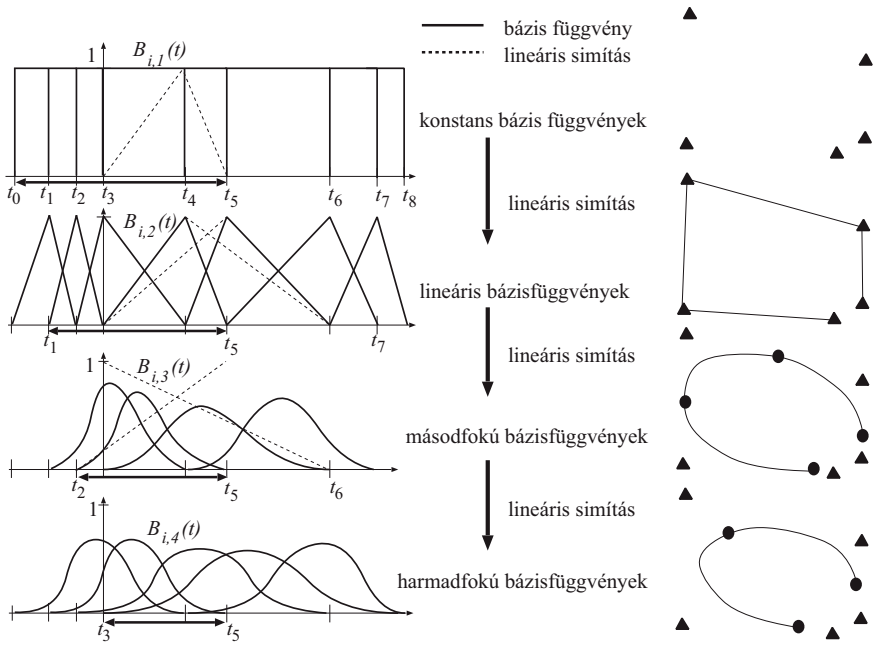
[1] Definíciószerűen egy pontrendszer konvex burka az a minimális konvex halmaz, amely tartalmazza a pontrendszert.

ábrán láthatjuk. A $t = 0$ paraméterértékeknél a legelső bázisfüggvény 1 értékű, a többi pedig nulla, ezért a görbe az első vezérlőpontból indul. Hasonló okok miatt $t = 1$ paraméterértéknél a görbe az utolsó vezérlőpontba érkezik. A többi paraméterértéknél azonban mindegyik bázisfüggvény pozitív, így a görbére az összes vezérlőpont együttesen hat. A görbe ezért általában nem megy át a többi vezérlőponton.

B-spline

A második görbénk, a ***B-spline*** bázisfüggvényeit lineáris simítások sorozatával konstruálhatjuk meg. A B-spline az $m+1$ darab vezérlőpontot $(k-1)$-edfokú bázisfüggvényekkel súlyozza. A k a görbe ***rendszáma,*** amely kifejezi a görbe simaságát. Vegyünk fel egy $m+k+1$ elemű, nem csökkenő paramétersorozatot, amelyet ***csomóvektornak*** nevezünk:

$$\mathbf{t} = [t_0, t_1, \ldots, t_{m+k}], \quad t_0 \le t_1 \le \cdots \le t_{m+k} \ .$$



15.5. ábra. A B-spline bázisfüggvények létrehozása. Egy magasabb rendű bázisfüggvényt a megelőző rend két egymást követő bázisfüggvényének lineárisan növekvő, illetve lineárisan csökkenő súlyozását követő összeadásával kapjuk. A vezérlőpontok száma 5, azaz $m = 4$. Az ábrán nyíllal jelöltük a $[t_{k-1}, t_{m+1}]$ hasznos tartományt, ahová $m+1$ bázisfüggvény lóg be, amelyek összege itt azonosan 1. Az ábra jobb oldalán a görbék mellett a kis háromszögek a vezérlőpontokat, a körök pedig a csomóértékeknek megfelelő görbepontokat mutatják.

Az elsőrendű, i-edik bázisfüggvényt tekintsük 1 értékűnek az i-edik paraméterintervallumban, azon kívül pedig zérusnak (15.5. ábra):

$$B_{i,1}^{\mathrm{BS}}(t) = \begin{cases} 1, & \text{ha } t_i \le t < t_{i+1} \ , \\ 0 & \text{különben} \ . \end{cases}$$

Ezzel $m+k$ darab bázisfüggvényünk született, amelyek nulladfokú polinomok, nem negatívak és összegük minden $t \in [t_0, t_{m+k})$ paraméterre azonosan 1. Ezek a bázisfüggvények

annyira alacsonydimenziósak, hogy a súlypont nem is egy görbén halad végig, csak a vezérlőpontokon ugrál.

A bázisfüggvények fokszámát, és ezáltal a görbe simaságát úgy növelhetjük, hogy a következő szint bázisfüggvényeinek az előállításához a jelenlegi készletből két egymást követő bázisfüggvényt lineáris súlyozással összeadunk (15.5. ábra). Az első bázisfüggvényt a $t_i \le t < t_{i+1}$ nem zérus értékű tartományában a lineárisan növekvő $(t-t_i)/(t_{i+1}-t_i)$ kifejezéssel szorozzuk, így a hatását lineárisan zérusról egyre növeljük. A következő bázisfüggvényt pedig annak $t_{i+1} \le t < t_{i+2}$ tartományában lineárisan csökkenő $(t_{i+2}-t)/(t_{i+2}-t_{i+1})$ függvénnyel skálázzuk. Az így súlyozott bázisfüggvényeket összeadva kapjuk a másodrendű változat sátorszerű bázisfüggvényeit. Figyeljük meg, hogy míg az elsőrendű, konstans bázisfüggvények egy-egy intervallumon voltak nem zérus értékűek, a másodrendű, sátorszerű bázisfüggvényekre ez már két intervallumra igaz. Mivel mindig két egymást követő bázisfüggvényből hoztunk létre egy újat, az új bázisfüggvények száma eggyel kevesebb, már csak $m+k-1$ darab van belőlük. A két szélső elsőrendű bázisfüggvény kivételével mindegyik egyszer növekvő, egyszer pedig csökkenő súlyozással épült be a másodrendű bázisfüggvényekbe, így a két szélső intervallum kivételével, azaz a $[t_1, t_{m+k-1}]$ tartományban továbbra is fennáll, hogy az ide belógó[2] $m+k-1$ darab bázisfüggvény összege azonosan 1.

A másodrendű bázisfüggvények elsőfokú polinomok. A bázisfüggvények fokszámát, azaz a görbe rendjét, az ismertetett eljárás rekurzív alkalmazásával tetszőleges szintig növelhetjük. A következő rend bázisfüggvényei az alábbi módon függenek az előzőtől:

$$B_{i,k}^{\mathrm{BS}}(t) = \frac{(t-t_i)B_{i,k-1}^{\mathrm{BS}}(t)}{t_{i+k-1}-t_i} + \frac{(t_{i+k}-t)B_{i+1,k-1}^{\mathrm{BS}}(t)}{t_{i+k}-t_{i+1}}, \qquad \text{ha } k > 1 \ .$$

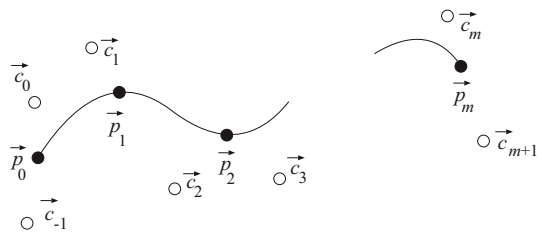
Figyeljük meg, hogy mindig két egymást követő bázisfüggvényt veszünk, és az elsőt a pozitív tartományában (azaz abban a tartományban, ahol nem azonosan zérus) lineárisan növekvő $(t-t_i)/(t_{i+k-1}-t_i)$ függvénnyel, a következőt pedig annak a pozitív tartományában lineárisan csökkenő $(t_{i+k}-t)/(t_{i+k}-t_{i+1})$ függvénnyel szorozzuk meg. Az eredményeket összeadva megkapjuk a még simább bázisfüggvényeket. A műveletsort $(k-1)$-szer megismételve k-adrendű bázisfüggvényekhez jutunk, amelyekből a $[t_{k-1}, t_{m+1}]$ hasznos tartományba éppen $m+1$ darab lóg be, amelyek összege továbbra is azonosan 1. A csomóvektor megadásánál megengedtük, hogy a csomóértékek ne szigorúan monoton növekvő sorozatot alkossanak, így az egyes intervallumok hossza akár zérus is lehet. Ez 0/0 jellegű törtekhez vezethet, amelyeket az algoritmus megvalósítása során 1 értékkel kell helyettesíteni.

A k-adrendű i-edik bázisfüggvény értékét a t helyen a következő ***Cox-deBoor algoritmussal*** számíthatjuk:

[2] Akkor mondjuk, hogy a bázisfüggvény egy intervallumba lóg, ha értéke ebben az intervallumban nem azonosan zérus.

B-SPLINE$(i, k, t, \mathbf{t})$

```
1  if k = 1                                                    ▷ Triviális eset vizsgálata.
2     then if t_i ≤ t < t_{i+1}
3             then return 1
4             else return 0
5  if t_{i+k-1} − t_i > 0
6     then b_1 ← (t − t_i)/(t_{i+k-1} − t_i)                   ▷ Előző lineárisan növekvő súllyal.
7     else  b_1 ← 1                                            ▷ Itt: 0/0 = 1.
8  if t_{i+k} − t_{i+1} > 0
9     then b_2 ← (t_{i+k} − t)/(t_{i+k} − t_{i+1})             ▷ Következő lineárisan növekvő súllyal.
10    else  b_2 ← 1                                            ▷ Itt: 0/0 = 1.
11 B ← b_1 · B-SPLINE(i, k − 1, t, t) + b_2 · B-SPLINE(i + 1, k − 1, t, t)   ▷ Rekurzió.
12 return B
```



15.6. ábra. A B-spline interpoláció. A $\vec{p}_0, \ldots, \vec{p}_m$ interpolálandó pontok alapján számítjuk a $\vec{c}_{-1}, \ldots, \vec{c}_{m+1}$ vezérlőpont sorozatot úgy, hogy az egyes szegmensek kezdő- és végpontjai éppen az interpolálandó pontok legyenek.

A gyakorlatban a bázisfüggvényeket negyedrendűnek választjuk ($k = 4$), amelyek harmadfokú polinomok, a görbe pedig kétszer folytonosan differenciálható. Ennek az a magyarázata, hogy a meghajlított rudak alakja és a newtoni mozgástörvényeket kielégítő mozgáspályák ugyancsak kétszer folytonosan differenciálhatók.

Amíg a vezérlőpontok száma a görbe rendszámánál nagyobb, az egyes bázisfüggvények csak az érvényes paramétertartomány egy-egy részében nem nulla értékűek. Ez azt jelenti, hogy egy vezérlőpont csak a görbe egy részére hat, így megváltoztatása a görbét csak ***lokálisan módosítja.*** Ez egy nagyon kedvező tulajdonság, hiszen ekkor a tervezőt nem fenyegeti az a veszély, hogy a görbe egy részének kicsiny módosítása elrontja a görbe alakját távolabb.

A negyedrendű B-spline általában nem megy át a vezérlőpontjain. Ha interpolációs célokra szeretnénk használni, a görbe vezérlőpontjait az interpolálandó pontokból kell kiszámítani. Tegyük fel, hogy egy olyan görbét keresünk, amely a $t_0 = 0, t_1 = 1, \ldots, t_m = m$ paraméterértékeknél éppen a $\vec{p}_0, \vec{p}_1, \ldots, \vec{p}_m$ pontokon megy át (15.6. ábra). Ehhez a görbénk $[\vec{c}_{-1}, \vec{c}_0, \vec{c}_1, \ldots, \vec{c}_{m+1}]$ vezérlőpontjait úgy kell kitalálni, hogy a következő interpolációs feltétel teljesüljön:

$$\vec{r}(t_j) = \sum_{i=-1}^{m+1} \vec{c}_i \cdot B_{i,4}^{\mathrm{BS}}(t_j) = \vec{p}_j, \qquad j = 0, 1, \ldots, m\ .$$

Ez egy $m + 3$ ismeretlenes lineáris egyenletrendszer $m + 1$ egyenletét határozza meg, tehát több megoldás is lehetséges. A feladatot teljesen meghatározottá tehetjük, ha még két járulékos feltételt felveszünk, azaz megadjuk például a görbénk deriváltját (mozgásgörbéknél a

sebességet) a kezdő- és a végpontban.

A B-spline görbék egy fontos továbbfejlesztésében az i-edik vezérlőpont hatását a $B_i(t)$ B-spline bázisfüggvény aktuális paraméter melletti értéke és a vezérlőpont saját w_i súlytényezőjének szorzata adja meg. A görbe neve ***NURBS,*** amely a jelenlegi geometriai tervezőrendszerek egyik legfontosabb eszköze.

A szokásos mechanikai analógiánkban a NURBS görbénél egy vezérlőpontba $w_i B_i(t)$ súlyt teszünk, így a rendszer súlypontja:

$$\vec{r}(t) = \frac{\sum_{i=0}^{m} w_i B_i^{\mathrm{BS}}(t) \cdot \vec{r}_i}{\sum_{j=0}^{m} w_j B_j^{\mathrm{BS}}(t)} = \sum_{i=0}^{m} B_i^{\mathrm{NURBS}}(t) \cdot \vec{r}_i \ .$$

A B-spline és a NURBS bázisfüggvények közötti kapcsolat tehát a következő:

$$B_i^{\mathrm{NURBS}}(t) = \frac{w_i B_i^{\mathrm{BS}}(t)}{\sum_{j=0}^{m} w_j B_j^{\mathrm{BS}}(t)} \ .$$

Mivel a B-spline bázisfüggvények polinomok, a NURBS bázisfüggvények két polinom hányadosaként írhatók fel. A NURBS görbék a másodrendű görbéket (például kört, ellipszist stb.) közelítési hiba nélkül képesek leírni.

15.2.6. Felületmodellezés

A parametrikus felületek kétváltozós $\vec{r}(u, v)$ függvényekkel adhatók meg. A függvény közvetlen megadása helyett véges számú $\vec{r}_{ij}$ vezérlőpontot veszünk fel, amelyeket a bázisfüggvényekkel súlyozva kapjuk meg a felületet leíró függvényeket:

$$\vec{r}(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \vec{r}_{ij} \cdot B_{ij}(u, v) \ . \tag{15.7}$$

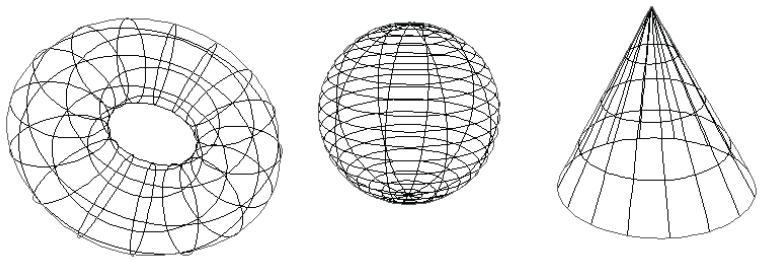
A bázisfüggvényektől továbbra is elvárjuk, hogy összegük minden paraméterre egységnyi legyen, azaz $\sum_{i=0}^{n} \sum_{j=0}^{m} B_{ij}(u, v) = 1$ mindenütt fennálljon. Ekkor ugyanis a súlypont analógia szerint most is képzelhetjük úgy, mintha a vezérlőpontokban u, v-től függő $B_{ij}(u, v)$ súlyok lennének, és a rendszer súlypontját tekintjük a felület ezen u, v párhoz tartozó pontjának.

A $B_{ij}(u, v)$ bázisfüggvények definíciójánál visszanyúlhatunk a görbéknél megismert eljárásokra. Rögzítsük gondolatban a v paraméter értéket. Az u paraméterértéket szabadon változtatva egy $\vec{r}_v(u)$ görbét kapunk, amely a felületen fut végig. Ezt a görbét a megismert görbetervezési eljárásokkal adhatjuk meg:

$$\vec{r}_v(u) = \sum_{i=0}^{n} B_i(u) \cdot \vec{r}_i \ , \tag{15.8}$$

ahol a $B_i(u)$ a kívánt görbe bázisfüggvénye.

Természetesen, ha más v értéket rögzítünk, akkor a felület más görbéjét kell kapnunk. Mivel egy adott típusú görbét a vezérlőpontok egyértelműen definiálnak, az $\vec{r}_i$ vezérlőpontoknak függeniük kell a rögzített v paramétertől. Ahogy a v változik, az $\vec{r}_i = \vec{r}_i(v)$ ugyancsak



15.7. ábra. Felületek izoparametrikus – azaz különböző u és v értékek rögzítésével kapott – vonalai.

egy görbén fut végig, amit érdemes ugyanazon görbetípussal az $\vec{r}_{i,0}, \vec{r}_{i,2}, \ldots, \vec{r}_{i,m}$ vezérlőpontok segítségével felvenni:

$$\vec{r}_i(v) = \sum_{j=0}^{m} B_j(v) \cdot \vec{r}_{ij} \ .$$

Ezt behelyettesítve a (15.8) egyenletbe, a felület paraméteres függvénye a következő lesz:

$$\vec{r}(u,v) = \vec{r}_v(u) = \sum_{i=0}^{n} B_i(u) \left(\sum_{j=0}^{m} B_j(v) \cdot \vec{r}_{ij} \right) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i(u) B_j(v) \cdot \vec{r}_{ij} \ .$$

A görbékkel összehasonlítva most a vezérlőpontok egy kétdimenziós rácsot alkotnak, a kétváltozós bázisfüggvényeket pedig úgy kapjuk, hogy a görbéknél megismert bázisfüggvények u-val és v-vel parametrizált változatait összeszorozzuk.

15.2.7. Testmodellezés buborékokkal

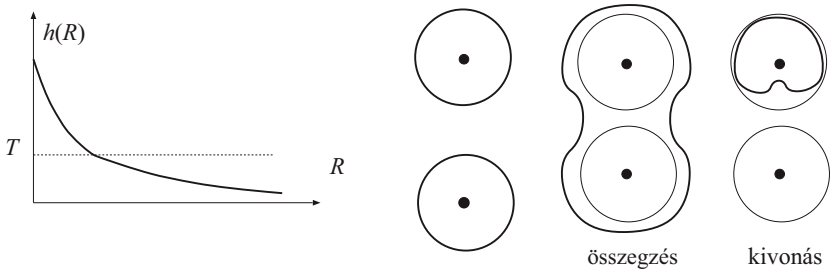
A szabad formájú, amorf testek létrehozását – a parametrikus görbékhez és felületekhez hasonlóan – a vezérlőpontok megadására vezetjük vissza. Rendeljünk minden $\vec{r}_i$ vezérlőponthoz egy $h(R_i)$ hatásfüggvényt, amely kifejezi a vezérlőpont hatását egy tőle $R_i = |\vec{r} - \vec{r}_i|$ távolságban lévő pontban. Összetett testnek azokat a pontokat tekintjük, ahol a hatások összege egy alkalmas T küszöbérték felett van (15.8. ábra):

$$f(\vec{r}) = \sum_{i=0}^{m} h_i(R_i) - T \geq 0, \qquad \text{ahol } R_i = |\vec{r} - \vec{r}_i| \ .$$

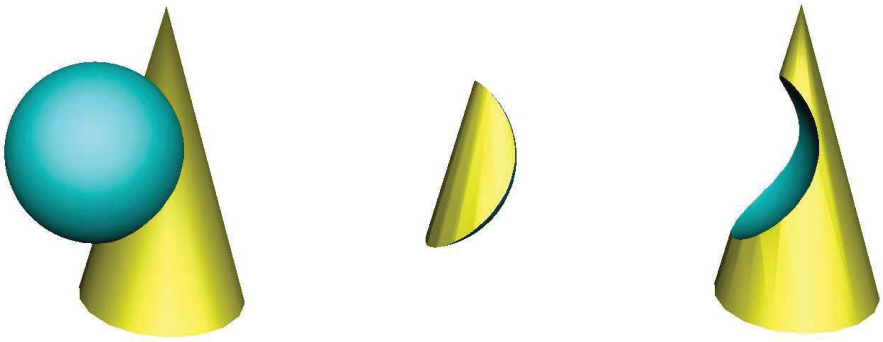
Egy hatásfüggvénnyel egy gömböt írhatunk le, a gömbök pedig cseppszerűen összeolvadnak (15.8. ábra). A pont hatását bármilyen csökkenő, a végtelenben nullához tartó folytonos függvénnyel kifejezhetjük. Például Blinn a $h_i(R) = a_i \cdot e^{-b_i R^2}$ hatásfüggvényeket javasolta a ***csepp*** módszerében.

15.2.8. Konstruktív tömörtest geometria

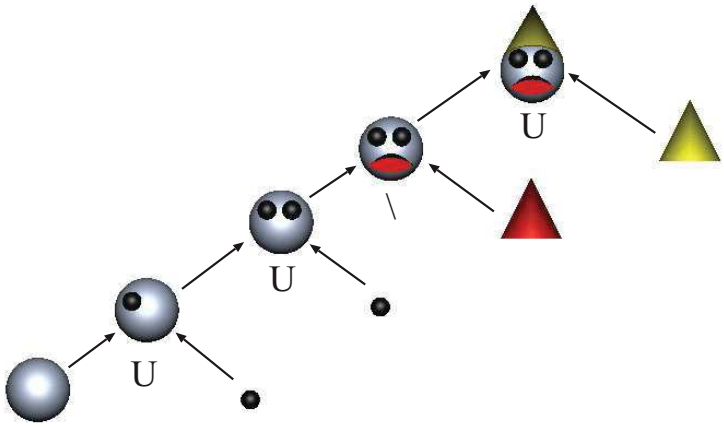
A testmodellezés másik irányzata, a ***konstruktív tömörtest geometria*** az összetett testeket primitív testekből halmazműveletek (egyesítés, metszet, különbség) ismételt alkalmazásával építi fel (15.9. és 19.10. ábra). A primitív testek általában a téglatestet, a gömböt, a gúlát,



15.8. ábra. A hatás a távolsággal csökken. Az azonos előjelű hatásfüggvények által leírt gömbök összeolvadnak, a különböző előjelűek csökkentik egymás hatását.



15.9. ábra. A konstruktív tömörtest geometria alapműveletei egy f implicit függvényű kúpra és egy g implicit függvényű gömbre: egyesítés ($\max(f,g)$), metszet ($\min(f,g)$) és különbség ($\min(f,-g)$).



15.10. ábra. Összetett objektum felépítése halmazműveletekkel. A gyökere az összetett objektumot, a levelei a primitíveket azonosítják. A többi csomópont a halmazműveleteket adja meg (U: egyesítés, \: különbség).

a kúpot, a félteret stb. foglalják magukban, amelyek implicit függvényei ismertek.

A halmazművelet eredményének implicit függvényét a résztvevő testek implicit függvényeiből kifejezhetjük:

- f és g metszete: $\min(f, g)$;
- f és g egyesítése: $\max(f, g)$.
- f komplemense: $-f$.
- f és g különbsége: $\min(f, -g)$.

Az implicit függvények segítségével két test közötti ***átmenet*** is könnyen kezelhető. Tegyük fel, hogy két testünk van, például egy kocka és egy gömb, amelyek implicit függvényei f_1 és f_2. Ebből egy olyan testet, amely t részben az első objektumhoz, $(1-t)$ részben pedig a második objektumhoz hasonlít, az

$$f^{morph}(x,y,z) = t \cdot f_1(x,y,z) + (1-t) \cdot f_2(x,y,z)$$

egyenlettel állíthatunk elő.

Gyakorlatok

15.2-1. Írjuk fel a tórusz paraméteres egyenletét.

15.2-2. Bizonyítsuk be, hogy a 4 vezérlőpontra illeszkedő, [0,0,0,0,1,1,1,1] csomóvektorú, negyedrendű B-spline egy Bézier-görbe.

15.2-3. Adjuk meg a lobogó zászló és az egy pontból induló hullámfelület paraméteres egyenleteit, és a normálvektoraikat is egy tetszőleges pontban.

15.2-4. Bizonyítsuk be, hogy a Bézier-görbe érinti az első két és utolsó két vezérlőpont közötti szakaszokat.

15.2-5. Vezessük le a másod-, harmad- és negyedrendű B-spline görbe bázisfüggvényeinek képletét.

15.2-6. Készítsünk algoritmust a Bézier-görbék és B-spline görbék ívhosszának meghatározására két paraméterérték között. Az ívhossz számítás alapján vezessünk végig egy pontot egyenletes sebességgel a görbén.

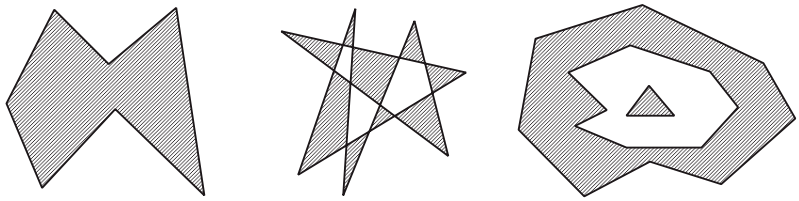
15.3. Geometriai feldolgozó és tesszellációs algoritmusok

A 15.2. alfejezetben megismerkedtünk azokkal a paraméteres és implicit függvényeket használó eljárásokkal, amelyekkel görbéket és felületeket írhatunk le. A képszintézis során különösen nagy jelentőségük van a szakaszoknak és a háromszögeknek, ezért ebben a fejezetben olyan eljárásokat ismertetünk, amelyek más reprezentációkat szakaszokkal vagy háromszögekkel közelítenek, illetve, amelyek szakaszokkal vagy háromszögekkel leírt modelleket alakítanak át. A végpontjaikban egymáshoz kapcsolódó szakaszok sorozatát ***töröttvonalnak,*** a háromszögek élekben illeszkedő gyűjteményét pedig ***háromszöghálónak*** nevezzük. A görbéket töröttvonallal közelítő eljárások neve ***vektorizáció,*** és kimenete egy, a töröttvonal csúcsait megadó pontsorozat. A felületeket háromszöghálókkal közelítő ***tesszellációs*** algoritmusok pedig háromszögek sorozatát állítják elő, ahol az egyes háromszögeket a csúcspontjaikkal adjuk meg. Gyakran a fényforrások hatásának számításához a csúcspontokban az eredeti felület normálvektorait is tároljuk, így végül egy háromszöghöz három

pont és három normálvektor tartozik. A háromszöghálókat feldolgozó eljárások még további topológiai információkat is felhasználnak, például azt, hogy egy csúcsra mely lapok és élek illeszkednek, és egy élben mely lapok találkoznak.

15.3.1. Sokszög és poliéder

15.2. definíció. *Egy **sokszög** vagy más néven **poligon** a síknak véges sok szakasszal határolt, korlátos, azaz egyenest nem tartalmazó része. A sokszöget a határoló szakaszokat tartalmazó zárt töröttvonalak felsorolásával definiáljuk. Egy töröttvonalat a csúcsaival adunk meg.*



15.11. ábra. A sokszögek fajtái. **(a)**: egyszerű; **(b)** nem egyszerű, egyszeresen összefüggő; **(c)** többszörösen összefüggő.

15.3. definíció. *Egy sokszög **egyszeresen összefüggő,** ha egyetlen zárt töröttvonal határolja (15.11. ábra).*

15.4. definíció. *Egy sokszög **egyszerű,** ha egyszeresen összefüggő, és a határoló töröttvonal nem metszi saját magát (15.11(a) ábra).*

Egy tetszőleges síkbeli pontról úgy dönthető el, hogy a sokszög belsejében van-e, hogy egy félegyenest indítunk a pontból, és megszámláljuk a sokszög éleivel keletkező metszéspontokat. Ha a metszéspontok száma páratlan, akkor a pont a sokszög belsejében, egyébként a külsejében van.

A háromdimenziós térben több, nem feltétlenül egy síkban lévő sokszögből sokszöghálókat építhetünk fel. Két sokszöget szomszédosnak mondunk, ha van közös élük.

15.5. definíció. *Egy **poliéder** egy olyan korlátos, azaz egyenest nem tartalmazó térrész, amelyet véges sok sokszög határol.*

A sokszöghöz hasonlóan, egy tetszőleges pontról úgy dönthető el, hogy a poliéderhez tartozik-e, hogy egy félegyenest indítunk a pontból, és megszámláljuk a poliéder lapjaival keletkező metszéspontokat. Ha a metszéspontok száma páratlan, akkor a pont a poliéder belsejében, egyébként a külsejében van.

15.3.2. Paraméteres görbék vektorizációja

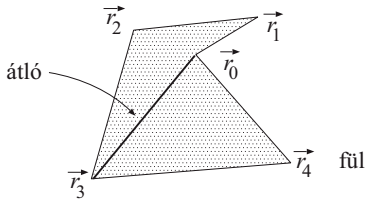
A paraméteres görbék a $[t_{\min}, t_{\max}]$ intervallumot képezik le a görbe pontjaira. A vektorizáció során a paraméterintervallumot osztjuk fel. A legegyszerűbb felbontás a t tartományt N

részre bontja fel, és az így kialakult $t_i = t_{\min} + (t_{\max} - t_{\min}) \cdot i/N$ $(i = 0, 1, \ldots, N)$ paraméterértékekből kapott $\vec{r}(t_i)$ pontokra illeszt töröttvonalat.

15.3.3. Egyszerű sokszögek háromszögekre bontása

A célként megfogalmazott háromszögsorozathoz az egyszerű sokszögek állnak a legközelebb, ezért először ezek háromszögesítésével foglalkozunk. Konvex sokszögekre a feladat egyszerű, egy tetszőleges csúcspontot kiválasztva, és azt az összes többivel összekötve, a felbontás lineáris időben elvégezhető. Konkáv sokszögeknél azonban ez az út nem járható, ugyanis előfordulhat, hogy a két csúcsot összekötő szakasz nem a sokszög belsejében fut, így ez a szakasz nem lehet valamelyik, a sokszöget felbontó háromszög oldala.

Kezdjük két alapvető definícióval:



15.12. ábra. A sokszög átlója és füle.

15.6. definíció. *Egy sokszög **átlója** egy, a sokszög két csúcsát összekötő olyan szakasz, amely teljes egészében a sokszög belsejében van (15.12. ábra).*

Az átló tulajdonság egy szakaszra úgy ellenőrizhető, ha azt az összes oldallal megpróbáljuk elmetszeni és megmutatjuk, hogy metszéspont csak a végpontokban lehetséges, valamint azt is, hogy az átlójelölt egy tetszőleges, a végpontoktól különböző pontja a sokszög belsejében van. Ez a tetszőleges pont lehet például a jelölt középpontja. Egy pontról a 15.4.1. pontban tárgyalt eljárással dönthető el, hogy egy sokszög belsejében van-e.

15.7. definíció. *A sokszög egy csúcsa **fül,** ha az adott csúcsot megelőző és követő csúcsokat összekötő szakasz a sokszög átlója (15.12. ábra).*

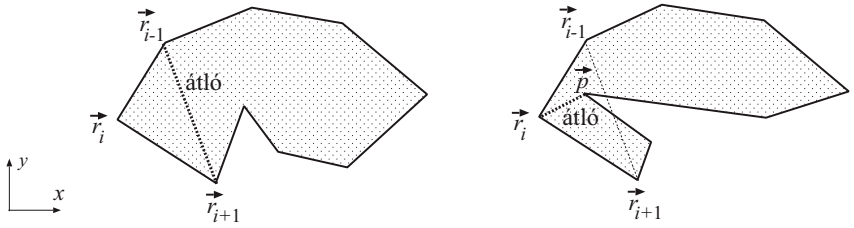
Nyilván csak azok a csúcsok lehetnek fülek, amelyekben a belső szög 180 foknál nem nagyobb. Az ilyen csúcsokat ***konvex csúcsoknak*** nevezzük, a nem konvexeket pedig ***konkáv csúcsoknak.***

Az egyszerű sokszögekre kimondhatjuk a következő alapvető tételeket:

15.8. tétel. *Egy egyszerű sokszögnek mindig van átlója.*

Bizonyítás. Legyen a háromszög legbaloldalibb (minimális x koordinátájú) csúcsa $\vec{r}_i$, a két szomszédos csúcs pedig $\vec{r}_{i-1}$, illetve $\vec{r}_{i+1}$ (15.13. ábra). A legbaloldalibb tulajdonság miatt az $\vec{r}_i$ konvex csúcs. Ha az $\vec{r}_i$ fül, akkor az $(\vec{r}_{i-1}, \vec{r}_{i+1})$ szakasz átló (15.13. ábra bal oldala), így az állítást erre az esetre beláttuk. Mivel az $\vec{r}_i$ konvex csúcs, csak akkor nem fül, ha az $\vec{r}_{i-1}$, $\vec{r}_i$, $\vec{r}_{i+1}$ háromszög tartalmaz legalább egy poligon csúcspontot (15.13. ábra jobb oldala). Válasszuk ki a tartalmazott csúcspontok közül az $\vec{r}_{i-1}, \vec{r}_{i+1}$ pontokra illeszkedő egyenestől

a legtávolabbit, és helyvektorát jelöljük $\vec{p}$-vel. Mivel $\vec{p}$-nél nincs az $(\vec{r}_{i-1}, \vec{r}_{i+1})$ egyenestől távolabbi csúcs, a $\vec{p}$ és $\vec{r}_i$ között nem futhat él, tehát a $(\vec{p}, \vec{r}_i)$ szakasz átló. ■



15.13. ábra. Az átló létezésének bizonyítása egyszerű sokszögre.

15.9. tétel. *Egy egyszerű sokszög az átlóival mindig háromszögekre bontható. Ha a sokszög csúcsainak száma n, akkor a keletkező háromszögek száma* $n - 2$.

Bizonyítás. A tétel igazolásához teljes indukciót használunk. Az állítás $n = 3$ esetre, azaz egyetlen háromszögre nyilván igaz. Tegyük fel, hogy az állítás minden m ($m = 3, \ldots, n-1$) csúcsú sokszögre igaz, és vegyünk egy n szöget. Az előző tétel szerint az n szögnek van átlója, tehát felbonthatjuk egy átlója mentén egy n_1 szögre és egy n_2 szögre, ahol $n_1, n_2 < n$, és $n_1 + n_2 = n + 2$, hiszen az átló végén lévő két csúcs mindkét sokszögben megjelenik. Az indukciós feltétel értelmében a két sokszöget külön-külön háromszögekre bonthatjuk, ami az eredeti sokszög háromszögekre bontását adja. A háromszögek száma pedig $n_1 - 2 + n_2 - 2 = n - 2$. ■

A sokszögek felbontási tételének bizonyítása konstruktív, így közvetlenül sugall egy felbontási algoritmust: vegyük sorra a lehetséges átlójelölteket, és egy tényleges átló mentén bontsuk fel a sokszöget, végül rekurzívan folytassuk az eljárást a két új sokszögre.

Ennek az algoritmusnak a futási ideje $\Theta(n^3)$ (az átlójelöltek száma ugyanis $\Theta(n^2)$, az átló ellenőrzésének ideje pedig $\Theta(n)$). A következőkben ezért egy másik, egyszerű algoritmust ismertetünk, amely egy konvex vagy konkáv $\vec{r}_0, \vec{r}_1, \ldots, \vec{r}_n$ sokszöget háromszögekre oszt fel.

A háromszögekre bontó ***fülvágó algoritmus*** füleket keres, és azokat levágja addig, amíg egyetlen háromszögre egyszerűsödik az eredeti sokszög. Az algoritmus az $\vec{r}_2$ csúcstól indul. Amikor egy csúcsot dolgozunk fel, először ellenőrizzük, hogy a megelőző csúcspont füle. Ha az nem fül, a következő csúcspontra lépünk. Ha a megelőző csúcs fül, akkor a két előző és az aktuális csúcsból egy háromszöget hozunk létre, és a megelőző csúcsot töröljük a sokszög csúcsai közül. Ha a törlés után az aktuális csúcsot megelőző csúcspont éppen a 0 indexű, akkor a következő csúcspontra lépünk.

Az algoritmus egy háromszöget vág le a sokszögből amíg talál fület, a befejeződését pedig az biztosítja, hogy minden egyszerű sokszögnek van füle. Az következő ***két fül tétel*** alábbi bizonyítása Joseph O'Rourke-től származik.

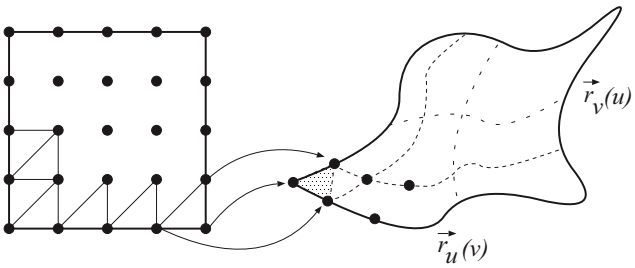
15.10. tétel. *Egy egyszerű, legalább négy csúcsú sokszögnek mindig van legalább két, nem szomszédos, így egymástól függetlenül levágható, füle.*

Bizonyítás. A 15.9. tétel értelmében minden egyszerű sokszög háromszögekre bontható úgy, hogy a keletkező háromszögek oldalai a sokszög élei vagy pedig átlói. A háromszögeket feleltessük meg egy gráf csomópontjainak, és két csomópont közé akkor vegyünk fel egy élt, ha a két háromszög egy sokszögátlón osztozik. A keletkező gráf összefüggő, és nincsenek benne körök, tehát fagráf, amelynek neve ***duális fa.*** A sokszög csúcsainak száma legalább négy, így a fagráf csomópontjainak száma legalább kettő. Minden, legalább két csomópontból álló fagráfnak legalább két levele[3] van. A leveleknek megfelelő háromszögek pedig fülek. ■

A két fül tétel miatt az algoritmusunk $O(n)$ lépésben mindig talál levágandó fület, amelynek eltávolításával a sokszög csúcsainak száma eggyel csökken, így az eljárás $O(n^2)$ lépésben befejeződik.

15.3.4. Paraméteres felületek tesszellációja

A paraméteres felületek a paramétertartomány egy $[u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}]$ téglalapját képezik le a felület pontjaira.



15.14. ábra. Paraméteres felületek tesszellációja.

A tesszelláció elvégzéséhez a paramétertéglalapot háromszögesítjük. A paraméter háromszögek csúcsaira alkalmazva a paraméteres egyenletet, éppen a felületet közelítő háromszöghálóhoz jutunk. A legegyszerűbb felbontás az u tartományt N részre, a v-t pedig M részre bontja fel, és az így kialakult

$$[u_i, v_j] = \left[u_{\min} + (u_{\max} - u_{\min})\frac{i}{N},\ v_{\min} + (v_{\max} - v_{\min})\frac{j}{M}\right]$$

párokból kapott pontok közül az $\vec{r}(u_i, v_j)$, $\vec{r}(u_{i+1}, v_j)$, $\vec{r}(u_i, v_{j+1})$ ponthármasokra, illetve az $\vec{r}(u_{i+1}, v_j)$, $\vec{r}(u_{i+1}, v_{j+1})$, $\vec{r}(u_i, v_{j+1})$ ponthármasokra háromszögeket illeszt.

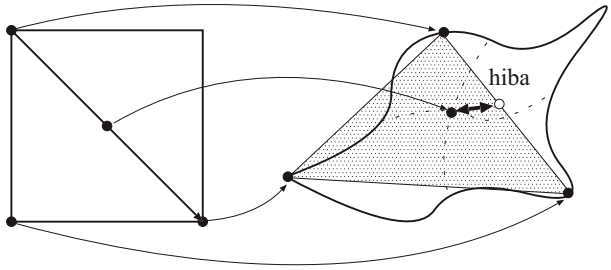
A tesszelláció lehet ***adaptív*** is, amely csak ott használ kis háromszögeket, ahol a felület gyors változása ezt indokolja. Induljunk ki a paraméter tartomány négyzetéből és bontsuk fel két háromszögre. A háromszögesítés pontosságának vizsgálatához a paramétertérben lévő háromszög élfelezőihez tartozó felületi pontokat összehasonlítjuk a közelítő három-

[3] A levél olyan csúcs, amelyhez pontosan 1 él kapcsolódik.

szög élfelező pontjaival, azaz képezzük a következő távolságot (15.15. ábra):

$$\left| \vec{r}\left(\frac{u_1 + u_2}{2}, \frac{v_1 + v_2}{2} \right) - \frac{\vec{r}(u_1, v_1) + \vec{r}(u_2, v_2)}{2} \right| ,$$

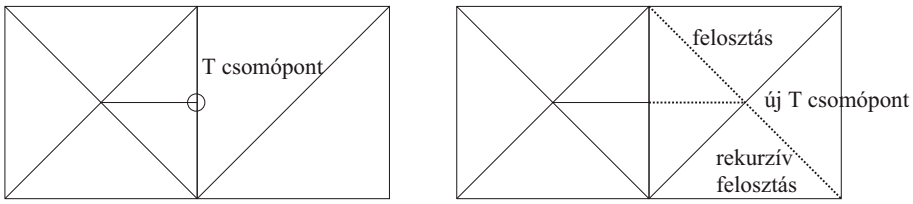
ahol (u_1, v_1) és (u_2, v_2) az él két végpontjának a paramétertérbeli koordinátái.



15.15. ábra. A tesszellációs hiba becslése.

Ha ez a távolság nagy, az arra utal, hogy a paraméteres felületet a háromszög rosszul közelíti, tehát azt fel kell bontani kisebb háromszögekre. A felbontás történhet úgy, hogy a háromszöget két részre vágjuk a legnagyobb hibával rendelkező felezőpont és a szemben lévő csúcs közötti súlyvonal segítségével, vagy pedig úgy, hogy a háromszöget négy részre vágjuk a három felezővonala segítségével. Az adaptív felbontás nem feltétlenül robusztus, ugyanis előfordulhat, hogy a felezőponton a hiba kicsi, de a háromszög mégsem közelíti jól a paraméteres felületet.

Az adaptív felbontásnál előfordulhat, hogy egy közös élre illeszkedő két háromszög közül az egyiket az élfelező ponton átmenő súlyvonallal felbontjuk, de a másikat nem, így a felbontás után az egyik oldalon lévő háromszög nem illeszkedik a másik oldalon lévő két másikhoz, azaz a felületünk kilyukad. Az ilyen problémás élfelező pontokat ***T csomópontnak*** nevezzük (15.16. ábra).

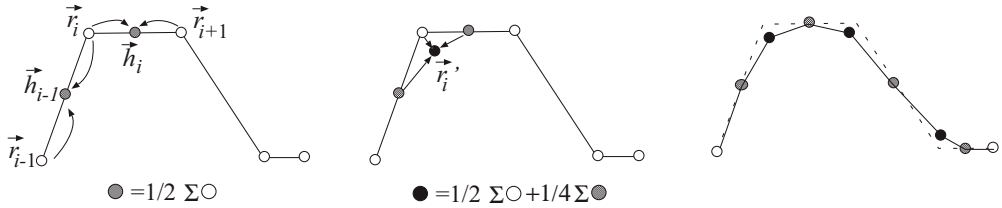


15.16. ábra. T csomópontok és kiküszöbölésük erőszakos felosztással.

Amennyiben a felosztást mindig csak arra az élre végezzük el, amelyik megsérti az előírt, csak az él tulajdonságain alapuló hibamértéket, T csomópontok nem jelenhetnek meg. Ha a felosztásban az él tulajdonságain kívül a háromszög egyéb tulajdonságai is szerepet játszanak, akkor viszont fennáll a veszélye a T csomópontok feltűnésének, amit úgy háríthatunk el, hogy ekkor rekurzívan arra az illeszkedő háromszögre is kierőszakoljuk a felosztást, amelyre a saját hibakritérium alapján nem tettük volna meg.

15.3.5. Töröttvonal és felület simítás, felosztott görbék és felületek

Ebben a pontban olyan algoritmusokat ismertetünk, amelyek a töröttvonal és sokszögháló modelleket simítják, azaz a töröttvonalat és sokszöghálót több vonalból, illetve lapból álló olyan változatokkal cserélik fel, amelyek kevésbé tűnnek szögletesnek.

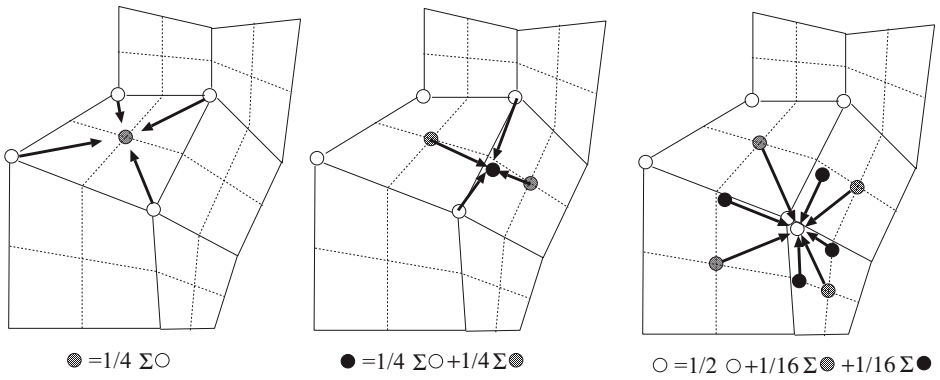


15.17. ábra. Felosztott görbe létrehozása: minden lépésben felezőpontokat veszünk fel, majd az eredeti csúcspontokat a szomszédos felezőpontok és a csúcspont súlyozott átlagára helyezzük át.

Tekintsük először az $\vec{r}_0, \ldots, \vec{r}_m$ töröttvonalat. Egy látszólag simább töröttvonalhoz jutunk a következő, a vezérlőpontokat megduplázó eljárással (15.17. ábra). Minden szakaszt megfelezünk és a felezőpontokban egy-egy új $\vec{h}_0, \ldots, \vec{h}_{m-1}$ vezérlőpontot veszünk fel. Bár már kétszer annyi vezérlőpontunk van, a görbénk éppen annyira szögletes, mint eredetileg volt. A régi vezérlőpontokat ezért úgy módosítjuk, hogy azok a régi helyük és a két oldalukon lévő felezőpontok közé kerüljenek, az alábbi súlyozással:

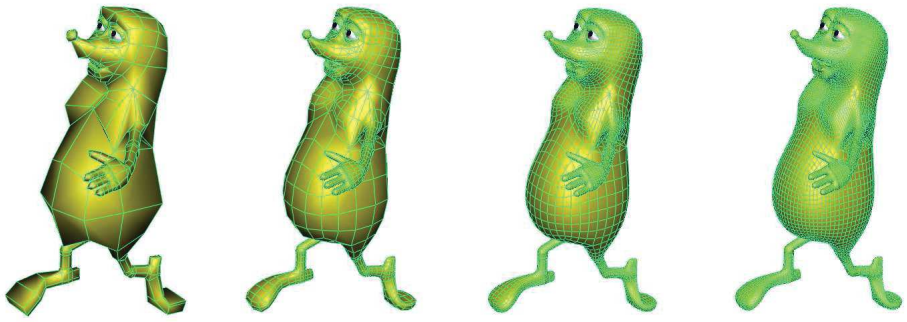
$$\vec{r}_i' = \frac{1}{2}\vec{r}_i + \frac{1}{4}\vec{h}_{i-1} + \frac{1}{4}\vec{h}_i = \frac{3}{4}\vec{r}_i + \frac{1}{8}\vec{r}_{i-1} + \frac{1}{8}\vec{r}_{i+1} .$$

Az új töröttvonal valóban sokkal simábbnak látszik. Ha még ez sem elégít ki bennünket, az eljárást tetszőleges mélységig ismételhetjük. Ha végtelen sokszor tennénk ezt, akkor éppen a B-spline-t állítanánk elő.



15.18. ábra. A Catmull–Clark-felosztás egy lépése. Először a lappontokat számítjuk, majd az élfelezők környezetében veszünk fel átlagolással egy pontot, végül pedig az eredeti csúcspontokat módosítjuk a környezet átlagának megfelelően.

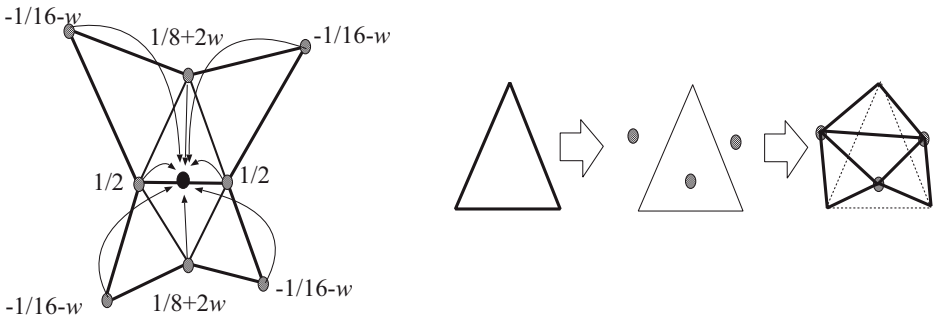
Az eljárás közvetlenül kiterjeszthető háromdimenziós hálókra, amelynek eredménye



15.19. ábra. Az eredeti háló, valamint egyszer, kétszer és háromszor felosztott változatai.

a ***Catmull–Clark felosztott felület.*** Induljunk ki egy háromdimenziós négyszöghálóból (15.18. ábra) (az algoritmus nemcsak négyszögeket képes felosztani, de a létrehozott lapok mindig négyszögek). Első lépésként minden él közepén felveszünk egy-egy ***élpontot*** mint az él két végpontjának az átlagát, és minden lap közepén egy-egy ***lappontot*** mint a négyszög négy csúcspontjának az átlagát. Az új élpontokat a lappontokkal összekötve ugyanazt a felületet négyszer annyi négyszöggel írtuk le. A második lépésben kezdődik a simítás, amikor az élpontokat módosítjuk az élhez illeszkedő lapok lappontjai alapján úgy, hogy az új élpont éppen a két lappont és az él két végén levő csúcspont átlaga legyen. Ugyanezt az új élpontot úgy is megkaphatjuk, hogy az élre illeszkedő két lap négy-négy eredeti sarokpontjának, valamint az él két végpontjának képezzük az átlagát (azaz az él végpontjait háromszor szerepeltetjük az átlagban). A simítás utolsó lépésében az eredeti csúcspontok új helyét súlyozott átlaggal határozzuk meg, amelyben az eredeti csúcspont 1/2 súlyt, az illeszkedő élek összesen 4 módosított élpontja és az illeszkedő lapok összesen 4 lappontja pedig 1/16 súlyt kap. Az eljárást addig ismételjük, amíg a felület simasága minden igényünket ki nem elégíti (15.19. ábra).

Ha a háló egyes éleinek és csúcsainak környezetét nem szeretnénk simítani, akkor a megőrzendő éleken túl lévő pontokat nem vonjuk be az átlagolási műveletekbe.



15.20. ábra. Az új élpont meghatározása és a háromszög pillangó felosztása.

A Catmull–Clark-felosztással kapott felület általában nem megy át az eredeti háló csúcspontjain. Ezt a hátrányt küszöböli ki a háromszöghálókon működő ***pillangó felosztás.*** A pillangó felosztás a háromszögek élfelező pontjainak közelébe egy-egy új élpontot helyez, majd az eredeti háromszögeket négy új háromszöggel váltja fel. Az új háromszögek csúcsai egyrészt az eredeti háromszög csúcsai, másrészt a módosított élfelező pontjai (15.20. ábra). Az élpontok kialakításában az élre illeszkedő háromszögek csúcspontjai és az ezen két háromszöggel közös élt birtokló még további négy háromszög vesz részt. Az élpontra ható háromszögek elrendezése egy pillangóra emlékeztet, ami magyarázza az eljárás elnevezését. Az élpont koordinátáit az él végpontjainak koordinátáiból számítjuk $1/2$-es súlyozással, az élre illeszkedő két háromszög harmadik csúcsaiból $1/8 + 2w$ súlyozással, valamint az élre illeszkedő két háromszöghöz tapadó négy háromszögnek az illeszkedő háromszögön kívül lévő csúcsaiból $-1/16-w$ súlyozással. A w a művelet paramétere, amellyel azt állíthatjuk be, hogy az eljárás mennyire görbítse meg a felületet az élek környezetében. A $w = -1/16$-os beállítás megtartja a háló szögletességét, a $w = 0$-t használó felosztás pedig erősen lekerekíti az eredeti éleket.

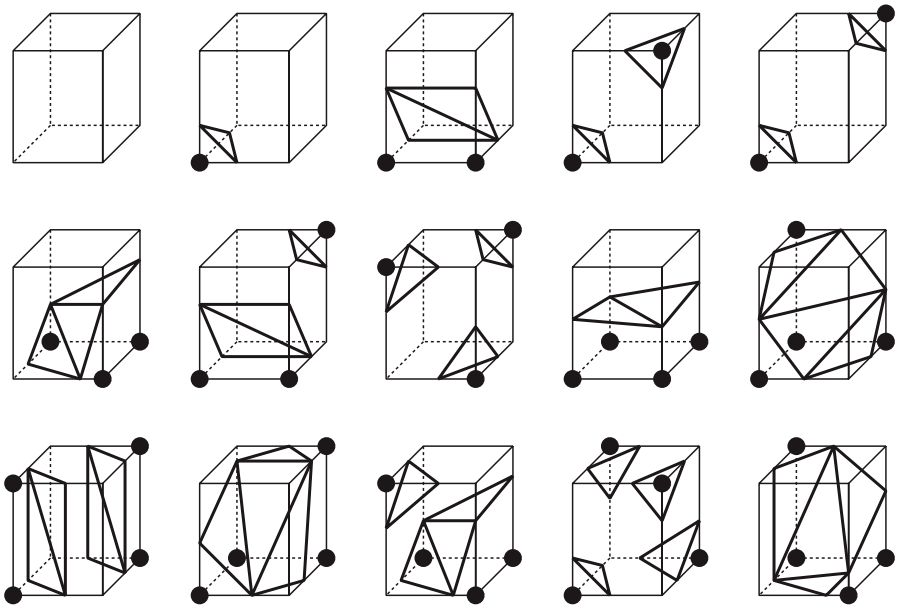
15.3.6. Implicit felületek tesszellációja

Egy implicit egyenlettel leírt test felületének háromszöghálós közelítését úgy állíthatjuk elő, hogy elegendően sűrűn olyan pontokat keresünk, amelyekre az $f(x,y,z) \approx 0$ teljesül, és a közeli pontokat összekötve háromszöghálót építünk fel.

Először az f függvényt a háromdimenziós koordinátarendszer rácspontjaiban kiértékeljük és az eredményt egy háromdimenziós tömbben, úgynevezett ***voxeltömbben*** tároljuk. A továbbiakban két rácspontot szomszédosnak nevezünk, ha két koordinátájuk páronként megegyezik, a harmadik koordináták különbsége pedig éppen az adott tengely menti rácsállandó. A rács pontjaiban ismerjük a függvény pontos értékét, a szomszédos rácspontok közötti változást pedig általában lineárisnak tekintjük. Az árnyaláshoz szükséges normálvektorokat a mintapontokban az f függvény gradienseként számítjuk (15.4. egyenlet), amelyet a rácspontok között ugyancsak lineárisan interpolálunk.

A voxeltömb alkalmazása azt jelenti, hogy az eredeti f függvény helyett a továbbiakban annak egy voxelenként ***tri-lineáris*** közelítésével dolgozunk (a tri-lineáris jelző arra utal, hogy a közelítő függvényben bármely két koordinátaváltozó rögzítésével a harmadik koordinátában a függvény lineáris). A lineáris közelítés miatt egy – két szomszédos rácspont közötti – él legfeljebb egyszer metszheti a közelítő felületet, hiszen a lineáris függvénynek legfeljebb egyetlen gyöke lehet. A rácspontokat olyan sűrűn kell felvenni, hogy a tri-lineáris közelítés során ne veszítsünk el gyököket, azaz a felület topológiája ne változzon.

A felületet háromszöghálóval közelítő módszer neve ***masírozó kockák algoritmus.*** Az algoritmus a mintavételezett érték előjele alapján minden mintavételezési pontra eldönti, hogy az a test belsejében vagy azon kívül van-e. Ha két szomszédos mintavételezési pont eltérő típusú, akkor közöttük felületnek kell lennie. A határ helyét és az itt érvényes normálvektort a szomszédos mintavételezési pontok közötti élen az értékek alapján végzett lineáris interpolációval határozhatjuk meg. Ha az egyik mintavételezési pont helyvektora $\vec{r}_1$, a szomszédos mintavételi ponté pedig $\vec{r}_2$, és az f implicit függvény a két pontban eltérő előjelű, akkor a tri-lineáris felület és az $(\vec{r}_1, \vec{r}_2)$ szakasz $\vec{r}_i$ metszéspontja, valamint a felület $\vec{n}_i$ normálvektora:



15.21. ábra. Egy voxelenkénti tri-lineáris implicit függvényű felület és egy voxel lehetséges metszetei. A 256-ból ez a 15 topológiailag különböző eset választható ki, amelyekből a többiek forgatással előállíthatók. Az ábrán az azonos előjelű mintavételezett értékeket körrel jelöltük.

$$\vec{r}_i = \vec{r}_1 \cdot \frac{f(\vec{r}_2)}{f(\vec{r}_2) - f(\vec{r}_1)} + \vec{r}_2 \cdot \frac{f(\vec{r}_1)}{f(\vec{r}_2) - f(\vec{r}_1)} ,$$

$$\vec{n}_i = \mathrm{grad} f(\vec{r}_1) \cdot \frac{f(\vec{r}_2)}{f(\vec{r}_2) - f(\vec{r}_1)} + \mathrm{grad} f(\vec{r}_2) \cdot \frac{f(\vec{r}_1)}{f(\vec{r}_2) - f(\vec{r}_1)} .$$

Végül az éleken kijelölt pontokra háromszögeket illesztünk, amelyekből összeáll a közelítő felület. A háromszögillesztéshez figyelembe kell venni, hogy a tri-lineáris felület a szomszédos mintavételezési pontokra illeszkedő kocka éleinek mindegyikét legfeljebb egyszer metszheti. Metszéspont akkor keletkezik, ha az él két végén lévő mintavételezési pontban a függvényérték eltérő előjelű. A kocka 8 csúcsán érvényes előjelek alapján 256 eset lehetséges, amiből végül 15 topológiailag különböző konfiguráció különíthető el (15.21. ábra).

Az algoritmus sorra veszi az egyes voxeleket és megvizsgálja a csúcspontok előjeleit. Rendeljünk a negatív előjelhez 0 kódbitet, a nem negatívhoz 1-et. A 8 kódbit kombinációja egy 0–255 tartományba eső kódszónak tekinthető, amely éppen az aktuális metszési esetet azonosítja. A 0 kódszavú esetben az összes sarokpont a testen kívül van, így a felület a voxelt nem metszheti. Hasonlóan, a 255 kódszavú esetben minden sarokpont a test belsejében található, ezért a felület ekkor sem mehet át a voxelen. A többi kódhoz pedig egy táblázatot építhetünk fel, amely leírja, hogy az adott konfiguráció esetén mely kockaélek végpontjai eltérő előjelűek, ezért metszéspont lesz rajtuk, valamint azt is, hogy a metszéspontokra miként kell háromszöghálót illeszteni.

Gyakorlatok

15.3-1. Igazoljuk a két fül tételt teljes indukcióval.
15.3-2. Készítsünk adaptív görbetesszellációs algoritmust.
15.3-3. Bizonyítsuk be, hogy a Catmull–Clark felosztási módszer a B-spline görbéhez, illetve felülethez konvergál.
15.3-4. Készítsünk egy táblázatot a masírozó kockák algoritmus számára, amely minden kódszóhoz megadja a keletkező háromszögek számát, és azt, hogy a háromszögek csúcspontjai mely kockaélekre illeszkednek.
15.3-5. Készítsünk olyan masírozó kocka algoritmust, amely nem igényli az implicit függvény gradiensét a mintavételi pontokban, hanem azt is az implicit függvény mintáival becsli.

15.4. Tartalmazási algoritmusok

A modellek feldolgozása során gyakran kell megválaszolnunk azt a kérdést, hogy egy alakzat tartalmazza-e egy másik alakzat valamely részét. Ha csak igen/nem válaszra vagyunk kíváncsiak, akkor ***tartalmazás vizsgálatról*** beszélünk. Ha elő is kell állítani a tartalmazott alakzat azon részét, amely a másik alakzat belsejében van, akkor az algoritmuscsalád neve ***vágás.***

A tartalmazás vizsgálatot gyakran diszkrét idejű ***ütközésfelismerésnek*** is nevezzük. Az ütközéseket ugyanis közelítőleg vizsgálhatjuk úgy is, hogy csak diszkrét időpillanatokban nézünk rá a virtuális világ elemeire, és az ütközésekre utólag abból következtetünk, hogy megvizsgáljuk, tartalmazzák-e az alakzatok más alakzatok részeit. A diszkrét idejű ütközésfelismerés hibázhat. Ha az objektumok sebessége a mintavételezési időhöz képest nagy, akkor előfordulhat, hogy nem veszünk észre ütközéseket. Az ütközési feladat robusztus és pontos megoldásához folytonos idejű ütközésfelismerő eljárások szükségesek, amelyek az ütközés időpontját is kiszámítják. A folytonos idejű ütközésfelismerést a sugárkövetésre (15.6. pont) építhetjük, így ebben a fejezetben csak a pont tartalmazással, a poliéderek közötti diszkrét idejű ütközésfelismeréssel, és végül néhány egyszerű alakzat vágásával foglalkozunk.

15.4.1. Pont tartalmazásának vizsgálata

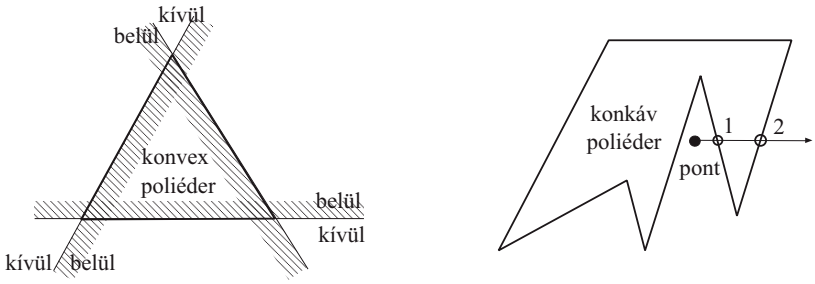
Az f implicit függvényű test azon (x, y, z) pontokat tartalmazza, amelyekre az $f(x, y, z) \geq 0$ egyenlőtlenség teljesül, így az adott pont koordinátáit az implicit függvénybe helyettesítve, az eredmény előjele alapján dönthetünk a tartalmazásról.

Féltér
A féltér pontjait a (15.1) egyenlet alapján az

$$(\vec{r} - \vec{r}_0) \cdot \vec{n} \geq 0, \qquad n_x \cdot x + n_y \cdot y + n_z \cdot z + d \geq 0 \tag{15.9}$$

egyenlőtlenséggel adhatjuk meg, ahol a határoló sík normálvektora „befelé” mutat.

Konvex poliéder
Bármely konvex poliéder előállítható a lapjaira illeszkedő síkok által határolt félterek metszeteként (15.22. ábra bal oldala). Minden lap síkja tehát a teret két részre bontja, egy belső részre, amelyikben maga a poliéder található, és egy külső tartományra. Vessük össze a pontot a poliéder lapjaival, pontosabban azok síkjaival. Ha a pontunk minden sík tekintetében a belső részben van, akkor a pont a poliéderen belül van. Ha viszont valamely sík esetén a külső tartományban van, akkor a pont nem lehet a poliéder belsejében.



15.22. ábra. Poliéder-pont tartalmazás vizsgálat. Konvex poliéder akkor tartalmaz egy pontot, ha a pont minden lapsíkjának ugyanazon az oldalán van, mint maga a test. Konkáv poliéderre egy félegyenest indítunk a pontból és megszámláljuk a metszéspontokat. Páratlan számú metszés esetén a pont belül van, páros számú metszéskor pedig kívül.

Konkáv poliéder
A 15.22. ábra jobb oldalán látható módon indítsunk egy félegyenest a vizsgált pontból a végtelen felé, és próbáljuk elmetszeni a poliéder lapjait a félegyenessel (a metszéspontok számításához a 15.6. szakaszban a sugárkövetéshez kidolgozott eljárások használhatók). Ha páratlan számú metszéspontot számolunk össze, akkor a poliéder belsejében, egyébként pedig azon kívül van a pontunk. A numerikus pontatlanságok miatt a lapok találkozásánál gondot jelenthet annak eldöntése, hogy félegyenesünk itt hány lapot is metszett egyszerre. Ha ilyen helyzetbe kerülünk, akkor a legegyszerűbb egy olyan új félegyenest választani, amely elkerüli a lapok találkozását.

Sokszög
Természetesen a konkáv poliédereknél megismert eljárás a síkban is használható annak eldöntéséhez, hogy egy tetszőleges sokszög tartalmaz-e egy adott, ugyanebben a síkban lévő pontot. A síkon egy félegyenest indítunk a végtelen felé, és megszámláljuk a sokszög éleivel keletkező metszéspontokat. Ha a metszéspontok száma páratlan, akkor a pont a sokszög belsejében, egyébként a külsejében van.

Ezen kívül a konvex lapoknál ellenőrizhetjük, hogy a pontból a lap éleinek a látószögét összegezve 360 fokot kapunk-e, vagy megvizsgálhatjuk, hogy minden élre a pont ugyanazon az oldalon van-e, mint a lap többi csúcspontja. Az algoritmus működését részleteiben egy speciális esetre, a háromszögre tárgyaljuk.

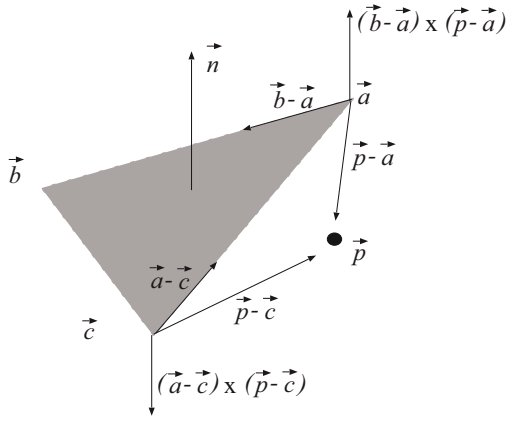
Háromszög

Tekintsünk egy háromszöget $\vec{a}, \vec{b}$ és $\vec{c}$ helyvektorú csúcsokkal, és egy, a háromszög síkjában lévő $\vec{p}$ pontot. A pont akkor van a háromszögön belül, ha a háromszög mind a három oldalegyeneséhez viszonyítva ugyanazon az oldalon van, mint a harmadik csúcs. A $(\vec{b}-\vec{a})\times(\vec{p}-\vec{a})$ vektoriális szorzat az $\vec{ab}$ egyenes két oldalán lévő $\vec{p}$ pontokra ellentétes irányú, így ezen vektor irányát felhasználhatjuk a pontok osztályozására (amennyiben a $\vec{p}$ pont éppen az $\vec{ab}$ egyenesen lenne, a vektoriális szorzat eredménye zérus). A $(\vec{b}-\vec{a})\times(\vec{p}-\vec{a})$ vektor irányát tehát az $\vec{n}=(\vec{b}-\vec{a})\times(\vec{c}-\vec{a})$ vektor irányával kell összevetni, amelyben a vizsgált $\vec{p}$ pontot a háromszög harmadik $\vec{c}$ csúcsával cseréltük fel. Vegyük észre, hogy ez az $\vec{n}$ vektor éppen a háromszög síkjának normálvektora (15.23. ábra).

Két vektorról például úgy állapíthatjuk meg, hogy azonos irányúak (bezárt szögük nulla), vagy ellentétesek (bezárt szögük 180 fok), hogy képezzük a skaláris szorzatukat, és megvizsgáljuk az eredmény előjelét. Azonos irányú vektorok skaláris szorzata pozitív, az ellentétes irányúaké negatív. Tehát, ha a $((\vec{b}-\vec{a})\times(\vec{p}-\vec{a}))\cdot\vec{n}$ skaláris szorzat pozitív, akkor a $\vec{p}$ az $\vec{ab}$ egyenesnek ugyanazon az oldalán van, mint a $\vec{c}$, ha negatív, akkor az ellentétes oldalon, ha pedig zérus, akkor a $\vec{p}$ az $\vec{ab}$ egyenesre illeszkedik. A vizsgálatot mindhárom oldalegyenesre el kell végezni, és a $\vec{p}$ pont akkor lesz a háromszög belsejében, ha mindhárom alábbi feltétel teljesül:

$$\begin{aligned}((\vec{b}-\vec{a})\times(\vec{p}-\vec{a}))\cdot\vec{n} &\geq 0\,,\\ ((\vec{c}-\vec{b})\times(\vec{p}-\vec{b}))\cdot\vec{n} &\geq 0\,,\\ ((\vec{a}-\vec{c})\times(\vec{p}-\vec{c}))\cdot\vec{n} &\geq 0\,.\end{aligned} \tag{15.10}$$

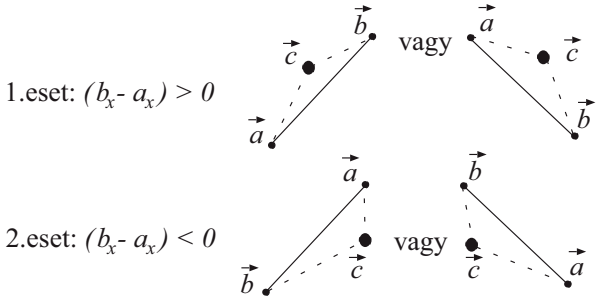
A vizsgálat robusztus, azaz akkor is helyes eredményt ad, ha a numerikus pontatlanságok miatt a $\vec{p}$ pont nincs pontosan a háromszög síkján, csak a síkra merőleges háromszög alapú hasábban.



15.23. ábra. Háromszög-pont tartalmazás. Az ábra azt az esetet illusztrálja, amikor a síkon levő $\vec{p}$ pont az $\vec{ab}$ és $\vec{bc}$ egyenesektől balra, a $\vec{ca}$ egyenestől pedig jobbra helyezkedik el, azaz nincs bent a háromszög belsejében.

Az egyenlőtlenségrendszer kiértékelését gyorsíthatjuk, ha a háromdimenziós tér helyett annak kétdimenziós vetületén dolgozunk. Vetítsük le a vizsgált $\vec{p}$ pontot, és vele együtt a háromszöget valamelyik koordinátasíkra, és ezen a síkon végezzük el a háromszög három ol-

dalára a tartalmazás vizsgálatot. A koordinátasík kiválasztásakor ügyelnünk kell arra, hogy a háromszög vetülete a numerikus pontosság érdekében a lehető legnagyobb legyen és semmi esetre se zsugorodjon egy szakasszá. Ha a normálvektor három Descartes-koordinátája közül az n_z a legnagyobb abszolút értékű, akkor a legnagyobb vetület az xy síkon keletkezik. A következőkben csak ezzel az esettel foglalkozunk. Ha n_x vagy n_y lenne maximális abszolút értékű, akkor az yz, illetve az xz síkon a vizsgálat hasonlóan elvégezhető.



15.24. ábra. Háromszög-pont tartalmazás vizsgálata az xy vetületen. A $\vec{c}$ harmadik csúcs az $\vec{ab}$ bal vagy jobb oldalán helyezkedhet el, amit a csúcspontok sorrendjének felcserélésével mindig a baloldali esetre vezetünk vissza.

Először átalakítjuk a csúcsok sorrendjét úgy, hogy $\vec{a}$-ból $\vec{b}$-be haladva a $\vec{c}$ pont mindig a bal oldalon helyezkedjen el. Ehhez először vizsgáljuk meg az $\vec{ab}$ egyenes egyenletét:

$$\frac{b_y - a_y}{b_x - a_x} \cdot (x - b_x) + b_y = y \ .$$

A 15.24. ábra alapján a $\vec{c}$ pont akkor van az egyenes bal oldalán, ha $x = c_x$-nél c_y az egyenes felett van:

$$\frac{b_y - a_y}{b_x - a_x} \cdot (c_x - b_x) + b_y < c_y \ .$$

Mindkét oldalt $(b_x - a_x)$-szel szorozva:

$$(b_y - a_y) \cdot (c_x - b_x) < (c_y - b_y) \cdot (b_x - a_x) \ .$$

A második esetben a meredekség nevezője negatív. A $\vec{c}$ pont akkor van az egyenes bal oldalán, ha $x = c_x$-nél c_y az egyenes alatt van:

$$\frac{b_y - a_y}{b_x - a_x} \cdot (c_x - b_x) + b_y > c_y \ .$$

A negatív $(b_x - a_x)$ nevezővel való szorzás miatt a relációs jel megfordul:

$$(b_y - a_y) \cdot (c_x - b_x) < (c_y - b_y) \cdot (b_x - a_x) \ ,$$

azaz mindkét esetben ugyanazt a feltételt kaptuk. Ha ez a feltétel nem teljesül, akkor $\vec{c}$ nem az $\vec{ab}$ egyenes bal oldalán, hanem a jobb oldalán helyezkedik el. Ez pedig azt jelenti, hogy $\vec{c}$ a $\vec{ba}$ egyenes bal oldalán található, tehát az $\vec{a}$ és $\vec{b}$ sorrendjének cseréjével biztosítható, hogy $\vec{c}$ az $\vec{ab}$ egyenes bal oldalán tartózkodjon. Fontos észrevenni, hogy ebből következik az is,

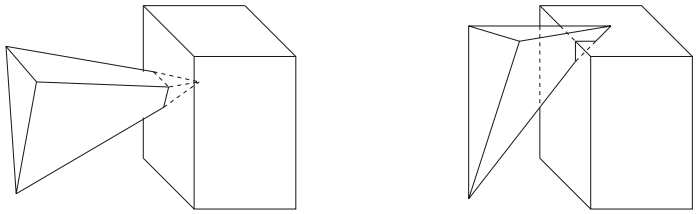
hogy az $\vec{a}$ a $\vec{bc}$ egyenes, valamint a $\vec{b}$ a $\vec{ca}$ egyenes bal oldalán helyezkedik el.

Az algoritmus második lépésében pedig azt ellenőrizzük, hogy a $\vec{p}$ pont mindhárom oldalra a bal oldalon van-e, mert ekkor a háromszög tartalmazza a pontot, egyébként pedig nem:

$$\begin{aligned}(b_y - a_y)\cdot(p_x - b_x) &\le (p_y - b_y)\cdot(b_x - a_x)\ ,\\ (c_y - b_y)\cdot(p_x - c_x) &\le (p_y - c_y)\cdot(c_x - b_x)\ ,\\ (a_y - c_y)\cdot(p_x - a_x) &\le (p_y - a_y)\cdot(a_x - c_x)\ .\end{aligned} \tag{15.11}$$

15.4.2. Poliéder-poliéder ütközésvizsgálat

Két poliéder ütközhet egymással úgy, hogy az egyikük egy csúcsa a másik egy lapjával találkozik, és ha semmi sem állítja meg, akkor a csúcs a másik test belsejébe hatol (15.25. ábra bal oldala). Ez az eset a korábban tárgyalt tartalmazás vizsgálattal felismerhető. Először az első poliéder összes csúcsára ellenőrizzük, hogy a másik poliéder tartalmazza-e, majd a két poliéder szerepét felcserélve vizsgáljuk, hogy a második csúcsai az első poliéder belsejében vannak-e.



15.25. ábra. Poliéder-poliéder ütközésvizsgálat. Az ütközési esetek csupán egy részét ismerhetjük fel az egyik test csúcsainak a másik testtel való összevetésével. Ütközés keletkezhet úgy is, hogy csak az élek hatolnak a másik testbe, de a csúcsok nem.

A csúccsal történő ütközésen kívül előfordulhat, hogy két poliéder élei a másikba hatolnak anélkül, hogy csúcsaik a másik belsejébe kerülnének (15.25. ábra jobb oldala). Ennek felismeréséhez az egyik poliéder összes élét össze kell vetni a másik poliéder összes lapjával. Egy él és lap tekintetében a (15.9) egyenlőtlenség felhasználásával először ellenőrizzük, hogy az él két végpontja a lap síkjának két ellentétes oldalán van-e. Ha igen, akkor kiszámítjuk az él és a lap síkjának a metszéspontját, végül pedig eldöntjük, hogy a metszéspont a lapon belül van-e.

Vegyük észre, hogy az él behatolás és egy tetszőleges pont tartalmazásának ellenőrzése együttesen magában foglalja a csúcs behatolás esetét is, tehát a csúcsok egyenkénti vizsgálata szükségtelennek látszik. Azonban csúcs behatolás nélküli él behatolás ritkábban fordul elő, ráadásul a csúcs behatolását kevesebb számítással is felismerhetjük, így érdemes először mégis a csúcs behatolást vizsgálni.

A poliéderek ütközésvizsgálata során az egyik poliéder összes élét a másik poliéder összes lapjával össze kell vetni, amely négyzetes idejű algoritmushoz vezet. Szerencsére a módszer a befoglaló keretek (15.6.2. pont) alkalmazásával jelentősen gyorsítható. Keressünk minden objektumhoz egy olyan egyszerű alakzatot, amely tartalmazza azt. Különösen népszerűek a befoglaló gömbök, illetve téglatestek. Ha a befoglaló alakzatok nem

találkoznak, akkor nyilván a befoglalt objektumok sem ütközhetnek. Amennyiben a befoglaló alakzatok egymásba hatolnak, akkor folytatni kell a vizsgálatot. Az egyik objektumot összevetjük a másik befoglaló alakzatával, és ha itt is ütközés mutatkozik, akkor magával az objektummal. Remélhetőleg ezen utóbbi eset nagyon ritkán fordul elő, és az ütközésvizsgálatok döntő részét a befoglaló alakzatokkal gyorsan el lehet intézni.

15.4.3. Vágási algoritmusok

A *vágás* egy vágó és egy vágandó alakzatot használ, és a vágandóból eltávolítja az összes olyan részt, amely a vágó külsejében van. A vágás megváltoztathatja a vágandó alakzat jellegét, így nehézséget okozhat, hogy a vágás után már nem lehet leírni olyan típusú függvénnyel, mint a vágás előtt. Ezért általában csak olyan vágó és vágandó alakzatokat engedünk meg, ahol a vágandó jellege (azaz a függvény típusa) a vágás után sem módosul. A továbbiakban feltételezzük, hogy a vágó alakzat féltér, általános, illetve speciális tulajdonságú poliéder, a vágandó pedig pont, szakasz, illetve sokszög.

Ha a vágandó alakzat egy pont, akkor a tartalmazást ellenőrizzük az előző pont algoritmusaival, és annak eredményétől függően a pontot eltávolítjuk vagy megtartjuk.

Szakaszok vágása féltérre

Tekintsük az $\vec{r}_1$ és $\vec{r}_2$ végpontú $\vec{r}(t) = \vec{r}_1 \cdot (1-t) + \vec{r}_2 \cdot t$, $(t \in [0,1])$ paraméteres egyenletű szakaszt és a (15.1) egyenletből származtatott

$$(\vec{r} - \vec{r}_0) \cdot \vec{n} \geq 0, \qquad n_x \cdot x + n_y \cdot y + n_z \cdot z + d \geq 0$$

egyenlőtlenséggel adott félteret.

Három esetet kell megkülönböztetni:

1. Ha a szakasz mindkét végpontja a féltér belsejében van, akkor a teljes szakasz belső pontokból áll, így megtartjuk.
2. Ha a szakasz mindkét végpontja a féltér külsejében van, akkor a szakasz minden pontja külső pont, így a vágás a teljes szakaszt eltávolítja.
3. Ha a szakasz egyik végpontja külső pont, a másik végpontja belső pont, akkor ki kell számítani a szakasz és a féltér határoló síkjának metszéspontját, és a külső végpontot fel kell cserélni a metszésponttal. A metszéspontot megkaphatjuk, ha a szakasz egyenletét a féltér határoló síkjának egyenletébe helyettesítjük, és az egyenletet az ismeretlen paraméterre megoldjuk:

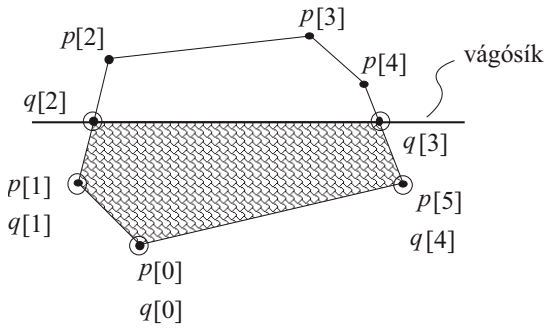
$$(\vec{r}_1 \cdot (1-t_i) + \vec{r}_2 \cdot t_i - \vec{r}_0) \cdot \vec{n} = 0, \quad \Longrightarrow \quad t_i = \frac{(\vec{r}_0 - \vec{r}_1) \cdot \vec{n}}{(\vec{r}_2 - \vec{r}_1) \cdot \vec{n}} .$$

A t_i paramétert a szakasz egyenletébe visszahelyettesítve a metszéspont koordinátáit is előállíthatjuk.

Sokszögek vágása féltérre

A vágás során egyrészt az egyes csúcspontokat kell megvizsgálni, hogy azok belső pontok-e vagy sem. Ha egy csúcspont belső pont, akkor a vágott sokszögnek is egyben csúcspontja. Ha viszont a csúcspont külső pont, nyugodtan eldobhatjuk. Másrészt vegyük észre, hogy

az eredeti sokszög csúcsain kívül a vágott sokszögnek lehetnek új csúcspontjai is, amelyek az élek és a féltér határoló síkjának a metszéspontjai. Ilyen metszéspont akkor keletkezhet, ha két egymást követő csúcs közül az egyik belső, míg a másik külső pont. A csúcsok egyenkénti vizsgálata mellett tehát arra is figyelni kell, hogy a következő pont a határoló síkhoz viszonyítva ugyanazon az oldalon van-e (15.26. ábra).



15.26. ábra. A $\vec{p}[0], \ldots, \vec{p}[5]$ sokszög vágása, amelynek eredménye a $\vec{q}[0], \ldots, \vec{q}[4]$ sokszög. Az eredménysokszög csúcsai az eredeti sokszög belső csúcsai és az éleknek a vágósíkkal képzett metszéspontjai.

Tegyük fel, hogy az eredeti egyszeresen összefüggő sokszögünk csúcsai a $\mathbf{p} = \langle \vec{p}[0], \ldots, \vec{p}[n-1] \rangle$ tömbben érkeznek, a vágott sokszög csúcsait pedig a $\mathbf{q} = \langle \vec{q}[0], \ldots, \vec{q}[m-1] \rangle$ tömbbe kell elhelyezni. A vágott sokszög csúcsainak számát az m változóban tároljuk. A megvalósítás során kellemetlenséget okoz, hogy általában az i-edik csúcsot követő csúcs az $(i+1)$-edik, kivéve az utolsó, az $(n-1)$-edik csúcsot, amelyet a 0-adik követ. Ezt a kellemetlenséget elháríthatjuk, ha a $\mathbf{p}$ tömböt kiegészítjük még egy ($\vec{p}[n] = \vec{p}[0]$) elemmel, amely még egyszer tárolja a 0-adik elemet.

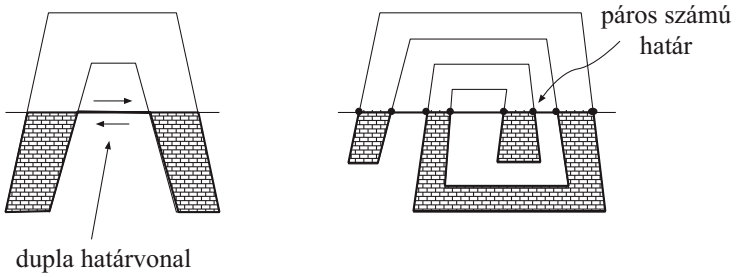
Ezek alapján a ***Sutherland–Hodgeman-poligonvágás:***

SUTHERLAND–HODGEMAN-POLIGONVÁGÁS($\mathbf{p}$)

```
1  m ← 0
2  for i ← 0 to n − 1
3      do if p⃗[i] belső pont
4           then q⃗[m] ← p⃗[i]                ▷ Az i-edik csúcs része a vágott poligonnak.
5                m ← m + 1
6                if p⃗[i + 1] külső pont
7                   then q⃗[m] ← METSZÉS-VÁGÓSÍKKAL(p⃗[i], p⃗[i + 1])
8                        m ← m + 1
9           else if p⃗[i + 1] belső pont
10                  then q⃗[m] ← METSZÉS-VÁGÓSÍKKAL(p⃗[i], p⃗[i + 1])
11                       m ← m + 1
12 return q
```

Alkalmazzuk gondolatban ezt az algoritmust olyan konkáv sokszögre, amelynek a vágás következtében több részre kellene esnie (15.27. ábra). A sokszöget egyetlen tömbben nyilvántartó algoritmusunk képtelen a szétesõ részek elkülönítésére, és azokon a helyeken,

ahol valójában nem keletkezhet él, páros számú élt hoz létre.



15.27. ábra. Konkáv sokszögek vágásakor a széteső részeket páros számú élek tartják össze.

A páros számú extra él azonban nem okoz problémát a továbbiakban, ha a sokszög belső pontjait a következő elv alapján határozzuk meg: a kérdéses pontból egy félegyenest indítunk a végtelen felé és megvizsgáljuk, hogy hányszor metszi a sokszög éleit. Páratlan számú metszéspont esetén a pontot a sokszög belső pontjának tekintjük, páros számú metszés esetén külső pontnak.

Az ismertetett algoritmus többszörösen összefüggő sokszögek vágására is alkalmazható, csak ebben az esetben a bemutatott eljárást minden határoló zárt töröttvonalra különkülön kell végrehajtani.

Szakaszok és poligonok vágása konvex poliéderre

Miként korábban megállapítottuk, egy konvex poliéder előállítható a lapjaira illeszkedő síkok által határolt félterek metszeteként (15.22. ábra bal oldala), így a konvex poliéderre vágást visszavezethetjük félterekre történő vágásra. Egy féltérre vágás kimenete a következő féltérre vágás bemeneti vágandó alakzata lesz, a végső eredményt pedig az utolsó féltéren végrehajtott vágáson is átjutó alakzat jelenti.

Szakaszok vágása AABB-re

A képszintézis algoritmusokban különösen fontos szerepet kap egy speciális típusú konvex poliéder, az AABB.

15.11. definíció. *A koordinátatengelyekkel párhuzamos oldalú téglatesteket* ***AABB****-nek nevezzük. Egy AABB-t a minimális és maximális Descartes-koordinátáival adunk meg:* $[x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max}]$.

Bár az AABB-re vágáshoz használható lenne az általános konvex poliéderre kidolgozott vágás, az AABB-k jelentősége miatt erre az esetre különlegesen gyors eljárást dolgoztak ki.

Az AABB koordinátatengelyekkel párhuzamos oldalsíkjai a teret két részre bontják, egy belső részre, amelyikben maga az AABB található, és egy külső részre. A konvex poliéderre végrehajtott vágáshoz hasonlóan vessük össze a vizsgált pontot az AABB lapjaival, pontosabban azok síkjaival. Ha a pontunk minden sík tekintetében a belső részben van, akkor a pont az AABB-ben van, ha viszont valamely sík esetén a külső részben van, akkor a pont nem lehet az AABB belsejében.

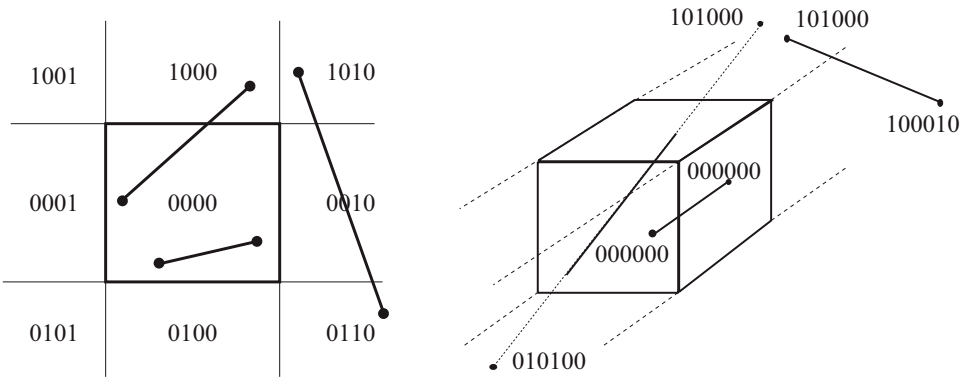
A szakasz AABB-re vágásához ezt az algoritmust mind a hat síkra végre kell hajtani, így előfordulhat, hogy kiszámítunk olyan metszéspontot is, amelyet egy másik vágósík fe-

leslegessé tesz. Érdemes tehát a síkok sorrendjét ügyesen megválasztani. Az egyik legegyszerűbb módszer a ***Cohen–Sutherland szakaszvágó algoritmus.***

Jelöljük 1-gyel, ha a pont nem a vágási tartomány félterében helyezkedik el, míg 0-val, ha az AABB-vel azonos féltérben található. Mivel 6 határoló sík létezik, 6 darab 0 vagy 1 értékünk lesz, amelyeket egymás mellé téve egy 6-bites kódot kapunk (15.28. ábra). Egy pont $C[0], \ldots, C[5]$ kódbitjei:

$$C[0] = \begin{cases} 1, & x \le x_{min}, \\ 0 & \text{egyébként.} \end{cases} \qquad C[1] = \begin{cases} 1, & x \ge x_{max}, \\ 0 & \text{egyébként.} \end{cases} \qquad C[2] = \begin{cases} 1, & y \le y_{min}, \\ 0 & \text{egyébként.} \end{cases}$$

$$C[3] = \begin{cases} 1, & y \ge y_{max}, \\ 0 & \text{egyébként.} \end{cases} \qquad C[4] = \begin{cases} 1, & z \le z_{min}, \\ 0 & \text{egyébként.} \end{cases} \qquad C[5] = \begin{cases} 1, & z \ge z_{max}, \\ 0 & \text{egyébként.} \end{cases}$$



15.28. ábra. A sík pontjainak 4-bites és a tér pontjainak 6-bites kódjai. A szakasz vágásnál a koordinátasíkok sorrendjét a szakasz végpontjainak kódjai választják ki.

Nyilvánvalóan a 000000 kóddal rendelkező pontok a vágási tartományban, a többiek pedig azon kívül találhatók (15.28. ábra). Alkalmazzuk ezt a szakaszok vágására. Legyen a szakasz két végpontjához tartozó kód C_1 és C_2. Ha mindkettő 0, akkor mindkét végpont a vágási tartományon belül van, így a szakaszt nem kell vágni (triviális elfogadás). Ha a két kód ugyanazon a biten 1, akkor egyrészt egyik végpont sincs a vágási tartományban, másrészt ugyanabban a „rossz” féltérben találhatók, így az őket összekötő szakasz is ebben a féltérben helyezkedik el. Ez pedig azt jelenti, hogy nincs a szakasznak olyan része, amely „belelógna” a vágási tartományba, így az ilyen szakaszokat a további feldolgozásból ki lehet zárni (triviális eldobás). Ezt a vizsgálatot legegyszerűbben úgy végezhetjük el, hogy a C_1 és C_2 kódokon végrehajtjuk a bitenkénti ÉS műveletet (a C programozási nyelv jelöléseit alkalmazva C_1 & C_2), és ha az eredményül kapott kód nem nulla, akkor az azt jelenti, hogy a két kód ugyanazon a biten 1.

Végül, ha egyik eset sem áll fenn, akkor kell lennie olyan bitnek, ahol az egyik kód 0, a másik pedig 1 értékű. Ez azt jelenti, hogy van olyan vágósík, amelyre nézve az egyik végpont a belső, a másik pedig a külső („rossz”) tartományban van, így a szakaszt erre a síkra vágni kell. A vágás után a metszéspont kódbitjeit kiértékeljük és a rossz oldalon lévő végpontot a metszéspontra cseréljük. Az eljárást a feltételek ellenőrzésétől kezdve addig ismételjük, amíg „triviális elfogadással” vagy „triviális eldobással” nem tudunk végső döntést

hozni.

A Cohen–Sutherland szakaszvágó algoritmus a vágott szakasz végpontjait a paraméterként kapott végpontok módosításával adja meg, és visszatérési értéke akkor igaz, ha a vágás után a szakasz legalább részben a vágási tartomány belsejében van:

COHEN–SUTHERLAND-SZAKASZ-VÁGÁS($\vec{r}_1, \vec{r}_2$)

```
 1  C_1 ← az r_1 végpont kódja            ▷ Kódbitek az egyenlőtlenségek ellenőrzésével.
 2  C_2 ← az r_2 végpont kódja
 3  while IGAZ
 4        do if C_1 = 0 és C_2 = 0
 5              then return IGAZ            ▷ Triviális elfogadás: van belső szakasz.
 6           if C_1 & C_2 ≠ 0
 7              then return HAMIS           ▷ Triviális eldobás: nincs belső szakasz.
 8           f ← a legelső bit indexe, amelyen a C_1 és a C_2 különbözik
 9           r_i ← az (r_1, r_2) szakasz és az f indexű sík metszéspontja
10           C_i ← az r_i metszéspont kódja
11           if C_1[f] = 1
12              then r_1 ← r_i
13                   C_1 ← C_i              ▷ r_1 van az f-edik sík rossz oldalán.
14              else r_2 ← r_i
15                   C_2 ← C_i              ▷ r_2 van az f-edik sík rossz oldalán.
```

Gyakorlatok

15.4-1. Tegyünk javaslatokat a poliéder-poliéder ütközésfelismerés négyzetes időbonyolultságának csökkentésére.

15.4-2. Készítsünk CSG-fa adatszerkezethez pont tartalmazást vizsgáló algoritmust.

15.4-3. Készítsünk algoritmust, amely egy sokszöget egy másik, akár konkáv sokszögre vág.

15.4-4. Készítsünk algoritmust, amely egy poliéder befoglaló gömbjét, illetve befoglaló AABB-jét kiszámítja.

15.4-5. Adjunk algoritmust a síkban két háromszög ütközésének felismeréséhez.

15.4-6. Általánosítsuk a Cohen–Sutherland szakaszvágó módszert konvex poliéder vágási tartományra.

15.4-7. Dolgozzuk ki a szakaszt gömbre vágó módszert.

15.5. Mozgatás, torzítás, geometriai transzformációk

A virtuális világ szereplői mozoghatnak, torzulhatnak, nőhetnek, illetve összemehetnek, azaz az eddig ismertetett egyenleteket elvileg minden időpillanatban újra fel kell venni. A gyakorlatban ehelyett két függvénnyel dolgozunk. Az első függvény az előző alfejezet módszerei szerint kiválasztja a tér azon pontjait, amelyek az adott objektumhoz tartoznak annak egy referenciahelyzetében. A második függvény pedig a referenciahelyzetben az objektumhoz tartozó pontokat leképezi az aktuális időpillanathoz tartozó pontokra. A teret

önmagára leképező függvény a ***transzformáció.*** A mozgatást invertálható $\mathcal{T}(\vec{r})$ transzformációval írhatjuk le. Az $\vec{r}$ kiindulási pont neve ***tárgypont,*** az $\vec{r}\,' = \mathcal{T}(\vec{r})$ pedig a ***képpont.*** Az invertálhatóság miatt minden transzformált $\vec{r}\,'$ ponthoz megkereshetjük annak eredeti alakzatbeli ősképét a $\mathcal{T}^{-1}(\vec{r}\,')$ inverz transzformáció segítségével.

Ha az alakzatot a referenciahelyzetében az f implicit függvény definiálja, akkor a transzformált alakzat pontjainak halmaza

$$\{\vec{r}\,' : f(\mathcal{T}^{-1}(\vec{r}\,')) \geq 0\} \,, \tag{15.12}$$

hiszen a transzformált alakzat pontjainak ősképei az eredeti alakzatban vannak.

A paraméteres egyenletek közvetlenül az alakzat pontjainak Descartes-koordinátáit adják meg, így a ponthalmaz transzformációjához a paraméteres alakot kell transzformálni. Egy $\vec{r} = \vec{r}(u, v)$ egyenletű felület transzformáltjának egyenlete tehát

$$\vec{r}\,'(u, v) = \mathcal{T}(\vec{r}(u, v)) \,. \tag{15.13}$$

Hasonlóan egy $\vec{r} = \vec{r}(t)$ egyenletű görbe transzformáltjának egyenlete:

$$\vec{r}\,'(t) = \mathcal{T}(\vec{r}(t)) \,. \tag{15.14}$$

A $\mathcal{T}$ transzformáció általános esetben megváltoztathatja az alakzat és egyenletének a jellegét. Könnyen előfordulhat, hogy egy háromszögből vagy egy gömbből bonyolult, torz alakzat keletkezik, ami csak ritkán kívánatos. Ezért a transzformációk körét érdemes szűkíteni. Nagy jelentősége van például az olyan transzformációknak, amelyek síkot síkba, egyenest egyenesbe visznek át. Ehhez a csoporthoz tartoznak a ***homogén lineáris transzformációk,*** amelyekkel a következő fejezetben foglalkozunk.

15.5.1. Projektív geometria és homogén koordináták

Idáig a virtuális világ felépítését az euklideszi geometria alapján tárgyaltuk, amely olyan fontos fogalmakat adott a kezünkbe, mint a távolság, a párhuzamosság, a szög stb. A transzformációk mélyebb tárgyalása során azonban ezek a fogalmak érdektelenek, sőt bizonyos esetekben zavart is kelthetnek. A párhuzamosság például az egyenesek egy speciális viszonyát jelenti, amelyet külön kell kezelni akkor, ha egyenesek metszéspontjáról beszélünk. Ezért a transzformációk tárgyalásához az euklideszi geometria helyett egy erre alkalmasabb eszközt, a projektív geometriát hívjuk segítségül.

A ***projektív geometria*** axiómái ügyesen kikerülik az euklideszi geometria párhuzamosainak a problémáját azzal, hogy teljesen mellőzik ezt a fogalmat, és kijelentik, hogy két különböző egyenesnek mindig van metszéspontja. Ennek érdekében a projektív geometriában minden egyeneshez hozzáveszünk egy „végtelen távoli” pontot, mégpedig úgy, hogy két egyeneshez akkor és csak akkor tartozzon ugyanaz a pont, ha a két egyenes párhuzamos. Az új pontot ***ideális pontnak*** nevezzük. A ***projektív tér*** az euklideszi tér pontjait (az úgynevezett ***közönséges pontokat***) és az ideális pontokat tartalmazza. Az ideális pont „összeragasztja” az euklideszi egyenes „végeit”, ezért topológiailag az egyenes a körhöz lesz hasonlatos. Továbbra is érvényben szeretnénk tartani az euklideszi geometria azon axiómáját, hogy két pont egy egyenest határoz meg. Annak érdekében, hogy ez két ideális pontra is igaz legyen, az euklideszi sík egyeneseinek halmazát bővítjük az ideális pontokat tartalmazó, úgynevezett ***ideális egyenessel.*** Mivel két egyenes ideális pontja csak akkor egyezik meg, ha a

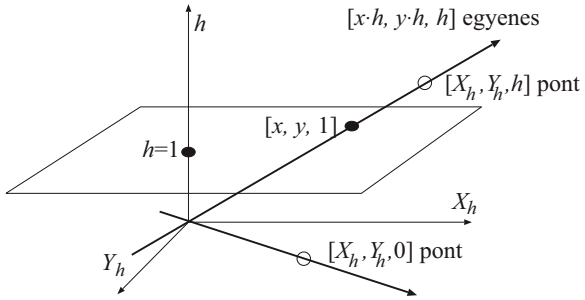
két egyenes párhuzamos, két sík ideális egyenese akkor és csak akkor azonos, ha a két sík párhuzamos. Az ideális egyenesek az ***ideális síkra*** illeszkednek, amelyet ugyancsak hozzáveszünk a tér síkjainak halmazához. Ezen döntések után nem kell különbséget tennünk az euklideszi tér pontjai és az ideális pontok között, ők a projektív tér ugyanolyan tagjai.

Az analitikus geometria bevezetésénél említettük, hogy a számítógépes grafikában mindent számokkal kell leírni. Az idáig használt Descartes-koordináták egy-egy értelmű kapcsolatban állnak az euklideszi tér pontjaival, így az ideális pontok jellemzésére alkalmatlanok. Az euklideszi geometria pontjait és az ideális pontokat egyaránt tartalmazó ***projektív sík*** és ***projektív tér*** jellemzésére tehát más algebrai alapra van szükségünk.

Projektív sík

Először tekintsük a projektív síkot, amelynek pontjait szeretnénk számszerűsíteni, és vegyünk fel egy x, y koordinátarendszert ezen a síkon. Válasszunk az euklideszi térben egy X_h, Y_h, h Descartes-koordinátarendszert úgy, hogy az X_h, Y_h tengelyek az x, y tengelyekkel párhuzamosak legyenek, a sík koordinátarendszerének origója a tér $(0, 0, 1)$ pontjában legyen, és a síkunk pontjaira a $h = 1$ egyenlet teljesüljön. A vizsgált projektív síkot tehát beágyaztuk egy háromdimenziós euklideszi térbe, amelynek pontjait Descartes-koordinátákkal adjuk meg (15.29. ábra). A projektív sík pontjainak számokkal történő azonosításához pedig kapcsolatot teremtünk a beágyazó euklideszi tér pontjai és a projektív sík pontjai között. Ez a kapcsolat a projektív sík egy közönséges, vagy akár ideális P pontját a beágyazó euklideszi tér azon egyenesén lévő pontoknak felelteti meg, amelyet a beágyazó koordinátarendszer origója és a P pont meghatároz.

Az origón átmenő egyenes pontjait a $[t \cdot X_h, t \cdot Y_h, t \cdot h]$ paraméteres formában adhatjuk meg a t paraméter függvényében. Amennyiben a P pont a sík közönséges pontja, az egyenes nem párhuzamos a $h = 1$ síkkal (azaz $h \neq 0$). Az egyenes az $[X_h/h, Y_h/h, 1]$ pontban metszi a síkot, így a P pontnak a síkhoz rendelt Descartes-koordinátarendszerbeli koordinátái $(X_h/h, Y_h/h)$. Ha viszont a P pont ideális, az egyenes párhuzamos a $h = 1$ síkkal (azaz $h = 0$). Az ideális pontok irányát ebben az esetben az (X_h, Y_h) vektor jelöli ki.



15.29. ábra. A projektív sík beágyazott modellje: a projektív síkot a háromdimenziós euklideszi térbe ágyazzuk, és a projektív sík egy pontjának az euklideszi tér azon egyenesét feleltetjük meg, amelyet az origó és az adott pont meghatároz.

Ezzel az eljárással tehát a síknak mind a közönséges, mind pedig az ideális pontjaihoz $[X_h, Y_h, h]$ számhármasokat rendeltünk, amelyeket a síkbeli pont ***homogén koordinátáinak*** nevezünk. A homogén koordinátákat szögletes zárójelek közé tesszük, hogy a Descartes-koordinátáktól megkülönböztessük.

A homogén koordináták bevezetésénél a projektív sík egy pontjának az euklideszi tér-

nek az origón átmenő egyenesét feleltettünk meg, amelyet bármely, az origótól különböző pontjával megadhatunk. Ebből következik, hogy mindhárom homogén koordináta nem lehet egyszerre zérus, és hogy a homogén koordináták egy nem zérus skalárral szabadon szorozhatók, ettől még a projektív sík ugyanazon pontját azonosítják. Ez a tulajdonság indokolja a „homogén" elnevezést.

A közönséges pontokat azonosító homogén koordinátahármasok közül gyakran célszerű azt kiválasztani, ahol a harmadik koordináta 1 értékű, ugyanis ekkor az első két homogén koordináta a Descartes-koordinátákkal egyezik meg:

$$X_h = x, \qquad Y_h = y, \qquad h = 1 \ . \tag{15.15}$$

Descartes-koordinátákat tehát úgy alakíthatunk homogén koordinátákká, hogy hozzájuk csapunk egy harmadik, 1 értékű elemet.

A beágyazott modell alapján könnyen felírhatjuk a projektív sík egyeneseinek és szakaszainak homogén koordinátás egyenletét is. Vegyünk fel a projektív síkon két, különböző pontot, és adjuk meg őket a homogén koordinátáikkal. A különbözőség azt jelenti, hogy az $[X_h^1, Y_h^1, h^1]$ hármasból egy skalárral való szorzással nem állítható elő az $[X_h^2, Y_h^2, h^2]$ hármas. A síkot beágyazó térben az $[X_h, Y_h, h]$ hármas Descartes-koordinátának tekinthető, így az $[X_h^1, Y_h^1, h^1]$ és $[X_h^2, Y_h^2, h^2]$ pontokat összekötő ***egyenes egyenlete:***

$$\begin{aligned} X_h(t) &= X_h^1 \cdot (1-t) + X_h^2 \cdot t \ , \\ Y_h(t) &= Y_h^1 \cdot (1-t) + Y_h^2 \cdot t \ , \\ h(t) &= h^1 \cdot (1-t) + h^2 \cdot t \ . \end{aligned} \tag{15.16}$$

Ha $h(t) \neq 0$, akkor projektív síkon lévő közönséges pontokat úgy kapjuk meg, hogy a háromdimenziós tér pontjait a $h = 1$ síkra vetítjük. A vetítés egyenest egyenesbe visz át, hiszen a különböző pontok megkövetelésével kizártuk azt az esetet, amikor a vetítés az egyenest egyetlen pontra képezné le. Tehát az egyenlet valóban egy egyenes pontjait azonosítja. Ha viszont $h(t) = 0$, akkor az egyenlet az egyenes ideális pontját fejezi ki.

Ha t tetszőleges értéket felvehet, akkor az egyenes pontjait kapjuk. Ha viszont t értékét a $[0, 1]$ intervallumra korlátozzuk, a két pont által kijelölt szakasz egyenletéhez jutunk.

Projektív tér

A projektív tér homogén koordinátáinak bevezetéséhez ugyanazt az utat követhetnénk, mint amit a síknál használtunk, de ehhez a háromdimenziós projektív teret a négydimenziós euklideszi térbe kellene beágyazni, ami kevésbé szemléletes. Ezért egy másik konstrukciót is tárgyalunk, amely tetszőleges dimenzióban működik. A pontjainkat mechanikai rendszerek súlypontjaiként írjuk le. Egyetlen pont azonosításához egy $\vec{p}_1$ referencia pontban X_h súlyt helyezünk el, egy $\vec{p}_2$ referencia pontban Y_h súlyt, egy $\vec{p}_3$ pontban Z_h súlyt és végül egy $\vec{p}_4$ pontban w súlyt. A mechanikai rendszer súlypontja:

$$\vec{r} = \frac{X_h \cdot \vec{p}_1 + Y_h \cdot \vec{p}_2 + Z_h \cdot \vec{p}_3 + w \cdot \vec{p}_4}{X_h + Y_h + Z_h + w} \ .$$

Vezessük be az összsúly fogalmát a $h = X_h + Y_h + Z_h + w$ egyenlettel. Definíciószerűen az $[X_h, Y_h, Z_h, h]$ négyest a súlypont ***homogén koordinátáinak*** nevezzük.

A homogén és a Descartes-koordináták közötti összefüggés felállításához a két koordinátarendszer viszonyát (a Descartes-koordinátarendszer bázisvektorainak, origójának és

a homogén koordinátarendszer referencia pontjainak kapcsolatát) rögzíteni kell. Tegyük fel például, hogy a referencia pontok a Descartes-koordinátarendszer (1,0,0), (0,1,0), (0,0,1) és (0,0,0) pontjaiban vannak. A mechanikai rendszerünk súlypontja (ha a h összsúly nem zérus) a Descartes-koordinátarendszerben:

$$\vec{r}[X_h, Y_h, Z_h, h] = \frac{1}{h} \cdot (X_h \cdot (1,0,0) + Y_h \cdot (0,1,0) + Z_h \cdot (0,0,1) + w \cdot (0,0,0)) = \left(\frac{X_h}{h}, \frac{Y_h}{h}, \frac{Z_h}{h}\right) .$$

Tehát az $[X_h, Y_h, Z_h, h]$ homogén koordináták és az (x, y, z) Descartes-koordináták közötti összefüggés ($h \neq 0$):

$$x = \frac{X_h}{h}, \qquad y = \frac{Y_h}{h}, \qquad z = \frac{Z_h}{h} . \tag{15.17}$$

A ***projektív tér egyeneseinek*** és szakaszainak homogén koordinátás egyenletét akár a négydimenziós térbe ágyazott projektív tér modell felhasználásával, akár a súlypont analógia alapján is megkaphatjuk:

$$\begin{aligned} X_h(t) &= X_h^1 \cdot (1-t) + X_h^2 \cdot t , \\ Y_h(t) &= Y_h^1 \cdot (1-t) + Y_h^2 \cdot t , \\ Z_h(t) &= Z_h^1 \cdot (1-t) + Z_h^2 \cdot t , \\ h(t) &= h^1 \cdot (1-t) + h^2 \cdot t . \end{aligned} \tag{15.18}$$

Ha t értékét a $[0, 1]$ intervallumra korlátozzuk, a két pont által kijelölt ***projektív szakasz*** egyenletéhez jutunk.

A ***projektív sík*** egyenletének felírásához induljunk ki az euklideszi tér síkjának (15.1) egyenletéből. A sík pontjainak Descartes-koordinátái kielégítik az

$$n_x \cdot x + n_y \cdot y + n_z \cdot z + d = 0$$

implicit egyenletet. A (15.17) egyenletben szereplő Descartes és homogén koordináták közötti összefüggést behelyettesítve az egyenletbe továbbra is az euklideszi sík pontjait írjuk le:

$$n_x \cdot \frac{X_h}{h} + n_y \cdot \frac{Y_h}{h} + n_z \cdot \frac{Z_h}{h} + d = 0 .$$

Szorozzuk meg az egyenlet mindkét oldalát h-val, majd a $h = 0$ koordinátájú, az egyenletet kielégítő pontokat is adjuk síkhoz. Ezzel az euklideszi sík pontjait kiegészítettük az ideális pontokkal, azaz a projektív síkhoz jutottunk. A projektív sík egyenlete tehát egy homogén lineáris egyenlet:

$$n_x \cdot X_h + n_y \cdot Y_h + n_z \cdot Z_h + d \cdot h = 0 , \tag{15.19}$$

vagy mátrixos alakban:

$$[X_h, Y_h, Z_h, h] \cdot \begin{bmatrix} n_x \\ n_y \\ n_z \\ d \end{bmatrix} = 0 . \tag{15.20}$$

Figyeljük meg, hogy a tér pontjait négyelemű sorvektorokkal, a síkjait pedig négyelemű oszlopvektorokkal írhatjuk le. Mind a pontok négyesei, mind pedig a síkok négyesei homogén tulajdonságúak, azaz skalárral szabadon szorozhatók anélkül, hogy a homogén lineáris egyenlet megoldásai változnának.

15.5.2. Homogén lineáris transzformációk

Azokat a leképzéseket, ahol a képpont homogén koordinátáit a tárgypont homogén koordinátáinak és egy állandó 4×4-es $\mathbf{T}$ transzformációs mátrixnak a szorzataként írhatjuk fel, ***homogén lineáris transzformációknak*** nevezzük:

$$[X'_h, Y'_h, Z'_h, h'] = [X_h, Y_h, Z_h, h] \cdot \mathbf{T} \,. \tag{15.21}$$

15.12. tétel. *A homogén lineáris transzformációk pontot pontba visznek át.*

Bizonyítás. A transzformáció egy tárgypontját homogén koordinátákban $\lambda \cdot [X_h, Y_h, Z_h, h]$ alakban adhatjuk meg, ahol λ tetszőleges, nem zérus konstans. Ezekből a négyesekből a transzformáció $\lambda \cdot [X'_h, Y'_h, Z'_h, h'] = \lambda \cdot [X_h, Y_h, Z_h, h] \cdot \mathbf{T}$ alakú négyeseket hoz létre, amelyek ugyanazon négyes λ-szorosai, így az eredmény egyetlen pont homogén koordinátáit adja. ■

Figyeljük meg, hogy a homogén tulajdonság miatt a homogén lineáris transzformációk csak egy skalárral való szorzás erejéig meghatározottak, azaz a transzformáció nem változik, ha a $\mathbf{T}$ mátrix minden elemét ugyanazzal a nem zérus skalárral szorozzuk.

15.13. tétel. *Az invertálható homogén lineáris transzformációk egyenest egyenesbe visznek át.*

Bizonyítás. Induljunk ki a tárgyegyenes paraméteres egyenletéből:

$$[X_h(t), Y_h(t), Z_h(t), h(t)] = [X_h^1, Y_h^1, Z_h^1, h^1] \cdot (1-t) + [X_h^2, Y_h^2, Z_h^2, h^2] \cdot t, \qquad t = (-\infty, \infty) \,,$$

és állítsuk elő a képalakzatot a tárgyalakzat pontjainak egyenkénti transzformálásával:

$$\begin{aligned}
[X'_h(t), Y'_h(t), Z'_h(t), h'(t)] &= [X_h(t), Y_h(t), Z_h(t), h(t)] \cdot \mathbf{T} \\
&= [X_h^1, Y_h^1, Z_h^1, h^1] \cdot \mathbf{T} \cdot (1-t) + [X_h^2, Y_h^2, Z_h^2, h^2] \cdot \mathbf{T} \cdot t \\
&= [X_h^{1\prime}, Y_h^{1\prime}, Z_h^{1\prime}, h^{1\prime}] \cdot (1-t) + [X_h^{2\prime}, Y_h^{2\prime}, Z_h^{2\prime}, h^{2\prime}] \cdot t \,,
\end{aligned}$$

ahol $[X_h^{1\prime}, Y_h^{1\prime}, Z_h^{1\prime}, h^{1\prime}]$ az $[X_h^1, Y_h^1, Z_h^1, h^1]$ pont, $[X_h^{2\prime}, Y_h^{2\prime}, Z_h^{2\prime}, h^{2\prime}]$ pedig az $[X_h^2, Y_h^2, Z_h^2, h^2]$ pont transzformáltja. A transzformáció invertálhatósága miatt a két pont különböző. A transzformált alakzat egyenlete éppen egy egyenes egyenlete, amely a két transzformált pontra illeszkedik. ■

Megjegyezzük, hogy ha nem kötöttük volna ki a transzformáció invertálhatóságát, akkor előfordulhatott volna, hogy a két tartópont képe ugyanaz a pont lenne, így az egyenesből a transzformáció következtében egyetlen pont keletkezik.

Ha a t paramétert a $[0, 1]$ tartományra korlátozzuk, akkor a projektív szakasz egyenletét kapjuk, így kimondhatjuk, hogy a homogén lineáris transzformáció projektív szakaszt projektív szakaszba visz át. Sőt általánosan is igaz, hogy a homogén lineáris transzformációk konvex-kombinációkat konvex-kombinációkba visznek át, így például háromszögből háromszög keletkezik.

Ezzel a tétellel azonban nagyon óvatosan kell bánnunk akkor, ha az euklideszi geometria szakaszairól és háromszögeiről beszélünk. Vegyünk egy szakaszt. Ha a két végpont h

koordinátája eltérő előjelű, a pontokat összekötő projektív szakasz tartalmazza az ideális pontot is. Az ilyen szakasz intuitíve két félegyenesből és a félegyenesek „végét" összekapcsoló ideális pontból áll, azaz a szokásos euklideszi szakasz fogalmunk kifordult változata. Előfordulhat, hogy a transzformáció tárgyszakaszában a végpontok azonos előjelű h koordinátákkal rendelkeznek, azaz a projektív szakasz megfelel az euklideszi geometria szakaszfogalmának, a transzformáció képszakaszának végpontjaiban viszont a h koordináták már eltérő előjelűek lesznek. Így szemléletesen a transzformáció kifordítja a szakaszunkat.

15.14. tétel. *Az invertálható homogén lineáris transzformációk síkot síkba visznek át.*

Bizonyítás. Az $[X_h, Y_h, Z_h, h] = [X'_h, Y'_h, Z'_h, h'] \cdot \mathbf{T}^{-1}$ pontok – azaz az $[X'_h, Y'_h, Z'_h, h']$ transzformált pontok ősképei – egy síkon vannak, ezért kielégítik az eredeti sík egyenletét:

$$[X_h, Y_h, Z_h, h] \cdot \begin{bmatrix} n_x \\ n_y \\ n_z \\ d \end{bmatrix} = [X'_h, Y'_h, Z'_h, h'] \cdot \mathbf{T}^{-1} \cdot \begin{bmatrix} n_x \\ n_y \\ n_z \\ d \end{bmatrix} = 0 \ .$$

A mátrixszorzás asszociativitása miatt a transzformált pontok kielégítik az

$$[X'_h, Y'_h, Z'_h, h'] \cdot \begin{bmatrix} n'_x \\ n'_y \\ n'_z \\ d' \end{bmatrix} = 0$$

egyenletet, amely ugyancsak egy sík egyenlete, ahol

$$\begin{bmatrix} n'_x \\ n'_y \\ n'_z \\ d' \end{bmatrix} = \mathbf{T}^{-1} \cdot \begin{bmatrix} n_x \\ n_y \\ n_z \\ d \end{bmatrix} \ .$$

Ezt az összefüggést felhasználhatjuk a transzformált sík normálvektorának számításához. ■

A homogén lineáris transzformációk fontos speciális esetei az ***affin transzformációk,*** amelyekben a képpont Descartes-koordinátái a tárgypont Descartes-koordinátáinak lineáris függvényei:

$$[x', y', z'] = [x, y, z] \cdot \mathbf{A} + [p_x, p_y, p_z] \ , \tag{15.22}$$

ahol a különálló $\vec{p}$ vektor az eltolásért felelős, az $\mathbf{A}$ pedig egy 3×3-as mátrix, amely a forgatást, a skálázást, a tükrözést stb., sőt ezek tetszőleges kombinációját is kifejezheti. Például az origón átmenő (t_x, t_y, t_z), $(|(t_x, t_y, t_z)| = 1)$ irányú tengely körül ϕ szöggel forgató transzformációban

$$\mathbf{A} = \begin{bmatrix} (1-t_x^2)\cos\phi + t_x^2 & t_x t_y(1-\cos\phi) + t_z \sin\phi & t_x t_z(1-\cos\phi) - t_y \sin\phi \\ t_y t_x(1-\cos\phi) - t_z \sin\phi & (1-t_y^2)\cos\phi + t_y^2 & t_x t_z(1-\cos\phi) + t_x \sin\phi \\ t_z t_x(1-\cos\phi) + t_y \sin\phi & t_z t_y(1-\cos\phi) - t_x \sin\phi & (1-t_z^2)\cos\phi + t_z^2 \end{bmatrix} .$$

Ez az összefüggés ***Rodrigues-képlet*** néven ismeretes.

Az affin transzformációk nem vezetnek ki az euklideszi térből, és párhuzamos egyeneseket párhuzamos egyenesekbe visznek át. Az affin transzformációk egyben homogén lineáris transzformációk, hiszen a (15.22) egyenlet egy 4×4-es mátrixművelettel is leírható, miután a Descartes-koordinátákról áttérünk homogén koordinátákra a negyedik homogén koordinátát 1-nek választva:

$$[x', y', z', 1] = [x, y, z, 1] \cdot \begin{bmatrix} A_{11} & A_{12} & A_{13} & 0 \\ A_{21} & A_{22} & A_{23} & 0 \\ A_{31} & A_{32} & A_{33} & 0 \\ p_x & p_y & p_z & 1 \end{bmatrix} = [x, y, z, 1] \cdot \mathbf{T} \,. \tag{15.23}$$

Az affin transzformációk körén belül további speciális eset a távolságtartó transzformáció, amelyet ***egybevágósági transzformációnak*** nevezünk. Az egybevágósági transzformációk egyben szögtartóak is.

15.15. tétel. *Az egybevágósági transzformációkban az* $\mathbf{A}$ *mátrix sorai egységvektorok és egymásra merőlegesek.*

Bizonyítás. Induljunk ki az egybevágósági transzformációk szög- és távolságtartó tulajdonságából, és alkalmazzuk ezt arra az esetre, amikor éppen az origót és a bázisvektorokat transzformáljuk. Az origóból a transzformáció a (p_x, p_y, p_z) pontot állítja elő, az $(1, 0, 0)$, $(0, 1, 0)$ és $(0, 0, 1)$ pontokból pedig rendre az $(A_{11} + p_x, A_{12} + p_y, A_{13} + p_z)$, $(A_{21} + p_x, A_{22} + p_y, A_{23} + p_z)$ és $(A_{31} + p_x, A_{32} + p_y, A_{33} + p_z)$ pontokat. A távolságtartás miatt transzformált pontok és az új origó távolsága továbbra is egységnyi, azaz $|A_{11}, A_{12}, A_{13}| = 1$, $|A_{21}, A_{22}, A_{23}| = 1$ és $|A_{31}, A_{32}, A_{33}| = 1$. Másrészt, a szögtartás miatt a bázisvektorok transzformáltjai, az (A_{11}, A_{12}, A_{13}), (A_{21}, A_{22}, A_{23}) és (A_{31}, A_{32}, A_{33}) vektorok egymásra merőlegesek. ■

Gyakorlatok

15.5-1. A Descartes-koordinátarendszer, mint algebrai alap felhasználásával igazoljuk az euklideszi geometria axiómáit, például, hogy két pontra egy egyenes illeszkedik, és hogy két különböző egyenes legfeljebb egyetlen pontban metszi egymást.

15.5-2. A homogén koordináták, mint algebrai alap felhasználásával igazoljuk a projektív geometria egy axiómáját, miszerint két különböző egyenes mindig egy pontban metszi egymást.

15.5-3. Bizonyítsuk be a súlypont analógia alapján, hogy a homogén lineáris transzformációk egy háromdimenziós szakaszt szakaszba visznek át.

15.5-4. Hogyan változtatja meg egy affin transzformáció egy test térfogatát?

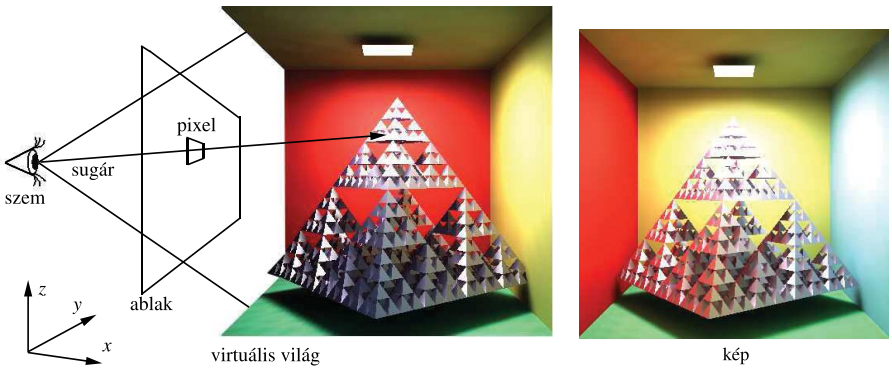
15.5-5. Írjuk fel a $\vec{p}$ vektorral eltoló transzformáció mátrixát.

15.5-6. Igazoljuk a Rodrigues-képletet.

15.5-7. Egy $f(\vec{r}) \geq 0$ egyenlőtlenséggel leírt test $\vec{v}$ sebességgel egyenletesen mozog. Írjuk fel a test pontjait leíró egyenlőtlenséget a t időpillanatban.

15.5-8. Igazoljuk, hogy ha az $\mathbf{A}$ mátrix sorai egymásra merőleges egységvektorok, akkor az affin transzformáció egyben egybevágósági transzformáció is. Mutassuk meg, hogy ilyen mátrixokra $\mathbf{A}^{-1} = \mathbf{A}^T$.

15.5-9. Írjuk fel azon homogén lineáris transzformáció mátrixát, amely a teret $\vec{c}$ középponttal az $\vec{n}$ normálvektorú, $\vec{r}_0$ helyvektorú síkra vetíti.



15.30. ábra. Képszintézis sugárkövetéssel.

15.5-10. Mutassuk meg, hogy 5 tárgypont-képpont pár egyértelműen azonosít egy homogén lineáris transzformációt, ha az 5 pont közül semelyik négy sincs egy síkon.

15.6. Megjelenítés sugárkövetéssel

A virtuális világ lefényképezéséhez azt kell meghatároznunk, hogy a virtuális megfigyelő a különböző irányokban milyen felületi pontokat lát. A lehetséges irányokat egy téglalap alakú ablakkal jelölhetjük ki, amelyet a képernyő pixeleinek megfelelően egy négyzetrácsra bontunk fel. Az ablak a képernyőt képviseli a virtuális világban (19.12. ábra). Mivel a képernyő pixeleit csak egyetlen színnel lehet kitölteni, elegendő a négyzetrács négyzeteinek egy-egy pontjában (célszerűen a pixel középpontoknak megfelelő pontokban) vizsgálnunk a látható felületet.

A szempozícióból egy irányban az a felület látható, amelyet a szempozícióból az adott irányban induló félegyenes – a ***sugár*** – a szempozícióhoz a legközelebb metsz. A sugár és a felületek legközelebbi metszéspontjának kiszámítását ***sugárkövetésnek*** nevezzük.

A sugárkövetés nem csak a láthatóság eldöntésénél kap szerepet. Ha ***árnyékot*** kívánunk számítani, azaz arra vagyunk kíváncsiak, hogy a felületek egy pontot mely fényforrások elől takarnak el, akkor a pontból a fényforrások irányába ugyancsak sugarakat küldünk és eldöntjük, hogy ezek a sugarak metszenek-e valamilyen felületet mielőtt elérnék a fényforrásokat. A sugárkövetés szerepet kap a virtuális világ objektumai közötti ***ütközések felismerésénél*** is, ugyanis egy egyenes vonalú egyenletes mozgást végző pont azzal a felülettel ütközik, amelyet a pont pályáját leíró sugár először metsz.

A sugarat a következő egyenlettel adjuk meg:

$$r\vec{a}y(t) = \vec{s} + \vec{v} \cdot t, \qquad (t > 0)\ , \tag{15.24}$$

ahol $\vec{s}$ a kezdőpont helyvektora, $\vec{v}$ a sugár iránya, a t ***sugárparaméter*** pedig a kezdőponttól való távolságot jellemzi. A továbbiakban azzal a feltételezéssel élünk, hogy $\vec{v}$ egységvektor,

mert ekkor t a tényleges távolságot jelenti, egyébként csak arányos volna a távolsággal[4]. Ha t negatív, akkor a pont a szem mögött helyezkedik el, így nem jelent metszéspontot a félegyenessel (nem látható). A legközelebbi metszéspont megkeresése a legkisebb, pozitív sugárparaméterű metszéspont előállítását jelenti. A legközelebbi metszéspont előállításához elvileg minden felülettel meg kell kísérelni a sugár és a felület metszéspontjának előállítását, és a ténylegesen létező metszéspontok közül a legközelebbit kell kiválasztani. Egy sugár legközelebbi metszéspontjának számításához a következő algoritmus alkalmazható:

SUGÁR-ELSŐ-METSZÉSPONT($\vec{s}, \vec{v}$)

```
1  t ← t_max                                   ▷ Inicializálás a tér legnagyobb méretére.
2  for minden o objektumra
3      do t_o ←SUGÁR-FELÜLET-METSZÉSPONT(s, v)   ▷ Negatív, ha nincs metszéspont.
4         if 0 ≤ t_o < t                        ▷ Az új metszéspont közelebbi-e?
5            then t ← t_o                       ▷ A legközelebbi metszés sugárparamétere.
6                 o_metszett ← o                ▷ A legközelebb metszett objektum.
7  if t < t_max then                           ▷ Volt egyáltalán metszéspont?
8     then x ← s + v · t                       ▷ A metszéspont helye a sugár egyenletéből.
9          return t, x, o_metszett
10    else return „nincs metszéspont”          ▷ Nincs metszéspont.
```

Az algoritmus a sugarat kapja bemeneti paraméteréül, és az $\vec{x}$ változóban a metszéspont helyét, az $o_{metszett}$ változóban pedig a metszett objektumot adja meg. A harmadik visszaadott érték a metszésponthoz tartozó sugárparaméter. Az algoritmus az objektumonkénti SUGÁR-FELÜLET-METSZÉSPONT eljárást használja fel, amely az adott objektum és a sugár metszéspontjához tartozó sugárparamétert számítja ki, illetve negatív értékkel jelzi, ha a metszéspont nem létezne. A SUGÁR-FELÜLET-METSZÉSPONT algoritmust objektumtípusonként külön-külön kell megvalósítani.

15.6.1. Sugár-felület metszéspont számítás

A sugár-felület metszéspont megkeresése lényegében egy egyenlet megoldását jelenti. A metszéspont a sugár és a vizsgált felület közös része, amit megkaphatunk, ha a sugár egyenletét behelyettesítjük a felület egyenletébe, és a keletkező egyenletet megoldjuk az ismeretlen sugárparaméterre.

Implicit egyenletű felületek metszése

Az $f(\vec{r}) = 0$ implicit egyenletű felületeknél az $f(\vec{s} + \vec{v} \cdot t) = 0$ skalár egyenletet kell megoldani.

Tekintsük példaként a gömböt, ellipszist, hengert, kúpot, paraboloidot stb. magában foglaló ***másodrendű felületek*** családját, amelyek implicit egyenlete egy kvadratikus alakkal

[4] Az ütközésfelismerésnél ezzel szemben a $\vec{v}$ nem egységvektor, hanem a mozgó pont sebességvektora, mert ekkor a t sugárparaméter az ütközés idejét fejezi ki.

adható meg:

$$[x, y, z, 1] \cdot \mathbf{Q} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0 \,,$$

ahol $\mathbf{Q}$ egy 4×4-es mátrix. A sugár egyenletét a felület egyenletébe helyettesítve, az

$$[s_x + v_x \cdot t, s_y + v_y \cdot t, s_z + v_z \cdot t, 1] \cdot \mathbf{Q} \cdot \begin{bmatrix} s_x + v_x \cdot t \\ s_y + v_y \cdot t \\ s_z + v_z \cdot t \\ 1 \end{bmatrix} = 0 \,,$$

egyenletet kapjuk, amit átrendezve egy másodfokú egyenlethez jutunk:

$$t^2 \cdot (\mathbf{v} \cdot \mathbf{Q} \cdot \mathbf{v}^T) + t \cdot (\mathbf{s} \cdot \mathbf{Q} \cdot \mathbf{v}^T + \mathbf{v} \cdot \mathbf{Q} \cdot \mathbf{s}^T) + (\mathbf{s} \cdot \mathbf{Q} \cdot \mathbf{s}^T) = 0 \,,$$

ahol $\mathbf{v} = [v_x, v_y, v_z, 0]$ és $\mathbf{s} = [s_x, s_y, s_z, 1]$.

A másodfokú egyenlet megoldóképletével a gyököket megkaphatjuk, amelyek közül most csak a pozitív, valós gyökök értelmesek. Ha két ilyen gyök is volna, akkor a kisebb felel meg a legközelebbi metszéspontnak.

Paraméteres felületek metszése

Az $\vec{r} = \vec{r}(u, v)$ paraméteres felület és a sugár metszéspontját úgy kereshetjük meg, hogy először az ismeretlen u, v, t paraméterekre megoldjuk az

$$\vec{r}(u, v) = \vec{s} + t \cdot \vec{v}$$

háromváltozós, nemlineáris egyenletrendszert, majd ellenőrizzük, hogy a kapott t pozitív-e, és az u, v paraméterek valóban a megengedett paramétertartomány belsejében vannak-e.

A nemlineáris egyenletrendszer gyökeit általában numerikus módszerekkel állíthatjuk elő, vagy a felületeket háromszöghálóval közelítjük, majd ezt próbáljuk meg a sugárral elmetszeni. Ha sikerül metszéspontot találni, az eredményt úgy lehet pontosítani, hogy a metszéspont környezetének megfelelő paramétertartományban egy finomabb tesszellációt készítünk, és a metszéspontszámítást újra elvégezzük.

Háromszög metszése

A ***háromszögek*** metszéséhez először előállítjuk a sugár és az $\vec{a}$, $\vec{b}$ és $\vec{c}$ csúcsú háromszög síkjának metszéspontját, majd eldöntjük, hogy a metszéspont a háromszögön belül van-e. A háromszög síkjának normálvektora $\vec{n} = (\vec{b} - \vec{a}) \times (\vec{c} - \vec{a})$, egy helyvektora pedig $\vec{a}$, tehát a sík $\vec{r}$ pontjai kielégítik a következő egyenletet:

$$\vec{n} \cdot (\vec{r} - \vec{a}) = 0 \,. \tag{15.25}$$

A sugár és a sík közös pontját megkaphatjuk, ha a sugár egyenletét ((15.24) egyenlet) behelyettesítjük a sík egyenletébe ((15.25) egyenlet), majd a keletkező egyenletet megoldjuk az ismeretlen t paraméterre. Ha a kapott t^* érték pozitív, akkor visszahelyettesítjük a sugár egyenletébe, ha viszont negatív, akkor a metszéspont a sugár kezdőpontja mögött helyezkedik el, így nem érvényes. A sík metszése után azt kell ellenőriznünk, hogy a kapott $\vec{p}$ pont vajon a háromszögön kívül vagy belül helyezkedik-e el. A tartalmazás eldöntéséhez a 15.4.1. pont algoritmusát használhatjuk fel.

AABB metszése

Egy AABB egy koordinátasíkokkal párhuzamos oldalú téglatest, amelynek felületét 6 téglalapra, illetve 12 háromszögre bonthatjuk fel, így a metszését az előző pont algoritmusaira vezethetjük vissza. Az általános sík-sugár metszés helyett azonban lényegesen hatékonyabb megoldáshoz juthatunk, ha felismerjük, hogy ebben a speciális esetben a számítások a három koordinátára külön-külön végezhetők el. Egy AABB ugyanis az $x_{min} \leq x \leq x_{max}$ egyenlőtlenséggel definiált x-réteg, az $y_{min} \leq y \leq y_{max}$ egyenlőtlenséggel definiált y-réteg, és a $z_{min} \leq z \leq z_{max}$ egyenlőtlenséggel definiált z-réteg metszete. Tekintsük például az x-réteget, amellyel a metszés sugárparaméterei:

$$t_x^1 = \frac{x_{min} - s_x}{v_x}, \qquad t_x^2 = \frac{x_{max} - s_x}{v_x} .$$

A két paraméter közül a kisebbik a rétegbe történő belépést, a másik pedig a kilépést azonosítja. Jelöljük a belépés sugárparaméterét t^{be}-vel, a kilépését t^{ki}-vel. A sugár tehát a $[t^{be}, t^{ki}]$ tartományban tartózkodik az x-réteg belsejében. Ugyanezt a számítást az y és z-rétegre is elvégezve három intervallumot kapunk. A sugár az intervallumok metszetében lesz az AABB belsejében. Ha a metszet t^{ki} paramétere negatív, akkor az AABB a szempozíció mögött van, így nincs metszéspont. Ha csak a t^{be} negatív, akkor a sugár a doboz belsejéből indul, a metszéspont pedig a t^{ki} értéknél következik be. Végül ha t^{be} is pozitív, akkor a sugár kívülről hatol a dobozba, mégpedig a t^{be} értéknél.

A felesleges metszéspontok számát a Cohen–Sutherland szakaszvágó algoritmus (15.4.3 pont) alkalmazásával csökkenthetjük, amelyhez először a sugár félegyenesét egy szakasszal váltjuk fel. A szakasz egyik pontja a sugár kezdőpontja. A másik pontot pedig a sugáregyenletnek a maximális sugárparaméter[5] melletti értéke adja.

15.6.2. A metszéspontszámítás gyorsítási lehetőségei

Egy naiv sugárkövető algoritmus egy sugarat minden objektummal összevet, és eldönti, hogy van-e köztük metszéspont. Ha N objektum van a térben, a futási idő $\Theta(N)$ mind átlagos, mind pedig legrosszabb esetben. A sugárkövetés tárigénye ugyancsak lineáris.

A módszer jelentősen gyorsítható lenne, ha az objektumok egy részére kapásból meg tudnánk mondani, hogy az adott sugár biztosan nem metszheti őket (mert például azok a sugár kezdőpontja mögött, vagy nem a sugár irányában helyezkednek el), illetve miután találunk egy metszéspontot, akkor ki tudnánk zárni az objektumok egy másik körét azzal, hogy ha a sugár metszi is őket, akkor azok biztosan ezen metszéspont mögött lesznek. Ilyen döntésekhez ismernünk kell az objektumteret. A megismeréshez egy előfeldolgozási fázis szükséges, amelyben a metszéspontszámítás gyorsításához szükséges adatstruktúrát építjük fel. Az előfeldolgozásnak természetesen ára van, amely akkor térül meg, ha utána nagyon sok sugarat kell követnünk.

Befoglaló keretek

A legegyszerűbb gyorsítási módszer a ***befoglaló keret*** alkalmazása. A befoglaló keret egy egyszerű geometriájú objektum, tipikusan gömb vagy AABB, amely egy-egy bonyolultabb objektumot teljes egészében tartalmaz. A sugárkövetés során először a befoglaló keretet próbáljuk a sugárral elmetszeni. Ha nincs metszéspont, akkor nyilván a befoglalt objek-

[5] t_{max} = a kamerával együtt értendő színtér átmérője.

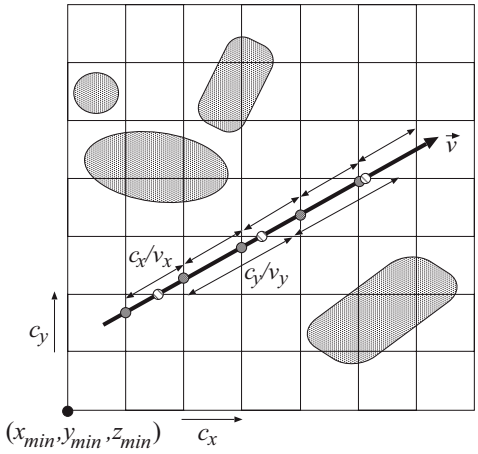
tummal sem lehet metszéspont, így a bonyolultabb számítást megtakaríthatjuk. A befoglaló keretet úgy kell kiválasztani, hogy a sugárral alkotott metszéspontja könnyen kiszámítható legyen, és kellően szorosan körbe ölelje az objektumot.

A naiv sugárkövetés az objektumok számával lineárisan növekvő időt igényel. A befoglaló keretek alkalmazása után az algoritmus továbbra is lineáris. A lineáris tag együtthatója viszont várhatóan kisebb.

A befoglaló keretek azonban hierarchikus rendszerbe is szervezhetők, azaz a kisebb keretek magasabb szinteken nagyobb keretekbe foghatók össze. Ekkor a sugárkövetés során a befoglaló keretek által definiált hierarchiát járjuk be, ami szublineáris futási idejű algoritmusokhoz vezethet.

Az objektumtér szabályos felosztása

Tegyünk az objektumtérre egy szabályos (c_x, c_y, c_z) cellaméretű, a koordinátatengelyekkel párhuzamos oldalú rácsot (15.31. ábra), amit a teljes objektumteret befoglaló AABB felosztásával kapunk.



15.31. ábra. Az objektumtér szabályos felosztása. A sugárnak a rács egyes koordinátasíkjaival képzett metszéspontjai mindig c_x/v_x,c_y/v_y, illetve c_z/v_z távolságra vannak.

Az előfeldolgozás során minden cellára határozzuk meg a cellában lévő, vagy a cellába lógó objektumokat. Ehhez az egyes alakzat-cella párokra azt kell megvizsgálni, hogy a cella téglatestének és az alakzatnak van-e közös része. A vizsgálatot megoldhatjuk egy vágási algoritmus (15.4.3. pont) futtatásával is, vagy pedig egyszerűen úgy, hogy ellenőrizzük, hogy az alakzat koordinátatengelyekkel párhuzamos oldalú befoglaló téglatestének és a cellának van-e közös része. Ez az egyszerű módszer konzervatív, azaz akkor is belógónak minősíthet egy alakzatot, ha maga nem, csak a befoglaló doboza hatol a cellába. Ez persze a sugárkövetésnél nem okoz hibát, legfeljebb felesleges metszési kísérleteket.

SZABÁLYOS-RÁCS-FELÉPÍTÉS()

```
1  számítsuk ki a rács minimális sarkát (x_min, y_min, z_min) és cella méreteit (c_x, c_y, c_z)
2  for minden c cellára
3      do c cella objektumlistája ← üres
4          for minden o objektumra                ▷ A cellába lógó objektumok regisztrálása.
5              do if a c cella és az o objektum AABB-je egymásba lóg
6                  then a c cella objektumlistájához hozzáadjuk az o objektumot
```

A sugárkövetés fázisában a sugár által metszett cellákat a kezdőponttól való távolságuk sorrendjében látogatjuk meg. Egy cellánál csak azon objektumokat kell tesztelni, amelyeknek van közös része az adott cellával, azaz, amelyeket az előfeldolgozás során a cellában regisztráltunk. Másrészt, ha egy cellában az összes ide tartozó objektum tesztelése után megtaláljuk a legközelebbi metszéspontot, be is fejezhetjük a sugár követését, mert a többi cellában esetlegesen előforduló metszéspont biztosan a megtalált metszéspontunk mögött van. A cellába lógó objektumok metszése után azt is ellenőrizni kell, hogy a metszéspont is a cellában van-e, mert csak ilyen metszéspontokra jelenthetjük ki, hogy a további cellák metszéspontjait megelőzik. Előfordulhat, hogy egy objektummal egy későbbi cellában újból találkozunk. Sugár-felület metszéseket takaríthatunk meg, ha a korábbi számításokról nem feledkezünk meg, hanem a vizsgált objektumokhoz hozzárendeljük a korábbi metszésszámítás eredményét.

Amíg nincs metszéspont, a sugár által metszett cellákon megyünk végig. Az első cella X, Y, Z indexei a sugár $\vec{s}$ kezdőpontjából, az objektumokat befoglaló rács $(x_{min}, y_{min}, z_{min})$ sarokpontjából és a cellák (c_x, c_y, c_z) méreteiből számíthatók ki:

SZABÁLYOS-RÁCS-TARTALMAZÓ-CELLA($\vec{s}$)

```
1  X ← EGÉSZRÉSZ((s_x − x_min)/c_x)
2  Y ← EGÉSZRÉSZ((s_y − y_min)/c_y)
3  Z ← EGÉSZRÉSZ((s_z − z_min)/c_z)
4  return X, Y, Z
```

Ez az algoritmus feltételezi, hogy a sugár kezdőpontja is a rács által lefedett tartományban van. Ha ez a feltétel nem állna fenn, akkor ki kell számítani a sugár és a rácsot befoglaló doboz metszéspontját, és oda áthelyezni a sugár kezdőpontját.

A koordinátánkénti t_x, t_y, t_z sugárparaméterek kezdeti értéke a koordinátasíkokkal képzett első metszéspont lesz, melynek koordinátáit a SZABÁLYOS-RÁCS-SUGÁRPARAMÉTER-KEZDETI-ÉRTÉK algoritmus határozza meg.

A cellasorozat következő cellája egy ***3D szakaszrajzoló algoritmus*** (***3DDDA algoritmus***) segítségével állítható elő. Az algoritmus arra a felismerésre épít, hogy az x (és hasonlóképpen az y és a z) tengelyre merőleges síkok és a sugár metszéspontjaira érvényes sugárparaméterek mindig c_x/v_x távolságra (c_y/v_y, illetve c_z/v_z távolságra) vannak egymástól, tehát a következő metszésponthoz tartozó sugárparaméter egyetlen összeadással számítható (15.31. ábra). Az x, y és z síkokkal keletkező metszéspontot a t_x, t_y és t_z globális sugárparaméter változókban tartjuk nyilván, amelyeket mindig ugyanazzal az értékkel növelünk. A három sugárparaméterérték közül mindig az jelöli ki a tényleges következő cella metszéspontot, amelyik a legkisebb.

SZABÁLYOS-RÁCS-SUGÁRPARAMÉTER-KEZDETI-ÉRTÉK($\vec{s}, \vec{v}, X, Y, Z$)

```
1  if v_x > 0
2     then t_x ← (x_min + (X + 1) · c_x − s_x)/v_x
3     else if v_x < 0
4             then t_x ← (x_min + X · c_x − s_x)/v_x
5             else  t_x ← t_max                          ▷ A legnagyobb távolság.
6  if v_y > 0
7     then  t_y ← (y_min + (Y + 1) · c_y − s_y)/v_y
8     else if v_y < 0
9             then t_y ← (y_min + Y · c_y − s_y)/v_y
10            else  t_y ← t_max
11 if v_z > 0
12    then  t_z ← (z_min + (Z + 1) · c_z − s_z)/v_z
13    else if v_z < 0
14            then t_z ← (z_min + Z · c_z − s_z)/v_z
15            else  t_z ← t_max
16 return t_x, t_y, t_z
```

A következő cella X, Y, Z indexeit előállító és a t_x, t_y, t_z sugárparamétereket karbantartó eljárás:

SZABÁLYOS-RÁCS-KÖVETKEZŐ-CELLA(X, Y, Z, t_x, t_y, t_z)

```
1  if t_x = min(t_x, t_y, t_z)          ▷ A következő metszés az x tengelyre merőleges síkon.
2     then X ← X + sgn(v_x)             ▷ A sgn(x) az előjel (signum) függvény.
3          t_x ← t_x + c_x/|v_x|
4  else if t_y = min(t_x, t_y, t_z)     ▷ A következő metszés az y tengelyre merőleges síkon.
5          then Y ← Y + sgn(v_y)
6               t_y ← t_y + c_y/|v_y|
7  else if t_z = min(t_x, t_y, t_z)     ▷ A következő metszés a z tengelyre merőleges síkon.
8          then Z ← Z + sgn(v_z)
9               t_z ← t_z + c_z/|v_z|
```

A következőkben egy teljes sugárkövetési algoritmust mutatunk be, amely az előfeldolgozás során előállított szabályos rács adatstruktúra felhasználásával egyetlen sugárra megkeresi a legközelebbi sugár-felület metszéspontot. A koordinátánkénti sugárparaméterek minimuma, a t_{ki} változó határozza meg, hogy a cellában mekkora távolságot tehet meg a sugár. Ezt a paramétert használjuk annak eldöntésére, hogy a cellában regisztrált objektumok metszéspontja valóban a cellában következett-e be.

SUGÁR-ELSŐ-METSZÉSPONT-SZABÁLYOS-RÁCCSAL($\vec{s}, \vec{v}$)

```
1  (X, Y, Z) ← SZABÁLYOS-RÁCS-TARTALMAZÓ-CELLA(s⃗)
2  (t_x, t_y, t_z) ← SZABÁLYOS-RÁCS-SUGÁRPARAMÉTER-KEZDETI-ÉRTÉK(s⃗, v⃗, X, Y, Z)
```

```
3  while X, Y, Z a rács belsejében van
4        do t_ki ← min(t_x, t_y, t_z)                    ▷ A cellából itt lépünk ki.
5           t ← t_ki                                     ▷ Inicializálás: még nincs metszéspont.
6           for az (X, Y, Z) cellában regisztrált o objektumokra
7                 do t_o ←Sugár-felület-metszés(s⃗, v⃗, o)          ▷ Negatív, ha nincs.
8                    if 0 ≤ t_o < t                      ▷ Az új metszéspont közelebbi-e?
9                    then t ← t_o                        ▷ A legközelebbi metszés sugárparamétere.
10                        o_metszett ← o                 ▷ A legközelebb metszett objektum.
11                if t < t_ki                            ▷ Volt metszéspont a cellában?
12                  then x⃗ ← s⃗ + v⃗ · t                  ▷ A metszéspont helye a sugár egyenletéből.
13                       return t, x⃗, o_metszett         ▷ Nem kell továbblépni.
14          Szabályos-rács-következő-cella(X, Y, Z, t_x, t_y, t_z)   ▷ 3DDDA.
15 return „nincs metszéspont”
```

A szabályos felosztási algoritmus idő és tárbonyolultsága

A szabályos felosztási algoritmus előfeldolgozási lépése minden cellát minden objektummal összevet, így N objektumra és C cellára a futási ideje $\Theta(N \cdot C)$. A gyakorlatban a felosztó rács felbontását úgy választjuk meg, hogy C arányos legyen N-nel, mert ekkor az egy cellába eső objektumok várható száma nem függ az objektumok számától. Ilyen felosztás mellett az előfeldolgozási idő négyzetes, azaz $\Theta(N^2)$-es. Az objektumok előrendezésével megtakaríthatjuk az összes cella-objektum pár vizsgálatát, de ennek kisebb a jelentősége, hiszen általában nem az előfeldolgozás, hanem a sugárkövetés ideje kritikus. Mivel a legrosszabb esetben minden objektum minden cellába belóghat, a tárigény ugyancsak $O(N^2)$-es.

A sugárkövetés ideje a következő egyenlettel fejezhető ki:

$$T = T_o + N_I \cdot T_I + N_S \cdot T_S\,, \tag{15.26}$$

ahol T_o a sugár kezdőpontját tartalmazó cella azonosításához szükséges idő, N_I a legközelebbi metszéspont megtalálásáig végrehajtott metszési kísérletek száma, T_I egyetlen sugár–felület metszéspontszámítás ideje, N_S a meglátogatott cellák száma, T_S pedig következő cellára lépéshez szükséges idő.

Az első cella azonosításához a sugár kezdőpontjának koordinátáit a cellaméretekkel kell osztani, amiből egészre kerekítés után megkapjuk a cella sorszámát a három tengely mentén. Ez a lépés nyilván konstans időt igényel. Egyetlen sugár–felület metszés ugyancsak konstans idejű. A következő cellára lépés a 3DDDA algoritmusnak köszönhetően ismét állandó időt vesz igénybe. Az algoritmus bonyolultságát így a metszési kísérletek és a meglátogatott cellák száma határozza meg.

Egy szerencsétlen elrendezésben előfordulhat, hogy egy cellába az összes objektum belelóg, így a cellába lépő sugárral az összes objektumot meg kell próbálni elmetszeni. Így $O(N)$ metszéspont számításra van szükség, minek következtében a teljes algoritmus lineáris időbonyolultságú.

Eszerint a szabályos felosztás négyzetes előfeldolgozás és tárigény után is ugyanúgy lineáris futási idejű, mint a naiv megoldás. A valóságban azonban mégis érdemes használni, mert a legrosszabb esetek nagyon ritkán fordulnak elő. Arról van szó, hogy a klasszikus, legrosszabb esetre vonatkoztatott bonyolultságelemzés alkalmatlan a naiv sugárkövetés és a

szabályos felosztási módszer összehasonlítására. A megfelelő összehasonlításhoz az algoritmus valószínűségi elemzése szükséges, amelyhez a virtuális világ valószínűségi modelljét kell megalkotnunk.

A virtuális világ valószínűségi modellje

A valószínűségi modell felállításához a lehetséges objektumkonfigurációk valószínűségeit kell megadnunk. A modell nem lehet túlságosan bonyolult, hiszen ekkor a számítási idő várható értékét nem tudnánk kiszámítani. Egy lehetséges, a gyakorlati eseteket jól jellemző modell az alábbi: *Az objektumok r sugarú gömbök, amelyek középpontjai egyenletesen oszlanak el a térben.*

A bonyolultságelemzés az algoritmusok aszimptotikus viselkedését írja le egyre növekvő objektumszámot feltételezve. Ha az egyre több objektumot egy véges tartományban tartanánk, azok előbb-utóbb teljesen kitöltenék a rendelkezésre álló teret. Ezért a valószínűségi elemzés során feltételezzük, hogy a virtuális világunk által elfoglalt tértartomány az objektumok számával együtt nő, mialatt az objektumsűrűség állandó. Ez a valószínűségszámítás egy jól ismert módszere, amely az egyenletes eloszlásból határátmenettel egy Poisson-pontfolyamatot hoz létre.

15.16. definíció. *Egy $N(A)$* ***Poisson-pontfolyamat*** *a mintapontokat számlálja meg a tér A részhalmazaiban úgy, hogy*

- *$N(A)$ egy $\rho V(A)$ paraméterű Poisson-eloszlás, ahol ρ egy pozitív konstans, a folyamat intenzitása, $V(A)$ az A halmaz térfogata, így annak valószínűsége, hogy A éppen k mintapontot tartalmaz*

$$\Pr\{N(A) = k\} = \frac{(\rho V(A))^k}{k!} \cdot e^{-\rho V(A)},$$

  és egy $V(A)$ térfogatban található mintapontok számának várható értéke $\rho V(A)$,

- *$A_1, A_2, \ldots, A_n$ diszjunkt halmazokra az $N(A_1), N(A_2), \ldots, N(A_n)$ valószínűségi változók függetlenek.*

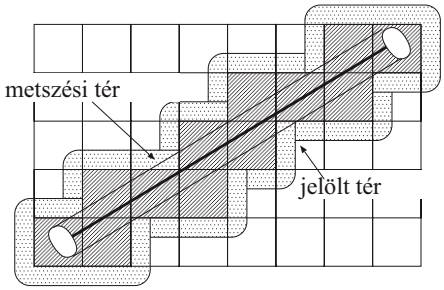
A Poisson-pontfolyamat alkalmazásával a virtuális világ valószínűségi modelljét úgy pontosítjuk, hogy az r sugarú gömbök középpontjait egy ρ intenzitású Poisson-pontfolyamat realizációinak tekintjük.

A metszési kísérletek számának várható értéke

A 15.32. ábrán egy sugarat látunk, amely áthalad a térfelbontó adatszerkezet celláin. Azon gömböket kell sugár metszésnek alávetni, amelyek belógnak valamelyik, a sugár által metszett cellába. A belógó gömbök középpontjainak halmazát ***jelölt térnek*** nevezzük.

A sugár-gömb metszési kísérlet csak abban az esetben lehet sikeres, ha a gömb középpont a sugár körüli r sugarú hengerben van. Ezt a hengert ***metszési térnek*** nevezzük (valójában a metszési térben a henger két végét egy-egy félgömb zárja le, de ezektől az egyszerűség kedvéért eltekintünk).

A sugárkövető algoritmus a sugár által metszett cellákat a sugár mentén egymás után járja be. Ha egy cella üres, ennél a cellánál nincs teendő. A nem üres cellához gömbök tartoznak, amelyekkel a metszési kísérletet el kell végezni. A továbbiakban feltesszük, hogy a térpartícionáló adatstruktúra felbontásához képest az objektumok sűrűsége alacsony, ezért eltekintünk azoktól az esetektől, amikor egy cellába több objektum is belógna.



15.32. ábra. A metszési tér és a jelölt tér egy r sugarú gömböket tartalmazó tér szabályos felosztásánál. A metszési tér egy r sugarú henger, melynek tengelye a sugár. A jelölt tér olyan r sugarú gömbök középpontjainak a halmaza, amelyek belelógnak valamely, a sugár által metszett cellába.

Az algoritmusnak meg kell kísérelnie a metszéspontszámítást minden olyan gömbre, amelynek középpontja a jelölt térben van, de csak a metszési térben lévő középponttal rendelkező gömbök vezetnek sikerhez. A siker s valószínűsége a nem üres cellához kapcsolódó metszési és jelölt terek sugárra merőleges vetületeinek a területaránya. Mivel a cellák mérete azonos, a siker valószínűsége állandó. Ha a metszés sikerét az egyes cellákban független valószínűségi változónak tekinthetjük, akkor annak valószínűsége, hogy az első, második, harmadik stb. nem üres cellában találunk először metszéspontot, rendre s, $(1-s)s$, $(1-s)^2 s$ stb. A metszési kísérletek várható száma ezen eloszlás várható értéke:

$$\mathrm{E}\left[N_I\right] = \frac{1}{s} \; . \tag{15.27}$$

Szabályos felosztásnál a cellák állandó c élhosszúságú kockák, ezért ahhoz, hogy a gömb a cellába lógjon, a középpontjának a $c + 2r$ élhosszúságú legömbölyített „kockában” kell lennie. Ha a sugár párhuzamos a kocka élével, a jelölt tér sugárra merőleges vetületének területe $c^2 + 4cr + r^2\pi$. A másik szélső esetben, amikor a sugár a cella átlójával párhuzamos, ez a terület $\sqrt{3}c^2 + 6cr + r^2\pi$. Mivel a metszési tér egy henger, amelynek a sugárra merőleges vetülete $r^2\pi$, a metszési kísérlet sikerének valószínűsége:

$$\frac{r^2\pi}{\sqrt{3}c^2 + 6cr + r^2\pi} \le s \le \frac{r^2\pi}{c^2 + 4cr + r^2\pi} \; .$$

A (15.27) egyenlet szerint, a metszési kísérletek várható száma ezen valószínűség reciproka:

$$\frac{1}{\pi}\left(\frac{c}{r}\right)^2 + \frac{4}{\pi}\frac{c}{r} + 1 \le \mathrm{E}\left[N_I\right] \le \frac{\sqrt{3}}{\pi}\left(\frac{c}{r}\right)^2 + \frac{6}{\pi}\frac{c}{r} + 1 \; . \tag{15.28}$$

Például, ha a cella élhossza és a gömbátmérő megegyező ($c = 2r$), akkor

$$3.54 < \mathrm{E}\left[N_I\right] < 7.03 \; .$$

Ezt az eredményt azon feltételezés mellett kaptuk, hogy az objektumok (gömbök) száma a végtelenhez tart. A várható metszési kísérletek száma viszont véges, az objektumszámtól független és viszonylag kicsiny konstans.

A cellalépések várható száma

A várható érték számításához a feltételes várható érték tételt fogjuk felhasználni. Egy alkalmas feltétel a metszéspont helye, azaz a metszéspontnál felvett t^* sugárparaméter. A feltételes várható érték tétel értelmében a cellalépések N_S számának várható értéke felírható a metszés helyére vonatkoztatott feltételes várható értéknek a feltétel valószínűségével súlyozott integráljaként:

$$\mathrm{E}[N_S] = \int_0^\infty \mathrm{E}[N_S | t^* = t] \cdot p_{t^*}(t)\, dt \ ,$$

ahol p_{t^*} a metszés t^* sugárparaméterének a valószínűségsűrűsége. Esetünkben a metszési tér egy henger, amelynek térfogata $r^2\pi t$. Annak valószínűsége, hogy a metszés egy adott t paraméternél korábban következik be, a Poisson-pontfolyamat definíciója alapján:

$$\Pr\{t^* < t\} = 1 - e^{-\rho r^2 \pi t} \ .$$

A valószínűségeloszlásból a sűrűségfüggvényt deriválással kaphatjuk meg:

$$p_{t^*}(t) = \rho r^2 \pi \cdot e^{-\rho r^2 \pi t} \ .$$

A valószínűségi modellben a szabályos felosztásnál minden cella c élű kocka, így egy t hosszú sugár által metszett cellák száma $E[N_S | t^* = t] \approx t/c + 1$ értékkel becsülhető. Ez a becslés a koordinátatengelyek valamelyikével párhuzamos sugarakra pontos (a becslési hiba legfeljebb 1), egyéb esetekben a cellák száma ennek az értéknek legfeljebb $\sqrt{3}$-szorosa lehet. A becslést a cellalépések számát kifejező integrálba helyettesítve:

$$\mathrm{E}[N_S] \approx \int_0^\infty \left(\frac{t}{c} + 1\right) \cdot \rho r^2 \pi \cdot e^{-\rho r^2 \pi t}\, dt = \frac{1}{c \rho r^2 \pi} + 1 \ . \tag{15.29}$$

Például, ha a cellaméret az objektummérettel összevethető ($c = 2r$), és a cellában lévő gömbközéppontok várható száma 0.1, akkor $\mathrm{E}[N_S] \approx 14$. Figyeljük meg, hogy a szabályos felosztás módszernél a cellalépések száma is konstans.

Várható futási idő és memóriaigény

Megállapítottuk, hogy a metszési kísérletek és a cellalépések számának várható értéke aszimptotikusan konstans, következésképpen a szabályos felosztást alkalmazó sugárkövetés négyzetes előfeldolgozási idő után, átlagos esetben konstans idő alatt megoldja a sugárkövetési feladatot. A futási idő tényleges értékét a (15.28) és (15.29) egyenletek alapján a c cellaméretttel szabályozhatjuk. A kis cellaméret csökkenti a metszési kísérletek számát, viszont növeli a cellalépések számát és a tárigényt, így megválasztása kompromisszum eredménye.

A valószínűségi modell szerint az egy cellába lógó objektumok várható száma is konstans, tehát a tárigény a cellák számával arányos. Ha a cellák számát az objektumszámmal arányosan választjuk meg, akkor a várható tárigény lineáris, szemben a legrosszabb eset négyzetes memóriaigényével.

A várható konstans, azaz aszimptotikusan az objektumok számától független futási idő és a lineáris tárigény a magyarázata annak, hogy a szabályos felosztás — és a következő pontok heurisztikus sugárkövető algoritmusai is — a gyakorlatban sokkal gyorsabbak a naiv sugárkövetésnél, holott a legrosszabb esetre vett bonyolultsági mértékeik nem jobbak, mint a naiv algoritmuséi.

Az oktális fa

A szabályos felosztás feleslegesen sok cellalépést igényel. Az üres térrészeket például nem érdemes felosztani, és két szomszédos cellát is elég lenne csak akkor szétválasztani, ha azokhoz az objektumok egy más halmaza tartozik. Ezt a felismerést teszik magukévá az adaptív felosztó algoritmusok. Az objektumtér adaptív felosztása rekurzív megközelítéssel és egy hierarchikus adatszerkezet felépítésével lehetséges. A hierarchikus szerkezet általában egy fa, az adaptív algoritmusokat pedig a fa típusa szerint osztályozzuk.

Az ebben a pontban tárgyalt adaptív módszer oktális (nyolcas) fát alkalmaz, amelyben egy nem üres csomópontnak 8 gyereke van. Az oktális fa építésének folyamata a következő:

- Kezdetben foglaljuk az objektumainkat egy-egy koordinátatengelyekkel párhuzamos oldalú dobozba, azaz AABB-be, majd határozzuk meg az objektumonkénti AABB-k közös, befoglaló AABB-jét is. Ez lesz az oktális fa gyökere, és egyben a rekurzió kiindulópontja, azaz az első feldolgozandó cella.
- Ha az aktuális cellában a belógó objektumok (vagy az objektumok befoglaló dobozainak) száma nagyobb, mint egy előre definiált érték, akkor a cellát a felezősíkjai mentén 8 egybevágó részcellára bontjuk. A hierarchikus adatszerkezetben a részcellákat a felbontott cella gyerekeiként vesszük fel, majd a keletkező részcellákra ugyanazt a lépést rekurzívan megismételjük.
- A gráfépítő folyamat egy adott szinten megáll, ha az adott cellához vezető út elér egy előre definiált maximális mélységét, azaz a cellaméret egy adott szint alá csökken, vagy az adott cellában az objektumok száma egy előre definiált érték alá esik.

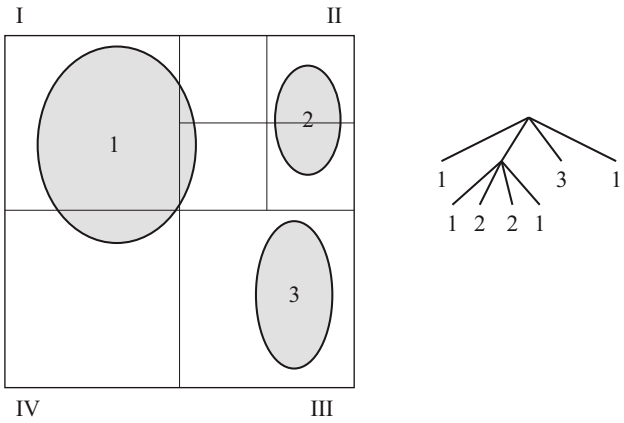
Az eljárás eredménye egy ***oktális fa*** (15.33. ábra). A fa levelei azon elemi cellák, amelyekhez a belógó objektumokat nyilvántartjuk.

A metszéspontszámítás során végig kell menni a fa azon elemi celláin – a fa levelein – amelyeket a sugár metsz és az itt regisztrált objektumok metszését kell ellenőrizni:

SUGÁR-ELSŐ-METSZÉSPONT-OKTÁLIS-FÁVAL($\vec{s}, \vec{v}$)

```
1  q⃗ ← a sugár metszéspontja az objektumteret befoglaló AABB-vel
2  while q⃗ az objektumteret befoglaló AABB-ben van          ▷ Végigmegy a cellákon.
3        cella ← CELLA-KERESÉS-OKTÁLIS-FÁBAN(oktális fa gyökere , q⃗)
4        t_ki ← a cella és sugár metszéspontjához tartozó sugárparaméter
5        t ← t_ki                                  ▷ Inicializálás: még nincs metszéspont.
6        for a cella listájának o objektumaira
7            do t_o ←SUGÁR-FELÜLET-METSZÉS(s⃗, v⃗)                ▷ Negatív, ha nincs.
8               if 0 ≤ t_o < t                          ▷ Az új metszéspont közelebbi-e?
9                  then t ← t_o                  ▷ A legközelebbi metszés sugárparamétere.
10                      o_metszett ← o                 ▷ A legközelebb metszett objektum.
11       if t < t_ki                                    ▷ Volt metszéspont a cellában ?
12          then x⃗ ← s⃗ + v⃗ · t                  ▷ A metszéspont helye a sugár egyenletéből.
13               return t, x⃗, o_metszett
14       q⃗ ← s⃗ + v⃗ · (t_ki + ε)      ▷ A sugár következő cellában lévő legközelebbi pontja.
15 return „nincs metszéspont”
```

A szabályos felosztáshoz képest most a kezdő és következő cella megtalálása nehezebb. A kezdőcellát megkereső CELLA-KERESÉS-OKTÁLIS-FÁBAN eljárás az adatstruktúra bejárásával



15.33. ábra. A síkot felosztó négyes fa, amelynek a háromdimenziós változata az oktális fa. A felépítés során a cella oldalainak a felezését addig folytatjuk, amíg egy cellába „kevés” objektum jut, vagy a cellaméret egy megadott minimális érték alá csökken. A fa leveleiben regisztráljuk a belógó objektumokat.

arra ad választ, hogy egy pont melyik cellához tartozik. A ponttal a fa csúcsán belépünk az adatstruktúrába. A pont koordinátáit a felosztási feltétellel (oktális fánál a cella középpontjának koordinátáival) összehasonlítva eldönthetjük, hogy a 8 lehetőség közül melyik úton kell folytatni az adatszerkezet bejárását. Ugyanezt a vizsgálatot rekurzívan ismételgetve előbb-utóbb eljutunk egy levélig, azaz megtaláljuk a pontot tartalmazó elemi cellát.

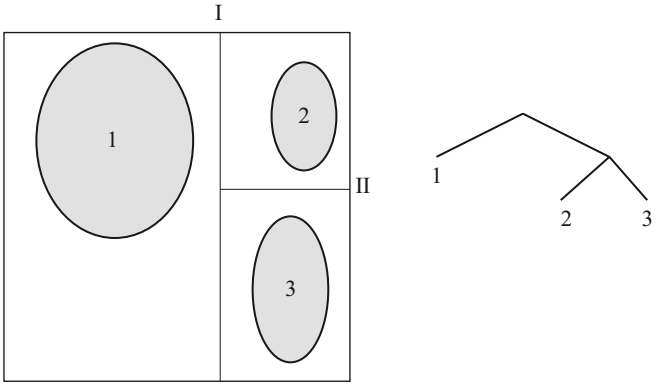
A következő cella azonosításához az aktuális cellában számítsuk ki a sugár kilépési pontját, azaz a sugárnak és a cellának a metszéspontját, majd adjunk hozzá a metszéspont t_{ki} sugárparaméteréhez egy „kicsit” (a SUGÁR-ELSŐ-METSZÉSPONT-OKTÁLIS-FÁVAL algoritmusban ε-t). A kicsivel továbblendített sugárparamétert visszahelyettesítve a sugáregyenletbe, egy, a következő cellában lévő pontot (az algoritmusban a $\vec{q}$ pontot) kapunk. A cellát a pont alapján ismét a CELLA-KERESÉS-OKTÁLIS-FÁBAN algoritmussal azonosíthatjuk.

Az oktális fa cellái nagyobbak lehetnek, mint a megengedett minimális cella, így kevesebb cellalépést igényelnek, mint a szabályos felosztás. Ugyanakkor a nagyobb cellák csökkentik a metszési kísérlet sikerének valószínűségét, mert kisebb eséllyel fordul elő, hogy a cellát metsző véletlen sugár a cellába lógó alakzatot is metszi. A kisebb metszési sikerhez viszont várhatóan több metszési kísérlet tartozik, ami kedvezőtlen. Eszerint az oktális fa felépítésénél érdemes a nem üres cellák méretét mindig a minimális méretre zsugorítani, még akkor is, ha csak egyetlen objektumot tartalmaznak. Ilyen stratégia mellett viszont a nem üres cellák most is azonos méretűek, tehát a szabályos hálónál a metszési kísérletek várható számára kapott eredmények most is alkalmazhatók. Mivel a siker valószínűsége a nem üres cellák méretétől függ, a metszéspontok számát ismét a (15.28) egyenlőtlenséggel adhatjuk meg. Ez azt is jelenti, hogy ha a minimális cellák mérete a szabályos rács cellaméretével megegyező, akkor a szabályos rácsban és az oktális fában a metszési kísérletek száma is hasonló lesz. Az üres térrészek átugrása viszont az oktális fa előnyeként könyvelhető el. A cellalépések számának várható értéke tehát várhatóan továbbra is konstans, de a szabályos felosztásénál kisebb. Az oktális fa hátránya ellenben, hogy a következő cellát nem lehet konstans idejű algoritmussal meghatározni. Mint láttuk, a következő cella azonosítása az oktális fa bejárását igényli. Ha az oktális fát addig építjük, amíg egy cella csak konstans

számú objektumot tartalmaz, akkor a cellák száma az objektumok számával arányos. Így a fa mélysége és a következő cella azonosításának ideje $O(\lg N)$ nagyságrendű.

A kd-fa

Az oktális fa adaptálódik az objektumok elhelyezkedéséhez. A felbontás azonban mindig felezi a cellaoldalakat, tehát nem veszi figyelembe, hogy az objektumok hol helyezkednek el, így az adaptivitás nem tökéletes. Tekintsünk egy olyan felosztást, amely egy lépésben nem mind a három felezősík mentén vág, hanem egy olyan síkkal, amely az objektumteret a lehető legigazságosabban felezi meg. Ez a módszer egy bináris fához vezet, amelynek neve ***bináris térpartícionáló fa,*** vagy ***BSP-fa.*** Ha a felezősík mindig merőleges a koordinátarendszer valamely tengelyére, akkor ***kd-fa*** adatszerkezetről beszélünk.



15.34. ábra. A kd-fa. A „sok” objektumot tartalmazó cellát rekurzívan egy valamely koordinátatengelyre merőleges síkkal két cellára bontjuk.

A kd-fában a felezősíkot többféleképpen elhelyezhetjük:

- a ***térbeli középvonal módszer*** a befoglaló keretet mindig két egyforma részre osztja.
- a ***test középvonal módszer*** úgy osztja fel a teret, hogy annak bal és jobb oldalán egyforma számú test legyen.
- a ***költségvezérelt módszer*** becsli azt az átlagos időt, amelyet egy sugár a kd-fa bejárása során felhasznál, és ennek minimalizálására törekszik. Egy megfelelő költségmodell szerint úgy felezzük a cellát, hogy a metszés ugyanakkora valószínűséggel következzen be a gyerek cellákban.

A metszési valószínűség kiszámításához az ***integrálgeometria*** egyik alapvető tételét alkalmazhatjuk:

15.17. tétel. *Ha egy konvex A test tartalmaz egy ugyancsak konvex B testet, akkor annak valószínűsége, hogy egy egyenletes eloszlású véletlen egyenes metszi a B testet feltéve, hogy metszette az A-t, egyenlő a B és az A testek felületeinek arányával.*

A következőkben egy általános kd-fa építő rekurzív algoritmust mutatunk be. A *cella* paraméter az aktuális cellát, a *mélység* a rekurzió mélységét, a *koordináta* pedig az aktuális vágósík orientációját jelenti. A *cella*-hoz a két gyerekcella (*cella.jobb*, illetve *cella.bal*),

és a bal-alsó-közeli, illetve jobb-felső-távoli sarokpontok (*cella.min*, illetve *cella.max*) tartoznak. A cellákhoz rendeljük hozzá a belógó objektumok listáját. A felezősík irányát a fa építésekor a mélység növekedésével a KÖVETKEZŐ-KOORDINÁTA függvény ciklikusan változtathatja ($x, y, z, x, y, z, x, \ldots$). A következő rekurzív eljárás első hívásakor a *cella* a teljes objektumteret tartalmazó AABB, a *mélység* pedig zérus:

KD-FA-FELÉPÍTÉS(*cella*, *mélység*, *koordináta*)

```
 1  if a cella-ba lógó objektumok száma kevés vagy a mélység nagy
 2     then return
 3  cella.bal és cella.jobb befoglalódoboza ← cella befoglalódoboza
 4  if koordináta = x
 5     then cella.jobb.min.x ← cella felezősíkja x irányban
 6          cella.bal.max.x ← cella felezősíkja x irányban
 7     else if koordináta = y
 8             then cella.jobb.min.y ← cella felezősíkja y irányban
 9                  cella.bal.max.y ← cella felezősíkja y irányban
10     else if koordináta = z
11             then cella.jobb.min.z ← cella felezősíkja z irányban
12                  cella.bal.max.z ← cella felezősíkja z irányban
13  for a cella o objektumaira
14       do if az o objektum a cella.bal befoglaló dobozában van
15             then adjuk az o objektumot a cella.bal listájához
16          if az o objektum a cella.jobb befoglaló dobozában van
17             then adjuk az o objektumot a cella.jobb listájához
18  KD-FA-FELÉPÍTÉS(cella.bal, mélység + 1, KÖVETKEZŐ-KOORDINÁTA(koordináta))
19  KD-FA-FELÉPÍTÉS(cella.jobb, mélység + 1, KÖVETKEZŐ-KOORDINÁTA(koordináta))
```

A kd-fa felépítése után egy olyan algoritmusra is szükségünk van, amely egy adott sugárra megmondja annak útját a fában, és meghatározza a sugár által elsőként metszett testet is. Legelső lépésként a kezdőpontot kell meghatározni a sugár mentén, ami vagy a sugár kezdőpontja, vagy pedig az a pont, ahol a sugár belép a befoglaló keretbe[6]. A pont helyzetének meghatározása során azt a cellát kell megtalálnunk, amelyben az adott pont van. A ponttal a fa csúcsán belépünk az adatstruktúrába. Az adott pont koordinátáit a felezősík koordinátájával összehasonlítva eldönthetjük, hogy melyik úton kell folytatni az adatszerkezet bejárását. Előbb-utóbb eljutunk egy levélig, azaz azonosítjuk a pontot tartalmazó elemi cellát. Ha ez a cella nem üres, akkor megkeressük a sugár és a cellában lévő, illetve a cellába belógó testek metszéspontját. A metszéspontok közül azt választjuk ki, amelyik a legközelebb van a sugár kezdőpontjához. Ezután ellenőrizzük, hogy a metszéspont a vizsgált cellában van-e (mivel egy test több cellába is átlóghat, előfordulhat, hogy nem ez a helyzet). Ha a metszéspont az adott cellában van, akkor megtaláltuk az első metszéspontot, így befejezhetjük az algoritmust. Ha a cella üres, vagy nem találtunk metszéspontot, esetleg a metszéspont nem a cellán belül van, akkor tovább kell lépnünk a következő cellára. Ehhez a sugár azon pontját határozzuk meg, ahol elhagyja a cellát. Ezután a kilépés sugárparaméterét (és ezzel a kilé-

[6] Attól függően, hogy a sugár kezdőpontja az összes test közös befoglaló dobozán belül van-e vagy sem.

pési pontot) egy „kicsit” előre toljuk, hogy egy, a következő cellában lévő pontot kapjunk. Innentől az algoritmus a tárgyalt lépéseket ismétli.

Ennek az algoritmusnak hátránya, hogy mindig a fa gyökerétől indul, pedig valószínűsíthető, hogy két egymás után következő cella esetén a gyökérből indulva részben ugyanazon cellákat járjuk be. Ebből adódóan a fa egy csúcsát többször is meglátogatjuk.

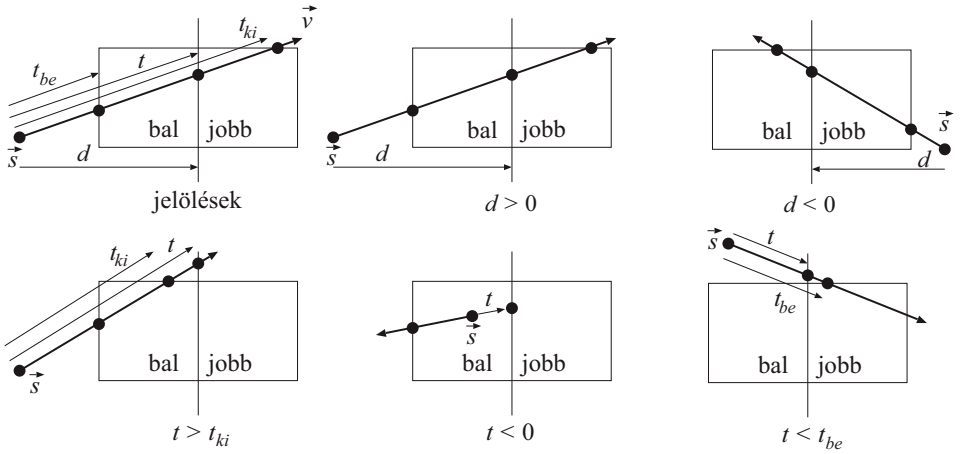
Ezt a hátrányt úgy küszöbölhetjük ki, hogy a meglátogatandó cellákat egy veremtárba tesszük, és mindig csak addig lépünk vissza, amíg szükséges. Így minden belső csúcsot és levelet csak egyszer látogatunk meg. Amikor a sugár egy olyan belső csúcshoz ér, amelynek két gyerekcsúcsa van, eldöntjük, hogy a gyerekeket milyen sorrendben dolgozzuk fel. A gyerekcsomópontokat „közeli” és „távoli” gyerekcsomópontként osztályozzuk aszerint, hogy sugár kezdete a felezősík melyik oldalán van. Ha a sugár csak a „közeli” gyerekcsomóponton halad keresztül, akkor az algoritmus csak ezt a csomópontot dolgozza fel. Ha a sugárnak mindkét gyerekcsomópontot meg kell látogatnia, akkor az algoritmus egy veremtárban megjegyzi az információkat a „távoli” gyerekcsomópontról, és a „közeli” csomópont irányába mozdul el. Ha a „közeli” csomópont irányában nem találunk metszéspontot, akkor a veremből a következő feldolgozásra váró csomópontot vesszük elő.

A kd-fa bejárásával a legközelebbi metszéspontot kiszámító algoritmus, amelynek jelöléseit a 15.35. ábrán követhetjük végig:

SUGÁR-ELSŐ-METSZÉSPONT-KD-FÁVAL($gyökér, \vec{s}, \vec{v}$)

```
1  (t_be, t_ki) ← SUGÁR-AABB-METSZÉS(s, v, gyökér)        ▷ Metszés a tér-AABB-jével.
2  if nincs metszéspont
3     then „nincs metszéspont”
4  VEREMBE(gyökér, t_be, t_ki)
5  while a verem nem üres                                  ▷ Amíg a fát be nem jártuk.
6        do VEREMBŐL(cella, t_be, t_ki)
7           while cella nem levél
8                 do koordináta ← a cella felezősíkjának orientációja
9                    d ← cella.jobb.min[koordináta] − s[koordináta]
10                   t ← d/v[koordináta]                  ▷ Felezősík metszés sugárparamétere.
11                   if d > 0                             ▷ s a felezősík bal vagy jobb oldalán van?
12                      then (közeli, távoli) ← (cella.bal, cella.jobb)   ▷ Bal.
13                      else (közeli, távoli) ← (cella.jobb, cella.bal)   ▷ Jobb.
14                   if t > t_ki vagy t < 0
15                      then cella ← közeli               ▷ Csak a közeli cellát metszi.
16                      else if t < t_be
17                              then cella ← távoli       ▷ Csak a távoli cellát metszi.
18                              else VEREMBE(távoli, t, t_ki)   ▷ Mindkét cellát metszi.
19                                   cella ← közeli       ▷ Először a közeli-t.
20                                   t_ki ← t             ▷ A sugár t-nél lép ki a közeli cellából.
           ▷ Ha az aktuális cella egy levél.
```

```
21        t ← t_ki                                  ▷ Maximális sugárparaméter a cellában.
22        for (a cella listájának o objektumaira)
23            do t_o ←SUGÁR-FELÜLET-METSZÉS(s, v)    ▷ Negatív, ha nincs.
24               if t_be ≤ t_o < t                  ▷ Az új metszéspont közelebbi-e?
25               then t ← t_o                       ▷ A legközelebbi metszés sugárparamétere.
26                    o_metszett ← o                ▷ A legközelebb metszett objektum.
27        if t < t_ki                               ▷ Volt metszéspont a cellában?
28          then x ← s + v · t                      ▷ A metszéspont helye a sugár egyenletéből.
29               return t, x, o_metszett            ▷ A metszéspontot megtaláltuk.
30  return -1                                       ▷ Nincs metszéspont.
```

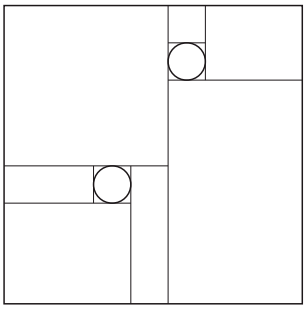


15.35. ábra. A SUGÁR-ELSŐ-METSZÉSPONT-KD-FÁVAL algoritmus jelölései és különböző esetei. A t_{be} a belépés, a t_{ki} a kilépés, t pedig a felezősík metszés sugárparamétere. d a sugár kezdőpont és a felezősík előjeles távolsága.

Az oktális fához hasonlóan a metszési siker valószínűsége a kd-fában is növelhető, ha a felosztást addig folytatjuk, amíg az objektumok körül minden levágható üres térrészt eltávolítunk (15.36. ábra).

A valószínűségi elemzéshez használt világmodellünkben azonos méretű r sugarú gömbök találhatók, így a nem üres cellák ismét kockák lesznek, amelyek élhosszúsága $c = 2r$. A szabályos ráccsal és az oktális fával ellentétben a kd-fa felosztó síkjai nem függetlenek az objektumoktól, hanem azok szélső pontjaira illeszkednek. Így nem kell a kívülről belógó gömbökkel foglalkozni, mert a cellahatárok a gömböt teljesen körülveszik. A metszési valószínűség pontos értékét a 15.17. tétel felhasználásával számíthatjuk ki. A jelenlegi esetben a tartalmazó A konvex test egy $2r$ oldalú kocka, a tartalmazott B konvex test pedig egy r sugarú gömb, így a metszési valószínűség:

$$s = \frac{4r^2\pi}{6a^2} = \frac{\pi}{6}\,.$$



15.36. ábra. Kd-fa alapú térpartícionálás az üres terek levágásával.

A metszési kísérletek várható száma tehát:

$$\mathrm{E}[N_I] = \frac{6}{\pi} \approx 1.91 \ .$$

A valószínűségi modell szerint a kd-fa igényli a legkevesebb sugár–felület metszési kísérletet.

Gyakorlatok

15.6-1. Bizonyítsuk be, hogy a valószínűségi modellben minden olyan sugárkövető algoritmus konstans időben fut, amely az objektumokat a sugár kezdőpontjától számított távolság sorrendjében dolgozza fel.

15.6-2. Készítsünk sugár-felosztott felület metszéspontszámító algoritmust.

15.6-3. Készítsünk sugár-B-spline felület metszéspontszámító algoritmust.

15.6-4. Készítsünk metszéspontszámító algoritmust CSG modellekhez – feltételezve, hogy a primitív testekre a metszéspontszámítás már működik.

15.6-5. Készítsünk metszéspontszámító algoritmust transzformált testekhez feltételezve, hogy az eredeti testre a metszéspontszámítás már működik (segítség: transzformáljuk a sugarat).

15.7. Az inkrementális képszintézis algoritmusai

A képszintézis a virtuális világot a virtuális kamera képpontjaira képezi le, melynek során takarási és színszámítási feladatokat kell megoldani. A sugárkövetés a számításokat képpontonként egymástól függetlenül hajtja végre, azaz nem használja fel újra az egyszer már nagy nehezen megszerzett láthatósági és színinformációkat. A jelen alfejezet inkrementális képszintézis algoritmusai néhány egyszerű elv alkalmazásával az alapfeladatok végrehajtási idejét jelentősen lerövidíthetik:

1. A feladatok egy részének elvégzése során elvonatkoztatnak a pixelektől, és az objektumtér nagyobb részeit egységesen kezelik.
2. Ahol csak lehet, kihasználják az ***inkrementális elv*** nyújtotta lehetőségeket. Az inkrementális elv alkalmazása azt jelenti, hogy egy pixel takarási és árnyalási információinak meghatározása során jelentős számítási munkát takaríthatunk meg, ha a szomszédos pixel hasonló adataiból indulunk ki, és nem kezdjük a számításokat elölről.

3. Minden műveletet a hozzá optimálisan illeszkedő koordinátarendszerben végeznek el, azok között pedig homogén lineáris geometriai transzformációkkal váltanak.
4. Feleslegesen nem számolnak, ezért a *vágás* során eltávolítják azon geometriai elemeket, amelyek a képen nem jelennének meg.

A transzformáció és a vágás általában megváltoztatja az alakzatok jellegét, kivéve a pontokat, szakaszokat és sokszögeket[7]. Ezért a modellben levő szabad formájú elemeket a képszintézis megkezdése előtt pontokkal, szakaszokkal és sokszögekkel közelítjük (15.3. szakasz).

Az inkrementális képszintézis lépéseit a 15.37. ábrán láthatjuk. A virtuális világ objektumait azok egy referenciahelyzetében adjuk meg, a további műveletek miatt sokszögekkel közelítjük a felületeket, majd a modellezési transzformációval a virtuális világ koordinátarendszerébe helyezzük el őket. A virtuális világot a kamera szempontjából kell lefényképezni. Ehhez el kell dönteni, hogy az objektumok hogyan takarják egymást, és csak a látható objektumokat kell megjeleníteni. Ezen műveleteket közvetlenül a virtuális világ koordinátarendszerében is el tudnánk végezni, azonban ekkor egy pont vetítése egy általános helyzetű egyenes és az ablak metszéspontjának kiszámítását igényelné, a takarás pedig az általános pozíciójú szemtől való távolsággal dolgozna. A takarási számításokat egyszerűsíthetjük, ha áttranszformáljuk a teljes objektumteret egy olyan koordinátarendszerbe, ahol a vetítés és a takarás triviálissá válik. Ezt a rendszert ***képernyő-koordinátarendszernek*** nevezzük, amelyben az X, Y koordináták azon pixelt jelölik ki, amelyre a pont vetül, a Z koordináta alapján pedig eldönthetjük, hogy két pont közül melyik van a szemhez közelebb. A képernyő-koordinátarendszerben tehát az X, Y tengelyek egységei éppen a pixelek. Mivel általában nem érdemes a képet a pixel méreténél pontosabban kiszámítani, az X, Y koordináták egészek. Hatékonysági okokból a Z koordináta is gyakran egész. A képernyő-koordinátarendszer koordinátáit a továbbiakban nagy betűvel jelöljük.

A képernyő-koordinátarendszerbe átvivő transzformációt egy koordinátarendszereken átvezető transzformáció sorozattal definiáljuk, amelynek elemeit külön tárgyaljuk. A tényleges transzformációt viszont egyetlen 4×4-es mátrixszorzással valósítjuk meg, ahol a transzformációs mátrix az elemi transzformációk mátrixainak a szorzata.

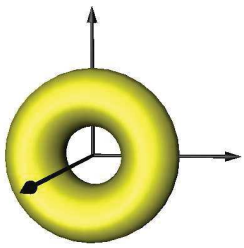
15.7.1. A kamera transzformáció

A képszintézis során általában egy kameraállásból látható látványra vagyunk kíváncsiak, ahol a ***szempozíció*** határozza meg a kamera helyét ($\vec{eye}$), irányát pedig a nézeti célpont ($\vec{lookat}$) és az $\vec{eye}$ vektor különbsége definiálja (15.38. ábra). Az $\vec{up}$ egységvektor a kamera függőleges irányát adja meg.

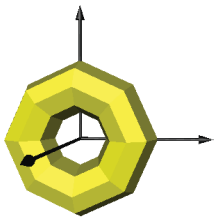
A *fov* a kamera függőleges irányú látószögét, az *aspect* az ablak szélességének és magasságának arányát, az f_p és a b_p pedig az úgynevezett első és hátsó vágósík szemtől mért távolságát jelenti. A vágósíkok segítségével figyelmen kívül hagyhatjuk azokat az objektumokat, amelyek a szem mögött, vagy a szemhez túl közel, vagy éppenséggel a szemtől túl távol helyezkednek el.

A kamerához egy koordinátarendszert, azaz három egymásra merőleges egységvektort rendelünk. Az $\vec{u} = (u_x, u_y, u_z)$ vízszintes, a $\vec{v} = (v_x, v_y, v_z)$ függőleges és a $\vec{w} = (w_x, w_y, w_z)$

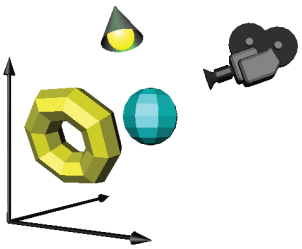
[7] Bár a Bézier és B-Spline görbék és felületek az affin transzformációkra, a NURBS pedig még a homogén lineáris transzformációkra is invariáns, de a vágás ezen görbék és felületek típusát is megváltoztatja.



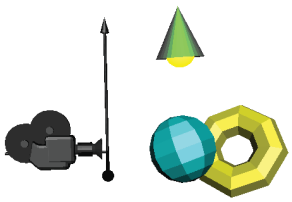
(a) Modellezés



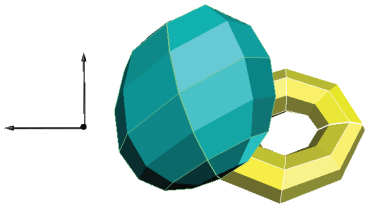
(b) Tesszelláció



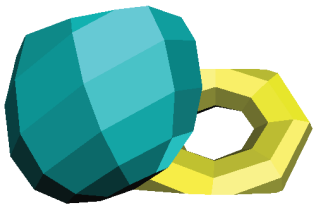
(c) Modellezési transzformáció



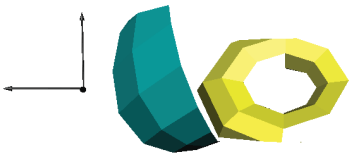
(d) Kamera transzformáció



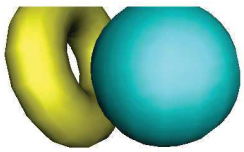
(e) Perspektív transzformáció



(f) Vágás

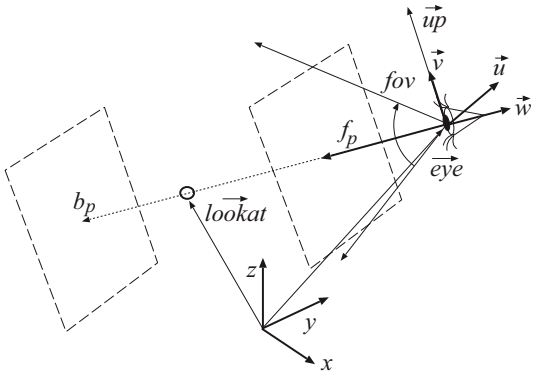


(g) Takarás



(h) Vetítés és színezés

15.37. ábra. Az inkrementális képszintézis lépései. **(a)** A modellezés során a tárgyakat egy referenciahelyzetükben adjuk meg. **(b)** A tárgyakat a további műveletek miatt tesszelláljuk. **(c)** A modellezési transzformáció a virtuális világbeli helyzetébe viszi a tárgyat. **(d)** A kamera transzformáció a világot eltolja és elforgatja úgy, hogy a kamera az origóba kerüljön és a $-z$ irányba nézzen. **(e)** A perspektív transzformáció a kamerában találkozó vetítősugarakból párhuzamosokat csinál, azaz a kamerát egy ideális pontra képezi le. **(f)** A vágás eltávolítja azokat a felületelemeket, amelyek nem vetülhetnek az ablakra. **(g)** A takarás során megszabadulunk azoktól a felületi pontoktól, amelyeket más felületek takarnak a kamera irányából. **(h)** Végül vetítjük a látható sokszögeket, és kitöltjük a vetületüket.



15.38. ábra. A virtuális kamera paraméterei: az $\vec{eye}$ szempozíció, a $\vec{lookat}$ nézeti célpont, az $\vec{up}$ függőleges irány, amelyekből a kamera $\vec{u}, \vec{v}, \vec{w}$ bázisvektorait számítjuk, valamint az f_p, b_p vágósík távolságok, és a *fov* függőleges látószög (a vízszintes látószöget az *aspect* arányból számítjuk).

nézeti irányba mutató egységvektorokat a következő módon határozhatjuk meg:

$$\vec{w} = \frac{\vec{eye} - \vec{lookat}}{|\vec{eye} - \vec{lookat}|}, \qquad \vec{u} = \frac{\vec{up} \times \vec{w}}{|\vec{up} \times \vec{w}|}, \qquad \vec{v} = \vec{w} \times \vec{u}\ .$$

A ***kamera transzformáció*** a virtuális teret a kamerával és a testekkel együtt úgy forgatja és úgy tolja el, hogy a transzformáció után a szem az origóba kerüljön, a $-z$ irányba nézzen, és a kamera függőleges iránya az y tengellyel essen egybe, azaz az $\vec{u}, \vec{v}, \vec{w}$ egységvektorokat a világkoordinátarendszer bázisvektoraiba viszi át. A $\mathbf{T}_{kamera}$ transzformációs mátrixot a szempozíciót az origóba vivő eltolás mátrixának és az $\vec{u}, \vec{v}, \vec{w}$ egységvektorokat a bázisvektorokkal fedésbe hozó forgatás mátrixának a szorzataként írhatjuk fel:

$$[x', y', z', 1] = [x, y, z, 1] \cdot \mathbf{T}_{kamera} = [x, y, z, 1] \cdot \mathbf{T}_{eltol} \cdot \mathbf{T}_{forgat}\ , \tag{15.30}$$

ahol

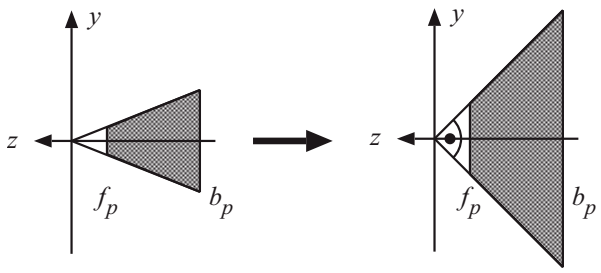
$$\mathbf{T}_{eltol} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -eye_x & -eye_y & -eye_z & 1 \end{bmatrix}, \qquad \mathbf{T}_{forgat} = \begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Vegyük észre, hogy a forgatás mátrixának oszlopai éppen az $\vec{u}, \vec{v}, \vec{w}$ vektorok koordinátái. Mivel ezek a vektorok egymásra merőlegesek, könnyen látható, hogy a forgatás után éppen az x, y, z tengelyekkel kerülnek fedésbe. Például az $\vec{u}$ elforgatottja:

$$[u_x, u_y, u_z, 1] \cdot \mathbf{T}_{forgat} = [\vec{u} \cdot \vec{u}, \vec{u} \cdot \vec{v}, \vec{u} \cdot \vec{w}, 1] = [1, 0, 0, 1].$$

15.7.2. A normalizáló transzformáció

A további műveletekhez normalizáljuk a látható pontokat tartalmazó, a szem és az ablak által meghatározott gúlát oly módon, hogy a gúla csúcsában a nyílásszög 90 fok legyen.



15.39. ábra. A normalizáló transzformáció a látószöget 90 fokra állítja.

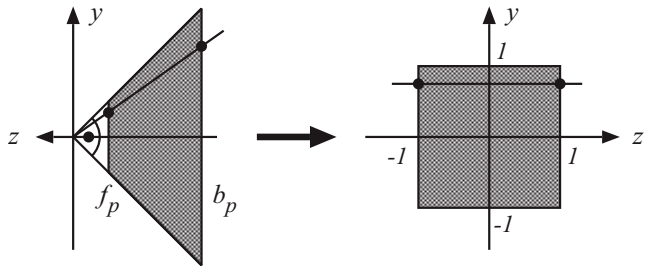
A normalizálás egy egyszerű skálázás:

$$\mathbf{T}_{norm} = \begin{bmatrix} 1/(\tan(fov/2)\cdot aspect) & 0 & 0 & 0 \\ 0 & 1/\tan(fov/2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

15.7.3. A perspektív transzformáció

A perspektív transzformációval a virtuális világot úgy torzítjuk, hogy a szemben találkozó vetítő sugarak párhuzamosak legyenek egymással, azaz a középpontos vetítést párhuzamos vetítéssel helyettesítjük.

A normalizáló transzformáció után a képszintézisben résztvevő pontok tartománya egy szimmetrikus csonka gúla (15.39. ábra). A perspektív transzformáció a csonka gúlát egy kockára képezi le, azaz az origóban találkozó vetítősugarakból egymással és a z tengellyel párhuzamos sugarakat hoz létre (15.40. ábra).



15.40. ábra. A perspektív transzformáció az első és hátsó vágósíkok és az ablak oldalaival leírt, csonka gúla alakú látható tartományt egy origó középpontú, 2 egység oldalú kockába viszi át.

A perspektív transzformációnak pontot pontba, egyenest egyenesbe kell átvinnie, ám a gúla csúcsát, azaz a szempozíciót, a végtelenbe kell elhelyeznie. Ez azt jelenti, hogy a perspektív transzformáció nem lehet az euklideszi tér lineáris transzformációja. Szerencsére a homogén lineáris transzformációkra is igaz az, hogy pontot pontba, egyenest egyenesbe visznek át, viszont képesek az ideális pontokat is kezelni. Ezért keressük a perspektív transz-

formációt a homogén lineáris transzformációk között a következő alakban:

$$\mathbf{T}_{persp} = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{bmatrix} .$$

A 15.40. ábrán berajzoltunk egy egyenest (vetítősugarat) és annak a transzformáltját. Jelöljük m_x-szel és m_y-nal az egyenes x/z, illetve y/z meredekségét. A normalizált nézeti gúlában a $[-m_x \cdot z, -m_y \cdot z, z]$ egyenesből a transzformáció után egy, az $[m_x, m_y, 0]$ ponton átmenő, z-tengellyel párhuzamos („vízszintes") egyenest kapunk. Vizsgáljuk meg ezen egyenes vágósíkokkal való metszéspontjait, azaz a z helyébe helyettesítsük a $(-f_p)$-t, illetve a $(-b_p)$-t. Ekkor az $[m_x, m_y, -1]$, illetve az $[m_x, m_y, 1]$ transzformált pontokhoz jutunk.

Az eddigiek alapján írjuk fel a perspektív transzformációt például az első vágósíkon levő metszéspontra:

$$\left[m_x \cdot f_p, m_y \cdot f_p, -f_p, 1\right] \cdot \mathbf{T}_{persp} = \left[m_x, m_y, -1, 1\right] \cdot \lambda ,$$

ahol λ tetszőleges, nem zérus szám lehet, hisz a homogén koordinátákkal leírt pont nem változik, ha a koordinátákat egy nem zérus konstanssal megszorozzuk. A λ konstanst f_p-nek választva:

$$\left[m_x \cdot f_p, m_y \cdot f_p, -f_p, 1\right] \cdot \mathbf{T}_{persp} = \left[m_x \cdot f_p, m_y \cdot f_p, -f_p, f_p\right] . \qquad (15.31)$$

Vegyük észre, hogy a transzformált pont első koordinátája megegyezik a metszéspont első koordinátájával tetszőleges m_x, m_y és f_p esetén. Ez csak úgy lehetséges, ha a $\mathbf{T}_{persp}$ mátrix első oszlopa $[1, 0, 0, 0]^T$. Hasonló okokból következik, hogy a mátrix második oszlopa $[0, 1, 0, 0]^T$. Ráadásul a (15.31) egyenletben jól látszik, hogy a vetített pont harmadik és negyedik koordinátájára a metszéspont első két koordinátája nem hat, ezért $t_{13} = t_{14} = t_{23} = t_{24} = 0$. A harmadik és a negyedik homogén koordinátára a következő egyenleteket állíthatjuk fel:

$$-f_p \cdot t_{33} + t_{43} = -f_p, \qquad -f_p \cdot t_{34} + t_{44} = f_p .$$

Az egyenes hátsó vágósíkkal vett metszéspontjára ugyanezt a gondolatmenetet alkalmazva két újabb egyenletet kapunk:

$$-b_p \cdot t_{33} + t_{43} = b_p, \qquad -b_p \cdot t_{34} + t_{44} = b_p .$$

Ezt az egyenletrendszert megoldva kapjuk a perspektív transzformáció mátrixát:

$$\mathbf{T}_{persp} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -(f_p + b_p)/(b_p - f_p) & -1 \\ 0 & 0 & -2 \cdot f_p \cdot b_p/(b_p - f_p) & 0 \end{bmatrix} .$$

Mivel a perspektív transzformáció nem affin transzformáció, így a keletkező homogén koordinátanégyes negyedik koordinátája nem lesz 1 értékű. Ezért, ha a transzformáció eredményét Descartes-koordinátákban szeretnénk megkapni, akkor a negyedik homogén koordinátával végig kell osztani a többi koordinátát. A homogén lineáris transzformációk szakaszt

szakaszba, háromszöget háromszögbe visznek át, de előfordulhat, hogy az eredmény szakasz, illetve háromszög az ideális pontot is tartalmazza (15.5.2. pont). A homogén osztás intuitíve a projektív térből az euklideszi térbe való átlépésnek felel meg, amelynek során az ideális pontot tartalmazó projektív szakaszból két félegyenes lesz. Mivel a szakasz két végpontját transzformáljuk, a művelet után nem tudjuk, hogy a kapott két pontra szakaszt kell-e illeszteni, vagy pedig a szakasz komplementerének megfelelő két félegyenest kell-e megoldásnak tekinteni. Ezt a jelenséget ***átfordulási problémának*** nevezi a szakirodalom.

Az átfordulási probléma elkerülhető, ha biztosak lehetünk benne, hogy a tárgyalakzat nem tartalmaz olyan pontot, amely ideális pontba kerülne. A perspektív transzformáció mátrixát megvizsgálva megállapíthatjuk, hogy a transzformáció után a negyedik homogén koordináta a transzformáció előtti $-z$ koordináta lesz. Ideális, azaz $h = 0$ koordinátájú pont a kamerát tartalmazó, a képsíkkal párhuzamos sík pontjaiból keletkezhet. Az első vágósíkra vágás viszont úgyis eltávolítja azokat az objektumrészleteket, amelyek a kamerához z-ben túl közel, vagy a kamera mögött vannak, ezért az átfordulási probléma úgy oldható meg, ha a homogén osztás előtt, még a projektív térben vágunk.

A homogén osztást követően a pontokat (X, Y, Z) Descartes-koordinátákban kapjuk meg.

15.7.4. Vágás homogén koordinátákban

A ***vágás*** célja az összes olyan objektumrészlet eltávolítása, amely nem vetülhet az ablakra, vagy amely nem az első és a hátsó vágósík között van. Az ***átfordulási probléma*** kiküszöbölése miatt a vágást a homogén osztás előtt kell végrehajtani. A homogén koordinátás vágási határokat a képernyő-koordinátarendszerben megfogalmazott, egy AABB-t definiáló feltételek visszatranszformálásával kaphatjuk meg. A homogén osztás után a belső pontok kielégítik a következő egyenlőtlenségeket:

$$-1 \le X = X_h/h \le 1, \qquad -1 \le Y = Y_h/h \le 1, \qquad -1 \le Z = Z_h/h \le 1\ . \tag{15.32}$$

Másrészt a szem előtti tartományok – a kamera transzformáció után – negatív z koordinátákkal rendelkeznek, és a perspektív transzformációs mátrixszal való szorzás után a negyedik homogén koordináta $h = -z$ lesz, amely így mindig pozitív. Tehát további követelményként megfogalmazzuk a $h > 0$ feltételt. Ekkor viszont szorozhatjuk a (15.32) egyenlőtlenségeket h-val, így eljutunk a vágási tartomány homogén koordinátás leírásához:

$$-h \le X_h \le h, \qquad -h \le Y_h \le h, \qquad -h \le Z_h \le h\ . \tag{15.33}$$

A pontok vágása triviális feladat, hisz a homogén koordinátás alakjukra csak ellenőrizni kell, hogy teljesülnek-e a (15.33) egyenlőtlenségek. A pontoknál összetettebb primitívekre (szakaszok, sokszögek stb.) azonban ki kell számítani a vágási tartomány határoló lapjaival való metszéspontokat, és a primitívnek pedig csak azt a részét kell meghagyni, amelynek pontjai kielégítik a (15.33) egyenlőtlenségeket.

A Descartes-koordinátákkal dolgozó vágási algoritmusokkal a 15.4.3. pontban foglalkoztunk. Az ott megismert módszerek alkalmazhatók homogén koordinátákra is, azzal a különbséggel, hogy most a (15.33) egyenlőtlenségek jelölik ki, hogy egy pont belső, illetve külső pontnak minősül-e, valamint a szakaszoknak a vágósíkkal képzett metszéspontját a szakasz és a sík homogén koordinátás egyenletéből kell számítani.

Tekintsünk egy $[X_h^1, Y_h^1, Z_h^1, h^1]$ és $[X_h^2, Y_h^2, Z_h^2, h^2]$ végpontú szakaszt, amely lehet önálló objektum, vagy egy sokszög egyik éle, és az $X_h \leq h$ félteret (a többi féltérre a vizsgálat teljesen hasonló). Három esetet kell megkülönböztetni:

1. Ha a szakasz mindkét végpontja belső pont, azaz $X_h^1 \leq h^1$ és $X_h^2 \leq h^2$, akkor a teljes szakasz belső pontokból áll, így megtartjuk.

2. Ha a szakasz mindkét végpontja külső pont, azaz $X_h^1 > h^1$ és $X_h^2 > h^2$, akkor a szakasz minden pontja külső pont, így a vágás a teljes szakaszt eltávolítja.

3. Ha a szakasz egyik végpontja külső pont, a másik végpontja belső pont, akkor ki kell számítani a szakasz és a vágósík metszéspontját, és a külső végpontot fel kell cserélni a metszésponttal. Figyelembe véve, hogy a szakasz pontjai kielégítik a (15.19) egyenletet, a vágósík pontjai pedig kielégítik az $X_h = h$ egyenletet, a metszéspont t_i paraméterét a következőképpen határozhatjuk meg:

$$X_h(t_i) = h(t_i) \quad \Longrightarrow \quad X_h^1 \cdot (1-t_i) + X_h^2 \cdot t_i = h^1 \cdot (1-t_i) + h^2 \cdot t_i \quad \Longrightarrow \quad t_i = \frac{X_h^1 - h^1}{X_h^1 - X_h^2 + h^2 - h^1}$$

A t_i paramétert a szakasz egyenletébe visszahelyettesítve a metszéspont $[X_h^i, Y_h^i, Z_h^i, h^i]$ koordinátáit is előállíthatjuk.

A vágás során új szakasz végpontok és új sokszög csúcspontok keletkeznek. Ha az eredeti objektum csúcspontjai járulékos információkat is hordoznak (például a felület színét vagy normálvektorát ebben a pontban), akkor a járulékos információkat az új csúcsokra is át kell számítani. Ehhez egyszerű lineáris interpolációt alkalmazhatunk. Ha a járulékos információ értéke a két végpontban I^1, illetve I^2, akkor a vágás során keletkező új $[X_h(t_i), Y_h(t_i), Z_h(t_i), h(t_i)]$ pontban az értéke $I^1 \cdot (1 - t_i) + I^2 \cdot t_i$.

15.7.5. A képernyő-transzformáció

A perspektív transzformáció után a látható pontok koordinátái a $[-1, 1]$ tartományban vannak, amelyeket még a képernyőn lévő megjelenítési ablak elhelyezkedésének és felbontásának megfelelően tolni és skálázni kell. Ha a keletkező kép bal-alsó sarkát az (X_{min}, Y_{min}), a jobb-felső sarkát az (X_{max}, Y_{max}) pixelen szeretnénk látni, a szemtől való távolságot kifejező Z koordinátákat pedig a (Z_{min}, Z_{max}) tartományban várjuk, akkor a képernyő-transzformáció mátrixa:

$$\mathbf{T}_{kep} = \begin{bmatrix} (X_{max} - X_{min})/2 & 0 & 0 & 0 \\ 0 & (Y_{max} - Y_{min})/2 & 0 & 0 \\ 0 & 0 & (Z_{max} - Z_{min})/2 & 0 \\ (X_{max} + X_{min})/2 & (Y_{max} + Y_{min})/2 & (Z_{max} + Z_{min})/2 & 1 \end{bmatrix}.$$

A perspektív transzformáció utáni koordinátarendszerek, így a képernyő-koordinátarendszer is ***balsodrású,*** szemben a virtuális világ és a kamera koordinátarendszereinek ***jobbsodrású*** állásával. A balsodrású elrendezés felel meg ugyanis annak a természetes elvárásnak, hogy a képernyőn az X koordináták balról-jobbra, az Y koordináták alulról-felfelé, a Z koordináták pedig a megfigyelőtől távolodva nőjenek.

15.7.6. Raszterizációs algoritmusok

A vágás, a homogén osztás és a képernyő-transzformáció után az alakzataink a képernyő-koordinátarendszerben vannak, ahol egy (X, Y, Z) pont vetületének koordinátái úgy határozhatók meg, hogy a koordinátahármasból csak az (X, Y) párt ragadjuk ki.

A ***raszterizáció*** során azokat a pixeleket azonosítjuk, amelyek átszínezésével a képernyő-koordinátarendszerbe transzformált geometriai alakzat formáját közelíthetjük. A raszterizációs algoritmusok kialakítása során a legfontosabb szempont az, hogy az algoritmus nagyon gyors legyen, és lépései egyszerű hardverrel megvalósíthatók legyenek. Ezt a követelményt azzal magyarázhatjuk, hogy a folyamatos mozgás érzékeléséhez másodpercenként legalább 20 olyan képet kell kiszámítani, amelyek nagyságrendileg egymillió pixelből állnak. Így az egyetlen pixelre jutó átlagos számítási idő mindössze 50 nanosec lehet.

Szakaszok rajzolása

Jelöljük a vetített szakasz végpontjait (X_1, Y_1), (X_2, Y_2)-vel. Tegyük fel továbbá, hogy midőn az első végpontból a második felé haladunk, mindkét koordináta nő, és a gyorsabban változó irány az X, azaz

$$\Delta X = X_2 - X_1 \geq \Delta Y = Y_2 - Y_1 \geq 0.$$

Ebben az esetben a szakasz enyhén emelkedő. A többi eset a végpontok és az X, Y koordináták megfelelő felcserélésével analóg módon kezelhető.

A szakaszrajzoló algoritmusokkal szemben alapvető elvárás, hogy az átszínezett képpontok között ne legyenek lyukak, és a keletkezett kép ne legyen vastagabb a feltétlenül szükségesnél. Ez az enyhén emelkedő szakaszok esetén azt jelenti, hogy minden pixel oszlopban pontosan egy pixelt kell átszínezni, nyilván azt, amelynek középpontja a szakaszhoz a legközelebb van. Az egyenes egyenlete:

$$y = m \cdot X + b, \qquad \text{ahol} \quad m = \frac{Y_2 - Y_1}{X_2 - X_1}, \quad \text{és} \quad b = Y_1 - X_1 \cdot \frac{Y_2 - Y_1}{X_2 - X_1}\ , \tag{15.34}$$

alapján, az X koordinátájú oszlopban a legközelebbi pixel függőleges koordinátája az $m \cdot x + b$ értékhez legközelebbi egész. A képlet minden pixel előállításához lebegőpontos szorzást, összeadást és lebegőpontos-egész átalakítást végez, ami megengedhetetlenül lassú.

A gyorsítás alapja a számítógépes grafika alapvető módszere, amelyet ***inkrementális elvnek*** nevezünk. Ez azon a felismerésre épít, hogy általában könnyebben meghatározhatjuk az $y(X + 1)$ értéket az $y(X)$ felhasználásával, mint közvetlenül az X-ből. Mivel egy enyhén emelkedő szakasz rajzolásakor az oszlopokat úgyis egymás után látogatjuk meg, az $(X + 1)$-dik oszlop feldolgozása során az $y(X)$ már rendelkezésre áll. Egy szakasz esetén:

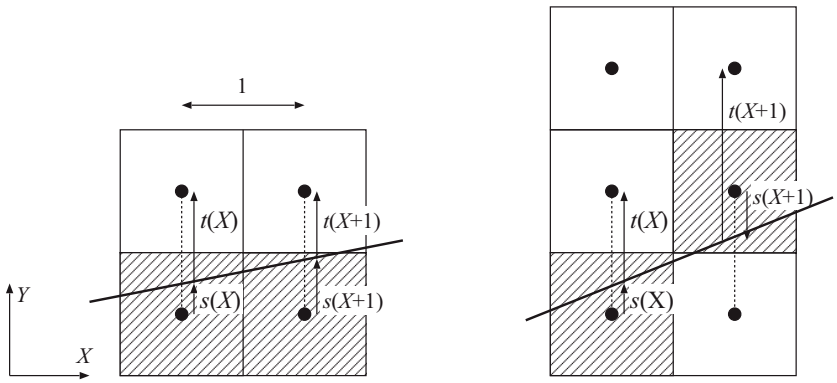
$$y(X + 1) = m \cdot (X + 1) + b = m \cdot X + b + m = y(X) + m\ ,$$

ehhez egyetlen lebegőpontos összeadás szükséges (m törtszám). Az elv gyakorlati alkalmazását ***digitális differenciális analizátor algoritmusnak*** (DDA-algoritmus) nevezik. A DDA-elvű szakaszrajzoló algoritmus:

```
DDA-SZAKASZRAJZOLÁS(X1, Y1, X2, Y2, szín)
1  m ← (Y2 − Y1)/(X2 − X1)
2  y ← Y1
3  for X ← X1 to X2
4      do Y ← KEREKÍT(y)
5         PIXEL-ÍRÁS(X, Y, szín)
6         y ← y + m
```

További gyorsítás érhető el ***fixpontos számábrázolás*** segítségével. Ez azt jelenti, hogy a törtszám 2^T-szeresét tároljuk egy egész változóban, ahol T a törtbitek száma. A törtbitek számát úgy kell megválasztani, hogy a leghosszabb ciklusban se halmozódhasson fel akkora hiba, hogy elrontsa a pixelkoordinátákat. Ha a leghosszabb szakasz hossza L, akkor az ehhez szükséges bitek száma $\log_2 L$. A vágásnak köszönhetően csak a képernyőn elférő szakaszokat raszterizáljuk, így L a képernyő vízszintes, illetve függőleges felbontásának maximumával egyenlő.

A DDA algoritmussal még mindig nem lehetünk teljes mértékben elégedettek. Egyrészt a szoftver implementáció során a fixpontos ábrázolás és a kerekítés eltolási (shift) műveleteket igényel. Másrészt – igaz szakaszonként csupán egyszer – az m meredekség kiszámításához osztani kell. Mindkét problémával sikeresen birkózik meg a ***Bresenham-algoritmus.***



15.41. ábra. A Bresenham-algoritmus által használt jelölések. Az s a legközelebbi pixelközéppont és a szakasz előjeles távolsága az Y tengely mentén, amely akkor pozitív, ha a szakasz a pixelközéppont felett van. A t a legközelebbi pixelközéppont feletti pixel és a szakasz távolsága az Y tengely mentén.

Jelöljük a szakasz és a legközelebbi pixel középpont függőleges, előjeles távolságát s-sel, a szakasz és a legközelebbi pixel feletti pixel függőleges távolságát t-vel (15.41. ábra). Ahogy a következő oszlopra lépünk, az s és t értékei változnak. Nyilván az eredetileg legközelebbi pixel sora és az eggyel feletti sor közül addig választjuk az alsó sort, amíg $s < t$. Bevezetve az $e = s - t$ hibaváltozót, addig nem kell megváltoztatnunk az átfestendő pixel sorát, amíg $e < 0$. Az s, t, e változók számításához az inkrementális elvet használhatjuk

$(\Delta X = X_2 - X_1, \Delta Y = Y_2 - Y_1)$:

$$s(X+1) = s(X) + \frac{\Delta Y}{\Delta X}, \quad t(X+1) = t(X) - \frac{\Delta Y}{\Delta X} \quad \Longrightarrow \quad e(X+1) = e(X) + 2\frac{\Delta Y}{\Delta X} \ .$$

Ezek az összefüggések akkor igazak, ha az $(X+1)$-dik oszlopban ugyanazon sorokban lévő pixeleket tekintjük, mint a megelőzőben. Előfordulhat azonban, hogy az új oszlopban már a felső pixel kerül közelebb a szakaszhoz (az e hibaváltozó pozitívvá válik), így az s, t, e mennyiségeket ezen pixelre és az ezen pixel feletti pixelre kell meghatározni. Erre az esetre a következő képletek vonatkoznak:

$$s(X+1) = s(X) + \frac{\Delta Y}{\Delta X} - 1, \quad t(X+1) = t(X) - \frac{\Delta Y}{\Delta X} + 1 \Longrightarrow e(X+1) = e(X) + 2\left(\frac{\Delta Y}{\Delta X} - 1\right) \ .$$

Figyeljük meg, hogy az s előjeles távolságot jelent, azaz az s negatív, ha a szakasz az alsó pixelközéppont alatt található. Feltételezhetjük, hogy az algoritmus indulásakor egy pixel középpontban vagyunk, tehát:

$$s(X_1) = 0, \qquad t(X_1) = 1 \quad \Longrightarrow \quad e(X_1) = s(X_1) - t(X_1) = -1 \ .$$

Az algoritmusnak az e hibaváltozót kell növelnie, és amikor az előjelet vált, a szakasz a következő pixelsorra lép, mialatt a hibaváltozó ismét a negatív tartományba csúszik vissza. A hibaváltozó kezeléséhez nem egész összeadás szükséges, a növekmény meghatározása pedig osztást igényel.

Vegyük észre azonban, hogy a hibaváltozó előjelváltásait úgy is nyomon követhetjük, ha nem közvetlenül a hibaváltozóval, hanem annak valamely pozitív számszorosával dolgozunk. Használjuk a hibaváltozó helyett az $E = e \cdot \Delta X$ ***döntési változót.*** Enyhén emelkedő szakaszok esetén a döntési változó pontosan akkor vált előjelet, amikor a hibaváltozó. A döntési változóra érvényes képleteket a hibaváltozóra vonatkozó képletek ΔX-szel történő szorzásával kapjuk meg:

$$E(X+1) = \begin{cases} E(X) + 2\Delta Y, \text{ ha } Y\text{-t nem kell léptetni} \ , \\ \\ E(X) + 2(\Delta Y - \Delta X), \text{ha } Y\text{-t léptetni kell} \ . \end{cases}$$

A döntési változó kezdeti értéke pedig $E = e(X_1) \cdot \Delta X = -\Delta X$.

A döntési változó egész kezdeti értékről indul és minden lépésben egész számmal változik, tehát az algoritmus egyáltalán nem használ törteket. Ráadásul a növekmények előállításához csupán egész összeadás (illetve kivonás), és 2-vel való szorzás szükséges.

A teljes ***Bresenham-algoritmus:***

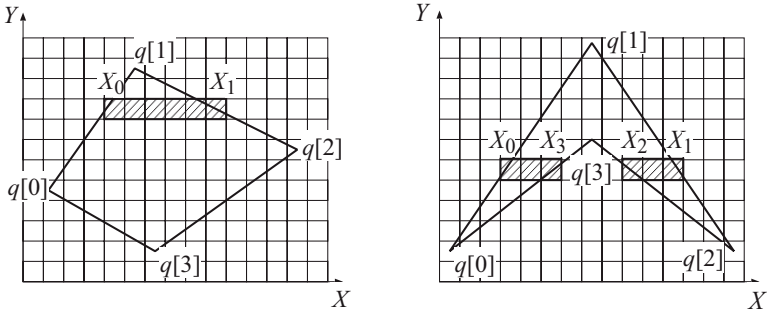
BRESENHAM-SZAKASZRAJZOLÁS(X_1, Y_1, X_2, Y_2, *szín*)

```
1  ΔX ← X2 − X1
2  ΔY ← Y2 − Y1
3  (dE+, dE−) ← (2(ΔY − ΔX), 2ΔY)
4  E ← −ΔX
5  Y ← Y1
6  for X ← X1 to X2
7      do if E ≤ 0
8           then E ← E + dE−    ▷ Döntési változó nempozitív, maradunk a pixelsorban.
9           else E ← E + dE+    ▷ Döntési változó pozitív, a következő pixelsorra lépünk.
10               Y ← Y + 1
11          PIXEL-ÍRÁS(X, Y, szín)
```

A Bresenham-algoritmus bevezetésénél a tört hibaváltozót úgy váltottuk ki egy egész változóval, hogy a kritikus egyenlőtlenséget az összes változójával együtt egy pozitív számmal szoroztuk, így az eredeti egyenlőtlenséggel ekvivalens, de csak egészeket tartalmazó kifejezéshez jutottunk. Ezt a megközelítést ***invariánsok módszerének*** nevezik, és sok raszterizációs eljárásban hasznos segédeszköznek bizonyul.

Poligonkitöltés

Az egyszeresen összefüggő sokszögeket kitöltő algoritmus bemenete a csúcsok $\vec{q}[0], \ldots, \vec{q}[m-1]$ tömbje (ez a tömb általában a poligonvágó algoritmus kimenete), amelyben az e-edik él a $\vec{q}[e]$ és $\vec{q}[e+1]$ csúcsokat köti össze. Az utolsó pont különleges kezelését a vágásnál megismert módon most is megtakaríthatjuk, ha a legelső csúcsot még egyszer betesszük a tömb végére. A többszörösen összefüggő sokszögeket egynél több zárt törött-vonallal, azaz több csúcstömbbel adhatjuk meg.



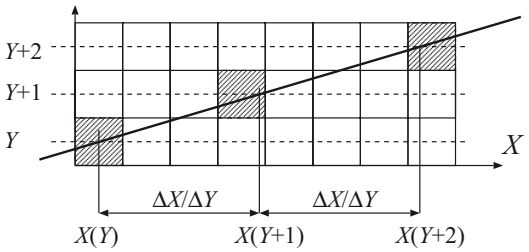
15.42. ábra. Poligonkitöltés. A sokszög belsejében lévő pixeleket vízszintes pásztánként keressük meg.

A kitöltést célszerűen vízszintes pixelsoronként, azaz ***pásztánként*** végezzük. Egyetlen pásztára az átszínezendő pixelek a következőképpen határozhatók meg. Kiszámítjuk a poligon éleinek metszéspontjait a vízszintes pásztával. A metszéspontokat az X koordináta alapján nagyság szerint rendezzük, majd átszínezzük a első és a második pont közötti, a harmadik és a negyedik pont közötti, általában a $(2i+1)$-edik és $(2i+2)$-edik pont közötti

pixeleket (15.42. ábra). Ez a módszer azokat a pontokat színezi ki, amelyeket ha végtelen távolból közelítünk meg, akkor páratlan számúszor kell átlépnünk a poligon határán.

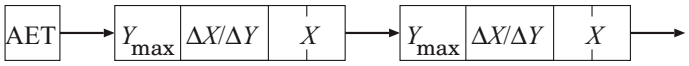
A pászták és az élek metszéspontjainak kiszámítását a következő megfigyelésekkel gyorsíthatjuk:

1. Egy él és a pászta között csak akkor keletkezhet metszéspont, ha a pászta Y koordinátája az él minimális és maximális Y koordinátája között van, ezért csak ezekre érdemes a metszéspontot kiszámítani. Az ilyen éleket ***aktív éleknek*** nevezzük. Az implementációhoz létre kell hoznunk az ***aktív él listát,*** amely mindig csak az aktív éleket tartalmazza.
2. A két szakasz közötti metszéspontszámítás lebegőpontos szorzást, osztást és összeadást tartalmaz, ezért időigényes. Az ***inkrementális elv*** felhasználásával azonban a metszéspont meghatározható a megelőző pászta metszéspontjából egyetlen fixpontos, nemegész összeadással (15.43. ábra).



15.43. ábra. A pászta és az élek közötti metszéspont inkrementális számítása. Az X koordináta mindig az egyenes meredekségének a reciprokával nő.

Az inkrementális elv használatakor figyelembe kell vennünk, hogy az X koordináta növekménye az egymást követő Y egész értékekre állandó. Ha az él nagyobb Y koordinátájú végpontjának koordinátái (X_{max}, Y_{max}), a kisebb Y koordinátájú végpontjának koordinátái pedig (X_{min}, Y_{min}), akkor a növekmény $\Delta X/\Delta Y$, ahol $\Delta X = X_{max} - X_{min}$ és $\Delta Y = Y_{max} - Y_{min}$. A növekmény általában nem egész szám, tehát a $\Delta X/\Delta Y$ és az X érték tárolására fixpontos tört ábrázolást kell használnunk. Egy aktív él reprezentációja tehát tartalmazza a fixpontos ábrázolású $\Delta X/\Delta Y$ növekményt, az ugyancsak fixpontos ábrázolású X metszéspontot, valamint a szakasz maximális függőleges koordinátáját (Y_{max}). Erre azért van szükségünk, hogy el tudjuk dönteni, hogy az Y pászták növelése során mikor fejezi be az él aktív pályafutását, azaz mikor kell eltávolítani az aktív él listából.



15.44. ábra. Az aktív él lista szerkezete.

Az Y pásztákat egymás után töltjük ki. Minden pásztára megnézzük, hogy mely élek válnak pont ekkor aktívvá, azaz mely élek minimális Y koordinátája egyezik meg a pászta koordinátájával. Ezeket az éleket betesszük az aktív él listába. Egyúttal az aktív él listát átvizsgáljuk, hogy vannak-e ott nyugdíjba vonuló élek is, amelyek maximális Y koordinátája megegyezik a pászta koordinátájával. A nyugdíjba vonuló éleket kivesszük a listából (vegyük észre, hogy ebben a megoldásban az él alsó végpontját az él részének tekintjük, a felső

végpontját viszont nem). A kitöltés előtt gondoskodunk arról, hogy az aktív él listában az élek az X koordináta szerint rendezettek legyenek, majd minden második élpár közötti pixeleket átszínezzük. A kitöltés után az aktív él lista tagjaiban a metszéspontokat felkészítjük a következő pásztára, azaz minden él X tagjához hozzáadjuk az él $\Delta X/\Delta Y$ növekményét. Majd kezdjük az egészet elölről a következő pásztára.

POLIGONKITÖLTÉS(*poligon*, *szín*)

```
 1  for Y ← 0 to Y_max
 2      do for a poligon összes él-ére                ▷ Aktívvá váló élek az AET-be.
 3             do if él.ymin = Y
 4                   then AETBE-RAK(él)
 5         for minden él-re az AET-ben  ▷ Az aktív létet befejező élek törlése az AET-ből.
 6             do if él.ymax ≤ Y
 7                   then AETBŐL-TÖRÖL(él)
 8         AET-RENDEZÉS                                ▷ Rendezés X szerint.
 9         for minden második egymást követő (él1, él2) párra az AET-ben
10             do for X ← KEREKÍT(él1.x) to KEREKÍT(él2.x)
11                   do PIXEL-ÍRÁS(X, Y, szín)
11         for minden él-re az AET-ben                 ▷ Inkrementális elv.
12             do él.x ← él.x + él.ΔX/ΔY
```

Az algoritmus vízszintes pásztánként dolgozik, egy pászta feldolgozását az aktívvá váló élek ($él.ymin = Y$) aktív listába fűzésével kezdi. Az aktív él listát három művelet kezeli. Az AETBE-RAK($él$) művelet az él adatai alapján előállítja az aktív él lista egy elemének az adatait ($Y_{max}, \Delta X/\Delta Y, X$), és a keletkező rekordot beteszi a listába. Az AETBŐL-TÖRÖL művelet egy listaelemet töröl a listából, amikor egy él éppen befejezi az aktív létet ($él.ymax \leq Y$). Az AET-RENDEZÉS az X mező alapján átrendezi a listát. A rendezés után az algoritmus minden második él és a következő él közötti pixelt kiszínez, és végül az inkrementális képletek alkalmazásával az aktív él lista elemeit a következő pásztára lépteti.

15.7.7. Inkrementális láthatósági algoritmusok

A ***láthatósági feladatot*** a képernyő-koordinátarendszerben oldjuk meg. Általában feltételezzük, hogy a felületeket sokszögháló formájában kapjuk meg.

Z-buffer algoritmus

A ***z-buffer algoritmus*** minden pixelre megkeresi azt a felületet, amelynél a pixelen keresztül látható pontban a Z koordináta minimális. A kereséshez minden pixelhez, a feldolgozás adott pillanatának megfelelően tároljuk az abban látható felületi pontok közül a legközelebbi Z koordinátáját. Ezt a Z értékeket tartalmazó tömböt nevezzük ***z-buffernek*** vagy ***mélységpuffernek.***

A továbbiakban az egyszerűség, valamint a gyakorlati jelentőség miatt feltételezzük, hogy a felület háromszögekből áll. A háromszögeket egyenként dolgozzuk fel, és meghatározzuk az összes olyan pixelt, amely a háromszög vetületén belül van. Ehhez egy háromszögkitöltő algoritmust kell végrehajtani.

Amint a kitöltés során egy pixelhez érünk, kiszámítjuk a felületi pont Z koordinátáját és összehasonlítjuk a z-bufferben lévő értékkel. Ha az ott található érték kisebb, akkor a már feldolgozott háromszögek között van olyan, amelyik az aktuális háromszöget ebben a pontban takarja, így az aktuális háromszög ezen pontját nem kell kirajzolni. Ha viszont a z-bufferbeli érték nagyobb, akkor az idáig feldolgozott háromszögeket az aktuális háromszög takarja ebben a pontban, ezért az aktuális háromszög színét kell beírni a pixelbe és egyúttal a Z értékét a z-bufferbe.

A z-buffer módszer algoritmusa tehát:

```
Z-BUFFER-ALGORITMUS()
1  for minden p pixelre                                   ▷ Képernyő törlés.
2      do PIXEL-ÍRÁS(p, háttér-szín)
3         z-buffer[p] ← a legnagyobb ábrázolható érték
4  for minden o háromszögre                               ▷ Rajzolás.
5      do for az o háromszög vetületének minden p pixelére
6         do Z ← az o háromszög p-re vetülő pontjának Z koordinátája
7            if Z < z-buffer[p]
8               then PIXEL-ÍRÁS(p, az o színe ebben a pontban)
9                    z-buffer[p] ← Z
```

Alkalmazhatnánk az előző fejezet poligonkitöltő algoritmusát is, de célszerűbb kihasználni a háromszög speciális tulajdonságaiból adódó előnyöket. Rendezzük a csúcsokat az Y koordináták alapján és sorszámozzuk újra őket úgy, hogy az elsőnek legyen a legkisebb és a harmadiknak a legnagyobb Y koordinátája, és gondolatban vágjuk ketté a háromszöget az Y_2 pásztával. Ezzel két hasonló tulajdonságú háromszöget, egy alsó és egy felső háromszöget kapunk, amelyeken belül a kezdő (baloldali) és a záró (jobboldali) él nem változik. A háromszög éleinek jobb-, illetve baloldali élként történő osztályozása attól függ, hogy az (X_2, Y_2) vetített csúcs az (X_1, Y_1)-ből az (X_3, Y_3) felé tartó irányított egyenes bal, vagy jobb oldalán van-e. Ha az (X_2, Y_2) a bal oldalon található, a vetített háromszöget ***balállásúnak,*** egyébként pedig ***jobbállásúnak*** nevezzük. A csúcspontok Y koordináta szerinti rendezése, a háromszög felvágása és az állás eldöntése után az általános poligonkitöltőnkből az aktív él lista adminisztrációja kihagyható, csupán az inkrementális metszéspontszámítást kell megtartani.

Az algoritmus részleteinek a bemutatása során feltesszük, hogy aktuálisan az

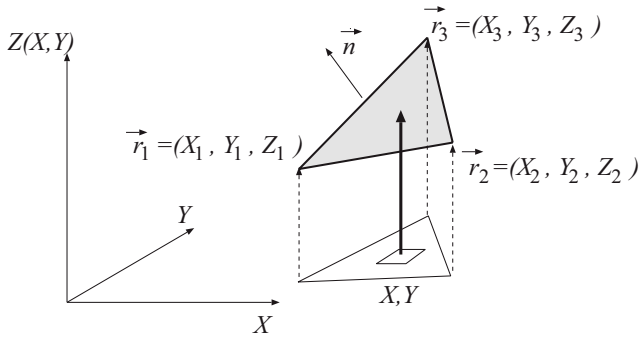
$$\vec{r}_1 = [X_1, Y_1, Z_1], \qquad \vec{r}_2 = [X_2, Y_2, Z_2], \qquad \vec{r}_3 = [X_3, Y_3, Z_3]$$

csúcspontokkal definiált háromszöget dolgozzuk fel. A raszterizációs algoritmusnak elő kell állítania a háromszög vetületébe eső X, Y pixel címeket a Z koordinátákkal együtt (15.45. ábra).

Az X, Y pixel címből a megfelelő Z koordinátát a háromszög síkjának az egyenletéből számíthatjuk ((15.1) egyenlet), amely szerint a Z koordináta az X, Y koordináták lineáris függvénye. A háromszög síkjának az egyenlete:

$$n_X \cdot X + n_Y \cdot Y + n_Z \cdot Z + d = 0, \qquad \text{ahol } \vec{n} = (\vec{r}_2 - \vec{r}_1) \times (\vec{r}_3 - \vec{r}_1) \quad \text{és} \quad d = -\vec{n} \cdot \vec{r}_1 \ . \tag{15.35}$$

A háromszög balállású, illetve jobbállású voltát a normálvektor Z koordinátájának előjele



15.45. ábra. Egy háromszög a képernyő-koordinátarendszerben. A háromszög XY síkon levő vetületébe eső pixeleket látogatjuk meg. A pixeleknek megfelelő pont Z koordinátáját a háromszög síkjának egyenletéből számítjuk.

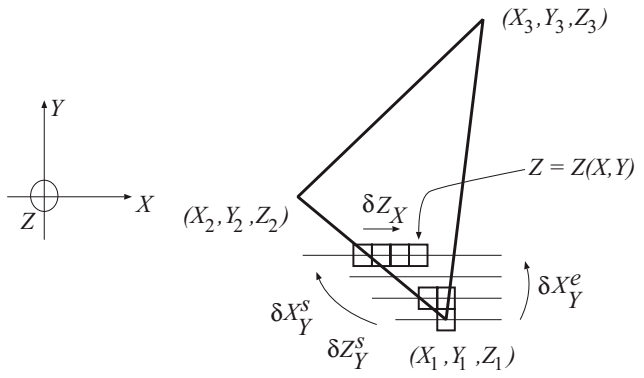
alapján állapíthatjuk meg. Ha n_Z negatív, a háromszög balállású, ha pozitív, akkor jobbállású. Ha n_Z zérus, akkor a vetítés következtében a háromszögből egyetlen szakasz lesz, így a kitöltésére nincs szükség.

A háromszög síkjának az egyenletéből a $Z(X, Y)$ függvény:

$$Z(X, Y) = -\frac{n_X \cdot X + n_Y \cdot Y + d}{n_Z} \ . \tag{15.36}$$

Az inkrementális elv felhasználásával ezen képlet jelentősen egyszerűsíthető:

$$Z(X + 1, Y) = Z(X, Y) - \frac{n_X}{n_Z} = Z(X, Y) + \delta Z_X \ . \tag{15.37}$$



15.46. ábra. Inkrementális Z érték számítás egy balállású háromszögre.

Mivel a δZ_X paraméter állandó az egész háromszögre, csak egyszer kell kiszámítani. Egyetlen pásztán belül a Z koordináta kiszámítása tehát egyetlen összeadást igényel. A határvonalakat a poligonkitöltésnél megismert módon ugyancsak előállíthatjuk az inkrementális elv felhasználásával, sőt a határvonal mentén a pászták kezdeti Z koordinátája is egyetlen

összeadással kiszámítható a megelőző pászta kezdeti Z koordinátájából (15.46. ábra). A teljes inkrementális algoritmus, amely egy balállású háromszög alsó felét tölti ki (a jobbállású eset, illetve a felső felet kitöltő algoritmus nagyon hasonló):

Z-BUFFER-ALSÓ-FÉLHÁROMSZÖG$(X_1, Y_1, Z_1, X_2, Y_2, Z_2, X_3, Y_3, Z_3, szín)$

```
1  n⃗ ← ((X2, Y2, Z2) − (X1, Y1, Z1)) × ((X3, Y3, Z3) − (X1, Y1, Z1))      ▷ Normálvektor.
2  δZX ← −nX/nZ                                                            ▷ Z inkremens, ha X eggyel nő.
3  (δXYs, δZYs, δXYe) ← ((X2 − X1)/(Y2 − Y1), (Z2 − Z1)/(Y2 − Y1), (X3 − X1)/(Y3 − Y1))
4  (Xbal, Xjobb, Zbal) ← (X1, X1, Z1)
5  for Y ← Y1 to Y2
6      do Z ← Zbal
7          for X ← KEREKÍT(Xbal) to KEREKÍT(Xjobb)                          ▷ Egy pászta rajzolása.
8              if Z < z-buffer[X, Y]                                        ▷ Takarás vizsgálat.
9                  then PIXEL-ÍRÁS(X, Y, szín)
10                      z-buffer[X, Y] ← Z
11                 Z ← Z + δZX
12         (Xbal, Xjobb, Zbal) ← (Xbal + δXYs, Xjobb + δXYe, Zbal + δZYs)   ▷ Következő pászta.
```

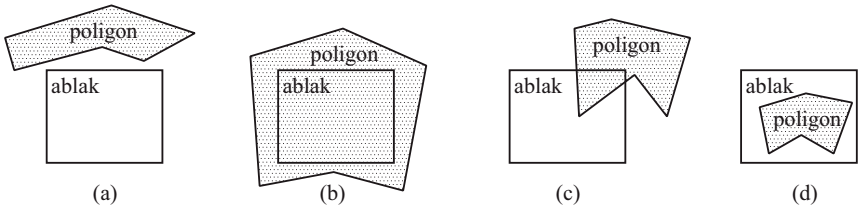
A megismert algoritmus a háromszög kitöltés, azaz a háromszögben lévő pixelek azonosításával párhuzamosan lineárisan interpolálja a Z koordinátát. A lineáris interpolációhoz pixelenként egyetlen összeadás elegendő. Ugyanez a megoldás más háromszög tulajdonságok esetén is alkalmazható. Például, ha ismerjük a háromszög csúcsainak színét, a belső pontokra folytonos színátmenetet valósíthatunk meg a szín lineáris interpolációjával [166]. Ha a számokat fixpontosan ábrázoljuk, a lineáris interpolátor egyszerű áramköri elemek felhasználásával hardverben is realizálható. A mai grafikus kártyák ilyen egységekkel rendelkeznek.

A z-buffer algoritmus a háromszögeket egyenként tölti ki, így $\Theta(N \cdot P)$ időt igényel, ahol N a háromszögek, P pedig a kép pixeleinek a száma. A gyakorlatban a helyzet ennél kedvezőbb, mert a háromszögek száma általában a tesszelláció finomítása miatt nő, így ha több háromszögünk van, akkor méretük is kisebb, tehát kitöltésükhöz kevesebb pixel szükséges. A futási idő így a háromszögek vetületei által lefedett pixelszámmal arányos, amely ekkor csak a felbontástól függ, azaz $\Theta(P)$ típusú.

Warnock-algoritmus

A különböző felületelemek a képen összefüggő pixeltartományon keresztül látszanak. Ezen koherencia tulajdonság miatt célszerű a láthatóságot a pixelnél nagyobb egységekre vizsgálni. A vetített poligonok és az ablak lehetséges viszonyai alapján, a 15.47. ábra szerint, különálló, körülvevő, metsző és tartalmazott poligonokat különböztethetünk meg. Ha szerencsénk van, akkor az objektumtérben csak különálló és körülvevő poligonok vannak. A különálló sokszögek nem látszhatnak, így ezekkel nem kell foglalkozni. A körülvevő sokszögek vetülete pedig az összes pixelt magában foglalja. Ha feltételezhetjük, hogy a sokszögek nem metszik egymást, akkor a körülvevő poligonok közül csak egyetlen egy látható, amelyet például az ablak középpontján átmenő sugár követésével választhatunk ki.

Az egyetlen sugár követésével megoldható szerencsés eset akkor áll fenn, amikor egyetlen poligonél sem vetül az ablakra. Ezt úgy ellenőrizhetjük, hogy a poligonélek vetületére



15.47. ábra. Poligon-ablak relációk: különálló (A), körülvevő (B), metsző (C), tartalmazott (D).

alkalmazzuk a kétdimenziós szakaszvágó algoritmust (15.4.3. pont). Ha a vágás minden szakaszra úgy találja, hogy a szakasz teljes egészében eldobandó, akkor a sokszögek valóban csak különálló és körülvevő típusúak lehetnek. Ha viszont nem vagyunk ebben a szerencsés helyzetben, akkor az ablakot négy egybevágó ablakra bontjuk fel, és újra megvizsgáljuk, hogy szerencsénk van-e vagy sem. Az eljárás, amelyet ***Warnock-algoritmusnak*** neveznek, rekurzívan ismételgeti ezt a lépést, amíg vagy sikerül visszavezetni a takarási feladatot a szerencsés esetre, vagy az ablak mérete a pixel méretére zsugorodik. A pixel méretű ablaknál az újabb felosztások már értelmetlenné válnak, így erre a pixelre már a szokásos módon (például sugárkövetéssel) kell megoldanunk a takarási feladatot. A módszer algoritmusának leírása során (X_1, Y_1)-gyel jelöljük az ablak bal-alsó sarkának és (X_2, Y_2)-vel a jobb-felső sarkának egész értékű koordinátáit:

WARNOCK-ALGORITMUS(X_1, Y_1, X_2, Y_2)

if $X_1 \neq X_2$ vagy $Y_1 \neq Y_2$ ▷ Az ablak a pixelnél nagyobb?
  then if legalább egy él esik az ablakba ▷ Ablakfelezés és rekurzió.
    then WARNOCK-ALGORITMUS$(X_1, Y_1, (X_1 + X_2)/2, (Y_1 + Y_2)/2)$
      WARNOCK-ALGORITMUS$(X_1, (Y_1 + Y_2)/2, (X_1 + X_2)/2, Y_2)$
      WARNOCK-ALGORITMUS$((X_1 + X_2)/2, Y_1, X_2, (Y_1 + Y_2)/2)$
      WARNOCK-ALGORITMUS$((X_1 + X_2)/2, (Y_1 + Y_2)/2, X_2, Y_2)$
      return
▷ Triviális eset: az (X_1, Y_1, X_2, Y_2) téglalap homogén.
poligon ← az $((X_1 + X_2)/2, (Y_1 + Y_2)/2)$ pixelben látható poligon
if nincs poligon
  then (X_1, Y_1, X_2, Y_2) téglalap kitöltése háttér színnel
  else (X_1, Y_1, X_2, Y_2) téglalap kitöltése a *poligon* színével

Festő algoritmus

A festés során a későbbi ecsetvonások elfedik a korábbiakat. Ezen egyszerű elv kiaknázásához rendezzük a poligonokat oly módon, hogy egy P poligon csak akkor állhat a sorrendben egy Q poligon után, ha *nem takarja* azt. Majd a poligonokat a kapott sorrendben visszafelé haladva egymás után raszterizáljuk. Ha egynél több poligon vetül egy pixelre, a pixel színe az utoljára rajzolt poligon színével egyezik meg. Mivel a rendezés miatt a korábban rajzoltak nem takarhatják az utolsó poligont, ezzel a ***festő algoritmussal*** a takarási feladatot megoldottuk.

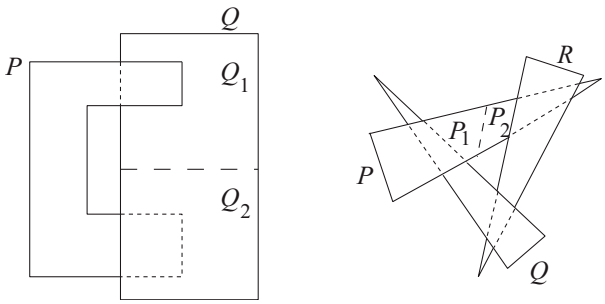
A poligonok megfelelő rendezése több problémát is felvet, ezért vizsgáljuk meg ezt a kérdést részletesebben. Azt mondjuk, hogy egy „P poligon *nem takarja* a Q poligont”, ha P-nek semelyik pontja sem takarja Q valamely pontját. Ezen reláció teljesítéséhez a következő feltételek valamelyikét kell kielégíteni:

1. a P poligon minden pontja hátrébb van (nagyobb Z koordinátájú) a Q poligon bármely pontjánál;
2. a P poligon vetületét befoglaló téglalapnak és a Q poligon vetületét befoglaló téglalapnak nincs közös része;
3. P valamennyi csúcsa (így minden pontja) messzebb van a szemtől, mint a Q síkja;
4. Q valamennyi csúcsa (így minden pontja) közelebb van a szemhez, mint a P síkja;
5. a P és Q vetületeinek nincs közös része.

A felsorolt feltételek bármelyike elégséges feltétel, amelyek ellenőrzése a sorrendnek megfelelően egyre nehezebb, ezért az ellenőrzéseket a fenti sorrendben végezzük el.

Első lépésként rendezzük a poligonokat a maximális Z koordinátájuk szerint úgy, hogy a közeli poligonok a lista elején, a távoli poligonok pedig a lista végén foglaljanak helyet. Ez önmagában még nem elég, hiszen előfordulhat, hogy az így kapott listában valahol borul a „P poligon *nem takarja* a Q poligont” reláció.

Ezért minden egyes poligont össze kell vetni valamennyi, a listában előtte álló poligonnal, és ellenőrizni kell a megadott feltételeket. Ha azok valamelyike minden előbb álló poligonra teljesül, akkor az adott poligon helye megfelelő. Ha viszont a poligonunk takar egy előbb álló poligont, akkor a takart poligont az aktuális poligon mögé kell vinni a listában, és a mozgatott poligonra visszalépve újra kell kezdeni a feltételek ellenőrzését.



15.48. ábra. Ciklikus takarás, amelyet úgy oldhatunk fel, hogy az egyik sokszöget a másik síkjával kettévágunk.

Előfordulhat, hogy ez az algoritmus végtelen ciklusba kerül. Például ha két poligon egymást kölcsönösen takarja (15.48. ábra bal oldala), az ismertetett algoritmus ezen két poligont vég nélkül cserélgetné. Még nehezebben felismerhető esetet mutat be ugyanezen ábra jobb oldala, amikor kettőnél több poligon ciklikus takarásának lehetünk tanúi.

Ezeket a ciklikus átlapolásokat a poligonok megfelelő vágásával oldhatjuk fel, és ezáltal átsegíthetjük az algoritmusunkat a kritikus pontokon. A ciklikus átlapolások felismeréséhez a mozgatáskor a poligonokat megjelöljük. Ha még egyszer mozgatni kell őket, akkor valószínűsíthető, hogy ennek oka a ciklikus átlapolás. Ekkor az újból mozgatott poligont a másik poligon síkja mentén két részre vágjuk.

BSP-fa

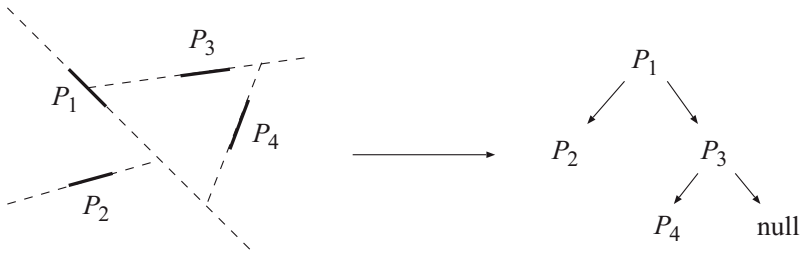
A ***BSP-fa*** egy bináris térpartícionáló fa, amely minden szinten a reprezentált térrészt egy alkalmas síkkal két részre bontja. A BSP-fa egy közeli rokona a 15.6.2. pontban megismert kd-fa, amely koordinátatengelyekkel párhuzamos elválasztó síkokat használ. Jelen fejezetünk BSP-fája azonban a háromszögek síkját választja elválasztó síkként.

A fa csomópontjaiban sokszögeket találunk, amelyek síkja választja szét a két gyerek által definiált térrészt (15.49. ábra). A fa levelei vagy üresek, vagy egyetlen sokszöget tartalmaznak.

A BSP-fát felépítő BSP-FA-FELÉPÍTÉS algoritmus egy S sokszöglistát kap. Az algoritmusban a bináris fa egy csomópontját *csomópont*-tal, a csomóponthoz tartozó sokszöget *csomópont.sokszög*-gel, a csomópont két gyerekét pedig *csomópont.bal*-lal, illetve *csomópont.jobb*-bal jelöljük. Egy $\vec{r}$ pontot az $\vec{n} \cdot (\vec{r} - \vec{r}_0)$ skaláris szorzat előjele alapján sorolunk az $\vec{n}$ normálvektorú és $\vec{r}_0$ helyvektorú sík nem negatív (jobb) és negatív (bal) tartományába.

BSP-FA-FELÉPÍTÉS(S)

```
1  hozzunk létre egy új csomópont-ot
2  if S üres vagy egyetlen sokszöget tartalmaz
3     then csomópont.sokszög ← S
4          csomópont.bal ← üres
5          csomópont.jobb ← üres
6     else csomópont.sokszög ← egy sokszög az S listából
7          távolítsuk el csomópont.sokszög-et az S listából
8          S⁺ ← S-ből a csomópont.sokszög síkjának nemnegatív félterébe lógók
9          S⁻ ← S-ből a csomópont.sokszög síkjának negatív félterébe lógók
10         csomópont.jobb ← BSP-fa-felépítés(S⁺)
11         csomópont.bal ← BSP-fa-felépítés(S⁻)
12 return csomópont
```



15.49. ábra. A BSP-fa. A térrészeket a tartalmazott sokszögek síkjai osztják fel.

A hatékonyság érdekében a BSP-fát úgy érdemes felépíteni, hogy mélysége minimális legyen. A BSP-fa mélysége függ az elválasztó síkot meghatározó sokszög kiválasztási stratégiájától, de ez a függés nagyon bonyolult, ezért heurisztikus szabályokat kell alkalmazni.

A BSP-fa segítségével megoldhatjuk a takarási feladatot. A sokszögeket a fa bejárása során rajzoljuk fel. Minden csomópontnál eldöntjük, hogy a kamera a csomópont síkjának

melyik oldalán van. Először azon gyerek irányába lépünk, amely a kamera átellenes oldalán van, majd felrajzoljuk a csomópont saját sokszögét, végül pedig a kamera oldali gyereket dolgozzuk fel.

Gyakorlatok

15.7-1. Készítsük el a Bresenham-algoritmus teljes, mind a 8 síktartományt kezelő változatát.

15.7-2. A poligonkitöltő eljárás minden pásztára minden élt megvizsgál, hogy az aktív él listába teheti-e őket. Módosítsuk az eljárást, hogy erre minden élre csak egyszer legyen szükség.

15.7-3. Készítsük el a z-buffer algoritmus teljes változatát, amely mind balállású, mind pedig jobbállású háromszögekre működik.

15.7-4. Az átlátszó tárgyak színét egy egyszerű modell szerint a tárgy saját színének és a mögöttes tárgy színének súlyozott átlagaként számíthatjuk. Mutassuk meg, hogy ekkor az ismertetett takarási algoritmusok közül csak a festő algoritmus és a BSP-fa ad helyes megoldást.

15.7-5. Az ismertetett Warnock-algoritmus akkor is felbontja az ablakot, ha arra egyetlen sokszög éle vetül. Javítsuk a módszert úgy, hogy ebben az esetben a poligont már újabb rekurziók nélkül felrajzolja.

15.7-6. Alkalmazzuk a BSP-fát diszkrét idejű ütközésfelismeréshez.

15.7-7. Alkalmazzuk a BSP-fát a sugárkövető eljárás térpartícionáló adatszerkezeteként.

Feladatok

15-1. Megjelenítő rendszer sugárkövetéssel

Készítsünk megjelenítő rendszert a sugárkövetés algoritmusával. A testeket háromszöghálóval, illetve kvadratikus felületként adjuk meg, és diffúz fényvisszaverődési tényezőt rendelünk hozzájuk. A virtuális térben pontszerű fényforrásokat is felveszünk. Egy pont látható színe arányos a diffúz fényvisszaverődési tényezővel, a fényforrás teljesítményével, a pontot a fényforrással összekötő irány és a felületi normális közötti szög koszinuszával (Lambert-féle koszinusz-törvény), és fordítottan arányos a pont és a fényforrás távolságával. A fényforrások láthatóságának eldöntéséhez ugyancsak a sugárkövetést használjuk.

15-2. Folytonos idejű ütközésfelismerés sugárkövetéssel

A sugárkövetés felhasználásával adjunk javaslatot egy folytonos ütközésfelismerő algoritmusra, amely egy mozgó, forgó poliéderre és egy mozdulatlan síkra kiszámítja az ütközés várható idejét. A megoldás során a poliéder csúcsainak mozgását kis dt időintervallumokban egyenesvonalú egyenletes mozgásnak tekintsük.

15-3. Megjelenítő rendszer inkrementális képszintézissel

Készítsünk teljes háromdimenziós megjelenítő rendszert, amelyben a modellezési és kamera-transzformációk beállíthatók. A virtuális világ szereplőit háromszögenként adjuk meg, a csúcspontokhoz színinformációt is kapcsolva. A transzformációk és vágás után z-buffer algoritmussal oldjuk meg a takarási feladatot, a belső pontok színének számításánál pedig a csúcspontok színét lineárisan interpoláljuk.

Megjegyzések a fejezethez

Az euklideszi, analitikus és projektív geometria elemeiről Hajós György [189] kiváló könyvében, a projektív geometriáról általában Maxwell [314, 315] és Coxeter [92] műveiben, a számítógépes grafikai alkalmazásáról pedig Herman Iván [204] és Krammer Gergely [266] cikkeiben olvashatunk. A görbék és felületek modellezésével a számítógépes geometriai tervezés (CAD, CAGD) foglalkozik, amelyet átfogóan Gerald Farin [125], valamint Rogers és Adams [395] tárgyalnak. A geometriai modellek mérési eredményekből történő felépítése a *mérnöki visszafejtés* [481] területe. Az implicit felületek tanulmányozásához Bloomenthal művét [54] ajánljuk. A testeknek implicit egyenletekkel történő leírása napjainkban újabb reneszánszát éli a *funkcionális-reprezentáció (F-Rep)* alapú modellezés elterjedésével, amelynek részleteivel a http://cis.k.hosei.ac.jp/~F-rep honlap foglalkozik. A cseppmodellezésre először Blinn tett javaslatot [53]. Később az általa javasolt exponenciális függvényt kicserélték polinomfüggvényekre [502]. A polinomfüggvények, különösen a másodfokú alakok azért népszerűek, mert ekkor a sugárkövetés során csak másodfokú egyenleteket kell megoldani.

A geometriai algoritmusok a geometriai problémákra – mint a konvex burok létrehozása, metszés, tartalmazás vizsgálat, háromszögekre bontás, geometriai keresés stb. – adnak megoldást. A témakörrel az *Új algoritmusok* című könyv egy fejezete is foglalkozott, további információk Preparata és Shamos [376], valamint Marc de Berg műveiben [101, 102] találhatók. A egyszerű vagy akár többszörösen összefüggő sokszögek háromszögekre bontásához robusztus algoritmust adni annak ellenére meglepően nehéz, hogy a témakör már évtizedek óta fontos kutatási terület. A gyakorlatban használt algoritmusok $O(n \lg n)$ időben futnak [102, 422, 506], bár Chazelle [74] egy optimális, lineáris idejű algoritmus elvét is kidolgozta. A *két fül tétel* idézett bizonyítása Joseph O'Rourke-től származik [357]. A háromszöghálókon működő pillangó felosztást Dyn és társai [117] javasolták. A *Sutherland–Hodgeman-poligonvágást* a [442] cikk alapján mutattuk be.

Az ütközésfelismerés a számítógépes játékok [450] egyik legkritikusabb algoritmusa. Ez akadályozza meg ugyanis, hogy a szereplők a falakon átléphessenek, valamint ezzel dönthetjük el, hogy egy lövedék eltalált-e valamit a virtuális térben. Az ütközésfelismerési algoritmusokat Jiménez, Thomas és Torras tekintik át [230]. A felosztott felületekről sok hasznos információt kaphatunk Catmull és Clark eredeti cikkéből [70], Warren és Heimer könyvéből [485], valamint Brian Sharp ismertetőiből [429, 428]. A pillangófelosztást Dyn, Gregory és Levin [117] javasolta.

A sugárkövetés elveivel Glassner [154] könyvében ismerkedhetünk meg. A *3D szakaszrajzoló algoritmust* Fujimoto és társai [143] cikke alapján tárgyaljuk. A sugárkövető algoritmusok bonyolultságát számos publikációban vizsgálták. Bebizonyosodott, hogy N objektumra a sugárkövetési feldatot $O(\lg N)$ időben meg lehet oldani [101, 451], de ez csak elméleti eredmény, mert ehhez $\Omega(N^4)$ memóriaigény és előfeldolgozási idő szükséges, és a konstans szorzó is olyan nagy, hogy az erőforrásigény a gyakorlat számára elfogadhatatlan. A gyakorlatban inkább a fejezetben is tárgyalt heurisztikus módszereket alkalmazzák, amelyek a legrosszabb esetben ugyan nem, de várható értékben csökkentik a sugárkövetési feladat megoldási idejét. A heurisztikus módszereket Márton Gábor [339, 451] elemezte valószínűségi módszerekkel, és ő javasolta a fejezetben is használt modellt. A heurisztikus algoritmusokról, elsősorban a kd-fa alapú módszer hatékony megvalósításáról Vlastimil Havran disszertációjában [196] olvashatunk, egy konkrét, optimalizált megvalósítás pedig

Szécsi László cikkében [444] található.

A virtuális világ valószínűségi modelljében használt Poisson-pontfolyamat ismertetése megtalálható például Karlin és Taylor [240] és Lamperti [275] könyveiben.

Az alkalmazott cellafelezési módszer Havrantól [196] származik. Az idézett integrálgeometriai tétel megtalálható Santaló[412] könyvében.

A négyfa és az oktálisfa geoinformatikai alkalmazásait a könyv 16. fejezete tárgyalja.

Az inkrementális képszintézis algoritmusaival Jim Blinn foglalkozik részletesen [53], C++ nyelvű implementációt a [448] könyvben találhatunk, valamint más általános számítógépes grafika könyvekhez is fordulhatunk [133, 450]. A láthatósági algoritmusok összehasonlító elemzését például a [443, 449] művekben találjuk meg. A Bresenham-algoritmus forrása [58]. Az inkrementális képszintézis algoritmusok, és azokon belül a z-buffer algoritmus, a valós-idejű megjelenítés legnépszerűbb módszere, ezért a grafikus kártyák ennek lépéseit valósítják meg, és az elterjedt grafikus könyvtárak is (*OpenGL, DirectX*) erre a megközelítésre épülnek.

A takarási feladatot megoldó *festő algoritmust* Newell és társai [349] javasolták. A BSP-fa felépítésére Fuchs [142] javasolt heurisztikus szabályokat.

A mai grafikus hardver több szinten programozható, ezért a megjelenítési algoritmuslánc működését módosíthatjuk. Sőt arra is lehetőség van, hogy a grafikus hardveren nem grafikus számításokat végezzünk el. A grafikus hardver a nagyon nagyfokú párhuzamosságnak köszönhetően óriási teljesítményű, de a felépítése miatt csak speciális algoritmusok végrehajtására képes. Már megjelentek olyan, a grafikus hardverre optimalizált algoritmusok, amelyek általános célú feladatokat oldanak meg (lineáris egyenletek, gyors Fourier-transzformáció, integrálegyenletek megoldása stb.). Ilyen algoritmusokról a http://www.gpgpu.org honlapon és Randoma Fernando könyvében [128] olvashatunk.

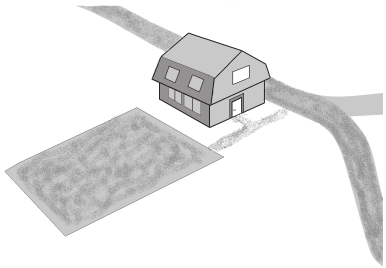
16. Térinformatika

A térinformatika (vagy geoinformatika) a térképészet és az informatika határán kialakult tudományterület.

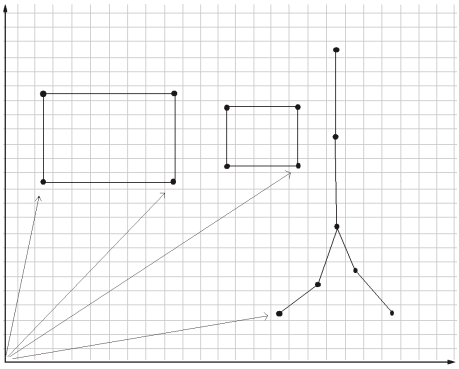
Ebben a fejezetben a legfontosabb térinformatikai fogalmakat és módszereket mutatjuk be: a 16.1. alfejezetben az adatmodellekkel, a 16.2. alfejezetben a térbeli indexeléssel, a 16.3. alfejezetben a digitális szűrési eljárásokkal, és végül a 16.4. alfejezetben a mintavételezéssel foglalkozunk.

16.1. A térinformatika adatmodelljei

A térinformatikában az adatok tekintélyes részét teszik ki a térbeliséget megtestesítő digitális térképek. Ezeknek alapvetően két nagy csoportja van: az egyik csoportot a vektoros adatmodellt követő, a másik csoportot pedig a raszteres adatmodellt követő térképek alkotják. A kétféle térkép nagymértékben különbözik egymástól, mivel lényegesen különböző adatmodellre épülnek. Az 16.1. ábrán a mintaterület egy nem térképszerű ábrázolását látjuk, amelynek vektoros, majd a raszteres reprezentációját is ismertetjük.



16.1. ábra. A mintaterület madártávlatból.



16.2. ábra. A mintaterület vektoros modellje.

16.1.1. A vektoros adatmodell

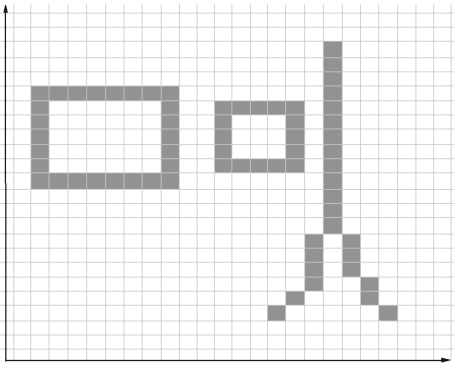
A ***vektoros rendszerek*** helyvektorokkal írják le az objektumok térbeliségét megadó jellegzetes pontokat (és egy összekötési szabállyal mondják meg, hogy mely pontok alkotnak poligont, vonalat vagy vonalláncot). A vektoros rendszerek nagy előnye, hogy nemcsak hierarchikusan felépülő komplex objektumok létrehozására alkalmasak, hanem a relációs adatkapcsolatok kiépítése is viszonylag könnyen megvalósítható. A vektoros térképekhez kapcsolódnak az objektumokat leíró adatok, amelyek megjelenítése és elemzése a térinformatikai funkciókör egyik fő feladata. A vektoros adatok erőforrásigénye lényegesen kisebb, mint a rasztereseké. A 16.2. ábrán az előbbi terület vektoros modellje látható.

16.1.2. A raszteres modell

A ***raszteres rendszerek*** világa a térinformatika másik jelentős területe. Adatainak forrása a digitális kép. A digitális kép elemi objektuma a ***pixel,*** azaz a legkisebb képpont, amit a képalkotó eszköz még képes létrehozni. A képalkotó eszköz nemcsak műhold lehet, hanem akár egy egyszerű szkenner, vagy egy digitális fényképezőgép. A pixel optikai állapota homogén, azaz színe, fényereje a pixelen belül állandó. A raszteres rendszerek a megfigyelt felületet egy $n \times m$-es mátrixra képezik le (n a sorok, m az oszlopok száma). A kép által átfogott földterület alapján a pixelnek valós méret is megfeleltethető, ekkor a kép térképként is használható. A 16.3. ábrán a mintaterület raszteres modelljét láthatjuk. A raszteres térképek legnagyobb előnye a felszín rendkívül részlet-gazdag leírása. Hátrányuk a nagy erőforrásigény, és a bonyolult objektumok felépítésének, valamint a relációs adatkapcsolatok létrehozásának nehézsége.

Gyakorlatok

16.1-1. Hasonlítsuk össze a vektoros és raszteres adatmodellt.



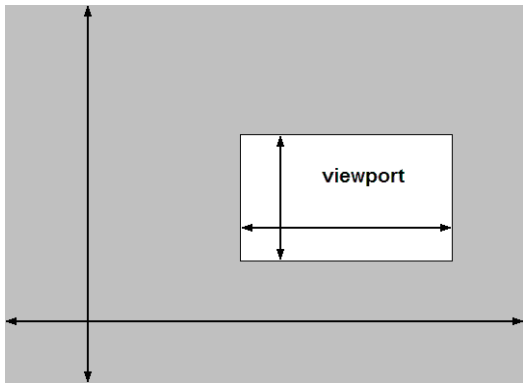
16.3. ábra. A mintaterület raszteres modellje.

16.2. Térbeli indexelés

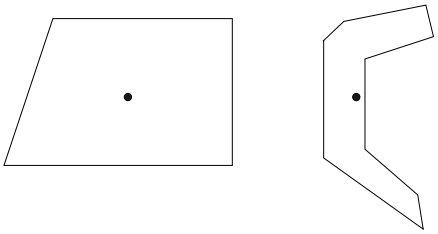
A digitális térképek kezelése – mind raszteres, mind a vektoros adatmodell esetében – általában nagy erőforrásigényű. Az ilyen térképekkel dolgozó térinformatikai rendszerek alapja általában valamilyen adatbáziskezelő rendszer, ami megoldja a térbeli objektumok tárolását, kezelését, valamint lehetőséget biztosít interaktív használatra. A megfelelő minőség biztosításához speciális módszerek állnak rendelkezésünkre, melyek támogatják a szintén speciális műveleteinket – mint amilyenek például az adott területre eső objektumok lekérdezése (tartomány-lekérdezések), adott koordinátát tartalmazó objektumok lekérdezése, legközelebbi szomszéd keresése, vagy a térbeli összekapcsolás.

A következőkben a legfontosabb műveletre, az adatbázis tartalmának megjelenítésére koncentrálunk. Megjelenítéskor megfeleltetést hozunk létre az adatbázis tartalma és a megjelenítő eszköz között (16.4. ábra). A szürke terület az adatbázisunk által lefedett síkrész teljes kiterjedését mutatja, míg a fehér terület az úgynevezett ***viewport,*** amely a megtekinteni kívánt területet jelzi. A megjelenítés legegyszerűbb módja lenne beolvasni az egész adatrendszert a memóriába, és onnan végezni a kirajzolást. Ez az eljárás több szempontból sem követendő. Egyrészt minek beolvasni olyan területek adatait, amik kívül esnek a viewporton, másrészt a memória mérete sem korlátlan, és célszerű csak a látni kívánt terület adatait betölteni. Bonyolítja a helyzetet, hogy a háttértár (lemez) lényegesen lassabb működésű, mint a memória, valamint hogy a háttértár adatait hatékonysági okokból nagyobb egységekben, úgynevezett *blokkokban* kezelhetjük. Egy elemi olvasási vagy írási művelet tehát egy blokk olvasását vagy írását jelenti, a művelet sebessége pedig alapvetően az érintett blokkok számától függ. A térképi adatok indexelése megfelelő segítséget nyújt az előzőekben felvázolt keresési problémák megoldásához. Az ***indexelés*** során olyan adatstruktúrákat hozunk létre, melyek segítségével adott tulajdonságú (például a viewportba eső) objektumok hatékonyan visszanyerhetők anélkül, hogy az összes objektum térbeli elhelyezkedését meg kellene vizsgálnunk.

A viewportba eső objektumok gyors megjelenítéséhez tehát ki kell tudnunk zárni a keresésből a viewporton kívüli területeket. Sajnos, a hagyományos adatbázis-kezelőkben megszokott indexelési technikák – mint az általánosan használt B-fa – térbeli indexelésre



16.4. ábra. A viewport és az adatbázis térbeli kiterjedésének viszonya.

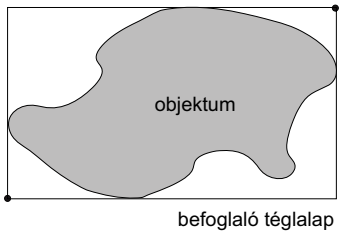


16.5. ábra. Konvex és konkáv poligonok centroidjai.

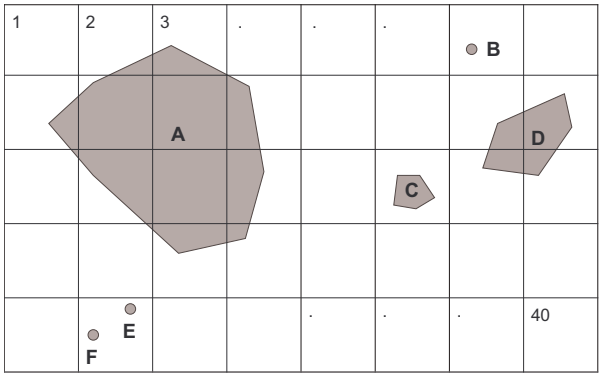
nem alkalmasak, mivel csak az egy dimenzió (attribútum) szerinti lekérdezéseket támogatják hatékonyan. Ezzel szemben a térinformatikai adatok lekérdezései jellemzően két vagy három dimenzió, azaz két vagy három koordináta szerinti kereséseknek felelnek meg.

Még mielőtt rátérnénk néhány térbeli indexelési algoritmus bemutatására, ismerkedjünk meg a centroid és a befoglaló téglalap fogalmával. A ***centroid*** egy olyan pont, amely a poligon belsejében van, és alkalmas a poligon térbeli elhelyezkedésének ábrázolására. Konvex poligonok esetében a centroid kézenfekvő definíciója a poligon súlypontja, míg konkáv esetben egy súlyvonalra illeszkedő belső pont (konkrét definíciója többféleképp is lehetséges és elterjedt) (16.5. ábra). A ***befoglaló téglalap*** (MBR) az a legkisebb területű téglalap, melynek oldalai párhuzamosak a koordináta-tengelyekkel, valamint teljes egészében tartalmazza a kérdéses objektumot (16.6. ábra). A befoglaló téglalap ábrázolásához mindössze két pont szükséges, a bal alsó és jobb felső csúcsok használata általánosnak mondható.

A következőkben feltesszük, hogy az adatbázisban síkbeli objektumokat tárolunk – a vektoros adatmodellnek megfelelően. Az objektumok lehetnek nulladimenziósak, vagyis egy darab ponttal reprezentáltak, lehetnek egydimenziósak, mint például utak, folyók vagy általában a vonalláncok, vagy kétdimenziósak, például poligonok. A fent bevezetett két fogalom a poligonok, vonalláncok, valamint általában síkbeli objektumok helyettesítésére való. Azért használjuk őket, mert a térbeli indexeléskor ezen egyszerű objektumokkal reprezentáljuk az időnként rendkívüli összetettségű térbeli objektumokat.



16.6. ábra. A befoglaló téglalap.



16.7. ábra. Különböző méretű objektumok egy adott rácsállandójú négyzethálón.

A térbeli adatok indexelési módszerei vagy a térbeli objektumokat, vagy pedig a vizsgált térrészt osztják fel részekre. A következőkben bemutatott módszerek a második kategóriába sorolhatók, tehát a tér valamely felosztásával próbálják gyorsítani az objektumok elérését.

16.2.1. Grid index

Illesszünk a síkbeli objektumainkat tartalmazó síkrészre egy szabályos négyzethálót, ahogy a 16.7. ábrán látható. A síkrészen a négyzetháló segítségével kijelölt felosztás azonos méretű, egymást nem átfedő négyzeteit nevezzük a továbbiakban ***celláknak,*** magát a négyzethálót ***gridnek,*** az objektumokat a felosztás celláihoz rendelő relációt pedig ***grid indexnek.*** Az indexelés során tároljuk, hogy az egyes síkbeli objektumok melyik cellába esnek.

Az objektumok koordináták szerinti gyors keresésének – mint alapvető fontosságú műveletnek – első lépése az úgynevezett első szűrés, amikor jelölteket állítunk a lekérdezés feltételeit kielégítő objektumok halmazára. Az index objektum azonosítói segítségével beolvassuk ezt a jelölthalmazt, ami a teljes adatbázisnál várhatóan jóval kevesebb objektumot tartalmaz. A jelöltállításhoz felhasználjuk, hogy a koordinátákra vonatkozó feltételekből gyorsan meghatározhatók a szóba jöhető cellák, valamint hogy a grid index a cella azonosítója szerint rendezett, ezáltal gyorsan kereshető formában tárolható. A grid index gyors

cella azonosító	objektum azonosító
7	B
22	C
34	F
34	E

objektum azonosító	egyéb attribútumok
A	
B	
C	
D	
E	
F	

16.8. ábra. Az index tábla és az objektumok jellemzőit tartalmazó tábla.

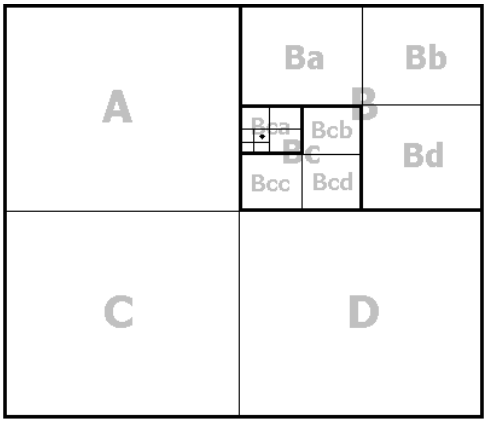
kereséséhez felhasználhatunk a cella azonosítójára vonatkozó hagyományos adatbázis indexeket is. Második lépésben egy második szűréssel dolgozzuk fel az így megtalált objektumok részletes geometriáját (például poligonok esetén a csúcsok koordinátáinak beolvasása itt történik), és ellenőrizzük a keresési feltételek teljesülését. A 16.8. ábra az index tábla felépítését szemlélteti a 16.7. ábra példáján keresztül. Az egyszerűség kedvéért csak azokkal az objektumokkal foglalkozunk, melyek csak egy cellához tartoznak, így elkerülve az objektum reprezentálásának problémáját. A grid index táblája a cella azonosítója alapján, az objektumokat tartalmazó tábla pedig az objektum azonosítója alapján gyorsan kereshető. Ennél fogva az objektumok koordináták szerinti keresései során az első szűrés gyorsan végrehajtható, valamint a második szűréshez szükséges jelöltek – várhatóan jóval kisebb számosságú – halmaza is gyorsan elérhető az objektumok táblájában. Ezzel szemben egy hasonló keresés az index tábla használata nélkül az objektumokat tartalmazó tábla teljes végigolvasását igényli, mivel a koordinátákra tett feltételek teljesülése csak az objektumok megfelelő attribútumainak egyenkénti feldolgozásával dönthető el.

Amint a 16.7. ábrán megfigyelhető, a cellák mérete nem mindig van összhangban az objektum méretével. Ha túl nagyok a cellák, akkor túl sok objektum esik egy adott cellába, ami szélsőséges esetben nem gyorsít a keresésen. Ha túl kicsik a cellák, akkor a nagyobb objektumok túl sok cellát fednek le, ami szintén nem előnyös. A probléma megoldható több, eltérő rácsállandójú grid-szint bevezetésével.

A grid index egyszerű és hatékony módja a térbeli indexelésnek. Számos kereskedelmi szoftver alkalmazza is, annak ellenére, hogy hiányosságai nyilvánvalók, különösen olyan esetekben, amikor az objektumok térbeli eloszlása nem egyenletes. Az volna ugyanis a kívánatos, hogy lehetőleg minden cellába azonos számú objektum essen, ami az állandó rácsméret miatt nagyon kis valószínűséggel fordul elő. Az objektumok inhomogén térbeli eloszlása esetén a második szűrési fázis hatékonysága erősen lecsökkenhet, ezért összetettebb és rugalmasabb indexelési algoritmusok használata is szükségessé válhat.

16.2.2. Négy-fa

A ***négy-fa*** egy hierarchikus adatszerkezet, amely az oszd-meg-és-uralkodj elvhez hasonló rekurzív felbontás elvén alapszik. A sokféle négy-fa változatból mi most az úgynevezett pont-tartomány négy-fát vázoljuk fel. Az indexelés alapja az adott síktartomány rekurzív felosztása négy síknegyedre (16.9. ábra).

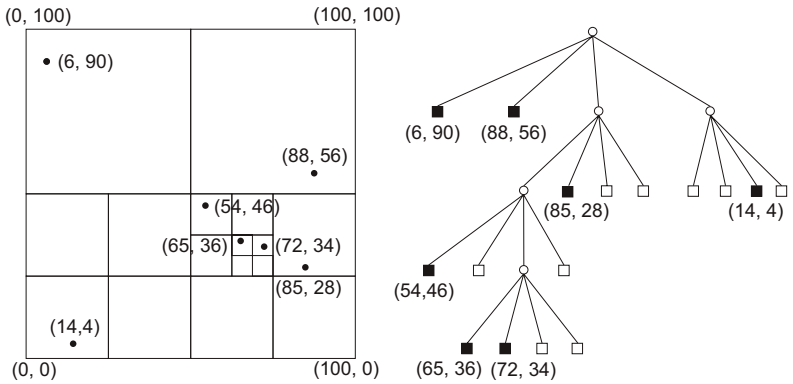


16.9. ábra. A rekurzív térfelosztás.

Az indexeléshez használt struktúra egy olyan fa szerkezetként ábrázolható, melynek minden belső csúcsa egy-egy síktartományt reprezentál, leveleiben pedig az objektumok helyezkednek el. A fa gyökere megfelel a kezdeti tartománynak. Minden belső csúcs tartalmazza saját felosztásának középpontját, valamint négy mutatót a felosztás után keletkező négy új tartománynak megfelelő csúcsokra (jelölje ezeket az égtájak angol rövidítéseinek megfelelően *NW*, *NE*, *SE* és *SW*). Ha egy csúcs levél, akkor a csúcs egy ezt jelző mezőjét igazra állítjuk. A levelek vagy üresek – ezt egy külön mezőben feljegyezzük –, vagy pedig egy objektumot tartalmaznak, koordinátáival és egyéb attribútumaival, vagy ezt kiváltandó egy mutatót az objektum adatbázisbeli reprezentációjára. A levelek *NW*, *NE*, *SE* és *SW* mezői definiálatlanok. Megjegyzendő, hogy a belső csúcsok felosztást reprezentáló pontját csúcsonként tárolni nem feltétlenül szükséges. Mivel a felosztás mindig egyértelmű, ezért a fa műveleteinek megfelelő megvalósításával a felosztásra vonatkozó információk tárolása a csomópontokban elkerülhető. A 16.10. ábra egy hét objektumot tartalmazó négy-fát szemléltet, feltüntetve az objektumok koordinátáit.

A fa felépítése az objektumok egyenkénti beszúrását jelenti. Ehhez elindulunk a fa gyökerétől, és egészen addig haladunk lefelé, amíg levélbe nem érkezünk. A bejárás során a beszúrandó objektumot reprezentáló pont koordinátái alapján választjuk ki az adott csomópont megfelelő gyerekét. Ha levélbe érkeztünk és az üres, beszúrjuk az új pontot. Ha a levél már tartalmaz egy objektumot, akkor az aktuális tartományt újra és újra fel kell osztanunk, egészen addig, amíg a régi és új csúcs már nem esik azonos tartományba. Ez a felosztás sok új altartomány létrehozását is jelentheti, ha a régi és új pontok távolsága kicsi. A felosztásokat megszüntethetjük pontok törlésekor. Ha egy pont törlésével az adott tartomány már csak egy pontot tartalmaz, akkor az altartományai összevonhatók.

Nézzük meg, hogyan valósítható meg egy olyan keresés, amely a négy-fában reprezentált, viewportba eső objektumokat állítja elő. A NÉGY-FA-KERES algoritmus adott (x_1, y_1) és (x_2, y_2) pontokkal definiált viewportba eső objektumokat adja vissza, ahol $x_1 < x_2$ és $y_1 < y_2$, valamint feltesszük, hogy a viewport a reprezentált síkrészbe esik. A keresés során egy mélységi bejárásnak megfelelő sorrendben bejárjuk azokat a csúcsokat, melyek tartalmazhatnak a viewportba eső objektumot. Az eljárás megkapja az épp aktuális csúcsot (*n* pa-



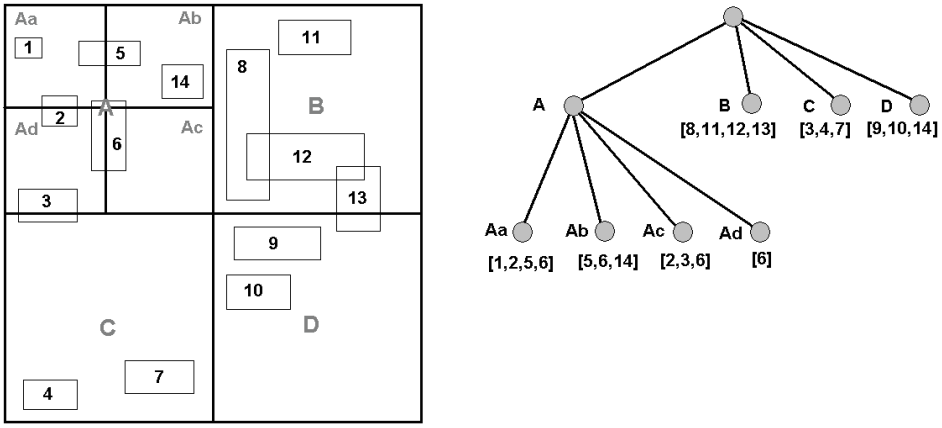
16.10. ábra. Hét objektumot tartalmazó pont-tartomány négy-fa.

raméter), valamint a viewport koordinátáit. Az n csúcs koordinátáit $koordX[n]$ és $koordY[n]$, levél tulajdonságát $levél[n]$, üres voltát $üres[n]$, az általa reprezentált felosztás középpontjának koordinátáit pedig $felosztásX[n]$ és $felosztásY[n]$ mezők tartalmazzák. A teljes fában való kereséshez a fa gyökerével kell elindítani az algoritmust.

NÉGY-FA-KERES(n, x_1, y_1, x_2, y_2)

```
1  if levél[n]
2    then if ¬ üres[n] ∧ x1 ≤ koordX[n] ≤ x2 ∧ y1 ≤ koordY[n] ≤ y2
3           then return n                ▷ Megtaláltunk egy viewporton belüli objektumot.
4    else if felosztásX[n] > x1 ∧ felosztásY[n] < y1        ▷ Gyerekek rekurzív bejárása.
5           then NÉGY-FA-KERES(NW[n], x1, y1, x2, y2)
6         if felosztásX[n] < x2 ∧ felosztásY[n] < y1
7           then NÉGY-FA-KERES(NE[n], x1, y1, x2, y2)
8         if felosztásX[n] < x2 ∧ felosztásY[n] > y2
9           then NÉGY-FA-KERES(SE[n], x1, y1, x2, y2)
10        if felosztásX[n] > x1 ∧ felosztásY[n] > y2
11          then NÉGY-FA-KERES(SW[n], x1, y1, x2, y2)
```

A fa mérete és alakja fontos a keresés, beszúrás és törlés műveletek hatékonyságának szempontjából. A pont-tartomány négy-fa szerkezete a fix térfelosztás miatt független a beszúrások sorrendjétől, alakja és mérete viszont függ a beszúrt objektumoktól. A fa műveleteinek futásideje alapvetően a fa mélységétől függ. A fa minimális mélysége M objektum esetén $\lceil \log_4(M-1) \rceil$, ekkor minden objektum azonos szinten helyezkedik el, a fa kiegyensúlyozott. A fa azonban általában nem rendelkezik ezzel a hasznos tulajdonsággal, sőt felső korlátot sem tudunk adni mélységére – az függ ugyanis az objektumok páronkénti távolságától. Ahhoz ugyanis, hogy a fába beszúrjunk két pontot, mindenképp fel kell építenünk a fát legalább olyan mélységben, hogy a két pont közé essen egy tartomány-határ. Minél kisebb viszont a pontok távolsága, várhatóan ez annál később (annál mélyebb fát felépítve) következik be. Ha feltesszük, hogy az objektum-halmazunkban két pont távolsága



16.11. ábra. Négy-fa és a poligonok befoglaló négyszögei.

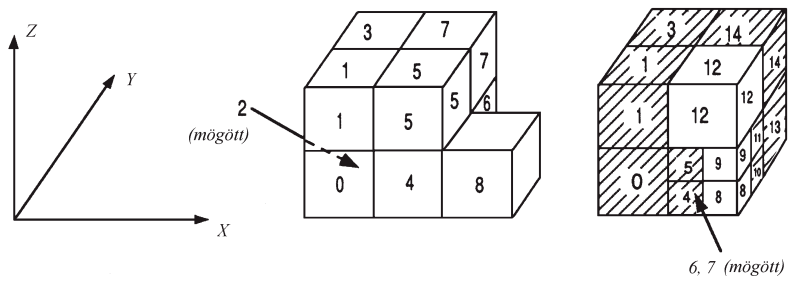
legalább d, valamint a tartományunk kiterjedése kezdetben r, akkor a fa mélységére adható egy $\lceil \log_2(\sqrt{2}(r/d)) \rceil$ korlát. Legrosszabb esetben ugyanis egy d hosszú szakasz valamely koordináta-tengelyre eső vetülete $d/\sqrt{2}$ hosszú, amiből legfeljebb $r/(d/\sqrt{2})$ darab illeszthető egy r hosszú szakaszra.

Indexeléshez célszerű lehet egy olyan négy-fa változat használata, ahol a tartományok felosztásának feltételét a blokkmérethez igazítjuk, azaz csak akkor osztjuk tovább az adott síkrészt, ha a síkrészbe eső pontok tárolásához szükséges lemezterület túlcsordul a blokk méretén. A fa ezen változatában a levelek nem egy-egy objektumot, hanem objektumok halmazát tartalmazzák, melyeket azonos blokkban tárolunk.

Poligonok esetére a helyzet nem olyan egyszerű, mint pontokra. A poligon összetett objektum, amelyet számos pont alkot. A poligonok indexeléséhez felhasználjuk a centroid vagy a befoglaló négyszög fogalmát. Mindkét helyettesítő objektum alkalmas arra, hogy valamely térnegyedbe történő besoroláshoz megfelelően reprezentálja a poligont. A centroid egy és csak egy térnegyedbe való besorolást enged meg, míg a befoglaló négyszög alkalmazásával egy-egy objektum több térnegyedbe is eshet, amint az a 16.11. ábrán látható.

Az eddigi példákban vektoros modellt követő adatokra vizsgáltuk a négy-fa algoritmust. A módszer raszteres adatmodellre is jól alkalmazható, mivel az adatok térbeli eloszlása teljes mértékben egyenletes, tekintettel a raszteres modell természetére.

Raszteres adatmodell esetében célszerű lehet az úgynevezett ***tartomány négy-fák*** használata. Ezek felépítése hasonló, mint a pont-tartomány négy-fáké, csak a levelek az adott al-tartomány egészére vonatkozó információkat tárolnak. Ez a fa jól használható például képek tömörítésére: végezzük a felosztást addig, amíg az al-tartomány homogén területet nem reprezentál, ekkor a tartomány intenzitását tároljuk a levelekben.



16.12. ábra. A nyolc-fa térfelosztása.

16.2.3. Nyolc-fa

A négy-fa háromdimenziós kiterjesztése a ***nyolc-fa***. Speciális esetekben használatos, mint például háromdimenziós objektumok (geológiai képződmények, földalatti olajtárolók stb.) leírására. A 16.12. ábra a nyolc-fa térfelosztását szemlélteti.

Osszuk a teret nyolc tértartományra, majd minden tartományt további nyolcra egészen addig, amíg a fa adott szintjén lévő térnyolcadban már csak egyetlen pont van. A négy-fa mintájára megalkotható egy olyan fa, amely segítségével az adott objektum attribútumaival együtt gyorsan azonosítható.

A 16.12. ábra tartományait egy egyértelmű rendezés szerint megszámoztuk. Ezen és hasonló rendezések célja az, hogy háromdimenziós tértartományok (vagy az őket reprezentáló pontok) egy egyértelmű sorrendjét állítsuk elő. Ezzel lehetővé válik, hogy a tértartományokat – vagy általánosabban tetszőleges térbeli objektumhalmazt – egydimenziós térbe képezzünk le, ahol már használhatjuk a hagyományos, elterjedt indexelési technikákat, például a B-fát.

A legismertebb eljárás olyan térbeli görbe előállítására, amely egyszer érinti a tér minden objektumát, az úgynevezett ***Peano rendezésen*** alapszik. Lényege, hogy összefésüljük az adott pont kettes számrendszerben felírt koordinátáit valamely rögzített sorrendben (például x első bitje, y első bitje, z első bitje, x második bitje és így tovább). A pontokhoz így rendelt értékek szerinti növekvő sorrendet használva oldjuk meg a tér pontjainak bejárását, rendezését.

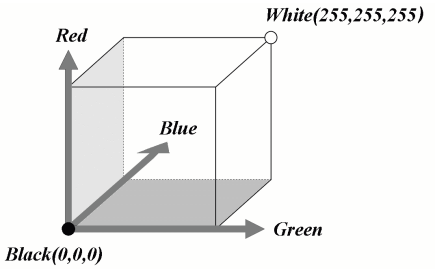
Gyakorlatok

16.2-1. Bővítsük a 16.10. ábrán szereplő négy-fát egy $(60, 38)$ és egy $(67, 30)$ koordinátájú objektummal.

16.2-2. Lássuk be, hogy adott koordinátájú pont keresése a négy-fában $O(h)$ időben megoldható, ahol h a fa mélysége.

16.3. Digitális szűrési eljárások

A térinformatika egy másik fontos területe a digitális képek, űrfotók feldolgozása. A képi adatokat kezelő szoftverek számos képfeldolgozó, szűrő algoritmuson alapuló eljárást tar-



16.13. ábra. Az RGB színkocka.

talmaznak. Ezen eljárások közül mutatunk be néhány fontosabbat, mint például alul- és felülvágó szűrők, élmegőrző, éldetektáló szűrők. A digitális képek feldolgozása során számos szűrési eljárásnak vetjük alá a képeket annak érdekében, hogy számunkra kedvező hatást, változást érjünk el. Felhasználási céljainktól függően lehetséges, hogy szeretnénk eltüntetni a képekről a nagyfrekvenciás változásokat, vagyis simító szűrést hajtunk végre, vagy szeretnénk kiemelni a képen látható éleket. Az eljárások megértéséhez át kell tekintenünk néhány alapfogalmat.

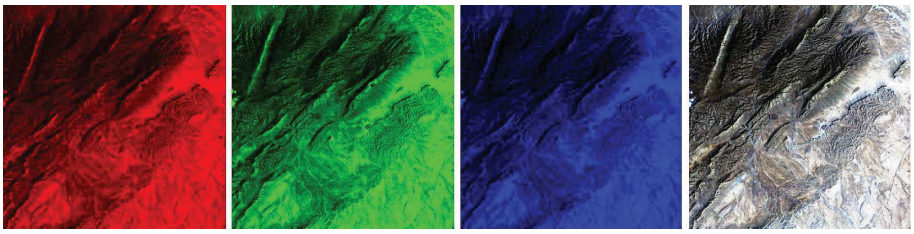
16.3.1. Az RGB színmodell

A számítástechnika fejlődésével lehetővé vált, hogy a szemünk színlátó képességét meghaladó számú színt legyünk képesek ábrázolni. Amióta létezik a 24 bites színmodell, valósághűnek látszanak a digitális képek. Legyen három koordináta tengelyünk, amely a három alapszínt szimbolizálja, RGB: *Red* (vörös), *Green* (zöld) és *Blue* (kék). E három szín intenzitását ábrázoljuk 0–255 között. A teljesen sötét zöld (vagyis fekete) legyen 0, a teljesen világos pedig 255, valamint a többi két alapszínre is ugyanígy járjunk el. Az abszolút fekete pontban (Black) mindhárom alapszín intenzitás értéke 0, míg az abszolút fehér pontban (White) 255. A 24 bites színmodellt szemlélteti az 16.13. ábra. Bármely szín előállítható e három alapszín kombinációjából, a következő módon:

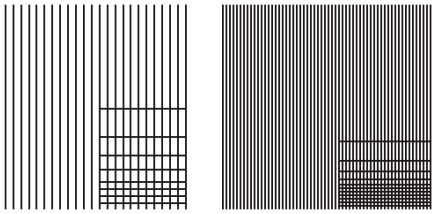
$$colour = a \cdot Red + b \cdot Green + c \cdot Blue \ ,$$

ahol a, b, c az egyes alapszínek intenzitása.

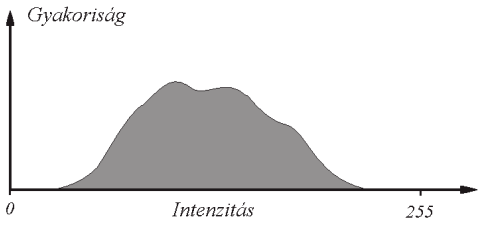
Egy-egy alapszín intenzitása tehát 256 féle értéket vehet fel, így 256 féle zöld árnyalatból, 256 féle vörös árnyalatból és 256 féle kék árnyalatból állítható össze egy tetszőleges szín, vagyis összesen $256 \cdot 256 \cdot 256$ szín ábrázolható. Mivel $256 = 2^8$, így $2^8 \cdot 2^8 \cdot 2^8 = 2^{24}$ féle színárnyalatot tudunk előállítani. A 16.14. ábrán a NASA egy LANDSAT műholdja által készített kép három RGB sávjának intenzitásait és az azokból képzett színes képet láthatjuk. Az erőforrás kutató műholdak frekvencia sávokban érzékelnek, amelyek nemcsak a látható tartományt fogják át, hanem gyakran infravörös sávokat is. Így az RGB színkocka több dimenziós is lehet, mint három. Az RGB színkocka mintájára a látható színeket más sávokkal is (pl. infravörös sávok) helyettesíthetjük, így hamis színes képeket állíthatunk elő. Vezessük még be a térbeli frekvencia fogalmát, amely az egységnyi távolságra eső különböző intenzitás értékek számát méri. A 16.15. ábra a térbeli frekvencia fogalmát szemlélteti.



16.14. ábra. Egy LANDSAT műhold felvétele: az RGB sávok és a sávokból összeállított kép. Az ábra színes változata az 812. oldalon látható.



16.15. ábra. A térbeli frekvencia szemléletes jelentése.

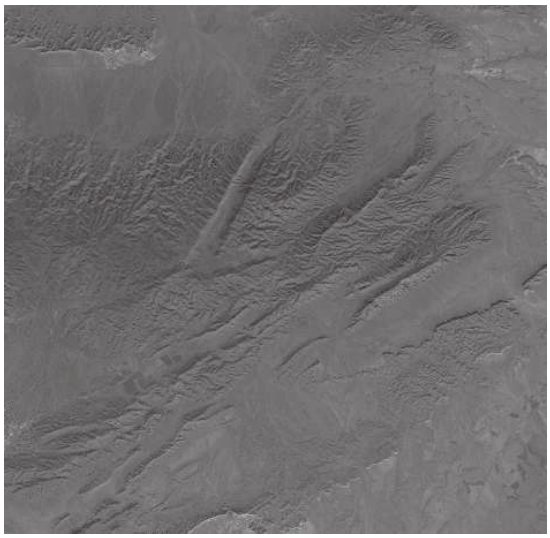


16.16. ábra. Egy kép intenzitás értékeinek eloszlása (hisztogramja).

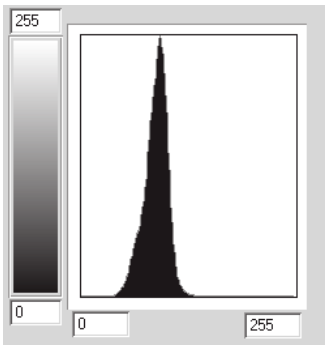
16.3.2. Hisztogram kiegyenlítés

A digitális leképező módszerek és a környezeti körülmények miatt a digitális képek intenzitásának dinamika tartománya kisebb, mint a lehetséges, vagyis a kép legsötétebb pontja általában világosabb, mint az abszolút fekete pont, valamint a legvilágosabb pontja sötétebb, mint az abszolút fehér pont, ahogy az a 16.16. ábrán látható. Toljuk el a kép legsötétebb pontjának intenzitását az abszolút fekete pontba, a legvilágosabb pont intenzitását az abszolút fehér pontba, és az összes többi pont intenzitás ezzel arányosan változtassuk meg. Ezzel megnöveltük a pixelek közti intenzitás különbséget, ezáltal kontrasztosabbá tettük a képet.

Tekintsünk egy valós esetet, mégpedig a 16.17. ábrát, mely egy francia SPOT műhold által készített kép. A kép hisztogramját a 16.18. ábra mutatja. Amint látható elég keskeny sávban mozognak a kép intenzitás értékei, ezért is találjuk a képet kicsit túl szürkének. Végezzük el a kontrasztnövelő transzformációt, amelynek az eredményét a 16.19. ábra mutatja.



16.17. ábra. Az eredeti SPOT felvétel.



16.18. ábra. A SPOT kép intenzitás eloszlása.

16.3.3. Fourier-transzformáció

A Fourier-transzformációt számos mű részletesen tárgyalja, tehát csak érintőlegesen mutatjuk be a legfontosabb összefüggéseket. Legyen $s(t)$ az idő nem periodikus függvénye. Írjuk fel a következő kifejezést:

$$S(f) = \int_{-\infty}^{\infty} s(t)e^{-2\pi i f t}dt \ ,$$

ahol $S(f)$ az $s(t)$ függvény ***Fourier-transzformáltja,*** vagyis a frekvencia tartománybeli „képe", f a frekvencia és $i = \sqrt{-1}$. $S(f)$-t gyakran nevezik $s(t)$ ***spektrumának*** is, az eljárást pedig ***Fourier-transzformációnak.***

16.19. ábra. A hisztogram kiegyenlítéssel megnövelt kontrasztú kép. Az ábra színes változata az 813. oldalon látható.

Írjuk fel a következő kifejezést, amelyet inverz Fourier-transzformációnak is neveznek:

$$s(t) = \int_{-\infty}^{\infty} S(f) e^{2\pi i f t} dt \ .$$

A Fourier-transzformált létezésének elégséges feltétele, hogy $s(t)$ függvény szakaszonként sima (vagyis deriváltja véges számú hely kivételével folytonos), valamint az

$$\int_{-\infty}^{\infty} |s(t)| dt$$

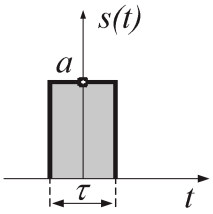
kifejezés konvergens legyen. Ebben az esetben az $S(f)$ Fourier-transzformált meghatározható.

16.3.4. Néhány speciális függvény Fourier-transzformáltja

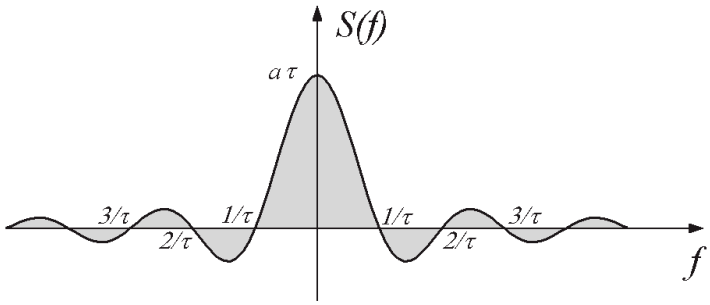
A szűrési eljárások bemutatásakor szükségünk lesz néhány speciális függvény Fourier-transzformáltjára. Az egyik ilyen kiemelten fontos függvény a négyszög függvény.

Négyszögimpulzus
Számunkra az úgynevezett ***tranziens függvények*** érdekesek, vagyis amelyek nem periodikusak, hanem véges hosszúságú intervallumon különböznek a nullától. Vizsgáljuk meg néhány tranziens függvény Fourier-transzformáltját. Az egyik fontos függvény a négyszögimpulzus (16.20. ábra), amely a következő:

$$s(t) \quad = \quad \begin{cases} a, & \text{ha } |t| < \tau/2 \ , \\ 0 & \text{egyébként} \ . \end{cases}$$



16.20. ábra. A négyszögimpulzus.



16.21. ábra. A négyszög impulzus Fourier-transzformáltja a sinc függvény.

Végezzük el a függvény Fourier-transzformációját:

$$S(f) = \int_{-\infty}^{\infty} s(t)e^{-2\pi i f t}dt = a\int_{-\tau/2}^{\tau/2} e^{-2\pi i f t}dt = a\tau\frac{\sin \pi f \tau}{\pi f \tau} \ .$$

Ezt a függvényt szinusz kardinálisz függvénynek is nevezik és *sinc*-vel jelölik (16.21. ábra). Kézenfekvő, hogy egy frekvencia tartománybeli négyszög függvény Fourier-transzformáltja az idő tartománybeli *sinc* függvény lesz. Ennek a ténynek nagy jelentősége lesz az ideális felülvágó szűrő megvalósításakor.

Nézzük meg most azt a speciális esetet, amikor a négyszög egységnyi területű, vagyis $a\tau = 1$, vagyis $a = 1/\tau$. Ez szavakban kifejezve annyit jelent, hogy minél keskenyebb a jel, annál magasabb lesz a négyszögimpulzus.

Dirac-δ

Paul Dirac vezette be a róla elnevezett δ-t pontszerű objektumok leírására. Használata megkönnyíti a digitális adatrendszerek leírását számos vonatkozásban. Definíció szerint legyen δ egy olyan függvény (klasszikus értelemben nem az, hanem úgynevezett disztribúció), melyre $\delta(t) = 0$ ha $t \neq 0$, és

$$\int_{-\infty}^{\infty} \delta(t)dt = 1 \ .$$

A ***Dirac-δ*** szemléletes jelentése egy végtelenül keskeny, és végtelenül magas amplitúdójú impulzus, amelynek görbe alatti területe egységnyi. Számunkra azért fontos a δ-val való

foglalkozás, mert számos olyan tulajdonsága van, amely a jelfeldolgozás szempontjából lényeges. A $t = t_0$ esetre alkalmazva a δ-t, azt kapjuk, hogy $\delta(t - t_0) = 0$, ha $t \neq t_0$.

Lássuk egyik fontos tulajdonságát, amelyet kiválasztó tulajdonságnak nevezhetünk. Szorozzunk meg az $f(t)$ függvényt a $\delta(t - t_0)$ -val. Ekkor

$$\delta(t - t_0)f(t) = \delta(t - t_0)f(t_0) \ ,$$

mivel

$$\int_{-\infty}^{\infty} \delta(t - t_0)f(t)dt = f(t_0) \ .$$

Ez a kiválasztási képesség, vagyis a $\delta(t_0)$ az $f(t)$ függvényből kiválasztotta a t_0 -hoz tartozó értéket.

A következőkben vizsgáljuk meg egy Dirac-impulzus és egy Dirac-impulzus sorozat Fourier-transzformáltját. Jelölje F a Fourier-transzformációt, amely a definíciója alapján

$$F\{\delta(t)\} = \int_{-\infty}^{\infty} \delta(t)e^{-2\pi i f t} = 1 \ .$$

Nem az origóban lévő Dirac-impulzus Fourier-transzformáltja pedig

$$F\{\delta(t - a)\} = \int_{-\infty}^{\infty} \delta(t - a)e^{-2\pi i f t} = e^{-2\pi i f a} \ ,$$

amiről Euler óta tudjuk, hogy

$$e^{-2\pi i f a} = \cos 2\pi f a - i \sin 2\pi f a \ .$$

Dirac-impulzus sorozat (az idő tartományban) Fourier-transzformáltja:

$$F\left\{\sum_{k=-\infty}^{\infty} \delta(t - k\tau)\right\} = \frac{1}{\tau}\sum_{k=-\infty}^{\infty} \delta(f - k/\tau) \ .$$

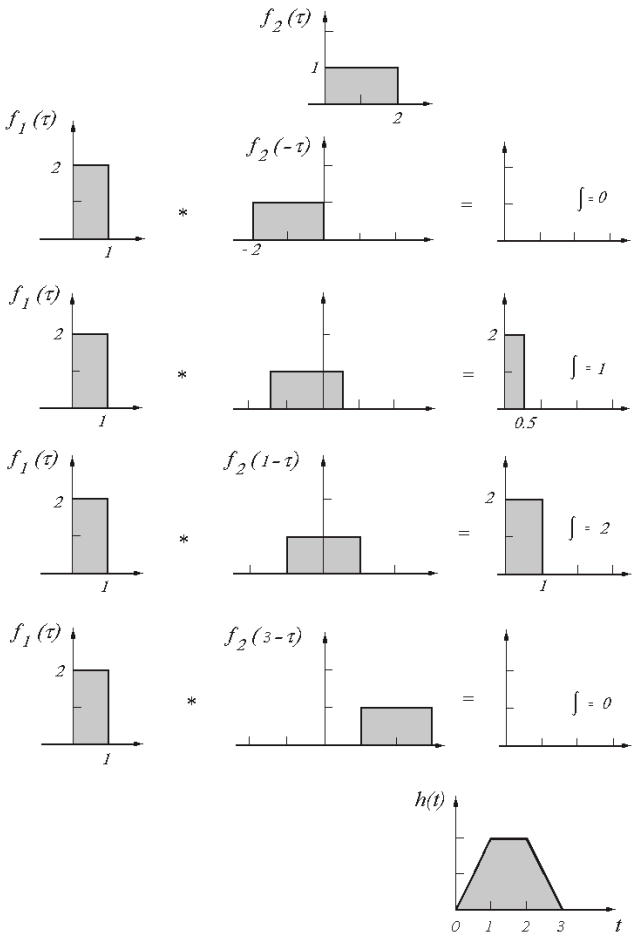
16.3.5. Konvolúció

Legyenek f_1 és f_2 folytonos függvények. Jelölje ***konvolúciójukat*** $h = f_1 * f_2$, melyet a következő kifejezés definiál:

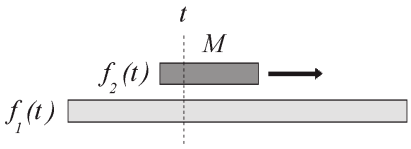
$$h(t) = f_1(t) * f_2(t) = \int_{-\infty}^{\infty} f_1(\tau)f_2(t - \tau)d\tau \ .$$

A 16.22. ábra grafikusan szemlélteti két négyszög függvény konvolúcióját az idő tartományban.

Vizsgáljuk meg egy konkrét esetet, ahol digitális jelekre alkalmazzuk az összefüggést: legyen $h(t)$ a t-edik pillanatban az f_1 és f_2 függvények konvolúciója, amit úgy kapunk, hogy az f_1 t-edik pillanatban felvett értékét összeszorzunk az f_2 $(t - \tau)$ -edik értékével, majd végigfutunk f_2 egész intervallumán (ami valóságos esetekben véges intervallum, jelen esetben M) és összegezzük a szorzatokat (futó összegzés). A folyamatot a 16.23. ábra szemlélteti.



16.22. ábra. Két négyszög függvény konvolúciója az idő tartományban.



16.23. ábra. A konvolúció szemléletes jelentése.

Az eddigiekben csak egydimenziós függvényekkel foglalkoztunk. Könnyen általánosítható a konvolúció fogalma kétdimenziós függvényekre is, mint amilyen a digitális kép.

A konvolúció tulajdonságai
A következőkben röviden összefoglaljuk a konvolúció főbb tulajdonságait, bizonyítás nélkül.

- A konvolúció kommutatív:

$$f(t) * g(t) = g(t) * f(t) \ .$$

- A konvolúció asszociatív:

$$f(t) * [g(t) * h(t)] = [f(t) * g(t)] * h(t) \ .$$

- A konvolúció disztributív:

$$f(t) * [g(t) + h(t)] = f(t) * g(t) + f(t) * h(t) \ .$$

- Szorzat Fourier-transzformáltja a tényezők spektrumainak konvolúciója. Jelölje F a Fourier-transzformációt. Ekkor

$$F\{g(t)h(t)\} = G(f) * H(f) \ ,$$

  ahol $G(f)$ és $H(f)$ a $g(t)$ és $h(t)$ függvények spektrumai.
- Függvények konvolúciójának Fourier-transzformáltja a spektrumok szorzata:

$$F\{g(t) * h(t)\} = G(f)H(f) \ ,$$

  ahol $G(f)$ és $H(f)$ az $g(t)$ és $h(t)$ függvények spektrumai.
- Konvolváljuk az $f(t)$ függvényt egy Dirac-impulzussal.

$$f(t) * \delta(t - t_0) = f(t - t_0) \ .$$

  A Dirac-impulzus eltolja a függvényt a saját argumentumában szereplő helyre.
- Szorzás végtelen Dirac-δ sorozattal:

$$g(t) \sum_{k=-\infty}^{\infty} \delta(t - k\tau) = \sum_{k=-\infty}^{\infty} g(k\tau)\delta(t - k\tau) \ .$$

A végtelen Dirac-δ sorozattal szorozva csak a $k\tau$ helyeken felvett értékeket tartjuk meg, és ez éppen a mintavételezésnek felel meg.

16.3.6. Szűrési algoritmusok

A digitális szűrési módszerek egyik legfontosabb fogalma a ***kernel.*** Jelentése mag. A szűrési eljárások, amikor az idő tartományban dolgoznak, a kernellel konvolválják a szűrendő képet. A szűrés hatása attól függ, hogy milyen függvény értékeit tesszük be a kernelbe, ami egy $n \times n$-es mátrix. Ha meg tudjuk adni, hogy milyen átviteli függvényt kívánunk megvalósítani a frekvencia tartományban, akkor annak inverz Fourier-transzformálásával megkapjuk az idő tartománybeli függvényt, amelyet megfelelően mintavételezve megkapjuk a szűrőegyüttható kat, vagyis a kernelbe töltendő szűrőegyütthatókat. A digitális konvolúció tehát a szűrések végrehajtásának egyik lehetséges módja. Ebben az esetben a szűrést az idő tartományban végezzük a következő módon:

$$g'(t) = g(t) * s(t) \ ,$$

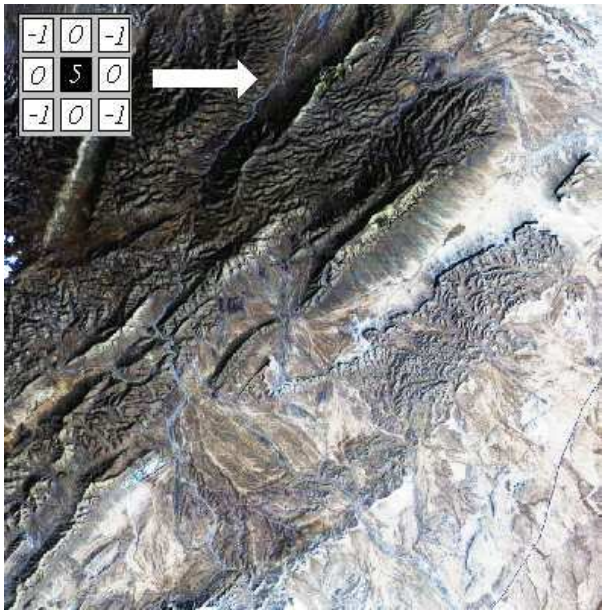
ahol $g(t)$ az eredeti adatrendszer az idő tartományban, $g'(t)$ a szűrt adatrendszer és $s(t)$ a kernel.

Egy másik lehetséges megoldás, hogy a szűrendő adatrendszert Fourier-transzformáljuk, majd a frekvencia tartományban végezzük el a szűrést (a Fourier-transzformáltat megszorozzuk a kívánt hatást biztosító átviteli függvénnyel), majd az így kapott spektrumot inverz Fourier-transzformáljuk.

$$\begin{aligned} G(f) &= F\{g(t)\} \ , \\ G'(f) &= G(f)S(f) \ , \\ g'(t) &= F^{-1}\{G'(f)\} \ , \end{aligned}$$

ahol $g(t)$ az eredeti adatrendszer, $G(f)$ az adatrendszer Fourier-transzformáltja, $S(f)$ a kívánt átviteli függvény, $G'(f)$ a szűrt adatrendszer a frekvencia tartományban, $g'(t)$ a szűrt adatrendszer az idő tartományban, F a direkt, és F^{-1} az inverz Fourier-transzformációt szimbolizálja. A digitális Fourier-transzformáció gyors végrehajtásához létezik az úgynevezett gyors Fourier-transzformáció algoritmus (FFT), amely gyorsan képes elvégezni a transzformációt. A kernel megállapítása nemcsak a frekvencia szerinti szűrők esetén játszik kulcsfontosságú szerepet, hanem más egyéb esetekben is, mint például az élmegőrző, élkiemelő szűrők. Az elérendő cél néha olyan, hogy nem adható meg egy egyszerű átviteli függvénnyel a művelet, hiszen pontról pontra változhat az algoritmus által előírt tennivaló.

A Konvolúció algoritmus az előzőekben felvázolt szűrési módszer egy lehetséges megvalósítása. Bemenetként megkap egy $M \times M$ méretű F képet és egy $K \times K$ méretű H konvolúciós mátrixot (kernelt), eredményként pedig visszaadja a G képet, amely az eredeti F kép konvolúciója a H kernellel. A képek, valamint a kernel pontjainak indexelését nullával kezdjük. A képekről az összeadás és szorzás műveletek értelmezhetőségének érdekében feltesszük, hogy szürkeárnyalatosak. A feldolgozás során az algoritmus illeszti a kernelt F kezdeti sarkába, majd meghatározza a kernel középpontjának megfelelő kimeneti képpont színét a kernel által lefedett képpontok súlyozott összegeként. Ezután elcsúsztatja a kernelt vízszintesen majd függőlegesen egészen addig, amíg az összes lehetséges illesztés meg nem történt. A 16.24. ábra a feldolgozás folyamatát szemlélteti egy háromszor hármas – erősen felnagyított – kernellel.



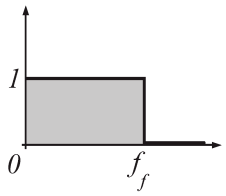
16.24. ábra. Az erősen felnagyított kernel mozgása a képen (LANDSAT képrészlet). Az ábra színes változata az 813. oldalon látható.

KONVOLÚCIÓ(F, H, K, M)

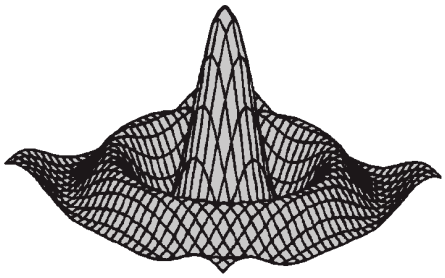
```
1  N ← K^2                                          ▷ Beállítjuk a normalizációs konstanst.
2  for x ← 0 to M − K
3      do for y ← 0 to M − K                        ▷ Sorra vesszük az eredeti kép pontjait.
4          do for i ← 0 to K − 1
5              do for j ← 0 to K − 1                ▷ Sorra vesszük a kernel pontjait.
6                  do G(x, y) ← G(x, y) + (H(i, j)F(x + i, y + j))/N
7  return G
```

Figyeljük meg, hogy a kimeneti G kép tetszőleges (x, y) pontja nem esik egybe az eredeti F kép (x, y) pontjával. Ennek oka, hogy az eredeti kép határaira csak úgy tudjuk illeszteni a kernel középpontját, hogy az „lelóg” az eredeti képről, tehát a művelethez szükséges környezet nem teljesen ismert. Ezt a problémát a KONVOLÚCIÓ algoritmus egyszerűen úgy oldja meg, hogy az eredményként keletkező kép nem tartalmazza az itt említett határpontokat. Csak azon pontokhoz rendel pontot az eredmény képben, melyekre megfelelően illeszthető a konvolúciós mátrix. Egy másik lehetséges megoldás a hiányzó szomszédsági pontok pótlása valamilyen értékkel, például a határolópontok másolása az ismeretlen területre.

Az N normalizációs konstans elméletileg tetszőleges (nullától különböző) értéket felvehet, az algoritmus most a kernel pontjainak számával normalizál.



16.25. ábra. A felülvágó szűrő átviteli függvénye (Fourier-transzformáltja a *sinc* függvény).



16.26. ábra. A kétdimenziós *sinc* függvény, mint az ideális felülvágás kernel függvénye.

A feldolgozás során bejárjuk az eredeti $M \times M$ méretű F kép minden – nem képhatáron lévő – pontját, melyek száma $(M-K+1)^2$, mindig feldolgozva a $K \times K$ méretű H kernel által lefedett K^2 darab pontot. Ezt figyelembe véve – vagy megvizsgálva az egymásba ágyazott ciklusokat – a KONVOLÚCIÓ algoritmus futási ideje $O((M-K+1)^2K^2)$.

Felülvágó szűrő

Akkor használunk felülvágó szűrőt, amikor a frekvencia tartományban egy bizonyos felső határfrekvenciánál (f_f) nagyobb frekvenciákat 0-val szorzunk, és a nála kisebbeket 1-gyel. A 16.25. ábra mutatja az ideális felülvágás átviteli függvényét. Ami a frekvencia tartományban szorzás, az idő tartományban konvolúció.

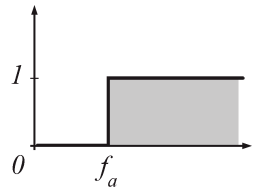
A négyszög függvényről tudjuk, hogy Fourier-transzformáltja a *sinc* függvény. A 16.26. ábrán a kétdimenziós *sinc*-t láthatjuk, mint az ideális felülvágás kernel függvényét.

A felülvágó algoritmus végrehajtásához a kernelbe betöltjük a kétdimenziós *sinc* függvény megfelelően mintavételezett értékeit, majd a fent leírt módon végrehajtjuk a digitális konvolúciót.

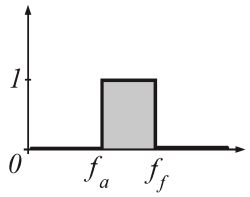
Alulvágó szűrő

Az alulvágó szűrőt akkor alkalmazzuk, ha az alacsony frekvenciás jeleket el akarjuk tüntetni a képről. Átviteli függvénye a 16.27. ábrán látható.

Ha alaposabban szemügyre vesszük az alulvágás és a felülvágás átviteli függvényét, akkor észrevehetjük, hogy $S_{fa}(f) = 1 - S_{ff}(f)$, vagyis az f_a alsó határfrekvenciájú alulvágó átviteli függvényének és az f_f felső határfrekvenciájú felülvágó szűrő átviteli függvényének összege azonosan 1 (feltéve, hogy $f_a = f_f$). Ebből következően, ha az f_f felső határfrek-



16.27. ábra. Az ideális alulvágás átviteli függvénye.



16.28. ábra. Az ideális sávszűrő átviteli függvénye.

venciájú felülvágóhoz tartozó *sinc* függvényt levonjuk 1-ből, akkor az ugyanazon (alsó) határfrekvenciájú alulvágó szűrő szűrőegyütthatóit kapjuk meg.

Jelölje F a direkt, F^{-1} az inverz Fourier-transzformációt, melynek azonosságai alapján felírható a következő:

$$F^{-1}\{S_{fa}(f)\} = F\{1\} - F^{-1}\{S_{fa}(f)\} = 1 - s_{fa}(t)\ ,$$

ahol $S_{fa}(f)$ az f_a alsó határfrekvenciájú alulvágó szűrő átviteli függvénye, $s_{fa}(t)$ pedig az átviteli függvény inverz Fourier-transzformáltja.

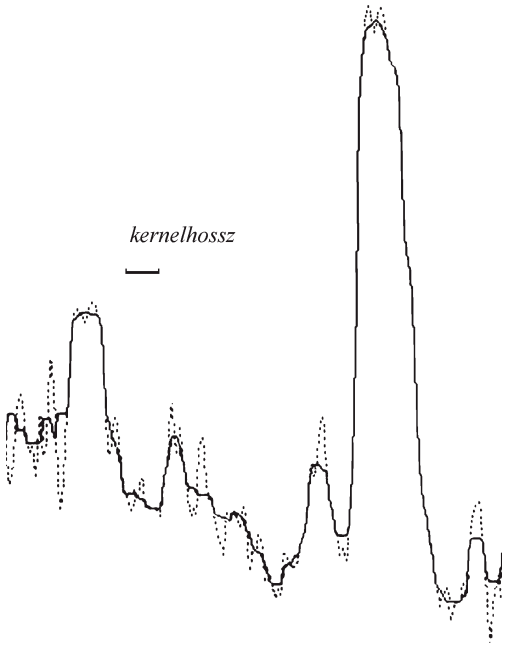
Sávszűrő

A sávszűrő egy olyan szűrő, amely egy adott alsó határfrekvencia alatt és egy felső határfrekvencia felett nem enged át. Átviteli függvénye a 16.28. ábrán látható. Kerneljét úgy kaphatjuk meg, hogy kiszámítjuk az f_f felső határfrekvenciához, majd az f_a alsó határfrekvenciához tartozó *sinc* függvényeket, és ezeket kivonjuk egymásból.

Az f_a és f_f frekvenciák egymáshoz közelítésével nagyon keskeny sávot átengedő szűrőt hozhatunk létre. Ha ezt alkalmazzuk, majd az eredményt az eredeti képadatokból kivonjuk, úgynevezett ***lyukszűrést*** végzünk.

Élmegőrző szűrők

Az élmegőrzők olyan speciális szűrők, amelyek átviteli függvényei nem adhatók meg. Működésük meglehetősen egyszerű algoritmus szerint történik. A kernelt mozgassuk végig a képen, és töltsük fel az éppen alatta lévő pixelek értékeit. Rendezzük nagyság szerint sorba a kernel elemeit, és a rendezett adatsor valamelyik elemét rendeljük hozzá a kernel szimmetria középpontja alatt lévő pixelhez, amelynek ez lesz a szűrt értéke. Ezek a szűrők az úgynevezett ***rangszűrők.*** Az egyik legismertebb rangszűrő a ***medián szűrő,*** amely a sorba rendezett értékek sorban középső elemének értékét rendeli a pixel szűrt értékének.



16.29. ábra. Egy egydimenziós időfüggvény (szaggatott vonal) és medián-szűrt változata (folytonos vonal).

A 16.29. ábrán egy idősorra alkalmaztuk a medián szűrőt.

Jól megfigyelhető, hogy a fel- vagy lefutó éleken a szűrő nem változtatja meg az eredeti adatokat, hiszen azok az éleken már eleve nagyság szerint rendezettek. Nem éleken azonban erőteljesen simít. A simítás mértéke a kernel hosszától függ, annál jobban simít, minél hosszabb. A 16.30. ábrán egy LANDSAT képrészlet és medián-szűrt változata látható.

Éldetektorok

Az éldetektálás különösen fontos szerepet játszik az alakfelismerésben, a raszteres térképek vektorossá alakításában. Az élek a képnek azon helyei, ahol az intenzitás megváltozása a legnagyobb. Először is döntsük el, hogy mennyire kifinomult élek kimutatását szeretnénk. A legtöbbször érdemes simító vagy medián szűrésnek alávetni a képet, hogy ne mutassunk ki minden apró, jelentéktelen élt. A simítás egyik ismert és egyszerű módja a kép és egy

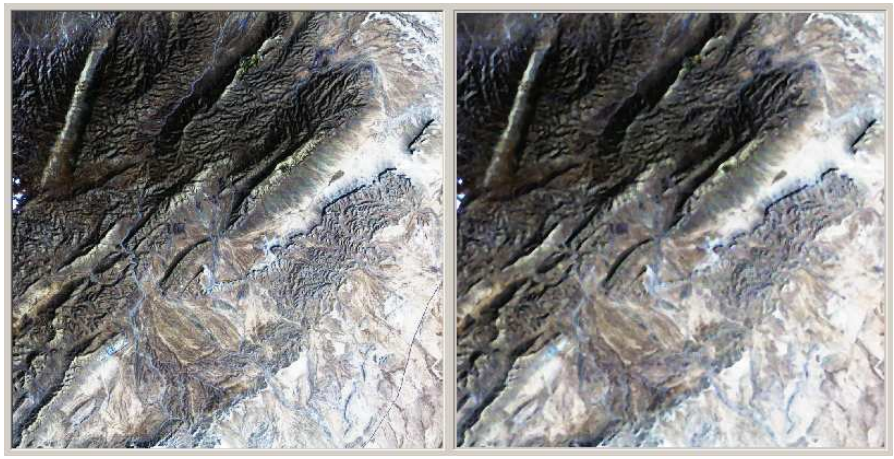
$$g_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-x^2}{2\pi\sigma^2}}$$

Gauss-függvény konvolúciója.

Legyen h az f és g függvények konvolúciója. Kimutatható, hogy

$$h = (f * g)' = f * g' ,$$

vagyis egy jel (jelöljük f-fel) Gauss-függvénnyel (g) való konvolúciójának a deriváltja



16.30. ábra. A bal oldali kép az eredeti, míg a jobb oldali egy kilenc pontos (9×9 pixeles) medián-szűrt kép. Az ábra színes változata az 814. oldalon látható.

egyenlő a jel és a Gauss-függvény deriváltjának a konvolúciójával. Ezek alapján az éldetektálás algoritmusa a következő.

ÉLDETEKTÁLÁS(f, g')

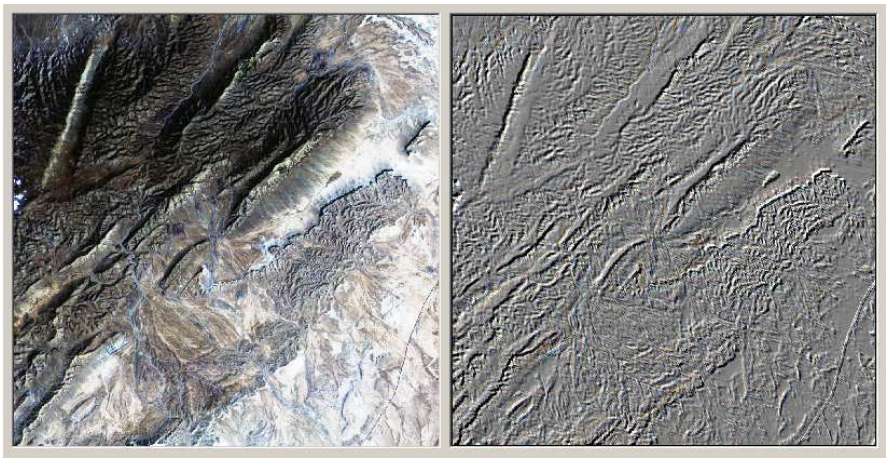
1 konvolváljuk f-et g'-vel
2 számítsuk ki h abszolút értékét
3 definiáljuk éleknek mindazokat a helyeket, ahol a h abszolút értéke meghalad
egy előre meghatározott küszöb értéket

Nem használtuk ki sehol a gondolatmenet során, hogy egy vagy kétdimenziós esettel van-e dolgunk, így az éldetektálás fenti módja képek esetére is működőképes. Ez az eljárás a ***Canny-féle éldetektor.*** Sokféle éldetektáló kernel létezik még. Egy másik, ismert éldetektáló szűrő az ***emboss-szűrő.*** Kernelje (3×3-as méretű esetben) a következő:

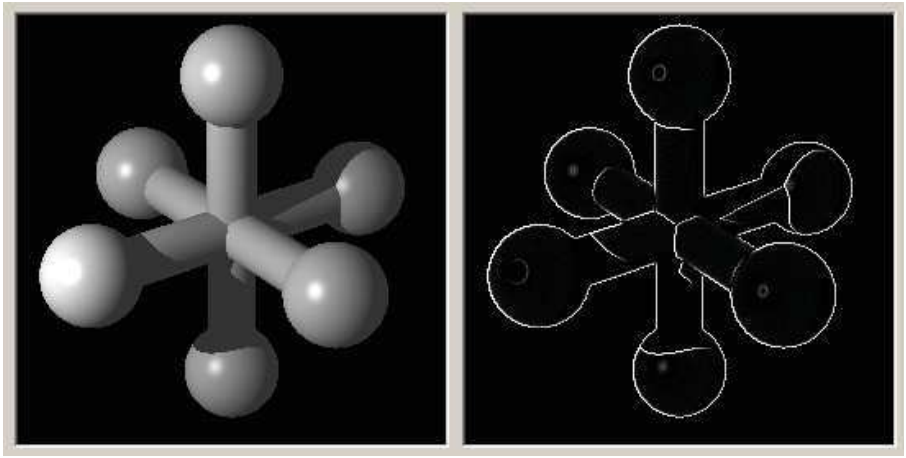
$$\begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{matrix} .$$

Feltűnő tulajdonsága, hogy az ÉNY-DK irány kitüntetett. Hatására a középponti kernel mező alatti képpixel lenullázódik, ellenben a kitüntetett irányba eső két pixel közül az egyik pixel saját értékével, a másik pixel pedig értékének ellentettjével esik latba. Ezzel a művelettel az ÉNY-DK irányba eső változások kiemelődnek, minden más elnyomódik. Ha másik irányt tüntetünk ki, akkor az abba az irányba eső változások kapnak hangsúlyt. Az emboss szűrés eredményét mutatja 16.31. ábra.

Az éldetektálók egy másik, gyakran alkalmazott eljárása a ***Laplace-szűrés.*** Hatása hasonlít az emboss szűrőhöz, de nem tüntet ki irányokat. Mindenféle irányú éleket kiemel,



16.31. ábra. A bal oldali kép az eredeti, a jobb oldali az emboss-szűrt változat. Az ábra színes változata az 814. oldalon látható.

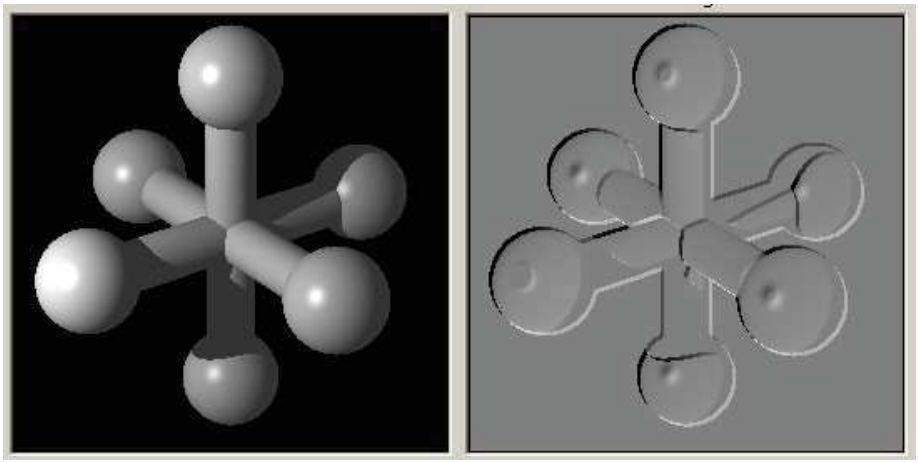


16.32. ábra. A bal oldali az eredeti kép, a jobb oldali a Laplace-szűrt változata. Az ábra színes változata az 815. oldalon látható.

minden mást elnyom. Kernelje is érdekes.

$$\begin{matrix} -1 & -2 & -1 \\ -2 & 12 & -2 \\ -1 & -2 & -1 \end{matrix}.$$

Hatását egy szintetikus képen érzékeltetjük a 19.18. ábrán.



16.33. ábra. A bal oldali az eredeti kép, a jobb oldali az emboss-szűrt változata. Az ábra színes változata az 815. oldalon látható.

Érdekes összehasonlítani a kétféle éldetektor eredményét a szintetikus képre. A 19.19. ábrán a szintetikus képet és az emboss-szűrt változatát láthatjuk.

Gyakorlatok

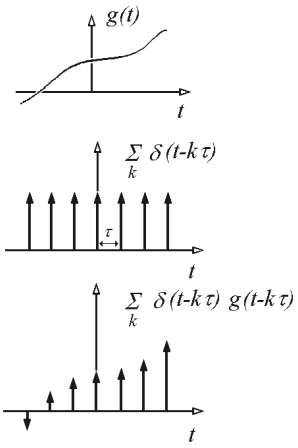
16.3-1. Bizonyítsuk be, hogy igaz a következő kifejezés:

$$(f * g)' = f * g' \ .$$

16.3-2. Készítsünk három olyan 3×3-as méretű kernelt, ahol az első az ÉK–DNy, a második az É–D, a harmadik a K–Ny irányú éleket emeli ki.

16.4. Mintavételezés

A szaktudományokban alapvető jelentőségű az adatnyerés folyamata, amely a technika mai színvonalán digitális adatnyerést vagy mintavételezést jelent. Mintavételezéskor gyakran analóg jelekből állítunk elő digitális adatokat, máskor eleve diszkrét, esetleg nem szabályosan elhelyezkedő mintavételi pontokban történik az adatnyerés. A mintavételezés folyamatának megértése nagy jelentőségű, mivel az adatokból levonható következtetések függhetnek a mintavételezés módjától. Képzeljük el a mintavételezést, mint egy kétállású kapcsolóval szabályozott mérő berendezést, amely bekapcsolt állapotban rögzíti a mérendő paramétert, kikapcsolt állapotban pedig mit sem tud a környezetéről. Más szavakkal azt is mondhatjuk, hogy két mintavételi (idő)pont között bármi történik is, arról nem fogunk tudomást szerezni, mivel mérő berendezésünk ilyenkor kikapcsolt állapotban van. Belátható, hogy ez a fajta hiányosság csökkenthető, ha sűrítjük a mintavétel mintavételezési gyakoriságát, vagyis csökkentjük az érzékelés nélkül töltött üzemmód részarányát a működő állapothoz



16.34. ábra. A mintavételezés eredménye egy olyan impulzus sorozat, melynek tagjai az eredeti függvénynek a mintavételi helyen felvett értékei.

képest. Nevezzük ***mintavételi távolságnak*** a két érzékelés közötti intervallumot, amely, ha idősorokról van szó, akkor idő dimenziójú, de lehet távolság dimenziójú is, ha két mintavételi pont között térbeli távolságról van szó. A következőkben vázlatosan áttekintjük a mintavételezés legfontosabb törvényszerűségeit, egyenletes mintavételezést feltételező esetekben. Az egyszerűség kedvéért idősorokkal, vagyis egydimenziós problémákkal foglalkozunk, amik teljes mértékben általánosíthatók több dimenziós esetekre is, mint amilyen a digitális kép, vagy a háromdimenziós terepmodell.

16.4.1. Mintavételi tétel

Legyen τ az úgynevezett mintavételi távolság. Ábrázoljuk a $g(t)$ időfüggvényt és a mintavételezés eszközét, a Dirac-impulzusok sorozatát, majd a mintavételezés eredményét, a digitalizált időfüggvényt (16.34. ábra). A ***mintavételezés*** tehát nem más, mint a $g(t)$ időfüggvény és a Dirac-impulzus sorozat szorzata:

$$g(t)\sum_{k=-\infty}^{\infty}\delta(t-k\tau)=\sum_{k=-\infty}^{\infty}g(k\tau)\delta(t-k\tau)\ .$$

Vizsgáljuk meg az analóg és a mintavételezett függvény spektrumát. Jelöljük $G(f)$-fel az eredeti, és $G_d(f)$-fel a digitalizált függvény spektrumát. A Dirac-δ Fourier-transzformáltja és a konvolúció tételek felhasználásával felírható a mintavételezett függvény spektruma:

$$G_d(f)=G(f)*\frac{1}{\tau}\sum_{k=-\infty}^{\infty}\delta(f-\frac{k}{\tau})\ ,$$

vagyis

$$G_d(f)=\frac{1}{\tau}\sum_{k=-\infty}^{\infty}G(f-\frac{k}{\tau})\ .$$

Értelmezzük a kapott eredményt. A kifejezés jobb oldala szerint a digitalizált jel spektruma periodikus, ami azért érdekes, mert a periodikus függvények spektruma nem periodikus függvény, vagyis a mintavételezés az eredetileg nem periodikus spektrumot periodikussá teszi. A spektrumnak a $-1/2\tau$ és $1/2\tau$ közé eső részét a ***spektrum fő részének,*** az $f_N = 1/2\tau$ értéket ***Nyquist-frekvenciának*** nevezzük. A spektrum többi részén a fő rész f_N periódussal ismétlődik. A fenti formulákból világosan kiolvasható, hogy az analóg és a digitalizált függvény spektruma jelentősen eltérhet egymástól, ha az analóg függvény bármilyen frekvenciájú jeleket is tartalmazhat. Ha azonban létezik egy olyan felső határfrekvencia (f_f), amelynél nagyobb frekvenciájú jel nem fordulhat elő (vagyis létezik a spektrumra felső határfrekvencia), akkor belátható, hogy a felső határfrekvencia és a τ mintavételi távolsággal még átvihető legnagyobb frekvencia között igaz a következő összefüggés:

$$f_f \leq f_N \ ,$$

vagyis

$$\tau \leq \frac{1}{2f_f} \ . \tag{16.1}$$

Ez az összefüggés a ***mintavételi tétel.*** Jelentése, hogy mintavételezéskor a még átvihető legnagyobb frekvenciához, f_f-hez úgy kell megválasztanunk a τ mintavételi távolságot, hogy teljesüljön a 16.1. egyenlőtlenség. Ha túl kicsire választjuk a mintavételi távolságot, akkor feleslegesen sűrűn mintavételezett adatrendszert kapunk, ha viszont túl nagyra, mellyel nem áll fenn a 16.1. egyenlőtlenség, akkor nem fog teljesülni az f_f felső határfrekvencia szerinti jelátvitel. Túlzottan ritka mintavételezéssel tehát felülvágást, és a fő rész ismétlődése miatt kisebb-nagyobb torzulást okozunk az adatrendszer spektrumán.

16.4.2. A mintavételi tétel néhány következménye

Láthattuk, hogy bizonyos esetekben a digitalizálás (mintavételezés) adatvesztéssel járhat. Tekintsük át, hogy mikor nem vesztünk adatot. A megfelelően mintavételezett digitális adatrendszerből az eredeti analóg jel pontosan visszaállítható. (Akkor mondjuk megfelelően mintavételezettnek az adatrendszert, ha teljesült a mintavételi tétel 16.1. egyenlőtlensége.) Az előző részben láthattuk, hogy a mintavételezés periodikussá teszi a spektrumot. Ha pontosan vissza kívánjuk állítani az eredeti analóg jelet a mintavételezett jel $G_d(f)$ spektrumából, akkor el kell tüntetnünk a spektrum fő részein kívüli részeit, vagyis meg kell szoroznunk $G_d(f)$-t egy olyan négyszög függvénnyel, amelynek magassága τ, szélessége $1/\tau$. Így egyszerűen levágjuk a spektrum periodikus részeit, vagyis kiküszöböljük a mintavételezéssel belevitt periodicitást, azaz visszakapjuk az analóg jel spektrumát. Az ismertetett gondolatmenet akkor adja vissza a jelet az idő tartományban, ha a visszakapott spektrumot inverz Fourier-transzformáljuk. Ismerve a konvolúció tulajdonságait és a négyszög függvény (inverz) Fourier-transzformáltját, belátható, hogy a τ magasságú és $1/\tau$ szélességű négyszög függvénnyel való szorzás a frekvenciatartományban a *sinc* függvénnyel való konvolúciót jelent az idő tartományban. Ez alapján a jel visszaállítása az idő tartományban a következő módon lehetséges:

$$\left\{ \sum_{k=-\infty}^{\infty} g(t)\delta(t-k\tau) \right\} sinc(t/\tau) = \sum_{k=-\infty}^{\infty} g(k\tau) sinc(t/\tau - k) \ ,$$

vagyis a visszaállított értékek a minták és a *sinc* függvény megfelelő argumentummal vett értékeinek szorzata. Ez egyrészt azt jelenti, hogy az így visszakapott értékek pontosan azonosak az egyes minták értékeivel, valamint az analóg függvény tetszőleges helyén felvett értékek előállíthatók a mintáknak és a megfelelő argumentummal megadott *sinc* függvényértékek szorzatának összegzésével. Természetesen csak akkor igazak ezek a megállapítások, ha teljesült a mintavételi tétel 16.1. feltétele. Túl nagyra választott mintavételi távolság esetén az analóg jel spektruma nem állítható vissza pontosan, mivel a túl ritka mintavétel felülvágást hajtott végre a spektrumban.

16.4.3. Két tipikus mintavételi probléma

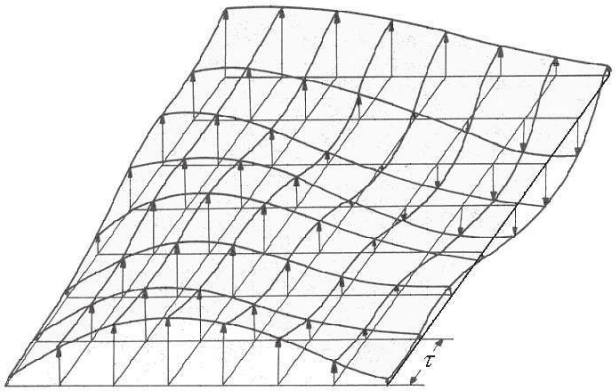
A következőkben bemutatunk két, a térinformatikában tipikus mintavételezési feladatot, melyek megoldása során jól használhatók a mintavételről elmondottak.

Digitális fénykép
A raszteres rendszerek tipikus adatnyerési módja a műholdakról vagy repülőgépekről történő fényképezés. A bemutatott időfüggvényekhez képes ezekben az esetekben a mintavételezés kétdimenziós, mivel a mintavételezést egy kétdimenziós Dirac-δ sorozattal végezzük, amelynek mindkét irányban τ a mintavételi távolsága, a minták értéke pedig az adott Dirac-impulzus felett lévő intenzitás és színérték. Mindez a gyakorlatban érzékelő kamerákkal valósítható meg, amelyeknek τ rácsállandójú fényérzékelői vannak. Ezen érzékelők mérete meghatározza az elérhető legnagyobb térbeli felbontó képességet, vagyis a fényképeken nem mutatható ki 2τ -nál kisebb részlet.

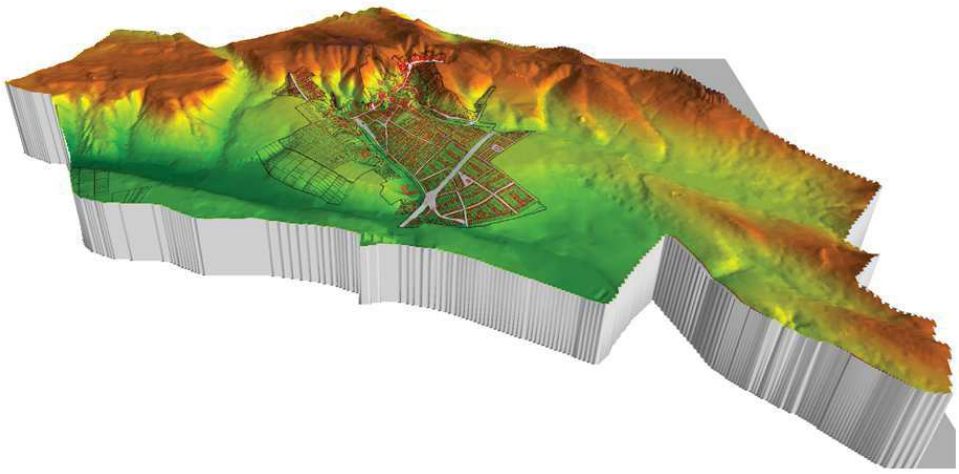
Domborzati modellek
A domborzati modellek megalkotásának egyik fontos állomása, hogy ismerjük szabályos rácspontokban a felszíni magasság értékeket. Ezen adatnyerési technikák ismertetését mellőzzük, ugyanakkor fontosnak tartjuk kiemelni a mintavételi tétel ide vonatkozó következményeit. A magasság értékek egy a felszínt leíró függvénynek egy τ rácsállandójú Dirac-impulzus sorozattal való szorzásával kaphatók meg (19.20. ábra).

A 16.36. ábrán egy valóságos domborzati modellt láthatunk, amelynek felbontóképessége körülbelül 5 méter, amivel a mintavételi tétel miatt 10 méternél kisebb horizontális kiterjedésű felszíni egyenetlenség nem mutatható ki. Ezért csak építészeti, várostervezési célú felhasználást enged meg. Ha például a felhőszakadások alkalmával lezúduló csapadékvíz lefolyását kívánnánk modellezni, akkor pontosabb (nagyobb felbontású) domborzati modellt kellene megalkotni.

Általánosságban kimondható, hogy nincsenek univerzális, minden célra alkalmas domborzati modellek. A pontossággal szembeni elvárások eltérő mivolta egyben különböző mintavételezési kritériumokat is jelent. Ezért fordulhat elő, hogy ami a telekommunikáció számára megfelelően pontos domborzati modell, az az árvízvédelem számára teljesen használhatatlan.



16.35. ábra. A τ rácsállandójú Dirac-δ sorozattal mintavételezett felszín.



16.36. ábra. Egy település domborzati modelljének perspektivikus megjelenítése északi irányból. Az ábra színes változata az 816. oldalon látható.

Gyakorlatok

16.4-1. Hogyan célszerű megválasztani a τ mintavételi távolságot?

16.4-2. Elemezzük, hogy mit történik az adatrendszerrel és a spektrummal, ha τ_1 mintavételi távolságról áttérünk a τ_2 mintavételi távolságra, abban az esetben, ha

- $\tau_1 < \tau_2$ (ritkítás);
- $\tau_1 > \tau_2$ (sűrítés).

Feladatok

16-1. A négy-fa további műveletei

Az előzőekben felvázoltuk a pont-tartomány négy-fa egy lekérdező műveletét. Készítsünk hatékony algoritmust objektum törléséhez a fából, valamint a következő lekérdezésekhez:

a. határozzuk meg egy P pont legközelebbi szomszédját, azaz azon objektumot, melynek távolsága P-től a legkisebb (NN-query), valamint

b. határozzuk meg egy P pont legközelebbi k darab szomszédját, azaz a P-hez legközelebb fekvő k darab objektumot (k-NN query).

16-2. Konvolúció RGB színkódok esetében

A színes képek pontjainak RGB kódjait a KONVOLÚCIÓ algoritmus összeadást és szorzást használó művelete nem megfelelően kezeli. Készítsük fel az algoritmust a színes képek kezelésére.

16-3. Konvolúció kiterjesztése a teljes képre

A KONVOLÚCIÓ algoritmus a bemeneti kép azon pontjait, melyekre nem tudja illeszteni a kernel középpontját, eldobja. Készítsük el az algoritmus olyan változatát, amely ezen határolópontokhoz is rendel képpontokat a kimeneti képben. Gyakorlatban gyakran előforduló megoldásnak számít a kernel által lefedett területek tükrözése vagy másolása a kép szélén túli területre. Gondoljuk át a lehetséges megoldások előnyeit és hátrányait.

16-4. Zajszűrés

Tegyük fel, hogy van három képünk pontosan ugyanarról a területről. Mindhárom véletlen zajjal fertőzött.

a. Tegyünk több javaslatot a zaj csökkentésének módjára.

b. Mi történik a képpel és a jel/zaj aránnyal, ha a három képet egyetlen képben összegezzük?

16-5. Csonkítás elemzése

A sinc függvény, mint láthattuk, fontos szerepet játszik mind a szűrések, mind pedig a mintavételezés területén. Ideális esetben a frekvenciatartományban megkívánt – végtelenhez

közeli – levágási meredekség miatt a sinc függvénynek nagyon hosszúnak kellene lennie, mivel a sinc csak lassan tart nullához. Ezért a magfüggvény viszonylag hosszú lesz, ami viszont hátrányos az eljárás futási idejének szempontjából. Ezért a sinc függvény helyett annak csonkított változatát használják a gyakorlatban a szűrők, és az újra-mintavételezés végrehajtásakor. A csonkító függvények általában valamiféle „harang-görbék”, amivel módosítják a sinc függvényt. A módosított súlyfüggvény az eredeti súlyfüggvény és a csonkító függvény szorzata.

Bizonyítsuk be a konvolúció azonosságai segítségével, hogy a legrosszabb eredményt adja az a csonkítás, amikor egyszerűen elhagyjuk a magfüggvény széleit, vagyis ha a csonkító függvény a négyszög függvény.

Megjegyzések a fejezethez

Laurini [279], valamint Maguire és társai [304] könyvei részletesen tárgyalják a különböző térinformatikai adatmodelleket. A centroid és a befoglaló négyszög fogalmát is bevezetik, aminek jelentőségét [386] több helyen ki is fejti. Hatvany Csaba [195] és Stephens [438] grafikus programozással foglalkozó műveikben részletesen foglalkoznak a grafikus adatok és a megjelenítés kérdéseivel. Rigaux és társai [386] elméleti alapművükben részletesen foglalkoznak a térbeli indexelés problematikájával és összevetik a [90] tankönyvben is részletesen tárgyalt B-fa működésével. [294] és [386] összehasonlítják a grid index és a rekurzív térfelosztási algoritmusok hatékonyságát. A négy-fa és más kapcsolódó adatszerkezetek részletes elemzésével foglalkozik Samet [410] cikke. [279], [294] és [386] részletesen tárgyalják a négy-fa algoritmust, valamint ugyanitt olvashatunk a nyolc-fáról is. Szintén megtalálható ezen művekben az R-fa, melyről nem esett szó a fejezetben, de a gyakorlatban sokszor használt adatszerkezet. Az R-fa a befoglaló négyszög fogalmára építő, a B-fához hasonló összetettebb keresőfa, melyet tetszőleges dimenziószámú térben alkalmazhatunk. Hasznos tulajdonságai, hogy csúcsainak méretét az aktuális blokkmérethez igazíthatjuk, kiegyensúlyozott, valamint megfelelően támogatja a gyakori lekérdezés-típusokat. A négy-fa és a nyolc-fa grafikai alkalmazásával foglalkozik a 15.4. alfejezet.

[195] és [438] a gyakorlati programozás oldaláról mutatják be az RGB színmodellt, és több, a digitális szűrési technikában ismert szűrő algoritmust, mint például az alul- és felülvágó szűrők, sávszűrők, élmegőrzők, élkiemelők, és sok más érdekes szűrőt. Richards könyve [385] elméleti alapmű, amely széleskörű áttekintést nyújt az űrfotók, a távérzékelés problémaköréről az észleléstől a feldolgozáson át az értelmezésig. Duncan művéből [115] jó áttekintést kapunk a komplex függvénytanról, amely nélkül nem érthető a Fourier-analízis. [93] érthető formában vezeti be az Olvasót a disztribúcióelmélet fogalmaiba. Tárgyalja a Dirac-δ-t, a Heaviside-féle lépcsőfüggvényt és sok más érdekességet, amely nélkül a jelfeldolgozás és mintavételezés nehezen lenne megérthető. [90], [321] és [322] részletesen ismertetik a Fourier-transzformációt, sok példával, alapos elméleti áttekintéssel. Meskó Attila tankönyveiben [321, 322] részletesen olvashatunk a különböző szűrési technikák, valamint a mintavételezés elméleti alapjairól, a mintavételi törvényről, ezek gyakorlati alkalmazásáról.

A térinformatikával foglalkozik magyar nyelven Detrekői Ákos és Szabó György [109, 110], Kertész Ádám [253], Márkus Béla [338], valamint Márton Mátyás és társai [340] könyve.

17. Tudományos számítások

A tudományos számítások cím egy kísérlet a nemzetközi szakirodalomban „Scientific Computing”, „Scientific Engineering”, „Computational Science and Engineering” stb. névvel hivatkozott diszciplína magyar megnevezésére. Ezt a területet a matematika, a szoftverek és az alkalmazási területek közti interdiszciplináris területként is szokás jellemezni. Célja a számítógépek és matematikai algoritmusok – a hardvert is tekintetbe vevő – hatékony felhasználása tudományos és mérnöki problémák megoldására. A témakör irodalmának tanulmányozása alapján – némi egyszerűsítéssel – azt mondhatjuk, hogy témánk valójában a numerikus matematika, a szoftverfejlesztés, az eredmények grafikus megjelenítése (számítógépes grafika) és a szakterületi alkalmazási ismeretek együttese. A terjedelmi korlátokra és az ésszerűségre tekintettel csak a kérdéskör néhány alapvetően fontos elemét tárgyaljuk, amelyek a *lebegőpontos aritmetika* (17.1. alfejezet) és a *hibaelemzés* (17.2. alfejezet) alapjaira, a *lineáris algebra* (17.3. alfejezet) alapvető módszereire és a *matematikai szoftverekre, szoftverkönyvtárakra* (17.4. alfejezet) vonatkoznak.

17.1. Lebegőpontos aritmetika és hibaelemzés

17.1.1. Hibaszámítási alapismeretek

Legyen x a kiszámítandó pontos érték, a az x közelítése ($a \approx x$). A közelítés ***hibáját*** a $\Delta a = x - a$ képlettel definiáljuk (néha fordított előjellel). A δa-t az a közelítő érték ***abszolút hibakorlátjának*** (vagy röviden csak ***hibakorlátjának***) nevezzük, ha fennáll $|x - a| = |\Delta a| \leq \delta a$. Például a $\sqrt{2} \approx 1.41$ közelítés hibája legfeljebb 0.01, más szóval egy hibakorlátja 0.01. Az x és az a mennyiségek (és természetesen a Δa, δa is) vektorok vagy mátrixok is lehetnek. Ilyenkor az abszolútérték és a relációs jelek komponensenként (mátrixelemenként) értendők. A hiba nagyságának mérésére normát is használunk. Ez esetben a $\delta a \in \mathbb{R}$ akkor hibakorlát, ha fennáll $\|\Delta a\| \leq \delta a$.

Az abszolút hibakorlát sok esetben semmitmondó. Például egy 0.05 abszolút hibakorlátú közelítés lehet egészen kiváló is, de egy 0.001 nagyságrendű elméleti mennyiség becslésénél nem sokat ér. A becslés jóságát sokszor a ***relatív,*** azaz az egységre eső ***hibakorláttal*** jellemezzük, ami $\delta a / |x|$ (vektorok, mátrixok esetén $\delta a / \|x\|$). Mivel az x pontos érték általában nem ismeretes, ezért a $\delta a / |a|$, (illetve a $\delta a / \|a\|$) közelítést használjuk. Az így elkövetett hiba elhanyagolható, ha x és a abszolút értéke (normája) lényegesen nagyobb a másodrendű $(\delta a)^2$ mennyiségnél. A relatív hibát sokszor százalékokban fejezzük ki.

Minthogy a hibát rendszerint nem ismerjük, ezért hibán gyakran a hibakorlátot értjük.

A *klasszikus hibaszámítás* alapmodelljében azt vizsgáljuk, hogy közelítő, de ismert hi-

bakorlátú bemeneti adatokkal pontosan végrehajtott számítások során mekkora a végeredmény hibakorlátja. Legyen x és y két pontos érték, a az x, b pedig az y közelítése. Tegyük fel, hogy az a és b közelítések abszolút hibakorlátai δa, ill. δb. A klasszikus hibaszámítás eszközeivel a négy alapműveletre a következő hibakorlátokat kapjuk:

$$\delta(a+b) = \delta a + \delta b, \qquad \frac{\delta(a+b)}{|a+b|} = \max\left\{\frac{\delta a}{|a|}, \frac{\delta b}{|b|}\right\} \quad (ab > 0)\ ,$$

$$\delta(a-b) = \delta a + \delta b, \qquad \frac{\delta(a-b)}{|a-b|} = \frac{\delta a + \delta b}{|a-b|} \quad (ab > 0)\ ,$$

$$\delta(ab) \approx |a|\,\delta b + |b|\,\delta a, \qquad \frac{\delta(ab)}{|ab|} \approx \frac{\delta a}{|a|} + \frac{\delta b}{|b|} \quad (ab \neq 0)\ ,$$

$$\delta(a/b) \approx \frac{|a|\,\delta b + |b|\,\delta a}{|b|^2}, \qquad \frac{\delta(a/b)}{|a/b|} \approx \frac{\delta a}{|a|} + \frac{\delta b}{|b|} \quad (ab \neq 0)\,.$$

A képletekből látható, hogy nullához közeli számmal való osztás során az abszolút hiba, míg nullához közeli végeredményű kivonás esetén a relatív hiba akármilyen naggyá válhat. Ezeket az eseteket kerülni kell. Különösen a kivonás lehet nagyon alattomos.

17.1. példa. Számítsuk ki a $\sqrt{1996} - \sqrt{1995}$ mennyiséget, ha ismertek a $\sqrt{1996} \approx 44.67$ és a $\sqrt{1995} \approx 44.66$ közelítő értékek, amelyek közös abszolút hibakorlátja 0.01, a közös relatív hibakorlát pedig 0.022%. A kivonás elvégzésével kapjuk, hogy $\sqrt{1996} - \sqrt{1995} \approx 0.01$, amelynek relatív hibakorlátja az általános képletből

$$\frac{0.01 + 0.01}{0.01} = 2\ ,$$

azaz 200%. Most lehetőségünk van a tényleges relatív hiba kiszámolására is, ami „csak” 10.66%. Ez a valóságos hiba is jelentős mértékű, a kiinduló adatok hibájához képest kb. 4.84×10^2-szoros. A különbség képzését elkerülhetjük a

$$\sqrt{1996} - \sqrt{1995} = \frac{1996 - 1995}{\sqrt{1996} + \sqrt{1995}} = \frac{1}{\sqrt{1996} + \sqrt{1995}} \approx \frac{1}{89.33} \approx 0.01119$$

átalakítással. A számláló pontos érték. A nevező abszolút hibája 0.02, a hányados relatív hibája pedig $0.02/89.33 \approx 0.00022 = 0.022\%$. Ez összhangban van a kiinduló adatok relatív hibáival és lényegesen kisebb, mint amit a közvetlen kivonásnál kaptunk.

Kétszer folytonosan differenciálható egy-, illetve többváltozós függvények közelítő (elsőrendű) hibáit az első fokú *Taylor-polinomjuk* segítségével kapjuk:

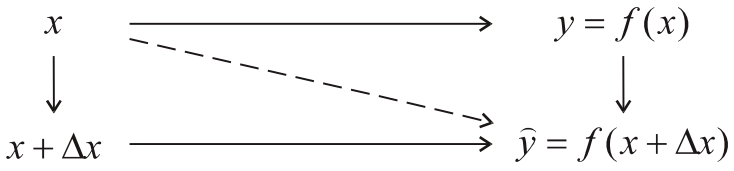
$$\delta(f(a)) \approx \left|f'(a)\right| \delta a, \quad f : \mathbb{R} \to \mathbb{R}\ ,$$

$$\delta(f(a)) \approx \sum_{i=1}^{n} \left|\frac{\partial f(a)}{\partial x_i}\right| \delta a_i, \quad f : \mathbb{R}^n \to \mathbb{R}\ .$$

Függvények adott a pontbeli numerikus stabilitására jellemző az úgynevezett ***kondíciószám.*** Ez a közelítő függvényérték és a közelítő bemeneti mennyiség relatív hibájának hányadosa ($F : \mathbb{R}^n \to \mathbb{R}^m$ esetén $F'(a)$-n az F függvény $a \in \mathbb{R}^n$ helyen számított *Jacobi-mátrixát* értjük):

$$c(f,a) = \frac{|f'(a)|\,|a|}{|f(a)|}, \quad f : \mathbb{R} \to \mathbb{R}\ ,$$

$$c(F,a) = \frac{\|a\|\,\|F'(a)\|}{\|F(a)\|}, \quad F : \mathbb{R}^n \to \mathbb{R}^m\ .$$



17.1. ábra. Direkt hiba és inverz hiba.

A kondíciószám tehát a relatív hiba nagyítási számának tekinthető: ennyiszeresre nő a bemeneti relatív hiba a függvényérték kiszámítása során. Ennek megfelelően a függvényünk ***numerikusan stabil,*** más néven ***jól kondicionált*** az a helyen, ha $c(f,a)$ kicsi, egyébként ***numerikusan instabil*** (rosszul kondicionált). A kondicionáltság helyfüggő. Egy függvény lehet valamely a helyen jól, míg ugyanaz a függvény egy b helyen rosszul kondicionált. Természetesen a $c(f,a)$-ra vonatkozó „kicsi" jelző relatív. Mint később látni fogjuk, ez a relatív jelző adott feladat esetén a rendelkezésre álló számítógép aritmetikájától és a közelítés megkövetelt pontosságától függ.

Mátrixok kondíciószámát is mint függvény kondíciószám felső korlátját vezethetjük be. Definiáljuk az $F : \mathbb{R}^n \to \mathbb{R}^n$ leképezést az $Ay = x$ egyenletrendszer megoldásával, azaz legyen $F(x) = A^{-1}x$ ($A \in \mathbb{R}^{n\times n}$, $\det(A) \neq 0$). Ekkor $F' \equiv A^{-1}$ és

$$c(F,a) = \frac{\|a\|\left\|A^{-1}\right\|}{\left\|A^{-1}a\right\|} = \frac{\|Ay\|\left\|A^{-1}\right\|}{\|y\|} \leq \|A\|\left\|A^{-1}\right\| \quad (Ay = a) \ .$$

A jobboldali felső korlátot az A ***mátrix kondíciószámának*** hívjuk. Ez a korlát pontos, mert létezik olyan $a \in \mathbb{R}^n$, hogy $c(F,a) = \|A\|\left\|A^{-1}\right\|$.

17.1.2. Direkt és inverz hibák

Vizsgáljuk egy $f(x)$ függvényérték kiszámítását. Ha az $\hat{y}$ közelítést számoljuk a pontos $y = f(x)$ érték helyett, akkor a ***direkt hiba*** $\Delta y = \hat{y} - y$. Ha egy $x + \Delta x$ értékre fennáll, hogy $\hat{y} = f(x + \Delta x)$, azaz $\hat{y}$ a perturbált (megváltoztatott) $\hat{x} = x + \Delta x$ értékhez tartozó pontos függvényérték, akkor a Δx értéket ***inverz hibának*** nevezzük. A kétfajta hibát mutatja a 17.1. ábra.

A folytonos vonal az elméleti értéket, a szaggatott pedig a számított értéket jelöli. Az inverz hiba elemzését és becslését ***inverz hibaanalízisnek*** nevezzük. Ha több inverz hiba is létezik, akkor a legkisebb inverz hiba meghatározása az érdekes.

Az $y = f(x)$ értéket számító algoritmust ***inverz stabilnak*** nevezzük, ha bármely x értékre olyan $\hat{y}$ számított értéket ad, amelyre a Δx inverz hiba kicsi. A „kicsi" jelző környezetfüggő.

A direkt és az inverz hiba kapcsolatát a hibaszámítási ökölszabálynak is nevezett

$$\frac{\delta\hat{y}}{|y|} \leq c\,(f,x)\,\frac{\delta\hat{x}}{|x|} \tag{17.1}$$

közelítő egyenlőtlenség fejezi ki. Szavakban megfogalmazva:

relatív direkt hiba ≤ kondíciószám × relatív inverz hiba.

Az egyenlőtlenség azt mutatja, hogy egy rosszul kondicionált probléma számított megoldásának nagy lehet a (relatív) direkt hibája. Egy algoritmust ***direkt stabilnak*** nevezünk, ha a direkt hiba kicsi. Egy direkt stabil módszer nem feltétlenül inverz stabil. Ha az inverz hiba és a kondíciószám kicsi, akkor az algoritmus direkt stabil.

17.2. példa. Vizsgáljuk az $f(x) = \log x$ függvényt. Ennek kondíciószáma $c(f, x) = c(x) = 1/\left|\log x\right|$, amely $x \approx 1$ esetén nagy. Tehát az $x \approx 1$ értékekre a relatív direkt hiba nagy lesz.

17.1.3. Kerekítési hibák és hatásuk a lebegőpontos aritmetikában

A hibaszámítás klasszikus felfogásában csak a bemeneti adatok hibáiból származó, úgynevezett öröklött hibákat vizsgáljuk. A digitális számítógépek a számokat véges sok számjeggyel ábrázolják, így egy F véges számhalmaz elemeivel hajtják végre az aritmetikai műveleteket és ezek eredményeit is az F elemei közül kell kiválasztaniuk. Tehát már a bemeneti adatok eleve meglévő hibái is módosulnak az ábrázolásuk során, majd minden egyes művelet eredménye további torzulást szenvedhet. Ha a művelet eredménye az F halmazbeli szám, akkor a művelet eredményét pontosan kapjuk meg. Egyébként pedig három eset léphet fel:

- kerekítés ábrázolható (nemnulla) számhoz;
- alulcsordulás (kerekítés 0-hoz);
- túlcsordulás (ábrázolhatatlanul nagy abszolút értékű eredmény esetén).

A tudományos-műszaki számítások zömét ún. *lebegőpontos aritmetikában* végezzük. Ennek legáltalánosabban elfogadott modellje a következő.

17.1. definíció (a lebegőpontos számok halmaza).

$$F(\beta, t, L, U) = \left\{ \pm m \times \beta^e \mid \frac{1}{\beta} \le m < 1,\ m = 0.d_1 d_2 \ldots d_t,\ L \le e \le U \right\} \cup \{0\} \ , \qquad (17.2)$$

ahol

- β *a számrendszer alapja;*
- m *a lebegőpontos szám mantisszája a* β *alapú számrendszerben;*
- e *az ábrázolt szám kitevője (karakterisztikája, exponense);*
- t *a mantissza hossza (az aritmetika pontossága);*
- L *a legkisebb kitevő (alulcsordulási határ kitevője);*
- U *a legnagyobb kitevő (túlcsordulási határ kitevője).*

A három leggyakrabban használt számrendszert táblázatban foglaltuk össze.

elnevezés	β	felhasználás
bináris	2	legtöbb számítógép
decimális	10	legtöbb számológép
hexadecimális	16	IBM és hasonló nagyszámítógépek

A mantisszát felírhatjuk az

$$m = 0.d_1 d_2 \dots d_t = \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_t}{\beta^t} \tag{17.3}$$

alakban is. Innen látható, hogy az $1/\beta \le m < 1$ feltétel miatt az első jegyre teljesülnie kell az $1 \le d_1 \le \beta - 1$ egyenlőtlenségnek. A többi számjegyre fennáll, hogy $0 \le d_i \le \beta - 1$ $(i = 2, \dots, t)$. Az ilyen számrendszereket ***normalizáltaknak*** nevezzük. A 0 jegyet és a tizedespontot értelemszerűen nem szokás ábrázolni. Ha $\beta = 2$, akkor az első jegy csak az 1 lehet, amelyet szintén nem ábrázolnak. A (17.3) felírást használva az $F = F(\beta, t, L, U)$ halmazt megadhatjuk az

$$F = \left\{ \pm \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_t}{\beta^t} \right) \beta^e \mid \; L \le e \le U \right\} \cup \{0\} \tag{17.4}$$

alakban is, ahol $0 \le d_i \le \beta - 1$ $(i = 1, \dots, t)$ és $1 \le d_1$.

17.3. példa. A $\beta = 2$, $t = 3$, $L = -1$ és $U = 2$ esetén a 33 elemű F halmaz pozitív része

$$\left\{ \frac{1}{4}, \frac{5}{16}, \frac{6}{16}, \frac{7}{16}, \frac{1}{2}, \frac{5}{8}, \frac{6}{8}, \frac{7}{8}, 1, \frac{10}{8}, \frac{12}{8}, \frac{14}{8}, 2, \frac{20}{8}, 3, \frac{28}{8} \right\} .$$

Az F halmaz elemei nem egyenletesen helyezkednek el a számegyenesen. Az $[1/\beta, 1]$ intervallumba eső F-beli szomszédos számok távolsága β^{-t}. Minthogy az F halmaz elemeit a $\pm m \times \beta^e$ számok alkotják, a szomszédos F-beli számok távolsága az exponens értékének megfelelően változik. A szomszédos elemek legnagyobb távolsága β^{U-t}, a legkisebb pedig β^{L-t}.

A β alapú számrendszerben $m \in [1/\beta, 1 - 1/\beta^t]$, ugyanis

$$\frac{1}{\beta} \le m = \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_t}{\beta^t} \le \frac{\beta - 1}{\beta} + \frac{\beta - 1}{\beta^2} + \dots + \frac{\beta - 1}{\beta^t} = 1 - \frac{1}{\beta^t} .$$

Fenti összefüggés segítségével könnyen igazolható az ábrázolható számok nagyságrendjét jellemző következő tétel.

17.2. tétel. *Ha $a \in F$, $a \neq 0$, akkor $M_L \le |a| \le M_U$, ahol*

$$M_L = \beta^{L-1}, \quad M_U = \beta^U (1 - \beta^{-t}) .$$

Legyen $a, b \in F$ és jelölje $\square$ a négy aritmetikai művelet $(+, -, *, /)$ bármelyikét. A következő esetek lehetségesek:

(1) $a \square b \in F$ (pontos eredmény),

(2) $|a \square b| > M_U$ (aritmetikai túlcsordulás),

(3) $0 < |a \square b| < M_L$ (aritmetikai alulcsordulás),

(4) $a \square b \notin F$, $M_L < |a \square b| < M_U$ (nem ábrázolható eredmény).

Az utolsó két esetben a *lebegőpontos aritmetika* az $a \square b$ eredményhez hozzárendeli a legközelebbi F-beli számot. Ha két szomszédos F-beli szám az $a \square b$ eredménytől egyformán távol van, akkor általában a nagyobbik számhoz kerekítünk. Például ötjegyű decimális

aritmetika esetén a 2.6457513 számot a 2.6458 számhoz kerekítjük.

Legyen $G = [-M_u, M_u]$. Világos, hogy $F \subset G$. Legyen $x \in G$. A kerekítéssel x-hez rendelt F-beli számot jelölje $fl(x)$. Az $x \to fl(x)$ leképezést ***kerekítésnek*** nevezzük, az $|x - fl(x)|$ mennyiséget pedig ***kerekítési hibának.*** Világos, hogy ha $fl(x) = 1$, akkor a kerekítési hiba legfeljebb $\beta^{1-t}/2$. Ezért az $u = \beta^{1-t}/2$ mennyiséget az ***egységnyi kerekítés mértékének*** nevezzük. Az u az $fl(x)$ relatív hibakorlátja. Igaz ugyanis a

17.3. tétel. *Ha $x \in G$, akkor*

$$fl(x) = x(1 + \varepsilon), \quad |\varepsilon| \leq u \ .$$

Bizonyítás. Az általánosság megszorítása nélkül feltehetjük, hogy $x > 0$. Legyen az x számot közrefogó két szomszédos F-beli szám $m_1\beta^e$ és $m_2\beta^e$. Ekkor tehát

$$m_1\beta^e \leq x \leq m_2\beta^e \ ,$$

és vagy $1/\beta \leq m_1 < m_2 \leq 1 - \beta^{-t}$, vagy $1 - \beta^{-t} = m_1 < m_2 = 1$ fennáll. Minthogy $m_2 - m_1 = \beta^{-t}$ mindkét esetben teljesül, akár az $fl(x) = m_1\beta^e$, akár az $fl(x) = m_2\beta^e$ kerekítés következik be, igaz, hogy

$$|fl(x) - x| \leq \frac{|m_2 - m_1|}{2}\beta^e = \frac{\beta^{e-t}}{2} \ .$$

Ebből következik

$$\frac{|fl(x) - x|}{|x|} \leq \frac{|fl(x) - x|}{m_1\beta^e} \leq \frac{\beta^{e-t}}{2m_1\beta^e} = \frac{\beta^{-t}}{2m_1} \leq \frac{1}{2}\beta^{1-t} = u \ .$$

Fennáll tehát, hogy $fl(x) - x = \lambda x u$, ahol $|\lambda| \leq 1$. Ezt átrendezve kapjuk, hogy

$$fl(x) = x(1 + \lambda u) \ .$$

Mivel $|\lambda u| \leq u$, az $\varepsilon = \lambda u$ jelöléssel megkaptuk a tétel állítását. ■

A tétel tulajdonképpen azt mondja ki, hogy a lebegőpontos aritmetikában a kerekítés relatív hibája korlátos és ez a korlát u, az egységnyi kerekítés mértéke.

A kerekítés pontosságának jellemzésére az u kétszeresét szokás használni. Az $\epsilon_M = 2u = \beta^{1-t}$ értéket szokás ***gépi epszilonnak*** is nevezni. Az ϵ_M az 1 és a hozzá legközelebbi 1-nél nagyobb szám távolsága. Bináris alap esetén a következő algoritmussal határozhatjuk meg ϵ_M értékét.

GÉPI-EPSZILON

```
1  x ← 1
2  while 1 + x > 1
3        do x ← x/2
4  ε_M ← 2x
5  return ε_M
```

A MATLAB rendszerben $\epsilon_M \approx 2.2204 \times 10^{-16}$.

A lebegőpontos aritmetikai műveletek eredményére vonatkozóan a következő feltevéssel élünk (szabvány modell):

$$fl(a \square b) = (a \square b)\,(1+\varepsilon)\,, \quad |\varepsilon| \le u \quad (a, b \in F)\ . \tag{17.5}$$

Az IEEE aritmetikai szabvány, amelyet később ismertetünk, kielégíti ezt a feltevést. A feltevés fontos következménye, hogy $a \square b \neq 0$ esetén a műveletek relatív hibájára ugyancsak teljesül, hogy

$$\frac{|fl(a \square b) - (a \square b)|}{|a \square b|} \le u\ .$$

Tehát az aritmetikai műveletek relatív hibája kicsi.

Vannak bizonyos lebegőpontos aritmetikák, amelyek nem elégítik ki a (17.5) feltevést. Ennek az az oka, hogy a kivonásnál nincs egy ún. ellenőrző jegyük.

Az egyszerűség kedvéért vizsgáljuk az $1 - 0.111$ különbséget háromjegyű bináris aritmetikában. Az első lépésben a kitevőket azonos értékre hozzuk

$$\begin{array}{rrcrcccc} & 2 & \times & 0 & . & 1 & 0 & 0 \\ - & 2 & \times & 0 & . & 0 & 1 & 1 & 1 \end{array}\ .$$

Ha a számítást négy értékes jegyre végezzük, akkor az eredmény

$$\begin{array}{rrcrcccc} & 2^1 & \times & 0 & . & 1 & 0 & 0 \\ - & 2^1 & \times & 0 & . & 0 & 1 & 1 & 1 \\ \hline & 2^1 & \times & 0 & . & 0 & 0 & 0 & 1 \end{array}\ ,$$

amelyből a normalizált eredmény $2^{-2} \times 0.100$. Vegyük észre, hogy a kivonásra került szám nem normalizált, mert első jegye 0. A felhasznált ideiglenes negyedik mantisszajegyet ellenőrző jegynek nevezzük. Ha nincs ilyen ellenőrző jegy, akkor a megfelelő számítások a következőképpen alakulnak:

$$\begin{array}{rrcrcccc} & 2^1 & \times & 0 & . & 1 & 0 & 0 \\ - & 2^1 & \times & 0 & . & 0 & 1 & 1 \\ \hline & 2^1 & \times & 0 & . & 0 & 0 & 1 \end{array}\ ,$$

így a normalizált eredmény $2^{-1} \cdot 0.100$. Ennek relatív hibája 100%. Nincs ellenőrző jegye a CRAY szuperszámítógépeknek, valamint egy sor zsebkalkulátornak.

Ha nincs ellenőrző jegy, akkor a műveletek eredményeire az

$$fl\,(x \pm y) = x\,(1+\alpha) \pm y\,(1+\beta)\,, \quad |\alpha|\,, |\beta| \le u\ , \tag{17.6}$$

$$fl\,(x \square y) = (x \square y)\,(1+\delta)\,, \quad |\delta| \le u, \quad \square = *, / \tag{17.7}$$

összefüggések teljesülnek.

Tegyük fel a továbbiakban, hogy van ellenőrző jegy a kivonásnál és teljesül a (17.5) feltevés. Vezessük be a következő jelöléseket:

$$|z| = [|z_1|\,, \ldots, |z_n|]^T \quad (z \in \mathbb{R}^n)\ , \tag{17.8}$$

$$|A| = \left[\left|a_{ij}\right|\right]_{i,j=1}^{m,n} \quad (A \in \mathbb{R}^{m \times n})\ , \tag{17.9}$$

$$A \leq B \Leftrightarrow a_{ij} \leq b_{ij} \quad (A, B \in \mathbb{R}^{m\times n}) \ . \tag{17.10}$$

Igazolhatók az alábbi eredmények, ahol E az aktuális művelet hibáját (hibamátrixát) jelöli:

$$\left|fl\left(x^T y\right) - x^T y\right| \leq 1.01nu\,|x|^T\,|y| \quad (nu \leq 0.01) \ , \tag{17.11}$$

$$fl\,(\alpha A) = \alpha A + E \quad (|E| \leq u\,|\alpha A|) \ , \tag{17.12}$$

$$fl\,(A + B) = (A + B) + E \quad (|E| \leq u\,|A + B|) \ , \tag{17.13}$$

$$fl\,(AB) = AB + E \quad \left(|E| \leq nu\,|A|\,|B| + O\left(u^2\right)\right) \ . \tag{17.14}$$

A szabvány modellnek eleget tevő *lebegőpontos aritmetikáknak* számos sajátos tulajdonsága van. Fontos tulajdonságuk, hogy az összeadás a kerekítés miatt nem asszociatív. Ezt mutatja a következő példa.

17.4. példa. Ha $a = 1$, $b = c = 3 \cdot 10^{-16}$, akkor a MATLAB rendszerben AT386 számítógépen

$$1.000000000000000e + 000 = (a + b) + c \neq a + (b + c) = 1.000000000000001e + 000 \ .$$

Pentium1 100MHz-es processzorú gépen a $b = c = 1.15 \times 10^{-16}$ választás ad hasonló eredményt.

A példa azt is is mutatja, hogy eltérő (numerikus) processzorok esetén azonos számítások eredményei különbözők lehetnek. Nagyszámú adat összegzésénél a kommutativitással (tulajdonképpen asszociativitással) is probléma lehet. Vizsgáljuk most a $\sum_{i=1}^{n} x_i$ összeg kiszámítását. A természetes algoritmus az ún. rekurzív összegzés:

REKURZÍV-ÖSSZEGZÉS(n, x)

```
1  s ← 0
2  for i ← 1 to n
3       do s ← s + x_i
4  return s
```

17.5. példa. Számítsuk ki az

$$s_n = 1 + \sum_{i=1}^{n} \frac{1}{i^2 + i}$$

összeget $n = 4999$ esetén. A rekurzív összegzéssel kapott MATLAB eredmény

$$1.999800000000002e + 000 \ .$$

Ha az összegzést fordított (azaz nagyság szerint növekedő sorrendben végezzük, akkor az eredmény

$$1.999800000000000e + 000 \ .$$

Ha a kétféleképpen kapott értékeket összevetjük az elméleti $s_n = 2 - 1/(n + 1)$ összeggel, akkor láthatjuk, hogy a második összegzés adott pontos eredményt. Ennek magyarázata az, hogy amikor a kisebb tagokkal kezdjük, akkor ezek összegei értékes jegyeket érnek a végső eredményben.

Nagy mennyiségű, előjelben és nagyságrendben eltérő szám nagy pontosságú összeadása nem egyszerű feladat. A következő algoritmus, amely az egyik legérdekesebb ilyen

célra kifejlesztett eljárás, W. Kahantól származik.

KOMPENZÁLT-ÖSSZEGZÉS(n, x)

```
1  s ← 0
2  e ← 0
3  for i ← 1 to n
4       do t ← s
5          y ← x_i + e
6          s ← t + y
7          e ← (t − s) + y
8  return s
```

17.1.4. A lebegőpontos aritmetikai szabvány

Az ANSI/IEEE Std 754-1985 bináris ($\beta = 2$) *lebegőpontosó aritmetikai szabványt* 1985-ben hozták nyilvánosságra. A szabvány specifikálja az alapvető lebegőpontos műveleteket, összehasonlításokat, kerekítési módokat, az aritmetikai kivételeket és kezelésüket, valamint a különböző aritmetikai formák közti konverziót. A négyzetgyökvonás az alapvető műveletek közé tartozik. A szabvány nem mond semmit az exponenciális és transzcendens függvényekről.

A szabvány két fő lebegőpontos formátumot ismer: az egyszeres és a dupla pontosságút.

típus	méret	mantissza	e	u	$[M_L, M_u] \approx$
egyszeres	32 bit	23 + 1 bit	8 bit	$2^{-24} \approx 5.96 \times 10^{-8}$	$10^{\pm 38}$
dupla	64 bit	52 + 1 bit	11 bit	$2^{-53} \approx 1.11 \times 10^{-16}$	$10^{\pm 308}$

Mindkét formátumban egy bitet az előjelnek tartanak fenn. Minthogy a lebegőpontos számok normalizálva vannak és az első jegy mindig 1, ez a jegy nincs tárolva. A mantisszában szereplő +1 ezt a rejtett bitet jelzi.

A szabvány előírja a nem F-beli vagy F-beli számra nem kerekíthető aritmetikai kivételek kezelését is.

kivétel típusa	példa	előírt eredmény
érvénytelen művelet	$0/0$, $0 \times \infty$, $\sqrt{-1}$	NaN (Not a Number)
túlcsordulás	$\lvert x \square y \rvert > M_u$	$\pm\infty$
osztás nullával	véges nemnulla$/0$	$\pm\infty$
alulcsordulás	$0 < \lvert x \square y \rvert < M_L$	szubnormális számok

(***Szubnormális számok*** a $\pm m \cdot \beta^{L-t}$, $0 < m < \beta^{t-1}$ alakú számok.) Az IEEE aritmetika zárt rendszer. Minden aritmetikai műveletnek van matematikailag értelmes vagy értelmetlen eredménye. A kivételes műveletek esetén jelzést ad ki, amely után a számításokat előírásszerűen folytatja. Az IEEE aritmetikai szabvány kielégíti a (17.5) modellt.

Az IEEE szabvány hardver megvalósításai közül ki kell emelni az Intel 80x87 matematikai koprocesszorokat, a 80486 és a Pentium processzorokat, a DEC Alpha, a HP (Precision Architecture), az IBM RS/6000, az INMOS T800 és T900 processzorokat, a Motorola (680x0) és Sun (SPARCstation) processzorokat, valamint a HP tudományos kalkulátorait.

Megjegyzés. Egyszeres pontosság esetén a mantissza hossza kb. 7 értékes jegyet enged meg a tízes számrendszerbe átszámolva. Ugyanez dupla pontosság esetén kb. 16 értékes jegyet jelent. Létezik még egy 80 biten ábrázolt, ún. kiterjesztett pontosság is, ahol $t = 63$, a kitevő pedig 15 bites.

Gyakorlatok

17.1-1. Két ellenállást műszerrel megmértünk és a következő értékeket kaptuk: $R_1 = 110.2 \pm 0.3\Omega$, $R_2 = 65.6 \pm 0.2\Omega$. A párhuzamos kapcsolással kapott eredő ellenállást az ismert $R_e = R_1R_2/(R_1 + R_2)$ képlettel számoljuk. Határozzuk meg a bemeneti adatok *relatív hibakorlátjait* és az eredő ellenállás R_e közelítő értékét. Számítsuk ki a közelítő érték δR_e abszolút és $\delta R_e/R_e$ relatív hibakorlátját háromféleképpen is:

(i) a bemeneti adatoknak csak az abszolút korlátjait használva a δR_e-t és ezután a $\delta R_e/R_e$ relatív korlátot;

(ii) a bemeneti adatoknak csak a relatív korlátjait használva a $\delta R_e/R_e$ relatív korlátot és ezután a δR_e abszolút korlátot;

(iii) az eredő ellenállást $R_e = F(R_1, R_2)$ kétváltozós függvényként tekintve.

17.1-2. Tegyük fel, hogy $\sqrt{2}$ -t 10^{-8} hibakorláttal tudjuk kiszámítani. Melyik kifejezést lehet kisebb relatív hibával kiszámítani és hányszor kisebbel az alábbi két, elméletileg egyenlő két kifejezés közül:

(i) $1/\left(1 + \sqrt{2}\right)^6$;

(ii) $99 - 70\sqrt{2}$.

Magyarázzuk is meg, miért.

17.1-3. Tekintsük az alapműveleteket kétváltozós függvényeknek, azaz $f(x, y) = x\square y$, ahol $\square$ a $+, -, *, /$ műveletek valamelyike.

(i) Vezessük le az alapműveletek *hibakorlátjait,* mint kétváltozós függvényekét.

(ii) Adjuk meg ezen függvények *kondíciószámát.* Mikor rosszul kondicionáltak ezek a függvények?

(iii) Vezessünk le *hibakorlátot* a hatványozásra, úgy is, hogy a kitevőt pontos értéknek tekintjük és úgy is, hogy mind az alap, mind a kitevő hibával terhelt.

(iv) Legyen $y = 16x^2$ és $x \approx a$. A másodrendű hibatagot is figyelembe véve adjuk meg x függvényében az a legnagyobb és legkisebb értékét úgy, hogy az $y \approx b = 16a^2$ közelítéssel számolva b *relatív hibakorlátja* legfeljebb 0.01 legyen.

17.1-4. A $C = \exp(4\pi^2/\sqrt{83})$ $(= 76.1967868\ldots)$ számot 24 bit hosszúságú lebegőpontos aritmetikában számoljuk ki. (Az exponenciális függvényt is 24 értékes bittel adja a számítógépünk.)

Becsüljük meg az eredmény *abszolút hibáját.* Adjuk meg a *relatív hibát* is a C konkrét értékének felhasználása nélkül.

17.1-5. Tekintsünk egy $F(\beta, t, L, U)$ *lebegőpontos számhalmazt.*

(i) Mutassuk meg, hogy bármelyik aritmetikai alapművelet eredményezhet *aritmetikai túlcsordulást.*

(ii) Mutassuk meg, hogy bármelyik művelet okozhat alulcsordulást is.

17.1-6. Mutassuk meg, hogy a következő kifejezések *numerikusan instabilak* $x \approx 0$ esetén:

(i) $(1 - \cos x)/\sin^2 x$;

(ii) $\sin(100\pi + x) - \sin(x)$;

(iii) $2 - \sin x - \cos x - e^{-x}$.

Számítsuk ki a fenti kifejezések értékét $x = 10^{-3}, 10^{-5}, 10^{-7}$ esetén és becsüljük meg a

hibát. Alakítsuk át a kifejezéseket *numerikusan stabillá.*

17.1-7. Hány eleme van az $F = F(\beta, t, L, U)$ halmaznak? Ezek között hány *szubnormális szám* található?

17.1-8. Ismert, hogy nemnegatív x és y mellett a számtani közép nem kisebb a mértaninál, azaz $(x+y)/2 \geq \sqrt{xy}$,és az egyenlőség csak $x = y$ esetén áll fenn. Így van-e ez numerikusan is? Ellenőrizzük kísérletileg az állítást, ha x és y nagyságrendje azonosan igen kicsiny, illetve igen nagy, valamint jelentősen eltérő nagyságrendű x és y esetén is.

17.2. Lineáris egyenletrendszerek

A lineáris egyenletrendszerek általános alakja m egyenlet és n ismeretlen esetén:

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1j}x_j + \cdots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{i1}x_1 + \cdots + a_{ij}x_j + \cdots + a_{in}x_n &= b_i \\ &\vdots \\ a_{m1}x_1 + \cdots + a_{mj}x_j + \cdots + a_{mn}x_n &= b_m\ . \end{aligned} \tag{17.15}$$

Az egyenletrendszert megadhatjuk a tömörebb

$$Ax = b \tag{17.16}$$

formában is, ahol

$$A = \left[a_{ij}\right]_{i,j=1}^{m,n} \in \mathbb{R}^{m\times n},\ x \in \mathbb{R}^n,\ b \in \mathbb{R}^m\ .$$

Ha $m < n$, akkor az egyenletrendszert *alulhatározottnak* nevezzük. Ha $m > n$, akkor *túlhatározott* egyenletrendszerről beszélünk. Az $m = n$ esetben az egyenletrendszert ***négyzetesnek*** nevezzük. Itt csak a négyzetes egyenletrendszerekkel foglalkozunk és feltesszük, hogy az A^{-1} inverz mátrix létezik (ekvivalens feltétellel: $\det(A) \neq 0$). Ebben az esetben az egyenletrendszernek egyértelmű megoldása van.

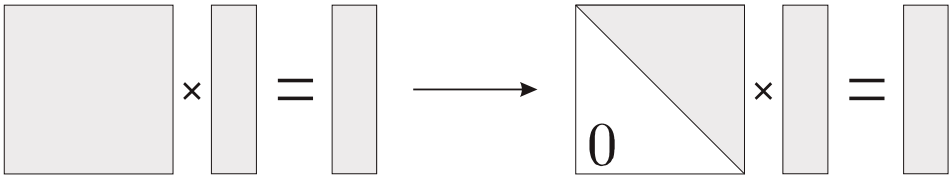
17.2.1. Lineáris egyenletrendszerek megoldásának közvetlen módszerei

Ebben a pontban a Gauss- és Cholesky-módszert, valamint az LU-felbontást mutatjuk be.

Háromszögmátrixú egyenletrendszerek

17.4. definíció. *Az $A = [a_{ij}]_{i,j=1}^n$ mátrix **felső háromszög alakú,** ha minden $i > j$ esetén $a_{ij} = 0$. Ha pedig az $a_{ij} = 0$ minden $i < j$ esetén teljesül, akkor **alsó háromszög alakú.***

Például a felső háromszögmátrixok alakja sematikusan a következő:



17.2. ábra. Gauss-elimináció.

$$\begin{bmatrix} * & * & \cdots & \cdots & * \\ 0 & * & & & \vdots \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & * & * \\ 0 & \cdots & \cdots & 0 & * \end{bmatrix}.$$

Megjegyzés. A diagonálmátrixok egyidejűleg alsó és felső háromszögmátrixok.

Igazolható, hogy alsó vagy felső háromszögmátrixok esetén $\det(A) = a_{11}a_{22}\dots a_{nn}$. A háromszögmátrixú egyenletrendszerek megoldása igen egyszerű. Tekintsük az

$$\begin{array}{ccccccc} a_{11}x_1+ & \cdots & +a_{1i}x_i+ & \cdots & +a_{1n}x_n & = & b_1 \\ \ddots & & \vdots & & \vdots & & \vdots \\ & & a_{ii}x_i+ & \cdots & +a_{in}x_n & = & b_i \\ & & & \ddots & \vdots & & \vdots \\ & & & & a_{nn}x_n & = & b_n \end{array}$$

felső háromszögmátrixú egyenletrendszert. Ennek megoldását a következő, ún. *visszahelyettesítő* algoritmus adja:

VISSZAHELYETTESÍTÉS(A, b, n)

1 $x_n \leftarrow b_n/a_{nn}$
2 **for** $i \leftarrow n-1$ **downto** 1
3 **do** $x_i \leftarrow (b_i - \sum_{j=i+1}^{n} a_{ij}x_j)/a_{ii}$
4 **return** x

Az alsó háromszögmátrixú egyenletrendszer megoldása hasonló.

A Gauss-módszer

A Gauss-féle eliminációs módszer (lásd 17.2) két fázisból áll:

I. Azonos átalakításokkal az $Ax = b$ egyenletrendszert felső háromszög alakúra hozzuk. háromszög alakúra hozzuk. (A 17.2 ábra sematikusan mutatja, hogy a teli együtthatómátrixú $A \times x = b$ egyenletrendszerből olyan $\hat{A} \times \hat{x} = \hat{b}$ egyenletrendszert hozunk létre, amelynek az $\hat{A}$ együtthatómátrixa már felső háromszögmátrix.)

II. A kapott felső háromszögmátrixú egyenletrendszert a *visszahelyettesítő* algoritmussal megoldjuk. A felső háromszög alakra hozás során az együtthatómátrix főátló alatti ele-

meit az egyenletrendszer ekvivalens átalakításával „nullázzuk”. Ehhez azt az észrevételt használjuk fel, hogy alkalmasan megválasztott γ konstanssal valamely i-edik egyenletből egy másik, pl. k-adik egyenlet γ-szorosát kivonva, az i-edik egyenletből az egyik ismeretlen kiiktatható (*eliminálható*), miközben az egyenletrendszer megoldása természetesen nem változik.

Tegyük fel, hogy az első $(k-1)$ oszlopban a nullázást már elvégeztük és az

$$\begin{array}{ccccccccccc}
a_{11}x_1+ & \cdots & \cdots & +a_{1k}x_k & + & \cdots & + & a_{1n}x_n & = & b_1 \\
 & \ddots & & \vdots & & & & \vdots & & \vdots \\
 & & \ddots & \vdots & & & & \vdots & & \vdots \\
 & & & a_{kk}x_k & + & \cdots & + & a_{kn}x_n & = & b_k \\
 & & & \vdots & & & & \vdots & & \vdots \\
 & & & a_{ik}x_k & + & \cdots & + & a_{in}x_n & = & b_i \\
 & & & \vdots & & & & \vdots & & \vdots \\
 & & & a_{nk}x_k & + & \cdots & + & a_{nn}x_n & = & b_n
\end{array}$$

egyenletrendszert kaptuk. Ha $a_{kk} \neq 0$, akkor az a_{kk} alatti x_k együtthatókat az i-edik sorból a k-adik sor $\gamma = a_{ik}/a_{kk}$-szorosa kivonásával nullázhatjuk ki. Ugyanis az

$$(a_{ik} - \gamma a_{kk})x_k + (a_{i,k+1} - \gamma a_{k,k+1})x_{k+1} + \cdots + (a_{in} - \gamma a_{kn})x_n = b_i - \gamma b_k$$

egyenletben a választott γ-val éppen a kívánt $a_{ik} - \gamma a_{kk} = 0$ adódik. Ezt az előírást végrehajtva a $k = 1, 2, \ldots, n-1$ oszlopok és egy-egy oszlopon belül az $i = k+1, \ldots, n$ sorszámú sorok mindegyikére, kialakul a felsőháromszög mátrix. A következőkben $A[i, j]$ az A mátrix a_{ij} elemét, az $A[i, j : n]$ pedig a mátrix i-edik sorának a j-edik oszloptól kezdődő szeletét jelöli. Ezzel az egyenletrendszert megoldó algoritmus (az algoritmusban már itt feltüntettük a következő pontban tárgyalt pivotálás helyét is):

GAUSS-MÓDSZER(A, b)

```
0  ▷ I. (eliminációs) fázis.
1  n ← sorok[A]
2  for k ← 1 to n − 1
3      do {pivotálás esetén főelemkiválasztás, majd sor-, illetve oszlopcsere}
4          for i ← k + 1 to n
5              do γ_ik ← A[i, k] /A[k, k]
6                 A[i, k + 1 : n] ← A[i, k + 1 : n] − γ_ik ∗ A[k, k + 1 : n]
7                 b_i ← b_i − γ b_k
8  ▷ II. (visszahelyettesítő) fázis: lásd a VISSZAHELYETTESÍTÉS algoritmust.
9  return x
```

Az algoritmus felülírja az eredeti mátrix és a jobboldali vektor elemeit, ugyanakkor a főátló alatti nullákat nem írja be. A nullák beírása egyrészt felesleges lenne, hisz azokra az elemekre a II. fázis nem hivatkozik. Ugyanakkor a nullák helyén a később tárgyalandó LU-felbontáshoz szükséges információkat őrizhetjük meg.

A Gauss-módszert természetesen csak akkor lehet végrehajtani, ha a mindenkori a_{kk}

elem nem nulla. Emiatt is, de főként a numerikus stabilitás miatt a módszert főelemkiválasztással szoktuk alkalmazni.

A főelemkiválasztásos Gauss-módszer
Amennyiben az a_{kk} elem nulla, az *eliminációs eljárást* megkísérelhetjük úgy folytatni, hogy sorcserével a (k,k) pozícióra nullától különböző elem kerüljön. Amennyiben ez sem lehetséges, mert az $a_{kk}, a_{k+1,k}, \ldots, a_{nk}$ mindegyike nulla, akkor az együtthatómátrix determinánsa nulla, egyértelmű megoldás nem is létezik. Az a_{kk} elemet ***k-adik pivotelemnek*** nevezzük. A sorok felcserélésével új pivot elem választható. A pivot elem megválasztása nagymértékben befolyásolja az eredmények megbízhatóságát. Már csak amiatt is, hogy osztunk vele, amely művelet hibája – mint korábban láttuk – a nevező négyzetével fordítva arányos. A lehetőség szerinti minél nagyobb abszolút értékű pivot elem választás az előnyös. Az ilyen választást *pivotálási,* vagy *főelemkiválasztási eljárásnak* nevezzük. Kétféle pivotálási stratégiát említünk meg.

Részleges főelemkiválasztás: A k-adik lépésben a k-adik oszlop a_{jk} $(k \le j \le n)$ elemei közül kiválasztjuk a maximális abszolút értékűt. Ha ennek indexe i, akkor a k-adik és az i-edik sort felcseréljük. A pivotálás után teljesül, hogy

$$|a_{kk}| = \max_{k \le i \le n} |a_{ik}| \ .$$

Teljes főelemkiválasztás : A k-adik lépésben az a_{ij} $(k \le i, j \le n)$ mátrixelemek közül kiválasztjuk a maximális abszolút értékűt. Ha ennek indexe (i, j), akkor a k-adik és az i-edik sort, valamint a k-adik oszlopot és j-edik oszlopot felcseréljük. A pivotálás után teljesül, hogy

$$|a_{kk}| = \max_{k \le i,j \le n} \left|a_{ij}\right| \ .$$

Megjegyezzük, hogy oszlopcsere esetén változócsere is történik.

Jól illusztrálja a pivotálás jelentőségét a következő

17.6. példa.

$$\begin{array}{rcrcl} 10^{-17}x & + & y & = & 1\,, \\ x & + & y & = & 2\,. \end{array}$$

Ezen egyenletrendszer pontos megoldása: $x = 1/(1-10^{-17})$, $y = 1 - 10^{-17}/(1-10^{-17})$. A MATLAB duplapontos szabvány szerinti számábrázolásában $fl(x) = fl(y) = 1$, vagyis ennél jobb közelítést a MATLAB-ban nem is érhetünk el. Pivotálás nélküli Gauss-módszerrel megoldva az $x = 0$, $y = 1$ katasztrofálisan hibás eredmény adódik, míg részleges főelemkiválasztással megkapjuk az elméletileg elérhető legjobb, azaz $x = y = 1$ közelítő megoldást.

17.5. megjegyzés. *Nem kell végrehajtani főelemkiválasztást a következő esetekben:*
1. Ha A szimmetrikus és pozitív definit ($A \in \mathbb{R}^{n\times n}$ pozitív definit $\Leftrightarrow x^T Ax > 0$, $\forall x \in \mathbb{R}^n$, $x \neq 0$) .
2. Ha A diagonálisan domináns a következő értelemben:

$$|a_{ii}| > \sum_{j \neq i} \left|a_{ij}\right| \quad (1 \le i \le n) \ .$$

Szimmetrikus és pozitív definit A mátrix esetén a Gauss-elimináció egy később ismertetendő speciális alakját, a Cholesky-módszert használjuk az egyenletrendszer megoldására.

A Gauss-elimináció során valójában egymástól különböző együttható mátrixú egyenletrendszereket kapunk (habár a közölt algoritmus szerint a számítógép fizikailag egy helyen tárolja valamennyit és az alsó háromszög részbe nem is írja be a nullákat), azaz az

$$A^{(0)}x = b^{(0)} \to A^{(1)}x = b^{(1)} \to \cdots \to A^{(n-1)}x = b^{(n-1)}$$

ekvivalens egyenletrendszerekből álló sorozatot, ahol

$$A^{(k)} = \left[a_{ij}^{(k)}\right]_{i,j=1}^{n} .$$

Az eliminációs fázis végén az

$$A^{(n-1)} = \begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & a_{nn}^{(n-1)} \end{bmatrix}$$

mátrix alakul ki, ahol $a_{kk}^{(k-1)}$ a k-adik fő-, vagy pivotelem. A ***pivotelemek növekedési tényezője:***

$$\rho = \rho_n = \max_{1 \le k \le n} \left| a_{kk}^{(k-1)} / a_{11}^{(0)} \right| .$$

Igen fontos kérdés a ρ növekedési tényező nagyságrendje, mert ez összefüggésben áll az eljárás numerikus stabilitásával. Wilkinson igazolta, hogy a közelítő megoldás hibája arányos a ρ növekedési tényezővel, amelyre teljes főelemkiválasztás esetén a

$$\rho \le \sqrt{n}\left(2 \cdot 3^{\frac{1}{2}} \cdots n^{\frac{1}{n-1}}\right)^{\frac{1}{2}} \sim c n^{\frac{1}{2}} n^{\frac{1}{4} log(n)} ,$$

részleges főelemkiválasztás esetén pedig a

$$\rho \le 2^{n-1}$$

korlát teljesül. Wilkinson azt sejtette, hogy teljes főelemkiválasztás esetén $\rho \le n$. Ezt kis n értékekre többen is igazolták. Véletlen mátrixokon végzett statisztikai vizsgálatok ($n \le 1024$) azt mutatják, hogy ρ nagyságrendje átlagosan $\Theta\left(n^{2/3}\right)$ a részleges és $\Theta\left(n^{1/2}\right)$ a teljes főelemkiválasztás esetén. Tehát a $\rho > n$ eset statisztikai értelemben ritkán fordulhat elő.

Megjegyezzük, hogy Wilkinson konstruált egy példát, amelyre részleges főelemkiválasztás esetén $\rho = 2^{n-1}$, tehát ez a korlát pontos. Újabban találtak néhány, a gyakorlatban is (differenciál- és integrálegyenletek közelítő megoldásánál) előforduló esetet, amikor a részleges főelemkiválasztáson alapuló Gauss-módszer ρ növekedési tényezője exponenciálisan nő és a módszer csődöt mond.

A főelemkiválasztás nélküli Gauss-elimináció esetén a növekedési tényező nagyon nagy lehet. Például az

$$A = \begin{bmatrix} 1.7846 & -0.2760 & -0.2760 & -0.2760 \\ -3.3848 & 0.7240 & -0.3492 & -0.2760 \\ -0.2760 & -0.2760 & 1.4311 & -0.2760 \\ -0.2760 & -0.2760 & -0.2760 & 0.7240 \end{bmatrix}$$

mátrix esetén a növekedési tényező $\rho = \rho_4\,(A) = 1.23 \times 10^5$.

A Gauss-módszer műveletigénye

A Gauss-módszer véges sok lépésben, véges sok aritmetikai alapművelet ($+, -, *, /$) elvégzése után megadja az $Ax = b$ ($A \in \mathbb{R}^{n\times n}$) egyenletrendszer megoldását. A szükséges aritmetikai műveletszám (műveletigény) az egyenletrendszer megoldó eljárások fontos minőségi jellemzője, mert az ilyen algoritmusok számítógépideje nagyjából arányos az aritmetikai műveletigénnyel. Megfigyelték, hogy a lineáris algebra számítási eljárásaiban az additív és a multiplikatív műveletek száma nagyon gyakran közel azonos. Egy multiplikatív művelet ideje hagyományos számítógépeken lényegesen több mint egy additív műveleté, de ez a nagyságrendi eltérés nem akkora, hogy az additív műveleteket elhanyagolhatnánk. C. B. Moler a számítási igény mérésére bevezette az *1 (régi) flop* fogalmát. A nagyteljesítményű szuperszámítógépeken nincs lényeges különbség az additív és a multiplikatív műveletekre fordított idő között, ezért újabban az *új flop* fogalmát is használják.

17.6. definíció. ***1 (régi) flop*** *az a számítási munka, amely az $s = s+x*y$ művelet (1 összeadás + 1 szorzás) elvégzéséhez kell,* ***1 (új) flop*** *pedig az a számítási munka, amely egy $+, -, *, /$ aritmetikai művelet elvégzéséhez kell.*

Egy régi flop 2 új floppal azonos. Mi a régi flop értelmezést használjuk. A Gauss-módszer additív és multiplikatív műveletigényét egyszerű számolással megkapjuk, a teljes műveletigényt mondja ki a következő

17.7. tétel. *A Gauss-módszer műveletigénye $n^3/3 + \Theta(n^2)$ flop.*

Klyuyev és Kokovkin-Shcherbak igazolta, hogy ha csak sor- és oszlopműveleteket (sor, vagy oszlop számmal való szorzása; sorok, vagy oszlopok cseréje; sorok, vagy oszlopok számszorosának sorokhoz, vagy oszlopokhoz való hozzáadása) engedünk meg, akkor nem lehet $n^3/3 + \Omega(n^2)$ flopnál kevesebb művelettel az $Ax = b$ lineáris egyenletrendszert megoldani.

Az $Ax = b$ alakú $n \times n$-es egyenletrendszerek megoldásához szükséges műveletigény *gyors mátrixinvertáló* eljárásokkal $O(n^{2.808})$ flopra leszorítható. A jelenleg ismert eljárásokat numerikus instabilitásuk miatt gyakorlatilag nem használják.

Az *LU*-felbontás

Sok esetben könnyebb a problémát megoldani, ha valamely mátrixot két, bizonyos szempontból előnyösebb tulajdonságú mátrix – pl. két háromszögmátrix – szorzatára tudunk bontani.

17.8. definíció. *Az $A \in \mathbb{R}^{n\times n}$ mátrix* ***LU-felbontásán*** *a mátrix $A = LU$ szorzatalakban történő felbontását értjük, ahol $L \in \mathbb{R}^{n\times n}$ alsó, $U \in \mathbb{R}^{n\times n}$ pedig felső háromszögmátrix.*

Az *LU*-felbontás nem egyértelmű. Viszont, ha egy nemszinguláris mátrixnak létezik *LU*-felbontása, akkor olyan is létezik, amelyben valamelyik tényező főátlójában csupa egyes áll. Az ilyen háromszögmátrixot ***egység háromszögmátrixnak*** nevezzük. Ez a felbontása a nemszinguláris mátrixoknak már egyértelmű (persze, ha egyáltalán létezik).

Nemszinguláris mátrixok *LU*-felbonthatóságára a Gauss-eliminációval való kapcsolat ad választ. Igazolható, hogy abban az $A = LU$ szorzatban, amelyben L egység alsóháromszög mátrix, a főátló alatti l_{ik} elemekre $l_{ik} = \gamma_{ik}$, ahol γ_{ik} a Gauss-módszer algoritmus szerinti. Az U pedig az algoritmus eliminációs fázisa végeredményeként előállított felső

háromszögmátrix. A L is kiolvasható ebből a táblából, amennyiben az alsó háromszögrész oszlopait végigosztjuk a főátlóbeli elemekkel. Emlékeztetünk arra, hogy a Gauss-módszer algoritmus végrehajtása során a főátló alatti elemeket fizikailag nem nulláztuk ki.) Világos, hogy nemszinguláris A esetén akkor és csak akkor létezik LU-felbontás, ha a pivotálás nélküli Gauss-elimináció során $a_{kk}^{(k-1)} \neq 0$ minden pivotelemre teljesül.

17.9. definíció. *A négyzetes P mátrixot **permutációmátrixnak** nevezzük, ha minden sorában és oszlopában pontosan egy darab 1-es van és a többi elem zérus.*

Részleges főelemkiválasztás esetén a sorokat permutáljuk (más szóval: egy permutációmátrixszal szorzunk balról) és az $a_{kk} \neq 0$ $(k = 1, \ldots, n)$ minden nemszinguláris mátrixra teljesül. Igaz tehát a következő

17.10. tétel. *Ha az $n \times n$-es A mátrix nemszinguláris, akkor létezik olyan P permutációmátrix, hogy a PA mátrixnak van LU-felbontása.*

Az LU-felbontás algoritmusa tehát lényegében a Gauss-elimináció algoritmusa. Természetesen, ha részleges főelemkiválasztással keressük a felbontást, akkor a sorcseréket a főátló alatti (fizikailag nem nullázott) elemeken is el kell végezni és a P mátrixot is meg kell jegyezni (pl. úgy, hogy egy vektorban tároljuk a mátrixsoroknak az eredetihez képesti mindenkori sorrendjét).

Az LU- és Cholesky-módszerek

Legyen $A = LU$ és vizsgáljuk az $Ax = b$ megoldását. Ez az $Ax = LUx = L(Ux) = b$ összefüggés miatt felbontható az $Ly = b$ alsó háromszögmátrixú és az $Ux = y$ felső háromszögmátrixú egyenletrendszerek megoldására.

LU-MÓDSZER(A, b)

1 határozzuk meg az $A = LU$ felbontást
2 oldjuk meg az $Ly = b$ egyenletrendszert
3 oldjuk meg az $Ux = y$ egyenletrendszert
4 **return** x

Megjegyzés. Ha részleges főelemkiválasztást alkalmazunk, akkor értelemszerűen az $\hat{A} = PA = LU$ felbontást kapjuk (kimenetként a P-t is), majd jobb oldalként a $\hat{b} = Pb$ vektort vesszük.

Az eredeti Gauss-módszer I. fázisában az $A = LU$ felbontást és az $Ux = L^{-1}b$ felső háromszögmátrixú egyenletrendszert állítjuk elő. A II. fázisban ezt az egyenletrendszert oldjuk meg. Az LU-módszerben a Gauss-módszer I. fázisát két lépésre bontjuk fel. Az első lépésben az $A = LU$ felbontást állítjuk elő és értelemszerűen nem végzünk számításokat a b oszlopvektoron. Az eljárás második lépésében az $y = L^{-1}b$ vektort állítjuk elő. Az eljárás harmadik lépése megegyezik az eredeti Gauss-módszer II. fázisával.

Az LU-módszer különösen előnyös, ha ugyanazon együtthatómátrixszal egynél több

$$Ax = b_1,\ Ax = b_2, \ldots,\ Ax = b_k$$

alakú egyenletrendszert kell megoldani. Ekkor elég az A mátrix LU-felbontását egyszer

meghatározni, majd rendre az $Ly_i = b_i$, $Ux_i = y_i$ ($x_i, y_i, b_i \in \mathbb{R}^n$, $i = 1, \ldots, k$) háromszögmátrixú egyenletrendszereket megoldani. Az eljárás összköltsége: $n^3/3 + kn^2 + \Theta(kn)$ flop.

Ezen észrevétel alapján egy $A \in \mathbb{R}^{n \times n}$ *mátrix invertálását* az LU-módszer segítségével a következőképpen végezhetjük:

1. Meghatározzuk az $A = LU$ felbontást.

2. Sorra meghatározzuk az $Ly_i = e_i$, $Ux_i = y_i$ (e_i az i-edik egységvektor, $i = 1, \ldots, n$) egyenletrendszerek x_i megoldásait. Az A inverze $A^{-1} = [x_1, \ldots, x_n]$.

Az eljárás műveletigénye $4n^3/3 + \Theta\left(n^2\right)$ flop.

Az LU-módszer pointeres technikával

Lényegében a 60-as évek eleje óta ismert. A P vektor tartalmazza a sorok indexeit. Induláskor $P[i] = i$ ($1 \leq i \leq n$). Sorcserék esetén ténylegesen csak a P vektor megfelelő indexű elemeit cseréljük ki.

LU-MÓDSZER-POINTERES-TECHNIKÁVAL(A, b)

```
 1  n ← sorok[A]
 2  P ← [1, 2, ..., n]
 3  for k ← 1 to n − 1
 4      do határozzuk meg a t indexet, amelyre |A[P[t], k]| = max_{k≤i≤n} |A[P[i], k]|
 5         if k < t
 6            then cseréljük fel a P[k] és a P[t] komponensek értékét
 7         for i ← k + 1 to n
 8             do A[P[i], k] ← A[P[i], k] / A[P[k], k]
 9                A[P[i], k + 1 : n] ← A[P[i], k + 1 : n] − A[P[i], k] * A[P[k], k + 1 : n]
10  for i ← 1 to n
11      do s ← 0
12         for j ← 1 to i − 1
13             do s ← s + A[P[i], j] * x[j]
14         x[i] ← b[P[i]] − s
15  for i ← n downto 1
16      do s ← 0
17         for j ← i + 1 to n
18             s ← s + A[P[i], j] * x[j]
19         x[i] ← (x[i] − s) / A[P[i], i]
20  return x
```

Ha az $A \in \mathbb{R}^{n \times n}$ mátrix *szimmetrikus és pozitív definit*, akkor felbontható $A = LL^T$ alakban, ahol L alsó háromszögmátrix. Ezt a felbontást ***Cholesky-felbontásnak*** nevezzük. Ekkor nem kell tárolni az A mátrixnak közel a felét és az LU-felbontás (LL^T-felbontás) kiszámításának is kb. a fele megtakarítható. Legyen

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ 0 & l_{22} & \cdots & l_{n2} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & l_{nn} \end{bmatrix}.$$

Figyelembe véve, hogy az L^T mátrix k-adik oszlopában csak az első k elem lehet nemnulla, kapjuk, hogy

$$\begin{aligned} a_{kk} &= l_{k1}^2 + l_{k2}^2 + \cdots + l_{k,k-1}^2 + l_{kk}^2 \ , \\ a_{ik} &= l_{i1}l_{k1} + l_{i2}l_{k2} + \cdots + l_{i,k-1}l_{k,k-1} + l_{ik}l_{kk} \quad (i = k+1, \ldots, n) \ . \end{aligned}$$

Innen pedig

$$\begin{aligned} l_{kk} &= (a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2)^{1/2}, \\ l_{ik} &= (a_{ik} - \sum_{j=1}^{k-1} l_{ij}l_{kj})/l_{kk} \quad (i = k+1, \ldots, n) \ . \end{aligned}$$

Ennek alapján az eljárás algoritmusa, felhasználva, hogy megállapodás szerint $\sum_{j=i}^{k} s_j = 0$, ha $k < i$:

CHOLESKY-MÓDSZER(A)

```
1  n ← sorok[A]
2  for k ← 1 to n
3      do a_kk ← (a_kk − Σ_{j=1}^{k−1} a_kj^2)^{1/2}
4         for i ← k + 1 to n
5             do a_ik ← (a_ik − Σ_{j=1}^{k−1} a_ij a_kj)/a_kk
6  return A
```

Az A mátrix alsó háromszög része fogja tartalmazni L-et. Az eljárás számítási költsége $n^3/6 + \Theta(n^2)$ flop és n négyzetgyökvonás. Az eljárás, amely a Gauss-elimináció speciális esetének tekinthető, nem igényel pivotálást.

Az *LU*- és a Cholesky-módszer sávmátrixokon
Gyakran találkozunk olyan egyenletrendszerrel, amely együttható mátrixa sávmátrix.

17.11. definíció. *Az $A \in \mathbb{R}^{n\times n}$ mátrix **sávmátrix** p alsó sávszélességgel és q felső sávszélességgel, ha teljesül, hogy*

$$a_{ij} = 0, \quad \textit{ha } i > j + p \textit{ vagy } i + q < j \ .$$

A sávot, amelynek elemei lehetnek nemnullák, azon a_{ij} elemek definiálják, amelynek indexeire teljesül, hogy $i - p \le j \le i + q$, vagy ekvivalens módon $j - q \le i \le j + p$. Sematikusan ábrázolva

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1,1+q} & 0 & \cdots & \cdots & 0 \\ a_{21} & a_{22} & & & & \ddots & & & \vdots \\ \vdots & & \ddots & & & & \ddots & & \vdots \\ a_{1+p,1} & & & \ddots & & & & \ddots & 0 \\ 0 & \ddots & & & \ddots & & & & a_{n-q,n} \\ \vdots & & \ddots & & & \ddots & & & \vdots \\ \vdots & & & \ddots & & & \ddots & & \vdots \\ \vdots & & & & \ddots & & & \ddots & a_{n-1,n} \\ 0 & \cdots & \cdots & \cdots & 0 & a_{n,n-p} & \cdots & a_{n,n-1} & a_{nn} \end{bmatrix}.$$

A *sávmátrixok* akkor érdekesek, ha p és q jóval kisebb mint n. Ismert, hogy ha létezik LU-felbontás, akkor L és U is sávmátrix, az A-beli alsó és felső sávszélességgel.A következőkben három részletben megadjuk az LU-módszer sávos változatát.

SÁVMÁTRIX-LU-FELBONTÁSA(A, n, p, q)

```
1  for k ← 1 to n − 1
2      do for i ← k + 1 to min {k + p, n}
3          do a_ik ← a_ik / a_kk
4             for j ← k + 1 to min {k + q, n}
5                 do a_ij ← a_ij − a_ik a_kj
6  return A
```

Az a_{ij} fogja tartalmazni l_{ij}-t, ha $i > j$ és u_{ij}-t, ha $i \leq j$. Az algoritmus műveletigénye $c\,(p, q)$ flop, ahol

$$c\,(p,q) = \begin{cases} npq - \frac{1}{2}pq^2 - \frac{1}{6}p^3 + pn, & p \leq q \\ npq - \frac{1}{2}qp^2 - \frac{1}{6}q^3 + qn, & p > q \end{cases}$$

A következő algoritmus felülírja a b vektort az $Ly = b$ egyenletrendszer megoldásával.

SÁVOS-ALSÓ-HÁROMSZÖGMÁTRIXÚ-EGYENLETRENDSZER(L, b, n, p)

```
1  for i ← 1 to n
2      do b_i ← b_i − Σ_{j=max{1,i−p}}^{i−1} l_ij b_j
3  return b
```

Az algoritmus műveletigénye $np - p^2/2$ flop.

A következő algoritmus felülírja a b vektort az $Ux = b$ egyenletrendszer megoldásával.

SÁVOS-FELSŐ-HÁROMSZÖGMÁTRIXÚ-EGYENLETRENDSZER(U, b, n, q)

```
1  for i ← n downto 1
2      do b_i ← ( b_i − Σ_{j=i+1}^{min{i+q,n}} u_ij b_j ) / u_ii
3  return b
```

Az algoritmus műveletigénye $n(q+1) - q^2/2$ flop.

Legyen $A \in \mathbb{R}^{n\times n}$ szimmetrikus, pozitív definit p alsó sávszélességgel. A *Cholesky*-felbontás sávos változata:

SÁVMÁTRIXOK-CHOLESKY-FELBONTÁSA(A, n, p)

1 **for** $i \leftarrow 1$ **to** n
2 **do for** $j \leftarrow \max\{1, i-p\}$ **to** $i-1$
3 **do** $a_{ij} \leftarrow \left(a_{ij} - \sum_{k=\max\{1,i-p\}}^{j-1} a_{ik}a_{jk}\right)/a_{jj}$
4 $a_{ii} \leftarrow \left(a_{ii} - \sum_{k=\max\{1,i-p\}}^{i-1} a_{ik}^2\right)^{1/2}$
5 **return** A

Az a_{ij} elemek tartalmazzák az l_{ij} elemeket ($i \geq j$). Az eljárás műveletigénye $\left(np^2/2\right) - \left(p^3/3\right) + (3/2)\left(np - p^2\right)$ flop és n négyzetgyökvonás.

Megjegyzés. Ha az $A \in \mathbb{R}^{n\times n}$ p alsó és q felső sávszélességű sávmátrixon részleges főelemkiválasztást kell végrehajtani, akkor az U sávos felső háromszögmátrix sávszélessége a sorcsere végrehajtásakor megnőhet, legfeljebb a $\hat{q} = p + q$ értékre.

17.2.2. Lineáris egyenletrendszerek iteratív megoldási módszerei

Lineáris egyenletrendszerek megoldására számos iteratív módszer ismert. Ezek közül legismertebbek a klasszikus *Jacobi*, a *Gauss-Seidel* és a különféle *relaxációs módszerek*. Az iteratív módszerek legfőbb előnye a nagyméretű rendszerekre történő könnyű alkalmazhatóságuk. Hátrányuk ugyanakkor az esetleges lassú konvergencia, amely háttérbe szorította ezeket a módszereket. Bizonyos típusú feladatok megoldásánál azonban, a könnyű párhuzamosíthatóság miatt újra előtérbe kerültek az ún. multifelbontású („multisplitting") iteratív algoritmusok.

Tekintsük az

$$x_i = Gx_{i-1} + b \qquad (i = 1, 2, \ldots)$$

iterációt, ahol $G \in \mathbb{R}^{n\times n}$ és $x_0, b \in \mathbb{R}^n$. Ismert, hogy az $\{x_i\}_{i=0}^{\infty}$ akkor és csak akkor konvergál minden x_0, $b \in \mathbb{R}^n$ esetén, ha a G mátrix spektrálsugarára $\rho(G) < 1$ teljesül ($\rho(G) = \max\{|\lambda| \mid \lambda \text{ a } G \text{ sajátértéke}\}$). Konvergencia esetén $x_i \to x^* = (I-G)^{-1}b$, azaz az $(I-G)x = b$ egyenletrendszer megoldását kapjuk. A konvergencia gyorsasága a $\rho(G)$ spektrálsugár nagyságától függ. Minél kisebb $\rho(G)$, annál gyorsabb a konvergencia.

Tekintsük most az

$$Ax = b$$

egyenletrendszert, ahol $A \in \mathbb{R}^{n\times n}$ nemszinguláris mátrix. Az $M_l, N_l, E_l \in \mathbb{R}^{n\times n}$ mátrixokat az A ***multifelbontásának*** nevezzük, ha

(i) $A = M_i - N_i$, $i = 1, 2, \ldots, L$,

(ii) M_i nemszinguláris, $i = 1, 2, \ldots, L$,

(iii) E_i nemnegatív diagonális mátrix, $i = 1, 2, \ldots, L$,

(iv) $\sum_{i=1}^{L} E_i = I$.

Adott $x_0 \in \mathbb{R}^n$ kezdővektorral a felbontáshoz tartozó iteratív módszer a következő.

ITERÁCIÓ-MULTIFELBONTÁSSAL($x_0, L, M_l, N_l, E_l, l = 1, \ldots, L$)

```
1  i ← 0
2  while kilépési feltétel = HAMIS
3        do i ← i + 1
4           for l ← 1 to L
5                do M_l y_l ← N_l x_{i-1} + b
5           x_i ← Σ_{l=1}^{L} E_l y_l
7  return x_i
```

Egyszerű számolással kapjuk, hogy $y_l = M_l^{-1} N_l x_{i-1} + M_l^{-1} b$ és

$$x_i = \sum_{l=1}^{L} E_l y_l = \sum_{l=1}^{L} E_l M_l^{-1} N_l x_{i-1} + \sum_{l=1}^{L} E_l M_l^{-1} b = H x_{i-1} + c \ .$$

Tehát a konvergencia feltétele: $\rho(H) < 1$. A *multifelbontás algoritmus* valódi *párhuzamos algoritmus* , mert iterációnként L lineáris egyenletrendszert lehet párhuzamosan megoldani *(szinkronizált párhuzamosság).* Az eljárás szűk keresztmetszete x_i iterált számítása.

Az M_i mátrixok és az E_i „súlymátrixok" megválasztása célszerűen úgy történik, hogy az $M_i y = c$ egyenletrendszer megoldása „olcsó" legyen. Legyen $S_1, S_2, \ldots, S_L$ az $\{1, \ldots, n\}$ halmaz partíciója, azaz $S_i \neq \emptyset$, $S_i \cap S_j = \emptyset$ $(i \neq j)$ és $\cup_{i=1}^{L} S_i = \{1, \ldots, n\}$. Legyenek továbbá az $S_i \subseteq T_i \subseteq \{1, \ldots, n\}$ halmazok $(i = 1, \ldots, L)$ olyanok, hogy legalább egy l indexre $S_l \neq T_l$.

Az A mátrix ***nemátfedő blokk Jacobi-multifelbontását*** az

$$M_l = \left[M_{ij}^{(l)}\right]_{i,j=1}^{n}, \quad M_{ij}^{(l)} = \begin{cases} a_{ij}, & \text{ha } i, j \in S_l \ , \\ a_{ii}, & \text{ha } i = j \ , \\ 0 & \text{egyébként,} \end{cases} \ ,$$

$$N_l = M_l - A \ ,$$

$$E_l = \left[E_{ij}^{(l)}\right]_{i,j=1}^{n}, \quad \widetilde{E}_{ij}^{(l)} = \begin{cases} 1, & \text{ha } i = j \in S_l \\ 0 & \text{egyébként} \end{cases}$$

előírás $(l = 1, 2, \ldots, L)$ definiálja.

Definiáljuk az

$$A = M - N$$

egyszerű felbontást is, ahol M nemszinguláris,

$$M = \left[M_{ij}\right]_{i,j=1}^{n}, \quad M_{ij} = \begin{cases} a_{ij}, & \text{ha } i, j \in S_l \text{ valamely } l \in \{1, \ldots, n\} \text{ értékre} \ , \\ 0 & \text{egyébként} \ . \end{cases} \ .$$

Megmutatható, hogy a nemátfedő blokk Jacobi-multifelbontásra fennáll, hogy

$$H = \sum_{l=1}^{L} E_l M_l^{-1} N_l = M^{-1} N \ .$$

Az A mátrix ***átfedő blokk Jacobi-multifelbontását*** az

$$\widetilde{M}_l = \left[\widetilde{M}_{ij}^{(l)}\right]_{i,j=1}^{n}, \quad \widetilde{M}_{ij}^{(l)} = \begin{cases} a_{ij}, & \text{ha } i,j \in T_l, \\ a_{ii}, & \text{ha } i = j, \\ 0 & \text{egyébként}. \end{cases},$$

$$\widetilde{N}_l = \widetilde{M}_l - A,$$

$$\widetilde{e}_l = \left[\widetilde{e}_{ij}^{(l)}\right]_{i,j=1}^{n}, \quad E_{ii}^{(l)} = 0, \text{ha } i \notin T_l$$

előírás ($l = 1, 2, \ldots, L$) definiálja.

Egy nemszinguláris $A \in \mathbb{R}^{n\times n}$ mátrixot ***M-mátrixnak*** nevezünk, ha $a_{ij} \leq 0$ ($i \neq j$) és A^{-1} elemei nemnegatívak. Igaz a következő

17.12. tétel. *Tegyük fel, hogy $A \in \mathbb{R}^{n\times n}$ nemszinguláris M-mátrix, $\{M_i, N_i, E_i\}_{i=1}^{L}$ az A nemátfedő, $\left\{\widetilde{M}_i, \widetilde{N}_i, E_i\right\}_{i=1}^{L}$ pedig az A átfedő blokk Jacobi-multifelbontása, amelyekben az E_i súlymátrixok közösek. Ekkor igaz, hogy*

$$\rho\left(\widetilde{H}\right) \leq \rho(H) < 1,$$

ahol $H = \sum_{l=1}^{L} E_l M_l^{-1} N_l$ és $\widetilde{H} = \sum_{l=1}^{L} E_l \widetilde{M}_l^{-1} \widetilde{N}_l$.

Tehát mindkét eljárás konvergens és az átfedő eljárás konvergenciája nem lassúbb mint a nemátfedő eljárásé. A tétel igaz marad akkor is, ha a *blokk Jacobi-multifelbontások* helyett *blokk Gauss-Seidel típusú multifelbontásokat* használunk. Ekkor a fent definiált M_i és $\widetilde{M}_i$ mátrixok helyett az alsó háromszög részüket kell venni.

A multifelbontás algoritmusnak többlépéses és aszinkron változatai is ismertek.

17.2.3. Lineáris egyenletrendszerek hibaelemzése

A vizsgálatban a direkt és az inverz hibákat elemezzük. Az $Ax = b$ egyenletrendszer megoldásával kapcsolatban a következő jelöléseket és fogalmakat használjuk. Az elméleti megoldást x, a közelítő megoldásokat pedig $\hat{x}$ jelöli. A közelítő megoldás direkt hibája $\Delta x = \hat{x} - x$. Az $r = r(y) = Ay - b$ mennyiséget ***reziduális hibának*** nevezzük. Az elméleti megoldás esetén $r(x) = 0$, a közelítő megoldás esetén pedig

$$r(\hat{x}) = A\hat{x} - b = A(\hat{x} - x) = A\Delta x.$$

Az inverz hiba meghatározásához különféle modelleket használunk. A legáltalánosabb esetben feltesszük, hogy az $\hat{x}$ számított megoldás kielégíti az $\hat{A}\hat{x} = \hat{b}$ egyenletrendszert, ahol $\hat{A} = A + \Delta A$ és $\hat{b} = b + \Delta b$. A ΔA és Δb mennyiségeket inverz hibáknak nevezzük.

Meg kell különböztetnünk a probléma érzékenységét és a megoldó algoritmusok stabilitását. Egy adott ***probléma érzékenységén*** a megoldás változásának mértékét értjük a probléma (bemeneti) paramétereinek függvényében. Egy ***algoritmus érzékenységén*** vagy ***stabilitásán*** a számítási hibák végeredményre gyakorolt hatásának mértékét értjük. Egy problémát vagy algoritmust annál stabilabbnak tekintünk mennél kisebb a bemeneti paraméterek, ill. számítási hibák megoldásra (számított megoldásra) gyakorolt hatása. Az érzékenység,

ill. stabilitás fogalmának egyik jellemzési formája a korábban látott *kondíciószám*, amely az eltérések relatív hibáit hasonlítja össze.

Algoritmusok felhasználásának a következő általános elveit lehet megfogalmazni:

1. A gyakorlatban csak stabil (jól kondicionált) algoritmusokat használunk.
2. Instabil (inkorrekt kitűzésű vagy rosszul kondicionált) feladatot általános célú algoritmusokkal általában nem tudunk megoldani.

Érzékenységvizsgálat

Tegyük fel, hogy az $Ax = b$ egyenlet helyett a perturbált

$$A\hat{x} = b + \Delta b \tag{17.17}$$

egyenletrendszert oldjuk meg. Legyen $\hat{x} = x + \Delta x$ és vizsgáljuk a két megoldás eltérését.

17.13. tétel. *Ha A nemszinguláris és $b \neq 0$, akkor*

$$\frac{\|\Delta x\|}{\|x\|} \leq \operatorname{cond}(A)\frac{\|\Delta b\|}{\|b\|} = \operatorname{cond}(A)\frac{\|r(\hat{x})\|}{\|b\|}, \tag{17.18}$$

ahol $\operatorname{cond}(A) = \|A\|\left\|A^{-1}\right\|$ *az A mátrix ún. kondíciószáma.*

A tételből látható, hogy az A mátrix kondíciószáma erősen befolyásolhatja az $\hat{x}$ perturbált megoldás relatív hibáját. Egy rendszert *jól kondicionáltnak* nevezünk, ha $\operatorname{cond}(A)$ kicsi, és *rosszul kondicionáltnak* nevezünk, ha $\operatorname{cond}(A)$ nagy. Értelemszerűen a nagy és kicsi jelzők relatívak és környezetfüggők. A kondíciószám függ a normától. Ha a normától való függés lényeges, akkor ezt külön jelöljük. Ennek megfelelően például $\operatorname{cond}_\infty(A) = \|A\|_\infty \left\|A^{-1}\right\|_\infty$. A kondicionáltság egy lehetséges geometriai jellemzését adja a következő példa.

17.7. példa. Az

$$\begin{array}{rcrcl} 1000x_1 & + & 999x_2 & = & b_1 \\ 999x_1 & + & 998x_2 & = & b_2 \end{array}$$

egyenletrendszer rosszul kondicionált ($\operatorname{cond}_\infty(A) = 3.99 \times 10^6$). A két egyenes majdnem párhuzamos. Ezért, ha perturbáljuk a jobb oldalt, az új metszéspont messze lesz az előzőtől.

A most vizsgált modellben az inverz hiba Δb, a 17.13. tétel pedig a relatív direkt hibára ad becslést. Ez teljes összhangban van a hibaszámítási ökölszabállyal. A tételből látszik: az $\|r(\hat{x})\| / \|b\|$ relatív reziduális hiba kicsi voltából csak akkor következik, hogy az $\hat{x}$ perturbált megoldás relatív hibája is kicsi, ha A kondíciószáma kicsi.

17.8. példa. Tekintsük az $Ax = b$ egyenletrendszert, ahol

$$A = \begin{bmatrix} 1+\epsilon & 1 \\ 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad x = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Legyen $\hat{x} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$. Ekkor $r = \begin{bmatrix} 2\epsilon \\ 0 \end{bmatrix}$ és $\|r\|_\infty / \|b\|_\infty = 2\epsilon$, de $\|\hat{x} - x\|_\infty / \|x\|_\infty = 2$.

Tegyük most fel, hogy az $Ax = b$ egyenletrendszer helyett a perturbált

$$(A + \Delta A)\,\hat{x} = b \tag{17.19}$$

egyenletrendszert oldjuk meg. Igazolható, hogy ennél a modellnél több *inverz hiba* is létezik, ezek közül $\hat{x}, r(\hat{x}) \neq 0$ esetén a *minimális spektrálnormájú inverz hiba*: $\Delta A = -r(\hat{x})\,\hat{x}^T / \hat{x}^T\hat{x}$.

A következő tétel azt mondja ki, hogy kis relatív reziduális hiba esetén a relatív inverz hiba is kicsi.

17.14. tétel. *Tegyük fel, hogy $\hat{x} \neq 0$ az $Ax = b$ egyenletrendszer közelítő megoldása,* $\det(A) \neq 0$ *és* $b \neq 0$. *Ha* $\|r(\hat{x})\|_2 / \|b\|_2 = \alpha < 1$, *akkor a* $\Delta A = -r(\hat{x})\,\hat{x}^T / \hat{x}^T\hat{x}$ *mátrixra fennáll, hogy* $(A + \Delta A)\,\hat{x} = b$ *és* $\|\Delta A\|_2 / \|A\|_2 \leq \alpha/(1-\alpha)$.

Ha a relatív inverz hiba kicsi és A kondíciószáma kicsi, akkor a relatív reziduális hiba is kicsi. Erre mutat rá a következő tétel.

17.15. tétel. *Ha* $r(\hat{x}) = A\hat{x} - b$, $(A + \Delta A)\,\hat{x} = b$, $A \neq 0$, $b \neq 0$ *és* $\operatorname{cond}(A)\,\|\Delta A\| / \|A\| < 1$, *akkor*

$$\frac{\|r(\hat{x})\|}{\|b\|} \leq \frac{\operatorname{cond}(A)\frac{\|\Delta A\|}{\|A\|}}{1 - \operatorname{cond}(A)\frac{\|\Delta A\|}{\|A\|}}\,. \tag{17.20}$$

Ha A rosszul kondicionált, akkor a 17.15. tétel állítása nem teljesül.

17.9. példa. Legyen $A = \begin{bmatrix} 1+\epsilon & 1 \\ 1 & 1-\epsilon \end{bmatrix}$, $\Delta A = \begin{bmatrix} 0 & 0 \\ 0 & \epsilon^2 \end{bmatrix}$ és $b = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$, $(0 < \epsilon \ll 1)$. Ekkor $\operatorname{cond}_\infty(A) = (2+\epsilon)^2/\epsilon^2 \approx 4/\epsilon^2$ és $\|\Delta A\|_\infty / \|A\|_\infty = \epsilon^2/(2+\epsilon) \approx \epsilon^2/2$. Legyen most

$$\hat{x} = (A + \Delta A)^{-1} b = \frac{1}{\epsilon^3}\begin{bmatrix} 2 - \epsilon + \epsilon^2 \\ -2 - \epsilon \end{bmatrix} \approx \begin{bmatrix} 2/\epsilon^3 \\ -2/\epsilon^3 \end{bmatrix}.$$

Ekkor $r(\hat{x}) = A\hat{x} - b = \begin{bmatrix} 0 \\ 2/\epsilon + 1 \end{bmatrix}$. Ezért $\|r(\hat{x})\|_\infty / \|b\|_\infty = 2/\epsilon + 1$, ami nem kicsi.

A legáltalánosabb esetben az $Ax = b$ egyenlet helyett a perturbált

$$(A + \Delta A)\,\hat{x} = b + \Delta b \tag{17.21}$$

egyenletrendszert oldjuk meg. Igaz a következő

17.16. tétel. *Ha A nemszinguláris,* $\operatorname{cond}(A)\frac{\|\Delta A\|}{\|A\|} < 1$ *és* $b \neq 0$, *akkor*

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\operatorname{cond}(A)\left(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|}\right)}{1 - \operatorname{cond}(A)\frac{\|\Delta A\|}{\|A\|}}\,. \tag{17.22}$$

Fenti tételből következik a következő „ökölszabály”.

Ökölszabály. *Tegyük fel, hogy $Ax = b$. Ha A és b elemei s decimális jegyre pontosak és* $\operatorname{cond}(A) \sim 10^t$, *ahol $t < s$. Ekkor a számított megoldás kb. $s - t$ jegyre lesz pontos.*

A 17.16. tétel $\operatorname{cond}(A)\,\|\Delta A\| / \|A\| < 1$ feltételének jelentése szemléletes: azt biztosítja, hogy az $A + \Delta A$ mátrix ne legyen szinguláris. A $\operatorname{cond}(A)\,\|\Delta A\| / \|A\| < 1$ egyenlőtlenség

ugyanis ekvivalens a $\|\Delta A\| < \frac{1}{\|A^{-1}\|}$ feltétellel és az A nemszinguláris mátrix legközelebbi szinguláris mátrixtól való távolsága éppen $1/\left\|A^{-1}\right\|$. Ez alapján a kondíciószám egy újabb jellemzését adhatjuk:

$$\frac{1}{\operatorname{cond}(A)} = \min_{A+\Delta A \text{ szinguláris}} \frac{\|\Delta A\|}{\|A\|} . \tag{17.23}$$

Eszerint, ha egy mátrix rosszul kondicionált, akkor közel van egy szinguláris mátrixhoz. Megjegyezzük, hogy mátrixok kondíciószámát az $F(x) = A^{-1}x$ leképezés kondíciószámának felső becsléseként már korábban is megkaptuk.

Vezessük be a következő definíciót.

17.17. definíció. *Egy lineáris egyenletrendszer megoldó eljárást* ***gyengén stabilnak*** *nevezünk egy H mátrixosztályon, ha minden jól kondicionált $A \in H$ mátrix és minden b esetén az $Ax = b$ egyenletrendszer $\hat{x}$ számított megoldásának $\|\hat{x} - x\| / \|x\|$ relatív hibája kicsi.*

Ha a 17.13–17.16. tételeket összerakjuk, akkor a következő eredményt kapjuk.

17.18. tétel (Bunch). *Egy lineáris egyenletrendszer megoldó algoritmus gyengén stabil a H mátrixosztályon, ha minden jól kondicionált $A \in H$ mátrix és minden b esetén az $Ax = b$ egyenletrendszer $\hat{x}$ számított megoldására fennáll az alábbi feltételek bármelyike:*
(1) $\|\hat{x} - x\| / \|x\|$ kicsi;
(2) $\|r(\hat{x})\| / \|b\|$ kicsi;
(3) Létezik ΔA úgy, hogy $(A + \Delta A)\hat{x} = b$ és $\|\Delta A\| / \|A\|$ kicsi.

A 17.16. tétel becslését a gyakorlatban akkor tudjuk jól használni, ha ismert $\Delta b, \Delta A$ és $\operatorname{cond}(A)$ vagy ezek egy becslése. Ezek hiányában a közelítő megoldás hibáját csak utólagosan (a posteriori) lehet becsülni.

A következőkben komponensenkénti hibabecslésekkel foglalkozunk. Elsőként a közelítő megoldás abszolút hibájára adunk becslést az inverz hibák komponenseinek segítségével.

17.19. tétel (Bauer-Skeel). *Legyen $A \in \mathbb{R}^{n\times n}$ nemszinguláris és tegyük fel, hogy az $Ax = b$ egyenletrendszer $\hat{x}$ közelítő megoldása kielégíti az $(A + E)\hat{x} = b + e$ egyenletrendszert. Ha $S \in \mathbb{R}^{n\times n}$, $s \in \mathbb{R}^n$ és $\varepsilon > 0$ olyanok, hogy $S \geq 0$, $s \geq 0$, $|E| \leq \varepsilon S$, $|e| \leq \varepsilon s$, valamint $\varepsilon \left\|\left|A^{-1}\right| S\right\|_\infty < 1$, akkor*

$$\|\hat{x} - x\|_\infty \leq \frac{\varepsilon \left\|\left|A^{-1}\right| (S|x| + s)\right\|_\infty}{1 - \varepsilon \left\|\left|A^{-1}\right| S\right\|_\infty} . \tag{17.24}$$

Ha $e = 0$ $(s = 0)$, $S = |A|$ és

$$k_r(A) = \left\|\left|A^{-1}\right| |A|\right\|_\infty < 1, \tag{17.25}$$

akkor a

$$\|\hat{x} - x\|_\infty \leq \frac{\varepsilon k_r(A)}{1 - \varepsilon k_r(A)} \tag{17.26}$$

becslést kapjuk. A $k_r(A)$ mennyiséget ***Skeel-féle normának*** nevezzük, noha a szokásos értelemben nem norma. A Skeel-féle norma kielégíti a

$$k_r(A) \leq \operatorname{cond}_\infty(A) = \|A\|_\infty \left\|A^{-1}\right\|_\infty \tag{17.27}$$

egyenlőtlenséget. Tehát a fenti becslés nem rosszabb, mint a hagyományos kondíciószámot használó becslés. Az inverz hiba komponensenkénti becslését teszi lehetővé Oettli és Práger következő eredménye.

Legyenek adottak az $A, \delta A \in \mathbb{R}^{n\times n}$ mátrixok és a $b, \delta b \in \mathbb{R}^n$ vektorok. Tegyük fel, hogy $\delta A \geq 0$ és $\delta b \geq 0$. Legyen továbbá

$$D = \{\Delta A \in \mathbb{R}^{n\times n} : |\Delta A| \leq \delta A\}, \quad G = \{\Delta b \in \mathbb{R}^n : |\Delta b| \leq \delta b\} \ .$$

17.20. tétel (Oettli-Práger). *Egy $\hat{x}$ számított megoldás akkor és csak akkor megoldása egy $(A + \Delta A)\,\hat{x} = b + \Delta b$ perturbált egyenletrendszernek, ahol $\Delta A \in D$ és $\Delta b \in G$, ha*

$$|r\,(\hat{x})| = |A\hat{x} - b| \leq \delta A\,|\hat{x}| + \delta b. \tag{17.28}$$

A tétel alkalmazásához nem kell a kondíciószám ismerete. A gyakorlatban δA és δb elemei a gépi epszilonnal arányosak.

17.21. tétel (Wilkinson). *Az $Ax = b$ egyenletrendszer Gauss-módszerrel lebegőpontos aritmetikában kapott $\hat{x}$ közelítő megoldása kielégíti az*

$$(A + \Delta A)\,\hat{x} = b \tag{17.29}$$

egyenletrendszert, ahol

$$\|\Delta A\|_\infty \leq 8n^3\rho_n\,\|A\|_\infty\,u + O(u^2), \tag{17.30}$$

ρ_n a pivot elemek növekedési tényezője és u az egységnyi kerekítés mértéke.

Minthogy a gyakorlatban ρ_n kicsi, a

$$\frac{\|\Delta A\|_\infty}{\|A\|_\infty} \leq 8n^3\rho_n u + O(u^2)$$

relatív inverz hiba is az. Ezért a 17.18. tétel alapján a Gauss-elimináció gyengén stabil mind a teljes, mind pedig a parciális főelemválasztás esetén.

A Wilkinson tételből kapjuk, hogy

$$\mathrm{cond}_\infty(A)\frac{\|\Delta A\|_\infty}{\|A\|_\infty} \leq 8n^3\rho_n\mathrm{cond}_\infty\,(A)\,u + O\left(u^2\right) \ .$$

Kis kondíciószám esetén feltehetjük, hogy $1-\mathrm{cond}_\infty(A)\,\|\Delta A\|_\infty\,/\,\|A\|_\infty \approx 1$. A 17.21. és a 17.16. tételek felhasználásával ($\Delta b = 0$ eset) a direkt hibára az alábbi közelítő becslést kapjuk:

$$\frac{\|\Delta x\|_\infty}{\|x\|_\infty} \leq 8n^3\rho_n\mathrm{cond}_\infty\,(A)\,u \ . \tag{17.31}$$

Ez az ökölszabály helyességét támasztja alá a Gauss-módszer esetén.

17.10. példa. Tekintsük a következő példát, amelynek együtthatóit pontosan tudjuk ábrázolni:

$$\begin{aligned} 888445x_1 + 887112x_2 &= 1,\\ 887112x_1 + 885781x_2 &= 0\ . \end{aligned}$$

Itt $\mathrm{cond}(A)_\infty$ ugyan nagy, de $\mathrm{cond}_\infty(A)\|\Delta A\|_\infty/\|A\|_\infty$ elhanyagolható az 1 mellett. A feladat pontos megoldása $x_1 = 885781$, $x_2 = -887112$. A MATLAB által adott közelítő megoldás $\hat{x}_1 = 885827.23$,

$\hat{x}_2 = -887158.30$, amelynek relatív hibája

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} = 5.22 \times 10^{-5} .$$

Minthogy $s \approx 16$ és $\text{cond}(A)_\infty \approx 3.15 \times 10^{12}$, az eredmény lényegében megfelel a Wilkinson tételnek, ill. az ökölszabálynak. A Wilkinson tétel az inverz hiba mértékére a

$$\|\Delta A\|_\infty \leq 1.26 \times 10^{-8}$$

becslést adja. Az Oettli-Práger tételt a $\delta A = \epsilon_M |A|$ és $\delta b = \epsilon_M |b|$ választással alkalmazva kapjuk, hogy $|r(\hat{x})| \leq \delta A |\hat{x}| + \delta b$. Minthogy $|\delta A\|_\infty = 3.94 \times 10^{-10}$, ez jobb becslést ad a $\|\Delta A\|_\infty$ inverz hibára, mint a Wilkinson-féle eredmény.

Skálázás és prekondicionálás

Számos, gyakorlati alkalmazásban előforduló mátrix rosszul kondicionált, ha n rendszáma nagy. Ilyen például a

$$H_n = \left[\frac{1}{i + j - 1} \right]_{i,j=1}^{n} \tag{17.32}$$

Hilbert-mátrix, amelyre $\text{cond}_2(H_n) \approx e^{3.5n}$, ha $n \to \infty$. Léteznek olyan egész elemű $2n \times 2n$ mátrixok is, amelyek a szabványos IEEE754 lebegőpontos aritmetikában pontosan ábrázolhatók és amelyek kondíciószáma közelítőleg 4×10^{32n}.

A nagy kondíciószámú egyenletrendszerek megoldásának két fő módja ismeretes: többszörös pontosságú aritmetika használata vagy a kondíciószám csökkentése. A kondíciószám csökkentésének két formája is ismert.

1. Skálázás. Az $Ax = b$ egyenletrendszert helyettesítjük az

$$(RAC)\, y = (Rb) \tag{17.33}$$

egyenletrendszerrel, ahol R és C diagonális mátrixok.

Erre a skálázott egyenletrendszerre alkalmazzuk a Gauss-eliminációt. Ennek y megoldásával kapjuk vissza az $x = Cy$ megoldást. Ha az RAC mátrix kondíciószáma kisebb, akkor y hibája is vélhetően kisebb lesz, így összességében az x pontosabb közelítését is megkaphatjuk. Számos stratégia ismeretes az R és C skálázó mátrixok megválasztására. Az egyik legismertebb stratégia az ún. ***kiegyensúlyozás,*** amelynek eredményeként az RAC mátrix valamennyi sora és oszlopa valamilyen normában mérve közel ugyanakkora lesz. Például a

$$\hat{D} = \text{diag}\left(\frac{1}{\left\| a_1^T \right\|_2}, \ldots, \frac{1}{\left\| a_n^T \right\|_2} \right)$$

választás esetén, ahol a_i^T az A mátrix i-edik sorvektorát jelöli, a $\hat{D}A$ skálázott mátrix sorainak euklideszi normája azonosan 1 lesz. Erre a skálázásra fennáll, hogy

$$\text{cond}_2\left(\hat{D}A\right) \leq \sqrt{n} \min_{D \in D_+} \text{cond}_2(DA) ,$$

ahol $D_+ = \{\text{diag}(d_1, \ldots, d_n) \mid d_1, \ldots, d_n > 0\}$. Ez azt jelenti, hogy $\hat{D}$ közelítőleg optimálisan skálázza az A mátrix sorait.

Tekintettel arra, hogy a skálázás és a főelemkiválasztás együttes hatása esetenként igen rossz eredményekre vezet, az eljárást újabban ritkán használják. A következő példa egy ilyen esetet mutat be.

17.11. példa. Tekintsük az

$$A = \begin{bmatrix} \epsilon/2 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

mátrixot az $0 < \epsilon \ll 1$ esetben. Igazolható, hogy $\text{cond}_\infty(A) = 12$. Legyen

$$R = C = \begin{bmatrix} 2/\sqrt{\epsilon} & 0 & 0 \\ 0 & \sqrt{\epsilon}/2 & 0 \\ 0 & 0 & \sqrt{\epsilon}/2 \end{bmatrix}.$$

Ekkor a skálázott mátrix

$$RAR = \begin{bmatrix} 2 & 1 & 1 \\ 1 & \epsilon/4 & \epsilon/4 \\ 1 & \epsilon/4 & \epsilon/2 \end{bmatrix},$$

amelynek kondíciószáma $\text{cond}_\infty(RAR) = 32/\epsilon$. Ez kis ϵ esetén nagyon nagy érték. Tehát a skálázás elrontja a példabeli mátrixot.

2. Prekondicionálás. A prekondicionálás tulajdonképpen a skálázással rokon. A szimmetrikus és pozitív definit mátrixú $Ax = b$ egyenletrendszert átírjuk az ekvivalens

$$\tilde{A}x = (MA)\,x = Mb = \tilde{b} \tag{17.34}$$

alakba, ahol M olyan, hogy $\text{cond}\left(M^{-1}A\right)$ a lehető legkisebb és könnyű megoldani az $Mz = y$ alakú egyenletrendszereket.

A prekondicionálást általában iteratív módszerekkel együtt használjuk, módszerenként specializált formában.

Utólagos hibabecslések

A valamilyen módszerrel kapott közelítő megoldás hibájának utólagos becslése azért szükséges, hogy valamilyen támpontunk legyen az eredmény megbízhatóságáról. Az irodalomban számos módszer ismeretes. Ezek közül ismertetünk három $\Theta\left(n^2\right)$ flop költségű módszert. Az $\Theta\left(n^2\right)$ egyszeri költségű becslések ésszerű számítási többletet jelentenek az $\Theta\left(n^3\right)$ költségű direkt módszerekhez képest és elfogadhatók az iteratív módszerek $\Theta\left(n^2\right)$ iterációs lépésenkénti számítási költségével szemben is.

A direkt hiba becslése a reziduális segítségével

17.22. tétel (Auchmuty). *Jelölje $\hat{x}$ az $Ax = b$ egyenletrendszer valamilyen módon kiszámított közelítő megoldását. Ekkor igaz, hogy*

$$\|x - \hat{x}\|_2 = \frac{c\,\|r(\hat{x})\|_2^2}{\left\|A^T r(\hat{x})\right\|_2},$$

ahol a hibakonstansnak nevezett c-re $c \geq 1$ teljesül.

Megemlítjük, hogy a c hibakonstans az A-tól függ, de értékét nem befolyásolja az $\hat{x} - x$ hibavektor nagysága, csak az iránya. Igaz továbbá, hogy

$$C_2(A) = \frac{1}{2}\left(\mathrm{cond}_2(A) + \frac{1}{\mathrm{cond}_2(A)}\right) \leq \mathrm{cond}_2(A) \ .$$

A c hibakonstans a felső $C_2(A)$ értéket csak kivételes, de egzakt módon megadható esetekben éri el. A számítógépes tapasztalatok azt mutatják, hogy c az A mátrix rendjével (n) együtt lassan nő és jobban függ n-től, mint A kondíciószámától. A számítógépes tesztelések statisztikai feldolgozása alapján a

$$\|x - \hat{x}\|_2 \lessapprox 0.5 \dim(A) \|r(\hat{x})\|_2^2 / \left\|A^T r(\hat{x})\right\|_2 \tag{17.35}$$

becslés nagy tapasztalati valószínűséggel teljesül.

Az $\left\|A^{-1}\right\|$ LINPACK becslése

A LINPACK programcsomagban használják a következő eljárást $\left\|A^{-1}\right\|$ becslésére. Oldjuk meg rendre az $A^T y = d$, $Aw = y$ egyenletrendszereket. Az $\left\|A^{-1}\right\|$ becslése:

$$\left\|A^{-1}\right\| \approx \frac{\|w\|}{\|y\|} \quad \left(\leq \left\|A^{-1}\right\|\right) \ . \tag{17.36}$$

Minthogy

$$\frac{\|w\|}{\|y\|} = \frac{\left\|A^{-1}\left(A^{-T}d\right)\right\|}{\left\|A^{-T}d\right\|} \ ,$$

az eljárás felfogható a sajátértékszámítási fejezetben ismertetésre kerülő hatványmódszer alkalmazásának. A becslés az $1, 2$ és ∞ normák esetén alkalmazható. A d vektor javasolt komponensei ± 1 értékűek. Az előjeleket lehet véletlenszerűen választani. A LINPACK rendszerben az előjelek megválasztása egy adaptív stratégiával történik.

Ha az $Ax = b$ egyenletrendszert az LU-módszerrel oldottuk meg, akkor a további egyenletrendszerek megoldása $\Theta(n^2)$ flop rendszerenként, így a LINPACK becslő eljárás költsége kicsi marad. Az $\left\|A^{-1}\right\|$ becslés segítségével $\mathrm{cond}(A)$ és a közelítő megoldás hibája már könnyen becsülhető (vö. a 17.16. tétellel, vagy az ökölszabállyal). Megjegyezzük, hogy más hasonló eljárások is ismertek.

Az inverz hiba Oettli-Práger-féle becslése

Az inverz hiba becsléséhez az Oettli-Práger eredményt a következő formában szokás használni. Legyen $r(\hat{x}) = A\hat{x} - b$ a reziduális hiba, $E \in \mathbb{R}^{n\times n}$ és $f \in \mathbb{R}^n$ adottak úgy, hogy $E \geq 0$ és $f \geq 0$. Legyen

$$\omega = \max_i \frac{|r(\hat{x})_i|}{(E|\hat{x}| + f)_i},$$

ahol $0/0$-át 0-nak, $\rho/0$-át pedig ∞-nek definiáljuk, ha $\rho \neq 0$. Az $(y)_i$ jelölés az y vektor i-edik komponensét jelöli. Ha $\omega \neq \infty$, akkor van egy ΔA mátrix és egy Δb vektor, amelyekkel fennáll

$$|\Delta A| \leq \omega E, \quad |\Delta b| \leq \omega f$$

és

$$(A + \Delta A)\hat{x} = b + \Delta b.$$

Továbbá ω a legkisebb olyan szám, amelyre a fenti tulajdonságú ΔA és Δb létezik. Az ω mennyiség az E és f mennyiségekben kifejezett *relatív inverz hibát* méri. Ha egy adott E, f és $\hat{x}$ esetén ω kicsi, akkor a perturbált probléma is (és ennek megoldása is) közel van az eredetihez (és ennek megoldásához). A gyakorlatban az $E = |A|$, $f = |b|$ választást preferálják.

A közelítő megoldás iteratív javítása

Jelölje $\hat{x}$ az $Ax = b$ egyenletrendszer közelítő megoldását. Legyen $r(y) = Ay - b$ az y pontbeli reziduális hiba. Az $\hat{x}$ közelítő megoldás pontosságát a következő iteratív eljárással lehet javítani.

ITERATÍV-JAVÍTÁS$(A, \hat{x}, tol)$

1 $k \leftarrow 1$
2 $x_1 \leftarrow \hat{x}$
3 $\hat{d} \leftarrow \infty$
4 **while** $= \left\|\hat{d}\right\| / \|x_k\| > tol$
5 **do** $r \leftarrow Ax_k - b$
6 számítsuk ki az $Ad = r$ egyenletrendszer $\hat{d}$ közelítő megoldását LU-módszerrel
7 $x_{k+1} \leftarrow x_k - \hat{d}$
8 $k \leftarrow k + 1$
9 **return** x_k

Az eljárásnak különféle változatai ismertek. Az LU-módszer helyett más módszer is használható.

Legyen η az a legkisebb relatív inverz hibakorlát, amelyre

$$(A + \Delta A)\,\hat{x} = b + \Delta b, \quad |\Delta A| \le \eta\,|A|\,, \quad |\Delta b| \le \eta\,|b|\ .$$

Legyen továbbá

$$\sigma(A, x) = \max_k (|A|\,|x|)_k \,/\, \min_k (|A|\,|x|)_k\,, \quad \min_k (|A|\,|x|)_k > 0\ .$$

Igaz a következő

17.23. tétel (Skeel). *Ha* $k_r\left(A^{-1}\right)\sigma(A, x) \le c_1 < 1/\epsilon_M$, *akkor elég nagy* k *esetén fennáll, hogy*

$$(A + \Delta A)\,x_k = b + \Delta b, \quad |\Delta A| \le 4\eta\epsilon_M\,|A|\,, \quad |\Delta b| \le 4\eta\epsilon_M\,|b|\ . \tag{17.37}$$

Ez az eredmény gyakran az első iteráció után, a $k = 2$ értékre teljesül. Jankowski és Wozniakowski az iteratív javítást a Gauss-elimináció helyett tetszőleges olyan ϕ módszerre vizsgálták, amely az $Ax = b$ egyenletrendszer 1-nél kisebb relatív hibájú $\hat{x}$ közelítését állítja elő, azaz amelyre $\|\hat{x} - x\| \le q\,\|x\|$ $(q < 1)$. Igazolták, hogy az iteratív javítás még egyszeres pontosságú lebegőpontos aritmetikában is javítja a közelítő megoldás pontosságát és a ϕ módszert gyengén stabillá teszi.

Gyakorlatok

17.2-1. Bizonyítsuk be a 17.7. tételt.

17.2-2. Tekintsük az (i) $Ax = b$ és az (ii) $Bx = b$ egyenletrendszert, ahol

$$A = \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1/3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & -1/2 \\ 1/2 & 1/3 \end{bmatrix},$$

b pedig kétkomponensű vektor.

A két egyenlet közül melyik érzékenyebb a b perturbációjára? Az érzékenyebb egyenletnél a b -t hányszor kisebb *relatív hibával* kell ismernünk, hogy ugyanolyan pontossággal határozhassuk meg a megoldást, mint a másiknál?

17.2-3. Legyen $\chi = 3/2^{29}$ és $\zeta = 2^{14}$. Oldjuk meg az $Ax = b$ egyenletrendszert $\varepsilon = 10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}, 10^{-10}$ -re, ahol

$$A = \begin{bmatrix} \chi\zeta & -\zeta & \zeta \\ \zeta^{-1} & \zeta^{-1} & 0 \\ \zeta^{-1} & -\chi\zeta^{-1} & \zeta^{-1} \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1+\varepsilon \\ 1 \end{bmatrix}.$$

Mit tapasztalunk? Adjunk magyarázatot.

17.2-4. Legyen A 10×10-es mátrix és válasszuk prekondicionáló mátrixnak az A főátlójából és két mellékátlójából álló sávmátrixot. Mennyit javul a rendszer *kondíciószáma*, ha (i) A véletlen mátrix; (ii) A Hilbert-mátrix?

17.2-5. Legyen

$$A = \begin{bmatrix} 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \\ 1/4 & 1/5 & 1/6 \end{bmatrix},$$

a $b \in \mathbb{R}^3$ komponenseinek közös hibakorlátja pedig ε. Adjuk meg az $Ax = b$ egyenletrendszer $[x_1, x_2, x_3]^T$ megoldásának, valamint az $(x_1 + x_2 + x_3)$ összeg (minél élesebb) hibakorlátját.

17.2-6. Tekintsük az $Ax = b$ egyenletrendszert, amelynek egy közelítő megoldása legyen $\hat{x}$.

(i) Vezessünk le az $\hat{x}$ -ra hibakorlátot, ha az $(A + E)\hat{x} = b$ pontosan teljesül és sem A, sem $A + E$ nemszinguláris.

(ii) Adjunk (ha lehetséges) A elemeire *relatív hibakorlátot*, melyen belül maradva nem változik a megoldás egyik komponensének sem az egész része, ha

$$A = \begin{bmatrix} 10 & 7 & 8 \\ 7 & 5 & 6 \\ 8 & 6 & 10 \end{bmatrix}, \quad b = \begin{bmatrix} 25 \\ 18 \\ 24 \end{bmatrix}.$$

17.3. Sajátértékszámítás

A mátrixok sajátérték-feladatának megfogalmazásához szükségünk van a *komplex elemű mátrixok* és *vektorok* bevezetésére. A komplex elemű n dimenziós oszlopvektorok halmazát $\mathbf{C}^n$-nel jelöljük. Hasonlóképpen az $m \times n$ típusú komplex elemű mátrixok halmazát $\mathbf{C}^{m\times n}$ jelöli. Nyilvánvalóan fennáll, hogy $\mathbb{R}^n \subset \mathbf{C}^n$ és $\mathbb{R}^{m\times n} \subset \mathbf{C}^{m\times n}$. A valós elemű vektorokra és mátrixokra bevezetett műveletek és a determináns értelemszerűen ugyanazok maradnak a komplex esetben is.

17.24. definíció. *Legyen $A \in \mathbf{C}^{n\times n}$ tetszőleges mátrix. A $\lambda \in \mathbf{C}$ számot az A mátrix* ***sajátértékének****, az $x \in \mathbf{C}^n$ $(x \neq 0)$ vektort pedig a λ sajátértékhez tartozó (jobb oldali)* ***sajátvektornak*** *nevezzük, ha*

$$Ax = \lambda x \ . \tag{17.38}$$

A sajátvektor egy olyan vektor, amelyet az $x \to Ax$ leképezés a saját hatásvonalán hagy (irányítás, nagyság változhat). A sajátérték-feladat megoldása a sajátértékek és a hozzájuk tartozó sajátvektorok meghatározását jelenti.

Az $Ax = \lambda x$ egyenletrendszer átrendezéssel az ekvivalens $(A - \lambda I)x = 0$ alakra hozható (I a megfelelő méretű egységmátrix). Ennek a homogén egyenletrendszernek akkor és csak akkor van nullától különböző megoldása, ha

$$\phi(\lambda) = \det(A - \lambda I) = \det\left(\begin{bmatrix} a_{11} - \lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} - \lambda & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} - \lambda \end{bmatrix}\right) = 0 \ . \tag{17.39}$$

A (17.39) egyenletet az A mátrix ***karakterisztikus egyenletének*** nevezzük, melynek gyökei a mátrix sajátértékei. A determinánst kifejtve a λ változó n-ed fokú polinomját, azaz a

$$\phi(\lambda) = (-1)^n(\lambda^n - p_1\lambda^{n-1} - \dots - p_{n-1}\lambda - p_n)$$

karakterisztikus polinomot kapjuk. Az algebra alaptétele miatt az $A \in \mathbb{R}^n$ mátrixnak, a multiplicitásokat is beleszámítva, pontosan n sajátértéke van, melyek között lehetnek valósak és komplex konjugált párok. Valós mátrixok komplex sajátértékeit és sajátvektorait valós aritmetika használata esetén csak speciális technikákkal kaphatjuk meg.

Ha $t \neq 0$, akkor egy x sajátvektor t-szerese is sajátvektor. Egy λ_k sajátértékhez tartozó lineárisan független sajátvektorok száma legfeljebb annyi, mint λ_k multiplicitása a (17.39) egyenletben. A különböző sajátértékekhez tartozó sajátvektorok lineárisan függetlenek.

A sajátértékek nagyságát és geometriai elhelyezkedését jellemzi a következő két tétel.

17.25. tétel. *Legyen λ az A mátrix tetszőleges sajátértéke. Tetszőleges indukált mátrixnormában fennáll, hogy $|\lambda| \leq \|A\|$.*

17.26. tétel (Gersgorin). *Legyen $A \in \mathbf{C}^{n\times n}$,*

$$r_i = \sum_{j=1, j\neq i}^{n} |a_{ij}| \quad (i = 1, \dots, n)$$

és

$$D_i = \{z \in \mathbf{C} \,|\, |z - a_{ii}| \leq r_i\} \quad (i = 1, \dots, n) \ .$$

Ekkor az A mátrix minden λ sajátértékére fennáll, hogy $\lambda \in \cup_{i=1}^{n} D_i$.

Bizonyos mátrixok esetén a (17.39) megoldása egyszerű. Például, ha A háromszögmátrix, akkor kiolvasható, hogy a sajátértékek a főátlóbeli elemek. A legtöbb esetben azonban az összes sajátérték és sajátvektor meghatározása elvileg is komoly gondokat okoz. Ezért is van gyakorlati jelentősége az olyan transzformációknak, melyek a sajátértékeket változatlanul hagyják. Mint később látni fogjuk, az ilyen transzformációk sorozatával nyert új mátrix sajátértékfeladata egyszerűbben kezelhető.

17.27. definíció. *Az $n \times n$ méretű A és B mátrix hasonló, ha van egy T mátrix úgy, hogy $B = T^{-1}AT$. Az $A \to T^{-1}AT$ leképezést* ***hasonlósági transzformációnak*** *nevezzük.*

17.28. tétel. *Tegyük fel, hogy az $n \times n$ méretű T mátrixra $\det(T) \neq 0$ teljesül. Ekkor az A és $B = T^{-1}AT$ mátrix sajátértékei megegyeznek. Ha x az A sajátvektora, akkor $y = T^{-1}x$ a B sajátvektora.*

A tétel tartalma az, hogy hasonló mátrixok sajátértékei megegyeznek.

A sajátérték feladat nehézségét tovább fokozza az a tény, hogy a sajátértékek és sajátvektorok a mátrixelemek megváltozására – ezért a számítás közbeni kerekítési hibákra is – nagyon érzékenyek. Az eredeti A mátrix és a perturbált $A+\delta A$ mátrix sajátértékei jelentősen eltérhetnek egymástól és a sajátértékek multiplicitása is megváltozhat. A sajátérték-feladat érzékenységét a következő tételekkel és példákkal jellemezhetjük.

17.29. tétel (Ostrowski-Elsner). *Az $A \in \mathbf{C}^{n\times n}$ mátrix minden λ_i sajátértékéhez létezik a perturbált $A + \delta A$ mátrix egy olyan μ_k sajátértéke, hogy*

$$|\lambda_i - \mu_k| \leq (2n-1)\left(\|A\|_2 + \|A+\delta A\|_2\right)^{1-\frac{1}{n}} \|\delta A\|_2^{\frac{1}{n}} \ .$$

A tétel azt mutatja, hogy a sajátértékek folytonosan változnak és azt, hogy a megváltozás mértéke arányos a δA perturbáció mértékének n-edik gyökével.

17.12. példa. Vizsgáljuk meg egy μ sajátértékhez tartozó $r \times r$ méretű *Jordan-mátrix* alábbi perturbációját:

$$\begin{bmatrix} \mu & 1 & 0 & \dots & 0 \\ 0 & \mu & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & & \ddots & \mu & 1 \\ \epsilon & 0 & \dots & 0 & \mu \end{bmatrix} .$$

A perturbált mátrixhoz tartozó karakterisztikus egyenlet $(\lambda-\mu)^r = \epsilon$, ahonnan az eredeti *Jordan-mátrix* r-szeres μ sajátértéke helyett az r különböző

$$\lambda_s = \mu + \epsilon^{1/r}\left(\cos(2s\pi/r) + i\sin(2s\pi/r)\right) \quad (s = 0, \dots, r-1)$$

sajátértéket kapjuk. A sajátértékek megváltozásának mértéke $\epsilon^{1/r}$, amely megfelel a 17.29. tétel állításának. Ha például $|\mu| \approx 1$, $r = 16$ és $\epsilon = \epsilon_M \approx 2.2204 \times 10^{-16}$ értékeket vesszük, akkor a sajátértékek eltérése ≈ 0.1051. Ez pedig a mátrix perturbációjához képest a sajátértékek egy igen jelentős mértékű megváltozását jelenti.

Speciális tulajdonságú mátrixok és perturbációk esetén a sajátértékek megváltozása az Ostrowski-Elsner tételben látottnál jóval kisebb is lehet.

17.30. tétel (Bauer-Fike). *Legyen $A \in \mathbf{C}^{n\times n}$ diagonalizálható mátrix, azaz létezzen olyan X mátrix, hogy $X^{-1}AX = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$. Jelölje μ az $A + \delta A$ mátrix sajátértékét. Ekkor*

$$\min_{1\leq i\leq n} |\lambda_i - \mu| \leq \operatorname{cond}_2(X)\,\|\delta A\|_2 \ . \tag{17.40}$$

Tehát diagonalizálható mátrix perturbációja esetén a sajátértékek megváltozásának mértéke arányos az általában ismeretlen X mátrix kondíciószámával és $\|\delta A\|_2$-val. Ez aszimptotikusan lényegesen jobb eredmény, mint az Ostrowski-Elsner tétel, azonban ne felejtsük, hogy $\mathrm{cond}_2(X)$ igen nagy is lehet.

Az A mátrix sajátértékei folytonos függvényei a mátrix elemeinek. Ez a normált sajátvektorokra is igaz, ha a sajátértékek egyszeresek. A következő példa is mutatja, hogy az utóbbi állítás többszörös sajátértékek esetén már nem igaz.

17.13. példa. Legyen

$$A(t) = \begin{bmatrix} 1 + t\cos(2/t) & -t\sin(2/t) \\ -t\sin(2/t) & 1 - t\cos(2/t) \end{bmatrix} \quad (t \neq 0) \ .$$

Az $A(t)$ mátrix sajátértékei $\lambda_1 = 1 + t$ és $\lambda_2 = 1 - t$. A λ_1 sajátértékhez tartozó sajátvektor $[\sin(1/t), \cos(1/t)]^T$, a λ_2-höz tartozó pedig $[\cos(1/t), -\sin(1/t)]^T$. Ha $t \to 0$, akkor

$$A(t) \to I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \lambda_1, \lambda_2 \to 1 \ ,$$

míg a sajátvektor sorozatnak nincs határértéke.

A következő alfejezetben a feladat közelítő megoldásával foglalkozunk. Sajnos, a közelítések jóságát igen nehéz megítélni, ugyanis abból, hogy a definíciós egyenlet milyen pontossággal teljesül, egyéb információk hiányában általában semmire nem következtethetünk.

17.14. példa. Tekintsük az

$$A(\epsilon) = \begin{bmatrix} 1 & 1 \\ \epsilon & 1 \end{bmatrix}$$

mátrixot, ahol $\epsilon \approx 0$ kicsi. Az $A(\epsilon)$ mátrix sajátértékei $1 \pm \sqrt{\epsilon}$, sajátvektorai $[1, \pm\sqrt{\epsilon}]^T$. Legyen a sajátérték közelítése $\mu = 1$, a sajátvektoré pedig $x = [1, 0]^T$. Ekkor

$$\|Ax - \mu x\|_2 = \left\| \begin{bmatrix} 0 \\ \epsilon \end{bmatrix} \right\|_2 = \epsilon.$$

Ha most például $\epsilon = 10^{-10}$, akkor a reziduális öt nagyságrenddel alulbecsüli a tényleges 10^{-5} hibát.

17.31. megjegyzés. *Mátrixok kondíciószámához hasonlóan* ***sajátértékek kondíciószáma*** *is bevezethető, ami egyszeres sajátérték esetén*

$$\nu(\lambda_1) \approx \frac{\|x\|_2 \|y\|_2}{|x^T y|} \ ,$$

ahol x és y a jobb, illetve bal oldali sajátvektor (komplex sajátérték esetén x^T helyett az ún. hermitikus transzponáltat vesszük). Többszörös sajátérték kondíciószáma nem véges. Itt is arról van szó, hogy a sajátérték közelítés relatív hibája a relatív reziduális hiba kondíciószámszorosa.

17.3.1. A sajátérték feladat iteratív megoldása

Csak valós elemű mátrixok valós sajátértékeit és sajátvektorait vizsgáljuk. A módszerek értelemszerű módosításokkal kiterjeszthetők a komplex esetre is.

A hatványmódszer

A von Mieses-től származó módszer alapgondolata a következő. Tegyük fel, hogy az $n \times n$ méretű A valós mátrixnak pontosan n különböző valós sajátértéke van. Ekkor a $\lambda_1, \ldots, \lambda_n$ sajátértékekhez tartozó $x_1, \ldots, x_n$ sajátvektorok lineárisan függetlenek. Tegyük fel, hogy a sajátértékek kielégítik a

$$|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$$

feltételt és legyen $v^{(0)} \in \mathbb{R}^n$ adott. A sajátvektorok lineáris függetlensége miatt $v^{(0)}$ egyértelműen előáll $v^{(0)} = \alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n$ alakban. Tegyük fel, hogy $\alpha_1 \neq 0$. Képezzük a $v^{(k)} = Av^{(k-1)} = A^k v^{(0)}$ $(k = 1, 2, \ldots)$ sorozatot. A kiinduló feltevések miatt

$$\begin{aligned} v^{(k)} = Av^{(k-1)} &= A(\alpha_1 \lambda_1^{k-1} x_1 + \alpha_2 \lambda_2^{k-1} x_2 + \cdots + \alpha_n \lambda_n^{k-1} x_n) \\ &= \alpha_1 \lambda_1^k x_1 + \alpha_2 \lambda_2^k x_2 + \cdots + \alpha_n \lambda_n^k x_n \\ &= \lambda_1^k \left(\alpha_1 x_1 + \alpha_2 \left(\tfrac{\lambda_2}{\lambda_1}\right)^k x_2 + \cdots + \alpha_n \left(\tfrac{\lambda_n}{\lambda_1}\right)^k x_n \right). \end{aligned}$$

Legyen $y \in \mathbb{R}^n$ tetszőleges olyan vektor, amelyre $y^T x_1 \neq 0$. Ekkor

$$\frac{y^T A v^{(k)}}{y^T v^{(k)}} = \frac{y^T v^{(k+1)}}{y^T v^{(k)}} = \frac{\lambda_1^{k+1} \left(\alpha_1 y^T x_1 + \sum_{i=2}^n \alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^{k+1} y^T x_i \right)}{\lambda_1^k \left(\alpha_1 y^T x_1 + \sum_{i=2}^n \alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^k y^T x_i \right)} \to \lambda_1 \ .$$

A $v^{(0)} \in \mathbb{R}^n$ kezdővektor felhasználásával a hatványmódszer a következő:

HATVÁNYMÓDSZER$(A, v^{(0)})$

```
1  k ← 0
2  while kilépési feltétel = HAMIS
3        do k ← k + 1
4           z^(k) ← A v^(k−1)
5           válasszunk y vektort úgy, hogy y^T v^(k−1) ≠ 0 teljesüljön
6           γ_k ← y^T z^(k) / y^T v^(k−1)
7           v^(k) ← z^(k) / ||z^(k)||_∞
8  return γ_k, v^(k)
```

A fentiek alapján fennáll, hogy

$$v^{(k)} \to x_1, \quad \gamma_k \to \lambda_1 \ .$$

A $v^{(k)} \to x_1$ konvergencián itt azt értjük, hogy $v^{(k)}$ hatásvonala tart x_1 hatásvonalához. Az y vektort általában egységvektornak választjuk úgy, hogy ha $\left|v_i^{(k)}\right| = \left\|v^{(k)}\right\|_\infty$, akkor legyen $y = e_i$. Ha az $y = v^{(k-1)}$ választást használjuk, akkor $\gamma_k = v^{(k-1)T} A v^{(k-1)} / \left(v^{(k-1)T} v^{(k-1)}\right)$ a $v^{(k-1)}$ vektorhoz tartozó, ún. $\mathbb{R}\left(v^{(k-1)}\right)$ *Rayleigh-hányadossal* azonos. Ez a választás adja a λ_1 sajátérték minimális reziduális hibájú közelítését (ami a 17.14. példa alapján persze nem jelenti azt, hogy a legjobb választás lenne).

Az eljárás konvergencia sebessége a $|\lambda_2/\lambda_1|$ nagyságától függ. A módszer erősen érzékeny a $v^{(0)}$ kezdővektor megválasztására is. Ha $\alpha_1 = 0$, akkor az eljárás nem konvergál a λ_1 domináns sajátértékhez. Bizonyos mátrixosztályok esetén igazolták, hogy véletlenül választott $v^{(0)}$ kezdővektorok esetén 1 valószínűséggel konvergál az eljárás. Komplex sajátértékek,

illetve többszörös λ_1 esetén az eljárást módosítani kell. Az eljárás konvergenciáját gyorsítani lehet, ha az $A-\sigma I$ ún. eltolt mátrixra alkalmazzuk, ahol σ alkalmasan megválasztott szám. Az $A-\sigma I$ mátrix sajátértékei ui. $\lambda_1-\sigma, \lambda_2-\sigma, \ldots, \lambda_n-\sigma$ és a megfelelő konvergencia tényező: $|\lambda_2-\sigma| / |\lambda_1-\sigma|$. Ez utóbbi σ ügyes megválasztásával kisebbé tehető, mint $|\lambda_2/\lambda_1|$.

A hatványmódszert az

$$\|E_k\|_2 = \frac{\|r_k\|_2}{\left\|v^{(k)}\right\|_2} = \frac{\left\|Av^{(k)} - \gamma_k v^{(k)}\right\|_2}{\left\|v^{(k)}\right\|_2} \leq \epsilon$$

kilépési feltétellel szokás leállítani. Ha a hatványmódszert szimultán alkalmazzuk az A^T mátrixra és $w_k = \left(A^T\right)^k w_0$, akkor

$$\nu(\lambda_1) \approx \frac{\left\|w^{(k)}\right\|_2 \left\|v^{(k)}\right\|_2}{\left|w_k^T v_k\right|}$$

a λ_1 kondíciószámának (lásd a 17.31. megjegyzést) egy becslését adja. Ekkor értelemszerűen a

$$\nu(\lambda_1)\,\|E_k\|_2 \leq \epsilon$$

kilépési feltételt használjuk.

A hatványmódszert, amely igen előnyös lehet nagyméretű ritka mátrixok esetén, leginkább a legnagyobb, ill. a legkisebb abszolút értékű sajátértékek meghatározására használjuk. Ez utóbbi a következőképpen történhet. Az A^{-1} sajátértékei: $1/\lambda_1, \ldots, 1/\lambda_n$. Ezek közül a legnagyobb abszolút értékű sajátérték $1/\lambda_n$ lesz. Ezt az értéket tehát a hatványmódszernek A^{-1}-re való alkalmazásával megkaphatjuk, mégpedig úgy, hogy algoritmusának 4-edik sorát a következő utasításra cseréljük:

oldjuk meg az $Az^{(k)} = v^{(k-1)}$ egyenletrendszert $z^{(k)}$-ra.

Az így módosított algoritmust nevezzük ***inverz hatványmódszernek***. Nyilvánvaló, hogy alkalmas feltételek mellett $\gamma_k \to 1/\lambda_n$ és $v^{(k)} \to x_n$. Az $Az^{(k)} = v^{(k-1)}$ egyenletrendszer megoldásához az LU-módszert célszerű használni. Az algoritmus szembetűnő előnye, hogy nem kell A^{-1}-et meghatározni.

Ha az inverz hatványmódszert az eltolt $A-\mu I$ mátrixra alkalmazzuk, akkor $(A-\mu I)^{-1}$ sajátértékei $(\lambda_i-\mu)^{-1}$. Ha μ közelít, mondjuk λ_t-hez, akkor $\lambda_i-\mu \to \lambda_i-\lambda_t$. Ezért az eltolt mátrix sajátértékeire teljesül, hogy

$$|\lambda_t-\mu|^{-1} > |\lambda_i-\mu|^{-1} \qquad (i \neq t)\ .$$

A konvergencia sebességét pedig a

$$q = |\lambda_t-\mu| / \{\max|\lambda_i-\mu|\}$$

hányados határozza meg. Ha μ elég közel van λ_t-hez, akkor q kicsinysége miatt az inverz hatványiteráció rendkívül sebesen fog konvergálni. Ezt a tulajdonságot használhatjuk ki közelítő sajátvektorok meghatározásánál, ha egy sajátérték μ közelítése ismert. Ekkor feltéve, hogy $\det(A-\mu I) \neq 0$, az $A-\mu I$ mátrixra alkalmazzuk az inverz hatványmódszert. Annak ellenére, hogy $A-\mu I$ közel szinguláris mátrix és ezért $(A-\mu I)\,z^{(k)} = v^{(k)}$ nem oldható meg nagy pontossággal, az eljárás sok esetben igen jó sajátvektor közelítést ad.

Végül megjegyezzük, hogy rangszámcsökkentő eljárásokkal és egyéb módosításokkal a Mieses-eljárás alkalmassá tehető az összes sajátérték-sajátvektor meghatározására is.

Ortogonalizálási eljárások
Szükségünk van a következő definícióra és tételre:

17.32. definíció. *Egy $Q \in \mathbb{R}^{n\times n}$ mátrix* ***ortogonális,*** *ha $Q^T Q = I$.*

17.33. tétel (*QR*-felbontás). *Minden $A \in \mathbb{R}^{n\times m}$ mátrix, amelynek oszlopvektorai lineárisan függetlenek, felbontható $A = QR$ alakban, ahol Q ortogonális mátrix és R felső háromszögmátrix.*

A *QR*-felbontást az *LU*-felbontáshoz hasonlóan felhasználhatjuk lineáris egyenletrendszer megoldására is. Ha ismert az A nemszinguláris mátrix *QR*-felbontása, akkor $Ax = QRx = b \Leftrightarrow Rx = Q^T b$ miatt csak egy a felső háromszögmátrixú egyenletrendszert kell megoldani.

Egy mátrix *QR*-felbontására több módszer is létezik. A gyakorlatban a Givens- és Householder-módszereket , valamint az MGS-módszert használják.

Az MGS- vagy módosított Gram–Schmidt-eljárás az önmagában is rendkívül fontos klasszikus Gram–Schmidt (CGS) ortogonalizációs eljárás numerikusan stabilizált, ekvivalens változata. Maga a feladat: az $a_1, \ldots, a_m \in \mathbf{R}^n$ $(m \le n)$, lineárisan független vektorok által kifeszített $L\{a_1, \ldots, a_m\} = \left\{\sum_{j=1}^m \lambda_j a_j \mid \lambda_j \in \mathbb{R},\ j = 1, \ldots, m\right\}$ lineáris altérnek keressük egy *ortonormált* új *bázisát*, azaz olyan lineárisan független $q_1, \ldots, q_m$ vektorokat, amelyekre fennáll:

$$q_i^T q_j = 0 \ (i \neq j), \|q_i\|_2 = 1 \ (i = 1, \ldots, m)$$

és

$$L\{a_1, \ldots, a_m\} = L\{q_1, \ldots, q_m\} \ .$$

A $\{q_i\}_{i=1}^m$ vektorrendszert *ortonormált* rendszernek mondjuk. A Gram-Schmidt eljárás alapgondolata a következő:

Legyen $r_{11} = \|a_1\|_2$ és $q_1 = a_1/r_{11}$. Tegyük fel, hogy a $q_1, \ldots, q_{k-1}$ vektorokat már előállítottuk. Keressük a $\tilde{q}_k$ vektort $\tilde{q}_k = a_k - \sum_{j=1}^{k-1} r_{jk} q_j$ alakban úgy, hogy $\tilde{q}_k \perp q_i$, azaz $\tilde{q}_k^T q_i = a_k^T q_i - \sum_{j=1}^{k-1} r_{jk} q_j^T q_i = 0 \quad (i = 1, \ldots, k-1)$ teljesüljön. Kihasználva, hogy a $q_1, \ldots, q_{k-1}$ ortonormált rendszer, kapjuk az $r_{ik} = a_k^T q_i \quad (i = 1, \ldots, k-1)$ eredményt. Végül normáljunk: $q_k = \tilde{q}_k / \|\tilde{q}_k\|_2$.

Az $a_1, \ldots, a_m \in \mathbb{R}^n$ lineárisan független vektorokra $(m \le n)$ tehát az eljárást a következő formában algoritmizálhatjuk.

CGS-KLASSZIKUS-GRAM–SCHMIDT-ORTOGONALIZÁCIÓ$(m, a_1, \ldots, a_m)$

1 **for** $k \leftarrow 1$ **to** m
2 **do for** $i \leftarrow 1$ **to** $k - 1$
3 **do** $r_{ik} \leftarrow a_k^T a_i$
4 $a_k \leftarrow a_k - r_{ik} a_i$
5 $r_{kk} \leftarrow \|a_k\|_2$
6 $a_k \leftarrow a_k / r_{kk}$
7 **return** $a_1, \ldots, a_m$

Az eljárás felülírja az a_i vektorokat az ortonormált q_i vektorokkal. A QR-felbontással való kapcsolatot az $a_k = \sum_{j=1}^{k-1} r_{jk} q_j + r_{kk} q_k$ összefüggés adja meg. Ugyanis fennáll, hogy

$$\begin{aligned} a_1 &= q_1 r_{11} \\ a_2 &= q_1 r_{12} + q_2 r_{22} \\ a_3 &= q_1 r_{13} + q_2 r_{23} + q_3 r_{33} \\ &\vdots \\ a_m &= q_1 r_{1m} + q_2 r_{2m} + \ldots + q_m r_{mm} \end{aligned} \quad .$$

Másképpen

$$A = [a_1, \ldots, a_m] = \underbrace{[q_1, \ldots, q_m]}_{} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & \ldots & r_{1m} \\ 0 & r_{22} & r_{23} & \ldots & r_{2m} \\ 0 & 0 & r_{33} & \ldots & r_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & r_{mm} \end{bmatrix}}_{} = QR\,.$$

A numerikusan stabilizált MGS módszer a következőképpen adható meg.

CGS-MÓDOSÍTOTT-GRAM–SCHMIDT-ORTOGONALIZÁCIÓ$(m, a_1, \ldots, a_m)$

```
1  for k ← 1 to m
2      do r_kk ← ||a_k||_2
3         a_k ← a_k / r_kk
4         for j ← k + 1 to m
5             do r_kj ← a_j^T a_k
6                a_j ← a_j − r_kj a_k
7  return a_1, ..., a_m
```

Az eljárás felülírja az a_i vektorokat az ortonormált q_i vektorokkal. Az MGS eljárás ekvivalens a CGS eljárással. Ugyanakkor numerikusan lényegesen stabilabb. Björck igazolta, hogy $m = n$ esetén a számított $\hat{Q}$ mátrixra fennáll, hogy

$$\hat{Q}^T \hat{Q} = I + E, \qquad \|E\|_2 \cong \mathrm{cond}\,(A)\, u,$$

ahol u az egységnyi kerekítés mértéke.

A *QR*-módszer

A ma használt legfontosabb általános eljárás az összes sajátérték meghatározására. Megmutatható, hogy a hatványmódszer általánosítása. Alapgondolata: az $A_1 = A$-ból indulva ortogonális hasonlósági transzformációkkal állítsunk elő olyan $A_{k+1} = Q_k^{-1} A_k Q_k = Q_k^T A_k Q_k$ sorozatot, melynek alsó háromszögrésze diagonálmátrixhoz konvergál; főátlóbeli elemei tehát az A sajátértékeihez. A QR-módszernek nevezett eljárásban Q_k az $A_k = Q_k R_k$-felbontásában szereplő ortogonális tényező. Ekkor $A_{k+1} = Q_k^T (Q_k R_k) Q_k = R_k Q_k$.

QR-MÓDSZER(A)

```
1  k ← 1
2  A_1 ← A
3  while kilépési feltétel = HAMIS
4        do számítsuk ki az A_k = Q_k R_k felbontást
5           A_{k+1} ← R_k Q_k
6           k ← k + 1
7  return A_k
```

Igaz a következő tétel.

17.34. tétel (Parlett). *Ha az A mátrix diagonalizálható, továbbá sajátértékeire fennáll*

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n| > 0$$

és az $X^{-1}AX = diag(\lambda_1, \lambda_2, \ldots, \lambda_n)$ hasonlósági transzformáció X mátrixának létezik LU-felbontása, akkor az A_k mátrixok alsó háromszög része konvergál egy diagonális mátrixhoz, amelynek diagonális elemei az A sajátértékei lesznek.

Az A_k mátrixok felső része nem feltétlenül konvergál egy meghatározott mátrixhoz. Ha az A mátrixnak p számú azonos abszolút értékű sajátértéke van, akkor az A_k mátrixok alakja bizonyos feltételek mellett a

$$\begin{bmatrix} \times & & & & & & & & \times \\ 0 & \ddots & & & & & & & \\ & & \times & & & & & & \\ 0 & & 0 & * & \cdots & * & & & \\ & & & \vdots & & \vdots & & & \\ & & & * & \cdots & * & & & \\ 0 & & & & & 0 & \times & & \\ & & & & & & & \ddots & \\ 0 & & & & & & & 0 & \times \end{bmatrix} \tag{17.41}$$

alakhoz közelít, ahol a $*$ elemekkel jelölt részmátrix elemei ugyan nem konvergálnak, de sajátértékei igen. Ezt a részmátrixot lehet azonosítani és alkalmas módon kezelni. Ha valós mátrixunk van, akkor a karakterisztikus egyenletnek valós vagy komplex konjugált gyökei lehetnek. Komplex konjugált gyökpárok esetén p legalább kettő. Tehát az A_k sorozat a most vázolt jelenséget fogja mutatni.

A QR-felbontás nagyon számításigényes, költsége $\Theta(n^3)$ flop. Ugyanakkor a QR-módszert rendkívül gazdaságosan lehet alkalmazni, ha a kiinduló A mátrix felső Hessenberg-alakú.

17.35. definíció. *Egy $A \in \mathbb{R}^{n\times n}$ mátrix* ***felső Hessenberg-alakú,*** *ha*

$$A = \begin{bmatrix} a_{11} & & & \dots & & a_{1n} \\ a_{21} & & & & & \\ 0 & a_{32} & & & & \vdots \\ \vdots & 0 & \ddots & & & \\ & & \ddots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & & \dots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}.$$

Az A mátrixból hozzá hasonló, de már Hessenberg-alakú mátrixot többféleképpen is előállíthatunk. Az egyik legolcsóbb, kb. $5/6n^3$ flop költségű eljárás a Gauss-elimináción alapul. Mivel egy felső Hessenberg-alakú mátrix QR-felbontása $\Theta(n^2)$ flop számítási költséget igényel és a következő tétel biztosítja, hogy ha A felső Hessenberg-alakú, akkor valamennyi A_k is az, érdemes az algoritmus első lépéseként a felső Hessenberg-alakra transzformálást elvégezni.

17.36. tétel. *Ha A felső Hessenberg-alakú és $A = QR$, akkor RQ is felső Hessenberg-alakú.*

A QR-módszer konvergenciája a hatványmódszerhez hasonlóan a sajátértékek $|\lambda_{i+1}/\lambda_i|$ hányadosaitól függ. Minthogy az $A-\sigma I$ mátrix sajátértékei $\lambda_1-\sigma, \lambda_2-\sigma, \dots, \lambda_n-\sigma$, az ehhez kapcsolódó sajátérték hányadosok: $|(\lambda_{i+1}-\sigma)\,/\,(\lambda_i-\sigma)|$. A σ ügyes megválasztásával ezek a hányadosok kicsivé tehetők, meggyorsítva ezáltal a konvergenciát. A Hessenberg-alakra hozást és az eltolást is alkalmazó QR-módszer algoritmusa a következő:

ELTOLÁSOS-QR-MÓDSZER(A)

1 $H_1 \leftarrow U^{-1}AU$ (H_1 felső Hessenberg-alakú)
2 $k \leftarrow 1$
3 **while** *kilépési feltétel* = HAMIS
4 **do** számítsuk ki a $H_k - \sigma_k I = Q_k R_k$ felbontást
5 $H_{k+1} \leftarrow R_k Q_k + \sigma_k I$
6 $k \leftarrow k + 1$
7 **return** H_k

A gyakorlatban a QR-módszert csak eltolásos formában használjuk. A σ_i paraméterek megválasztására különféle stratégiák léteznek. A leggyakrabban javasolt választás a következő: $\sigma_k = h_{nn}^{(k)}$ $\left(H_k = \left[h_{ij}^{(k)}\right]_{i,j=1}^{n}\right)$.

Az A sajátvektorai a QR-módszer segítségével többféleképpen is könnyen meghatározhatók. Ezek részletezése az irodalomban megtalálható.

Gyakorlatok

17.3-1. Alkalmazzuk a *hatványmódszert* az $A = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$ mátrixra a $v^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ kezdőértékkel. Mi a huszadik lépés eredménye?

17.3-2. Alkalmazzuk a hatványmódszert, az *inverz hatványmódszert* és a *QR*-módszert a

$$\begin{bmatrix} -4 & -3 & -7 \\ 2 & 3 & 2 \\ 4 & 2 & 7 \end{bmatrix}$$

mátrixra.

17.3-3. Alkalmazzuk az *eltolásos QR*-módszert az előző gyakorlat mátrixára egy rögzített $\sigma_i = \sigma$ értékkel.

17.4. Numerikus programkönyvtárak és szoftvereszközök

A tudományos és mérnöki feladatok megoldásához vezető algoritmusok számítógépi megvalósítására, hatékony kódok írásának támogatására sokféle eszközt fejlesztettek ki. A fejlesztések egyik célja, hogy a programozókat tehermentesítsék a sokszor előforduló problémák kódjainak megírásától. Vannak gyakran előforduló feladatok, ezekre biztonságos, jól működő, szabványos rutinok készülnek, amelyek nyilvános programkönyvtárakból letölthetők. A fejlesztések egy másik iránya, hogy programozási nyelvként is funkcionáló, olyan szoftvereket hozzanak létre, amelyek segítségével algoritmusok kódolása egyszerűen, gyorsan történhet. A lineáris algebrai szubrutin-gyűjtemény mellett megemlítjük a VISUAL NUMERICS (ez a régebbi, IMSL könyvtárból alakult) és a NAG könyvtárakat.

17.4.1. Szabványos lineáris algebrai szubrutinok

A BLAS (Basic Linear Algebra Subprograms) programcsomagok alapgondolata a gyakran előforduló mátrix-vektor műveletek hatékony implementálása és szabványosítása. A BLAS rutinokat eredetileg FORTRAN nyelven publikálták, de szabványos jellegük miatt számos számítógépen és programkönyvtárban optimalizált gépi kódú rutinként is elérhetők. A BLAS rutinoknak három szintje létezik:

- BLAS 1 (1979),
- BLAS 2 (1988),
- BLAS 3 (1989).

Az egyes szintek az implementált mátrixműveletek műveletigény nagyságrendjének felelnek meg. A BLAS 1-3 rutinok az adott műveletek legjobb vagy legjobbnak tartott algoritmikus megvalósításait tartalmazzák. Az egyes algoritmusok és szintek helyes megválasztása nagymértékben befolyásolja az adott program hatékonyságát. A BLAS rutinoknak létezik ún. sparse-BLAS változata is ritka mátrixok kezelésére.

Megjegyezzük, hogy a BLAS 3 rutinokat főként blokkosított párhuzamos algoritmusokhoz fejlesztették ki. A BLAS rutinokat használva épülnek fel a LINPACK, EISPACK és LAPACK szabványosított lineáris algebrai programcsomagok. A LAPACK tartalmazza az előbbi kettőt, a párhuzamosított változatok pedig a SCALAPACK csomagban találhatók. Az említett programok megtalálhatók a NETLIB nyilvános programkönyvtárban, amelynek címe:

`http:/www.netlib.org/index.html`

BLAS 1 rutinok
Legyen $\alpha \in R$, $x, y \in R^n$. A BLAS 1 rutinok a legfontosabb vektorműveleteket ($z = \alpha x$, $z = x + y$, $dot = x^T y$), az $\|x\|_2$ kiszámítását, változócseréket, forgatásokat, valamint a rendkívül gyakran előforduló ún. *saxpy* műveletet tartalmazzák, amelyet

$$z = \alpha x + y$$

definiál. A *saxpy* betűszó jelentése: „scalar alpha x plus y". A *saxpy* műveletet a következő algoritmus valósítja meg:

SAXPY(α, x, y)

```
1  n ← elemek[x]
2  for i ← 1 to n
3      do z[i] = αx[i] + y[i]
4  return z
```

A *saxpy* szoftver eredetű művelet. A BLAS 1 rutinok műveletigénye $\Theta(n)$ flop.

BLAS 2 rutinok
A BLAS 2 szint mátrix-vektor műveletei $\Theta\left(n^2\right)$ flop igényűek. Az idetartozó műveletek $y = \alpha Ax + \beta y$, $y = Ax$, $y = A^{-1}x$, $y = A^T x$, $A \leftarrow A + xy^T$ és ezek variánsai. Bizonyos műveletekben csak háromszögmátrixok szerepelhetnek. Két művelettel kell részletesen foglalkoznunk. A külső vagy diadikus szorzat módosítási művelet

$$A \leftarrow A + xy^T \qquad (A \in R^{m \times n},\ x \in R^m,\ y \in R^n)$$

kétféleképpen is megvalósítható.

Soronkénti vagy „*ij*" változat:

DIADIKUS-SZORZAT-MÓDOSÍTÁS-„IJ"-VÁLTOZAT(A, x, y)

```
1  m ← sorok[A]
2  for i ← 1 to m
3      do A[i, :] ← A[i, :] + x[i] y^T
4  return A
```

A „:" jelölés az összes megengedett indexet jelöli. Esetünkben az $1 \le j \le n$ indexhalmazt, azaz $A[i, :]$ az A mátrix teljes sorát.

Oszloponkénti, vagy „*ji*" változat:

DIADIKUS-SZORZAT-MÓDOSÍTÁS-„JI"-VÁLTOZAT(A, x, y)

```
1  n ← oszlopok[A]
2  for j ← 1 to n
3      do A[:, j] ← A[:, j] + y[j] x
4  return A
```

Itt $A[:, j]$ az A j-edik oszlopát jelöli. Vegyük észre, hogy mindkét változat *saxpy* alapú.

A *gaxpy* művelet a

$$z = y + Ax \quad (x \in R^n, y \in R^m, A \in R^{m \times n})$$

művelet elnevezése. A szintén szoftver eredetű *gaxpy* művelet a „general A x plus y" kifejezés rövidítése. A helyesen programozott *gaxpy* művelet általában előnyösebb a külső szorzat módosítási műveletnél, ezért ennek használatára kell törekedni. A *gaxpy* műveletet megvalósító algoritmus sémája a következő:

GAXPY(A, x, y)

```
1  n ← oszlopok[A]
2  z ← y
3  for j ← 1 to n
4      do z ← z + x[j] A[:, j]
5  return z
```

Vegyük észre, hogy oszloponként történik a számítás és a *gaxpy* művelet tulajdonképpen általánosított *saxpy*.

BLAS 3 rutinok

Ide tartoznak az $\Theta\left(n^3\right)$ műveletigényű mátrix-mátrix, mátrix-vektor műveletek, úgymint az $C \leftarrow \alpha AB + \beta C$, $C \leftarrow \alpha AB^T + \beta C$, $B \leftarrow \alpha T^{-1} B$ (T háromszögmátrix), valamint ezek különféle variánsai. A BLAS 3 szint műveleteit számos formában lehet algoritmizálni. Például a $C = AB$ mátrixszorzást legalább háromféleképpen tudjuk elvégezni. Legyen $A \in R^{m \times r}$, $B \in R^{r \times n}$.

MÁTRIXSZORZÁS-DOT(A, B)

```
1  m ← sorok[A]
2  r ← oszlopok[A]
3  n ← oszlopok[B]
4  C[1 : m, 1 : n] ← 0
5  for i ← 1 to m
6      do for j ← 1 to n
7              do for k ← 1 to r
8                      do C[i, j] ← C[i, j] + A[i, k] B[k, j]
9  return C
```

Az algoritmus c_{ij}-t az A mátrix i-edik sorának és a B mátrix j-edik oszlopának skalárszorzataként számítja ki (tulajdonképpen a definíciónak megfelelően).

Legyen most A, B, C oszloponként particionálva, azaz

$$\begin{aligned} A &= [a_1, \ldots, a_r] \quad (a_i \in R^m) \ , \\ B &= [b_1, \ldots, b_n] \quad (b_i \in R^r) \ , \\ C &= [c_1, \ldots, c_n] \quad (c_i \in R^m) \ . \end{aligned}$$

Ekkor fennáll, hogy

$$c_j = \sum_{k=1}^{r} b_{kj} a_k \quad (j = 1, \ldots, n) \ .$$

Tehát c_j az A oszlopainak lineáris kombinációja. A szorzat pedig felépíthető *saxpy* műveletek sorozatával.

MÁTRIXSZORZÁS-GAXPY(A, B)

```
1  m ← sorok[A]
2  r ← oszlopok[A]
3  n ← oszlopok[B]
4  C[1 : m, 1 : n] ← 0
5  for j ← 1 to n
6      do for k ← 1 to r
7             do for i ← 1 to m
8                    do C[i, j] ← C[i, j] + A[i, k] B[k, j]
9  return C
```

A *jki*-algoritmus következő ekvivalens formája mutatja, hogy ténylegesen *gaxpy* alapú eljárás.

MÁTRIXSZORZÁS-GAXPY-HÍVÁSSAL(A, B)

```
1  m ← sorok[A]
2  n ← oszlopok[B]
3  C[1 : m, 1 : n] ← 0
4  for j ← 1 to n
5      do C[:, j] = gaxpy(A, B[:, j], C[:, j])
6  return C
```

Tekintsük most az $A = [a_1, \ldots, a_r]$ $(a_i \in R^m)$ és

$$B = \begin{bmatrix} b_1^T \\ \vdots \\ b_r^T \end{bmatrix} \quad (b_i \in R^n)$$

particionálásokat. Ekkor $C = AB = \sum_{k=1}^{r} a_k b_k^T$.

MÁTRIXSZORZÁS-KÜLSŐ-SZORZAT(A, B)

```
1  m ← sorok[A]
2  r ← oszlopok[A]
3  n ← oszlopok[B]
4  C[1 : m, 1 : n] = 0
5  for k ← 1 to r
6      do for j ← 1 to n
7             do for i ← 1 to m
8                    C[i, j] ← C[i, j] + A[i, k] B[k, j]
9  return C
```

A belső ciklus egy *saxpy* műveletet valósít meg, amennyiben a_k többszörösét a C mátrix j-edik oszlopához adja.

17.4.2. Matematikai szoftverek

Ezen címszó alatt azon eszközöket értjük, melyek segítségével programfejlesztői integrált környezetben, rendkívül könnyen, tömör formában írhatunk programokat. Elsősorban matematikai feladatok kódolására fejlesztették ezeket, de olyan bővítésen mentek keresztül, hogy az élet számtalan területén alkalmazhatók. Például, a Nokia cégnél a mobiltelefonok alkatrészeinek automatikus tesztelését, minőségellenőrzését MATLAB programokkal vezérlik. A MATLAB-ról a következő alfejezetben adunk rövid ismertetést, mellette megemlítjük még a széles körben elterjedt MAPLE, a DERIVE és a MATEMATICA nevű szoftvereket is.

A MATLAB

A MATLAB matematikai szoftver a MATrix LABoratory kifejezésből kapta nevét. A név arra utal, hogy a mátrixszámítások rendkívül egyszerűen végezhetők el benne. A kezdeti változataiban egyetlen adattípust ismert: a komplex elemű mátrixot. A későbbi verziókban már megjelentek a magasabb dimenziójú tömbök, cellák, a rekord típusú adatok és az ún. objektumok. Könnyen tanulható és kezdő szintű ismeretekkel is viszonylag bonyolult programozási feladatok megoldását teszi lehetővé.

A mátrix műveletek kódolása a szokásos matematikai alakban történik. Például, ha A és B két azonos méretű mátrix, akkor az összegüket a $C = A + B$ utasítással írhatjuk elő. Tulajdonképpen csak négy utasítást tartalmaz, mindegyik szemantikája ismerős más programozási nyelvekből:

– a $Z = kifejezés$ alakú egyszerű értékadást,
– az **if** *kifejezés*, *utasítások* {**else**/**elseif** *utasítások*} **end** alakú feltételes utasítást,
– a **for** *ciklusváltozó értékeinek megadása, utasítások* **end** alakú taxatív ciklust és
– a **while** *kifejezés, utasítások* **end** alakú iteratív ciklust.

Rendkívül sok beépített függvény segíti az egyes részfeladatok elvégzését, csak szemelvényszerűen sorolunk fel néhány jellegzeteset:

– $\max(A)$ az A minden oszlopából kiválasztja a legnagyobb elemet,
– $[v, s] = \text{eig}(A)$ az A sajátértékeit és sajátvektorait adja vissza, az
– $A\backslash b$ utasítás pedig az $Ax = b$ egyenletrendszert megoldását.

Igen hatékonyan lehet a mátrixokkal elemenként is műveleteket végezni és kihasználni a mátrix particionálási lehetőségeket. Például a

$$A([2, 3], :) = 1./A([3, 2], :)$$

utasítás felcseréli az A mátrix 2-ik és 3-ik sorát, miközben ezen sorok minden elemének a reciprokát veszi.

A fenti példákkal csak ízelítőt kívántunk adni a lehetőségekből, illetve bemutatni, hogy a rendelkezésre álló eszközökkel milyen kényelmesen lehet olyan feladatokat megoldani, amelyek programja, mondjuk, PASCAL nyelven meglehetősen körülményes lenne. A beépített függvények körét ki-ki saját fejlesztésű programokkal bővítheti.

Az egyre magasabb verziószámú változatok egyre több nemlineáris feladatot megoldó függvényeket is tartalmaznak, ismét csak példákat említve: numerikus integrálásra, algebrai és differenciál egyenlet(rendszer) numerikus megoldására, optimalizálási, statisztikai feladatok megoldására szolgáló függvényt stb.

Az egy családba sorolható feladatok jól tagolt könyvtárakba, készletcsomagokba (toolboxokba) vannak csoportosítva, melyeket állandóan bővítenek.

Lehetőség van ritka mátrixok gazdaságos tárolására és az egyes beépített függvények ritka mátrixos változatának hívására (a bemeneti adatoktól függően automatikusan azt használja). Ezáltal a futási idő jelentősen csökken.

A legújabb változatok már rendkívül gazdag grafikus lehetőségeket is kínálnak.

Megjelent az intervallum aritmetikai csomag is, letölthető a

http:/www.ti3.tu-harburg.de\%7Erump\ intlab

helyről.

Más programozási nyelven (pl. C vagy FORTRAN) megírt programok is beépíthetők, alkalmas illesztéssel.

Végül meg kell az előnyei között említeni, hogy nagyon jó súgóval rendelkezik. Többszintes tájékoztatást kérhet az alkalmazó, HTML fájlokban pedig egészen részletes leírások olvashatók a matematikai háttér magyarázatokkal együtt.

Feladatok

17-1. Túlcsordulás nélkül

Írjunk olyan MATLAB programot, amely az $\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}$ normát a részeredmények *túlcsordulása* nélkül minden olyan esetben kiszámítja, amelyben a végeredmény nem okoz túlcsordulást, ugyanakkor a végeredmény hibája sem nagyobb, mint ami az eredeti formulával adódik.

17-2. Becslés

Az $x^3 - 3.330000x^2 + 3.686300x - 1.356531 = 0$ egyenletnek egy megoldása $x_1 = 1.01$. A perturbált $x^3 - 3.3300x^2 + 3.6863x - 1.3565 = 0$ egyenlet gyökei legyenek $y_1,\ y_2,\ y_3,\ y_4$. Adjunk becslést a $\min_i |x_1 - y_i|$ eltérésre.

17-3. Kétszeres szóhosszúság

Tekintsünk olyan kétszeres szóhosszúságú aritmetikai rendszert, amelyben minden $2t$ jeggyel ábrázolt szám 2 db, egyenként t jegyű szóban van tárolva. Tegyük fel, hogy a számítógép egyszerre csak t jegyű számokat tud összeadni. Tegyük fel továbbá, hogy a túlcsordulást felismeri a gép.

(i) Gondoljunk ki algoritmust két ilyen kétszeres szóhosszúságú szám összeadására, feltéve, hogy azok pozitívak.

(ii) Ha az ábrázolás megköveteli, hogy a számok mindkét felének legyen előjele, akkor módosítsuk az algoritmust úgy, hogy képes legyen mind a pozitív, mind a negatív számok helyes összeadására és az összeg mindkét részének azonos előjele legyen.

Feltehetjük, hogy a teljes összeg nem okoz túlcsordulást.

17-4. Auchmuty-tétel

Írjunk MATLAB programot az Auchmuty-féle (lásd 17.22 tétel.) hibabecslésre és végezzük el a következő vizsgálatokat.

(i) Oldjuk meg kis és nagy kondíciószámú mátrixok esetén is az $Ax = b_i$ egyenletrendszereket, ahol $A \in R^{n\times n}$ adott mátrix, $b_i = Ay_i$, $y_i \in R^n$ $(i = 1,\ldots,N)$ véletlen vektorok úgy, hogy $\|y_i\|_\infty \le \beta$. Hasonlítsuk össze a tényleges $\|\widetilde{x}_i - y_i\|$, $(i = 1,\ldots,N)$ és becsült $EST_i = \|r(\widetilde{x}_i)\|_2^2 / \left\|A^T r(\widetilde{x}_i)\right\|_2$ hibákat, ahol $\widetilde{x}_i$ az $Ax = b_i$ egyenletrendszer közelítő megoldása. Mekkora a c_i számok minimuma, maximuma, átlaga és szórása? A kapott mennyisé-

geket grafikusan is ábrázoljuk. Javasolt értékek: $n \leq 200$, $\beta = 200$, $N = 40$.
(ii) Vizsgáljuk meg a kondíciószám és a méret hatását.
(iii) Végezzük el az (i), (ii) feladatokat a LINPACK és BLAS programcsomagok segítségével.

17-5. Hilbert-mátrix

Tekintsük az $Ax = b$ lineáris egyenletrendszert, ahol A a negyedrendű Hilbert mátrix – vagyis $a_{i,j} = 1/(i + j)$ – és $b = [1, 1, 1, 1]^T$. Ismert, hogy A rosszul kondicionált, ezért az inverzét közelítsük B-vel, ahol

$$B = \begin{bmatrix} 202 & -1212 & 2121 & -1131 \\ -1212 & 8181 & -15271 & 8484 \\ 2121 & -15271 & 29694 & -16968 \\ -1131 & 8484 & -16968 & 9898 \end{bmatrix}.$$

Tehát az x megoldás egy x_0 közelítése: $x_0 = Bb$. Ez nyilván nem a pontos eredmény, de még csak nem is elfogadható közelítés, mert tudjuk, hogy a pontos megoldásnak is csak egész komponensei vannak. Alkalmazzuk az *iteratív javítást* az elfogadható egész megoldás megkeresésére, az A^{-1} helyett annak B közelítését használva.

17-6. Konzisztens norma

Legyen $\|A\|$ konzisztens norma és tekintsük az $Ax = b$ egyenletrendszert.
(i) Igazoljuk, hogy ha $A + \Delta A$ szinguláris, akkor $\text{cond}(A) \geq \|A\| / \|\Delta A\|$.
(ii) Mutassuk meg, hogy „2”-es norma esetén (i)-ben az egyenlőség áll fenn, ha $\Delta A = -bx^T/(b^t x)$ és $\left\|A^{-1}\right\|_2 \|b\|_2 = \left\|A^{-1}b\right\|_2$.
(iii) Az (i)-t felhasználva adjunk alsó korlátot a $\text{cond}_\infty(A)$-ra, ha

$$A = \begin{bmatrix} 1 & -1 & 1 \\ -1 & \varepsilon & \varepsilon \\ 1 & \varepsilon & \varepsilon \end{bmatrix}.$$

17-7. Cholesky-módszer

Tekintsük az $Ax = b$ lineáris egyenletrendszert, ahol

$$A = \begin{bmatrix} 5.5 & 0 & 0 & 0 & 0 & 3.5 \\ 0 & 5.5 & 0 & 0 & 0 & 1.5 \\ 0 & 0 & 6.25 & 0 & 3.75 & 0 \\ 0 & 0 & 0 & 5.5 & 0 & 0.5 \\ 0 & 0 & 3.75 & 0 & 6.25 & 0 \\ 3.5 & 1.5 & 0 & 0.5 & 0 & 5.5 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Mivel A szimmetrikus, pozitív definit, ezért a Cholesky-módszerrel oldjuk meg. Adjuk meg a pontos $A = LL^T$ felbontást és az egyenletrendszer pontos megoldását. A *Cholesky-felbontás* során a pontos L helyett kapott $\tilde{L}$ közelítésre $\tilde{L}\tilde{L}^T = A + F$. Igazolható, hogy β alapú, t mantisszahosszúságú *lebegőpontos aritmetikában* az F elemeire fennáll: $\left|f_{ij}\right| \leq e_{ij}$, ahol

$$E = \beta^{1-t} \begin{bmatrix} 11 & 0 & 0 & 0 & 0 & 3.5 \\ 0 & 11 & 0 & 0 & 0 & 1.5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 11 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 3.5 & 1.5 & 0 & 0.5 & 0 & 11 \end{bmatrix}.$$

IBM3033 típusú számítógépnél $\beta = 16$ és $t = 14$. Adjunk korlátot a kapott $\tilde{x}$ közelítő megoldás *relatív hibájára.*

17-8. Bauer–Fike-tétel

Legyen

$$A = \begin{bmatrix} 10 & 10 & & & & & \\ & 9 & 10 & & & & \\ & & 8 & 10 & & & \\ & & & \ddots & \ddots & & \\ & & & & & 2 & 10 \\ \varepsilon & & & & & & 1 \end{bmatrix}.$$

(i) Vizsgáljuk meg a *sajátértékek* megváltozását az $\varepsilon = 10^{-5}, 10^{-6}, 10^{-7}, 0$ esetén.
(ii) Vizsgáljuk meg a Bauer–Fike tétel becslését az $A = A(0)$ mátrixhoz képest.

17-9. Sajátérték

Határozzuk meg $B = AA^T$ sajátértékeit véletlen A mátrixokkal különböző n értékekre a MATLAB eig rutinjával, ahol $A \in R^{n\times n}$ adott mátrix. Határozzuk meg a $B + R_i$ mátrix sajátértékeit, ha R_i véletlen mátrix, amelynek elemei a $\left[-10^{-5}, 10^{-5}\right]$ intervallumba esnek $(i = 1, \ldots, N)$. Mekkora B és a $B + R_i$ sajátértékeinek maximális eltérése? Milyen pontos a Bauer-Fike-tétel becslése? Javasolt értékek: $N = 10$, $5 \le n \le 200$.

Hogyan alakulnak az eredmények a kondíciószám függvényében? Tapasztalunk-e függést az n rendszámtól? Ábrázoljuk grafikusan a maximális eltéréseket és a Bauer–Fike-tétel becslését.

Megjegyzések a fejezethez

A lineáris egyenletrendszerek közelítő megoldására alkalmazott utólagos hibabecslések nem teljesen megbízhatók. Demmel, Diament és Malajovich kimutatták, hogy az $\Theta\left(n^2\right)$ költségű kondíciószámbecslők esetén mindig vannak olyan esetek, amikor a becslés megbízhatatlan (a becslés hibája meghalad egy meghatározott nagyságrendet) [107].

A lineáris egyenletrendszerek közelítő megoldására vonatkozó iteratív javítás első ismert alkalmazása Fox, Goodwin, Turing és Wilkinson nevéhez fűződik (1946). A tapasztalatok szerint a reziduális hiba csökkenése nem monoton. A módszer alkalmazásának egy lehetséges változata a pointeres LU-módszerrel összekapcsolva a [441] könyvben is szerepel.

Az iterációs módszerek elméletének és alkalmazásainak kitűnő összefoglalását tartalmazzák Young , illetve Hageman és Young könyvei [503], [188]. A téma szoftverelvű áttekintését adják Barett, Berry és társaik [31]. Itt hívjuk fel a figyelmet Andreas Frommer könyvére is, amely az iteratív módszerek párhuzamosítására is kitér [141].

A *QR*-módszer konvergenciájára vonatkozó 17.34. tételnél lényegesen jobb konvergencia eredmények is ismertek. Számos, a *QR*-módszerrel rokon eljárás ismert a sajátérték feladat megoldására (lásd például [492], [488]). Ezek közül az egyik legismertebb, az ún. *LR* módszer, amit pozitív definit hermitikus mátrixra előnyös alkalmazni. Lényege: állítsuk elő az $A_k = LL^*$ Cholesky-felbontást, majd legyen $A_{k+1} = L^*L$.

Ismeretesek implicit, dupla eltolást alkalmazó, a *QR*-módszerhez hasonló eljárások is. Mindenesetre már a 3×3-as Hessenberg-alakú mátrixok között is van példa, hogy komplex

sajátértékek esetén többszörös eltolással sem érhető el konvergencia – azaz a (17.41)-ben megmutatott alak – pontos számítások mellett, amint azt Batterson kimutatta [34]:

$$\begin{bmatrix} 0.83116322648071935 & -0.34924697025783983 & 0.06972435460743861 \\ 0.35613892213531548 & 0.86568478391818912 & 0.34924697025783983 \\ 0 & -0.35613892213531548 & 0.83116322648071935 \end{bmatrix}.$$

(a kerekítési hibák miatt, paradox módon, mégis tapasztalható igen lassú konvergencia).

Irodalomjegyzék

[1] S. Abiteboul, V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995. 534, 535

[2] L. Addario-Berry, B. Chor, M. Hallett, J. Lagergren, A. Panconesi, T. Wareham. Ancestral maximum likelihood of phylogenetic trees is hard. *Lecture Notes in Bioinformatics*, 2812:202–215, 2003. 577

[3] R. G. Addie, M. Zukerman, T. Neame. Broadband traffic modeling: Simple solutions to hard problems. *IEEE Communications Magazine*, (8):88–95, 1998. 221

[4] M. Agrawal, N. Kayal, N. M. Saxena. PRIMES is in P. Available at http://www.cse.iitk.ac.in/users/manindra/primality.ps, 2002. 123

[5] A. Aho, C. Beeri, J. D. Ullman. The theory of joins in relational databases. *ACM Transactions on Database Systems*, 4(3):297–314, 1979. 534

[6] A. V. Aho, J. E. Hopcroft, J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974. Magyarul: *Számítógép-algoritmusok tervezése és analízise*. Műszaki Könyvkiadó, 1982. 9

[7] R. K. Ahuja, T. L. Magnanti, J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993. 604

[8] A. G. Akritas. *Elements of Computer Algebra with Applications*. John Wiley & Sons, 1989. 93

[9] T. Akutsu. Dynamic programming algorithms for RNA secondary prediction with pseudoknots. *Discrete Applied Mathematics*, 104:45–62, 2000. 578

[10] E. Althaus, A. Caprara, H. Lenhof, K. Reinert. Multiple sequence alignment with arbitrary gap costs: Computing an optimal solution using polyhedral combinatorics. *Bioinformatics*, 18:S4–S16, 2002. 576

[11] I. Althöfer. *13 Jahre 3-Hirn*. A szerző kiadása, 1998. 604

[12] I. Althöfer. List-3-Hirn vs. grandmaster Yussupov – report on a very experimental match. *ICCA Journal*, 21:52–60 and 131–134, 1998. 604

[13] I. Althöfer. Improved game play by multiple computer hints. *Theoretical Computer Science*, 313:315–324, 2004. 604

[14] I. Althöfer, F. Berger, S. Schwarz. Generating True Alternatives with a Penalty Method. http://www.minet.uni-jena.de/Math-Net/reports/shadows/02-04report.html, 2002. 604

[15] I. Althöfer, J. de Koning, J. Lieberum, S. Meyer-Kahlen, T. Rolle, J. Sameith. Five visualisations of the k-best mode. *ICCA Journal*, 26:182–189, 2003. 604

[16] G. M. Amdahl. Validity of the single-processor approach to achieving large-scale computer capabilities. In *AFIPS Conference Proceedings*, 30. kötet, 483–485. o., 1967. 267

[17] D. Anick, D. Mitra, M. Sondhi. Stochastic theory of a data handling system with multiple sources. *Bell System Technical Journal*, 61:1871–1894, 1982. 221

[18] V. Arlazanovan, A. Dinic, M. Kronrod, I. Faradzev. On economic construction of the transitive closure of a directed graph. *Doklady Academii Nauk SSSR*, 194:487–488, 1970. 576

[19] W. Armstrong. Dependency structures of database relationships. In *Proceedings of IFIP Congress*, 580–583. o. North Holland, 1974. 534

[20] V. Arvind, P. Kurur. Graph isomorphism is in SPP. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*, 743–750. o. IEEE Computer Society Press, 2002. 159

[21] K. Atteson. The performance of the neighbor-joining method of phylogeny reconstruction. *Algorithmica* 25(2/3):251–278, 1999. 578

[22] J-P. Aubin. *Mathematical Methods of Game and Economic Theory*. North-Holland, 1979. 360

[23] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, 421–429. o. ACM Press, 1985. 123, 124

[24] L. Babai, S. Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and Systems Sciences*, 36(2):254–276, 1988. 123, 124

[25] I. Bach. *Formális nyelvek (Formal languages)*. Typotex, 2001. 576

[26] B. Baker. A tight asymptotic bound for next-fit decreasing bin-packing. *SIAM Journal on Algebraic and Discrete Methods*, 2(2):147–152, 1981. 502

[27] E. Balas. Sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operation Research*, 17:941–953, 1969. 414

[28] B. Balkenhol, S. Kurtz. Universal data compression based on the Burrows–Wheeler transform: theory and practice. *IEEE Transactions on Compututers*, 49(10):1043–1053–953, 2000. 454

[29] J. Banks, J. Carson, B. Nelson. *Discrete-Event Simulation*. Prentice Hall, 1996. 221

[30] W. Banzhaf. Interactive evolution. In T. Back, D. B. Fogel, Z. Michalewicz, T. Baeck (szerkesztők), *Handbook of Evolutionary Computation*. IOP Press, 1997. 604

[31] R. Barett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozzo, C. Romine, H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 1994. 765

[32] J. R. Barker, G. B. McMahon. Scheduling the general job-shop. *Management Science*, 31:594–598, 1985. 414, 415

[33] T. Bartha, A. Pataricza. *Formális módszerek az informatikában (Formal Methods in Informatics)*. Typotex, 2004. 455

[34] S. Batterson. Convergence of the shifted QR algorithm on 3×3 normal matrices. , 58, 1990. 766

[35] C. Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Transactions on Database Systems*, 5:241–259, 1980. 534

[36] C. Beeri, P. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Transactions on Database Systems*, 4(1):30–59, 1979. 534

[37] C. Beeri, M. Dowd. On the structure of armstrong relations for functional dependencies. *Journal of ACM*, 31(1):30–46, 1984. 534

[38] C. Beeri, R. Fagin, J. Howard. A complete axiomatization for functional and multivalued dependencies. In *ACM SIGMOD Symposium on the Management of Data*, 47–61. o., 1977. 534

[39] T. C. Bell, I. H. Witten, J. G. Cleary. Modeling for text compression. *Communications of the ACM*, 21:557–591, 1989. 454

[40] T. C. Bell, I. H. Witten, J. G. Cleary. *Text Compression*. Prentice Hall, 1990. 454, 455

[41] A. Benczúr A. Kiss, T. Márkus. A relációs adatmodell függőségeinek egy általános osztálya és implikációs problémáinak vizsgálata. *Alkalmazott Matematikai Lapok*, 13:291–312, 1987-88. 535

[42] J. Beran. *Statistics for Long-Memory Processes*. Monographs on Statistics and Applied Probability. Chapman & Hall, 1986. 221

[43] J. Beran, R. Sherman, M. Taqqu, W. Willinger. Long-range dependence in variable-bit-rate video traffic. *IEEE Transactions on Communications*, 43:1566–1579, 1995. 221

[44] F. Berger. *k* alternatives instead of *k* shortest paths. Master's thesis, Friedrich Schiller Egyetem, Jéna, Matematikai és Informatikai Kar, 2000. Diplomamunka. 604

[45] K. A. Berman, J. L. Paul. *Sequential and Parallel Algorithms*. PWS Publishing Company, 1996. 266

[46] A. Beygelzimer, L. Hemaspaandra, C. Homan, J. Rothe. One-way functions in worst-case cryptography: Algebraic and security properties are on the house. *SIGACT News*, 30(4):25–40, 1999. 123

[47] A. Békéssy, J. Demetrovics. Contribution to the theory of data base relations. *Discrete Mathematics*, 27(1):1–10, 1979. 534

[48] A. Békéssy, J. Demetrovics. *Előadások adatbázis szerkezetekről (Lectures on data base structures)*. ELTE Eötvös Kiadó, 1999. 535

[49] A. Békéssy, J. Demetrovics. *Adatbázis szerkezetek (Data Base structures)*. Akadémiai Kiadó, 2004. Nyomdában. 535

[50] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, J. Weglarz. *Scheduling Computer and Manufacturing Processes*. Springer-Verlag, 2001 (2. kiadás). 415

[51] L. A. Bélády. A study of replacement algorithms for a virtual storage computer. *IBM Systems Journal*, 5(2):78–101, 1965. 502

[52] L. A. Bélády, R. Nelson, G. S. Shedler. An anomaly in space-time characteristics of certain programs running in paging machine. *Communications of the ACM*, 12(1):349–353, 1969. 502

[53] J. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):135–256, 1982. 683, 684

[54] J. Bloomenthal. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers, 1997. 683

[55] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999. 123

[56] B. Borchert, L. Hemaspaandra, J. Rothe. Restrictive acceptance suffices for equivalence problems. *London Mathematical Society Journal of Computation and Mathematics*, 86:86–95, 2000. 159

[57] R. P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21:201–206, 1974. 267

[58] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965. 684

[59] G. Brooks, C. R. White. An algorithm for finding optimal or near optimal solutions to the production scheduling problem. *Journal of Industrial Engineering*, 16:34–40, 1965. 414

[60] L. E. J. Brouwer. Über Abbildung von Manningfaltigkeiten. *Mathematische Annalen*, 97–115. o., 1912. 359

[61] P. Bruckner. *Scheduling Algorithms*. Springer-Verlag, 1998. 413

[62] B. Buchberger. Ein algorithmus zum auffinden der basiselemente des restklassenringes nach einem nulldimensionalen polynomideal, 1965. PhD disszertáció, Leopold-Franzens-Universität, Innsbruck. 92

[63] M. Burrows, D. J. Wheeler. A block-sorting lossless data compression algorithm. Research Report 124, http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-124.html, 1994. 454

[64] L. Buttyán, I. Vajda. *Kriptográfia és alkalmazásai (Cryptography and its Applications)*. Typotex, 2004. 124

[65] CACI. *COMNET III*. CACI Products Co., 1997. 221

[66] Calgary. The Calgary/Canterbury Text Compression. ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus, 2004. 455

[67] Canterbury. The Canterbury Corpus. http://corpus.canterbury.ac.nz, 2004. 455

[68] J. Carlier, E. Pinson. Algorithm for solving the job-shop problem. *Management Science*, 35:164–176, 1989. 414, 415

[69] J. L. Carter, M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. 159

[70] E. Catmull, J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350–355, 1978. 683

[71] B. F. Caviness. Computer algebra: past and future. *Journal of Symbolic Computations*, 2:217–263, 1986. 93

[72] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2000. 267

[73] J. Charlton, C. C. Death. A method of solution for general machine scheduling problems. *Operations Research*, 18:689–707, 1970. 414

[74] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry* 6(5):353–363, 1991. 683

[75] S. M. Christensen. Resources for computer algebra. *Computers in Physics*, 8:308–315, 1994. 93

[76] D. Christie. Sorting permutations by block-interchanges. *Information Processing Letters*, 60(4):165–169, 1996. 578

[77] E. F. Codd. Further normalization of the data base relational model. In R. Rustin (szerkesztő), *Courant Computer Science Symposium 6: Data Base Systems*, 33–64. o. Prentice Hall, 1972. 534

[78] E. F. Codd. Relational completeness of database sublanguages. In R. Rustin (szerkesztő), *Courant Computer Science Symposium 6: Data Base Systems*, 65–98. o. Prentice Hall, 1972. 534

[79] E. F. Codd. Recent investigations in relational data base systems. In *Information Processing 74*, 1017–1021. o. North-Holland, 1974. 534

[80] E. F. Codd. A relational model of large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. 534

[81] E. F. Codd. Normalized database structure: A brief tutorial. In *ACM SIGFIDET Workshop on Data Description, Access and Control*, 24–30. o., 1971. 534

[82] E. Coffman. *Computer and Job Shop Scheduling*. John Wiley & Sons, 1976. 415, 502

[83] E. Coffman, M. Garey, D. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7:1–17, 1978. 414

[84] E. G. Coffman, J. Csirik, G. J. Woeginger. Approximate solutions to bin packing problems. In P. M. Pardalos, M. G. C. Resende (szerkesztők), *Handbook of Applied Optimization*, 607–615. o. Oxford University Press, 2002. 414

[85] A. M. Cohen, L. van Gasten, S. Lunel (szerkesztők). *Computer Algebra for Industry 2, Problem Solving in Practice*. John Wiley & Sons, 1995. 93

[86] A. M. Cohen (szerkesztő). *Computer Algebra for Industry: Problem Solving in Practice*. John Wiley & Sons, 1993. 93

[87] S. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 151–158. o. ACM Press, 1971. 158

[88] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997. 123

[89] T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to Algorithms*. The MIT Press/McGraw-Hill, 1990 (Magyarul: *Algoritmusok*. Műszaki Kiadó, 2003, negyedik kiadás). 9, 266

[90] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms*. The MIT Press/McGraw-Hill, 2003 (Második kiadás negyedik, javított utánnyomása. Magyarul: *Új algoritmusok*. Scolar Kiadó, 2003). 9, 92, 93, 159, 263, 454, 502, 576, 716

[91] T. M. Cover, J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, 1991. 455

[92] H. S. M. Coxeter. *Projective Geometry*. University of Toronto Press, 1974 (2. kiadás). 683

[93] R. Cristescu, G. Marinescu. *Unele aplicatii ale teoriei distributilor*. Editura academiei republicii socialiste, Magyarul: *Bevezetés a disztribúcióelméletbe és alkalmazásaiba*, Műszaki könyvkiadó, 1969). 716

[94] M. Crovella, A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997. 221

[95] I. Csiszár, J. Körner. *Coding Theorems for Discrete Memoryless Systems*. Akadémiai Kiadó, 1981 (Magyarul: *Információelmélet*. Tankönyvkiadó, 1980). 455

[96] D. E. Culler, R. M. Karp, D. Patterson, A. Sahay, E. E. Santos, K. E. Schauser, R. Subramonian, T. von Eicken. Logp: A practical model of parallel computation. *Communication of the ACM*, 39(11):78–85, 1996. 267

[97] D. E. Culler, J. P. Singh, A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufman Publisher, 1983. 267

[98] E. Dantsin, A. Goerdt, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, U. Schöning. A deterministic $(2-2/(k+1))^n$ algorithm for k-sat based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002. 158, 159

[99] A. Darte, Y. Robert, F. Vivien. *Scheduling and Automatic Parallelization*. Birkhäuser Boston, 2000. 310

[100] J. Davenport, Y. Siret, E. Tournier, E.. *Computer Algebra: Systems and Algorithms for Algebraic Computation*. Academic Press, 2000. 93

[101] M. de Berg. *Efficient Algorithms for Ray Shooting and Hidden Surface Removal*. PhD thesis, Rijksuniversiteit te Utrecht, 1992. 683

[102] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2000. 683

[103] C. Delobel. Normalization and hierarchical dependencies in the relational data model. *ACM Transactions on Database Systems*, 3(3):201–222, 1978. 534

[104] J. Demetrovics, Gy. O. H. Katona, A. Sali. Minimal representations of branching dependencies. *Discrete Applied Mathematics*, 40:139–153, 1992. 535

[105] J. Demetrovics, Gy. O. H. Katona, A. Sali. Minimal representations of branching dependencies. *Acta Scientiarum Mathematicorum (Szeged)*, 60:213–223, 1995. 535

[106] J. Demetrovics, Gy. O. H. Katona, A. Sali. Design type problems motivated by database theory. *Journal of Statistical Planning and Inference*, 72:149–164, 1998. 535

[107] J. Demmel, D. Malajovich. On the complexity of computing error bounds. *Foundations of Computational Mathematics*, 1:101–125, 2001. 765

[108] P. Denning. Virtual memory. *Computing Surveys* 2(3):153–189, 1970. 501

[109] A. Detrekői, Gy. Szabó. *Bevezetés a térinformatikába (Introduction to Geoinformatics)*. Nemzeti Tankönyvkiadó, 1995. 716

[110] Á. Detrekői, Gy. Szabó. *Térinformatika (Geoinformatics)*. Nemzeti Tankönyvkiadó, 2003 (második kiadás). 716

[111] W. Diffie, M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory Theory*, IT-22(6):644–654, 1976. 122

[112] D. Doernberg. Interview with Donald Knuth. *Computer Literacy* 1993. 579

[113] N. Duffield, N. Oconnell. Large deviation and overflow probabilities for the general single-server queue, with applications. *Mathematical Proceedings of the Cambridge Philosophical Society*, 118:363–374, 1995. 221

[114] D. Duffy, A. McIntosh, M. Rosenstein, W. Willinger. Statistical analysis of ccsn/ss7 traffic data from working ccs subnetworks. *IEEE Journal on Selected Areas Communications*, 12:544–551, 1994. 221

[115] J. Duncan. *The Elements of Complex Analysis*. John Wiley & Sons, 1968 (Magyarul: Bevezetés a komplex függvénytanba, Műszaki Könyvkiadó, 1974). 716

[116] R. Durbin, S. Eddy, A. Krogh, G. Mitchison. *Biological Sequence Analysis*. University Press, 1998. 576

[117] N. Dyn, J. Gregory, D. Levin. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics* 9:160–169, 1990. 683

[118] M. Effros, K. Viswesvariah, S. Kulkarni, S. Verdú. Universal lossless source coding with the burrows–wheeler transform. *IEEE Transactions on Information Theory Theory*, 48(5):1061–1081, 2002. 454

[119] S. N. Elaydi. *An Introduction to Difference Equations*. Springer-Verlag, 1999 (2. kiadás). 37

[120] P. Erdős, M. Steel, L. Székely, T. Warnow. Local quartet splits of a binary tree infer all quartet splits via one dyadic inference rule. *Computers and Artificial Intelligence*, 16(2):217–227, 1997. 578

[121] A. Erramilli, O. Narayan, W. Willinger. Experimental queueing analysis with long-range dependent packet-traffic. *IEEE/ACM Transactions on Networking* 4(2):209–223, 1996. 221

[122] R. Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems*, 2:262–278, 1977. 534

[123] R. Fagin. Armstrong databases. In *Proceedings of IBM Symposium on Mathematical Foundations of Computer Science*, 1982. 534

[124] R. Fagin. Horn clauses and database dependencies. *Journal of ACM*, 29(4):952–985, 1982. 534

[125] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Morgan Kaufmann Publishers, 1998. 683

[126] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., 2003. 576

[127] S. Fenner, L. Fortnow, S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994. 159

[128] R. Fernando. *GPUGems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*. Addison-Wesley, 2004. 684

[129] M. L. Fisher. Optimal solution of scheduling problems using lagrange multipliers: Part I. *Operations Research*, 21:1114–1127, 1973. 414, 415

[130] M. L. Fisher. Optimal solution of scheduling problems using lagrange multipliers: Part II,. In S. E. Elmaghraby (szerkesztő), *Symposium on the Theory of Scheduling and Its Applications*. Springer-Verlag, 1973. 414

[131] Z. Fülöp. *Formális nyelvek és szintaktikus elemzésük (Formal Languages and their Syntactical Analysis)*. Polygon, 2000. 576

[132] M. J. Flynn. Very high-speed computer systems. *Proceedings of the IEEE* 5(6):1901–1909, 1966. 266

[133] J. D. Fooley, A., S. K. Feiner, J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1990. 684

[134] F. Forgó, J. Szép, F. Szidarovszky. *Introduction to the Theory of Games: Concepts, Methods and Applications*. Kluwer Academic Publishers, 1999. 360

[135] P. Fornai, A. Iványi. Bélády's anomaly is unbounded. In Kovács Előd és Winkler Zoltán (szerkesztő), *5th International Conference on Applied Informatics*, 65–72. o. Molnár és társa, 2002. 502

[136] S. Fortune, J. Wyllie. Parallelism in random access machines. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, 114–118. o., 1978. 267

[137] I. Foster, C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufman Publisher, 2004 (2. kiadás). 267

[138] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. Available at www.globus.org/research/papers/ogsa.pdf, 2002. 267

[139] D. K. Friesen. Tighter bounds for the multifit processor scheduling algorithm. *Journal of Algorithms* 7:35–59, 1984. 414

[140] D. K. Friesen, M. A. Langston. Evaluation of a multifit-based scheduling algorithm. *Journal of Algorithms*, 7:35–59, 1986. 414

[141] A. Frommer. *Lösung linearer Gleichungssysteme auf Parallelrechnern*. Vieweg Verlag, 1990. 765

[142] H. Fuchs, Z. M. Kedem, B. F. Naylor. On visible surface generation by a priory tree structures. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, 124–133. o., 1980. 684

[143] A. Fujimoto, T. Takayuki, I. Kansey. ARTS: accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, 1986. 683

[144] S. Gajdos. *Adatbázisok (Data Bases)*. Műegyetemi Kiadó, 2001. 535

[145] T. Gal, T. Stewart, T. Hanne (szerkesztők). *Multicriteria Decision Making*. Kluwer Academic Publisher, 1999. 604

[146] G. Galambos. *Operációs rendszerek (Operating Systems)*. Műszaki Könyvkiadó, 2003. 502

[147] H. Garcia-Molina, J. D. Ullman. *Database System Implementation*. Prentice Hall, 2000. Magyarul: *Adatbázisrendszerek megvalósítása*. Panem, 2001. 535

[148] M. R. Garey, R. L. Graham. Performance guarantees for scheduling algorithm. *Operations Research*, 26:3–21, 1978. 415

[149] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 10, 158

[150] W. H. Gates, C. Papadimitriou. Bounds for sorting by prefix reversals. *Discrete Mathematics*, 27:47–57, 1979. 578

[151] F. Gécseg, I. Peák. *Algebraic Theory of Automata*. Akadémiai Kiadó, 1972. 501

[152] K. O. Geddes S. Czapor, G. Labahhn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992. 92, 93

[153] P. B. Gibbons, Y. Matias, V. Ramachandran. Can a shared-memory model serve as a bridging model for parallel computation. *Theory of Computing Systems*, 32(3):327–359, 1999. 267

[154] A. Glassner. *An Introduction to Ray Tracing*. Academic Press, 1989. 683

[155] A. Gál, S. Halevi, R. Lipton, E. Petrank. Computing from partial solutions. In *Proceeding of the 14th Annual IEEE Conference on Computational Complexity*, 34–45. o. IEEE Computer Society Press, 1999. 159

[156] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989. 604

[157] N. Goldman, J. Thorne, D. Jones. Using evolutionary trees in protein secondary structure prediction and other comparative sequence analyses. *Journal of Molecular Biology*, 263(2):196–208, 1996. 577

[158] O. Goldreich. Randomness, interactive proofs, and zero-knowledge – a survey. In R. Herken (szerkesztő), *The Universal Turing Machine: A Half-Century Survey*, 377–405. o. Oxford University Press, 1988. 124

[159] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001. 123, 124

[160] O. Goldreich, S. Micali, A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991. 123

[161] S. Goldwasser. Interactive proof systems. In J. Hartmanis (szerkesztő), *Computational Complexity Theory*, *AMS Short Course Lecture Notes: Introductory Survey Lectures. Proceedings of Symposia in Applied Mathematics* 38. kötete, 108–128. o. American Mathematical Society, 1989. 124

[162] S. Goldwasser, S. Micali, C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. 123, 124

[163] S. Goldwasser, M. Sipser. Private coins versus public coins in interactive proof systems. In S. Micali (szerkesztő), *Randomness and Computation*, *Advances in Computing Research* 5. kötete, 73–90. o. JAI Press, 1989. A preliminary version appeared in *Proc. 18th Ann. ACM Symp. on Theory of Computing*, 1986, pp. 59–68. 123

[164] G. Gonnet, D. Gruntz, L. Bernardin. Computer algebra systems. In A. Ralston, E. D. Reilly, D. Hemmendinger (szerkesztők), *Encyclopedia of Computer Science*, 287–301. o. Nature Publishing Group, 4. kiadás, 2000. 93

[165] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982. 575

[166] H. Gouraud. Computer display of curved surfaces. *IEEE Transactions on Computers*, C-20(6):623–629, 1971. 678

[167] J. Grabowski. On two-machine scheduling with release and due-dates to minimize maximum lateness. *Opsearch*, 17:133–154, 1980. 414

[168] R. L. Graham. Bounds for certain multiprocessor anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966. 414

[169] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969. 415

[170] R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979. 413

[171] R. L. Graham, D. E. Knuth, O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 1994 (2. kiadás. Magyarul: *Konkrét matematika*, Műszaki Könyvkiadó, 1998). 37, 93

[172] A. Grama, A. Gupta, G. Karypis, V. Kumar. *Introduction to Parallel Computing*. Addison-Wesley, 2003 (2. kiadás). 266

[173] J. Grant, J. Minker. Inferences for numerical dependencies. *Theoretical Computer Science*, 41:271–287, 1985. 535

[174] J. Grant, J. Minker. Normalization and axiomatization for numerical dependencies. *Information and Control*, 65:1–17, 1985. 535

[175] D. H. Greene, D. E. Knuth. *Mathematics for the Analysis of Algorithms*. Birkhäuser, 3. kiadás, 1990. 37

[176] W. Gropp, M. Snir, B. Nitzberg, E. Lusk. *MPI: The Complete Reference*. Scientific and Engineering Computation Series. The MIT Press, 1998. 267

[177] A. Große, J. Rothe, G. Wechsung. Computing complete graph isomorphisms and hamiltonian cycles from partial ones. *Theory of Computing Systems*, 35(1):81–93, 2002. 159

[178] A. Grosse, S. Schwarz. Bigblackcell. http://www.minet.uni-jena.de/~BigBlackCell/. 604

[179] R. Grossman. *Symbolic Computation: Applications to Scientific Computing*, *Frontiers in Applied Mathematics* 5. kötete. SIAM, 1989. 93

[180] J. N. D. Gupta, S. S. Reddi. Improved dominance conditions for the three-machine flowshop scheduling problem. *Operations Research*, 26:200–203, 1978. 414

[181] R. Gusella. A measurement study of diskless workstation traffic on an ethernet. *IEEE Transactions on Communications*, 38:1557–1568, 1990. 221

[182] D. M. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997. 575, 576

[183] J. Gustafson. Reevaluating Amdahl's law. *Communications of ACM*, 28(1):532–535, 1988. 267

[184] T. Gyires. Simulation of the harmful consequences of self-similar network traffic. *The Journal of Computer Information Systems*, 42(4):94–111, 2002. 221

[185] L. Györfi. *Információ- és kódelmélet (Theory of Information and Coding)*. Typotex, 2002 (2. kiadás). 455

[186] J. Håstad. Solving simultaneous modular equations of low degree. *SIAM Journal on Computing*, 17(2):336–341, 1988. Special issue on cryptography. 123

[187] G. Hadley. *Nonlinear and Dynamic Programming*. Addison-Wesley, 1964. 360

[188] L. Hageman, D. Young. *Applied Iterative Methods*. Academic Press, 1981. 765

[189] Gy. F. Hajós. *Bevezetés a geometriába (Introduction to Geometry)*. Tankönyvkiadó, 1972. 683

[190] T. S. Han, K. Kobayashi. *Mathematics of Information and Coding*. American Mathematical Society, 2002. 455

[191] D. Hankerson, G. A. Harris, P. D. Johnson. *Introduction to Information Theory and Data Compression*. CRC Press, 2003 (2. kiadás). 455

[192] S. Hannenhalli. Polynomial-time algorithm for computing translocation distance between genomes. *Discrete Applied Mathematics*, 71:137–151, 1996. 578

[193] D. Harper, C. Wooff D. Hodginson. *A Guide to Computer Algebra Systems*. John Wiley & Sons, 1991. 93

[194] J. Hartmanis, L. Hemachandra. Complexity classes without machines: On complete languages for UP. *Theoretical Computer Science*, 58(1–3):129–142, 1988. 159

[195] B. Cs. Hatvany. *Bitképek feldolgozása Visual Basic programokból (Bitmap Processing by Visual Basic programms)*. Computer Books, 2003. 716

[196] V. Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Czech Technical University, 2001. 683, 684

[197] M. Höchsman, T. Töller, R. Giegerich, S. Kurtz. Local similarity in RNA secondary structure. In *Proceedings of IEEE Bioinformatics Conference 2003*, 159–168. o., 2003. 578

[198] A. C. Hearn. *Future Directions for Research in Symbolic Computation*. SIAM Reports on Issues in the Mathematical Sciences. SIAM, 1990. 93

[199] H. Hefles, D. Lucantoni. A markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance. *IEEE Journal on Selected Areas in Communication*, 4:856–868, 1986. 221

[200] E. A. Heinz. *Algorithmic Enhancements and Experiments at High Search Depths*. Vieweg Verlag, Series on Computational Intelligence, 2000. 604

[201] L. Hemaspaaandra, K. Pasanen, J. Rothe. If P ≠ NP then some strongly noninvertible functions are invertible. In *Proceedings of the 13th International Symposium on Fundamentals of Computation Theory*, *Lecture Notes in Computer Science* 2138. kötete, 162–171. o. Springer-Verlag, 2001. 123

[202] L. Hemaspaandra, J. Rothe. Creating strong, total, commutative, associative one-way functions from any one-way function in complexity theory. *Journal of Computer and Systems Sciences*, 58(3):648–659, 1999. 123

[203] L. A. Hemaspaandra, M. Ogihara. *The Complexity Theory Companion*. EATCS Texts in Theoretical Computer Science. Springer-Verlag, 2002. 158

[204] I. Herman. *The Use of Projective Geometry in Computer Graphics*. Springer-Verlag, 1991. 683

[205] P. Hermann, A. Iványi. A munkahatékonyság korlátai prefixszámításnál és összefésülésnél. *Alkalmazott Matematikai Lapok*, 2004. Submitted. 267

[206] F. Heylighen. Principia Cybernetica Project. http://pespmc1.vub.ac.be/HEYL.html, 2004. 221

[207] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18:341–343, 1975. 576

[208] C. Hoffman (szerkesztő). *Group-Theoretic Algorithms and Graph Isomorphism*, *Lecture Notes in Computer Science* 136. kötete. Springer-Verlag, 1982. 159

[209] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2. kiadás, 2001. 501

[210] W. A. Horn. Minimizing average flow time with parallel machines. *Operations Research*, 21:845–847, 1973. 414

[211] E. Horowitz, S. Sahni, S. Rajasekaran. *Computer Algorithms*. Computer Science Press, 1998. 266

[212] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. 454

[213] R. Hughey, A. Krogh. Hidden markov models for sequence analysis: Extension and analysis of the basic method. *CABIOS*, 12(2):95–107, 1996. 577

[214] M. Hujter. Egy ütemezéselméleti probléma vizsgálata és alkalmazásai (Investigation of a scheduling problem and its applications), 1987. Doktori értekezés, ELTE TTK. 414

[215] J. Hunt, T. Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, 1977. 576

[216] K. Hwang, Z. Xu. *Scalable Parallel Computing*. McGraw-Hill, 1998. 267

[217] Interoute, http://wwww.interoute.com/glossary.html, 2004. 221

[218] A. Iványi. On dumpling-eating giants. In *Finite and Infinite Sets (Eger, 1981)*, *Colloquia of Mathematical Society János Bolyai* 37. kötete, 379–390. o. North-Holland, 1984. 502

[219] A. Iványi. Performance bounds for simple bin packing algorithms. *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica*, 5:77–82, 1984. 502

[220] A. Iványi. Tight worst-case bounds for bin packing algorithms. In *Theory of Algorithms (Pécs, 1984)*, *Colloquia of Mathematical Society János Bolyai* 44. kötete, 233–240. o. North-Holland, 1985. 502

[221] A. Iványi. *Párhuzamos algoritmusok* (*Parallel Algorithms*). ELTE Eötvös Kiadó, 2003. 9, 266

[222] A. Iványi, R. Szmeljánszkij. *Elements of Theoretical Programming* (*in Russian*). Moszkvai Állami Egyetem, 1985. 502

[223] K. Iwama, S. Tamaki. Improved upper bounds for 3-SAT. Technical Report TR03-053, Electronic Colloquium on Computational Complexity, 2003. 159

[224] J. R. Jackson. Research Report 43, Management Science Research Project, University of California, 1955. 414

[225] J. R. Jackson. An extension of Johnson's results on job shop scheduling. *Naval Research Logistics Quaterly*, 3:201–224, 1956. 415

[226] A. Jagota, R. B. Lyngso, C. N. S. Pedersen. Comparing a hidden Markov model and a stochastic context-free grammar. *Lecture Notes in Computer Science* 2149. kötete, 69–84. o. Springer-Verlag, 2001. 578

[227] R. Jain, S. Routhier. Packet trains: Measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communication*, 4:986–995, 1986. 221

[228] J. Jaja. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992. 266

[229] M. R. Jerrum. The complexity of finding minimum-length generator sequences. *Theoretical Computer Science*, 36:265–289, 1986. 578

[230] P. Jiménez, F. Thomas, C. Torras. 3D collision detection: A survey. *Computers and Graphics* 25(2):269–285, 2001. 683

[231] D. Johnson. Near-optimal bin packing algorithms, 1973. PhD értekezés. 502

[232] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quaterly*, 1:61–68, 1954. 415

[233] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, R. L. Graham. Worst-case performance-bounds for simple one-dimensional bin packing algorithms. *SIAM Journal on Computing*, 3:299–325, 1974. 502

[234] G. A. Jones, J. Jones. *Information and Coding Theory*. Springer-Verlag, 2000. 221

[235] K. Jones. *Consultant's Guide to COMNET III*. CACI Product Company, 1997. 221

[236] S. Kakutani. A generalization of Brouwer's fixed point theorem. *Duke Mathematical Journal*, 8:457–459, 1941. 359

[237] M. Kandemir, J. Ramanujam, A. Choudhary. Compiler algorithms for optimizing locality and parallelism on shared and distributed-memory machines. *Journal of Parallel and Distributed Computing*, 60:924–965, 2000. 267

[238] S. Karamardian. The nonlinear complementarity problems with applications. I, II. *Journal of Optimization Theory and Applications*, 4:87–98 and 167–181, 1969. 360

[239] Z. Karian, A. Starrett. Use of symbolic computation in probability and statistics. In Z. Karian (szerkesztő), *Symbolic Computation in Undergraduate Mathematics Education*, number24 in Notes of Mathematical Association of America. Mathematical Association of America, 1992. 93

[240] S. Karlin, M. T. Taylor. *A First Course in Stochastic Processes*. Academic Press, 1975. 684

[241] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher (szerkesztők), *Complexity of Computer Computations*, 85–103. o. Plenum Press, 1972. 158

[242] R. M. Karp, R. E. Miller, S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM* 14:563–590, 1967. 310

[243] Z. Kása. *Combinatorică cu aplicaţii* (*Combinatorics with Applications*). Presa Universitară Clujeană, 2003. 37

[244] I. Kátai. *Szimuláció (Simulation)*. Tankönyvkiadó, 1981. 221

[245] J. Köbler, U. Schöning, S. Toda, J. Torán. Turing machines with few accepting computations and low sets for PP. *Journal of Computer and System Sciences*, 44(2):272–286, 1992. 159

[246] J. Köbler, U. Schöning, J. Torán. Graph isomorphism is low for PP. *Computational Complexity*, 2:301–330, 1992. 123, 159

[247] J. Köbler, U. Schöning, J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser, 1993. 124, 159

[248] A. Kóczy, K. Kondorosi. *Operációs rendszerek – mérnöki megközelítésben (magyarul: Operating Systems - An Engineering Approach)*. Panem, 1999. 502

[249] J. Ködmön. *Kriptográfia: Az informatikai biztonság alapjai, a Pgp kriptorendszer használata (Cryptography: Bases of the Security of Computer Systems. Use of Cryptosystem Pgp)*. Computerbooks, 2002. 124

[250] J. Kececioglu, H. Lenhof, K. Mehlhorn, P. Mutzel, K. Reinert, M. Vingron. A polyhedral approach to sequence alignment problems. *Discrete Applied Mathematics*, 104((1-3)):143–186, 2000. 576

[251] K. Kennedy, R. Allen. *Optimizing Compilers for Modern Architectures*. Morgan Kaufman Publishers, 2001. 267

[252] B. Kernighan, R. Pike. *The UNIX programming Environment*. Prentice Hall, 1984 (Magyarul: *A UNIX operációs rendszer*., Műszaki Könyvkiadó, 1987 és 1999). 502

[253] A. Kertész. *A térinformatika és alkalmazásai (Geoinformatics and it Applications)*. Holnap Kiadó, 1997. 716

[254] B. Kiss, A. Krebsz. *Játékelmélet (Game Theory)*. Széchenyi István Főiskola, 1999. 360

[255] S. Kleiman, D. Shah, B. Smaalders. *Programming with Threads*. Prentice Hall, 1996. 267

[256] L. Kleinrock. *Queueing Systems*. John Wiley & Sons, 1975 (Magyarul: *Sorbanállás – kiszolgálás: Bevezetés a tömegkiszolgálási rendszerek elméletébe*, http://www.muszakikiado.hu/kezdolap.php Könyvkiadó, 1979). 221

[257] B. Knudsen, J. Hein. Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Researchs*, 31(13):3423–3428, 2003. 577

[258] D.E. Knuth. *Fundamental Algorithms*, *The Art of Computer Programming* 1. kötete. Addison-Wesley, 1968 (3., javított kiadás 1997. Magyarul: *A számítógép-programozás művészete. 1. kötet. Alapvető algoritmusok*, Műszaki Könyvkiadó, 1993, 2. kiadás.). 9, 37, 93

[259] D. E. Knuth. *Seminumerical Algorithms*, *The Art of Computer Programming* 2. kötete. Addison-Wesley, 1969 (3. javított kiadás 1998. Magyarul: *A számítógép-programozás művészete. 2. kötet. Szeminumerikus algoritmusok*, Műszaki Könyvkiadó, 1993, 2. kiadás.). 9, 93

[260] D. E. Knuth. *Sorting and Searching*, *The Art of Computer Programming* 3. kötete. Addison-Wesley, 1973 (3., javított kiadás 1997. Magyarul: *A számítógép-programozás művészete. 3. kötet. Keresés és rendezés*, Műszaki Könyvkiadó, 1994, 2. kiadás.). 9, 93

[261] D. E. Knuth, J. H. Morris Jr., V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977. 576

[262] C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. Steele Jr., M. E. Zosel. *The High Performance Fortran Handbook*. The MIT Press, 1994. 267

[263] A. Kovács. Komputer algebra a tudományokban és a gyakorlatban (Computer algebra in science and practice). *Alkalmazott Matematikai Lapok*, 18:181–202, 1994-98. 93

[264] A. Kovács. Computer algebra: Impact and perspectives. *Nieuw Archief voor Wiskunde*, 17(1):29–55, 1999. 93

[265] M. Kovács. *Nemlineáris programozás* (*Nonlinear Programming*). Typotex, 1997. 360

[266] G. Krammer. Notes on the mathematics of the PHIGS output pipeline. *Computer Graphics Forum*, 8(8):219–226, 1989. 683

[267] H. W. Kuhn, A. Tucker (szerkesztők). *Contributions to the Theory of Games. II*. Princeton University Press, 1953. 359

[268] H. T. Kung, C. E. Leiserson. Systolic arrays (for VLSI). In I. S. Duff, G. W. (szerkesztők), *Sparse Matrix Proceedings*, 256–282. o. SIAM, 1978. 310

[269] S. Kurtz, B. Balkenhol. Space efficient linear time computation of the Burrows and Wheeler transformation. In I. Althöfer, N. Cai, L. Dueck, M. Khachatrian, A. Pinsker, I. Sarközy, I. Wegener, Z. Zhang (szerkesztők), *Numbers, Information and Complexity*, 375–383. o. Kluwer Academic Publishers, 2000. 454

[270] R. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, 1975. 159

[271] R. Ladner, N. Lynch, A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1(2):103–124, 1975. 158

[272] B. Lageweg, J. K. Lenstra, A. Rinnoy Kan. Job-shop scheduling by implicit enumeration. *Management Science*, 24:441–450, 1977. 414, 415

[273] B. Lageweg, J. K. Lenstra, A. Rinnoy Kan. A general bounding scheme for the permutation flow-shop problem. *Operations Research*, 26:53–67, 1978. 414

[274] T. Lai, S. Sahni. Anomalies in parallel branch and bound algorithms. *Communications of ACM*, 27(6):594–602, 1984. 502

[275] J. Lamperti. *Stochastic Processes*. Springer-Verlag, 1972. 684

[276] G. Lancia. Integer programming models for computational biology problems. *Journal of Computer Science and Technology*, 19(1):60–77, 2004. 576

[277] G. Landau, U. Vishkin. Eficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986. 576

[278] G. Langdon. An introduction to arithmetic coding. *IBM Journal of Research and Development*, 28:135–149, 1984. 454

[279] R. Laurini, D. Thompson. *Fundamentals of Spatial Information Systems*. Academic Press, 1992. 716

[280] A. Law, W. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 3. kiadás, 1999. 221.

[281] E. L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19:544–546, 1973. 414.

[282] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan, D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S. C. Graves, A. Kan, P. Zipkin, P. H.szerkesztők), *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*, 445–522. o. Elsevier, 1993. 413, 414.

[283] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubes*. Morgan Kaufman Publishers, 1992. 266.

[284] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Algorithms and VSLI*. Morgan Kaufman Publishers, 2001. 266.

[285] W. Leland, M. Taqqu, W. Willinger, D. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1), 1994. 221.

[286] W. Leland, M. Taqqu, W. Willinger, D. Wilson. Statistical analysis and stochastic modeling of self-similar data-traffic. In J. Labetoulle, J. W. Roberts (szerkesztők), *The Fundamental Role of Teletraffic in the Evolution of Telecommunications Networks*. Elsevier Science B. V., 1994. 221.

[287] W. Leland, M. Taqqu, D. Wilson. On the self-similar nature of ethernet traffic. *Computer Communication Reviews*, 23, 1993. 221.

[288] A. Lenstra, H. Lenstra. *The Development of the Number Field Sieve*, *Lecture Notes in Mathematics* 1554. kötete. Springer-Verlag, 1993. 123.

[289] C. Leopold. *Parallel and Distributed Computing*. John Wiley & Sons, Copyrights 2001. 266, 267.

[290] B. Lewis, D. J. Berg. *Multithreaded Programming with Phtreads*. Prentice Hall, 1998. 267.

[291] T. Linder, G. Lugosi. *Bevezetés az információelméletbe (Introduction to Information Theory)*. Tankönyvkiadó, 1990. 455.

[292] M. Listanti, V. Eramo, R. Sabella. Architectural and technological issues for future optical internet networks. *IEEE Communications Magazine*, (9):82–92, 2000. 221.

[293] A. Álmos, S. Győri, G. Horváth, A. Várkonyiné Kóczy. *Genetikus algoritmusok (Genetic Algorithms)*. Typotex, 2002. 604.

[294] P. Longley, D. Maguire, M. Goodschild, D. Rhind. *Geographic Information Systems and Science*. John Wiley & Sons, 2001. 716.

[295] L. Lovász, P. Gács. *Algoritmusok (Algorithms)*. Műszaki Könyvkiadó és Tankönyvkiadó, 1978 és 1987. 9.

[296] L. Lovász. *Combinatorial Problems and Exercises*. Akadémiai Kiadó, 1979 (Magyarul: *Kombinatorikai problémák és feladatok*, Typotex, 1999). 37.

[297] L. Lovász. *Algoritmusok bonyolultsága (Complexity of Algorithms)*. ELTE TTK, 1990. 159.

[298] C. Lucchesi. Candidate keys for relations. *Journal of of Computer and System Sciences*, 17(2):270–279, 1978. 534.

[299] G. Lunter, I. Miklós, A. Drummond, J. L. Jensen, J. Hein. Bayesian phylogenetic inference under a statistical indel model. *Lecture Notes in Bioinformatics*, 2812:228–244, 2003. 577.

[300] G. Lunter, I. Miklós, Y., J. Hein. An efficient algorithm for statistical multiple alignment on arbitrary phylogenetic trees. *Journal of Comp. Biology*, 10(6):869–889, 2003. 577.

[301] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufman Publisher, 2001 (Ötödik kiadás. Magyarul: *Osztott algoritmusok*. Kiskapu Kiadó, 2002). 9, 311.

[302] R. Lyngso, C. N. S. Pedersen. RNA pseudoknot prediction in energy based models. *Journal of Comp. Biology*, 7(3/4):409–428, 2000. 578.

[303] R. Lyngso, M. Zuker, C. Pedersen. Fast evaluation of internal loops in RNA secondary structure prediction. *Bioinformatics*, 15(6):440–445, 1999. 578.

[304] D. J. Maguire, M. Goodschild, D. Rhind. *Geographical Information Systems*. Longman, 1991. 716.

[305] D. Maier. Minimum covers in the relational database model. *Journal of the ACM*, 27(4):664–674, 1980. 534.

[306] D. Maier, A. O. Mendelzon, Y. Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4(4):455–469, 1979. 534.

[307] B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freeman, 1982. 221.

[308] B. Mandelbrot, J. W. Van Ness. Fractional brownian motions, fractional noises and applications. *SIAM Review*, 10:422–437, 1968. 221

[309] O. Mangasarian, H. Stone. Two-person zero-sum games and quadratic programming. *Journal of Mathematical Analysis and its Applications*, 9:348–355, 1964. 360

[310] A. S. Manne. On the job-shop scheduling problem. *Operations Research*, 9:219–223, 1960. 414

[311] B. Martos. *Nonlinear Programming Theory and Methods*. Akadémiai Kiadó, 1975. 360

[312] R. Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–132, 1979. 159

[313] R. L. Mattson, J. Gecsei, D. R. Slutz, I. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970. 501

[314] E. A. Maxwell. *Methods of Plane Projective Geometry Based on the Use of General Homogenous Coordinates*. Cambridge University Press, 1946. 683

[315] E. A. Maxwell. *General Homogenous Coordinates in Space of Three Dimensions*. Cambridge University Press, 1951. 683

[316] McAfee. Sniffer technologies. http://www.nai.com/us/index.asp, 2004. 221

[317] J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29:1105–1119, 1990. 578

[318] G. B. McMahon. Optimal production schedules for flow shops. *Canadian Operation Research Society Journal*, 7:141–151, 1969. 414

[319] R. McNaughton. Scheduling with deadlines loss functions. *Management Science*, 6:1–12, 1959. 414

[320] B. Melamed. An overview of tes processes and modeling methodology. In L. Donatiello, A. R. Nelson (szerkesztők), *Models and Techniques for Performance Evaluation of Computer and Communications Systems*, Lecture Notes in Computer Science, 359–393. o. Springer-Verlag, 1993. 221

[321] A. Meskó. *A digitális szeizmikus feldolgozás alapjai (Introduction to Processing of Digital Seismic Data)*. Tankönyvkiadó, 1978. 716

[322] A. Meskó. *Geofizikai adatfeldolgozás (Processing of Geophysical Data)*. Tankönyvkiadó, 1983. 716

[323] A. Meyer, L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*, 125–129. o., 1972. 159

[324] I. Meyer, R. Durbin. Comparative ab initio prediction of gene structures using pair HMMs. *Bioinformatics*, 18(10):1309–1318, 2002. 576, 577

[325] I. M. Meyer, R. Durbin. Gene structure conservation aids similarity based gene prediction. *Nucleic Acids Research*, 32(2):776–783, 2004. 577

[326] D. Micciancio, S. Goldwasser. *Complexity of Lattice Problems: A Cryptographic Perspective*, *The Kluwer International Series in Engineering and Computer Science* 671. kötete. Kluwer Academic Publishers, 2002. 123

[327] R. E. Mickens. *Difference Equations. Theory and Applications*. Van Nostrand Reinhold, 1990. 37

[328] M. E. Mignotte. *Mathematics for Computer Algebra*. Springer, 1992. 93

[329] Sz. Mihnovszkij, N. Shor. Estimation of the page fault number in paged memory (in russian). *Kibernetika (Kiev)*, 1(5):18–20, 1965. 501

[330] G. L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and Systems Sciences*, 13(3):300–317, 1976. 123

[331] W. Miller, E. Myers. A file comparison program. *Software – Practice and Experience*, 15(11):1025–1040, 1985. 576

[332] W. Miller, E. W. Myers. Sequence comparison with concave weighting functions. *Bulletin of Mathematical Biology*, 50:97–120, 1988. 575

[333] H. Mills. Equilibrium points of finite games. *SIAM Journal of Applied Mathematics*, 8:397–402, 1976. 360

[334] B. E. Mishra. *Algorithmic Algebra*. Springer, 1993. 93

[335] S. Molnár, A. Vidács, A. Nilsson. Bottlenecks on the way towards characterization of network traffic: Estimation and interpretation of the hurst parameter. http://hsnlab.ttt.bme.hu/molnar (Conference papers), 1997. 221

[336] B. Monien, E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985. 158, 159

[337] B. Morgenstern. DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15:211–218, 1999. 577

[338] B. Márkus. *Térinformatika Magyarországon'94*. Technológia Transzfer Centrum, 1994. 716

[339] G. Márton. Sugárkövető algoritmusok átlagos bonyolultságának vizsgálata, 1995. Kandidátusi értekezés. 683

[340] M. Márton, J. Paksi, B. Márkus. *Térinformatikai alapismeretek (Elements of Geoinformatics)*. Technológia Transzfer Centrum, 1994. 716

[341] J. Mészáros. *Játékelmélet (Game Theory)*. Gondolat Kiadó, 2004. 360

[342] J. Nash. Noncooperative games. *Annals of Mathematics*, 54:286–295, 1951. 359

[343] M. Nebel. Identifying good predictions of rna secondary structure. In R. B. Altman, A. K. Dunker, L. Hunter, T. E. Klein (szerkesztők), *Pacific Symposium on Biocomputing*, 9. kötet, 423–434. o. PSB Online, http://psb.stanford.edu/psb-online/, 2004. 578

[344] S. N. Needleman, C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970. 575

[345] M. Nelson, J. L. Gailly. *The Data Compression Book*. M&T Books, 1996. 455

[346] J. Neumann, O. Morgenstern. *Theory of Games and Economical Behaviour*. Princeton University Press, 1947 (2. kiadás). 360

[347] M. F. Neuts. A versatile markovian point process. *Journal of Applied Probability*, 18:764–779, 1979. 221

[348] M. F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Marcel Dekker, 1989. 221

[349] M. E. Newell, R. G. Newell, T. L. Sancha. A new approach to the shaded picture problem. In *Proceedings of the ACM National Conference*, 443–450. o., 1972. 684

[350] H. Nikaido, K. Isoda. Note on noncooperative games. *Pacific Journal of Mathematics*, 5:807–815, 1955. 360

[351] S. Oaks, H. Wong. *Java Threads*. O'Reilly, 1999. 267

[352] A. Odlyzko. *Applications of Symbolic Mathematics to Mathematics*. Kluwer Academic Publishers, 1985. 93

[353] K. Okuguchi. *Expectation and Stability of Oligopoly Models*. Springer, 1976. 360

[354] K. Okuguchi, F. Szidarovszky. *The Theory of Oligopoly with Multi-Product Firms*. Springer, 1999 (2. kiadás). 360

[355] OpenMP Website. http://www.openmp.org, 2004. 267

[356] OPNET. Opnet Modeler Documentation. www.opnet.com, 2004. 221

[357] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987. 683

[358] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994 (Magyarul: *Kiszámítási bonyolultság*, Novadat, 1999). 124, 158, 159

[359] C. Partridge. The end of simple traffic models. *IEEE Network*, (9), 1993. Editor's Note. 221

[360] R. Pasco. *Source Coding Algorithms for Fast Data Compression*. PhD thesis, Stanford University, 1976. 454

[361] R. Paturi, P. Pudlák, M. Saks, F. Zane. An improved exponential-time algorithm for k-SAT. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, 628–637. o. IEEE Computer Society Press, 1998. 159

[362] R. Pavelle, M. Rothstein. Computer algebra. *Scientific American*, 245(12):102–113, 1981. 93

[363] V. Paxson, S. Floyd. Wide-area traffic: The failure of Poisson modelling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995. 221

[364] J. S. Pedersen, J. Hein. Gene finding with a hidden Markov model of genome structure and evolution. *Bioinformatics*, 19(2):219–227, 2003. 577

[365] I. Peák. *Bevezetés az automaták elméletébe. Az automaták mint információátalakító rendszerek 1. (Introduction to the Theory of Automata. Automata an Systems for the Transformation of the Information)*. Tankönyvkiadó, 1977. 501

[366] G. Perlman. HCI bibliography, 2004. 604

[367] S. Petrov. Finite axiomatization of languages for representation of system properties. *Information Sciences*, 47:339–372, 1989. 534

[368] P. A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. The MIT Press, 2000. 576

[369] G. F. Pfister. *In Search of Clusters*. Prentice Hall, 1998 (2. kiadás). 267

[370] J. Piehler. Ein Beitrag zum Reihenfolgeproblem, Unternehmensforschung. *Unternehmerforschung*, 4:138–142, 1960. 414

[371] M. Pinedo. *Scheduling: Theory, Algorithms and Systems*. Pearson Education, 2001 (2. kiadás). 415

[372] M. Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer-Verlag, 2004 (megjelenő-ben). 415

[373] N. Pisanti, M. Sagot. Further thoughts on the syntenic distance between genomes. *Algorithmica*, 34(2):157–180, 2002. 578

[374] J. Podani. *Bevezetés a többváltozós biológiai adatfeltárás rejtelmeibe*. Scientia, 1997. 576, 604

[375] J. M. Pollard. Theorems on factorization and primality testing. *Proceedings of the Cambridge Philosophical Society*, 76:521–528, 1974. 123

[376] F. P. Preparata, M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985. 683

[377] A. Prékopa. *Valószínűségelmélet műszaki alkalmazásokkal*. Műszaki Könyvkiadó, 1962. 221

[378] T. Pupko, I. Peer, R. Shamir, D. Graur. A fast algorithm for joint reconstruction of ancestral amino acid sequences. *Molecular Biology and Evolution*, 17:890–896, 2000. 577

[379] P. Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. In *Proceedings of the 11th Annual International Symposium on Computer Architecture*, 208–214. o., 1984. 310

[380] M. Rabi, A. Sherman. An observation on associative one-way functions in complexity theory. *Information Processing Letters*, 64(5):239–244, 1997. 123

[381] M. O. Rabin. Probabilistic algorithms for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980. 123

[382] R. Rao, J. Rothe, O. Watanabe. Upward separation for FewP and related classes. *Information Processing Letters*, 52(4):175–180, 1994 (Corrigendum appears in the same journal,74(1–2):89, 2000). 159

[383] S. K. Rao. Regular iterative algorithms and their implementations on processor arrays. Doktori értekezés, Stanford University, 1985. 310

[384] P. Resnick, H. R. Varian. Recommender Systems. *Communications of the ACM*, 40(3):56–58, 1997. 604

[385] J. Richards. *Remote Sensing Digital Image Analysis*. Springer-Verlag, Australia, 1986. 716

[386] P. Rigaux, M. Scholl, A. Voisard. *Spatial Databases with Application to GIS*. Morgan Kaufman Publisher, 2001. 716

[387] J. Rissanen, G. Arithmetic coding. *IBM Journal of Research and Development*, 23:149–162, 1979. 454

[388] J. J. Rissanen. Generalized Kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20:198–203, 1976. 454

[389] J. Ritt. *Integration in Finite Terms*. Columbia University Press, 1948. 92

[390] E. Rivas, S. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology*, 285(5):2053–2068, 1999. 578

[391] R. L. Rivest, A. Shamir, L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. Lásd még az U.S. Patent 4 405 829 számú szabadalmat. 123

[392] L. Rónyai, G. Ivanyos, R. Szabó. *Algoritmusok* (*Algorithms*). Typotex, 1999. 9, 159

[393] A. Rényi. *Probability Theory*. Akadémiai Kiadó/North Holland Publ. House, 1970 (Magyarul: *Valószínűségelmélet*, Tankönyvkiadó, Budapest, 1973). 221

[394] J. Robinson. An iterative method of solving a game. *Annals of Mathematics*, 154:296–301, 1951. 360

[395] D. F. Rogers, J. Adams. *Mathematical Elements for Computer Graphics*. McGraw-Hill, 1989. 683

[396] S. H. Roosta. *Parallel Processing and Parallel Algorithms*. Springer-Verlag, 1999. 266, 502

[397] J. Rosen. Existence and uniqueness of equilibrium points for concave n-person games. *Econometrica*, 33:520–534, 1965. 360

[398] J. Rothe. Some facets of complexity theory and cryptography: A five-lecture tutorial. *ACM Computing Surveys*, 34(4):504–549, 2002. 123, 124

[399] J. Rothe. A promise class at least as hard as the polynomial hierarchy. *Journal of Computing and Information*, 1(1):92–107, 1995. 159

[400] J. Rothe. *Complexity Theory and Cryptology. An Introduction to Cryptocomplexity*. EATCS Texts in Theoretical Computer Science. Springer-Verlag, 2005. 158, 159

[401] M. H. Rothkopf. Scheduling independent tasks on parallel machines. *Management Science*, 12:437–447, 1966. 414

[402] R. O. Roundy, W. L. Maxwell, Y. T. Herer, S. R. Tayur, A. W. Getzler. A price-directed approach of production operations. *IIE Transactions*, 23:149–160, 1991. 415

[403] B. Roy, B. Sussmann. Les problémes d'ordonnuncement avec contraintes disjonctives. Note DS no. 9 bis, SEMA, 1964. 414

[404] S. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23:116–127, 1976. 414

[405] S. Sahni. Preemptive scheduling with due dates. *Operations Research*, 27:929–934, 1979. 414

[406] A. Sali, Sr., A. Sali. Generalized dependencies in relational databases. *Acta Cybernetica*, 13:431–438, 1998. 535

[407] A. Salomaa. *Public-Key Cryptography*, *EATCS Monographs on Theoretical Computer Science* 23. kötete. Springer-Verlag, 2. kiadás, 1996. 122, 123

[408] D. Salomon. *Data Compression*. Springer-Verlag, 2004 (3. kiadás). 455

[409] J. Sameith. On the generation of alternative solutions for discrete optimization problems with uncertain data – an experimental analysis of the penalty method. http://www.minet.uni-jena.de/Math-Net/reports/shadows/04-01report.html, 2004. 604

[410] H. Samet. The quadtree and related hierarchical data structures. *ACM Computer Surveys*, 16(2):187–260, 1984. 716

[411] D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal of Appliedl Mathematics*, 28:35–42, 1975. 575

[412] L. A. Santaló. *Integral Geometry and Geometric Probability*. Addison-Wesley, 1976. 684

[413] K. Sayood. *Introduction to Data Compression*. Morgan Kaufman Publisher, 2000 (2. kiadás). 455

[414] U. Schöning. A low and a high hierarchy within NP. *Journal of Computer and System Sciences*, 27:14–28, 1983. 159

[415] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37:312–323, 1987. 159

[416] U. Schöning. A probabilistic algorithm for k-SAT based on limited local search and restart. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, 410–414. o. IEEE Computer Society Press, 1999. 158, 159

[417] U. Schöning. *Algorithmik*. Spektrum Akademischer Verlag, 2001. 160

[418] U. Schöning. *Ideen der Informatik*. Oldenbourg Verlag, 2002. 158

[419] L. Schrage. Solving resource-constrained network problems by implicit enumeration. *Operations Research*, 18:263–278, 1970. 414

[420] L. Schrage. Obtaining optimal solutions to resource constrained network scheduling problems, 1971. Publikálatlan kézirat. 414

[421] R. Sedgewick, P. Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Wesley, 1996. 37

[422] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, 1(1):51–64, 1991. 683

[423] A. Selman. Polynomial time enumeration reducibility. *SIAM Journal on Computing*, 7(4):440–457, 1978. 158

[424] C. Semple, M. Steel. *Phylogenetics*. Number24 in Oxford Lecture Series in Mathematics and Its Applications. Oxford Press, 2003. 576

[425] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992. 123

[426] C. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):657–715, 1949. 123

[427] H. N. Shapiro. Note on a computation method in the theory of games. *Communications on Pure and Applied Mathematics*, 11:587–593, 1958. 360

[428] B. Sharp. Implementing subdivision theory. *Game Developer*, 7(2):40–45, 2000. 683

[429] B. Sharp. Subdivision Surface theory. *Game Developer*, 7(1):34–42, 2000. 683

[430] I. Shindyalov, P. Bourne. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Engineering*, 11(9):739–747, 1998. 577

[431] A. Silberschatz, P. Galvin, G. Gagne. *Applied Operating System Concepts*. John Wiley & Sons, 2000. 501

[432] D. Sima, T. Fountain, P. Kacsuk. *Advanced Computer Architectures: a Design Space Approach*. Addison-Wesley Publishing Company, 1998 (2. kiadás. Magyarul: *Korszerű számítógéparchitektúrák tervezésitér-megközelítésben*. Szak Kiadó, 1998). 221, 266, 311

[433] S. Singh. *The Code Book. The Secret History of Codes and Code Breaking*. Fourth Estate, 1999 (Magyarul: *Kódkönyv. A rejtjelezés és rejtjelfejtés története*. Park Könyvkiadó, 2002). 123, 124

[434] W. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quaterly*, 3:59–66, 1956. 414, 415

[435] M. Snir, S. W. Otto, S. D. W. Huss-Lederman, D. W. Walker, J. Dongarra. *MPI: The Complete Reference*. The MIT Press, 1996. 267

[436] R. Solovay, V. Strassen. A fast Monte Carlo test for primality. *SIAM Journal on Computing*, 6:84–85, 1977. Erratum appears in the same journal, 7(1):118, 1978. 123

[437] G. Stephen. *String searching algorithms*, *Lecture Notes Series on Computing* 3. kötete. World Scientific, Singapore, 1994. 576

[438] R. Stephens. *Visual Basic Graphics Programming*. John Wiley & Sons, 2000. 716

[439] D. Stinson. *Cryptography: Theory and Practice*. CRC Press, 2002 (2. kiadás). 123

[440] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1977. 159

[441] G. Stoyan (szerkesztő). *MATLAB 4. és 5. verzió*. Typotex, 1999. 765

[442] I. Sutherland, G. Hodgeman. Reentrant polygon clipping. *Communications of the ACM*, 17(1):32–42, 1974. 683

[443] I. E. Sutherland, R. Sproull, R. Schumacker. A characterization of ten hidden-surface algorithms. *Computing Surveys*, 6(1):1–55, 1974. 684

[444] L. Szécsi. An effective kd-tree implementation. In J. Lander (szerkesztő), *Graphics Programming Methods*. Charles River Media, 2003. 684

[445] F. Szidarovszky, C. Chiarella. Dynamic oligopolies, stability and bifurcation. *Cubo Mathemática Educational*, 3(2):267–284, 2001. 360

[446] F. Szidarovszky, S. Molnár. *Játékelmélet műszaki alkalmazásokkal: Többcélú programozás, klasszikus és differenciáljátékok (Game Theory with Technical Applications: Programming with Multiple Aims, Classical and Differential Games)*. Műszaki Könyvkiadó, 1986. 360

[447] F. Szidarovszky, S. Yakowitz. *Principles and Procedures of Numerical Analysis*. Plenum Press, 1998. 360

[448] L. Szirmay-Kalos. *Számítógépes grafika (Computer Graphics)*. ComputerBooks, 1999. 684

[449] L. Szirmay-Kalos (szerkesztő). *Theory of Three Dimensional Computer Graphics*. Akadémiai Kiadó, 1995. 684

[450] L. Szirmay-Kalos, Gy. Antal, F. Csonka. *Háromdimenziós grafika, animáció és játékfejlesztés + CD (Three Dimensional Graphics, Animation and Game Development)*. Computerbooks, 2003. 683, 684

[451] L. Szirmay-Kalos, G. Márton. Worst-case versus average-case complexity of ray-shooting. *Computing*, 61(2):103–131, 1998. 683

[452] J. Sztrik. Bevezetés a sorbanállási elméletbe és alkalmazásaiba. http://irh.inf.unideb.hu/user/jsztrik/, 2004. 221

[453] W. Szwarc. Optimal elimination methods in the $m \times n$ sequencing problem. *Operations Research*, 21:1250–1259, 1973. 414

[454] W. Szwarc. Dominance conditions for the three-machine flow-shop problem. *Operations Research*, 26:203–206, 1978. 414

[455] W. Szwarc. Elimination methods in the $m \times n$ sequencing problem. *CWI Quarterly*, 18:295–305, 1971. 414

[456] A. S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 2001. 267

[457] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 2004 (4. kiadás. Magyarul: *Számítógép-hálózatok*. Panem, 2004). 221

[458] A. S. Tanenbaum, M. van Steen. *Distributed systems. Principles and Paradigms*. Prentice Hall, 2002 (Magyarul: *Elosztott rendszerek*. Panem, 1999). 267

[459] A. S. Tanenbaum, A. Woodhull. *Operating Systems. Design and Implementation*. Prentice Hall, 1997 (Magyarul: *Operációs rendszerek*. Panem, 1999). 501, 502

[460] M. Taqqu, W. Teverovsky, W. Willinger. Estimators for long-range dependence: an empirical study. *Fractals*, 3(4):785–788, 1995. 221

[461] A. Tarski. A lattice-theoretical fixpoint theorem and its application. *Pacific Journal of Mathematics*, 5:285–308, 1955. 359

[462] D. S. Taubman, M. W. Marcelin. *JPEG 2000 – Image Compression, Fundamentals, Standards and Practice*. Society for Industrial and Applied Mathematics, 1983. 455

[463] J. Teich, L. Thiele. Control generation in the design of processor arrays. *International Journal of VLSI and Signal Processing*, 3(2):77–92, 1991. 310

[464] B. Thalheim. *Dependencies in Relational Databases*. Teubner Verlagsgesellschaft, 1991. 535

[465] J. D. Thompson, D. G. Higgins, T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994. 575

[466] I. Tinoco, O., Uhlenbeck M. Levine. Estimation of secondary structure in ribonucleic acids. *Nature*, 230:362–367, 1971. 577

[467] Top 500 Supercomputer Sites. http://www.top500.org, 2004. 266

[468] Trusoft Intl Inc. *Benoit 1.1*. Trusoft Intl Inc., 2004. 221

[469] D. M. Tsou, P. C. Fischer. Decomposition of a relation scheme into Boyce–Codd normal form. *SIGACT News*, 14(3):23–29, 1982. 534

[470] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, ser. 2*, 2:230–265, 1936 (Correction, *ibid*, vol. 43, pp. 544–546, 1937). 158

[471] Y. Uemura, A. Hasegawa, Y. Kobayashi, T. Yokomori. Tree adjoining grammars for RNA structure prediction. *Theoretical Computer Science*, 210:277–303, 1999. 578

[472] J. J. Uhl. MATHEMATICA and Me. *Notices of AMS*, 35:1345–1345, 1988. 93

[473] J. D. Ullman. *Principles of Database and Knowledge Base Systems. Vol. 1*. Computer Science Press, 2. kiadás, 1989. 535

[474] J. D. Ullman, J. . *A First Course in Database Systems*. Prentice Hall, 1997 (Magyarul: *Adatbázisrendszerek. Alapvetés*. Panem, Budapest, 1998). 535

[475] L. G. Valiant. The relative complexity of checking and evaluating. *Information Processing Letters*, 5(1):20–23, 1976. 159

[476] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990. 267

[477] L. Varga. *Rendszerprogramok elmélete és gyakorlata (Theory and Practice of System Programs)*. Akadémiai Kiadó, 1980. 502

[478] N. J. Vilenkin. *Kombinatorika (oroszul)*. Mir, 1969 (Angolul: *Combinatorial Mathematics*, Mir, 1972; magyarul: *Kombinatorika*, Műszaki Könyvkiadó, 1987, 2. kiadás). 37

[479] B. Vizvári. *Egészértékű programozás*. ELTE TTK, 1990. 414

[480] J. von zur Gathen. *Modern Computer Algebra*. Cambridge University Press, 2003. 92, 93

[481] T. Várady, R. R. Martin, J. Cox. Reverse engineering of geometric models - an introduction. *Computer-Aided Design*, 29(4):255–269, 1997. 683

[482] K. Wagner, G. Wechsung. *Computational Complexity*. D. Reidel Publishing Company, 1986 (and Kluwer Academic Publishers, 2001). 158

[483] G. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34:30–44, 1991. 455

[484] L. Wang, T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computatonal Biology*, 1:337–348, 1994. 575

[485] J. Warren, H. Weimer. *Subdivision Methods for Geometric Design: A Constructive Approach*. Morgan Kaufmann Publishers, 2001. 683

[486] M. S. Waterman. Efficient sequence alignment algorithms. *Journal of Theoretical Biology*, 108:333–337, 1984. 575

[487] M. S. Waterman, T. F. Smithand, W. A. Beyer. Some biological sequence metrics. *Advances in Mathematics*, 20:367–387, 1976. 575

[488] D. Watkins. Bulge exchanges in algorithms of QR type. *SIAM Journal on Matrix Analysis and Application*, 19(4):1074–1096, 1998. 765

[489] Web Dictionary of Cybernetics, Systems. http://pcp.wub.ac.be/ASC.html, 2004. 221

[490] G. Wechsung. *Vorlesungen zur Komplexitätstheorie*, 32. kötet. B. G. Teubner Verlagsgesellschaft, 2000. 158

[491] D. Welsh. *Codes and Cryptography*. Oxford University Press, 1988. 454

[492] J. Wilkinson. Convergence of the LR, QR, and related algorithms. *The Computer Journal*, 8(1):77–84, 1965. 765

[493] F. M. J. Willems, Y. M. Shtarkov, T. J. Tjalkens. The context-tree weighting method: basic properties. *IEEE Transactions on Information Theory Theory*, 47:653–664, 1995. 454

[494] F. M. J. Willems, Y. M. Shtarkov, T. J. Tjalkens. The context-tree weighting method: basic properties. *IEEE Information Theory Society Newsletter*, 1:1 és 20–27, 1997. 454

[495] W. Willinger, V. Paxson. Discussion of „heavy tail modeling and teletraffic data” by S. R. Resnick. *The Annals of Statistics* 25(5):1805–1869, 1997. 221

[496] W. Willinger, M. Taqqu, R. Sherman, D. Wilson. Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5:71–86, 1997. 221

[497] W. Willinger, D. Wilson, W. Leland, M. Taqqu. On traffic measurements that defy traffic models (and vice versa): self-similar traffic modeling for high-speed networks. *Connections* 8(11):14–24, 1994. 221

[498] F. Winkler. *Polynomial Algorithms in Computer Algebra*. Springer-Verlag, 1990. 93

[499] I. H. Witten, R. M. Neal, J. G. Cleary. Arithmetic coding for sequential data compression. *Communications of the ACM*, 30:520–540, 1987. 454

[500] M. J. Wolfe. *High Performance Compilers for Parallel Computing*. Addison Wesley Longman, 1996. 267

[501] S. Wu, E.. W. Myers, U. Manber, W. Miller. An O(NP) sequence comparison algorithm. *Information Processing Letters*, 35(6):317–323, 1990. 576

[502] G. Wyvill, C. McPheeters, B. Wyvill. Data structure for soft objects. *The Visual Computer*, 4(2):227–234, 1986. 683

[503] D. M. Young. *Iterative Solution of Large Linear Systems*. Academic Press, 1971 (Magyarul: *Nagy lineáris rendszerek iterációs megoldása*. Műszaki Könyvkiadó, 1979). 765

[504] C. Yu, D. Johnson. On the complexity of finding the set of candidate keys for a given set of functional dependencies. In *Information Processing 74*, 580–583. o. North-Holland, 1974. 534

[505] S. Zachos, H. Heller. A decisive characterization of BPP. *Information and Control*, 69:125–135, 1986. 124

[506] B. Zalik, G. Clapworthy. A universal trapezoidation algorithms for planar polygons. *Computers and Graphics* 23(3):353–363, 1999. 683

[507] C. Zaniolo. Analysis and design of relational schemata for database systems. Technical Report UCLA–Eng–7669, Department of Computer Science, University of California at Los Angeles, 1976. 534

[508] C. Zaniolo. A new normal form for the design of relational database schemata. *ACM Transactions on Database Systems*, 7:489–499, 1982. 534

[509] E. Zehendner. *Entwurf systolischer Systeme: Abbildung regulärer Algorithmen auf synchrone Prozessorarrays*. B. G. Teubner Verlagsgesellschaft, 1996. 310

[510] J. Ziv, A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory Theory*, 23:337–343, 1977. 454

[511] J. Ziv, A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory Theory*, 24:530–536, 1978. 454

[512] S. I. Zuhovitsky, R. A. Polyak, M. E. Primak. Concave n-person games (numerical methods). *Ékonomika i Matematicheskie Methody*, 7:888–900, 1971 (oroszul). 360

[513] M. Zuker, D. Sankoff. RNA structures and their prediction. *Bulletin of Mathematical Biology*, 46:591–621, 1986. 578

[514] AND Software GmbH. Car routing program ”Route Germany”, 1997. 604

Tárgymutató

A, Á
AABB, *635*
abszolút hiba, 726*gy*
abszolút hibakorlát, *717*
abszolút optimális algoritmus, *234*
absztrakt számítógép, *lásd* számítási modell
ACE, *lásd* Alkalmazásjellemző környezet
adatábrázolás
  a komputeralgebrában, 40
Adatcsere diagram, *184*
adatfolyam, *284*
adatfolyam iránya, 287
adatfüggőség, 284
adatpárhuzamosság, *228*, 235
adatsebesség, *291*
adatszerkezet indexe, *287*
adatszerkezet vektor, *287*
additív metrika, *566*
affin függvény, *542*
affin transzformáció, *643*
aktív él lista, *674*
alakzat, *605*
alaphalmaz, *588*
algebrai
  bővítés, 76, 79, 80, 83, 86
  elem, 78
  számtest, 79, 81
algoritmus
  abszolút optimális, 234
  aszimptotikusan optimális, 234
  munkahatékony, 233
algoritmus stabilitása, *739*
Alkalmazásjellemző környezet, *182*
állandó kommunikáció, *237*
állapot-átmenetfüggvény, *471*
állásidő, 366
állományelhelyező algoritmus, *499*
alsó-hierarchia, 144, 149*áb*, *150*
általános kiválasztási feladat, 262
általános megoldás, *14*
általános párhuzamos gépek, 366
alternatívák, valódi, *581*
alternatív áram együttható, *450*
alternatív megoldások, 600
alulcsordulás, *433*
alulvágó szűrő, 705, 706*áb*
Amdahl-törvény, *229*
analitikus geometria, *605*
anomália, *480*, *481*, 501, 511, 519
  beillesztési, 512
  frissítési, 512
  redundancia, 512
  törlési, 512
anomálisa mértéke, *481*
áramlási problémához, 595
architektúra
  szisztolikus, 275
ARITMETIKAI-DEKÓDOLÓ, *431*
aritmetikai kódolás, *429*
ARITMETIKAI-KÓDOLÓ, *431*
aritmetikai túlcsordulás, 726*gy*
Armstrong-axiómák, *504*, 507, 511*gy*, 519
Armstrong-reláció, 532
árnyék, *645*
asszociativitás, 101, 116*gy*
aszimptotikusan azonos nagyságrend, 234
aszimptotikusan optimális algoritmus, *234*
aszimptotikusan önhasonló folyamat, *196*
átfedő blokk Jacobi-multifelbontás, *739*
átfordulási probléma, *668*
átfutási idő, 391
áthelyezhető programok, *456*
átlagos, 232
átlagos kódhossz, *421*
átlapolás, 366, 379
átló, *548*, *620*
átlósan szigorúan konkáv, *342*
átmenet
  testek közötti, *618*
átmenetfüggvény, *472*
áttekinthető alternatívák, 580
attribútum, *238*, *503*
  külső, 533*fe*
  prím, *520*, 534*fe*
automatikus párhuzamosítás, 235
axiómarendszer
  ellentmondásmentes, *504*
  teljes, 505
axiomatizálás, *531*
azonnali kód, *lásd* prefix kód

B
BACKTRACKINGSAT, *140*, 144*gy*
BACKWARD, 556
balsodrású, *669*
báziscímzés, *456*
bázisfüggvény, *610*
bázisregiszter, 456
bázisvektor, *606*

be/kimeneti adatsebesség, 307, 308
be/kiviteli séma, 287, 288*áb*
  bővített, 291, 292*áb*
be/kiviteli séma, bővített, 292
be/kivitel kiterjesztése, 291, 292
befoglaló keret, *648*
  AABB, *648*
  gömb, *648*
  hierarchikus, *648*
befoglaló téglalap, *688*, 689*áb*
Bellman–Ford-algoritmus, 266
belső pont, *607*
BELÜLRŐL, 557
bemeneti adatfolyam, 276
bemeneti csatorna, 270
bemenő jelek halmaza, *471*
Bernstein-polinom, *611*
BEST-FIT, 469*gy*
BESZÚRÁS-AZ-UTAZÓ-ÜGYNÖK-PROBLÉMÁRA, *585*
beszúró rendezés, 308
beviteli séma, 269*áb*
Bézier-görbe, *611*, 618*gy*
BF, *493*, 494, *lásd* gazdaságos algoritmus
BFD, *lásd* rendező gazdaságos algoritmus
bimátrix-játék, *328*
bináris fák
  megszámolása, 28
bináris térpartícionáló fa, *658*
binomiális képlet általánosítása, 26
BLOSUM, 546
Bluetooth, 168
bonyolultságelmélet,
Bővített-Euklidesz, *45*, 63*gy*, 92
Bővített-Euklidesz-normalizált, *47*, 63*gy*
Bresenham-algoritmus, *671*, *672*
BRESENHAM-SZAKASZRAJZOLÁS, *673*
BSP, *242*
BSP-fa, *658*, *681*
BSP-FA-FELÉPÍTÉS, *681*
B-spline, *612*, *614*
  rendszám, *612*
buborékrendezés, 307, 308*áb*
Burrows–Wheeler-transzformáció, *443*
BÜNTETÉS-AZ-UTAZÓ-ÜGYNÖK-PROBLÉMÁRA-KETTŐS-CSERÉVEL, *599*
büntetés-optimális, *590*
büntető függvény
  additív büntetés, 600
  additív büntető tag, *588*
  relatív büntető tényező, *587*, 590
büntető módszer, 586, *587*, *589*
büntető módszer, iteratív, *594*
BÜNTETŐ-MÓDSZER-RELATÍV-BÜNTETŐ-TÉNYEZŐKKEL, *587*
büntető paraméter, optimális, *592*, 599
BWT-DEKÓDOLÓ, *444*
BWT-KÓDOLÓ, *444*

C

Caesar-rejtjelező, 97
Canny-féle éldetektor, 708
Catmull–Clark-felosztás, 624*áb*
Catmull–Clark felosztott felület, *625*
ccNUMA, 224
cella, 269, *689*
  vezérléstmentes, *295*
cellaprogram, *303*, 304, *305*
cellaszerkezet, 269*áb*, 279*áb*, 285
CENTROID, 565, *688*
CGS-KLASSZIKUS-GRAM–SCHMIDT-ORTOGONALIZÁCIÓ, *754*
CGS-MÓDOSÍTOTT-GRAM–SCHMIDT-ORTOGONALIZÁCIÓ, *755*
Cholesky-felbontás, *734*, 737, 764*fe*
CHOLESKY-MÓDSZER, *735*
Chomsky-féle normálforma, 557
Church-tézis, 126, 133*gy*
CIR, *lásd* információátviteli intenzitással
Clustal-W, 542, 575
COHEN–SUTHERLAND-SZAKASZ-VÁGÁS, *637*
Cohen–Sutherland szakaszvágó algoritmus, *636*
COMNET, 173
Cox–deBoor-algoritmus, *614*
CRAY, 723
CRCW, 261
CRCW PRAM, 242, 266
CREW PRAM, 242, 244
CYK-algoritmus, 557

CS

csatorna, 270
csavarvonal, *609*
csendes állapot, 559
csepp módszer, *616*
CSG, *lásd* konstruktív tömörtest geometria
CSG-fa, 617*áb*, *800*
*-FUTTAT, *472*
csomag, *168*
csomag érkezési időköz, *199*
csomópontszerkesztővel, *180*
csomóvektor, *612*
csoport, *101*
  generátorrendszer, 103
  kommutatív, 101
  rend, 101
CSOPORTÁTLAG, 565
csúszó ablak, *441*

D

DCT, *448*
DDA, *lásd* digitális differenciális analizátor algoritmus
DDA-SZAKASZRAJZOLÁS, *671*
deriválás, *73*
Descartes-koordináta, *606*
Descartes-koordinátarendszer, *606*
determinisztikus időbonyolultság, *130*
determinisztikus Turing-gép, *127*
DET-RANGSOROL, 248, 249, 250
DIADIKUS-SZORZAT-MÓDOSÍTÁS-„IJ”-VÁLTOZAT, *759*
DIADIKUS-SZORZAT-MÓDOSÍTÁS-„JI”-VÁLTOZAT, *759*
DiAligh, 548
DiAlign, 576
Dickson-lemma, 67, 71*gy*
differenciálalgebra, *72*
differenciálás, *73*
  szabályai, *73*
differenciálbővítés, *73*
differenciál-kiterjesztés, 80
differenciálrésztest, *73*
differenciáltest, *72*
  bővítése, *73*
  differenciálbővítése, 78
Diffie–Hellman-protokoll, 94

digitális differenciális analizátor algoritmus, *670*
digitális szűrés, 695
dinamikus programozás, 584
Dirac-δ, 699
direkt hiba, *719*
diszkrét emlékezet nélküli forrás, *419*
diszkrét időszelet, *274*
diszkrét koszinusz transzformáció, *448*
diszkrét logaritmus, *708*
diszkrimináns, 63*gy*
DMS, *lásd* diszkrét emlékezet nélküli forrás
DNS, 538
döntéshozó, 581
döntési változó, *672*
duál feladat (az egyutas ütemezési problémáé), 391
duális fa, *622*

E, É
EDD, 373, 410, *lásd* legkorábbi határidő
egészértékű lineáris programozás, *599*
EGÉSZET-KIVÁLASZT, *261*, *262*
egész számok, 39
egy (régi) flop, *732*
egy (új) flop, *732*
egyenáram együttható, *450*
egyenes, *609*
  egyenlete, *609*
  helyvektora, *609*
  irányvektora, *609*
egyenes egyenlete, *640*
  homogén koordinátákban, *641*
egyenlet megoldása, 273
EGYENLŐSÍT, *514*
egyértelmű egyensúly, 343
egyértelműen dekódolható kód, *421*
egyidejűség, 274
egyirányú függvény, 108
egymást követő rögzítések, *602*
egységcsoport, 121
egység háromszögmátrix, *732*
egységnyi kerekítés mértéke, *722*
egyszeresen összefüggő sokszög, *619*
egyszerű, *619*
egyszerű algoritmus, 252
egyszerű algoritmus (NF), 492
EGYSZERŰ-ÁTLAG, 565
Egyszerű hálózati menedzsment protokoll, 190
egyszerű hatvány algoritmus (SP), 495
EGYSZERŰ-LÁNC, 565
egyszerű sokszög, *619*
egyutas ütemezési probléma, 366, 389
EH-ÉPÍT, *61*
e-kereskedelem, 583
éldetektálás, 707, *708*
eldobott csomagok aránya, 172
elem
  invertálható, 101
  inverz, 101
elemi
  függvénytest, *78*, 79, 81
  kiterjesztés, *78*
elemi függvény, 78
  integrálása, 79
elem rangja, 247
Elemzőeszköz, 181
ELHELYEZ, *464*, 470*gy*
ellipszis, *609*
élmegőrző szűrő, 706
előbetöltés, *470*
ELÖLRŐL, 557
ELŐRE-MOZGATÁS, *446*
előre-mozgató kód, *445*
ELŐRETEKINTŐ, *543*
ELŐRETEKINTŐ-BINÁRISKERESŐ, *545*
előzéses egyutas ütemezési probléma, 391
előzésnélküli egyutas ütemezési probléma, 391
élpont, *625*
eltolás, *441*, *605*
ELTOLÁSOS-QR-MÓDSZER, *757*
eltolásos QR-módszer, 758*gy*
ember-gép kölcsönhatás, *580*
emboss-szűrő, 708, 710*áb*, 804*áb*
emuláció, 166
entrópia, *420*
EP, *lásd* gazdaságos hatványtípusú algoritmus
ε-alternatívák, 589, 592
  monotonitási tulajdonságai, *593*
  unimodalitási tulajdonság, *592*
ERCW PRAM, 242
EREW PRAM, 242, 265
EREW-PREFIX, 244, *246*
érintősík egyenlete, *609*
erőforrás, 364, 366
erős generátor, 103
erős ingadozás, 171
erős ingadozás érkezési időköz, *199*
érték, 332
értékadásmentes jelölés, *273*
értéktartomány, 273
  sűrű konvex, *281*
értéktartomány transzformációja, *303*
értelmezési tartomány, *503*
Euler-féle φ függvény, 102
exponenciális elem, *78*

F
fa, 122
  levél, 122
FÁBAN-VALÓ-KERESÉS-ITERATÍV-MÉLYÍTÉSSEL, *601*
Fan-egyenlőtlenség, 323
fekete doboz modellezés, *198*
feladat paraméterei, *271*
félcsoport, 101
felmelegedési periódus, *192*
felső és alsó becslések, 496
felső Hessenberg-alak, *757*
felső-hierarchia, 144, 149*áb*, *150*
Felsőoktatási Pályázatok Irodája, 4
felület, *607*
felülvágó szűrő, 705
fényesség, *448*
festő algoritmus, *679*, 684
FF, 384, 492, *lásd* mohó ládapakolási algoritmusmohó algoritmus
FFD, *lásd* rendező mohó algoritmus
Fibonacci-számok, 22
Fickett algoritmusa, 550
FIFO, 480
FIFO sorrend, 410
FIFO-VÉGREHAJT, *473*
FIRST-FIT, 469*gy*
Fitch-algoritmus, 554*gy*
fixpont módszerek, 321
fixpontos számábrázolás, *671*
fizikai memória, *470*

flipflop állapotkapcsoló, 298
fogoly dilemma, *315*
fokszám, 104
folyamat, 224, *228*
folyamatszerkesztő, *180*
folyamfüggőség, 284
forgalomirányító protokollok, 170
forgalomszervezés, 171
forrás, *418*
forrásmodellezés, *418*
forrás modellezése, *429*
FORWARD, 556
Fourier-Motzkin elimináció, 304
Fourier-transzformáció, 697
főegyüttható, *44*
fundamentális megoldás, *15*
funkcionális-reprezentáció, 683
FUTAMHOSSZ-KÓD, *451*
futamhossz kódolás, *451*
futási idő, *192*
futószalag-elvű feldolgozás, 277
futószalag-regiszter, 277
függőség
  egyenlőséggeneráló, 525
  elágazó, 532
  funkcionális, *503*
    ekvivalens családok, *508*
  numerikus, 532
  összekapcsolási, 530
  sorgeneráló, 525
  többértékű, 525
függőségi bázis, 527
FÜGGŐSÉGI-BÁZIS, 528, 534*fe*
függőségi vektor, 284
fül, *620*
fülvágó algoritmus, *621*

G
Gantt-diagram, 412*fe*
gát, *573*
Gauss-elimináció, 89
GAUSS-MÓDSZER, *729*
GAXPY, *760*
gazdagép, 269, 275
gazdaságos algoritmus (BF), 492
gazdaságos hatványtípusú algoritmus (EP), 495
gazdaságos rendező algoritmus (BFD), *494*
generatív tervezés, *602*
generátorfüggvény, 22
  bináris fák megszámolása, 28
  műveletek, 23
generikus operátor, 272, 286, 292
genetikus algoritmus, 585, *602*
gép, 364
gépi epszilon, *722*
GÉPI-EPSZILON, *722*
Givens-módszer, 754
globális helymeghatározó rendszer, *168*
gombócevési sebesség, *489*
Gotoh-algoritmus, 542
gömb, *607*, *608*
görbe, *608*
GPS, *lásd* globális helymeghatározó rendszer
gráf
  automorfizmus, 104
  diszjunkt egyesítés, 104
  izomorfizmus, 104
gráfautomorfizmus probléma, 104
grafikus megjelenítés, *603*
gráfizomorfizmus probléma, 104, 144, 150, 153
gravitációs segítség, 583
grid, *689*
grid index, *689*
Gröbner-bázis, 63, 67, 68, 69, 70, 88, 92*fe*
  minimális, *70*
  redukált, *69*, *70*
Gustafson-Barsis-törvény, *229*

GY
gyengén stabil, *742*
GYORSAN-ÖSSZEFÉSÜL, *254*
gyors Fourier-transzformáció, 703
gyorsítás, 228, 233, 266
  lineáris, 233
  szublineáris, 233
  szuperlineáris, 233
GYÖKÖS-KIVÁLASZT, *260*
gyűrű, *101*
  egység, 101
  egységelem, 101
  kommutatív, 101
  nullelem, 101
  nullosztó, 101

H
hálózat
  helyi, 168
  nagy kiterjedésű, 168
  pont-pont, 167
  személyes, 168
  üzenetszórásos, 167
hálózatok szimulációja, 164
Hamming-távolság, 141
Hanoi-tornyai, 22*gy*
hányados, *44*, *66*
hardver-algoritmus, 268
hármasítás, 134
harmonikus számok, 24
3-agy, 583
3DDDA algoritmus, *650*
3D szakaszrajzoló algoritmus, *650*, 683
háromdimenziós párosítás, *134*, *135*, 136
3-SAT, *134*
háromszög, *608*
  balállású, *676*
  jobbállású, *676*
háromszögháló, *618*
háromszögmátrix
  alsó, 308
hasítás, *151*
hasonló párhuzamos gépek, 366
hasonlósági transzformáció, *750*
határfelület, *607*
határidő, 366
határregiszter, 457
HATÉKONYAN-ÖSSZEFÉSÜL, 253
hatékony megoldás, *603*
HATÉKONY-PREFIX, 245
hatékonyság, 229, 233, 266, 277
hatékonysági mérték, 233
  abszolút, 231
  relatív, 231
hátizsák probléma, 589
háttérmemória, *470*
HATVÁNYMÓDSZER, 752, 757*gy*, 758*gy*

hatványsor, 42
házasítás probléma, 135
Heaviside-féle lépcsősfüggvény, 716
helyes, 505
helyiértékes számábrázolás, 39
helyi hálózat, 168
helyvektor, 606
helyzetkép, 289
henger, 608
Hermite-redukció, 74, 75, 81, 85, 88*gy*, 92*fe*
heurisztika, 585, 599, 601, 602
  cserélő, 585
heurisztikák, 584
hiba, 717
hibakorlát, 717, 726*gy*
Hilbert-bázis, 67
Hilbert-mátrix, 744
hiperexponenciális elem, 78
Hirschberg algoritmusa, 549
hisztogram kiegyenlítés, 696
hitelesség, 110
hivatkozási sorozat, 471
HMMER, 577
holtversenyes esetek, 586
homogén koordináta, 639, 640
HOMOGÉN-LINEÁRIS, 20
homogén lineáris transzformáció, 638, 642
Horowitz-módszer, 75
hosszú távon függő folyamat, 196
hosszú távú függőség, 171
Householder-módszer, 754
hozzárendelési probléma, 589
Huffman-algoritmus, 426
Huffman-kódolás, 418
Hurst-paraméter, 196, 199–203

I, Í
ideál
  bázisa, 63
  varietása, 63
ideális egyenes, 638
ideális pont, 638
ideális sík, 639
ideállánc, 67, 69
I-divergencia, 425
időszelet
  diszkrét, 274
idővektor, 280
IEEE, 723
IEEE Information Theory Society, 454
igény szerint, 470
illesztett pár, 540
implicit egyenlet, 607
indexhalmazok transzformációja, 303
indexkifejezés, 303
információátviteli intenzitással, 203
inhomogenitás, 309
inkrementális elv, 662, 670, 674
integrál
  logaritmikus része, 74
  racionális része, 74
integrálás
  parciális, 73
integrálgeometria, 658, 684
integrállogaritmus, 78
interaktív evolúció, 602
interaktív több feltételes döntéshozatal, 602
Internet, 167
internetwork, 167
invariánsok módszere, 673
inverz, 101
inverz elem, 101
inverz hatványmódszer, 753, 758*gy*
inverz hiba, 719, 741
inverz hibaanalízis, 719
inverz mátrix, 99
inverz stabil, 719, 720
irányított kapcsolat, 275
irreguláris célfüggvény, 367
ITERÁCIÓ-MULTIFELBONTÁSSAL, 738
iterációs vektor, 273
ITERATÍV-BÜNTETŐ-MÓDSZER, 594
iteratív javítás, 764*fe*
ITERATÍV-JAVÍTÁS, 747
iteratív mélyítés, 586, 601
izoparametrikus görbe, 609

J
játék
  folytonos, 320
  kevert bővítése, 327
  véges, 315
  zérussösszegű, 317
játékos, 314
jobbkéz szabály, 606
jobbsodrású, 669
jól kondicionált, 719
JPEG, 419, 447

K
k-adik pivotelem, 730
kamera transzformáció, 665
kampus-klaszter, 227
kapacitás, 364
KAPCSOLÁS-TESZT, 514, 515*áb*, 524
kapcsolat, 270, 275
kapcsolat kapacitás, 170
kapcsolatszerkezet, 284
  hexagonális, 285
karakterisztikus egyenlet, 15, 749
karakterisztikus polinom, 749
kd-fa, 658
KD-FA-FELÉPÍTÉS, 659
képernyő-koordinátarendszer, 663
képpont, 638
képszintézis, 605
kerekítés, 722
kerekítési hiba, 722
keresés, 259
keresési feladat, 259
keret méret, 171
kernel, 703
késés, 170
késleltetés, 230, 275
késleltetett kiértékelés, 42
kétdimenziós párosítás, 134, 135
két fül tétel, 621, 683
KETTÉVÁG-PARTÍCIÓ, 464
kettős csere, 599
kezdeti feltétel, 14
kezdeti vezérlő állapot, 471
ki/beviteli séma, 276
kiegyensúlyozás, 744
kielégíthetőség probléma, 133, 134, 139

kiemelt klaszter, *227*
kifejezések egyszerűsítése, 71
kifizetés, *314*
kifizetőfüggvény, *314*
kifizetőmátrix, *317*
kifizetővektor, *314*
kihasználtság, 293, 295
kimenetfüggvény, *471*
kimeneti csatorna, *270*
kimeneti normál forma, *304*
kimenő jelek halmaza, *471*
kiszámítható függvény, *129*
kiterjedés, *430*
kiválasztás, *259*
kiválasztó rendezés, 308
kivonatok, 583, 584
KÍVÜLRŐL, 557
klasszikus Cournot-modell, *349*
KLASSZIKUS-EUKLIDESZ, *44*, 92
KLASSZIKUS-EUKLIDESZ algoritmus működése, 45*áb*
klasszikus hibaszámítás, 717
klaszter, 224, *227*
klaszterező algoritmus, 584
k-legjobb algoritmus, 581
Knudsen-Hein-nyelvtan, 559*gy*
Kologorov-bonylultság, 440
KOMPENZÁLT-ÖSSZEGZÉS, 725
komplementer feladat, 356
kompozíció, 102, 539
komputeralgebra, 38
komputeralgebra-rendszerek
  általános célú, 90
  speciális célú, 90
kondíciószám, *718*, 719, 726*gy*, 740, 748*gy*, 763*fe*, 765*fe*
konjunktív normálforma, *134*
konkáv csúcs, *620*
konkáv résbüntető függvény, *543*
konstansok részteste, *73*
konstansok variálásának módszere, 21
konstruktív tömörtest geometria, *616*
konvex burok, *611*
konvex csúcs, *620*
konvex-kombináció, *608*, *609*
KONVOLÚCIÓ, 700, *704*, 710*gy*, 715
konvolúció kiterjesztése, 715*fe*
koordináta, *606*
KORLÁTOS-BEST-FIT, 469*gy*
KORLÁTOS-WORST-FIT, 470*gy*
költségvezérelt módszer, *658*
körmentes gráf, 266
környezetfa, *436*
KÖRNYEZET-FA-MÓDOSÍTÁS, *438*
környezetfa súlyozó algoritmus, *437*
közönséges pont, *638*
köztes számítási tárrobbanás, *42*
Kraft-egyenlőtlenség, 418, *422*
Krichevsky–Trofimov-becslés, *435*
kriptoanalízis, 95
kriptográfia, 94–124
kriptológia, 95
kriptorendszer, *96*
  aszimmetrikus, 96, 99
  monoalfabetikus, 97
  nyilvános kulcsú, 110
  polialfabetikus, 97
  szimmetrikus, 96, 99
Kruskal-algoritmus, 266*fe*
Kuhn–Tucker-feltételek, 324
kulcs, 504, *510*, 524
  elsődleges, *531*
kulcscsere-probléma, 94
KULCS-KERESŐ, *511*
kulcs tér, 96
kúp, *608*
külső pont, *607*
külvilág, 270, 275
kvantálás, *449*, *450*
kvantifikálás, 273

L

ládapakolási probléma, 384
LAN, *lásd* hálózat
lapcserélési algoritmus
  statikus, *471*
lapcserélési algoritmusok, 470–480
laphiba, *472*
lapkeret, *470*
Laplace-szűrés, *708*, 709*áb*, 804*áb*
lapozási sebesség, *481*
lappont, *625*
lassú indulás algoritmus, 172
láthatósági feladat, *675*
Laurent-sor, 85
Lazard–Rioboo–Trager-formula, 76
lebegőpontos aritmetika, 721, 722, 724, 764*fe*
lebegőpontos aritmetikai szabvány, 725
lebegőpontos számhalmaz, 726*gy*
leghosszabb megmunkálási idő, 377
legjobbválasz, 321
legkevesebb megmaradt megmunkálás, *410*
legkisebb megmunkálási idő, 385
legkorábbi határidők sorrendje, 373
legközelebbi szomszéd keresés, 715*fe*
LEGNAGYOBB-BEFÉRŐ, *458*, 469*gy*
LEGNAGYOBB-VAGY-RÉGÓTA-VÁRAKOZÓ-BEFÉRŐ, *460*, 469*gy*
legrövidebb kör, 582
legrövidebb megmaradt megmunkálási idők, 376
legrövidebb megmunkálási idő, *368*
legrövidebb út probléma, 582, 589, 595
lehetséges megoldások, 581, 582, 584
Leibniz-szabály, 73
lényegtelen transzformáció, 448
lépésszám
  átlagos, 232
  legjobb esetben, 232
  legrosszabb esetben, 232
  várható, 232
LERC, *154*
leszámlálás, 316
LEZÁRÁS, 506, 507, 524, 533*gy*
  attribútumhalmazé, 505, 506, 511*gy*
  funkcionális függőségé, *504*, 506, 511*gy*
LF, *491*
LFU-VÉGREHAJT, *476*
LIFO sorrend, 410
LINEÁRIS-INHOMOGÉN, *34*
LINEÁRIS-LEZÁRÁS, *508*, 518, 519, 522
lineáris programozás, 595
LINPACK, 746
Liouville-elv, 79, *80*, 82, 83, 86
listaröviditás, *496*
listás ütemezés, 382
logaritmikus
  bővítés, *73*, 83
  derivált, *73*, 78

elem, 78, 81
függvények, 80
LOGARITMIKUS-RÉSZ-INTEGRÁL, 77
logikai következmény, 504, 525
log-odds, 546
LOG-ÖSSZEFÉSÜL, 251
LogP, 242, 243
lokális keresés, 601
kettős cserével, 585, 599, 600
LOKÁLIS-KERESÉS-AZ-UTAZÓ-ÜGYNÖK-PROBLÉMÁRA, 585
lokális optimum megtalálása, 585
lokális vezérelhetőség, 614
LPT, *lásd* leghosszabb megmunkálási idő
LPT sorrend, 378, 385, 410, 412
LRU-VÉGREHAJT, 474
LU-felbontás, 732
LU-MÓDSZER, 733
LU-módszer-pointeres-technikával, 734
LWR, *lásd* legkevesebb megmaradt megmunkálás
lyuk, 470
lyukszűrés, 706

M
M-A, 156
MA az Arthur–Merlin hierarchiában, 117
MAN, *lásd* hálózat
maradék, 44, 66
Markov-modell, 575*fe*
masírozó kockák algoritmus, 626
másodrendű felületek, 646
matematikai szakértői rendszer, 91
MATLAB, 724, 762
mátrix
adjungált, 99
determináns, 99
inverz, 99
sűrű, 310
mátrix és vektor szorzása, 307
mátrix invertálás, 734
mátrixjáték, 332
Matrix Laboratory, *lásd* MATLAB
mátrixszorzás, 269, 270, 272, 278
MÁTRIXSZORZÁS-DOT-VÁLTOZATA, 760
MÁTRIXSZORZÁS-GAXPY, 761
MÁTRIXSZORZÁS-GAXPY-HÍVÁSSAL, 761
MÁTRIXSZORZÁS-KÜLSŐ-SZORZAT, 761
maximális adatátviteli egység, 171
MBR, *lásd* befoglaló négyszög
Mealy-automata, 471
MEDIÁN, 566
medián szűrő, 706, 707*ák*
megengedett akciók halmaza, 314
megengedett megoldás, 581, 588
megengedett részhalmaz, 588
megmunkálási idő, 365
MEGŐRIZ, 518
megszakítás, 366
mellékosztály
jobb oldali, 103
mélység-puffer, 675
memória, 275
memória elaprózódása, 465
Menedzsment információs adatbázisból, 190
mérnöki visszafejtés, 683
merőleges
vektorok, 606
mesterszál, 239
metszéspont számítás
háromszög, 647
sík, 647
METSZÉS-VÁGÓSÍKKAL, 634
MF algoritmus, 387, 388
MF ládapakolási algoritmus, 386
M-G, 156
MIB, *lásd* Menedzsment információs adatbázisból
Mieses-eljárás, 753
mikro mutációk, 587
MILLER-RABIN, 112
MIMD, 224
MINIMÁLIS-FEDÉS, 509, 523, 533*fe*
minimális törzsfejlődés, 538
minimális változtatás, 581
mintavételezés, 710
mintavételi tétel, 712
MISD, 224
modell
determinisztikus, 165
dinamikus, 165
diszkrét, 165
fluid-flow, 197
folytonos, 165
kötegelt markovi érkezés, 197
markovi érkezési folyamat, 197
Markov-modulált Poisson-folyamat, 197
statikus, 165
sztochasztikus, 165
modellfejlesztési életciklus, 189
MODULÁRIS-HATVÁNYOZÓ, 108
MODULÁRIS-LNKO-KISPRÍMEK, 60
MODULÁRIS-LNKO-NAGYPRÍM, 60
moduláris számábrázolás, 39
MOHÓ, 386
mohó algoritmus (FF), 492
mohó ládapakolási eljárás, 384
molekuláris óra, 566
monom, 64
monomiális
elem, 80
ideál, 67
rendezés, 64
Monte-Carlo-algoritmus, 112
Monte Carlo integráció, 601
MPI, 222
MPI-2, 238
MPMD, 228, *lásd* többszörös program többszörös adat
MTU, *lásd* maximális adatátviteli egység
multifelbontás, 737
multifelbontás algoritmus, 738
munka, 233, 266
munkadarab, 364
munkahatékony, 234
munkahatékony algoritmus, 233
mutatóugrás, 249
művelet, 364
MWKR sorrend, 410
MWR, *lásd* leghosszabb megmaradt megmunkálás

N
NAIV-BCNF, 522
Nash-egyensúly, 314
nccNUMA, 224
nccNUMA architektúra, 226
négycsúcs metrika, 566
négyes fa, 657*gy*
négy-fa, 690, 692*áb*, 694*gy*, 715*fe*

NÉGY-FA-KERES, *692*
négyszögimpulzus, 698
négyzetes egyenletrendszer, *727*
négyzetes szita, 115
nemátfedő blokk Jacobi-multifelbontás, *738*
nemdeterminisztikus időbonyolultság, *130*
nemdeterminisztikus Turing-gép, *127*
nem-gát, *573*
nemlineáris komplementer feladat, *357*
Neumann-módszer, *338*
NEXT-FIT, *466*, 469*gy*
NF, *492*, *lásd* egyszerű algoritmus
NFD, *lásd* rendező egyszerű algoritmus
Noether-gyűrű, *67*
NORMA, 224, *226*
normálalak, *44*
normálforma, *519*
  BCNF, 519, *520*, 533*fe*
  Boyce–Codd, 519, *520*
  3NF, 519, *520*
  3NF, 519, *520*, 534*fe*
  4NF, 519, *528*
  negyedik, 519, *528*
  5NF, *531*
  ötödik, *531*
normalizált, *721*
N-PRE-ISO, *146*
NP-teljes, 584
NP-teljes probléma, 582
NP-teljesség, *133*
NRU-ELŐKÉSZÍT, *477*
NRU-KIDOB, *478*
NRU-VÉGREHAJT, *477*
nulla-egy elv, *252*
nullosztó, *101*
numerikusan instabil, *719*, 726*gy*
numerikusan stabil, *719*, 727*gy*
numerikus instabilitások, 603
NURBS, *615*

NY

nyeregpont, *317*
nyílt szöveg, 96
nyilvános kulcs, 96
nyilvános kulcsú kriptográfia, 95
nyilvános kulcsú kriptorendszer, 95
nyolc-fa, *694*

O, Ó

oktális fa, *656*
Oktatási Minisztérium, 4
oligopol játék, *349*
on-line ütemezési feladat, 367
OpenMP, 222, 241
OPNET, 221, *lásd* Optimalizált hálózattervező
optikai szálas elosztott adatinterfész, 168
Optimalizált hálózattervező eszközök, *178*
optimumszámítási feladat, 326
OPT-KIDOB, *475*
OPT-VÉGREHAJT, *474*
orákulum, *145*
orákulumos Turing-gép, *145*
origó, *606*
ortogonális, *754*
ortonormált bázis, *754*
OSZD-MEG-ÉS-FEDD-LE, *591*
oszd-meg-és-uralkodj algoritmus, 590
osztó, 309

Ö, Ő

ökölszabály, *741*
önhasonló folyamat, *196*
önhasonlóság, 195
ön-visszavezethetőség, 147
összeadó, 270
összeg típusú optimalizálási probléma, *588*
összegzőfüggvény, 322
összekapcsolás, *293*
összetett művelet, *274*

P

pálya, 103
PAM, *543*, 546
PAML, 577
PAN, *lásd* hálózat
paraméteres egyenlet, *608*
PÁRATLAN-PÁROS-ÖSSZEFÉSÜL, *251*
PÁRATLAN-PÁROS-RENDEZ, *264*
parciális törtekre bontás, 81
párhuzamos
  vektor, *606*
párhuzamos: egyenes, *609*
párhuzamos: sík, *608*
párhuzamos algoritmus, 738
párhuzamos vektorok, *606*
páronkénti összeg, *551*
páronkénti összehasonlítás, 496
párosítás, *134*
párosító algoritmus, 493
páros rejtett Markov-modell, 559
partíció, *456*
  dinamikus, *463*
  rögzített, 457
partikuláris megoldás, *14*
pászta, *673*
Peano rendezés, 694
példány, *503*
peremcella, 270
periódus, *97*, *301*
permanens virtuális áramkör, *191*
permutáció, 102
  identikus, 103
permutációcsoport, *102*
permutációcsoportok, *102*
permutációmátrix, *733*
PF, *493*, *lásd* párosító algoritmus
PFD, *lásd* rendező párosító algoritmus
PFF, *479*
PFold, 577
pillangó felosztás, *626*, 683
pivotelemek növekedési tényezője, *731*
pixel, *605*
Poisson-pontfolyamat, *653*
poliéder, *619*
poligon, *619*
POLIGONKITÖLTÉS, *675*
poligonkitöltő algoritmus, *673*
polinom
  ábrázolása, 41
  összetevője, 50
  primitív, 50
  többváltozós, 41, 65

polinom diszkriminánsa, 63*gy*
polinomegyenletek
  ekvivalenciája, 70
  megoldáshalmaza végessége, 71
  megoldhatósága, 71
  véges megoldásai száma, 71
POLINOMIÁLIS-BCNF, 523
polinomiális idő-hierarchia, 145, *149*
POLINOMOK-MARADÉKOS-OSZTÁSA
  egyhatározatlanú, 45, 63
  többhatározatlanú, 66
politóp, 281
pont, *606*
pontos, de lassú eljárások, 584
PONTOS-ARITMETIKAI-KÓDOLÓ, *433*
pontos bonyolultság, *234*
pont-tartomány négy-fa, *690*
PRAM, 242
prefix kód, *421*
prefix kód redundanciája, 429*gy*
prefixszámítás, 244
prekondicionálás, *745*
PREPARATA, *264*
Prim-algoritmus, 266*fe*
prím attributum feladat, 534*fe*
PRIMITÍV-EUKLIDESZ, 57, 92*gy*
PRIMITÍV-EUKLIDESZ algoritmus működése, 51*áb*
primitív gyök, 107
probléma
  Diffie–Hellman-probléma, 109
probléma érzékenysége, *739*
program, 399
Projector, 577
projekció, 280
projekció iránya, *280*, 291
projekciós mátrix, *280*
projekciós vektor, *280*
projektív egyenes, *641*
projektív geometria, *638*
projektív sík, *639*, *641*
projektív szakasz, *641*
projektív tér, *638*, *639*
Projektszerkesztő, *180*
protokoll
  zéró-ismeretű, 118
protokoll többletterhelés, 171
pszeudo-hányados, 50
pszeudo-maradék, 50
puha határidő, 366
PVC, *lásd* permanens virtuális áramkör

Q
QFD, *494*, *lásd* rendező gyors algoritmus
QR-módszer, 758*gy*
QR-MÓDSZER, *756*
QSM, 242

R
R/S, *lásd* Újraskálázott Módosított Tartomány
racionális függvények integrálása, 72
racionális számok, 41
rács, 224
rács paraméterei, *271*
raktározás, 364
RANDOM-SAT, *142*
RANG-SZÁMÍT, *462*
rangszűrő, *706*
raszteres adatmodell, 686
raszterizáció, *670*
Rayleigh-hányados, 752
redundancia, *439*
regiszter, 270, 285
RÉGÓTA-VÁRAKOZÓ-VAGY-KISEBBE-NEM-FÉR, *462*, 469*gy*
reguláris célfüggvény, 367
rejtett Markov-folyamat, 555
rejtett Markov-modell, 577
rejtett szöveg, 96
rejtjelező
  blokkrejtjelező, 99
    affin, 99
    lineáris, 99
  eltolásos, 96
  helyettesítéses, 99
  Hill-rejtjelező, 99
  monoalfabetikus, 99
  permutációs, 99
  polialfabetikus, 99
  transzpozíciós, 99
REKURZÍV, 36
rekurzív egyenlet, 14, 273
  állandó együtthatós lineáris, 15
  homogén lineáris, 15, 20
  inhomogén lineáris, 20, 32, 33
  lineáris, 15, 20, 26, 32
  megoldása generátorfüggvénnyel, 26
rekurzív egyenletek, 13
rekurzív felbontás, 690, 691*áb*
REKURZÍV-ÖSSZEGZÉS, *724*
relációs séma, 503
relatív hiba, 726*gy*, 765*fe*
relatív hibakorlát, *717*, 722, 726*gy*, 748*gy*
relatív inverz hiba, 747
relativizálható bonyolultsági osztály, *145*
relatív lépésszám, *lásd* gyorsítás
rendelkezésre állási idő, 369
rendezés, 262–265, 307
  megengedett, 64
  monomiális, *64*, 71*gy*
rendezett fa, 562
rendező egyszerű algoritmus (NFD), *494*
rendező gyors algoritmus (QFD), 494
reprezentánsrendszer
  jobb oldali, 103
rés, *541*
résbüntetés, *541*
részcsoport, 101
részleges főelemkiválasztás, *730*
részstring, 546
reziduális hiba, 739
reziduum-tétel, 34
rezultáns, 52, 82, 85
  Sylvester-féle alak, 54
rezultánsmódszer, 52
R-fa, 716
RGB színmodell, 695
Risch-algoritmus, 77, 79, 87, 88
  exponenciális eset, 84
  logaritmikus eset, 81
Risch-differenciálegyenlet, 87
RMON, *lásd* Távoli megfigyelő
RNS-térszerkezet, *584*
Rodrigues-képlet, *643*, 644*gy*
rossz megosztás, 225
Rothstein–Trager-módszer, 76, 82, 85
rögzített partíciók, *457*
rövid távon függő folyamat, *196*

S
sajátérték, 749, 765*fe*
sajátérték kondíciószáma, 751
sajátvektor, 749
sakk, 583, 601
SAT, 134
sávmátrix, 735, 736
SÁVMÁTRIX-*LU*-FELBONTÁSA, 736
SÁVMÁTRIXOK-CHOLESKY-FELBONTÁSA, 737
SÁVOS-ALSÓ-HÁROMSZÖGMÁTRIXÚ-EGYENLETRENDSZER, 736
SÁVOS-FELSŐ-HÁROMSZÖGMÁTRIXÚ-EGYENLETRENDSZER, 736
sávszélesség, 170, 230, 310
sávszűrő, 706
SAXPY, 759
semleges elem, 101
Shannon–Fano-algoritmus, 426
Shannon–Fano–Elias-kód, 426
sík, 607
SIMD, 224
SISD, 224
skaláris szorzat, 605
skalárszorzat, 272
skálázás, 744
Skeel-féle norma, 742
SL, *lásd* listarövidítés
SMP, 224
Sniffer, 188
SNMP, *lásd* egyszerű hálózati menedzsment protokoll
SNR, *lásd* jel/zaj hányados
sokszög, 619
sorba fejtés, 272
sorozat, 364, 366
sorozat csoportosítási probléma, 589
sörétes-puska nukleinsavleolvasás, 574
SP, *lásd* egyszerű hatvány algoritmus
spektrum, 697
SPMD, 228, *lásd* egyszeres program többszörös adat
S-polinom, 68
Spouge algoritmusa, 550
SPT, 375, 376, 379, 380, 410, 412*fe*, 414, *lásd* legrövidebb megmunkálási idő|textitlegrövidebb megmunkálási idő
SRPT, *lásd* legrövidebb megmaradt megmunkálási idő
stabilizátor, 103
stabilizátor lánc, 103
stacionárius, 286
stacionárius számítás, 277
stacionárius változók, 277, 293, 296
stratégia, 314
stratégiahalmaz, 314
sugár, 645
SUGÁR-ELSŐ-METSZÉSPONT, 646
SUGÁR-ELSŐ-METSZÉSPONT-KD-FÁVAL, 660
SUGÁR-ELSŐ-METSZÉSPONT-OKTÁLIS-FÁVAL, 656
SUGÁR-ELSŐ-METSZÉSPONT-SZABÁLYOS-RÁCCSAL, 651
sugárkövetés, 645
sugárparaméter, 645
súlyfüggvény, 539
súlyozott eloszlás, 437
súlyozott legrövidebb megmunkálási idők, 375
Sutherland–Hodgeman-poligonvágás, 634, 683
SWPT, *lásd* súlyozott legrövidebb megmunkálási idők sorrendje
SWPT sorrend, 380

SZ
SZABÁLYOS-RÁCS-FELÉPÍTÉS, 650
SZABÁLYOS-RÁCS-KÖVETKEZŐ-CELLA, 651
SZABÁLYOS-RÁCS-TARTALMAZÓ-CELLA, 650, 651
szakasz, 609
szakasz egyenlete, 609
szál, 228
szalagmátrix, 310
szállítás, 364
szálprogramozás, 235, 241
számítási modell, 232
számítási tárrobbanás, 70
számítógépes biológia, 584
szekvencia, 538
személyazonosítás, 117
szempozíció, 663
szerencsés prím, 60
szétdarabolás, 276
szétvágás
  függőségőrző, 517
  függőségőrző 3NF-re, 523
  veszteségmentesen BCNF-be, 527
  veszteségmentes összekapcsolású, 512
szimbolikus
  integrálás, 71
  számítások, 38
szimbolikus meghatározás, 281
szimmetrikus mátrixjáték, 334
szimulált hőkezelés, 585
szimultán stratégiavektor, 314
színesség, 448
szinkron munkamód, 274
szisztolika, 268
szisztolikus, 268
szisztolikus algoritmus, 269
  egyenletes, 278
szisztolikus rács, 268, 269, 310
  elfajult, 307
  hexagonális, 279, 282
  lineáris, 308*íb*
  téglalap alakú, 269*áb*
  többdimenziós, 277
szisztolikus rács pereme, 306
szisztolikus rendszer, 268–311, 269
szorzás-összeadás, 274
szorzó, 270
sztochasztikus automata, 141
szublineáris gyorsítás, 233
szubnormális szám, 725, 727*gy*
szuperkulcs, 504, 510
szuperlépés, 242
szuperlineáris gyorsítás, 233

T
támadás
  aktív, 109
  faktorizálási, 114
  középen állás támadás, 109
  nyílt szövegű, 100
  passzív, 109
  rejtett szövegű, 97, 100
  rejtjelező kulcs, 100
  választott szövegű, 100, 114
tanácsadó rendszerek, 583
tárgypont, 638
tartomány kalkulus, 515

tartomány négy-fa, *693*
taszk, 224, *228*
taszkpárhuzamosság, *228*
Távoli megfigyelő, 190
T csomópont, *623*
technológiai útvonal, 365
téglatest, *607*
teljes főelemkiválasztás, *730*
teljesítmény, *301*
teljesítménymérték, 169
TELJES-LÁNC, 565
teljes végrehajtási idő, 275, 283
tényleges határidő, *366*
tér, *605*
térbeli frekvencia, 695
térbeli indexelés, 687
térbeli koordináták, *271*, 281
térbeli középvonal módszer, *658*
tér-idő-leképezés, 278, 279, *280*, 303
térinformatika,
termelési feladat, 365
természetes összekapcsolás, *512*, 522
test, *101*, 102, *607*
test középvonal módszer, *658*
tesszeláció
  adaptív, *622*
tesszelláció, *618*
tetszőleges futási idejű algoritmusok, *586*, *601*
tevékenység, 364
TIAT, *lásd* tranzakció érkezési időköz
titkos kulcs, 96
tórusz, *607*
többutas ütemezési probléma, 366, 397
több választási lehetőséges optimalizálás, *580*, 581
több választási lehetőséges teszt, 581
több választási lehetőséget adó algoritmus, 581
több választási lehetőséget kínáló rendszer, *580*, 586
több választást kínáló rendszer, 581
  példa, 582
többváltozós polinom
  főmonomja, *65*
  főtagja, *65*
  multifoka, *65*
  tagjai, *65*
tökéletes titkosság, 100
tömb feje, 248
tömbrangsorolás, *247*
tömörítés
  veszteséges, 419
  veszteségmentes, 419
töröttvonal, *618*
transzcendens
  elem, *78*, 81
  elemi függvénytest, *78*
  elemi kiterjesztés, *78*
transzformáció, *638*
transzformáció-kiterjesztés-minta modell, 197
tranzakció érkezési időköz, *199*
tranziens függvény, 698
tri-lineáris közelítés, *626*
túlcsordulás, *763fe*
Turing-gép, *126*, *127áb*, *145*
Turing-gép szemantikája, 128
Turing-gép szintaxisa, *127*
Turing-visszavezethetőség, 145, *148*

U, Ú

UDC, *lásd* egyértelműen dekódolható kód
Ukkonen algoritmusa, 550
ULE, *lásd* felső és alsó becslések
ultrametrika, *564*
unimoduláris mátrix, 281, *282*
univerzális hasítás, *151*
utazóügynök probléma, 589
utazóügynök probléma (TSP), *582*, 584, 599, 603
útvonal keresése, 581
útvonaltervező, 586, 587, 592

Ü, Ű

ütközésfelismerés, *628*, *645*
üzenet, 96

V

vágás, *628*, 633, 663, *668*
  szakaszok, *633*
válaszidő, *170*
választás, *437*
választott szövegű támadás, 123
valós idejű problémák, 581
váltási hely, *234*
változó intenzitásirányítás, 203
VBR, *lásd* változó intenzitásirányítás
véges játék, *375*
végső döntés, 580, 583
vektor, *605*
  abszolút értéke, *605*
  összeadás, *605*
  skaláris szorzás, *605*
  számmal szorzás, *605*
  vektoriális szorzás, *606*
vektorizáció, *618*
vektoros adatmodell, 686
véletlen séta, 141
veremtulajdonság, *490*
veszteséges tömörítés, *419*
veszteségmentes tömörítés, *419*
vezérfa, *547*
vezérlés
  globális, *296*
  helyi, *296*
vezérlés, osztott, *300*
vezérlő állapotok halmaza, *471*
vezérlőpont-sorozat, *610*
viewport, 687
Vigenère-négyzet, 97
virtuális lap indexe, *471*
virtuális memória, *470*
virtuális világ, *605*
visszaállítás, *296áb*
visszafelé indukció, *318*
VISSZAHELYETTESÍTÉS, *728*
visszalépés, 139, 140
visszavezethetőség, *133*, 139*gy*
Viterbi-algoritmus, 557
voxel, *626*

W

WAN, *lásd* hálózat
WARNOCK-ALGORITMUS, *679*
WORST-FIT, 469*gy*, 470*gy*
WS, *479*
WS-KIDOB, 479

Y
YCbCr-transzformáció, *447*

Z
zajmentes kódolás tétele, *422*
zártság, 101
z-buffer, *675*
z-buffer algoritmus, *675*
Z-BUFFER-ALGORITMUS, *676*
Z-BUFFER-ALSÓ-HÁROMSZÖG, *678*
zéró-ismeretű protokoll, *719*
Ziv–Lempel tömörítés, *440*
Z-transzformáció, 32

Színes ábrák és ismertetők

Ebben a fejezetben a kötet legfontosabb színes ábráit és a hazai informatikai felsőoktatás főszereplőit bemutató anyagokat gyűjtöttük össze.

Először a fekete-fehér alapszínű egyetemi ismertetők (Babeş-Bolyai Tudományegyetem, Budapesti Műszaki és Gazdaságtudományi Egyetem, Pécsi Tudományegyetem), azután a színes egyetemi ismertetők (Debreceni Egyetem, Eötvös Loránd Tudományegyetem, Eszterházy Károly Főiskola és Szegedi Tudományegyetem) következnek.

Ezután az 5. fejezet (*Hálózatok szimulációja*) fejezet hat, azután a 6. fejezet (*Párhuzamos számítások*) egy, majd a 16. (*Grafika*) fejezet négy és végül a 17. fejezet (*Térinformatika*) nyolc színes ábráját mutatjuk be – abban a sorrendben, ahogyan a könyvben előfordulnak.

BABEŞ-BOLYAI TUDOMÁNYEGYETEM
MATEMATIKAI ÉS INFORMATIKAI KAR

RO 400084 Cluj-Napoca, Str. Kogalniceanu 1
Tel: +40-264-405327, Fax: +40-264-591906,
Email: maths@math.ubbcluj.ro; Honlap:
http://www.cs.ubbcluj.ro

BBBE MATEMATIKAI ÉS INFORMATIKAI KARA

A kolozsvári **BBBE Matematikai és Informatikai Karán** a következő szakokon folyik alapképzés: **matematika** (román és magyar nyelven), **informatika** (román, magyar és angol nyelven), **matematika-informatika** (román, magyar és német nyelven), valamint **alkalmazott matematika** (román nyelven). A magyar tagozaton évente mintegy 80–100 hallgató kezdi el tanulmányait (matematika szakon 10–15, informatika szakon 40–50, matematika-informatika szakon 30-40). Ebből 70 körüli az államilag támogatott helyek száma. Azok a hallgatók, akik elvégzik a pedagógiai modult is, tanári diplomát is szerezhetnek.

A magyar tagozaton a hallgatók minden tárgyat tanulhatnak magyarul, de a választható tárgyak esetében (amelyek az egyes tagozatokon különbözhetnek) választhatnak más nyelvű tárgyakat is.

A karon mesterképzés (MSc) is folyik román, magyar és angol nyelven. Magyar nyelven **számítógépes matematika** szak létezik. 2005-től bevezetjük a bolognai egyezménynek megfelelő 3+2-es rendszert, azaz a 3 éves alapképzést és a 2 éves mesterképzést. Jelenleg az alapképzés 4 év, a mesterképzés pedig további 1 év.

A magyar tagozat oktatói igyekeznek magyar nyelvű szakkönyveket is beszerezni a kari könyvtár számára. Így sikerült néhány példányt szerezni az *Algoritmusok* (Műszaki Könyvkiadó, 1997), az *Osztott algoritmusok* (Kiskapu Kiadó, 2002), a *Párhuzamos algoritmusok* (ELTE Eötvös Kiadó, 2003) és az *Új algoritmusok* (Scolar Kiadó, 2003) című tankönyvekből is.

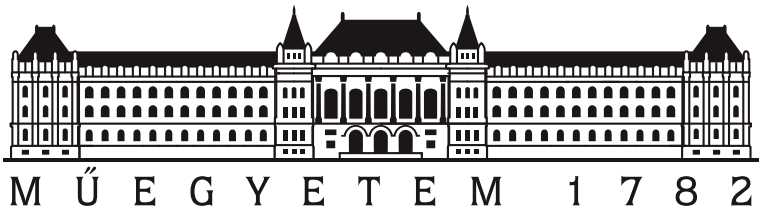
Az *Informatikai algoritmusok* című tankönyvet az informatikus alapképzés következő tárgyaihoz fogjuk használni: Algoritmika, Programozás, Adatszerkezetek, Algoritmusok elemzése és bonyolultsága, Numerikus módszerek, Operációkutatás, Számítógép-architektúrák, Operációs rendszerek, Mesterséges intelligencia, Az adatbázisok alapjai és Számítógépes grafika.

Kolozsvár, 2004. augusztus 18.

Dr. Kása Zoltán

egyetemi tanár

dékánhelyettes



BUDAPESTI MŰSZAKI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR

1117 Budapest, Magyar Tudósok krt. 2.
Tel: 463-1501, Fax: 463-3580, Email: vikdhvez@vdk.bme.hu

BME VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KARA

A **BME Villamosmérnöki és Informatikai Kar** alaptevékenységének meghatározó eleme az egyetemi mérnökképzés és az erre épülő doktori (PhD) képzés villamosmérnöki és műszaki informatika szakon. A mérnökképzésben évfolyamonként közel 500 villamosmérnöki, és nagyjából ugyanennyi informatikus hallgató vesz részt. A két egyetemi szintű szak oktatására az erős elméleti alapképzés, a mély szakmai ismereteket adó szakképzés, a széles területen gyakorlati tudást nyújtó laboratóriumi oktatás, és az önálló feladatok megoldására irányuló mérnöki gyakorlat a jellemző. A szakképzés során a villamosmérnöki hallgatók a beágyazott információs rendszerek, a híradástechnika, a távközlés és telematika, a számítógép technika, az irányítástechnika, a robotinformatika, valamint az elektronika és energetika területeinek valamelyikében mélyedhetnek el. Az informatikus hallgatók választható moduljai a rendszer- és szoftverfejlesztés, az infokommunikáció, a médiainformatika, az intelligens autonóm rendszerek és a gazdasági informatika.

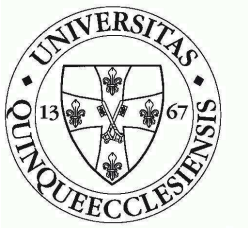
A villamosmérnöki és informatikai szakterület doktori képzésében a hallgatók – a kar erős, nemzetközileg is elismert tudományos iskoláihoz kapcsolódva – alapkutatással és alkalmazott kutatással foglalkoznak, de emellett szervezett szakmai kurzusokon folytatják tanulmányaikat és részt vállalnak a kar oktatási tevékenységében is.

Az *Informatikai algoritmusok* című tankönyv többek között az informatika szaknak a Bevezetés a számításelméletbe, Formális nyelvek, Algoritmusok elmélete, Adatbázisok, Operációs rendszerek, Számítógépes grafika és képfeldolgozás, valamint Kódelmélet tárgyaihoz, a villamosmérnöki szaknak pedig az Informatika, Számítástudomány alapjai, Számítógépes grafika és animáció nevű tárgyaihoz használható jegyzetként, illetve hasznos kiegészítő olvasmányként.

Budapest, 2004. augusztus 16.

Dr. ARATÓ PÉTER

egyetemi tanár
dékán



PÉCSI TUDOMÁNYEGYETEM
TERMÉSZETTUDOMÁNYI KAR,
MATEMATIKAI ÉS INFORMATIKAI INTÉZET
H-7624 Pécs, Ifjúság útja 6.
Telefon/Fax: +36 (72) 503-615,
E-mail: matinf@ttk.pte.hu

PÉCSI TUDOMÁNYEGYETEM MATEMATIKAI ÉS INFORMATIKAI INTÉZETE

A **PTE Természettudományi Karán** hosszabb ideje folyik **számítástechnika szakos tanárképzés,** elsősorban a technika, illetve matematika tanárszakkal együtt szakpárban. Az elmúlt évek jelentős fejlesztéseinek eredményeként önálló informatikus szak indítására került sor a 2003/2004 tanévben, ekkor indult el a TTK-n a **programozó matematikus** szak első évfolyama. Ezeknek a szakoknak a gazdája a **Matematikai és Informatikai Intézet,** amely ugyanakkor ellátja a Karon az informatika ismeretanyagának oktatását közismereti tantárgyként, illetve szaktárgyként a többi természettudományi szakos hallgatók számára.

Ebben az évben két új informatika orientált szak akkreditálására került sor a Természettudományi Karon – a 2004/2005 tanévtől kezdve elindul a képzés a **fizikus informatikus** és **könyvtáros informatikus** szakokon. A két szak közül az első az informatika komoly elméleti alapozását is megköveteli.

Jelenleg a **programtervező informatikus** alapképzési szak akkreditációs anyagának elkészítése van folyamatban, s az akkreditációs eljárásra történő benyújtása ez év szeptemberében fog megtörténni. A szak indítása a 2005/2006 tanévben várható.

Az *Informatikai algoritmusok* című könyvet (hasonlóan az *Új algoritmusok* című és más korábban elkészült könyvekhez) elsősorban a programtervező informatikus alapképzési szakon fogjuk használni tankönyvként. Várhatóan fontos segítséget fog jelenteni több tantárgy oktatásában: Számításelmélet, Operációkutatás, Adatbázisok elméleti alapjai, Számítógépes grafika, Numerikus módszerek.

A könyv számos fejezete hasznos oktatási anyagot képez a TTK-n jelenleg folyó matematika tanár és az ebben az évben induló fizikus informatikus képzéshez, továbbá a PTE más informatikai irányú képzéseihez is.

Pécs, 2004. július 15.

Dr. Szeidl László
egyetemi tanár
igazgató

DEBRECENI EGYETEM

INFORMATIKAI KAR

4010 Debrecen, Pf. 12. E-mail: office@inf.unideb.hu

Telefon: (52) 489-100, Fax: (52) 416-857

DEBRECENI EGYETEM INFORMATIKAI KARA

A **Debreceni Egyetem Informatikai Kara** a Kelet-Magyarországi régióban az egyetlen akkreditált egyetemi szintű informatikai szakemberképzés gazdája. A kar öt tanszékén (Alkalmazott Matematika és Valószínűség-számítás, Információ Technológia, Komputergrafika és Könyvtárinformatikai, Számítógépes Rendszerek és Hálózatok, Számítógéptudomány) dolgozó 6 professzor, 13 docens (főmunkatárs), 21 adjunktus (tudományos munkatárs), 14 tanársegéd és 21 felsőfokú végzettségű számítástechnikai munkatárs jelentős, nemzetközileg is jegyzett szellemi potenciált képvisel. A Informatikai Karon jelenleg

- **programtervező informatikus** (nappali, 6 félév, alapképzési BSc szint);
- **programozó matematikus** (esti, 8 félév, főiskolai szint);
- **programtervező matematikus** (nappali, 10 félév, egyetemi szint);
- **informatikatanár** (nappali, levelező, 10/6 félév, egyetemi szint);
- **informatikus könyvtáros** (nappali, levelező, 10/6 félév, egyetemi szint)

szakokon folyik képzés.

A programtervező informatikus, programozó- és programtervező matematikus szakokon olyan komplex szakmai elméleti ismeretekkel rendelkező szakemberek képzése a cél, akik képesek a mindennapi élet által felvetett gyakorlati problémák tudományos igényű modellezésére, a megfelelő megoldási módszerek megkeresésére, illetve kidolgozására, képesek ilyen feladatok elvégzésére szerveződött csoportok szakmai irányítására, rendelkeznek a szakterületükön folytatható kutatásokhoz szükséges alapvető elméleti-, módszertani- és nyelvismeretekkel. A programtervező informatikus alapképzési BSc szakon a képzés Magyarországon először 2004-ben a Kar gondozásában indult. Ennek, az ún. Bolognai folyamatnak a keretében folyó képzésnek fontos szerepe lesz abban, hogy hazánk az Európai Unió tagjaként szervezetileg is felzárkózzék az európai felsőoktatási térséghez. A szakindítás az informatikai képzés kínálatában elsősorban nem mennyiségi, hanem minőségi változást eredményez:

- A nemzetközi mércéhez alkalmazkodó képzési rendszer és végzettség jelentősen növeli végzett hallgatóink esélyeit a nemzetközi munkamegosztásba történő bekapcsolódásra.
- A mesterképzés (MSc) bevezetése után világos helyzetet teremt a továbbtanulásnál.

A Kar által gondozott tanárszakokon a cél olyan magas szintű informatikai, illetve pedagógiai és általános műveltséggel rendelkező tanárok képzése, akik szakirányú és pedagógiai ismereteik alapján képesek az informatika tanítás minden szintjén feladatok ellátására. A leendő tanár képes lesz a hazai tanítás értékes hagyományainak és színvonalának megőrzésére, a képzésben fontos szerepet játszik továbbá a kitekintés e szakterület oktatásának európai hagyományaira is.

Informatikus könyvtáros szakon olyan felsőfokú szakemberek képzése a cél, akik tanulmányaik során elsajátították a könyvtár- és tájékoztatástudomány korszerű elméleti alapismereteit, továbbá az önálló gyakorlati alkalmazásukhoz szükséges ismereteket, beleértve az információkezelést, illetve az erre vonatkozó kutatások módszertanát is.

A Kar saját szakjain oktatott *hallgatók létszáma* jelentős, jelenleg 1700. Ezonfelül teljes egészében az Informatikai Kar gondozza más karok, így a Közgazdaságtudományi, Jogi, Műszaki Főiskolai Kar informatikai kurzusait is. Munkatársaink jelentős szerepet játszanak a hat programmal működő Matematika és Számítástudományok doktori (PhD) iskolában is évi 8–10 nappali és 10–20 levelező hallgatót beiskolázva.

A Kar kiterjedt nemzetközi kapcsolati lehetővé teszik, hogy évente több tucat hallgatónk vegyen részt külföldi részképzésben Európa és a világ jó nevű felsőoktatási intézményeiben.

Munkatársaink gyümölcsöző hazai szakmai együttműködésének egy szép példája az *Informatikai algoritmusok* című könyv elkészítése is, amelyet minden bizonnyal Debrecenben is haszonnal tanulmányoznak majd az érintettek.

A **Debreceni Egyetem** klasszikus értelemben vett egyetem. 18 karán és kari jogállású intézetében nemcsak elszigetelt szakemberképzés folyik, hanem a szakterületek együttműködésén, a sokszínűség kihasználásán alapuló értelmiségképzés. Pezsgő kulturális életével, szabadidős kínálatával ebben méltó partnere az egyetemnek Debrecen városa is.

Debrecen, 2004. augusztus. 16.

Dr. Fazekas Gábor
mb. dékánhelyettes

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

1117 Budapest, Pázmány Péter sétány I/C.

Tel: 381-2139, 381-2222 Fax: 381-2140

Az **ELTE Informatikai Karán** jelenleg a nappali tagozaton **programtervező matematikus** (a 2003/2004-es tanévben 1847 államilag finanszírozott hallgató), **informatikatanár** (190 hallgató) és **térképész** (96 hallgató) egyetemi szakon folyik képzés.

Ugyanebben a tanévben az esti tagozaton **programozó matematikus** főiskolai alapszakon (332 hallgató), **számítástechnika-tanár** főiskolai alapszakon (57 hallgató) és levelező tagozaton (228 hallgató) is tanultak hallgatók. Az Informatikai Doktori Iskolának 23 államilag támogatott hallgatója volt.

A Kari és az Egyetemi Tanács jóváhagyta a **programtervező informatikus** alapszak (BSc) akkreditációs pályázatát. A szak indításának tervezett időpontja a 2005/2006-os tanév.

A Kar legfontosabb rendszeres rendezvényei az **Ipari Nap** (a 2003/2004-es tanévben az **ELTE Eötvös Napok** keretében volt május 13-án), a **Neumann-nap** (2003. november 6-án volt az első, hagyományteremtő szándékkal) és a **Térinformatikai Világnap** programjában való részvétel (2003. november 19.).

Az Informatikai Kar oktatói aktív szerepet játszottak az *Algoritmusok* (Műszaki Könyvkiadó, 1997), az *Osztott algoritmusok,* (Kiskapu Kiadó, 2002), a *Párhuzamos algoritmusok* (ELTE Eötvös Kiadó, 2003), a *Programozási nyelvek* (Kiskapu Kiadó, 2003) és az *Új algoritmusok* (Scolar Kiadó, 2003) című tankönyvek megjelenésében.

Az *Informatikai algoritmusok* című tankönyvet a programtervező informatikus alapképzés Bevezetés a matematikába, Programozás módszertani alapjai, Számításelmélet, Numerikus módszerek, Operációkutatás, Architektúrák és operációs rendszerek, Az adatbázisok elméleti alapjai és a Számítógépes grafika című tantárgyaihoz fogjuk tankönyvként használni.

A jelenlegi programtervező matematikus mesterképzésben a tankönyvet az Adatbázisrendszerek, Grafika, Hálózatok, Komputeralgebra, Mesterséges intelligencia, Operációkutatás, Párhuzamos rendszerek, Térinformatika sávok oktatásában hasznosítjuk.

2004-től kezdve az Oktatási Minisztérium támogatja elektronikus tankönyvek létrehozását. A pályázaton 4 informatikai és 6 matematikai könyv kapott támogatást. Ezek közül 4 elektronikus könyvnek – Fóthi Ákos és Horváth Zoltán (*Bevezetés a programozásba*), Iványi Antal (szerkesztő – *Informatikai algoritmusok I.*), Iványi Antal (*Párhuzamos algoritmusok*), Stoyan Gisbert (*Numerikus módszerek I.*) – és három nyomtatott könyvnek – Demetrovics János és Békéssy András (*Adatbázisrendszerek*), Iványi Antal (*Informatikai algoritmusok I., II.*) és Stoyan Gisbert (*Mathlab 6.5.*) – a szerzői az ELTE IK oktatói. Ugyancsak támogatást kapott Klinghammer István, Mosonyi László és Török Zsolt *A Kárpát-medence tektonikus térképei* című CD-je.

Budapest, 2004. szeptember 15.

Dr. Kozma László
dékán

Eszterházy Károly Főiskola
Természettudományi Kar
MATEMATIKAI ÉS INFORMATIKAI INTÉZETE

3300 Eger, Leányka út 4. E-mail: **matinf@ektf.hu**
Telefon:06 (36) 520-400 / 4223 Fax: 06 (36) 520-478

ESZTERHÁZY KÁROLY FŐISKOLA MATEMATIKAI ÉS INFORMATIKAI INTÉZETE

Az MM 51018/1989.-XVI. sz. ügyiratában Pusztai Ferenc az Akadémia egyetértésével 1989 szeptember 1-től a EKF előterjesztésére számítástechnika szakot alapított, amelynek alapján a főiskolánkon az 1989-90-es tanévben matematika-számítástechnika szakon az országban elsőként elindult a szakos tanárok képzése. A későbbi évek során valamennyi főiskola az engedélyezett tanterv és szakalapítás alapján elindította a **számítástechnika szakos tanárképzést.** Az eltelt több mint 10 évben közel 1000 számítástechnika szakos általános iskolai tanári diplomát adtunk ki nappali és esti tagozaton.

A felhalmozott óriási szakmai tapasztalat és a Debreceni Egyetem támogatásának eredményeként a főiskolák között először 2002/2003-as tanévben elindult a **programozó matematikus** szak első évfolyama nappali és esti tagozaton.

2004 őszén beadjuk a **programtervező informatikus alapszak (BsC)** 2005-ös indításának kérelmét.

Főiskolánk 2004-ben akkreditálta az informatikus könyvtáros egyetemi szakot, így a meglévő főiskolai szakok mellett már egyetemi szinten is tanulhatnak intézményünkben informatikát a hallgatók.

A Matematikai és Informatikai Intézetnek 12 főállású, teljes munkaidős közalkalmazotti munkaviszonnyal rendelkező minősített oktatója van, 1 éven belül további 2 fő, 3 éven belül további 4 fő szerzi meg a PhD tudományos fokozatot. Ezzel biztosított a minőségi informatikai képzés iránti elkötelezettség.

Az *Informatikai algoritmusok* című könyvet elsősorban a számítástechnika tanári szakon és a programtervező informatikus alapszakon fogjuk használni. 2004-ben az Oktatási Minisztérium által meghirdetett pályázat támogatja az elektronikus tankönyvek létrehozását. A pályázaton négy elektronikus informatikai könyv kapott támogatást, amelyből egy az *Informatikai algoritmusok*.

Az Eszterházy Károly Főiskola oktatóinak gondozásában készülő *C# programozási nyelv* című elektronikus tankönyv szintén a sikeres pályázók körébe tartozik.

Az intézettel és a képzésekkel kapcsolatos további információk elérhetők az interneten a **http://matinf.ektf.hu** címen.

Eger, 2004. szeptember 10.

Dr. Kovács Emőd

tanszékvezető főiskolai docens

SZEGEDI TUDOMÁNYEGYETEM
TERMÉSZETTUDOMÁNYI KAR
INFORMATIKAI TANSZÉKCSOPORT

6720 Szeged, Árpád tér 2.
Levelezési cím: 6701 Szeged, Postafiók 652.
Telefon: +36 62 546396, Fax: +36 62 546-397
E-mail: depart@inf.u-szeged.hu

SZEGEDI TUDOMÁNYEGYETEM INFORMATIKAI TANSZÉKCSOPORTJA

A Szegedi Tudományegyetem Természettudományi Kara 1921-ben kezdte meg működését, amikor Szeged városa befogadta a Kolozsvárról áttelepült tudományegyetemet. A Természettudományi Kar oktatási és kutatási tevékenységét tudományterületi alapon felosztotta intézetei, tanszékcsoportjai között. Hat tanszékcsoport működik, amelyek mindegyike részben önálló kutató és oktató intézetként is felfogható.

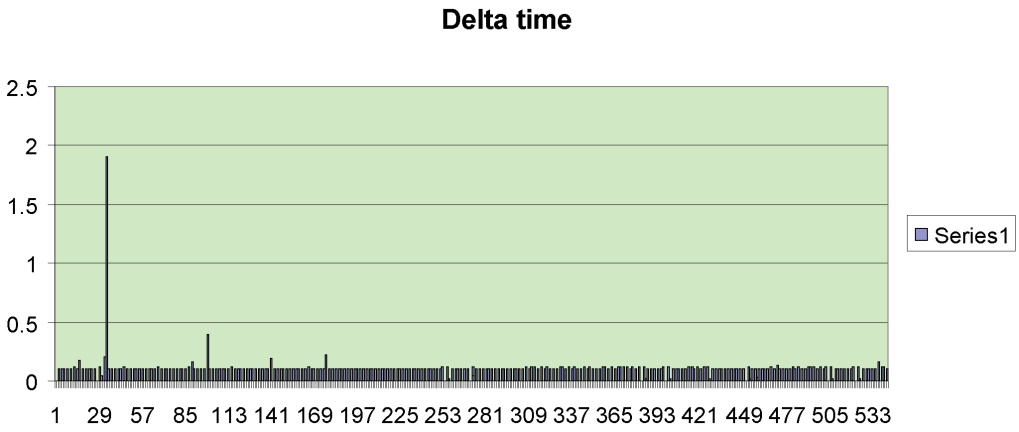
A Szegedi Tudományegyetem képzési rendszerén belül az Informatikai Tanszékcsoport öt, az informatika tudományághoz tartozó szaknak a gazdája: **programozó matematikus szak, programtervező matematikus szak, közgazdasági programozó matematikus szak, műszaki informatikai szak** és **informatika tanár szak.** Az Informatikai Tanszékcsoport 1990-ben alakult, ekkor vette át a nagy múltú szegedi informatikusképzés gondozását a TTK Matematikai Tanszékcsoportjától. A öt tanszéket *(Alkalmazott Informatika Tanszék, Képfeldolgozás és Számítógépes Grafika Tanszék, Számítástudomány Alapjai Tanszék, Számítógépes Algoritmusok és Mesterséges Intelligencia Tanszék, Szoftverfejlesztés Tanszék)* magában foglaló Informatikai Tanszékcsoport oktató- és kutatómunkáját az MTA-SZTE Mesterséges Intelligencia Tanszéki Kutatócsoport is segíti. A hallgatóink előtt is nyitva álló intézeti könyvtárunk folyamatosan bővül, könyvállománya meghaladja az 5000 kötetet és közel 200 tudományos folyóirat is elérhető. A Tanszékcsoport a gyakorlati oktatásra kiválóan felszerelt számítógépes kabinettermeket üzemeltet, amelyekben a hallgatók a tantervi órákon kívül is dolgozhatnak és számukra a szabad Internet-hozzáférés is biztosított.

A hallgatóink bekapcsolódhatnak a tanszékeken folyó kutatásokba, és ezt végzés után folytathatják PhD képzés keretében. A tudományos munka eredményeként rendszeresen színvonalas Országos Tudományos Diákköri dolgozatok születnek. A Tanszékcsoport lehetőség szerint támogatja a hallgatók külföldi részképzését. Ennek érdekében külföldi partnerekkel több nemzetközi projektben vesz részt az Intézet, melynek eredményeként számos hallgató élhet a külföldi részképzés lehetőségével.

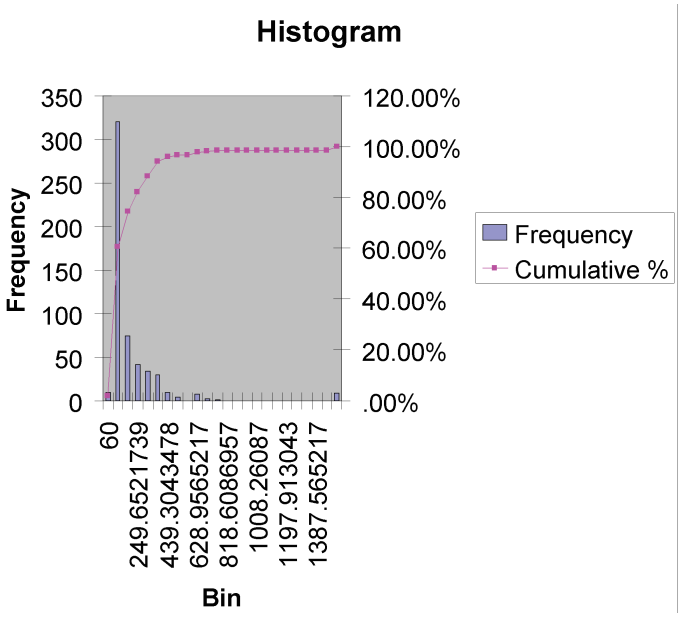
Szeged város pezsgő sport és kulturális élete ragyogó lehetőségeket nyújt a tanulás mellett pihenni, kikapcsolódni és szórakozni vágyó hallgatóinknak.

Szeged, 2004. szeptember 2.

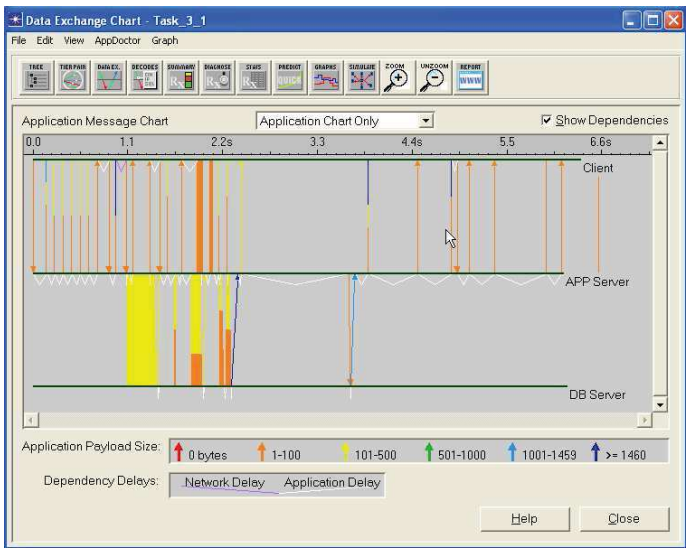
Dr. Csirik János
tanszékcsoportvezető egyetemi tanár



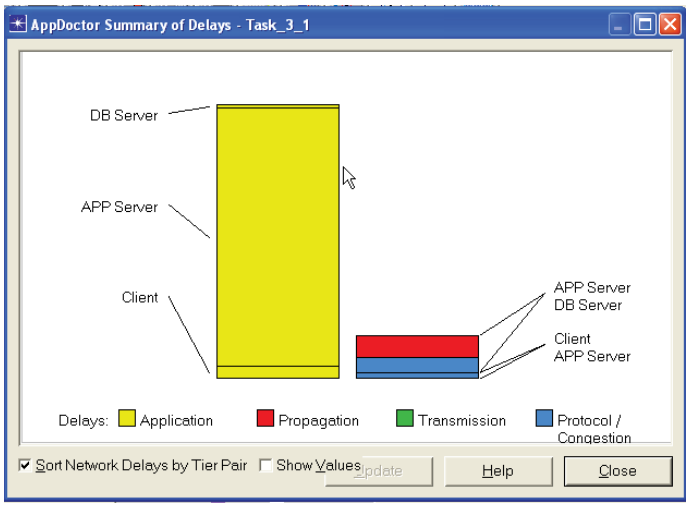
19.1. ábra. A beérkezési időközök nagy változékonysága. Az 5.10. ábra színes változata.



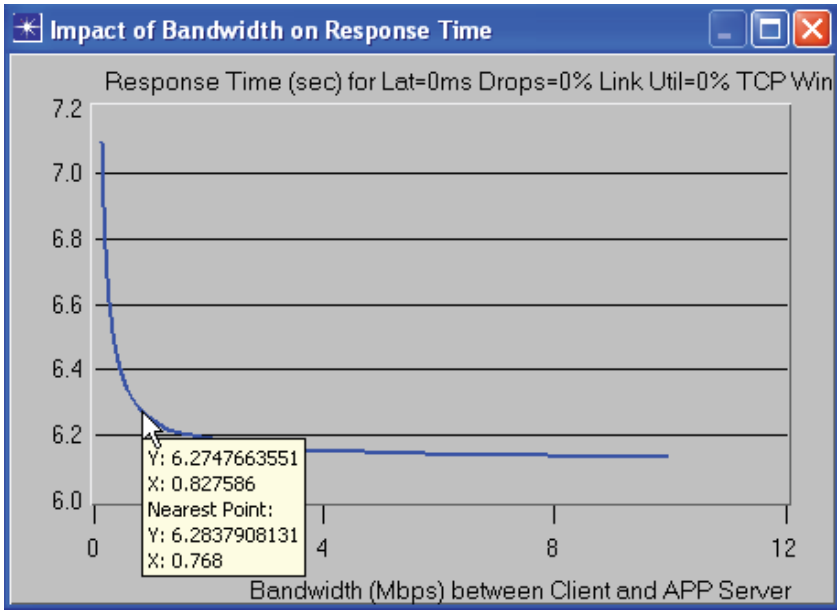
19.2. ábra. A kerethosszak hisztogramja. Az 5.11. ábra színes változata.



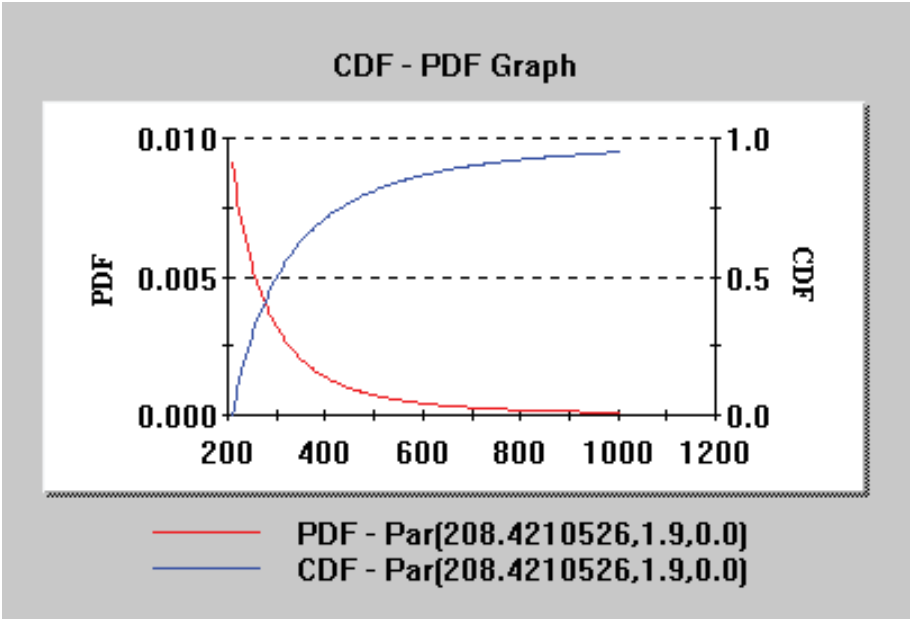
19.3. ábra. Adatcsere diagram. Az 5.15. ábra színes változata



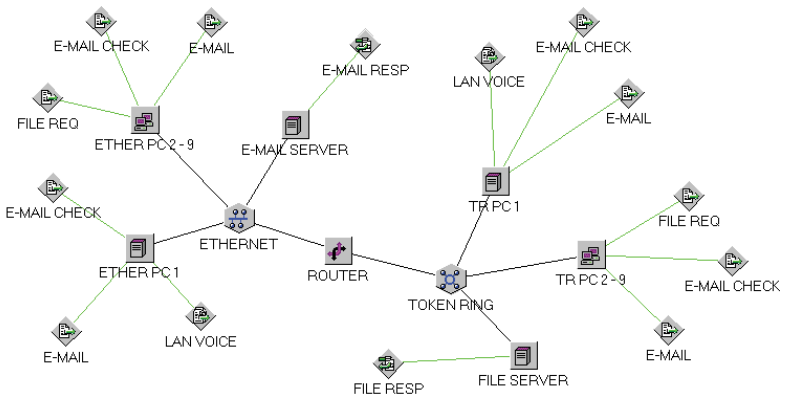
19.4. ábra. Késleltetések összegzése. Az 5.16. ábra színes változata



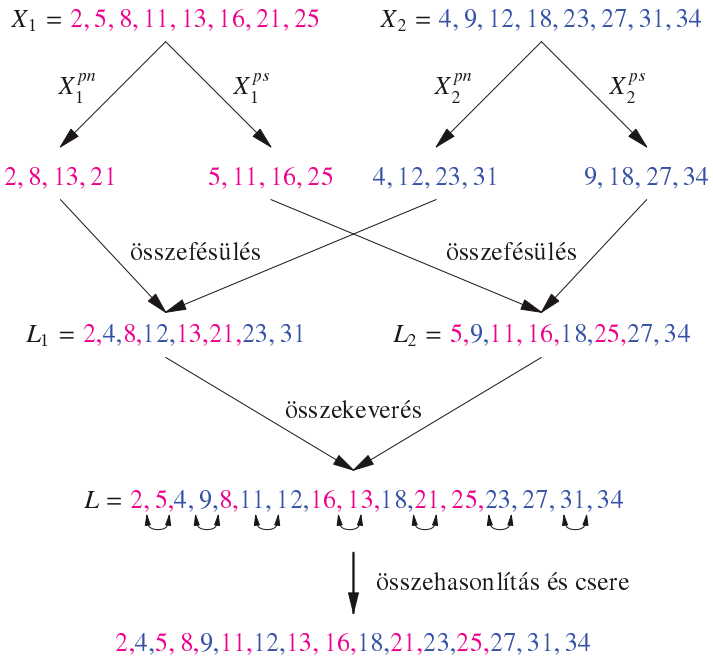
19.5. ábra. A sávszélesség hozzáadásának hatása a válaszidőre. Az 5.19. ábra színes változata.



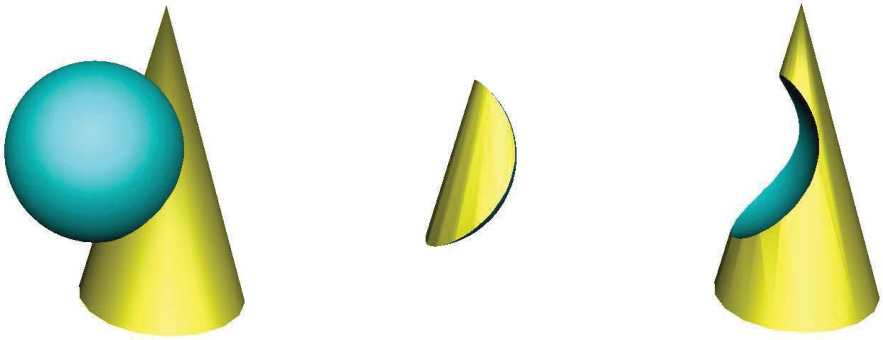
19.6. ábra. A Pareto-eloszlás 400 bájt várható érték és $H = 0.55$ Hurst-paraméter esetén. Az 5.28. ábra színes változata.



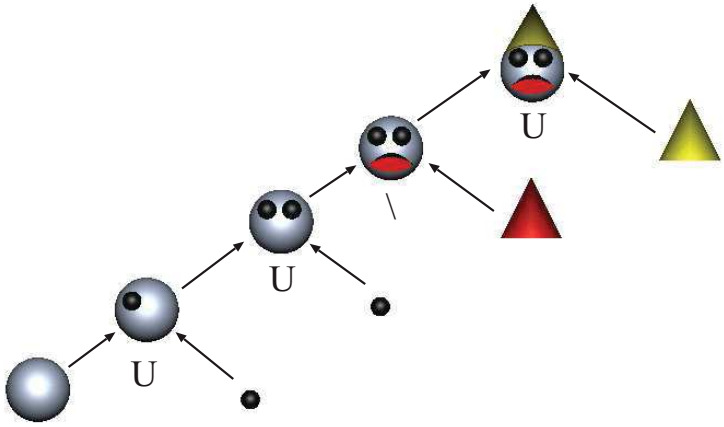
19.7. ábra. Hálózati topológia. Az 5.50. ábra színes változata.



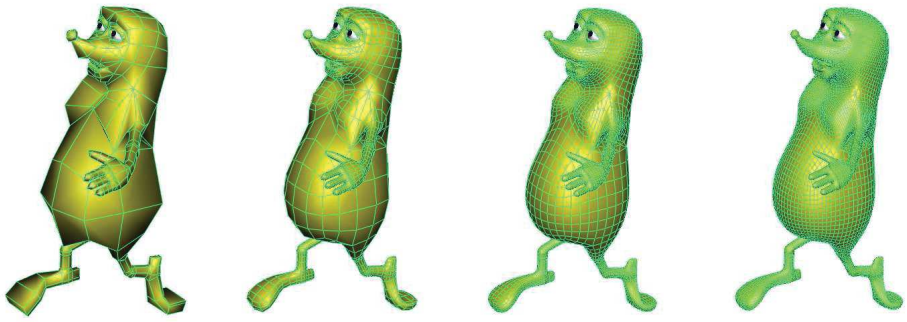
19.8. ábra. 16 szám rendezése a PÁRATLAN-PÁROS-ÖSSZEFÉSÜL algoritmussal. A 6.8. ábra színes változata.



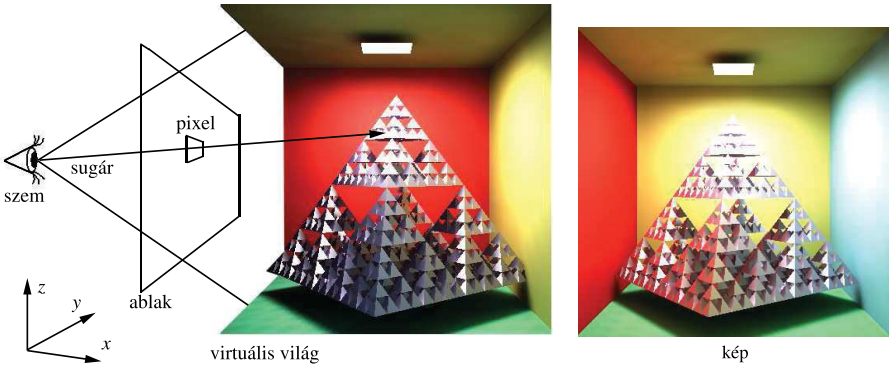
19.9. ábra. A konstruktív tömörtest geometria alapműveletei egy f implicit függvényű kúpra és egy g implicit függvényű gömbre: egyesítés ($\max(f,g)$), metszet ($\min(f,g)$), halmaz-különbség ($\min(f,-g)$). A 15.9. ábra színes változata.



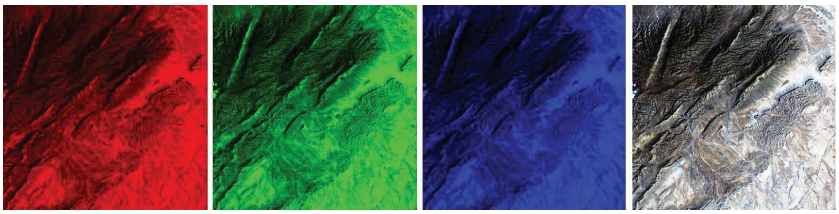
19.10. ábra. Összetett objektum felépítése halmazműveletekkel. A ***CSG-fa*** gyökere az összetett objektumot, a levelei a primitíveket azonosítják. A többi csomópont a halmazműveleteket adja meg (U: egyesítés, /: különbség). A 15.10. ábra színes változata.



19.11. ábra. Az eredeti háló, valamint egyszer, kétszer és háromszor felosztott változatai. A 15.19. ábra színes változata.

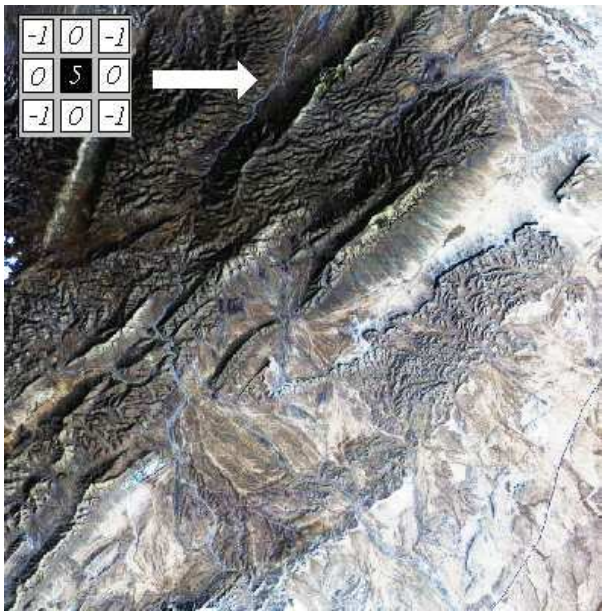


19.12. ábra. Képszintézis virtuális kamerával. A 15.30. ábra színes változata.

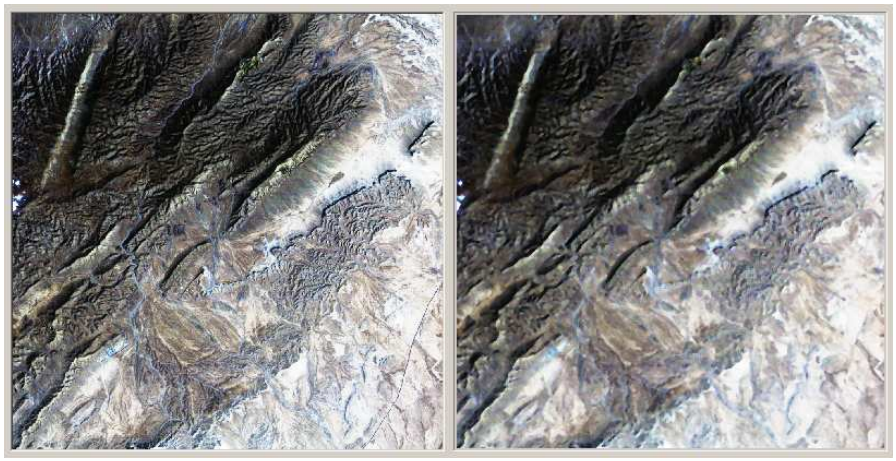


19.13. ábra. Egy LANDSAT műhold felvétele: az RGB sávok és a sávokból összeállított kép. A 16.14. ábra színes változata

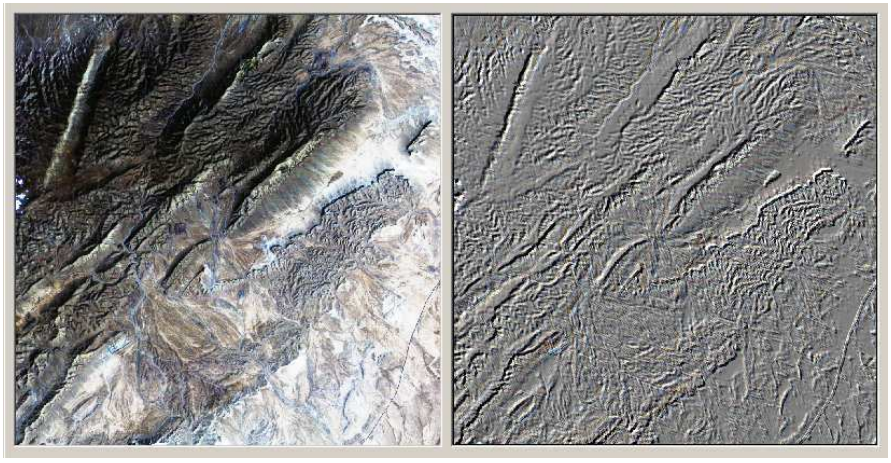
19.14. ábra. A hisztogram kiegyenlítéssel megnövelt kontrasztú kép. A 16.19. ábra színes változata.



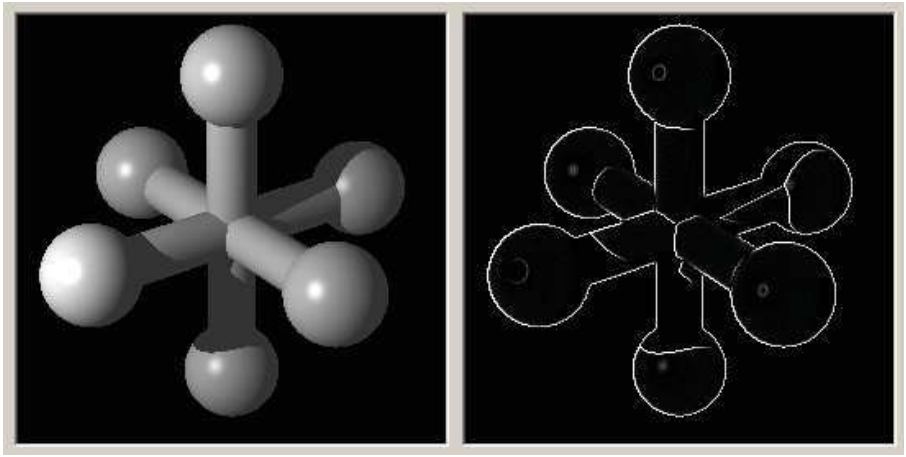
19.15. ábra. Az erősen felnagyított kernel mozgása a képen (LANDSAT képrészlet). A 16.24. ábra színes változata.



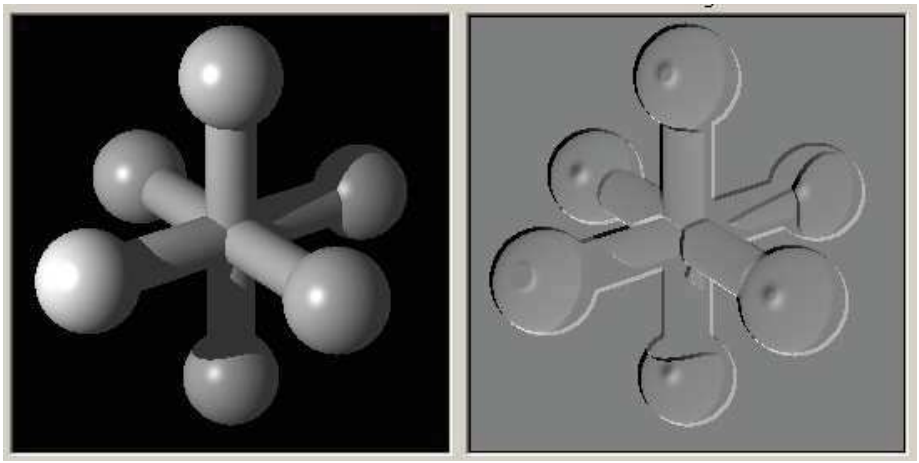
19.16. ábra. A bal oldali kép az eredeti, míg a jobb oldali egy kilenc pontos (9 × 9 pixeles) medián-szűrt kép. A 16.30. ábra színes változata.



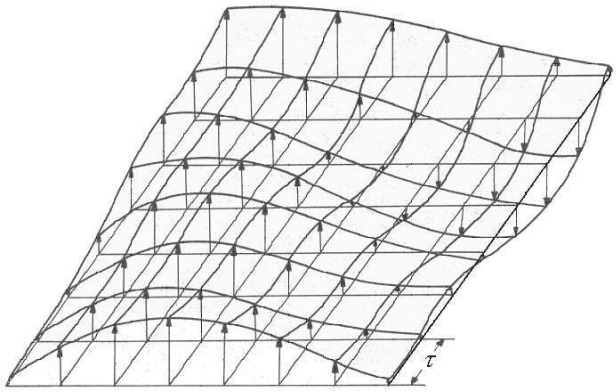
19.17. ábra. A bal oldali kép az eredeti, a jobb oldali az emboss-szűrt változat. A 16.31. ábra színes változata.



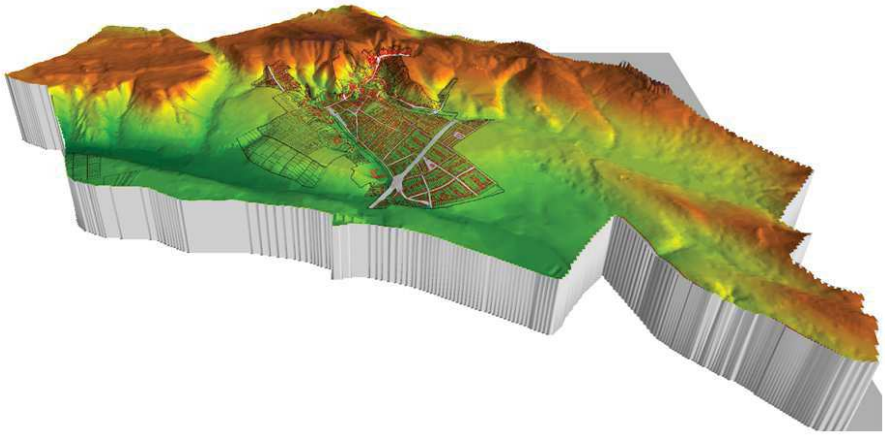
19.18. ábra. A bal oldali az eredeti kép, a jobb oldali a Laplace-szűrt változata. A 16.32. ábra színes változata.



19.19. ábra. A bal oldali az eredeti kép, a jobb oldali az emboss-szűrt változata. A 16.33. ábra színes változata.



19.20. ábra. A τ rácsállandójú Dirac-δ sorozattal mintavételezett felszín. A 16.35. ábra ismétlése



19.21. ábra. Törökbálint domborzati modelljének perspektivikus megjelenítése északi irányból. A 16.36. ábra színes változata.