INFORMATIKAI ALGORITMUSOK

3. kötet. Adatbázisok és alkalmazások

Második bővített és javított kiadás

HOUNTLER Kft.
Budapest, 2015

A könyv a Magyar Tudományos Akadémia támogatásával készült

Szerkesztő: Iványi Antal

Szerzők: Balogh Ádám (34. fejezet), Demetrovics János (33. és 35. fejezet), Bodon Ferenc (31. fejezet), Elek István (41. fejezet) Fogaras Dániel (32. fejezet), Galántai Aurél (42. fejezet), Gombos Gergő (36. fejezet) Iványi Antal (34. fejezet), Kiss Attila (30., 36. és 37. fejezet), Kósa Balázs (37. fejezet), Lukács András (32. fejezet), Matuszka Tamás (37. fejezet), Miklós István (38. fejezet), Pinczel Balázs (37. fejezet), Rácz Gábor (36. fejezet), Sali Attila (33. és 35. fejezet), Sidló Csaba (41. fejezet), Szirmay-Kalos László (40. fejezet), Ulrich Tamm (29. fejezet)

Lektorok: Antal György (35. fejezet), Benczúr A. András (30. fejezet), Csirik János (28. fejezet), Fridli Sándor (29. fejezet), Iványi Anna Barbara (Irodalomjegyzék), Katsányi István (38. fejezet) , Kiss Attila (33. és 35. fejezet), Mayer János (25. fejezet), Rónyai Lajos (31. fejezet), Szántai Tamás (39. fejezet), Varga László Zsolt (34. fejezet)

Nyelvi lektor: Bíró Gabriella

Kiadó: Hountler Kft.

Tartalomjegyzék

Előszó

Nagy örömmel ajánlom az Olvasók figyelmébe az *Informatikai algoritmusok* című tankönyvet, Iványi Antal gondos szerkesztésében. A számítógépes algoritmusok az informatika igen fontos és igen gyorsan fejlődő területét alkotják. Hatalmas hálózatok tervezése és üzemeltetése, nagyméretű tudományos számítások és szimulációk, gazdasági tervezés, adatvédelmi módszerek, titkosítás és még sok más alkalmazás igényel hatékony, gondosan tervezett és pontosan elemzett algoritmusokat.

Sok évvel ezelőtt Gács Péterrel írtunk egy kis könyvecskét [237] *Algoritmusok* címmel. Az *Informatikai algoritmusok* három kötete mutatja, hogy milyen sokrétű és szerteágazó területté fejlődött ez a téma. Külön örömet jelent, hogy a magyar informatika ilyen sok kiváló képviselője fogott össze, hogy ez a könyv létrejöjjön. Nyilvánvaló számomra, hogy diákok, kutatók és alkalmazók egyik legfontosabb forrásmunkája lesz hosszú ideig.

Budapest, 2014. október
Lovász László

Bevezetés a 3. kötethez

A harmadik kötet adatbáziskezeléssel és alkalmazásokkal foglalkozik és tizennégy fejezetet tartalmaz.

A kötet Ulrich Tamm (Chemnitzi Egyetem) *Adattömörítés* című 29. fejezetével kezdődik, majd Gyires Kiss Attila (ELTE) *Félig strukturált adatbázisok* című, 30. fejezetével folytatódik.

A *Gyakori elemhalmazok keresése* című **??**. fejezet szerzője Bodon Ferenc (BME), míg a *Klaszterezés* című 32. fejezeté Lukács András (ELTE).

A *Lekérdezés átírás relációs adatbázisokban* című **??**. fejezetet Demetrovics János (MTA SZTAKI) és Sali Attila (MTA Rényi Alfréd Matematikai Kutatóiintézet) írták. Balogh Ádám (Ericsson Hungary Limited) és Iványi Antal (ELTE) a szerzői a *Memóriagazdálkodás* című **??**. fejezetnek. A következő, 35. fejezet címe Relációs adatmodell tervezés, szerzői Demetrovics János és Sali Attila.

A *Szemantikus web alkalmazásai* című 36. fejezet Rácz Gábor, Gombos Gergő és Kiss Attila műve, a *Szemantikus web lekérdező nyelvei* című **??**. fejezeté pedig Pinczel Balázs, Kósa Balázs, Matuszka Tamás, és Kiss Attila (mindannyian ELTE).

A 38. fejezet címe *Bioinformatika*, szerzője Miklós István (MTA Rényi Alfréd Matematikai Kutatóiintézet), míg az *Ember-gép kapcsolat* című **??**. fejezet szerzői Ingo Althöfer és Stefan Schwarz (Jénai Friedrich Schiller Egyetem). A következő, 40. fejezet címe *Számítógépes grafika*, szerzője pedig Szirmay-Kalos László (BME).

A kötet a *Térinformatika* című **??**. és a *Tudományos számítások* című **??**. fejezettel zárul. Előbbi szerzői Elek István (ELTE) és Sidló Csaba (MTA SZTAKI), utóbbi szerzői Galántai Aurél (Óbudai Egyetem) és Jney András (Miskolci Egyetem). A LaTeX style fájlt Belényesi Viktor, Csörnyei Zoltán, és Iványi Antal írták. Az ábrákat a szerzők, Csergő Bálint, Iványi Antal jr., Kása Zoltán, Locher Kornél és Mathauser András rajzolták. A könyv ugrópontjait Iványi Anna Barbara lektorálta és egészítette ki.

A **??**. fejezetet Sike Sándor fordította.

A kolofon oldalon lévő ugrópontok segítségével az Olvasók kapcsolatba

léphetnek a kötet alkotóival. Új gyakorlatokat és feladatokat, valamint a tartalomra vonatkozó észrevételeket köszönettel fogadunk.

A nyomtatott első két kötet és az elektronikus első kötet megjelenését az Oktatási Minisztérium,, a harmadik nyomtatott kötet megjelenését a Nemzeti Kulturális Alap, ezeknek az elektronikus kötetnek a megjelenését pedig a Magyar Tudományos Akadémia, a Neumann János Számítógéptudományi Társaság támogatta. Mindhárom kötet megjelent angolul is, mind nyomtatott, mind pedig elektronikus formában. Ezek megjelenését az Európai Unió (az Európai Szociális Alap társfinanszírozásával), a Magyar Tudmányos Akadémia és a Nemzeti Kulturális Alap támogatta.

A későbbiekben szeretnénk az eddigi nyomtatott és elektronikus kiadások 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 új szöveget. A javított kiadásokban minden hiba első felfedezőjének megköszönjük a segítségét.

Olvasóink javaslataikkal, kérdéseikkel megkereshetik a könyv alkotóit is (címük megtalálható a kolofonoldalon).

Budapest, 2015. január

Iványi Antal (tony@inf.elte.hu)

Tartalomjegyzék

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.

29. 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.

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

29.1. ábra. Jelek gyakorisága egy 1000 jelet tartalmazó angol szövegben.

29.1. Információelméleti eredmények

29.1.1. 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 29.1. 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 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.

29.1.2. 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 $c(x) = \big(c_1(x), c_2(x), \ldots, c_{L(x)}(x)\big)$ 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 $L(x) \leq L(y)$, $\big(c_1(x), c_2(x), \ldots, c_{L(x)}(x)\big) \neq \big(c_1(y), c_2(y), \ldots, c_{L(x)}(y)\big)$ 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 $P = \big(P(1), P(2), \ldots, P(a)\big)$ 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 (29.1 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.

29.1.3. 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.

29.1. 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*

$$\sum_{x\in\mathcal{X}} 2^{-L(x)} \leq 1.$$

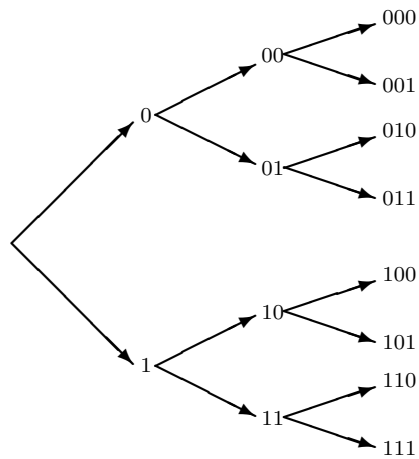
Bizonyítás. A bizonyítás alapötlete, hogy a kódszavakat egy $T = \max_{x\in\mathcal{X}}\{L(x)\}$ mélységű bináris fa csúcsainak tekintjük. A fának teljesnek és regulárisnak kell lennie, azaz a gyökérből levélbe vezető utak összhossza T, és minden belső csúcs fokszáma 3. Ezt szemlélteti a 29.2. ábrán 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 29.2. 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

[1] Ha a fát más elrendezésben, felülről-lefelé rajzoljuk.



29.2. ábra. Egy lehetséges kódfa.

nem így lenne, találhatnánk egy $x_1x_2\dots x_T$ szót, amelynek két kódszó is a prefixe. A kódszavak hossza legyen s és t, és feltehetjük, hogy $s < t$. Ekkor a két kódszó $x_1x_2\dots x_s$ és $x_1x_2\dots x_t$, 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

$$\sum_{x\in\mathcal{X}} 2^{T-L(x)} \le 2^T \ ,$$

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 $L(1),\dots,L(a)$ pozitív egészek. Tegyük fel továbbá, hogy $L(1) \le L(2) \le \dots \le L(a)$. Az első kódszó legyen

$$c(1) = \underbrace{00\dots0}_{L(1)} \ .$$

Mivel

$$\sum_{x\in\mathcal{X}} 2^{T-L(x)} \le 2^T$$

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.

29.2. 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 $\sum_{x \in \mathcal{X}} 2^{-L(x)} = \sum_{j=1}^{T} w_j \; 2^{-j}$, 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 $N_k = \sum_{i_1 + \cdots + i_s = k} w_{i_1} \ldots w_{i_s}$ 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 $\sum_{k=s}^{T \cdot s} N_k \; 2^{-k} \leq \sum_{k=s}^{T \cdot s} 1 = T \cdot s - s + 1 \leq T \cdot s$. s-edik gyököt vonva kapjuk, hogy $\sum_{j=1}^{T} w_j \; 2^{-j} \leq (T \cdot s)^{1/s}$.

Ez az egyenlőtlenség bármely s értékre fennáll, és $\lim_{s \longrightarrow \infty} (T \cdot s)^{1/s} = 1$, így a

$$\sum_{j=1}^{T} w_j \; 2^{-j} = \sum_{x \in \mathcal{X}} 2^{-L(x)} \leq 1 \; .$$

eredményhez jutunk. ■

29.3. 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

$$D(P||Q) = -H(P) - \sum_{x \in \mathcal{X}} P(x) \cdot \lg\left(2^{-L(x)} \cdot r^{-1}\right) \geq 0 \ .$$

Ebből következik, hogy

$$\begin{aligned} H(P) &\leq -\sum_{x \in \mathcal{X}} P(x) \cdot \lg\left(2^{-L(x)} \cdot r^{-1}\right) \\ &= \sum_{x \in \mathcal{X}} P(x) \cdot L(x) - \sum_{x \in \mathcal{X}} P(x) \cdot \lg r^{-1} = L_{min}(P) + \lg r \ . \end{aligned}$$

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: $L'(x) = \lceil -\lg P(x) \rceil$ $(x = 1, \ldots, a)$. 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.

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

29.3. ábra. Példa a Shannon kódra és a Shannon-Fano-Elias kódra.

29.1.4. 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.

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 29.3. ábrán látható példával szemléltetjük ezeket az eljárásokat.

Shannon és Fano közzétett egy hatékonyabb eljárást. A ***Shannon-Fano algoritmus*** működését szemlélteti az előző példán a 29.4. 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

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

29.4. ábra. Példa a Shannon-Fano algoritmus működésére.

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 29.4. 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.

29.1.5. A Huffman-algoritmus

A ***Huffman-algoritmus*** egy rekurzív eljárás, amelynek működését a 29.5. 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 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.

29.4. 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 opti-

p_1	0.25	p_1	0.25	p_1	0.25	p_{23}	0.31	p_{4567}	0.44	p_{123}	0.56
p_2	0.2	p_{67}	0.22	p_{67}	0.22	p_1	0.25	p_{23}	0.31	p_{4567}	0.44
p_3	0.11	p_2	0.2	p_{45}	0.22	p_{67}	0.22	p_1	0.25		
p_4	0.11	p_3	0.11	p_2	0.2	p_{45}	0.22				
p_5	0.11	p_4	0.11	p_3	0.11						
p_6	0.11	p_5	0.11								
p_7	0.11										

c_{123}	0	c_{4567}	1	c_{23}	00	c_1	01	c_1	01	c_1	01
c_{4567}	1	c_{23}	00	c_1	01	c_{67}	10	c_{67}	10	c_2	000
		c_1	01	c_{67}	10	c_{45}	11	c_2	000	c_3	001
				c_{45}	11	c_2	000	c_3	001	c_4	110
						c_3	001	c_4	110	c_5	111
								c_5	111	c_6	100
										c_7	101

29.5. ábra. Példa a Huffman-kódra.

mális prefix kód. Definiáljuk a P eloszláshoz a c prefix kódot a következőképpen:

$$\begin{aligned} c(x) &= c'(x) \quad ha \quad x = 1, \ldots, a-2\ , \\ c(a-1) &= c'(a-1)0\ , \\ c(a) &= c'(a-1)1\ . \end{aligned}$$

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

$$\begin{aligned} \overline{L}(c') - \overline{L}(c) &= P(x) \cdot L(y) + P(y) \cdot L(x) - P(x) \cdot L(x) - P(y) \cdot L(y) \\ &= (P(x) - P(y)) \cdot (L(y) - L(x)) \leq 0\ . \end{aligned}$$

(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

29.1-1. 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.

29.1-2. Határozzuk meg az $(\mathcal{X}, P)$ forrás entrópiáját, ha $\mathcal{X} = \{1, 2\}$ és $P = (0.8, 0, 2)$.

29.1-3. 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.

29.1-4. Bizonyítsuk be, hogy $0 \leq H(P) \leq \lg |\mathcal{X}|$.

29.1-5. 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.

29.1-6. 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.

29.2. 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.

29.2.1. Aritmetikai kódolás

Az aritmetikai kódolás alapgondolata, hogy egy $x^n = (x_1 \dots x_n)$ 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       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      j = 1
3      while c < Q_i(j)
4          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
       ...
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

[2]Nincs olyan pontosságú aritmetika, amellyel a probléma gyakorlati esetekben kezelhető lenne. *A fordító.*

bitjei megegyeznek, azaz az intervallum teljes egészében a $[0, 1/2)$ vagy az $[1/2, 1)$ 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       LO = LO + Q_i(x[i]) · A
7       A = P_i(x[i])
8       HI = LO + A
9       while HI − LO < 1/2 ∧ ¬(LO < 1/4 ∧ HI ≥ 1/2)
10            if HI < 1/2
11               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                        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                        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 29.6. 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.

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				

29.6. ábra. Példa az intervallumnyújtásos aritmetikai kódolás működésére.

29.2.2. 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

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

29.7. ábra. A Krichevsky-Trofimov-becslés első értékeit tartalmazó táblázat.

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

$$P_e(a, b) = \frac{\frac{1}{2} \cdots (a - \frac{1}{2}) \frac{1}{2} \cdots (b - \frac{1}{2})}{1 \cdot 2 \cdots (a + b)} ,$$

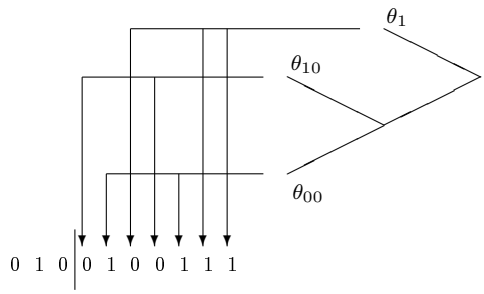
ahol kezdetben $a = 0$ és $b = 0$. Ezt mutatja a 29.7. ábrán látható táblázat, amelyben a ***Krichevsky-Trofimov-becslés*** $P_e(a, b)$ értékei szerepelnek kis (a, b) értékekre.

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 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.

29.1. példa. Legyen $\mathcal{S} = \{00, 10, 1\}$, továbbá $\theta_{00} = 1/2$, $\theta_{10} = 1/3$ és $\theta_1 = 1/5$. A



29.8. ábra. Példa egy fa forrásra.

megfelelő szuffix fát és egy sorozat feldolgozását szemlélteti a 29.8. ábra. A '0100111' sorozat aktuális valószínűsége a '...010' olvasása után $P^s(0100111|\ldots010) = (1-\theta_{10})\theta_{00}(1-\theta_1)(1-\theta_{10})\theta_{00}\theta_1\theta_1 = 2/3\cdot 1/2\cdot 4/5\cdot 2/3\cdot 1/2\cdot 1/5\cdot 1/5 = 4/1075$, 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.

29.2. 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

$$P_e^s(0100111|\ldots010) = \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{3}{4}\cdot\frac{1}{4}\cdot\frac{1}{2} = \frac{3}{8}\cdot\frac{3}{8}\cdot\frac{1}{16} = \frac{9}{1024}\,,$$

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:

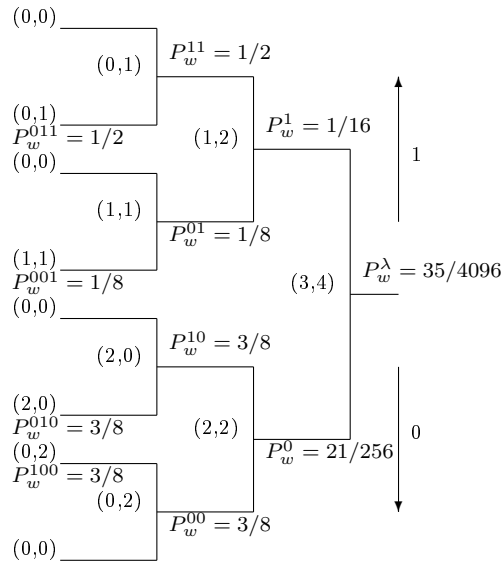
$$P_w^s = \begin{cases} \frac{P_e(a_s,b_s)+P_w^{0s}P_w^{1s}}{2}, & \text{ha } 0 \le L(s) < D \ , \\ P_e(a_s,b_s), & \text{ha } L(s) = D \ . \end{cases}$$

Itt $L(s)$ megadja a (bináris) s sorozat hosszát, és $P_e(a_s,b_s)$ a Krichevsky-Trofimov-becsléssel meghatározott becsült valószínűség.

29.3. példa. Ha kódoljuk a '0100111' sorozatot a '...010' elolvasása után a 29.9. 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 $x_{n-1} \dots x_{n-D+i} \quad (i = 1, \dots, D)$ 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.



29.9. 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 $P_w^s = P_e(0,0) = 1$.

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 $CT(x_1, \ldots, x_n | x_{-D+1}, \ldots x_0)$ környezetfát. Feltesszük, hogy az (x_{n-1}, x_n) rendezett pár jelöli a λ gyökeret.

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     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       s = (x_{n-1}, ..., x_{n-D+i})
9       if x_n = 0
10         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** $P_e(a_s, b_s) = P_e(a_s, b_s) \cdot (a_s + 1/2)(a_s + b_s + 1)$
13 $b_s \leftarrow b_s + 1$
14 $P_w^s = (1/2) \cdot (P_e(a_s, b_s) + P_w^{0s} \cdot P_w^{1s})$
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

29.2-1. 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 megfelelőHuffman-kódokkal.

29.2-2. Számítsuk ki az előző gyakorlatban meghatározott kódok esetén az egyes kódszavak egyedi redundanciáját és a kódok redundanciáját.

29.2-3. Határozzuk meg a Krichevsky-Trofimov-becslés alapján a 0100110 sorozat és összes részsorozatának $P_e(a, b)$ becsült valószínűségét.

29.2-4. 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 kódsz

29.2-5. 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.

29.2-6. Az előző gyakorlat eredményeit felhasználva módosítsuk a 01001101 sorozat 110 olvasása utáni becsült valószínűségét.

29.2-7. Lássuk be, hogy az $(x_1 \dots x_n)$ bináris sorozat összegzett valószínűsége

$$Q^n(x_1 x_2 \dots x_{n-1} x_n) = \sum_{j=1,\dots,n; x_j=1} P^j(x_1 x_2 \dots x_{j-1} 0) \ .$$

29.3. 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.

29.3.1. 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.

29.3.2. 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

29.3-1. Alkalmazzuk az LZ77 és LZ78 algoritmusokat az *abracadabra* szövegre.

29.3-2. Milyen típusú fájlokat lehet jól tömöríteni az LZ77, illetve az LZ78 algoritmussal? Milyen típusú fájlok esetén nem előnyös az LZ77, illetve az LZ78 tömörítés?

29.3-3. Elemezzük az első, illetve utolsó eltolás használatának előnyét az LZ77 kódolásban, ha több illeszkedő részsorozat található.

29.4. 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      M[0, j] = X[j]
3  for i = 0 to n − 1
4      for j = 0 to n − 1
5          M[i, j] = M[i − 1, j + 1 mod n]
6  for i = 0 to n − 1
7      M' i-edik sor = M i-edik sora lexikografikus sorrendben
8  for i = 0 to n − 1
9      L[i] = M'[i, n − 1]
10 i = 0
11 while M' i-edik sor ≠ X
12         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 az $X[0\,..\,n-1]$ tömb, munkaváltozója pedig az $E[0\,..\,n-1]$ tömb..

BWT-DEKÓDOLÓ(L, I)

```
1  E = L rendezése
2  pi[-1] = I
3  for i = 0 to n - 1
4       j = 0
5       while L[j]) ≠ E[π[i - 1]] ∨ j eleme π-nek
6            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.

od@Azonos jelből álló gyakori blokkok kihasználására Burrows és Wheeler az ***előre-mozgató 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 a $c[0\,..\,n-1]$ tömb, munkaváltozója pedig a $Q[..]$ 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 algorit-

musok 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

29.4-1. Alkalmazzuk a Burrows-Wheeler-transzformációt és az előre-mozgató kódolást az *abracadabra* szövegre.

29.4-2. űLássuk 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.

29.4-3. 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.

29.4-4. 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?

29.5. 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.

29.5.1. 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:

$$f = \begin{pmatrix} 139 & 144 & 149 & 153 & 155 & 155 & 155 & 155 \\ 144 & 151 & 153 & 156 & 159 & 156 & 156 & 155 \\ 150 & 155 & 160 & 163 & 158 & 156 & 156 & 156 \\ 159 & 161 & 162 & 160 & 160 & 159 & 159 & 159 \\ 159 & 160 & 161 & 161 & 160 & 155 & 155 & 155 \\ 161 & 161 & 161 & 161 & 160 & 157 & 157 & 157 \\ 162 & 162 & 161 & 163 & 162 & 157 & 157 & 157 \\ 161 & 162 & 161 & 161 & 163 & 158 & 158 & 158 \end{pmatrix} .$$

29.5.2. A diszkrét koszinusz transzformáció

Minden $(f(i,j))_{i,j=0,\dots,7}$ 8×8-as blokkot egy új $(F(u,v))_{u,v=0,\dots,7}$ 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 = w0 to 7
2      for v = 0 to 7
3          F(u,v) = 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.

29.5.3. 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).

29.5.4. 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 – 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      for i = 0 to l
6          if l = 1 mod 2
7              u = i
8              elseu = l − i
9          if T[u, l − u] = 0
10             futam = futam + 1
11             elsec = 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    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

29.5-1. Adjuk meg az 5, -19 és 32 egészek ábrázolásához használható méret és amplitúdó értékeket.

29.5-2. 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?

29.5-3. 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?

29.5-4. 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?

29.5-5. Mi lenne az előző gyakorlatban a cikkcakk kód (feltéve, hogy az előző DC tagtól vett eltérés -3)?

29.5-6. Bármely $(f(n))_{n=0,\ldots,m-1}$ sorozatra definiáljuk az $(\hat{f}(n))_{n=0,\ldots,2m-1}$ sorozatot a következő módon

$$\hat{f}(n) = \begin{cases} f(n), & \text{ha } n = 0, \ldots, m-1 \,, \\ f(2m-1-n), & \text{ha } n = m, \ldots, 2m-1 \,. \end{cases}$$

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

29-1 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.

29-2 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

$$\sum_{x \in \mathcal{X}} 2^{-L(x)} \leq 3/4 \; .$$

29-3 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 $1/2 \lg n + 3$.

29-4 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 könyve [365] tartalmazza. A Huffman-algoritmust Huffman tette közzé [186]. Az algoritmus megtalálható a [82] 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 [276] és Rissanen [305] fejlesztette ki. A megvalósítással kapcsolatos tudnivalókat tárgyalja [225, 304, 369]. A modellezésről szóló részben a Willems, Shtarkov és Tjalkens által alkalmazott megközelítést követtük [368]. A pontos számítások megtalálhatóak az eredeti cikkükben [367], amelyik az *IEEE Information Theory Society* legjobb cikk díját kapta 1996-ban.

A Lempel és Ziv által közzétett eredeti LZ77 és LZ78 [386, 387] 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 [38] cikkben vagy a [39] 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 [66] 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, 110, 218].

A veszteséges képtömörítéseknek csak az alapjait vázoltuk, kiemelve az adatok előkészítését, ami különböző módszerekhez, például a Huffman-kódoláshoz, szükséges. Részletes leírás található a [342] könyvben, ahol a JPEG2000 szabvány is megtalálható. A bemutatott példát [357] 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 [342] 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 ro-

molhat számottevően. Ilyenkor finomabb kvantálási módszereket alkalmaznak.

Az adattömörítő algoritmusokra vonatkozó lényeges ismeretek megtalálhatók az információelmélettel foglalkozó könyvekben, például [84, 168], mert az elérhető tömörítési arány elemzéséhez szükséges a forráskódolás elméletének ismerete. Nemrégiben több adattömörítéssel foglalkozó könyv is megjelent, például [39, 171, 271, 313, 320], amelyeket az Olvasó figyelmébe ajánlunk. A Calgary Corpus és Canterbury Corpus szintmérő fájlok elérhetők a [67] vagy [69] helyeken.

Információelmélettel (és adattömörítéssel) kapcsolatos magyar nyelvű szakkönyv Bartha Tamás és Pataricza András [280], Csiszár Imre és Körner János [88], Györfi László, Győri Sándor és Vajda István [163], valamint Linder Tamás és Lugosi Gábor [230] munkája.

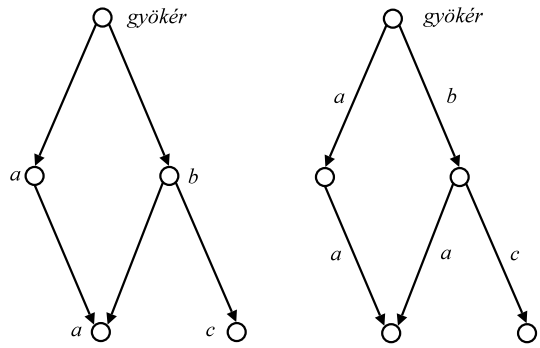
30. Félig strukturált adatbázisok

Az Internet elterjedése és az adatbázisok elméletének fejlődése kölcsönösen hatnak egymásra. Az Interneten megjelenő oldalak tartalmát sok esetben adatbázisrendszerek tárolják, másrészt az oldalak és a köztük kialakított kapcsolatok együttesen is tekinthetők egy olyan adatbázisnak, amelynek nincs a szokásos értelemben rögzített sémája. Az oldalak tartalmát és az oldalak közti kapcsolatokat maguk az oldalak írják le, ilyen értelemben csak félig strukturált adatokról beszélhetünk, melyeket legjobban irányított, címkézett gráfokkal lehet jellemezni. A félig strukturált adatok esetén az adatszerkezetek és a lekérdezések megadása során sokkal gyakrabban használunk rekurzív technikákat, mint a klasszikus relációs adatbázisok esetében. Ennek megfelelően kell az adatbázisok különböző problémaköreit, mint például a megszorításokat, függőségeket, lekérdezéseket, osztott tárolást, jogosultságokat, bizonytalanság kezelését, általánosítani. A félig strukturáltság új kérdések felvetését is jelenti. Mivel a klasszikus adatbázisoktól eltérően a lekérdezések nem minden esetben alkotnak zárt rendszert, azaz a lekérdezések egymás utáni alkalmazhatósága függ az eredmény típusától, ezért nagyobb jelentőséget kap a típusok ellenőrzésének problémaköre.

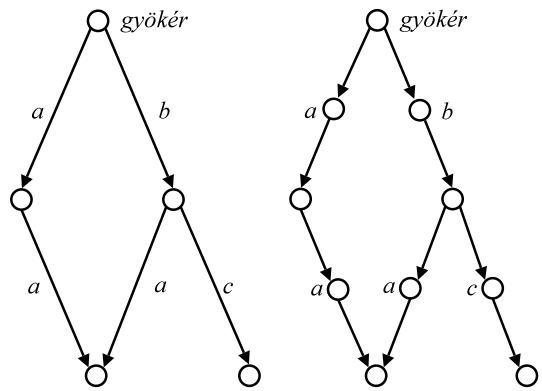
A relációs adatbázisok esetén az elméleti megalapozás szoros kapcsolatban áll a véges modellelmélettel. Félig strukturált esetben az automaták, azon belül is a faautomaták kapnak nagyobb hangsúlyt.

30.1. Félig strukturált adatok és az XML

A félig strukturált adatokon irányított, gyökérrel rendelkező címkézett gráfot értünk. A gyökér egy kitüntetett csúcs, amelybe nem mutat él. A gráf csúcsai azonosítóval megkülönböztetett objektumok. Az objektumok atomi vagy összetett típusúak. Az összetett típusú objektumot irányított élek kapcsolják egy vagy több objektumhoz. Az atomi típusú objektumokhoz társulnak az adatértékek. Kétféle modellt szokás használni, az egyikben a csúcsok címkézettek, a másikban az élek. Az utóbbi az általánosabb, mivel minden csúcscímkézett gráfnak egyértelműen megfeleltethető például az az élcímkézett gráf, amelyben minden él azt a címkét kapja, amilyen címkéje annak a csúcsnak van, ahová az él mutat. Ezzel olyan élcímkézett irányított



30.1. ábra. Csúcscímkézett gráfnak megfelelő élcímkézett gráf.

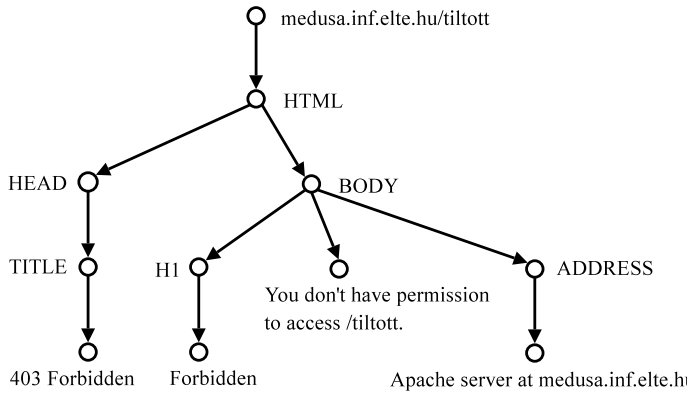


30.2. ábra. Élcímkézett gráf és a megfelelő csúcscímkézett gráf.

gráfot kapunk, amelyre teljesül, hogy az egy csúcsba mutató éleknek ugyanaz a címkéje. Ezzel a megfeleltetéssel az élcímkézett gráfokra vonatkozó fogalmak, definíciók, állítások speciális esetként átfogalmazhatók csúcscímkézett gráfokra.

Ha élcímkézett gráfból akarunk csúcscímkézett gráfot kapni, akkor a szokásos megfeleltetés a következő. Ha egy (u, v) él címkéje c, akkor töröljük ezt az élt, és vezessünk be egy új w csúcsot, amelynek címkéje legyen c, majd húzzuk be az (u, w) és (w, v) éleket. Így egy n csúcsból, és m élből álló élcímkézett gráfból $m + n$ csúcsból, és $2m$ élből álló csúcscímkézett gráfot kapunk. Ezáltal a csúcscímkézett gráfokra kimondott algoritmusok és költségbecslések átírhatók élcímkézett gráfokra.

Mivel a gyakorlatban használt leírások inkább csúcscímkézett gráfokat használnak, ezért a fejezetben végig csúcscímkézett gráfokról le Az ere-



30.3. ábra. A „tiltott” nevű XML fájlnak megfelelő gráf.

detileg dokumentumrendszerekhez kifejlesztett ***XML*** (e**X**tensible **M**arkup **L**anguage) nyelv egymásba ágyazott, rendezett, névvel ellátott elemek leírására szolgál, ily módon alkalmas a félig strukturált adatok közül a fák ábrázolására. Mivel a tágabban értelmezett XML nyelvben az elemek között hivatkozások is megadhatók, ezáltal tetszőleges félig strukturált adatok leírására is használható az XML nyelv.

Az alábbiakban a medusa.inf.elte.hu/tiltott címen található oldal leírását adtuk meg XML formában. A leírás szerkezeti jellemzői alapján, természetes módon kapható a 30.3. ábrán megadott csúcscímkézett fa.

```
<HTML>
        <HEAD>
                <TITLE>403 Forbidden</TITLE>
        </HEAD>
        <BODY>
                <H1>Forbidden</H1>
                You don't have permission to access /tiltott.
                <ADDRESS>Apache Server at medusa.inf.elte.hu </ADDRESS>
        </BODY>
</HTML>
```

Gyakorlatok

30.1-1. Adjunk meg egy csúcscímkézett gráfot, amely ennek a fejezetnek a szerkezetét és szövegformázásait reprezentálja.

30.1-2. Hány olyan különböző irányított, csúcscímkézett gráf adható meg, amelyben a csúcsok száma n, az élek száma m, a lehetséges címkék száma k? Ezek közül mennyi az izomorfia erejéig különböző? Milyen értékeket kapunk

$n = 5$, $m = 7$, $k = 2$ esetére?

30.1-3. Tekintsünk egy fát, amelyben bármely csúcs gyerekeit különböző sorszámokkal látjuk el. Bizonyítsuk be, hogy a fa csúcsai megcímkézhetők (a_v, b_v) párokkal, ahol a_v és b_v természetes számok, úgy, hogy teljesüljenek a következők:
a. $a_v < b_v$, minden v csúcsra.
b. Ha v-nek az u leszármazottja, akkor $a_v < a_u < b_u < b_v$.
c. Ha u és v testvérek, és $sorszám(u) < sorszám(v)$, akkor $b_u < a_v$.

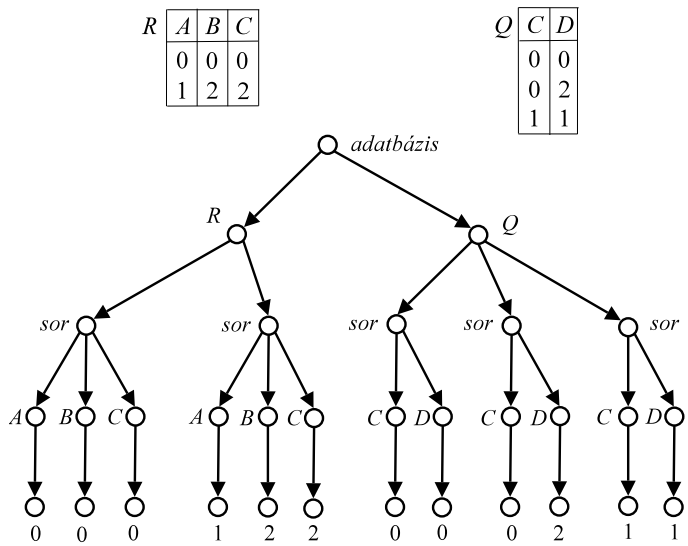
30.2. Sémák és szimulációk

A relációs adatbázisok esetében a sémák fontos szerepet töltenek be az adatok leírásában, lekérdezésében, a lekérdezések optimalizálásában, illetve a hatékonyságot növelő tárolási eljárásokban. A félig strukturált esetben a sémát a gráfból kell visszanyerni. A séma korlátozza a gráf útvonalaihoz tartozó címkesorozatokat.

A 30.4. ábrán az $R(A, B, C)$ és $Q(C, D)$ relációs sémával rendelkező relációkat látjuk, illetve a nekik megfelelő félig strukturált leírásukat. A fa leveleinek címkéi a relációs sorok adatértékei. A gyökérből az adatértékekhez vezető irányított útvonalak a következő különböző címkesorozatokat tartalmazzák: *adatbázis.R.sor.A, adatbázis.R.sor.B, adatbázis.R.sor.C, adatbázis.Q.sor.C, adatbázis.Q.sor.D*. Ezt tekinthetjük a félig strukturált adatbázis sémájának. Vegyük észre, hogy a séma is egy gráf, ahogy ez a 30.5. ábrán látható. A két gráf diszjunkt egyesítése is gráf, amelyen az alábbi módon értelmezhetünk egy szimulációs leképezést. Ezzel teremtünk kapcsolatot az eredeti gráf és a sémának megfelelő gráf között.

30.1. definíció. *Legyen $G = (V, E, A, címke())$ csúcscímkézett irányított gráf, ahol V a csúcsok, E az élek, A a címkék halmaza, és $címke(v)$ a v csúcs címkéjével egyenlő. Jelölje $E^{-1}(v) = \{u \mid (u, v) \in E\}$ a V csúcsba mutató élek kiinduló csúcsainak halmazát. Egy s bináris reláció ($s \subseteq V \times V$) **szimuláció**, ha $s(u, v)$ esetén a következő 2 feltétel teljesül:*
i) $címke(u) = címke(v)$ és
ii) minden $u' \in E^{-1}(u)$ esetén létezik olyan $v' \in E^{-1}(v)$, melyre $s(u', v')$
*Egy **v csúcs szimulálja az u csúcsot,** ha van olyan s szimuláció, amelyre $s(u, v)$. Az **u és v csúcs hasonló,** $u \approx v$, ha u szimulálja v-t és v szimulálja u-t.*

Könnyű látni, hogy az üres reláció szimuláció, szimulációk egyesítése is szimuláció, mindig létezik maximális szimuláció, és végül a hasonlóság ekvivalencia reláció. Nem jelent lényegi változtatást, ha a definícióban $E^{-1}(u)$



30.4. ábra. Egy relációs adatbázis megadása félig strukturált modellben.

helyett E-t írunk, mivel ez csak azt jelenti, hogy a gráfban az élek irányítását megfordítottuk.

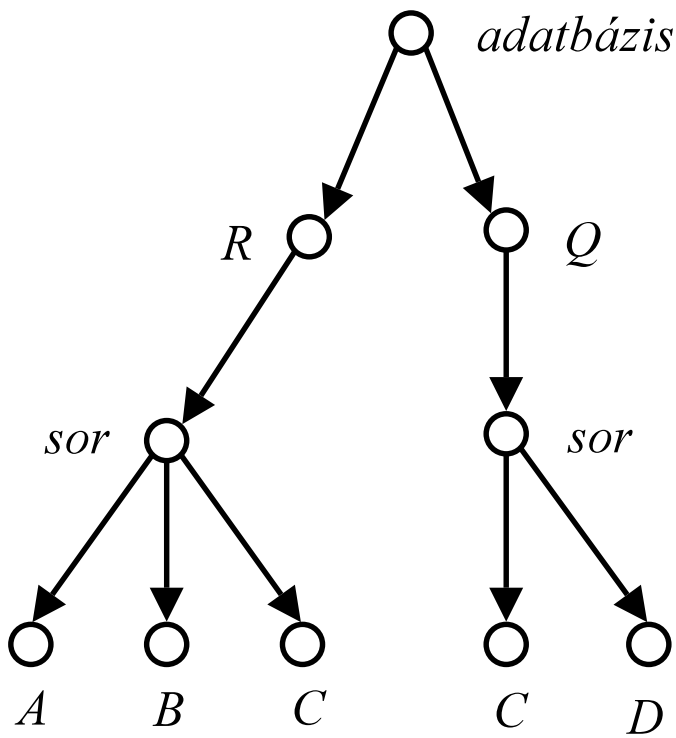
Azt mondjuk, hogy a D gráf szimulálja az S gráfot, ha létezik olyan $f : V_S \mapsto V_D$ leképezés, amelyre a $(v, f(v))$ reláció szimuláció a $V_S \times V_D$ halmazon.

Kétféle sémát szokás megkülönböztetni, alsó korlátot és felső korlátot. Ha a D adatgráf szimulálja az S sémagráfot, akkor azt mondjuk, hogy ***S alsó korlátja D-nek.*** Vegyük észre, hogy ez azt jelenti, hogy S-ben az irányított utakhoz tartozó összes címkesorozat D-ben is előfordul valamilyen út mentén. Ha S szimulálja D-t, akkor az ***S felső korlátja D-nek.*** Felső korlát esetén a D-ben előforduló címkesorozatok S-ben is előfordulnak.

Félig strukturált adatbázisok esetén fontos szerepet játszanak azok a sémák, amelyek maximális alsó korlátok, vagy minimális felső korlátok.

Az S és D gráfok közti éltartó leképezést morfizmusnak hívjuk. Vegyük észre, hogy az f akkor és csak akkor morfizmus, ha D szimulálja az S-et. Annak eldöntése, hogy létezik-e S-ről D-re morfizmus, NP-teljes probléma. Ezzel szemben az alábbiakban belátjuk, hogy a maximális szimuláció kiszámítása PTIME bonyolultságú.

Jelöljük $szim(v)$-vel azokat a csúcsokat, amelyek szimulálják az u-t. A maximális szimuláció kiszámítása ekvivalens az összes $szim(v)$ halmaz meghatározásával, ahol $v \in V$. Először a definíción alapuló naiv kiszámítást vesszük.



30.5. ábra. A 30.4. ábrán megadott félig strukturált adatbázis sémája.

MAXIMÁLIS-SZIMULÁCIÓ-NAIV-MÓDON(G)

```
1  for minden v ∈ V
2      szim(v) ← {u ∈ V | címke(u) = címke(v)}
3  while ∃ u, v, w ∈ V :  v ∈ E^{-1}(u) ∧ w ∈ szim(u) ∧ E^{-1}(w) ∩ szim(v) = ∅
4          szim(u) = szim(u) \ {w}
5  return {szim(u)|u ∈ V}
```

30.2. állítás. *Ha $m \geq nA$, akkor a* MAXIMÁLIS-SZIMULÁCIÓ-NAIV-MÓDON *algoritmus $O(m^2n^3)$ idő alatt kiszámolja a maximális szimulációt.*

Bizonyítás. Abból indulunk ki, hogy mik lehetnek a $szim(u)$ elemei. Ha egy (v, u) él alapján a $szim(u)$ w eleme a definíció szerint nem szimulálja az u-t, akkor kivesszük a w-t a $szim(u)$ halmazból. Ilyenkor azt mondjuk, hogy a (v, u) él szerint élesítettük a $szim(u)$ halmazt. Ha már egyik $szim(u)$ halmazt sem lehet egyik él szerint sem élesíteni, akkor a $szim(u)$ minden eleme szimulálja az u-t. Az állítás igazolásához már csak azt kell észrevenni, hogy a **while** ciklus legfeljebb n^2 iterációból állhat. ∎

Az algoritmus hatékonyságán speciális adatszerkezetek felhasználásával javíthatunk. Először vezessük be azt a $szim(u)$-nál bővebb *szim-jelölt*(u) halmazt, amelynek elemeiről meg akarjuk állapítani, hogy szimulálják-e az u-t.

MAXIMÁLIS-SZIMULÁCIÓ-JAVÍTVA(G)

```
1  for minden v ∈ V
2      szim-jelölt(u) = V
3      if E^{-1}(v) = ∅
4          szim(v) ← {u ∈ V|címke(u) = címke(v)}
5      > else szim(v) = {u ∈ V|címke(u) = címke(v) ∧ E^{-1}(u) ≠ ∅}
6  while ∃v ∈ V : szim(v) ≠ szim-jelölt(v)
7          törlésre-jelölt = E(szim-jelölt(v)) \ E(szim(v))
8          for minden u ∈ E(v)
9              szim(u) ← szim(u) \ törlésre-jelölt
10             szim-jelölt(v) = szim(v)
11 return {szim(u)|u ∈ V}
```

A javított algoritmus **while** ciklusa a következő invariáns tulajdonságokkal rendelkezik.

I_1: $\forall v \in V : szim(v) \subseteq$ *szim-jelölt*(v).

I_2: $\forall\ u, v, w \in V : (v \in E^{-1}(u) \ \wedge \ w \in szim(u)) \ \Rightarrow \ (E^{-1}(w) \cap$ *szim-jelölt*$(v) \neq \emptyset)$.

Amikor a $szim(u)$ halmazt egy (v, u) éllel élesítjük, akkor azt ellenőrizzük, hogy egy $w \in szim(u)$ elemnek van-e $szim(v)$-ben szülője. Az I_2 miatt $szim(v)$ helyett ezt elég a *szim-jelölt*(v) elemeire ellenőrizni, és ha egy $w' \in$ *szim-jelölt*$(v) \setminus szim(v)$ elemet egyszer már figyelembe vettünk, akkor véglegesen eltávolítjuk a *szim-jelölt*(v) halmazból.

Tovább javíthatjuk az algoritmust, ha a **while** ciklus iterációiban nem számoljuk ki újra a *törlésre-jelölt* halmazt, hanem dinamikusan tartjuk karban.

MAXIMÁLIS-SZIMULÁCIÓ-HATÉKONYAN(G)

```
 1 for minden v ∈ V
 2     szim-jelölt(v) ← V
 3     if E^{-1}(v) = ∅
 4        szim(v) = {u ∈ V | címke(u) = címke(v)}
 5        else szim(v) = {u ∈ V | címke(u) = címke(v) ∧ E^{-1}(u) ≠ ∅}
 6     törlésre-jelölt(v) = E(V) \ E(szim(v))
 7 while ∃v ∈ V : törlésre-jelölt(v) ≠ ∅
 8         for minden u ∈ E(v)
 9             for minden w ∈ törlésre-jelölt(v)
10                 if w ∈ szim(u)
11                    szim(u) = szim(u) \ {w}
12                    for minden w'' ∈ E(w)
13                        if E^{-1}(w'') ∩ szim(u) = ∅
14                           törlésre-jelölt(u) = törlésre-jelölt>(u) ∪ {w''}
15             szim-jelölt(v) = szim(v)
16             törlésre-jelölt(v) = ∅
17 return {szim(u) | u ∈ V}
```

A fenti algoritmus **while** ciklusa az alábbi invariánssal rendelkezik.
I_3: $\forall v \in V$: *törlésre-jelölt*$(v) = E($*szim-jelölt*$(v)) \setminus E(szim(v))$.

Az algoritmus megvalósításához használjunk egy *számláló* $n \times n$-es tömböt. A *számláló*$[w'', u]$ értéke legyen egyenlő az $|E^{-1}(w'') \cap szim(u)|$ nem negatív egész számmal. A számláló kezdeti értékeinek beállítása $O(mn)$ időben történik. Amikor w elemet eltávolítjuk a $szim(u)$ halmazból, akkor a w összes w'' gyerekére csökkenteni kell a *számláló*$[w'', u]$ értéket. Ezzel elérjük, hogy a legbelső **if** feltételt konstans időben tudjuk ellenőrizni. Az algoritmus elején a $szim(v)$ halmazok kezdeti értékeinek beállítása $O(n^2)$ időben történik, feltéve, hogy $m \geq n$. A *törlésre-jelölt*(v) halmazok beál-

lítása összesen $O(mn)$ időt igényel. Tetszőleges v és w csúcsok esetén, ha $w \in$ *törlésre-jelölt*(v) igaz a **while** ciklus i-edik iterációjában, akkor minden j-edik iterációban ugyanez már hamis lesz, feltéve, hogy $j > i$. Ugyanis $w \in$ *törlésre-jelölt*(v)-ből következik, hogy $w \notin E(szim(v))$, és $j > i$ esetén a *szim-jelölt*(v) értéke a j-edik iterációban részhalmaza a $szim(v)$ i-edik iterációban vett értékének, valamint tudjuk, hogy teljesül az I_3 invariáns. Emiatt annak ellenőrzése, hogy $w \in szim(u)$, végrehajtható $\sum_v \sum_w |E(v)| = O(mn)$ idő alatt. A $w \in szim(u)$ ellenőrzés minden w és u csúcsra legfeljebb egyszer ad igaz értéket, mivel ha egyszer teljesül a feltétel, akkor a w-t eltávolítjuk a $szim(u)$ halmazból, és többé már nem kerül oda vissza. Ebből következik, hogy a **while** ciklus külső **if** feltételének kiszámítása $\sum_v \sum_w (1 + |E(v)|) = O(mn)$ időben történik.

Ezzel beláttuk a következő állítást.

30.3. állítás. *Ha $m \geq n$, akkor a* Maximális-szimuláció-hatékonyan *algoritmus $O(mn)$ idő alatt kiszámolja a maximális szimulációt.*

Ha egy szimuláció inverze is szimuláció, akkor biszimulációról beszélünk. Az üres reláció biszimuláció, és mindig létezik maximális biszimuláció. A maximális biszimuláció hatékonyabban számolható ki, mint a szimuláció. A maximális biszimuláció a PT-algoritmussal $O(m \lg n)$ időben határozható meg. Élcímkézett gráfok esetén ugyanerre $O(m \lg(m+n))$ költség adódik.

Mint ahogy azt látni fogjuk, a biszimulációk fontos szerepet játszanak a félig strukturált adatbázisok indexelésében, mivel egy gráf biszimuláció szerinti hányados gráfja pontosan azokat a címkesorozatokat tartalmazza, mint a gráf. Megjegyezzük, hogy a gyakorlatban szimuláció helyett még úgynevezett ***DTD*** leírásokat is szokás sémának tekinteni. A DTD reguláris nyelven megfogalmazott adattípus-definíciókat tartalmaz.

Gyakorlatok

30.2-1. Mutassunk példát arra, hogy a szimulációból nem következik a biszimuláció.

30.2-2. Irányított, nem feltétlen körmentes, csúcscímkézett G gráfra értelmezzünk egy *fásít*(G) műveletet a következőképpen. A művelet eredménye egy nem feltétlen véges G' gráf, melynek csúcsai a G gyökérből induló irányított útjai, az utak címkéi a hozzájuk tartozó címkesorozatok. A p_1 csúcsból húzzunk egy élt p_2-be, ha p_2 végpontját elhagyva p_1-et kapjuk. Bizonyítsuk be, hogy G és *fásít*(G) hasonlók a biszimulációra nézve.

30.3. Lekérdezések és indexek

A félig strukturált adatbázisban tárolt információt lekérdezések segítségével lehet kinyerni. Ehhez először rögzítjük a feltehető kérdések formáját, vagyis megadunk egy ***lekérdező nyelvet,*** majd definiáljuk a kérdések értelmét, vagyis a ***lekérdezés kiértékelését*** egy félig strukturált adatbázisra vonatkozóan. A hatékony kiértékeléshez általában indexeket használunk. Az ***indexelés*** lényege, hogy az adatbázisban tárolt adatokat valamilyen hasonlósági elven összevonjuk, azaz egy olyan indexet állítunk elő, amely tükrözi az eredeti adatok struktúráját. Az eredeti lekérdezést az indexben hajtjuk végre, majd az eredmény alapján megkeressük az indexértékeknek megfelelő adatokat az eredeti adatbázisban. Az index mérete általában jóval kisebb az eredeti adatbázisnál, ezáltal a lekérdezések gyorsabb végrehajtását lehet elérni. Megjegyezzük, hogy a klasszikus adatbázisok esetén használatos invertált lista típusú indexet integrálni lehet a következőkben bevezetett sématípusú indexekkel. Ez különösen akkor előnyös, mikor XML dokumentumokban kulcsszavak alapján keresünk.

Elsőként a reguláris kifejezéseket tartalmazó lekérdező nyelvvel, és az ehhez használatos indextípusokkal ismerkedünk meg.

30.4. definíció. *Legyen adva a* $G = (V, E, gyökér, \Sigma, címke, azon, érték)$ *irányított, csúcs-címkézett gráf, ahol* V *a csúcsok,* $E \subseteq V \times V$ *az élek,* Σ *a címkék halmaza,* Σ *tartalmaz két speciális címkét, a GYÖKÉR és ÉRTÉK címkéket. A* $címke(v)$ *a* v *csúcs címkéje. Az* $azon(v)$ *a* v *csúcs azonosítója, a gyökér egy olyan csúcs, melynek címkéje GYÖKÉR, és amelyből minden csúcs elérhető irányított úton keresztül. Ha* v *levél, azaz nem vezet ki él belőle, akkor a címkéje ÉRTÉK, és* $érték(v)$ *a* v *levélhez tartozó adatérték. Úton mindig irányított utat értünk, azaz olyan* $n_0, \ldots, n_p$ *csúcsokból álló sorozatot, amelyben* n_i *csúcsból vezet él* n_{i+1}*-be, ha* $0 \leq i \leq p-1$*. Egy címkékből álló* $l_0, \ldots, l_p$ *sorozatot* ***címkesorozatnak*** *vagy másképpen* ***egyszerű kifejezésnek*** *hívunk. Az* $\mathbf{n_0, \ldots, n_p}$ ***út illeszkedik az*** $\mathbf{l_0, \ldots, l_p}$ ***címkesorozatra,*** *ha* $címke(n_i) = l_i$ *, minden* $0 \leq i \leq p$ *esetén.*

A reguláris kifejezéseket rekurzív módon definiáljuk.

30.5. definíció. *Legyen* $R ::= \varepsilon \mid \Sigma \mid \_ \mid R.R \mid R|R \mid (R) \mid R? \mid R^*$*, ahol* ***R reguláris kifejezés****, továbbá* ε *az üres kifejezés,* $\_$ *tetszőleges címkét jelöl,* $.$ *az egymás után következést,* $|$ *a logikai vagy műveletet,* $?$ *az opcionális választást,* $*$ *a véges ismétlést jelenti.* ***L(R)*** *jelölje az* R *által meghatározott címkesorozatokból álló reguláris nyelvet. Egy* n ***csúcs illeszkedik egy címkesorozatra,*** *ha van olyan gyökérből induló és* n *csúccsal végződő út, amely illeszkedik a címkesorozatra. Egy* n ***csúcs illeszkedik az*** R ***reguláris kifejezésre****, ha*

van olyan címkesorozat az $L(R)$ nyelvben, amelyre az n csúcs illeszkedik. Az R reguláris kifejezés által meghatározott ***lekérdezés eredménye egy G gráfon*** *azoknak a csúcsoknak az $R(G)$ halmaza, amelyek illeszkednek az R kifejezésre.*

Mivel a reguláris kifejezések kiértékelésénél mindig gyökérből induló utakat keresünk, ezért a címkesorozatok első eleme mindig a GYÖKÉR, amit a rövidség kedvéért elhagyhatunk.

Megjegyezzük, hogy a reguláris kifejezéshez tartozó $L(R)$ nyelvek halmaza metszetre zárt, és az $L(R) = \emptyset$ probléma eldönthető probléma.

A lekérdezések eredménye kiszámolható az R reguláris kifejezésnek megfelelő nem determinisztikus A_R automata segítségével. Az algoritmus rekurzív megadása a következő.

NAIV-KIÉRTÉKELÉS(G, A_R)

```
1 Bejárt = ∅                          == Ha s állapotban jártunk az u csúcsban,
                                         akkor (u, s) bekerül a Bejárt halmazba.
2 BEJÁR (gyökér(G), kezdőállapot(A_R))
```

BEJÁR(u, s)

```
1 if (u, s) ∈ Bejárt
2    return eredmény[u, s]
3 Bejárt = Bejárt ∪ {(u, s)}
4 eredmény[u, s] = ∅
5 for minden s --ε--> s' // Ha s állapotból ε jelet olvasva s' állapotba kerülünk.
6     if s' ∈ végállapot(A_R)
7        eredmény[u, s] = {u} ∪ eredmény[u, s]
8     eredmény[u, s] = eredmény[u, s] ∪ BEJÁR(u, s')
9 for minden s --címke(u)--> s'
            // Ha s állapotból címke(u) jelet olvasva s' állapotba kerülünk.
10    if s' ∈ végállapot(A_R)
11       eredmény[u, s] = {u} ∪ eredmény[u, s]
12    for minden v, ahol (u, v) ∈ E(G)
                          // Az u gyerekeire rekurzívan folytatjuk a bejárást.
13       eredmény[u, s] = eredmény[u, s] ∪ BEJÁR(v, s')
14 return eredmény[u, s]
```

30.6. állítás. *Adott R reguláris lekérdezés és G gráf esetén az $R(G)$ kiszámítási költsége a G élei számának és az R-nek megfelelő véges nem determinisztikus automata állapotai számának polinomja.*

Bizonyítás. A bizonyítás vázlatosan a következő. Legyen az R-hez tartozó nem determinisztikus véges automata A_R. Jelölje $|A_R|$ az A_R állapotainak számát. Tekintsük az m élből álló G gráfnak a BEJÁR algoritmus szerinti szélességi bejárását a gyökérből kiindulva. A bejárás közben a csúcsból kiolvasott címke alapján jutunk az automata új állapotába, amelyet minden csúcshoz eltárolunk. Ha az automata elfogadó végállapotba került, akkor a csúcs megoldás. A bejárás során időnként vissza kell lépnünk egy él mentén, hogy arra menjünk tovább, amerre még nem jártunk. Belátható, hogy a bejárás során minden állapotban legfeljebb egyszer kell egy élen áthaladni, vagyis az automata legfeljebb ennyi lépést hajt végre. Ez összesen $O(|A_R|m)$ lépés. Ezek alapján következik az állítás. ■

A G gráfban két csúcs ***nem különböztethető meg reguláris kifejezéssel,*** ha nincs olyan reguláris R, amelynek eredményében az egyik csúcs benne van, a másik pedig nincs benne. Nyilvánvaló, hogy ha két csúcs nem különböztethető meg, akkor a címkéjük megegyezik. Osztályozzuk a csúcsokat. Egy osztályba kerüljenek az azonos címkéjű csúcsok. Ezzel a csúcsok halmazának egy P partícióját állítottuk elő. Ez lesz az ***alappartíció.*** Az is könnyen végiggondolható, hogy ha két csúcs nem különböztethető meg, akkor ez a tulajdonság a szülőkre is öröklődik. Ebből következik, hogy a gyökérből a megkülönböztethetetlen csúcsokhoz vezető utakhoz tartozó címkesorozatok halmaza is megegyezik. Legyen minden n csúcsra $L(n) = \{l_0, \ldots, l_p | n$ illeszkedik az $l_0, \ldots, l_p$ címkesorozatra$\}$. Az n_1 és n_2 csúcsok akkor és csak akkor nem különböztethetők meg, ha $L(n_1) = L(n_2)$. Ha azok a csúcsok kerülnek egy osztályba, amelyek $L(n)$ értéke megegyezik, akkor a P partíciónak egy P' finomítását kapjuk. Erre a partícióra az is teljesül, hogy ha egy n csúcs szerepel egy R reguláris lekérdezés eredményében, akkor az n csúccsal egy osztályba tartozó minden csúcs is benne lesz a lekérdezés eredményében.

30.7. definíció. *Legyen adva egy $G = (V, E, gyökér, \Sigma, címke, azon, érték)$ gráf és legyen P a V egy olyan partíciója, amely az alappartíció finomítása, azaz az egy osztályba tartozó csúcsok címkéje megegyezik. Ekkor az $I(G) = (P, E', gyökér', \Sigma, címke', azon', érték')$ gráfot **indexnek** hívjuk. Az indexgráf csúcsai a P partíció osztályai, továbbá $(I, J) \in E'$ akkor és csak akkor, ha van olyan $i \in I$ és $j \in J$, melyekre $(i, j) \in E$. Ha $I \in P$, akkor $azon'(I)$ az I indexcsúcs azonosítója, és $címke'(I) = címke(n)$, ahol $n \in I$. A $gyökér'$ a P partíciónak az az osztálya, amely a G gyökerét tartalmazza. Ha $címke(I) = ÉRTÉK$, akkor $érték'(I) = \{érték(n) \mid n \in I\}$.*

Ha adott a V halmaznak egy P partíciója, akkor $n \in V$ csúcs esetén ***osztály(n)*** jelölje a P partíciónak azt az osztályát, amelybe n tartozik. Indexek esetén az $I(n)$ jelölést is használhatjuk az $osztály(n)$ helyett.

Vegyük észre, hogy az indexek lényegében a csúcsok különböző partícióival azonosíthatók, így ha nem okoz félreértést, akkor a partíciókat is indexeknek hívjuk. Azok lesznek a jó indexek, amelyek kis méretűek, és a lekérdezések eredménye a gráfon és az indexen megegyezik. Az indexeket gyakran úgy adjuk meg, hogy a csúcsokon definiálunk egy ekvivalenciarelációt, és az indexnek megfelelő partíció az ekvivalencia osztályokból áll.

30.8. definíció. *Legyen P az a partíció, amelynek bármelyik I osztályára igaz, hogy $n, m \in I$ akkor és csak akkor, ha $L(n) = L(m)$. Ekkor a P alapján készített $I(G)$ indexet* ***naiv indexnek*** *hívjuk.*

Naiv index esetén a P partícióban szereplő I osztály minden n eleméhez ugyanaz az $L(n)$ nyelv tartozik, melyet $L(I)$-vel fogunk jelölni.

30.9. állítás. *Legyen I a naiv index egy csúcsa és R egy reguláris kifejezés. Ekkor $I \cap R(G) = \emptyset$ vagy $I \subseteq R(G)$.*

Bizonyítás. Legyen $n \in I \cap R(G)$ és legyen $m \in I$. Ekkor van olyan $l_0, \ldots, l_p$ címkesorozat az $L(R)$-ben, amelyre n illeszkedik, azaz $l_0, \ldots, l_p \in L(n)$. Mivel $L(n) = L(m)$ így m is illeszkedik erre a címkesorozatra, azaz $m \in I \cap R(G)$. ■

NAIV-INDEXES-KIÉRTÉKELÉS(G, R)

```
1 legyen I_G a G naiv indexe
2 Q ← ∅
3 for minden I ∈ NAIV-KIÉRTÉKELÉS(I_G, A_R)
4       do Q ← Q ∪ I
5 return Q
```

30.10. állítás. *A* NAIV-INDEXES-KIÉRTÉKELÉS *által előállított Q megegyezik $R(G)$-vel.*

Bizonyítás. Az előző állítás miatt egy I osztálynak vagy minden eleme beletartozik a lekérdezés eredményébe vagy egyik sem tartozik hozzá. ■

A naiv index használatával tehát a lekérdezést ki tudjuk értékelni, de a következő állítás szerint nem elég hatékonyan. Az állítást Stockmeyer és Meyer bizonyította be 1973-ban.

30.11. állítás. *A* NAIV-INDEXES-KIÉRTÉKELÉS *algoritmushoz szükséges I_G naiv index előállítása PSPACE-teljes probléma.*

A másik probléma a naiv index használatával, hogy az $L(I)$ halmazok különböző I esetén nem feltétlen diszjunktak, így ez a tárolásban redundanciát jelenthet.

A fentiek miatt a naiv index partíciójának olyan finomítását keressük, amelyet már hatékonyan tudunk előállítani, és amely segítségével $R(G)$-t továbbra is elő tudjuk állítani.

30.12. definíció. *Az $I(G)$ index **biztonságos**, ha bármilyen $n \in V$ és $l_0, \ldots, l_p$ címkesorozat esetén, melyekre teljesül, hogy n illeszkedik az $l_0, \ldots, l_p$ címkesorozatra a G gráfban, igaz, hogy osztály(n) illeszkedik az $l_0, \ldots, l_p$ címkesorozatra az $I(G)$ gráfban. Az $I(G)$ index **pontos**, ha az index bármely I osztálya és $l_0, \ldots, l_p$ címkesorozat esetén, melyekre teljesül, hogy I illeszkedik az $l_0, \ldots, l_p$ címkesorozatra az $I(G)$ gráfban, igaz, hogy tetszőleges $n \in I$ csúcs illeszkedik az $l_0, \ldots, l_p$ címkesorozatra a G gráfban.*

A biztonságosság azt jelenti, hogy az index alapján kiértékelt eredményben szerepelő csúcsok tartalmazzák a reguláris lekérdezés eredményét, vagyis $R(G) \subseteq R(I(G))$, a pontosság ennek a fordítottja, azaz az index alapján történő kiértékelés nem ad hamis eredményt, azaz $R(I(G)) \subseteq R(G)$. A biztonságosság és az index éleinek definícióiból rögtön következik az alábbi állítás.

30.13. állítás. *1. Minden index biztonságos.*
2. A naiv index biztonságos és pontos.

Ha I a G csúcsainak egy halmaza, akkor az $L(I)$ nyelvet, vagyis azt a nyelvet, amelybe tartozó címkesorozatokhoz az I elemei illeszkednek, eddig a G gráf alapján értelmeztük. Ha ezt jelölni akarjuk, akkor az $L(I, G)$ jelölést használjuk. $L(I)$-t értelmezhetjük az $I(G)$ gráfban is, amelynek I egy csúcsa. Ekkor $L(I)$ helyett az $L(I, I(G))$ jelölést alkalmazzuk, mely az összes olyan címkesorozatot jelenti, amelyre az I csúcs illeszkedik az $I(G)$ gráfban. A biztonságos és pontos indexek esetén $L(I, G) = L(I, I(G))$, és ilyenkor egyszerűen $L(I)$-t írhatunk. Ebben az esetben az $L(I)$ kiszámítását az $I(G)$ alapján végezzük, mivel az $I(G)$ mérete általában kisebb a G méreténél.

Tetszőleges típusú indexgráfot lekérdezhetünk a NAIV-KIÉRTÉKELÉS algoritmussal. Ezután egyesítjük a talált indexcsúcsokat. Ha pontos indexet használtunk, akkor ugyanazt kapjuk, mintha az eredeti gráfot kérdeztük volna le.

INDEXES-KIÉRTÉKELÉS$(G, I(G), A_R)$

```
1 legyen I(G) a G indexe
2 Q = ∅
3 for minden I ∈ NAIV-KIÉRTÉKELÉS(I(G), A_R)
4       Q = Q ∪ I
5 return Q
```

Először egy hatékonyan előállítható, biztonságos és pontos indexet fogunk megadni, amely a csúcsokon értelmezett hasonlóságon alapul. Így kapjuk az 1-indexet. Ennek nagy méretét csökkenthetjük, ha csak lokális hasonlóságot követelünk meg. Az így kapott A(k)-index esetén elveszítjük a pontosságot, vagyis az INDEXES-KIÉRTÉKELÉS alapján olyan eredményt is kaphatunk, amely nem szerepel az R reguláris lekérdezésre adott válaszban, ezért az eredményeket még tesztelnünk kell, hogy megőrizzük a pontosságot.

30.14. definíció. *Legyen $\approx$ egy olyan ekvivalencia reláció a V halmazon, amelyre teljesül, hogy $u \approx v$ esetén*
i) címke(u) = címke(v),
ii) ha az u' csúcsból vezet él az u csúcsba, akkor van olyan v' csúcs, amelyből vezet él a v csúcsba és $u' \approx v'$.
iii) ha a v' csúcsból vezet él az v csúcsba, akkor van olyan u' csúcs, amelyből vezet él az u csúcsba és $u' \approx v'$.
*A fenti ekvivalencia relációt **biszimulációnak** nevezzük. Egy gráf u és v csúcsa **biszimuláns** akkor és csak akkor, ha létezik olyan $\approx$ biszimuláció, amelyre $u \approx v$.*

30.15. definíció. *Legyen P az a partíció, amely egy biszimuláció ekvivalencia osztályaiból áll. A P partíció alapján definiált indexet **1-indexnek** nevezzük.*

30.16. állítás. *Az 1-index a naiv index finomítása. Ha a G gráfban az egy csúcsba mutató élek kezdőpontjai különböző címkékkel rendelkeznek, azaz $x \neq x'$ és $(x, y), (x', y) \in E$ esetén címke$(x) \neq$címke(x'), akkor $L(u) = L(v)$ akkor és csak akkor, ha u és v biszimuláns.*

Bizonyítás. $u \approx v$ esetén *címke*(u) = *címke*(v). Az u csúcs illeszkedjen egy $l_0, \ldots, l_p$ címkesorozatra, és legyen u' az l_{p-1} címkéhez tartozó csúcs. Ekkor van olyan v', hogy $u' \approx v'$, és $(u', u), (v', v) \in E$. Az u' illeszkedik az $l_0, \ldots, l_{p-1}$ címkesorozatra, és indukció miatt ekkor v' is illeszkedik az $l_0, \ldots, l_{p-1}$ címkesorozatra, vagyis v illeszkedik az $l_0, \ldots, l_p$ címkesorozatra, tehát ha két csúcs ugyanabban az osztályban van az 1-index szerint, akkor

ugyanabban az osztályban van a naiv index szerint is.

Az állítás második részéhez elég belátni, hogy a naiv index biszimulációnak felel meg. Tartozzon u és v egy osztályba a naiv index szerint. Ekkor *címke*(u) = *címke*(v). Ha $(u', u) \in E$, akkor van olyan $l_0, \ldots, l_p$ címkesorozat, amelyhez tartozó két utolsó csúcs u' és u. Ekkor a szülőkre feltett különböző címkék miatt $L(u) = L' \cup L''$ diszjunkt felbontás, ahol $L' = \{l_0, \ldots, l_p | u'$ illeszkedik az $l_0, \ldots, l_{p-1}$ sorozatra, és $l_p =$ *címke*$(u)\}$, valamint $L'' = L(u) \setminus L'$. Mivel $L(u) = L(v)$, ezért van olyan v', amelyre $(v', v) \in E$ és *címke*(u') = *címke*(v'). A szülők különböző címkéi miatt $L' = \{l_0, \ldots, l_p \mid v'$ illeszkedik az $l_0, \ldots, l_{p-1}$ sorozatra, és $l_p =$ *címke*$(v)\}$, vagyis $L(u') = L(v')$ és indukció miatt $u' \approx v'$, tehát $u \approx v$. ∎

30.17. állítás. *Az 1-index biztonságos és pontos.*

Bizonyítás. Ha x_p illeszkedik az $l_0, \ldots, l_p$ címkesorozatra a G gráfban az $x_0, \ldots, x_p$ csúcsok miatt, akkor az indexgráf definíciója miatt *osztály*(x_i)-ből vezet él *osztály*(x_{i+1})-be, $0 \leq i \leq p-1$, vagyis *osztály*(x_p) illeszkedik az $l_0, \ldots, l_p$ címkesorozatra az $I(G)$ gráfban. A pontossághoz tegyük fel, hogy I_p illeszkedik az $l_0, \ldots, l_p$ címkesorozatra az $I(G)$ gráfban az $I_0, \ldots, I_p$ miatt. Ekkor van olyan $u' \in I_{p-1}$, $u \in I_p$, melyekre $(u', u) \in E$. Legyen $v \in I_p$ tetszőleges csúcs, ekkor van olyan v', melyre $u' \approx v'$, és $(v', v) \in E$, vagyis $v' \in I_{p-1}$. Indukciót alkalmazva következik, hogy v' illeszkedik az $l_0, \ldots, l_{p-1}$ címkesorozatra az $x_0, \ldots, x_{p-2}, v'$ csúcsok miatt, de akkor v illeszkedik az $l_0, \ldots, l_p$ címkesorozatra az $x_0, \ldots, x_{p-2}, v', v$ csúcsok miatt a G gráfban. ∎

Ha azt a biszimulációt tekintjük, amely esetén minden csúcs különböző partícióba kerül, akkor az ehhez az 1-indexhez tartozó $I(G)$ gráf megegyezik a G gráffal, vagyis legrosszabb esetben az $I(G)$ mérete megegyezik a G méretével, és az $I(G)$ I csúcsaihoz még tárolnunk kell az I elemeit, amely az összes I-re összesen a G csúcsainak tárolását jelenti. A lekérdezések gyorsabb kiértékelése miatt minimális méretű 1-indexet, vagyis a legdurvább 1-indexet keressük. Ellenőrizhető, hogy x és y akkor és csak akkor tartoznak ugyanabba az osztályba a legdurvább 1-index szerint, ha x és y biszimuláns.

1-INDEXES-KIÉRTÉKELÉS(G, R)

1 legyen I_1 a G legdurvább 1-indexe
2 **return** INDEXES-KIÉRTÉKELÉS(G, I_1, A_R)

Az algoritmus első lépésében a legdurvább 1-indexet kell megadni. Ezt a feladatot visszavezethetjük a legdurvább stabil partíció megtalálására, amelylyel a következő alfejezetben fogunk foglalkozni. Így az ehhez használható PT-algoritmus hatékony változatával a legdurvább 1-index $O(m \lg n)$ műveleti

költséggel, $O(m+n)$ tárköltséggel található meg, ahol n a G csúcsainak, m az éleinek száma.

Az I_1 gráf biztonságossága és pontossága miatt elég a lekérdezést az I_1 gráfban kiértékelni, vagyis megkeresni azokat az indexcsúcsokat, amelyek illeszkednek az R reguláris kifejezésre. Ennek költsége a 30.6. állítás alapján az I_1 gráf méretének polinomja.

Az I_1 méretét a következő paraméterek segítségével becsülhetjük. Legyen p különböző címke a G gráfban és legyen ***k a G gráf átmérője,*** vagyis a leghosszabb irányított út hossza. (Az irányított útban nem szerepelhet kétszer ugyanaz a csúcs.) Ha a gráf fa, akkor az átmérő a fa mélységével egyenlő. Gyakran készítünk olyan weboldalakat, amelyek egy d mélységű fát alkotnak, majd minden oldalhoz hozzáadunk egy q elemből álló navigációs sort, vagyis a gráf q darab kiválasztott oldalával összekötünk minden csúcsot. Belátható, hogy ekoor a kapott gráf átmérője legfeljebb $d+q(d-1)$. Gyakorlatban a gráf méretéhez képest a d és q általában nagyon kicsi. A következő állítás bizonyítása Milo és Suciu cikkében található meg.

30.18. állítás. *Legyen G gráfban legfeljebb p különböző címke. Legyen G átmérője kisebb, mint k. Ekkor tetszőleges biszimuláció által definiált I_1 1-index mérete felülről becsülhető a k és p függvényében, függetlenül a G méretétől.*

Gyakorlatok

30.3-1. Mutassuk meg a maximális szimuláció szerinti indexről, hogy finomításra nézve az 1-index és a naiv index közé esik. Adjunk példát arra, hogy mindkét tartalmazás valódi tartalmazás.

30.3-2. Jelölje $I_s(G)$ a maximális szimuláció szerinti indexet. Igaz-e, hogy $I_s(I_s(G)) = I_s(G)$?

30.3-3. Reprezentáljuk a G gráfot, és az R reguláris lekérdezésnek megfelelő automata állapotátmeneti gráfját relációs adatbázisban. Adjunk meg egy algoritmust relációs lekérdező nyelven, például PL/SQL-ben, amely $R(G)$-t számolja ki.

30.4. Stabil partíciók és a PT-algoritmus

A félig strukturált adatbázisok lekérdezéseinek hatékony kiértékeléséhez felhasznált indexstruktúrák többsége egy gráf csúcshalmazának valamilyen particionálásán alapul. Az indexek készítését gyakran vissza lehet vezetni a legdurvább stabil partíció előállítására.

30.19. definíció. *Legyen E bináris reláció egy véges V halmazon, azaz*

$E \subseteq V \times V$. Ekkor V a ***csúcsok****, E az irányított* ***élek*** *halmaza. Legyen tetszőleges $S \subseteq V$ esetén $E(S) = \{y \mid \exists x \in S, (x,y) \in E\}$ és $E^{-1}(S) = \{x \mid \exists\, y \in S, (x,y) \in E\}$. Tetszőleges $S \subseteq V$ és $B \subseteq V$ esetén azt mondjuk, hogy* ***B stabil az S-re nézve,*** *ha vagy $B \subseteq E^{-1}(S)$ vagy $B \cap E^{-1}(S) = \emptyset$. Legyen P a V egy partíciója, azaz V felbontása diszjunkt halmazokra, más néven blokkokra. Azt mondjuk, hogy* ***P stabil S-re nézve****, ha P minden blokkja stabil S-re nézve.* ***P stabil a P' partícióra nézve,*** *ha P minden blokkja stabil a P' minden blokkjára nézve. Ha P stabil minden saját blokkjára nézve, akkor a P partíció* ***stabil****. Legyen P és Q a V két partíciója.* ***Q finomítása P-nek,*** *vagy másképpen* ***P durvább mint Q,*** *ha Q minden blokkja előáll a P valahány blokkjának egyesítéseként.* ***A legdurvább stabil partíció problémája*** *azt jelenti, hogy adott V, E és P esetén keressük a P legdurvább stabil finomítását, azaz olyan stabil finomítását P-nek, amely durvább P minden más stabil finomításánál.*

Megjegyezzük, hogy szokás úgy is definiálni a stabilitást, hogy B stabil az S-re nézve, ha vagy $B \subseteq E(S)$ vagy $B \cap E(S) = \emptyset$. Ez nem jelent lényeges különbséget, csak az élek irányításának megfordítását, vagyis E bináris reláció helyett az inverz E^{-1} bináris relációra nézve mondjuk ki a stabilitást, ahol $(x,y) \in E^{-1}$ akkor és csak akkor, ha $(y,x) \in E$. Ugyanis $(E^{-1})^{-1}(S) = \{x \mid \exists\, y \in S, (x,y) \in E^{-1})\} = \{x \mid \exists\, y \in S, (y,x) \in E\} = E(S)$.

Legyen $|V| = n$ és $|E| = m$. Be fogjuk látni, hogy a legdurvább stabil partíció problémájára mindig létezik egyértelmű megoldás, és megadható olyan algoritmus, amely a megoldást $O(m \lg n)$ futási időben találja meg, $O(m+n)$ tárigény mellett. Ezt az algoritmust R. Paige és R. E. Tarjan tette közzé 1987-ben, ezért ezt ***PT-algoritmusnak fogjuk*** nevezni.

Az algoritmus lényege, hogy ha egy blokk nem stabil, akkor kettéhasítjuk úgy, hogy a részek már stabilak lesznek. Először egy naiv módszert adunk meg. A hasítási művelet tulajdonságai alapján ennek hatékonyságát növelhetjük azáltal, hogy az eljárást mindig a kisebbik félre folytatjuk.

30.20. definíció. *Legyen E bináris reláció V-n, $S \subseteq V$ és Q a V egy partíciója. Legyen* ***hasít****(S,Q) a Q-nak az a finomítása, amelyet úgy kapunk, hogy Q minden olyan B blokkját kettévágjuk, amelybe $E^{-1}(S)$ valódi módon belemetsz, azaz ha $B \cap E^{-1}(S) \neq \emptyset$ és $B \setminus E^{-1}(S) \neq \emptyset$, akkor B helyett vegyük a partícióba a $B \cap E^{-1}(S)$ és $B \setminus E^{-1}(S)$ két blokkot. Azt mondjuk, hogy S a Q* ***hasítója****, ha hasít$(S,Q) \neq Q$.*

Vegyük észre, hogy Q nem stabil S-re akkor és csak akkor, ha S a Q hasítója.

A stabilitás és a hasítás a következő tulajdonságokkal rendelkezik, amelyek bizonyítását az Olvasóra bízzuk.

30.21. állítás. *Legyen S és T a V két részhalmaza, P és Q a V két partíciója. Ekkor teljesülnek a következők.*
1. A stabilitás öröklődik a finomítás során, azaz ha Q finomítása P-nek, és P stabil S-re nézve, akkor Q is stabil S-re nézve.
2. A stabilitás öröklődik az egyesítés során, azaz ha P stabil S-re és T-re nézve, akkor P stabil $S \cup T$-re nézve.
3. A hasít művelet monoton a második argumentumban, azaz ha P finomítása Q-nak, akkor hasít(S, P) finomítása hasít(S, Q)-nak.
4. A hasít művelet kommutatív a következő értelemben. Tetszőleges S, T és P esetén teljesül, hogy hasít$(S$,hasít$(T, P)) =$ hasít$(T$,hasít$(S, P))$, és P legdurvább olyan partíciója, amely egyaránt stabil S-re és T-re nézve is megegyezik hasít$(S$,hasít$(T, P))$-vel.

A naiv algoritmus során a P partícióból kiindulva finomítjuk a Q partíciót, amíg Q minden blokkjára nézve stabil nem lesz. A finomítási lépés során Q olyan S hasítóját keressük, amely Q valahány blokkjának egyesítéseként állítható elő. Megjegyezzük, hogy elég lenne Q blokkjai közül hasítót találni, de ez a kicsit általánosabb választás lehetőséget teremt a későbbiekben az algoritmus hatékonyabbá tételére.

NAIV-PT(V, E, P)

```
1 Q = P
2 while Q nem stabil
3       legyen S a Q olyan hasítója,
        amely Q valahány blokkjának egyesítése
4       Q = hasít(S, Q)
5 return Q
```

Vegyük észre, hogy az algoritmus végrehajtása során nem használhatjuk fel kétszer ugyanazt az S halmazt, mivel finomításkor a stabilitás öröklődik, és a 4. lépésben kapott, finomított partíció S-re nézve stabil. A felhasznált S halmazok egyesítését sem lehet később újra felhasználni, mivel az egyesítés is örökli a stabilitást. Ugyanakkor az is nyilvánvaló, hogy egy stabil partíció stabil bármely olyan S-re nézve, amely a partíció valahány blokkjának egyesítése. Ezek alapján könnyen igazolhatók a következő állítások.

30.22. állítás. *A* NAIV-PT *algoritmus bármely lépésében a P legdurvább stabil finomítása a Q-ban tárolt aktuális partíciónak finomítása.*

Bizonyítás. A bizonyítás indukcióval történik arra nézve, hogy hányszor hajtjuk végre a ciklusmagot. A $Q = P$ eset triviális. Tegyük fel, hogy egy

S hasító alkalmazása előtt Q-ra igaz az állítás. Legyen R a P legdurvább stabil finomítása. Mivel S a Q blokkjaiból áll, és indukció miatt R finomítása Q-nak, ezért S az R valahány blokkjának egyesítése. Az R stabil minden blokkjára és tetszőleges számú blokkjának egyesítésére nézve is, így R stabil S-re nézve, azaz $R = \mathit{hasít}(S, R)$. Másrészt a hasítás monotonitását kihasználva $\mathit{hasít}(S, R)$ finomítása $\mathit{hasít}(S, Q)$-nak, vagyis a Q aktuális értékének. ∎

30.23. állítás. *A* Naiv-PT *algoritmus a P egyértelműen létező legdurvább stabil finomítását adja meg, miközben a ciklust legfeljebb $(n-1)$-szer hajtja végre.*

Bizonyítás. Nyilvánvaló, hogy Q blokkjainak száma legalább 1 és legfeljebb n. A hasítás alkalmazásával Q-nak legalább egy blokkját kettévágjuk, azaz Q blokkjainak száma határozottan nő. Ebből következik, hogy az algoritmus legfeljebb $(n-1)$-szer hajtja végre a ciklust. Megálláskor Q stabil finomítása P-nek, és az előző állítás miatt a P legdurvább stabil finomítása Q-nak finomítása, ami csak úgy lehet, hogy Q megegyezik a P legdurvább stabil finomításával. ∎

30.24. állítás. *Ha a V minden x elemére tároljuk az $E^{-1}(\{x\})$ halmazt, akkor a* Naiv-PT *algoritmus költsége legfeljebb $O(mn)$.*

Bizonyítás. Az algoritmus megvalósításához az általánosság megszorítása nélkül korlátozhatjuk magunkat arra az esetre, amikor a gráfban nincsenek nyelők, azaz minden csúcsból indul ki él. Vagyis a V tetszőleges x elemére teljesül, hogy $1 \leq |E(\{x\})|$. Legyen adott ugyanis egy P partíció. P minden B blokkját vágjuk ketté. A B' halmazba tartozzanak B azon csúcsai, amelyekből legalább egy él indul ki. Ekkor $B' = B \cap E^{-1}(V)$. Legyen $B'' = B \setminus E^{-1}(V)$, azaz a B-ben található nyelők halmaza. A B'' halmaz tetszőleges S-re nézve stabil, hiszen $B'' \cap E^{-1}(S) = \emptyset$, így B''-t az algoritmus során sosem kell széthasítani. Így P helyett elég a B' blokkokból álló P'-t venni, amely a $V' = E^{-1}(V)$ halmaz partíciója. Nyilvánvaló, hogy P' legdurvább stabil finomításához hozzávéve a B'' blokkokat, P legdurvább stabil finomítását kapjuk. Ez azt jelenti, hogy az algoritmust megelőzi egy előkészítés, melyben P'-t állítjuk elő, majd egy utófeldolgozás, amelyben a kapott legdurvább stabil finomításhoz hozzávesszük a B'' blokkokat. Az előkészítés és utófeldolgozás költségét a következőképpen becsülhetjük. V' elemszáma nyilván legfeljebb m. Ha a V minden x elemére rendelkezésünkre áll $E^{-1}(\{x\})$, akkor az előkészítés és utófeldolgozás $O(m+n)$ költséggel jár.

Ezentúl tehát feltesszük, hogy V tetszőleges x elemére teljesül, hogy $1 \leq |E(\{x\})|$, amiből az is következik, hogy $n \leq m$. Mivel tároljuk az $E^{-1}(\{x\})$ halmazokat, ezért a Q partíció blokkjai közül $O(m)$ költséggel találhatunk egy

hasító blokkot. Az előző állítással együtt ez azt jelenti, hogy $O(mn)$ költséggel megvalósítható az algoritmus. ■

Az algoritmust hatékonyabban is végrehajthatjuk, ha ügyesebben keresünk hasító halmazokat. A javított algoritmus alapötlete, hogy a P partíción kívül két másik partíciót is kezelünk, a Q mellett egy olyan X partíciót is, amelyre minden lépésben igaz, hogy a Q az X finomítása, és Q stabil az X minden blokkjára nézve. Kiindulásként legyen $Q = P$ és X az a partíció, amely egyetlen blokkot tartalmaz, a V halmazt. Az algoritmus finomító lépését addig ismételjük, amíg végül $Q = X$ teljesül.

PT(V, E, P)

1 $Q = P$
2 $X = \{V\}$
3 **while** $X \neq Q$
4 legyen S az X-nek egy olyan blokkja, amely Q-nak nem blokkja, és legyen S-ben B a Q-nak egy olyan blokkja, amelyre $|B| \leq |S|/2$
5 $X = (X \setminus \{S\}) \cup \{B, S \setminus B\}$
6 $Q = \mathit{hasít}(S \setminus B, \mathit{hasít}(B, Q))$
7 **return** Q

30.25. állítás. *A* PT*-algoritmus és a* Naiv-PT *algoritmus eredménye megegyezik.*

Bizonyítás. Kezdetben teljesül, hogy Q stabil finomítása P-nek az X blokkjaira nézve. Az 5. lépésben mindig az X egyik blokkját vágjuk ketté, így az X finomítását kapjuk. A 6. lépésben a Q hasításokkal történő finomításával elérjük, hogy a Q stabil legyen az X két új blokkjára nézve. A stabilitás tulajdonságaira vonatkozó 30.21. állítás, illetve a Naiv-PT algoritmus helyessége alapján következik, hogy a PT-algoritmus is a P egyértelműen létező legdurvább stabil finomítását határozza meg. ■

Bizonyos esetekben a 6. lépés két hasításából az egyik elhagyható. Ehhez elégséges, hogy az E az x változó függvénye legyen.

30.26. állítás. *Ha V minden x elemére $|E(\{x\})| = 1$, akkor a* PT*-algoritmus 6. lépése a $Q = \mathit{hasít}(B, Q)$ lépésre cserélhető.*

Bizonyítás. Tegyük fel, hogy Q stabil egy olyan S halmazra nézve, amely Q valahány blokkjának egyesítése. Legyen B a Q-nak egy olyan blokkja, amely S-nek részhalmaza. Elég belátni, hogy $\mathit{hasít}(B, Q)$ stabil az $(S \setminus B)$-re nézve. Ehhez legyen B_1 a $\mathit{hasít}(B, Q)$ egy blokkja. Mivel a B szerinti hasítás eredménye B-re nézve stabil partíció, így vagy az teljesül, hogy $B_1 \subseteq E^{-1}(B)$ vagy

az, hogy $B_1 \subseteq E^{-1}(S) \setminus E^{-1}(B)$. Kihasználva az $|E(\{x\})| = 1$ feltételt, első esetben $B_1 \cap E^{-1}(S \setminus B) = \emptyset$, második esetben $B_1 \subseteq E^{-1}(S \setminus B)$ következik, ami pont azt jelenti, hogy $(S \setminus B)$-re stabil partíciót kaptunk. ■

Megjegyezzük, hogy általában abból, hogy egy partíció stabil S-re és B-re nézve, nem lehet arra következtetni, hogy $(S \setminus B)$-re nézve is stabil a partíció. Ha ez teljesül, akkor a méretek felezése miatt az algoritmus végrehajtási költségét javítani tudjuk, mert csak B szerinti hasításokra van szükség.

A 6. lépés két hasítása általános esetben egy blokkot elvileg négy részre is felbonthat. A következő állítás szerint a második hasítás az első hasítással kettévágott blokk egyik felét már nem változtatja meg, azaz a két hasítás maximum három részre bontást eredményezhet. Ezt kihasználva általános esetben is növelhető az algoritmus hatékonysága.

30.27. állítás. *Legyen Q stabil partíció az S-re nézve, ahol S a Q valahány blokkjának egyesítése, és legyen B a Q-nak olyan blokkja, amely S-nek részhalmaza. Legyen D a Q-nak olyan blokkja, amelyet hasít(B, Q) valódi módon felbont D_1 és D_2 részekre, úgy hogy ezek egyike sem üres halmaz. Tegyük fel, hogy a D_1 blokkot a hasít$(S \setminus B,$ hasít$(B, Q))$ valódi módon tovább bontja D_{11} és D_{12} nem üres halmazokra. Ekkor a következők teljesülnek.*
1. $D_1 = D \cap E^{-1}(B)$ és $D_2 = D \setminus D_1$ akkor és csak akkor, ha $D \cap E^{-1}(B) \neq \emptyset$ és $D \setminus E^{-1}(B) \neq \emptyset$.
2. $D_{11} = D_1 \cap E^{-1}(S \setminus B)$ és $D_{12} = D_1 \setminus D_{11}$ akkor és csak akkor, ha $D_1 \cap E^{-1}(S \setminus B) \neq \emptyset$ és $D_1 \setminus E^{-1}(S \setminus B) \neq \emptyset$.
3. A D_2 blokkot a hasít$(S \setminus B,$ hasít$(B, Q))$ nem bontja tovább.
4. $D_{12} = D_1 \cap (E^{-1}(B) \setminus E^{-1}(S \setminus B))$.

Bizonyítás. Az első két állítás a hasítás definíciójából következik. A harmadik állítás bizonyításához tegyük fel, hogy D_2 valódi felbontással keletkezett D-ből. Ekkor $D \cap E^{-1}(B) \neq \emptyset$, továbbá $B \subseteq S$ miatt $D \cap E^{-1}(S) \neq \emptyset$ is teljesül. A Q partíció minden blokkja, így D is stabil az S-re nézve, amiből az következik, hogy $D \subseteq E^{-1}(S)$. Mivel $D_2 \subseteq D$, ezért az első állítás miatt $D_2 \subseteq E^{-1}(S) \setminus E^{-1}(B) = E^{-1}(S \setminus B)$, vagyis D_2 stabil az $S \setminus B$ halmazra nézve, így az $S \setminus B$ szerinti hasítás nem vágja ketté a D_2 blokkot. Végül a negyedik állítás abból következik, hogy $D_1 \subseteq E^{-1}(B)$, és $D_{12} = D_1 \setminus E^{-1}(S \setminus B)$. ■

Jelölje *számláló*(x, S) azt a számot, ahány S-beli csúcsot x-ből el lehet érni, azaz legyen *számláló*$(x, S) = |S \cap E(\{x\})|$. Vegyük észre, hogy ha $B \subseteq S$, akkor $E^{-1}(B) \setminus E^{-1}(S \setminus B) = \{x \in E^{-1}(B) \mid$ *számláló*$(x, B) =$ *számláló*$(x, S)\}$.

A méretek feleződése miatt a V halmaz tetszőleges x eleme legfeljebb $\lg n + 1$ olyan különböző B halmazban szerepelhet, amelyet a PT-algoritmus közben finomításra használtunk. A következőkben meg fogjuk

adni a PT-algoritmusnak egy olyan végrehajtását, amelyben egy adott B blokk szerinti finomítás meghatározása az algoritmus 5. és 6. lépésében $O(|B| + \sum_{y \in B} |E^{-1}(\{y\})|)$ költséggel jár. Ha ezt összegezzük az algoritmus során felhasznált összes B blokkra és az ilyen blokkok összes elemére, akkor azt kapjuk, hogy a HATÉKONY-PT bonyolultsága legfeljebb $O(m \lg n)$. Ahhoz, hogy megadjunk egy ilyen megvalósítást, az adatok ábrázolásához ügyesen kell kiválasztanunk az adatszerkezeteket.

- Az E halmaz minden (x, y) éléhez láncoljuk hozzá az x csúcsot, valamint minden y csúcshoz láncoljuk hozzá az $\{(x, y)|(x, y) \in E\}$ listát. Ezáltal az $E^{-1}(\{y\})$ halmaz végigolvasásának költsége az $E^{-1}(\{y\})$ méretével lesz arányos.

- Legyen a Q partíció az X partíció finomítása. Mindkét partíció minden egyes blokkját egy-egy rekorddal ábrázoljuk. Az X partíció S blokkját ***egyszerűnek*** nevezzük, ha a Q egyetlen blokkjából áll, egyébként pedig ***összetettnek*** hívjuk.

- Legyen C az X partícióban szereplő összetett blokkok listája. Kezdetben legyen $C = \{V\}$, mivel V a P blokkjainak egyesítése. Ha P csak egy blokkból áll, akkor P megegyezik a saját legdurvább stabil finomításával, így nincs szükség további számításokra.

- Az X partíció bármely S blokkja esetén legyen *Q-blokkok*(S) a Q partíció azon blokkjainak duplán láncolt listája, amelyek egyesítése az S halmaz. Ezenkívül az $E^{-1}(S)$ halmaz minden x eleméhez tároljuk a *számláló*(x, S) értéket, amelyre egy mutató mutat minden olyan (x, y) élből, amelyben y az S halmaznak eleme. Kezdetben minden x csúcshoz a *számláló*$(x, V) = |E(\{x\})|$ érték tartozik, és minden (x, y) élhez készítsünk egy olyan mutatót, amely a *számláló*(x, V) értékre mutat.

- A Q partíció bármely B blokkja esetén legyen *X-blokk*(B) az X partíciónak az a blokkja, amelyben B szerepel. Legyen továbbá *méret*(B) a B számossága, valamint *elemei*(B) a B elemeinek duplán láncolt listája. Minden elemhez tartozzon egy mutató, amely a Q azon blokkjára mutat, amelyben ez az elem szerepel. A duplán láncolás segítségével egy elemet $O(1)$ időben tudunk törölni.

A 30.24. állítás bizonyítása alapján az általánosság korlátozása nélkül feltehetjük, hogy $n \leq m$. Ekkor belátható, hogy a fenti adatszerkezetek elkészítéséhez szükséges tárméret $O(m)$.

HATÉKONY-PT(V, E, P)

```
 1 if |P| = 1
 2    return P
 3 Q = P
 4 X = {V}
 5 C = {V}                          ▷ C az X összetett blokkjainak listája.
 6 while C ≠ ∅
 7       legyen S a C egy eleme
 8       legyen B az S első két eleme közül a kisebbik méretű
 9       C = C \ {S}
10       X = (X \ {S}) ∪ {{B}, S \ {B}}
11       S = S \ {B}
12       if |S| > 1
13          C = C ∪ {S}
14          Az E halmaz azon (x, y) éleit végigolvasva, melyekre y a B eleme,
            állítsuk elő az E^{-1}(B) halmazt és ennek minden x elemére
            számoljuk ki a számláló(x, B) értéket.
15          Az E^{-1}(B) halmazt végigolvasva a Q minden D blokkjához ha-
            tározzuk meg a D_1 = D ∩ E^{-1}(B) és D_2 = D \ D_1 halmazokat.
16          Az E halmaz azon (x, y) éleit végigolvasva, melyekre y a B
            eleme, állítsuk elő az E^{-1}(B) \ E^{-1}(S \ B) halmazt a
            számláló(x, B) = számláló(x, S) feltétel ellenőrzésével.
17          Az E^{-1}(B) \ E^{-1}(S \ B) halmazt végigolvasva
            Q minden D blokkjához határozzuk meg a D_12 =
            D_1 ∩ (E^{-1}(B) \ E^{-1}(S \ B)) és D_11 = D_1 \ D_12 halmazokat.
18          for Q minden olyan D blokkjára, melyre
    D_11 ≠ ∅, D_12 ≠ ∅ és D_2 ≠ ∅
            D_11 ≠ ∅, D_12 ≠ ∅ és D_2 ≠ ∅
19             if D az X egyszerű blokkja
20                C = C ∪ {D}
21                Q = (Q \ {D}) ∪ {D_11, D_12, D_2}
22          Az E halmaz azon (x, y) éleit végigolvasva, melyekre y a B
            eleme, számoljuk ki a számláló(x, S) értéket.
23 return Q
```

30.28. állítás. *A* HATÉKONY-PT *algoritmus a P legdurvább stabil finomítását határozza meg. Az algoritmus műveleti költsége legfeljebb $O(m \lg n)$, tárköltsége legfeljebb $O(m+n)$.*

Bizonyítás. Az algoritmus helyessége a PT-algoritmus helyességéből, valamint a 30.27. állításból következik. A felhasznált adatszerkezetek miatt a ciklus magját alkotó lépéssorozat műveleti költsége arányos a megvizsgált élek számával és a finomításhoz használt B blokk elemszámával, ami összesen $O(|B| + \sum_{y \in B} |E^{-1}(\{y\})|)$. Összegezzük ezt az algoritmus során finomításra felhasznált összes B blokkra és az ilyen blokkok összes elemére. Mivel a B mérete legfeljebb az S méretének fele, így a V halmaz tetszőleges x eleme legfeljebb $\lg n + 1$ különböző B halmazban szerepelhet. Ebből következik, hogy az algoritmus teljes műveleti költsége $O(m \lg n)$. Könnyen igazolható, hogy $O(m + n)$ méretű tár elég az algoritmushoz felhasznált adatszerkezetek tárolására, valamint arra, hogy az adatszerkezeteket az algoritmus futása alatt karbantartsuk. ■

Megjegyezzük, hogy egyes lépések összevonásával tovább lehetne javítani az algoritmust, de ez csak egy konstans tényezővel csökkentené a műveleti költséget.

Legyen $G^{-1} = (V, E^{-1})$ az a gráf, amit G-ből úgy kapunk, hogy minden él irányítását megfordítjuk. Vegyünk egy 1-indexet, amelyet egy $\approx$ biszimuláció határoz meg a G gráfban. Legyen I, J a biszimuláció két osztálya, azaz az $I(G)$ két csúcsa. A biszimuláció definíciójából következik, hogy $J \subseteq E(I)$ vagy $E(I) \cap J = \emptyset$. Mivel $E(I) = (E^{-1})^{-1}(I)$, ez azt jelenti, hogy J stabil az I-re nézve a G^{-1} gráfban. Tehát a G legdurvább 1-indexe a legdurvább stabil finomítása a G^{-1} gráf alappartíciójának.

30.29. következmény. *A legdurvább 1-index meghatározható a* Hatékony-PT *algoritmussal. Az algoritmus műveleti költsége legfeljebb* $O(m \lg n)$*, tárköltsége legfeljebb* $O(m + n)$*.*

Gyakorlatok

30.4-1. Bizonyítsuk be a 29.21. állítást.

30.4-2. A P partíció méret-stabil egy S halmazra nézve, ha a P minden B blokkjának tetszőleges x, y eleme esetén $|E(\{x\}) \cap S| = |E(\{y\}) \cap S|$. Méret-stabil egy partíció, ha az összes blokkjára nézve méret-stabil. Bizonyítsuk be, hogy tetszőleges partíció legdurvább méret-stabil finomítása kiszámítható $O(m \lg n)$ időben.

30.4-3. Az 1-index minimális, ha semelyik két egyforma címkéjű I, J csúcs nem vonható össze, mert létezik olyan K csúcs, amelyre nézve $I \cup J$ nem stabil. Mutassunk példát arra, hogy a minimális 1-index nem egyértelmű, és így különbözik a legdurvább 1-indextől.

30.4-4. Bizonyítsuk be, hogy aciklikus gráf esetén a minimális 1-index egyértelmű és megegyezik a legdurvább 1-indexszel.

30.5. A(k)-indexek

1-indexek esetén az egy osztályba tartozó csúcsok ugyanazokra a gyökérből induló címkesorozatokra illeszkednek. Ez azt jelenti, hogy az egy osztályba tartozó csúcsok az őseik alapján nem különböztethetők meg. Enyhítsük ezt a feltételt úgy, hogy csak lokális megkülönböztethetetlenséget kívánunk meg, azaz a legfeljebb k-adik szintig felmenő ősök alapján nem lehet megkülönböztetni az egy osztályba tartozó csúcsokat. Ezzel az 1-indexnél durvább, kevesebb osztályból álló indexet kapunk, vagyis az index mérete csökken, ami csökkenti a lekérdezések kiértékelésének költségét. Az 1-index biztonságos és pontos volt, amit szeretnénk megőrizni, mert ez garantálja, hogy az index alapján kiértékelt lekérdezések eredménye ugyanaz, mintha az eredeti gráfon értékeltük volna ki a lekérdezést. Az A(k)-index esetén a biztonságosság érvényben marad, de a pontosság nem teljesül, ezért ezt a kiértékelő algoritmus módosításával kell elérnünk.

30.30. definíció. *A $\approx^k$ **k-biszimuláció** a gráf V csúcsain az alábbi rekurzív definícióval meghatározott ekvivalencia reláció.*
i) $u \approx^0 v$ akkor és csak akkor, ha címke(u) = címke(v),
ii) $u \approx^k v$ akkor és csak akkor, ha $u \approx^{k-1} v$ és ha egy u' csúcsból vezet él az u csúcsba, akkor van olyan v' csúcs, amelyből vezet él a v csúcsba és $u' \approx^{k-1} v'$, illetve ez fordítva is igaz, azaz ha egy v' csúcsból vezet él az v csúcsba, akkor van olyan u' csúcs, amelyből vezet él a u csúcsba és $u' \approx^{k-1} v'$.
*$u \approx^k v$ esetén azt mondjuk, hogy u és v **k-biszimuláns**. Az **A(k)-indexhez** tartozó partíció osztályait a k-biszimuláció ekvivalenciaosztályai alkotják.*

Az elnevezésben az „A” az „approximatív” (közelítő) szóra utal.

Vegyük észre, hogy a $k = 0$ értékhez az alappartíció tartozik, és k értékének növelésével ezt finomítjuk, míg el nem jutunk a legdurvább 1-indexig.

Jelöljük $L(u, k, G)$-vel azokat a legfeljebb k hosszú címkesorozatokat, amelyekre u illeszkedik a G gráfban. Könnyen ellenőrizhetők az A(k)-index alábbi tulajdonságai.

30.31. állítás.
1. Ha u és v k-biszimuláns, akkor $L(u, k, G) = L(v, k, G)$.
2. Ha I az $A(k)$-index egy csúcsa és $u \in I$, akkor $L(I, k, I(G)) = L(u, k, G)$.
3. Az $A(k)$-index pontos a legfeljebb k hosszú egyszerű kifejezések esetén.
4. Az $A(k)$-index biztonságos.
5. A $(k+1)$-biszimuláció a k-biszimuláció (nem feltétlen valódi) finomítása.

Az A(k)-index a csúcsok k távolságú, gyökér felöli félkörnyezeteit hasonlítja össze, így az ezen kívül eső módosítások nem befolyásolják a csúcsok

ekvivalenciáját, ahogy ezt az alábbi, könnyen belátható állítás mutatja.

30.32. állítás. *Tegyük fel, hogy a v csúcsból az x-be és y-ba vezető legrövidebb utak egyaránt k-nál több élből állnak. Ekkor egy u csúcsból v-be vezető él hozzáadása vagy törlése nem változtatja meg, hogy x és y k-biszimuláns vagy nem.*

Az A(k)-index készítéséhez a PT-algoritmus módosított változatát használjuk. Általánosan vizsgálhatjuk a legdurvább stabil finomítás közelítési problémáját.

30.33. definíció. *Legyen G=(V,E) irányított gráfban P a V partíciója. Legyen $P_0, P_1, \ldots, P_k$ partíciókból álló olyan sorozat, melyre $P_0 = P$, és P_{i+1} a P_i legdurvább olyan finomítása, amely stabil P_i-re nézve. Azt mondjuk, hogy a P_k partíció a* ***P legdurvább stabil finomításának k lépéses közelítése.***

Vegyük észre, hogy a P_i sorozat bármelyik tagja a P finomítása, és ha $P_k = P_{k-1}$, akkor P_k a P legdurvább stabil finomítása. Könnyen ellenőrizhető, hogy a PT-algoritmushoz hasonlóan a P legdurvább stabil finomításának tetszőleges közelítését mohó módon számolhatjuk ki, azaz, ha P_i valamelyik B blokkja nem stabil a P_{i-1} valamelyik S blokkjára nézve, akkor S-sel hasítsuk szét a B blokkot, azaz P_i helyett vegyük a *hasít*(S, P_i) partíciót.

NAIV-KÖZELÍTÉS(V, E, P, k)

```
1  P_0 = P
2  for i = 1 to k
3      P_i = P_{i-1}
4      for minden S ∈ P_{i-1}, amelyre hasít(S, P_i) ≠ P_i
5          P_i = hasít(S, P_i)
6  return P_k
```

Megjegyezzük, hogy a NAIV-KÖZELÍTÉS algoritmust is lehetne javítani a PT-algoritmushoz hasonlóan.

Az A(k)-index kiszámításához használhatjuk a NAIV-KÖZELÍTÉS algoritmust, csak azt kell észrevenni, hogy definíció alapján az A(k)-indexhez tartozó partíció a G^{-1} gráfban stabil az A($k-1$)-indexhez tartozó partícióra nézve. Belátható, hogy az így előállított A(k)-index készítésének műveleti költsége $O(km)$, ahol m a G gráf éleinek száma.

A(k)-INDEXES-KIÉRTÉKELÉS(G, A_R, k)

```
1 legyen I_k a G A(k)-indexe
2 Q = INDEXES-KIÉRTÉKELÉS(G, I_k, A_R)
3 for minden u ∈ Q
4     if L(u) ∩ L(A_R) = ∅
5     Q = Q \ {u}
6 return Q
```

Az A(k)-index biztonságos, de csak a legfeljebb k hosszú egyszerű kifejezésekre nézve pontos, ezért az 4. lépésben a Q halmaz minden u elemére ellenőrizni kell, hogy tényleg kielégíti az R lekérdezést, és elhagyjuk az eredményből azokat, amelyek nem illeszkednek az R lekérdezésre. Azt, hogy egy adott csúcs kielégíti-e az R kifejezést, egy véges nem determinisztikus automatával lehet eldönteni a 30.6. állításban leírtakhoz hasonlóan, annyi változtatással, hogy az automatát most fordított irányban kell működtetni. Az ilyen ellenőrzések számát csökkenteni lehet a következő állítás alapján, melynek bizonyítását az Olvasóra bízzuk.

30.34. állítás. *Tegyük fel, hogy az $A(k)$-indexnek megfelelő I_k gráfban az I indexcsúcs illeszkedik egy olyan címkesorozatra, amelynek a végződése $s = l_0, \ldots, l_p$, $p \leq k-1$. Ha a G gráfban minden gyökérből induló, s'.s alakú címkesorozat kielégíti az R kifejezést, akkor az I minden eleme kielégíti az R kifejezést.*

Gyakorlatok

30.5-1. Jelölje $A_k(G)$ a G A(k)-indexét. Igaz-e, hogy $A_k(A_l(G)) = A_{k+l}(G)$?

30.5-2. Bizonyítsuk be a 30.31. állítást.

30.5-3. Bizonyítsuk be a 30.32. állítást.

30.5-4. Bizonyítsuk be a 30.34. állítást.

30.5-5. Igazoljuk, hogy a NAIV-KÖZELÍTÉS algoritmus valóban a legdurvább k lépéses stabil közelítést állítja elő.

30.5-6. Legyenek az $A = \{A_0, A_1, \ldots, A_k\}$ indexekből álló halmaz elemei rendre A(0)-, A(1)-, …, A(k)-indexek. Az A *minimális,* ha az A_i bármelyik két elemét egyesítve az A_i nem marad stabil A_{i-1}-re nézve, $1 \leq i \leq k$. Bizonyítsuk be, hogy tetszőleges gráf esetén egyértelműen létezik olyan minimális A, melynek elemei legdurvább A(i)-indexek, $0 \leq i \leq k$.

30.6. D(k)- és M(k)-indexek

Az A(k)-index használata esetén a k értékét jól kell megválasztani. Túl nagy k esetén nagy lesz az index mérete, túl kicsi érték esetén a pontosság megőrzése céljából túl sokszor kell a talált megoldást ellenőrizni. Az egy osztályba tartozó csúcsok lokálisan hasonlóak, azaz a k távolságú környezetük, pontosabban a legfeljebb k hosszú hozzájuk vezető utak alapján nem különböztethetők meg. Minden csúcsra ugyanazt a k értéket használjuk, annak ellenére, hogy vannak csúcsok, amelyek kevésbé fontosak, mint más csúcsok. Például egyes csúcsok a gyakorlatban nagyon ritkán szerepelnek lekérdezések eredményében, és csak a rajtuk keresztülhaladó utak címkesorozatait vizsgáljuk meg. A kevésbé fontos csúcsokra nem érdemes nagyobb finomítást használni. Ez adja az ötletet a dinamikus D(k)-index használatához, amely különböző k értékeket rendel a csúcsokhoz, a lekérdezésektől függően. Feltételezzük, hogy adott a lekérdezéseknek egy halmaza. Ha például ezek között a lekérdezések között szerepel egy $R.a.b$ és egy $R'.a.b.c$ lekérdezés, ahol R, R'reguláris lekérdezés, akkor a b címkéjű csúcsok esetén legalább 1-biszimuláció, a c címkéjű csúcsok esetén legalább 2-biszimuláció szerinti felbontásra van szükség.

30.35. definíció. *Legyen $I(G)$ a G gráfhoz tartozó indexgráf, és minden I indexcsúcshoz tartozzon egy $k(I)$ nem negatív egész szám. Tegyük fel, hogy az I blokkba tartozó csúcsok $k(I)$-biszimulánsak. A $k(I)$ értékek elégítsék ki a következő feltételt: ha az $I(G)$ gráfban I-ből J-be vezet él, akkor $k(I) \geq k(J) - 1$. Az ilyen tulajdonságú $I(G)$ indexet* ***D(k)-indexnek*** *hívjuk.*

Az elnevezésben a „D” a „dinamikus” szóra utal. Vegyük észre, hogy az A(k)-index a D(k)-index speciális esete, mivel A(k)-index esetén bármely indexcsúcshoz tartozó elemek pontosan k-biszimulánsak. Mivel a címkék alapján történő osztályozás, vagyis az alappartíció A(0)-index, illetve az 1-index véges gráf esetén megegyezik valamilyen k-ra egy A(k)-indexszel, azért ezek is a D(k)-index speciális esetei. A D(k)-index, mint minden index, biztonságos, így elég rajta kiértékelni a lekérdezéseket. A pontosság biztosítására a válaszokat ellenőrizni kell. A következő állítás arról szól, hogy a lekérdezések egy részére a pontosság eleve garantált, így az ilyen lekérdezések esetén az ellenőrző lépés elhagyható.

30.36. állítás. *Legyen I_1, I_2, ..., I_s egy irányított út a D(k)-indexben, és tegyük fel, hogy $k(I_j) \geq j - 1$, ha $1 \leq j \leq s$. Ekkor címke(I_1), címke(I_2), ..., címke(I_s) címkesorozatra az I_s minden egyes eleme illeszkedik.*

Bizonyítás. A bizonyítás s-re vonatkozó indukción alapul. Az $s = 1$ eset triviális. Az indukciós feltevés miatt címke(I_1), címke(I_2), ..., címke(I_{s-1})

címkesorozatra az I_{s-1} minden egyes eleme illeszkedik. Mivel az $I(G)$ gráfban az I_{s-1} csúcsból vezet él az I_s csúcsba, ezért van olyan $u \in I_s$ és $v \in I_{s-1}$, hogy G-ben vezet él v-ből u-ba. Ez azt jelenti, hogy u illeszkedik az $s-1$ hosszú címke(I_1), címke(I_2), ..., címke(I_s) címkesorozatra. Az I_s elemei legalább $(s-1)$-biszimulánsak, ezért I_s minden egyes eleme illeszkedik erre a címkesorozatra. ■

30.37. következmény. *A $D(k)$-index pontos egy $l_0, \ldots, l_m$ címkesorozatra nézve, ha az indexgráfban minden erre illeszkedő I csúcs esetén $k(I) \geq m$,*

A D(k)-index készítése során az alappartíciót, vagyis az A(0)-indexet fogjuk finomítani. A lekérdezések alapján kiinduló értékeket rendelünk az azonos címkéjű csúcsokat tartalmazó osztályokhoz. Tegyük fel, hogy összesen t különböző értéket használunk. Álljon a K_0 halmaz ezekből az értékekből, és legyenek a K_0 elemei $k_1 > k_2 > \ldots > k_t$. Ha a K_0 elemei nem elégítik ki a D(k)-indexben megadott feltételt, akkor a SÚLYMÓDOSÍTÓ algoritmussal a legnagyobb értékből kiindulva növeljük őket úgy, hogy kielégítsék a feltételt. Így az azonos címkéjű csúcsokhoz tartozó osztályok már megfelelő k értékkel fognak rendelkezni. Ezután az osztályokat hasításokkal elkezdjük finomítani, hogy az osztályhoz tartozó elemek k-biszimulánsak legyenek, és a hasítás minden tagjához ezt a k értéket rendeljük. Az eljárás közben a finomítással kapott partíciónak megfelelően az indexgráf éleit is fel kell frissíteni.

SÚLYMÓDOSÍTÓ(G, K_0)

```
1  K = ∅
2  K_1 = K_0
3  while K_1 ≠ ∅
4        do for minden I, ahol I az A(0)-index csúcsa és k(I) = max(K_1)
5              for minden J, ahol J az A(0)-index csúcsa és J-ből vezet él I-be
6                    k(J) = max(k(J), max(K_1) − 1)
7        K = K ∪ {max(K_1)}
8        K_1 = {k(A)| A az A(0)-index csúcsa} \ K
9  return K
```

Könnyen ellenőrizhető, hogy a SÚLYMÓDOSÍTÓ algoritmus műveleti költsége $O(m)$, ahol m az A(0)-index éleinek száma.

D(k)-INDEX-KÉSZÍTŐ(G, K_0)

```
1  legyen I(G) a G gráfhoz tartozó A(0)-index, V_I az I(G) csúcsai,
   E_I az I(G) éleinek halmaza
2  K = SÚLYMÓDOSÍTÓ(G, K_0)                    // A kezdeti súlyok módosítása.
ltétele szerint. 3  for                                          k = 1 to max(K)
4      for minden I ∈ V_I
5          if k(I) ≥ k
6              for minden J, ahol (J, I) ∈ E_I
7                  V_I = (V_I \ {I}) ∪ {I ∩ E(J), I \ E(J)}
8                  k(I ∩ E(J)) ← k(I)
9                  k(I \ E(J)) ← k(I)
10                 E_I = {(A, B) | A, B ∈ V_I, ∃a ∈ A, ∃b ∈ B, (a, b) ∈ E}
11                 I(G) = (V_I, E_I)
12 return I(G)
```

Az algoritmus 7. lépésében egy hasítás műveletet végzünk. Ezzel elérjük, hogy a $(k-1)$-biszimuláció szerint ekvivalens elemeket tartalmazó osztályokat k-biszimuláció szerinti ekvivalencia osztályokra hasítjuk szét. A D(k)-INDEX-KÉSZÍTŐ algoritmusról belátható, hogy a műveleti költsége legrosszabb esetben $O(km)$, ahol m a G gráf éleinek száma, k pedig $max(K_0)$ értékkel egyenlő.

Előfordulhat, hogy a D(k)-index túlságosan finom felosztást eredményez, és nagy mérete miatt nem eléggé hatékony a használata. A túlfinomítás a következő okokra vezethető vissza. A D(k)-INDEX-KÉSZÍTŐ algoritmus az azonos címkéjű csúcsokhoz ugyanazt a k értéket rendeli, holott lehet, hogy ezeknek a csúcsoknak egy része a lekérdezések szempontjából kevésbé fontos, vagy gyakrabban fordul elő k-nál jóval rövidebb lekérdezések eredményében, ezért ezekre a csúcsokra kisebb finomság is elegendő lenne. A SÚLYMÓDOSÍTÓ algoritmus a csúcshoz tartozó k érték alapján a szülő értékét nem fogja csökkenteni, ha az nagyobb, mint $k-1$. Így ha ezek a szülők a gyakori lekérdezések alapján nem túl fontos csúcsok, akkor emiatt túlfinomítás fordulhat elő. A túlfinomítás elkerülésére vezetjük be az M(k)-indexet és az M*(k)-indexet, ahol az „M” a „mixed” („vegyes”) szóra utal, a „*” pedig arra, hogy nem egy indexet adunk meg, hanem fokozatosan finomodó indexekből álló véges hierarchiát. Az M(k)-index egy olyan D(k)-index, amelynek előállításához megadott algoritmus az egyforma címkéjű csúcsokat nem feltétlen sorolja ugyanolyan k-biszimulancia osztályokba.

Először vizsgáljuk meg, hogy egy $I(G) = (V_I, E_I)$ D(k)-indexet hogyan kell módosítani, ha az egyik I indexcsúcshoz tartozó kiindulási k_I súlyt megnöveljük. Ha $k(I) \geq k_I$, akkor az $I(G)$ változatlanul marad. Ellenkező esetben

ahhoz, hogy a D(k)-index súlyokra vonatkozó feltételei teljesüljenek, az I őseihez tartozó súlyokat rekurzívan növelnünk kell, amíg a szülők már legalább $k_I - 1$ súllyal rendelkeznek. Végül az I szülők szerinti széthasításával a szülők súlyainál eggyel nagyobb, tehát legalább k_I lesz a kapott indexcsúcsok finomsága, azaz a hozzájuk tartozó elemek legalább k_I-biszimulánsak lesznek. Mindezt a következő algoritmussal adjuk meg.

SÚLYNÖVELŐ($I, k_I, I(G)$)

```
1  if k(I) ≥ k_I
2     return I(G)
3  for minden (J, I) ∈ E_I
4        I(G) = SÚLYNÖVELŐ(J, k_I − 1, I(G))
5  for minden (J, I) ∈ E_I
6        V_I = (V_I \ {I}) ∪ {I ∩ E(J), I \ E(J)}
7        E_I = {(A, B)|A, B ∈ V_I, ∃a ∈ A, ∃b ∈ B, (a, b) ∈ E}
8           I(G) = (V_I, E_I)
9  return I(G)
```

Könnyen ellenőrizhető az alábbi állítás, amelynek segítségével a későbbiekben egy lépésben tudjuk majd a megfelelő finomságot elérni, azaz nem kell egyesével növelni a finomságokat.

30.38. állítás. *$u \approx^k v$ akkor és csak akkor, ha $u \approx^0 v$ és ha egy u' csúcsból vezet él az u csúcsba, akkor van olyan v' csúcs, amelyből vezet él a v csúcsba és $u' \approx^{k-1} v'$, illetve ez fordítva is igaz, azaz ha egy v' csúcsból vezet él a v csúcsba, akkor van olyan u' csúcs, amelyből vezet él az u csúcsba és $u' \approx^{k-1} v'$.*

Jelölje $GYAK$ a gyakori reguláris lekérdezések által meghatározott egyszerű kifejezések, azaz címkesorozatok halmazát. Azt akarjuk elérni, hogy az index annyira legyen csak finom, amennyi ahhoz kell, hogy a $GYAK$ halmazhoz tartozó lekérdezésekre pontos legyen. Ehhez meghatározzuk a lényeges csúcsokat, és a D(k)-INDEX-KÉSZÍTŐ algoritmust úgy módosítjuk, hogy a finomítást növelő széthasításkor mindig elhagyjuk a nem lényeges csúcsokat, illetve a nem lényeges csúcsok őseit.

Legyen $R \in GYAK$ egy gyakori egyszerű lekérdezés. Az R-re illeszkedő csúcsok halmazát az indexgráfban S, az adatgráfban T jelölje, azaz $S = R(I(G))$ és $T = R(G)$. Az $I(G)$ indexgráfban egy I indexcsúcs finomságát jelölje $k(I)$, vagyis az I-hez tartozó csúcsok maximálisan $k(I)$-biszimulánsak.

FINOMÍT(R, S, T)

```
1  for minden I ∈ S
2      I(G) = INDEXCSÚCSOT-FINOMÍT(I, hossz(R), I ∩ T)
3  while ∃ I ∈ V_I, melyre k(I) < hossz(R) és I illeszkedik R-re
4          I(G) = SÚLYNÖVELŐ(I, hossz(R), I(G))
5  return I(G)
```

Az indexcsúcsok finomítását a következő algoritmussal adjuk meg. Először az I indexcsúcs lényeges szüleit finomítjuk rekurzívan. Ezután I-t a lényeges szülők alapján szétvágjuk úgy, hogy a kapott új részek finomsága k-val legyen egyenlő. A H halmazba kerülnek be az I szétvágott részei. Végül ezek közül összevonjuk azokat, amelyek nem tartalmaznak lényeges csúcsot, és az összevont halmaznál megtartjuk az I eredeti finomságát.

INDEXCSÚCSOT-FINOMÍT(I, k, *lényeges-csúcsok*)

```
1  if k(I) ≥ k
2     return I(G)
3  for minden (J, I) ∈ E_I
4      lényeges-szülők ← E^{-1}(lényeges-csúcsok) ∩ J
5      if lényeges-szülők ≠ ∅
6          INDEXCSÚCSOT-FINOMÍT(J, k − 1, lényeges-szülők)
7  k-előző = k(I)
8  H = {I}
9  for minden (J, I) ∈ E_I
10     if E^{-1}(lényeges-csúcsok) ∩ J ≠ ∅
11         for minden F ∈ H
12             V_I = (V_I \ {F}) ∪ {F ∩ E(J), F \ E(J)}
13             E_I = {(A, B)|A, B ∈ V_I, ∃a ∈ A, ∃b ∈ B, (a, b) ∈ E}
14             k(F ∩ E(J)) = k
15             k(F \ E(J)) = k
16             I(G) = (V_I, E_I)
17             H = (H \ {F}) ∪ {F ∩ E(J), F \ E(J)}
18 maradék = ∅
19 for minden F ∈ H
20     if lényeges-csúcsok ∩F = ∅
21         maradék = maradék ∪ F
22         V_I = (V_I \ {F})
23 V_I = V_I ∪ {maradék}
```

24 $E_I = \{(A,B) | A,B \in V_I, \exists a \in A, \exists b \in B, (a,b) \in E\}$
25 $k(maradék) = k\text{-előző}$
26 $I(G) = (V_I, E_I)$
27 **return** $I(G)$

A FINOMÍT algoritmus egy gyakori egyszerű kifejezés alapján finomítja az $I(G)$ index gráfot úgy, hogy egy indexcsúcsot nem feltétlen azonos finomságú részekre bont fel, és ezzel a túlfinomítást elkerüli. Ha az A(0)-indexből indulunk ki, és egymás után minden gyakori lekérdezésre elkészítjük a finomítást, akkor a gyakori lekérdezésekre nézve pontos indexgráfot kapunk. Ezt nevezzük M(k)-indexnek. A gyakori lekérdezések $GYAK$ halmaza az idők folyamán változhat, ezért az indexet is dinamikusan módosítani kell.

30.39. definíció. *Az* ***M(k)-index*** *olyan $D(k)$-index, amelyet az alábbi* M(k)-INDEX-KÉSZÍTŐ *algoritmussal állítunk elő.*

M(k)-INDEX-KÉSZÍTŐ($G, GYAK$)

1 $I(G) =$ a G gráfhoz tartozó A(0)-index
2 $V_I =$ az $I(G)$ csúcsai
3 **for** minden $I \in V_I$
4 $\quad k(I) \leftarrow 0$
5 $E_I =$ az $I(G)$ éleinek halmaza
6 **for** minden $R \in GYAK$
7 $\quad I(G) =$ FINOMÍT(R, $R(I(G))$, $R(G)$)
8 **return** $I(G)$

Az M(k)-index a gyakori kifejezésekre nézve pontos index. Egy nem gyakori lekérdezés esetén a következőképpen járunk el. Az M(k)-index egyben D(k)-index is, ezért ha az $I(G)$ indexgráfban egy indexcsúcs illeszkedik egy R egyszerű lekérdezésre, és az indexcsúcs finomsága legalább akkora, mint az R hossza, akkor az indexcsúcs minden eleme illeszkedik az R lekérdezésre a G gráfban. Ha kisebb az indexcsúcs finomsága, akkor minden eleméről a NAIV-KIÉRTÉKELÉS alapján ellenőrizni kell, hogy megoldás-e a G gráfban.

Az M(k)-index használata esetén akkor a legkisebb a túlfinomítás, ha a gyakori egyszerű lekérdezések hossza megegyezik. Ha nagy eltérések fordulhatnak elő a gyakori lekérdezések hosszában, akkor előfordulhat, hogy a rövid lekérdezések számára túlságosan finom indexet használunk. Készítsük el azt a

fokozatosan finomodó indexsorozatot, amellyel az A(0)-indexből indulva eljutunk az M(k)-indexhez úgy, hogy egy lépésben egy indexcsúcsot legfeljebb eggyel nagyobb finomságú részekre bontunk fel. Ha a teljes indexsorozatot ismerjük, akkor egy egyszerű lekérdezés kiértékeléséhez nem kell a legfinomabb, és ezért legnagyobb indexet használni, hanem csak a lekérdezés hosszának megfelelő finomságú indexet.

30.40. definíció. *Az* ***M***(k)**-index** *olyan I_0, I_1, ... I_k indexsorozat, amelyre a következő tulajdonságok teljesülnek:*

1. *Minden I_i index M(i)-index, ahol $i = 0, 1, \ldots, k$.*
2. *Az I_i indexben minden indexcsúcs finomsága legfeljebb i, ahol $i = 0, 1, \ldots, k$.*
3. *Az I_{i+1} az I_i finomítása, ahol $i = 0, 1, \ldots, k-1$.*
4. *Ha az I_i index J csúcsát az I_{i+1} indexben felbontjuk, és a felbontás egyik halmaza J', azaz $J' \subseteq J$, akkor $k(J) \leq k(J') \leq k(J) + 1$.*
5. *Legyen J az I_i index csúcsa, és $k(J) < i$. Ekkor $i < i'$ esetén, az $I_{i'}$ minden olyan J' indexcsúcsára, amelyre $J' \subseteq J$, teljesül, hogy $k(J) = k(J')$.*

A definícióból következik, hogy M*(k)-index esetén I_0 az A(0)-indexszel egyezik meg. Az utolsó tulajdonság azt mondja ki, hogy ha valamelyik indexcsúcs finomítása megáll, akkor már később sem fog változni a finomsága. Az M*(k)-index rendelkezik az M(k)-index jó tulajdonságaival, a felépítése is hasonló, azaz a gyakori lekérdezések alapján szükség esetén tovább finomítjuk az indexet úgy, hogy a gyakori lekérdezésekre pontos legyen, de most az egymás utáni durvább indexeket is megtartjuk és frissítjük, nem csak a legfinomabbat.

Az M*(k)-index ábrázolásánál ki lehet használni, hogy ha egy indexcsúcsot már nem bontunk tovább, akkor onnan kezdve nem kell minden későbbi indexben tárolni ezt a csúcsot, hanem elég hivatkozni rá. Hasonlóan az ilyen csúcsok közti éleket sem kell ismételten ábrázolni a indexsorozatban, hanem elég hivatkozni rájuk. Az M*(k)-index készítése az M(k)-INDEX-KÉSZÍTŐ algoritmushoz hasonló módon történhet. Az algoritmus részletes leírása He és Yang cikkében található meg.

Az M*(k)-index segítségével többféle stratégiát is alkalmazhatunk a lekérdezések kiértékelésére. Legyen R egy gyakori egyszerű lekérdezés.

A legegyszerűbb stratégia, hogy a lekérdezés hosszával megegyező finomságú indexet használjuk.

M*(k)-INDEXES-NAIV-KIÉRTÉKELÉS(G, $GYAK$, R)

```
1  {I_0, I_1 ,..., I_k} = a G gráfhoz tartozó M*(k)-index
2  h = hossz(R)
3  return INDEXES-KIÉRTÉKELÉS(G, I_h, A_R)
```

A kiértékelés úgy is történhet, hogy a lekérdezés egyre hosszabb előtagjait értékeljük ki az előtag hosszával megegyező sorszámú index alapján. Az előtag kiértékeléséhez az eggyel rövidebb előtag kiértékelésénél megtalált csúcsok felbontásait vesszük a következő indexben, és ezekből indulva keresünk olyan éleket, amelyeknek a következő szimbólum a címkéje. Legyen $R = l_0, l_1, \ldots, l_h$ egyszerű gyakori lekérdezés, azaz $hossz(R) = h$.

M*(k)-INDEXES-FELÜLRŐL-LEFELÉ-KIÉRTÉKELÉS(G, $GYAK$, R)

```
1  {I_0, I_1, ..., I_k} = a G gráfhoz tartozó M*(k)-index
2  R_0 = l_0
3  H_0 = ∅
4  for minden C ∈ E_{I_0}(gyökér(I_0))      // A gyökér gyerekei az I_0 gráfban.
5      if címke(C) = l_0
6          H_0 = H_0 ∪ {C}
7  for j = 1 to hossz(R)
8      H_j = ∅
9      R_j = R_{j-1}.l_j
10     H_{j-1} = M*(k)-INDEXES-FELÜLRŐL-LEFELÉ-
       KIÉRTÉKELÉS(G,GYAK, R_{j-1})
11     for minden A ∈ H_{j-1}          // Az A csúcs az I_{j-1} gráfnak csúcsa.
12         if A = ∪B_m, ahol B_m ∈ V_{I_j}          // Az A csúcs felbontása
                                                    // az I_j gráfban.
13             for minden B_m
14                 for minden C ∈ E_{I_j}(B_m)
                                        // B_m minden gyerekére az I_j gráfban.
15                     if címke(C) = l_j
16                         H_j = H_j ∪ {C}
17 return H_h
```

Az is lehet a stratégiánk, hogy először keresünk egy olyan részsorozatot az egyszerű lekérdezésnek megfelelő címkesorozatban, amelyre kevés csúcs illeszkedik, vagyis nagy a szelektivitása. A részsorozat hosszához tartozó indexben megkeressük az illeszkedő csúcsokat, majd az indexsorozat

alapján megnézzük, hogy milyen csúcsokra bontjuk fel ezeket a csúcsokat a lekérdezés hosszához tartozó finomabb indexben. Végül ezekből a csúcsokból indulva megnézzük, hogy milyen csúcsok illeszkednek az eredeti lekérdezés maradék részére. Az ehhez a módszerhez tartozó $\mathrm{M}^*(k)$-INDEXES-ELŐSZŰRT-KIÉRTÉKELÉS algoritmus részletes kidolgozását az Olvasóra bízzuk.

Gyakorlatok

30.6-1. Dolgozzuk ki részletesen az $\mathrm{M}^*(k)$-INDEXES-ELŐSZŰRT-KIÉRTÉKELÉS algoritmust. Mi lesz az algoritmus költsége?

30.6-2. Bizonyítsuk be a 30.38. állítást.

30.6-3. Igazoljuk, hogy a SÚLYMÓDOSÍTÓ algoritmus műveletei költsége $O(m)$, ahol m az A(0)-index éleinek száma.

30.7. Elágazó lekérdezések

Reguláris lekérdezések segítségével a gráfból azokat a csúcsokat tudjuk kiválasztani, amelyekhez a gyökérből vezet egy olyan út, amelyen a címkék egy előre megadott reguláris mintára illeszkednek. Természetesnek tűnő általánosítás, hogy a csúcshoz vezető úton elhelyezkedő csúcsokra további feltételeket adjunk meg. Például kiköthetjük, hogy egy adott címkéjű csúcsból egy másik címkesorozattal legyen elérhető egy másik csúcs. Megfogalmazhatunk olyan feltételeket is, hogy egy adott címkéjű csúcshoz egy másik csúcsból valamilyen megadott címkesorozatú út vezessen. Ezekből a feltételekből többet is vehetünk, tagadásukat is használhatjuk, egymásba is ágyazhatjuk őket. A feltételek ellenőrzéséhez nemcsak előre kell lépegetni az élek irányításnak megfelelően, hanem időnként visszafele is. A következőkben megadjuk az elágazó lekérdezések nyelvének leírását, és bevezetjük az előre-hátra (forward-backward) indexeket. A összes elágazó lekérdezésre nézve biztonságos és pontos előre-hátra indexet FB-indexnek nevezzük. Az 1-indexhez hasonlóan ez általában túlságosan nagy méretű, ezért ehelyett inkább olyan $\mathrm{FB}(f, b, d)$-indexet használunk, amely akkor lesz pontos, ha az elágazó lekérdezésben előre egyszerre legfeljebb f hosszan megyünk, visszafele legfeljebb b hosszan, és a feltételek egymásba ágyazása valójában legfeljebb d mélységű. A gyakorlatban előforduló esetekben az f, b és d értéke általában kicsi. Azokra a lekérdezésekre, amelyeknél valamelyik paraméter értéke nagyobb, mint az indexben szereplő megfelelő érték, egy ellenőrző lépést is be kell iktatni, vagyis az indexen kiértékeljük a lekérdezést, és a kapott indexcsúcsokban szereplő csúcsok közül csak azokat tartjuk meg, amelyek szintén kielégítik a lekérdezést.

Ha az n csúcsból irányított él vezet az m csúcsba, akkor ezt n/m vagy

$m\backslash n$ jelöléssel adhatjuk meg. Ha az n csúcsból irányított úttal elérhető az m csúcs, akkor ezt $n//m$ vagy $m\backslash\backslash n$ formában jelöljük. (Eddig / helyett a . jelet használtuk, így // az _*, vagy röviden a * reguláris kifejezésnek felel meg.)

Címkesorozaton ezentúl olyan sorozatot értünk, amelyben elválasztójelek az előre-jelek (/, //) és a hátra-jelek (\, \\) lehetnek. Egy csúcssorozat illeszkedik egy címkesorozatra, ha az egymás utáni csúcsok rendre olyan viszonyban állnak egymással, amilyet a megfelelő elválasztójel meghatároz, és a csúcsok címkéi a címkesorozatban felsorolt címkék szerint követik egymást.

Előre-címkesorozatban csak előre-jelek, ***hátra-címkesorozatban*** csak hátra-jelek találhatók.

Az ***elágazó lekérdezéseket*** a következő nyelvtan definiálja.

elágazó_lekérdezés	::=	előre_címkesorozat [vagy_kifejezés] előre_jel elágazókifejezés
		| előre_címkesorozat [vagy_kifejezés]
		| előre_címkesorozat
vagy_kifejezés	::=	és_kifejezés or vagy_kifejezés
		| és_kifejezés
és_kifejezés	::=	elágazó_feltétel and és_kifejezés
		| nem_elágazó_feltétel and és_kifejezés
		| elágazó_feltétel
		| nem_elágazó_feltétel
nem_elágazó_feltétel	::=	not elágazó_feltétel
elágazó_feltétel	::=	feltétel_címkesorozat [vagy_kifejezés] elágazó_feltétel
		| feltétel_címkesorozat [vagy_kifejezés]
		| feltétel_címkesorozat
feltétel_címkesorozat	::=	előre_jel címkesorozat
		| hátra_jel címkesorozat

Az elágazó lekérdezésben egy adott címkéjű csúcsra vonatkozó feltétel akkor igaz, ha létezik olyan csúcssorozat, amely illeszkedik a feltételre. Például a $gyökér//a/b[\backslash c//d$ and not $\backslash e/f]/g$ lekérdezés azokat a g címkéjű csúcsokat keresi meg, amelyekhez a gyökérből el lehet jutni úgy, hogy a két utolsó előtti csúcs címkéje a és b, továbbá teljesül erre a b címkéjű csúcsra, hogy egyrészt létezik olyan c címkéjű szülője, amelynek van d címkéjű leszármazottja, másrészt nincs olyan e címkéjű gyereke, amelynek lenne f címkéjű szülője.

Ha egy elágazó lekérdezésből elhagyjuk az összes [ ] jelek közé zárt feltételt, akkor az elágazó lekérdezéshez tartozó ***főlekérdezést*** kapjuk. Az előbbi példában ez a $gyökér//a/b/g$ lekérdezés. A főlekérdezés mindig egy előre-címkesorozatnak felel meg.

Az elágazó lekérdezéseknek természetes módon megfeleltethetünk egy irányított gráfot. A lekérdezésben szereplő címkesorozatnak ugyanolyan címkéjű csúcsokat feleltetünk meg, a / és a \ elválasztójel esetén az egymás utáni csúcsokat a megfelelő irányítású éllel kötjük össze, a // és a \\ esetén

szintén a megfelelő irányított élt rajzoljuk be, és megcímkézzük az élt // vagy \\ címkével. Végül a logikai összekapcsolók jelét a hozzátartozó feltételek kezdő élének címkéjeként vesszük fel. Így előfordulhat, hogy egy élnek két címkéje is lesz például // és „and". Vegyük észre, hogy a megadott nyelvtanból következően a kapott gráf nem tartalmazhat irányított kört.

Az így kapott fa alapján definiálhatjuk a lekérdezés bonyolultságának egy egyszerű mértékét. A főlekérdezés csúcsaihoz, és azokhoz a csúcsokhoz, amelyekből irányított út vezet a főlekérdezés valamely csúcsához, 0-t rendelünk. Ezután a 0 jelű csúcsokból irányított úttal elérhető eddig nem jelölt csúcsokhoz 1-et rendelünk. Azokhoz az eddig nem jelölt csúcsokhoz rendelünk 2-t, amelyekből elérhető 1 jelű csúcs. A 2 jelű csúcsokból elérhető nem jelölt csúcsokhoz 3-at rendelünk, és így tovább, a $2k$ jelű csúcsokból elérhetők, nem jelölt csúcsokhoz $2k+1$-et rendelünk, ezután azokhoz a nem jelölt csúcsokhoz, amelyekből a $2k+1$ jelű csúcsok elérhető, $2k+2$-t rendelünk. A lekérdezéshez tartozó legnagyobb értékű csúcs értékét a ***fa mélységének*** nevezzük. A fa mélysége azt fejezi ki, hogy a lekérdezés kiértékelése során hányszor kell irányt váltani, azaz az élek irányításának megfelelően gyerekeket, vagy fordítva, szülőket kell keresni. Ugyanazt a lekérdezést a feltételek különböző mélységű beágyazásával többféle módon is megadhattuk volna, de belátható, hogy az így definiált érték független a felírás módjától, ezért nem a feltételek egymásba ágyazásának számával definiáltuk a lekérdezés bonyolultságát.

Az 1-index a beérkező útvonalak alapján osztályozta a csúcsokat, a biszimuláció alapján. A kiszámításhoz használt stabilitás fogalma az ***utód-stabilitás*** volt, azaz a gráf csúcsainak A halmaza ***utód-stabil*** a csúcsok egy B halmazára nézve, ha $A \subseteq E(B)$ vagy $A \cap E(B) = \emptyset$, ahol $E(B)$ a B-ből éllel elérhető csúcsok halmazát jelöli. Egy partíció utód-stabil, ha bármely két partíciós tag utód-stabil egymásra nézve. Az 1-index az azonos címkék alapján történő osztályozás legdurvább utód-stabil partíciója, amely a PT-algoritmussal számolható ki. Elágazó lekérdezések esetén az élek irányításával szemben is kell haladni, így az ellenkező irányultságú ***elől-stabilitás*** fogalmára is szükségünk lesz. A gráf csúcsainak A halmaza ***elől-stabil*** a csúcsok egy B halmazára nézve, ha $A \subseteq E^{-1}(B)$ vagy $A \cap E^{-1}(B) = \emptyset$, ahol $E^{-1}(B)$ azokat a csúcsokat jelöli, amelyekből a B valamelyik csúcsa elérhető.

30.41. definíció. *Az **FB-index** az alappartíció legdurvább olyan finomítása, amely egyszerre elől-stabil és utód-stabil.*

Vegyük észre, hogy ha a gráf éleinek irányítását megfordítjuk, akkor az elől-stabil partícióból utód-stabil partíció lesz, és fordítva, tehát a PT-algoritmus, illetve annak javításai használhatók a legdurvább elől-stabil partíció kiszámítására. A következő algoritmusban ezt használjuk fel. Kiin-

dulunk az azonos címkéjű osztályozásból, kiszámítjuk az ehhez tartozó 1-indexet, megfordítjuk az élek irányítását, és ezt tovább finomítjuk úgy, hogy kiszámoljuk az ehhez tartozó 1-indexet. Amikor megáll az algoritmus, akkor a kiindulási partíciónak olyan finomítását kapjuk, amely előd-stabil és utód-stabil is egyszerre. Az Olvasóra bízzuk annak bizonyítását, hogy a legdurvább ilyen tulajdonságú partíciót kaptuk.

FB-INDEX-KÉSZÍTŐ(V, E)

```
1  P = A(0)                    // Az azonos címkéjű osztályozásból indulunk ki.
2  while P változik
3        P = PT(V, E^{-1}, P)                    // Kiszámítjuk az 1-indexet.
4        P = PT(V, E, P)          // Megfordítjuk az élek irányítását, és
                                  // kiszámoljuk az 1-indexet.
5  return P
```

A kétféle stabilitásból egyszerűen adódik az alábbi következmény.

30.42. következmény. *Az FB-index biztonságos és pontos az elágazó lekérdezésekre nézve.*

Az algoritmus bonyolultságát vissza lehet vezetni a PT-algoritmus bonyolultságára. Mivel a P mindig az előző partíció finomítása, ezért az a legrosszabb eset, ha a finomítás egyesével történik, azaz mindig az egyik partíciós tagból veszünk ki egy elemet, és tesszük egy egyelemű, önálló partíciós tagba. Így legrosszabb esetben a ciklust $O(n)$-szer hajtjuk végre. Tehát az algoritmus költsége legfeljebb $O(mn \lg n)$.

Azt a partíciót, amelyet úgy kapunk, hogy csak egyszer hajtjuk végre a ciklusmagot, ***F+B-indexnek*** hívjuk, ha kétszer hajtjuk végre a ciklusmagot ***F+B+F+B-indexről*** beszélünk, és így tovább.

Könnyen belátható az alábbi állítás.

30.43. állítás. *Az F+B+F+B+$\cdots$+F+B-index, ahol d-szer szerepel az F+B tag, biztonságos és pontos a legfeljebb d mélységű elágazó lekérdezésekre nézve.*

Az FB-index alapján egy osztályba kerülő csúcsok elágazó lekérdezéssel nem különböztethetők meg egymástól. Általában ez túl erős megkötés, és ezért az FB-index mérete rendszerint nem sokkal kisebb, mint az eredeti gráf mérete. Nagyon hosszú elágazó lekérdezések a gyakorlatban ritkán fordulnak elő, ezért az A(k)-indexekhez hasonlóan csak lokális ekvivalenciát követelünk meg, de most ezt két paraméterrel jellemezzük attól függően, hogy az irányításnak megfelelő, vagy fordított iránynak megfelelő utak

hosszát akarjuk korlátozni. A lekérdezés mélységét is előre korlátozhatjuk. Bevezetjük az FB(f, b, d)-indexet, amellyel az ilyen típusú feltételekkel korlátozott elágazó lekérdezések pontosan kiértékelhetők. A korlátozásoknak eleget nem tevő elágazó lekérdezéseket is az indexen értékeljük ki, csak ebben az esetben az eredményt még ellenőrizni is kell.

FB(f, b, d)-INDEX-KÉSZÍTŐ(V, E, f, b, d)

```
1  P = A(0)                    // // Az azonos címkéjű osztályozásból indulunk ki.
2  for i = 1 to d
3      P = NAIV-KÖZELÍTÉS(V, E^{-1}, P, f) // Kiszámítjuk az A(f)-indexet.
4      P = NAIV-KÖZELÍTÉS(V, E, P, b)      // Megfordítjuk az élek
                                           // irányítását, és kiszámoljuk az A(b)-indexet.
5  return P
```

Az algoritmus költsége az A(k)-index kiszámítási költsége alapján legfeljebb $O(dm \max(f, b))$, ami sokkal kedvezőbb, mint az FB-index kiszámítási költsége, és eredményül általában jóval kisebb indexgráfot kapunk.

A kapott indexre nyilván teljesül a következő állítás.

30.44. állítás. *Az FB(f, b, d)-index biztonságos és pontos azokra az elágazó lekérdezésekre, amelyekben az előre-sorozatok hossza legfeljebb f, a hátra-sorozatok hossza legfeljebb b, és a lekérdezéshez tartozó fa mélysége legfeljebb d.*

Speciális esetként kapjuk, hogy az FB(∞, ∞, ∞)-index megegyezik az FB-indexszel, az FB(∞, ∞, d)-index megegyezik az F+B+$\cdots$+F+B-indexszel, ahol az F+B tag d-szer szerepel, az FB$(\infty, 0, 1)$-index megegyezik az 1-indexszel, és végül az FB$(k, 0, 1)$-index megegyezik az A(k)-indexszel.

Gyakorlatok

30.7-1. Bizonyítsuk be, hogy a FB-INDEX-KÉSZÍTŐ algoritmus az alappartíció legdurvább előd-stabil és utód-stabil finomítását állítja elő.

30.7-2. Bizonyítsuk be a 30.44. állítást.

30.8. Az indexek frissítése

Az adatbázis-kezelésben általában három fontos szempontot tartunk szem előtt. Minél kisebb legyen a tároláshoz szükséges hely, minél gyorsabbak legyenek a lekérdezések, és minél gyorsabban lehessen az adatbázisban

beszúrást, törlést, módosítást végrehajtani. Általában az egyik szempont szerinti jó megoldás rosszabb a másik szempontból. A tipikus lekérdezésekre vonatkozó indexek hozzáadásával növekszik a tárolás mérete, viszont cserébe a lekérdezéseket az indexeken lehet kiértékelni, így gyorsabbak a lekérdezések. A gyakran módosuló, dinamikus adatbázisok esetén számolnunk kell azzal is, hogy nem csak az eredeti adatokon kell a módosítást végrehajtani, hanem az indexeket is megfelelőképpen kell megváltoztatni. A triviálisan pontos, de legköltségesebb módszer, hogy minden egyes módosítás esetén újra elkészítjük az indexeket. Érdemes olyan eljárásokat keresni, hogy a meglevő indexek minél kisebb módosításával jussunk el azokhoz az indexekhez, amelyek már a változásokat tükrözik.

Előfordulhat, hogy az indexet, vagy annak módosítását is megindexeljük. Egy index indexe az eredeti gráfnak is indexe, bár formailag indexcsúcsokat tartalmazó osztályokból áll, de egyesíthetjük az egy osztályba tartozó indexcsúcsok elemeit. Látható, hogy ezzel a megfeleltetéssel a gráf csúcsainak egy partícióját kapjuk, vagyis egy indexet.

A következőkben a félig strukturált adatbázisok módosításai közül azt vizsgáljuk részletesebben, mikor egy új gráfot kapcsolunk a gyökérhez, illetve mikor egy új élt adunk a gráfhoz. Mindkettő tipikus művelet, mivel ezekre van szükség egy új weboldal készítésekor, illetve egy új hivatkozás elhelyezésekor.

Tegyük fel, hogy $I(G)$ a G gráf 1-indexe. Legyen H olyan gráf, amelynek nincs G-vel közös csúcsa. Jelölje $I(H)$ a H 1-indexét. Legyen $F = G + H$, azaz F az a gráf, amelyet úgy kapunk, hogy a G és H gyökerét egyesítjük. Az $I(G + H)$-t szeretnénk előállítani $I(G)$ és $I(H)$ segítségével. Ebben segít a következő állítás.

30.45. állítás. *Legyen a G gráf 1-indexe $I(G)$, és legyen J az $I(G)$ tetszőleges finomítása. Ekkor $I(J) = I(G)$.*

Bizonyítás. Legyen u, v a G két csúcsa. Azt kell belátni, hogy u és v biszimuláns a G-ben az 1-index szerint akkor és csak akkor, ha $J(u)$ és $J(v)$ biszimuláns az $I(G)$ indexgráfban az $I(G)$ 1-indexe szerint. Legyen u és v biszimuláns a G-ben az 1-index szerint. Belátjuk, hogy megadható olyan biszimuláció, amely szerint $J(u)$ és $J(v)$ biszimuláns az $I(G)$-ben. Mivel az 1-index a legdurvább biszimulációnak megfelelő partíció, ezért a megadott biszimuláció az 1-indexnek megfelelő biszimulációnak finomítása, vagyis $J(u)$ és $J(v)$ az $I(G)$ 1-indexének megfelelő biszimuláció szerint is biszimulánsak. Legyen $J(a) \approx' J(b)$ akkor és csak akkor, ha a és b biszimuláns G-ben az 1-index szerint. Vegyük észre, hogy mivel J az $I(G)$ finomítása, ezért $J(a) \approx' J(b)$ esetén $J(a)$ és $J(b)$ minden eleme biszimuláns egymással G-ben. Ahhoz, hogy megmutassuk, hogy a $\approx'$ reláció biszimuláció, legyen $J(u')$ a $J(u)$ szülője,

ahol u' az u_1 szülője, és u_1 a $J(u)$ eleme. Ekkor u_1, u és v biszimuláns G-ben, tehát van a v-nek olyan v' szülője, hogy u' és v' biszimuláns G-ben. Emiatt $J(v)$-nek $J(v')$ a szülője, és $J(u') \approx' J(v')$. Mivel a biszimuláció szimmetrikus, ezért a $\approx'$ reláció is szimmetrikus. Ezzel a bizonyítás egyik irányát beláttuk.

Legyen most $J(u)$ és $J(v)$ biszimuláns $I(G)$-ben az $I(G)$ 1-indexe szerint. A fentiekhez hasonlóan, elég belátni, hogy megadható olyan biszimuláció a G csúcsain, amely szerint u és v biszimulánsak. Legyen $a \approx' b$ akkor és csak akkor, ha $J(a) \approx J(b)$ az $I(G)$ 1-indexe szerint. A biszimuláció igazolásához legyen u' az u-nak szülője. Ekkor $J(u')$ is szülője $J(u)$-nak. Mivel $u \approx' v$ esetén $J(u)$ és $J(v)$ biszimulánsak, ezért van olyan $J(v'')$ szülője a $J(v)$-nek, hogy $J(u')$ és $J(v'')$ biszimuláns az $I(G)$ 1-indexe szerint, és v'' szülője a $J(v)$ valamely v_1 elemének. Mivel v és v_1 biszimulánsak, ezért v-nek van olyan v' szülője, amelyre igaz, hogy v' és v'' biszimuláns. Felhasználva a bizonyítás másik irányát, ebből következik, hogy $J(v')$ és $J(v'')$ biszimuláns az $I(G)$ 1-indexe szerint. A tranzitivitás miatt ekkor $J(u')$ és $J(v')$ biszimulánsak az $I(G)$ 1-indexe szerint, tehát $u' \approx' v'$. Mivel a definiált $\approx'$ relációs szimmetrikus, így biszimulációt kaptunk. ∎

Az állítás következményeként diszjunkt G, H gráfok esetén az $I(G+H)$-t az alábbi algoritmussal lehet előállítani.

GRÁFHOZZÁADÁS-1-INDEXE(G, H)

```
1  P_G = A_G(0)                          // P_G a címkék szerinti alappartíció.
2  P_H = A_H(0)                          // P_H a címkék szerinti alappartíció.
3  I_1 = PT(V_G, E_G^{-1}, P_G)                    // I_1 a G 1-indexe.
4  I_2 = PT(V_H, E_H^{-1}, P_H)                    // // I_2 a H 1-indexe.
5  J = I_1 + I_2          // Az 1-indexeket összekapcsoljuk a gyökerüknél.
6  P_J = A_J(0)                          // P_J a címkék szerinti alappartíció.
7  I = PT(V_J, E_J^{-1}, PJ)                       // I a J 1-indexe.
8  return I
```

Az algoritmus költségének kiszámításánál feltesszük, hogy a G-hez tartozó $I(G)$ 1-index már a rendelkezésünkre áll. Ekkor az $I(G+H)$ készítésének összköltsége $O(m_H \lg n_H + (m_{I(H)} + m_{I(G)}) \lg(n_{I(G)} + n_{I(H)}))$, ahol n, m a megfelelő gráf csúcsainak, illetve éleinek számát jelöli.

Ahhoz, hogy az algoritmus jól működik, csak azt kell észrevenni, hogy a diszjunktság miatt az $I(G) + I(H)$ az $I(G+H)$-nak finomítása. Ebből az is következik, hogy $I(G) + I(H)$ biztonságos és pontos index, így ezt is használhatjuk, ha nem célunk a minimális index megkeresése. Ez különösen akkor hasznos, ha többször kell gráfot hozzáadni a már meglevő gráfhoz. Ilyenkor

a ***lusta módszert*** használjuk, azaz nem páronként számoljuk ki a minimális indexet, hanem összeadjuk a tagok indexeit, és csak egyszer minimalizálunk.

30.46. állítás. *Legyen a G_i gráf 1-indexe $I(G_i)$, $i = 1, \ldots, k$, és a gráfoknak ne legyen közös csúcsuk. Ekkor a gyökereknél összekapcsolt gráfok $I(G_1 + \cdots + G_k)$ 1-indexére igaz, hogy $I(G_1 + \cdots + G_k) = I(I(G_1) + \cdots + I(G_k))$.*

A következőkben azt vizsgáljuk, hogy mi történik az indexszel, ha a gráfban behúzunk egy új élt. Egy ilyen műveletnek is jelentős lehet a hatása. Nem nehéz olyan gráfot konstruálni, hogy az indexe a gyökértől 2 távolságban két teljesen azonos részgráfot tartalmazzon, amelyek pont egy él hiánya miatt nem vonhatók össze. Ha ezt a kritikus élt húzzuk be, akkor a két részgráf összevonható, és az indexgráf mérete majdnem a felére csökken.

Tegyük fel, hogy a G gráfban egy új élt húzunk be u-ból v-be. Legyen az így kapott gráf G', azaz $G' = G + (u, v)$. Az $I(G)$ partíció legyen a G 1-indexe. Ha az $I(u)$-ból mutatott él az $I(v)$-be az $I(G)$-ben, akkor nem kell módosítani az indexgráfot, mert a stabilitás miatt az $I(v)$ elemeinek, azaz a v-vel biszimuláns összes csúcsnak létezik szülője az $I(u)$-ban, melynek elemei az u-val biszimulánsak. Vagyis $I(G') = I(G)$.

Ha az $I(u)$-ból nem vezetett él az $I(v)$-be, akkor be kellene húzni ezt az élt, de ezzel elromolhat az a tulajdonság, hogy $I(v)$ stabil az $I(u)$-ra nézve. Legyen Q az a partíció, amelyet úgy kapunk az $I(G)$-ből, hogy az $I(v)$-t kettévágjuk, az egyik tagba áttesszük a v-t, a többit pedig a másik tagba tesszük, és minden más partíciós tagot változatlanul hagyunk. A Q meghatározza az éleit a szokásos módon, azaz ha az egyik partíciós tag valamely eleméből vezet él egy másik partíciós tag valamely elemébe, akkor a két partíciós tagot ugyanilyen irányítással összekötjük.

Legyen az X partíció az eredeti $I(G)$. Ekkor Q finomítása X-nek, és Q stabil az X-re nézve a G' alapján. Vegyük észre, hogy a PT-algoritmusban használt X és Q partíciókra pont ez az invariáns tulajdonság szerepelt. A 30.45. állítás szerint elég az $I(G')$ egy finomítását megtalálni. Ha a G' alappartíciójának egy tetszőleges stabil finomítását meg tudjuk találni, akkor mivel az 1-index a legdurvább stabil finomítás, így ez az $I(G')$-nek finomítása lenne. Az X az alappartíciónak, vagyis a címkék szerinti osztályozásnak a finomítása, és ugyanez igaz a Q-ra is. Tehát ha Q stabil, akkor készen vagyunk. Ha nem stabil, akkor stabilizálhatjuk a PT-algoritmussal, amelyet most a fenti X és Q partíciókkal indítunk. Először azokat a partíciós tagokat kell megvizsgálni, amelyek v-nek valamelyik gyerekét tartalmazzák, mert ezek nem biztos, hogy stabilak maradtak a kettévágással keletkezett két partíciós tagra nézve. A PT-algoritmus kettéhasítással stabilizálja ezeket, de emiatt ezek gyerekeit is meg kell vizsgálni, mert ezeknél is elromolhatott a stabilitás, és így tovább.

Ezzel a stabilitás-terjesztő módszerrel végül egy stabil finomításhoz jutunk el. Mivel közben csak a v-ből elérhető csúcsokat járjuk be, így lehet, hogy nem a legdurvább stabil finomítást kaptuk. Ezzel beláttuk, hogy a következő algoritmus a $G + (u, v)$ gráf 1-indexét számolja ki.

ÉLHOZZÁADÁS-1-INDEXE(G, (u, v))

```
1  P_G = A_G(0)                          // P_G a címkék szerinti alappartíció.
2  I = PT(V_G, E_G^{-1}, P_G)            // a G 1-indexe.
3  G' = G + (u, v)                       // Behúzzuk az (u, v) élt.
4  if (I(u), I(v)) ∈ E_I                 // Ha I(u)-ből I(v)-be mutatott él,
                                         akkor nem kell módosítani.
5     return I
6  I' = {v}                              // Kettévágjuk az I(v)-t.
7  I'' = I(v) \ {v}
8  X = I                                 // X a régi partíció.
9  E_I = E_I ∪ {(I(u), I(v))}            // Behúzunk egy élt I(u)-ból I(v)-be.
10 Q = (I \ {I(v)}) ∪ {I', I''}          // I(v)-t kicseréljük I'-re és I''-re.
11 E = E_Q                               // Meghatározzuk Q éleit.
12 J = PT(V_G', E_G'^{-1}, P_G', X, Q)   // A PT-algoritmust X-szel és Q-val
                                         indítva futtatjuk le.
13 J = PT(V_J, E_J^{-1}, P_J)            // J a legdurvább stabil finomítás.
14 return J
```

A gyakorlati tapasztalatok azt mutatják, hogy a 13. lépést elhagyhatjuk, mert a 12. lépésben előállított stabil finomítás már elég jó közelítése a legdurvább stabil finomításnak, alig 5% a méretük közti különbség.

A következőkben az FB-indexek és A(k)-indexek frissítésével foglalkozunk. Az FB-index az 1-indextől annyiban tér el, hogy két csúcs akkor kerül egy hasonlósági osztályba, ha nemcsak a bejövő utak, hanem a kimenő utak mentén is ugyanolyan címkesorozatokból álló halmazokat kapunk mindkét csúcsra. Láttuk, hogy az FB-index készítéséhez a PT-algoritmust kell kétszer futtatni, úgy, hogy másodszor a fordított irányítású gráfra kell alkalmazni az algoritmust. Az FB-index frissítése az 1-indexhez hasonlóan történik. A következő állítás bizonyítása a 30.45. állítás bizonyításához hasonlóan történik, ezért ennek igazolását az Olvasóra bízzuk.

30.47. állítás. *Legyen a G gráf FB-indexe $I(G)$, és legyen J az $I(G)$ tetszőleges finomítása. Jelöljük $I(J)$-vel a J FB-indexét. Ekkor $I(J) = I(G)$.*

Az állítás következményeként diszjunkt G, H gráfok esetén a $G + H$ FB-

indexét az alábbi algoritmussal lehet előállítani.

GRÁFHOZZÁADÁS-FB-INDEXE(G, H)

1 $I_1 =$ FB-INDEX-KÉSZÍTŐ(V_G, E_G) // I_1 a G FB-indexe.
2 $I_2 =$ FB-INDEX-KÉSZÍTŐ(V_H, E_H) // I_2 a H FB-indexe.
3 $J = I_1 + I_2$ // Az FB-indexeket összekapcsoljuk a gyökerüknél.
4 $I =$ FB-INDEX-KÉSZÍTŐ(V_J, E_J) // I a J FB-indexe.
5 **return** I

Az (u, v) él hozzáadásánál most arra is figyelnünk kell, hogy a stabilitás mindkét irányban elromolhat, ezért nemcsak az $I(v)$-t vágjuk ketté {v}-re és $(I \setminus \{v\})$-re, hanem az $I(u)$-t is, $\{u\}$-ra és $(I(u) \setminus \{u\})$-ra. X legyen a módosítás előtti partíció, és Q a szétvágásokkal kapott partíció. Az FB-INDEX-KÉSZÍTŐ algoritmus 3. lépésében szereplő PT-algoritmus hívását az X és Q partíciókkal indítjuk. A stabilizáláskor most nemcsak a v utódjait, hanem az u elődjeit is be fogjuk járni.

ÉLHOZZÁADÁS-FB-INDEXE(G, (u, v))

1 $I =$ FB-INDEX-KÉSZÍTŐ(V_G, E_G) // I a G FB-indexe.
2 $G' = G + (u, v)$ // Behúzzuk az (u, v) élt.
3 **if** $(I(u), I(v)) \in E_I$ // Ha $I(u)$-ből $I(v)$-be mutatott él, akkor nem kell módosítani.
4 **return** I
5 $I_1 = \{v\}$ // Kettévágjuk az $I(v)$-t.
6 $I_2 = I(v) \setminus \{v\}$
7 $I_3 = \{u\}$ // Kettévágjuk az $I(u)$-t.
8 $I_4 = I(u) \setminus \{u\}$
9 $X = I$ // X a régi partíció.
10 $E_I = E_I \cup \{(I(u), I(v))\}$ // Behúzunk egy élt $I(u)$-ból $I(v)$-be.
11 $Q = (I \setminus \{I(v), I(u)\}) \cup \{I_1, I_2, I_3, I_4\}$ // $I(v)$-t I_1-re és I_2-re, $I(u)$-t I_3-ra és I_4-re cseréljük.
12 $E = E_Q$ // Meghatározzuk Q éleit.
13 $J =$ FB-INDEX-KÉSZÍTŐ$(V_{G'}, E_{G'}, X, Q)$ // Az FB-INDEX-KÉSZÍTŐ algoritmusban a PT-algoritmust X-szel és Q-val indítva futtatjuk le.
14 $J =$ FB-INDEX-KÉSZÍTŐ(V_J, E_J) // J a legdurvább elől-stabil és utód-stabil finomítás.
15 **return** J

Az A(k)-index frissítése az él hozzáadásakor eltér az eddigiektől. A gráf

hozzáadásával még nincs baj, mivel igaz a következő állítás, amelynek igazolását az Olvasóra bízzuk.

30.48. állítás. *Legyen a G gráf $A(k)$-indexe $I(G)$, és legyen J az $I(G)$ tetszőleges finomítása. Jelöljük $I(J)$-vel a J $A(k)$-indexét. Ekkor $I(J) = I(G)$.*

Az állítás következményeként diszjunkt G, H gráfok esetén a $G+H$ A(k)-indexét az alábbi algoritmussal lehet előállítani.

GRÁFHOZZÁADÁS-A(k)-INDEXE(G, H)

```
1  P_G = A_G(0)                                  // P_G a címkék szerinti alappartíció.
2  I_1 = NAIV-KÖZELÍTÉS(V_G, E_G^{-1}, P_G, k)   // I_1 a G A(k)-indexe.
3  P_H = A_H(0)                                  // P_H a címkék szerinti alappartíció.
4  I_2 = NAIV-KÖZELÍTÉS(V_H, E_H^{-1}, P_H, k)   // I_1 a H A(k)-indexe.
5  J = I_1 + I_2                                 // // Az A(k)-indexeket összekapcsoljuk.
6  P_J = A_J(0)                                  // P_J a címkék szerinti alappartíció.
7  I = NAIV-KÖZELÍTÉS(V_J, E_J^{-1}, P_J, k)     // // I a J A(k)-indexe.
8  return I
```

Ha egy (u, v) élt adunk a gráfhoz, akkor az eddigiek szerint $I(v)$-ből leválasztjuk v-t egy $I' = \{v\}$ tagba, ezután az elromló k-stabilitásokat kell kijavítani a v leszármazottjait bejárva, de csak k távolságnyira. A probléma abból adódik, hogy az A(k)-index csak a k-biszimulációra tartalmaz információt, $(k-1)$-re nem. Például legyen v-nek gyereke a v_1 és legyen $k = 1$. Az 1-index szerinti stabilizáláskor a v_1-et le kell választanunk az osztályából, ha ugyanebben az osztályban van olyan elem, amelynek v nem szülője. Ez a feltétel az A(1)-index esetén túl erős, és ezért túl sok fölösleges szétvágást okoz. Ebben az esetben ugyanis csak akkor kell v_1-et leválasztani, ha van ebben az osztályban olyan elem, amelynek nincs olyan szülője, amely 0-biszimuláns, azaz azonos címkéjű lenne a v-vel. Emiatt ha az A(k)-indexet az eddigiek szerint frissítenénk él hozzáadásakor, akkor nagyon rossz közelítését kapnánk a módosításhoz tartozó A(k)-indexnek, ezért nem ezt az eljárást használjuk. Az alapötlet, hogy nem csak az A(k)-indexet tartjuk nyilván, hanem az összes A(i)-indexet, ahol $i = 0, \ldots, k$. Yi és társai megadnak egy algoritmust, amely ezen az elven működik, és a módosításnak megfelelő A(k)-indexet állítja elő. A megadott algoritmusok kis változtatással élek törlésére is használhatók, 1-index és A(k)-index esetén is.

Gyakorlatok

30.8-1. Bizonyítsuk be a 30.47. állítást.

30.8-2. Adjunk meg algoritmust arra, hogyan módosítsuk az indexet, mikor egy élt törlünk az adatgráfból. Különböző típusú indexeket vizsgáljunk. Mi lesz az algoritmus költsége?
30.8-3. Adjunk algoritmusokat arra, hogyan frissítsük a D(k)-indexet, ha az adatgráfot módosítjuk.

Feladatok

30-1 Megszorításokra vonatkozó implikációs probléma

Legyen R, Q reguláris kifejezés, x, y két csúcs. Az $R(x, y)$ predikátum jelentse azt, hogy x-ből elérhető y egy R-re illeszkedő címkesorozattal. Jelölje $R \subseteq Q$ a $\forall x(R(\text{gyökér}, x) \to Q(\text{gyökér}, x))$ megszorítást. $R = Q$, ha mindkét irányú tartalmazás teljesül. Jelöljön C egy megszorításokból álló véges halmazt, és c egy megszorítást.

a. Bizonyítsuk be, hogy a $C \models c$ implikációs probléma eldöntése 2-EXPSPACE probléma.

b. Jelölje $R \subseteq Q@u$ a $\forall v(R(u, v) \to Q(u, v))$ megszorítást. Bizonyítsuk be, hogy erre az osztályra nézve az implikációs probléma nem eldönthető.

30-2 Fák transzformációs távolsága

A csúcscímkézett fák közti *transzformációs távolság* legyen a minimális száma annak, ahány elemi művelettel az egyik fa a másikká átalakítható. Három elemi műveletet használhatunk, új csúcs hozzáadását, csúcs törlését, és címke átnevezését.

a. Bizonyítsuk be, hogy a T, T' fák transzformációs távolsága kiszámítható $O(n_T n_{T'} d_T d_{T'})$ időben, $O(n_T n_{T'})$ tárköltséggel, ahol n_T a fa csúcsainak száma, d_T a fa mélysége.

b. Legyen S, S' két fa. Adjunk meg egy algoritmust, amely az összes olyan (T, T') párt előállítja, ahol T, illetve T' szimulálja az S, illetve S' gráfot, és T, T' transzformációs távolsága kisebb egy előre megadott n egész számnál. (Ezt a műveletet *közelítő összekapcsolásnak* nevezzük.)

30-3 Osztott adatbázisok lekérdezése

Az osztott adatbázis egy olyan csúcscímkézett, irányított gráf, amelynek csúcsait m darab partícióba (szerverre) osztottuk. A különböző partíciók közti élek a *kereszthivatkozások*. A kommunikáció a szerverek közti üzenetszórással történik. Egy lekérdezést kiértékelő algoritmus *hatékony,* ha a kommunikációs lépések száma konstans, vagyis független az adatoktól és a lekérdezéstől, valamint a kommunikáció során átvitt adatok összmérete csak a lekérdezésre

adott válasz méretétől és a kereszthivatkozások számától függ. Bizonyítsuk be, hogy osztott adatbázisok reguláris lekérdezésére megadható olyan hatékony algoritmus, amelyben a kommunikációs lépések száma 4, és az átvitt adatok mérete $O(n^2) + O(k)$, ahol n a lekérdezésre adott válasz mérete, és k a kereszthivatkozások száma. (*Útmutatás.* próbáljuk megfelelőképpen módosítani a NAIV-KIÉRTÉKELÉS algoritmust.)

Megjegyzések a fejezethez

A fejezet a félig strukturált adatbázisok világának azokat a területeit vizsgálta, amelyekben gráfok morfizmusait lehetett felhasználni. Így alapvetően a sémák, indexek készítését, használatát tárgyaltuk algoritmikus szemszögből. A félig strukturált adatbázisok, illetve az XML világa ennél jóval kiterjedtebb. A félig strukturált adatbázisok kialakulásának, aktuális témaköreinek, lehetséges további fejlődésének tömör összefoglalását adja Vianu [355] cikke.

A maximális szimuláció kiszámításával M. Henzinger, T. Henzinger és Kopke [181] cikke foglalkozik. Végtelen, de hatékonyan reprezentálható, úgynevezett effektív gráfokra is kiterjesztik a szimulációt, és belátják, hogy az ilyen gráfokra eldönthető, hogy két csúcs hasonló-e. Corneil és Gotlieb [83] cikke a hányados gráfokkal, és a gráfok izomorfizmusának eldöntésével foglalkozik. A relációs modellben használatos normálformákat terjeszti ki XML dokumentumokra Arenas és Libkin [22] cikke. Az általuk bevezetett XNF normálformáról kimutatják, hogy tetszőleges DTD átírható veszteségmentesen XNF normálformára.

Buneman, Fernandez és Suciu [64] tanulmányukban egy strukturális rekurzión alapuló lekérdező nyelvet, az UnQL-t vezetik be, ahol a használt adatmodell biszimulációval definiált. Gottlob, Koch és Pichler az XPath lekérdező nyelv osztályait vizsgálja bonyolultsági és párhuzamosíthatósági szempontból [154]. A bonyolultsági problémák áttekintéséhez Garey és Johnson klasszikus munkáját [142], valamint Stockmeyer és Meyer [331] cikkét ajánljuk.

A PT-algoritmus Paige és Tarjan [275] cikkében szerepel először. A biszimuláción alapuló 1-indexet Milo és Suciu részletesen tárgyalják [264], ezenkívül bevezetik a 2-indexet, és ennek általánosításaként a T-indexet.

Az A(k)-indexet Kaushik, Shenoy, Bohannon és Gudes [203] vezették be. A D(k)-index Chen, Lim és Ong [73] munkájában szerepelt először. A gyakori lekérdezéseken alapuló M(k)-index, illetve M*(k)-index He és Yang [179] eredménye. Az elágazó lekérdezésre vonatkozó FB-indexeket Kaushik, Bohannon, Naughton és Korth [205] vizsgálta először. Az 1-indexek, FB-indexek és A(k)-indexek módosítási algoritmusait Kaushik, Bohannon, Naughton és Shenoy

[206] foglalta össze. Az itt tárgyalt eljárásokat javítja és általánosítja Yi, He, Stanoi és Yang munkája [375]. A lekérdezések szelektivitásának vizsgálatára Polyzotis és Garafalakis valószínűségi modellt használ [290]. A strukturális indexek és az invertált listák együttes alkalmazhatóságát javasolja Kaushik, Krishnamurthy, Naughton és Ramakrishnan [204].

Az XML gyakorlati felhasználásával foglalkozó művek Tucker kézikönyve [349], valamint a Khosrow-Pour által szerkesztett enciklopédia [211].

Magyar nyelven az XML elméletének még nincs irodalma, viszont a gyakorlati felhasználással több könyv is foglalkozik [31, 60, 256].

31. Gyakori elemhalmazok keresése

Nagy adatbázisokból a hasznos, rejtett információ feltárása az adatbányászat fő célja. A hagyományos elemzések, mint az alapvető statisztikai értékek meghatározása, vagy a függetlenségvizsgálat, osztályozás, klaszterelemzés mellett egy új igény is megjelent, melynek során az adatbázisban gyakran előforduló elemeket, mintákat kell megkeresni. Egyszerű minták esetében a feladat általában könnyen és gyorsan megoldható megfelelő SQL lekérdezések segítségével. Bonyolult minták esetben a feladat nehezebb, mert vagy nem fogalmazható meg megfelelő lekérdezés, vagy a lekérdezés megválaszolása túl sok erőforrást igényel.

A legkutatottabb gyakori minta kinyerési terület a gyakori elemhalmazok meghatározása. Algoritmusai, fogalmai, ötletei többségét felhasználhatjuk más típusú gyakori minták keresésénél is. Ebben a fejezetben a gyakori elemhalmazok kinyerésének alapjait mutatjuk be.

A kutatási területet egy marketinges igény keltette életre. Óriási áruházakban a gyakran együtt vásárolt termékeket kellett meghatározni. Ezen információ alapján hatékonyabban lehet megszervezni a leárazásokat, akciókat, illetve kialakítani az áruház terméktérképét. A vásárlók viselkedése mellett vizsgálhatjuk a gyakori balesetet okozó helyzeteket, a számítógépes hálózatban gyakran előforduló, riasztással végződő eseménysorozatokat, vagy például azt, hogy az egyes nyomtatott médiumoknak milyen az olvasói összetétele. Amennyiben több magazinnak, újságnak hasonló a célcsoportja, érdemes üzenetünket több helyen is elhelyezni, hogy hatékonyabban ösztönözzük meglevő és potenciális vásárlóinkat. Oldalakon keresztül lehetne sorolni azon példákat, amikor a gyakran előforduló „dolgok" értékes információt rejtenek magukban. A szakirodalomban a dolgokat mintáknak nevezzük, és ***gyakori minták kinyeréséről*** beszélünk.

Vásárlói szokások felderítésénél gyakori *elemhalmazokat* keresünk, ahol a termékek felelnek meg a halmazok elemeinek. Utazásokkal kapcsolatos szokásoknál a gyakran igénybe vett, költséges szolgáltatások sorrendje is fontos, így itt gyakori *sorozatokat* keresünk. Telekommunikációs hálózatokban olyan feltételek (predikátumok) gyakori fennállását kutatjuk, amelyek gyakran eredményeznek riasztást. Ezeket a gyakori *Boole-formulákat* megvizsgálva kaphatjuk meg például a téves riasztások gyakori okait. A böngészési szokások alapján fejleszthetjük oldalaink struktúráját, linkjeit, így a láto-

gatók még gyorsabban és hatékonyabban találják meg a keresett információkat. A böngészés során egy weboldal bejárását *címkézett gyökeres fákkal* jellemezhetjük. Gyakori mintákat kinyerő algoritmusokat a rákkutatásban is alkalmaznak. Azt vizsgálják, hogy a rákkeltő anyagokban vannak-e gyakran előforduló molekula-struktúrák. Ezeket a struktúrákat *címkézett gráfokkal* írjuk le.

A példák arra utalnak, hogy a minta típusa sokféle lehet. Sejthetjük, hogy más technikákat kell majd alkalmazni pl. címkézett gráfok keresésénél, mint amikor csak egyszerű elemhalmazokat kell megtalálnunk. Terjedelmi korlátok miatt a jelen írásban csak elemhalmaz típusú mintákkal foglalkozunk. Bemutatunk három algoritmust, amelyek nemcsak azért tűnnek ki, mert a gyakori elemhalmazok kinyerésnek legmeghatározóbb „szereplői", hanem azért is, mert alapelveik alkalmazhatók, bármilyen típusú mintával van dolgunk. Végül ismertetünk egy sztochasztikus algoritmust, amely mintavételezésen alapszik, így nem biztos, hogy minden gyakori elemhalmazt megtalál, viszont rendkívül gyors.

31.1. Gyakori elemhalmazok keresése

Legyen $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ elemek halmaza és $\mathcal{T} = \langle t_1, \ldots, t_n \rangle$ az $\mathcal{I}$ hatványhalmaza felett értelmezett sorozat, azaz $t_j \subseteq \mathcal{I}$. A $\mathcal{T}$ sorozatot ***bemeneti sorozatnak*** hívjuk, amelynek t_j elemei a ***tranzakciók***. Az $I \subseteq \mathcal{I}$ elemhalmaz ***támogatottsága*** (jelölésben $supp(I)$) megegyezik azon tranzakciók számával, amelyeknek részhalmaza az I. Az I ***gyakori,*** ameny- nyiben támogatottsága nem kisebb egy előre megadott konstansnál, amelyet hagyományosan ***min-supp***-pal jelölünk, és ***támogatottsági küszöbnek*** hívunk. A gyakori elemhalmazok keresése során adott egy $\mathcal{I}$ elemhalmaz, $\mathcal{T}$ bemeneti sorozat, *min-supp* támogatottsági küszöb, feladatunk meghatározni a gyakori elemhalmazokat és azok támogatottságát.

Elterjedt, hogy a támogatottság helyett ***gyakoriságot,*** a támogatottsági küszöb helyett ***gyakorisági küszöböt*** használnak, melyeket $freq(I)$-vel, illetve ***min-freq***-kel jelölnek. Az I elemhalmaz gyakoriságán a $supp(I)/|\mathcal{T}|$ hányadost értjük.

A gyakorlatban előforduló adatbázisokban nem ritka, hogy az elemek száma $10^5 - 10^6$, a tranzakcióké pedig $10^9 - 10^{10}$. Elméletileg már az eredmény kiírása is az $\mathcal{I}$ elemszámában exponenciális lehet, hiszen előfordulhat, hogy $\mathcal{I}$ minden részhalmaza gyakori. A gyakorlatban a maximális méretű gyakori elemhalmaz mérete $|\mathcal{I}|$-nél jóval kisebb (legfeljebb 20-30). Ezen kívül minden tranzakció viszonylag kicsi, azaz $|t_j| \ll |\mathcal{I}|$. A keresési tér tehát nagy, ami azt jelenti, hogy az egyszerű nyers erő módszerek (határozzuk meg minden el-

emhalmaz támogatottságát, majd válogassuk ki a gyakoriakat) elfogadhatatlanul lassan futnának.

A példákban az elemeket $A, B, C, \ldots$ betűkkel fogjuk jelölni. Az egyszerűség kedvéért a halmazt jelölő kapcsos zárójeleket és az elemeket határoló vesszőket elhagyjuk, tehát például az $\langle\{A, C, D\}, \{B, C, E\}, \{A, B, C, E\}, \{B, E\}, \{A, B, C, E\}\rangle$ sorozatot $\langle ACD, BCE, ABCE, BE, ABCE\rangle$ formában írjuk.

31.1. példa. Legyen $\mathcal{T} = \langle ACD, BCE, ABCE, BE, ABCE\rangle$ bemeneti sorozat. Ekkor például $supp(\emptyset) = 5$, $supp(AC) = 3$, $supp(BE) = 4$. Amennyiben a támogatottsági küszöb 4, akkor a gyakori elemhalmazok a következők: $\emptyset$, B, C, E, BE.

Ábrázolhatjuk az adatbázist egy $|\mathcal{I}| \times |\mathcal{T}|$ méretű *bináris mátrixszal* is. A k-adik sor l-edik eleme 1, amennyiben a k-adik tranzakció tartalmazza az l-edik elemet, ellenkező esetben 0. A $|t_j| \ll |\mathcal{I}|$ feltevés miatt a mátrix ritka, így ez az adattárolási mód az esetek többségében pazarló.

tranzakció	elemhalmaz
1	C
2	A, B, C

(a) horizontális

elem	tranzakcióhalmaz
A	2
B	2
C	1,2

(b) vertikális

	A	B	C
1	0	0	1
2	1	1	1

(c) mátrixos

tranzakció	elem
1	C
2	A
2	B
2	C

(d) relációs

31.1. ábra. A négyféle tárolási mód.

Tudjuk, hogy egy tranzakcióban változó számú elem lehet (és fordítva: egy elem változó számú tranzakcióban szerepelhet). A legtöbb mai adatbázis ***relációs táblákat*** tárol, amelyekben csak rögzített számú attribútum szerepelhet. A valóságban ezért a tranzakciók két attribútummal rendelkező relációs tábla formájában találhatók, ahol az első attribútum a tranzakciót, a második pedig az elemet adja meg (pontosabban a tranzakciók és az elemek azonosítóit). A $\langle C, ABC\rangle$ sorozat négyféle tárolására mutatnak példát a 31.1.

táblázatok.

31.1.1. Asszociációs szabályok

A gyakori elemhalmazok kinyerése először az érvényes asszociációs szabályok meghatározásánál került elő. Adott bemeneti sorozat esetén és I, I' nem üres, diszjunkt elemhalmazok esetén az $R : I \xrightarrow{c,s,f} I'$ kifejezést c bizonyosságú, s támogatottságú f függetlenségi mutatójú ***asszociációs szabálynak*** nevezzük, ahol

$$c = \frac{supp(I \cup I')}{supp(I)} ,$$

$$s = supp(I \cup I') ,$$

$$f = \frac{freq(I \cup I')}{freq(I) \cdot freq(I')} = \frac{c}{freq(I)} .$$

Adott $\mathcal{T}$ bemeneti sorozat, *min-supp* támogatottsági, $min - conf$ bizonyossági, *min-lift* függetlenségi küszöb mellett az $I \xrightarrow{c,s,f} I'$ asszociációs szabály ***érvényes,*** ha $s \geq$ *min-supp*, $c \geq min - conf$ és $f \geq$ *min-lift*. Az asszociációs szabályok első alkalmazási területe a vásárlói szokások felderítése volt. Például az „azok akik sört vásárolnak, általában pelenkát is vesznek" állítást kifejezhetjük egy asszociációs szabállyal.

Az asszociációs szabályok kinyerésének feladatában adott egy bemeneti sorozat és három küszöbszám (*min-supp*, $min - conf$, *min-lift*). Feladatunk megtalálni az érvényes asszociációs szabályokat. A feladat hasonlít a gyakori elemhalmazok kinyeréséhez. Az $I \rightarrow I'$ szabály érvényességének feltétele, hogy az $I \cup I'$ gyakori elemhalmaz legyen, ahol a gyakorisághoz használt küszöb *min-supp*. Ha ismerjük a gyakori elemhalmazokat, akkor az asszociációs szabályokat könnyen megkaphatjuk: minden legalább kételemű gyakori elemhalmazt kettéosztunk feltétel- és következményrészre, majd ellenőrizzük, hogy az így kapott asszociációs szabály bizonyossága és függetlenségi mutatója nagyobb-e a megadott küszöböknél.

Egy asszociációs szabály azt fejezi ki, hogy azon tranzakciók c-szeresében, amelyben I megtalálható, az I' is megtalálható. A támogatottság adja meg, hogy ez a szabály mennyi tranzakcióban áll fenn. Az f értéke a függetlenséget próbálja megragadni. Amennyiben $f = 1$, akkor ez az jelenti, hogy ugyanolyan gyakran fordul elő I' az I-t tartalmazó tranzakciókban, mint általában a többi tranzakcióban (hiszen $f = 1$ ekvivalens azzal, hogy $freq(I') = freq(I \cup I')/freq(I)$), azaz I nincs hatással (független) az I' előfordulására. Az $f > 1$ egyfajta pozitív korrelációt jelent (I' gyakrabban fordul elő az I-t tartalmazó tranzakciókban, mint általában), $f < 1$ pedig negatívat.

Valójában a függetlenségi mutató nem ragadja meg kellőképpen a két

esemény (I és I' előfordulása) statisztikai függetlenségét. Tudjuk, hogy az I, I' események függetlenek, ha $p(I)p(I') = p(I, I')$, amelyet átírhatunk $1 = p(I'|I)/p(I)$ alakra. A jobb oldal annyiban tér el a függetlenségi mutatótól, hogy abban a valószínűségek helyén relatív gyakoriságok szerepelnek. Pusztán a relatív gyakoriságok hányadosa nem elég jó mérték a függetlenség mérésére. Nézzünk például a következő két esetet. Első esetben négy tranzakció van, $supp(I) = 2$, $c = 0.5$, amiből $f = 1$. A másodikban a tranzakciók száma négyezer, $supp(I) = 1992$, $c = 0.504$, amiből $f = 1.012$. Ha csak a függetlenségi mutatókat ismernénk, akkor azt a téves következtetést vonhatnánk le, hogy az első esetben a két esemény függetlenebb, mint a második esetben. Holott érezzük, hogy az első esetben olyan kevés a tranzakció, hogy abból nem tudunk függetlenségre vonatkozó következtetéseket levonni. Minél több tranzakció alapján állítjuk, hogy két elemhalmaz előfordulása összefüggésben van, annál jobban kizárjuk ezen állításunk véletlenségének (esetlegességének) esélyét.

A függetlenség mérése alkalmasabb az ún. χ^2 próbastatisztika. Az $A_1, A_2, \ldots, A_r$ és $B_1, B_2, \ldots, B_s$ két teljes eseményrendszer χ^2 próbastatisztikáját az alábbi képlet adja meg:

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{s} \frac{\left(k_{ij} - \frac{k_{i.}k_{.j}}{n}\right)^2}{\frac{k_{i.}k_{.j}}{n}},$$

ahol k_{ij} az $A_i \cap B_j$ esemény, $k_{i.} = \sum_{j=1}^{s} k_{ij}$ az A_i esemény és $k_{.j} = \sum_{i=1}^{r} k_{ij}$ a B_j esemény bekövetkezésének számát jelöli. Minél kisebb a próbastatisztika, annál inkább függetlenek az események.

A mi esetünkben az egyik eseményrendszer az I elemhalmaz a másik az I' elemhalmaz előfordulásához tartozik, és mindkét eseményrendszernek két eseménye van[1] (előfordul az elemhalmaz az adott tranzakcióban, vagy sem). A következő táblázat mutatja, hogy a χ^2 próbastatisztika kiszámításához szükséges értékek közül melyek állnak rendelkezésünkre támogatottság formájában.

	I	nem I	$\sum$
I'	$supp(I \cup I')$		$supp(\mathrm{I'})$
nem I'			
$\sum$	$supp(\mathrm{I})$		$\lvert\mathcal{T}\rvert$

A hiányzó értékeket a táblázat ismert értékei alapján könnyű pótolni lehet,

[1] Amennyiben mindkét eseményrendszer két eseményből áll, akkor az eredeti képletet módosítani szokás a Yates-féle korrekciós együtthatóval, azaz $\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{2} \left(\left|k_{ij} - \frac{k_{i.}k_{.j}}{n}\right| - \frac{1}{2}\right)^2 / \frac{k_{i.}k_{.j}}{n}$.

hiszen például $k_{2,1} = supp(I) - supp(I \cup I')$.

A gyakori elemhalmazok első alkalmazásának bemutatása után térjünk rá arra, hogy miként tudjuk a gyakori elemhalmazokat hatékonyan meghatározni.

Gyakorlatok

31.1-1. Legyen $I = I' \cup i'_1 \cup i'_2 \cdots \cup i'_j$, ahol minden i'_k $(1 \leq k \leq j)$ az I elemhalmaz egy eleme. Mutassuk meg, hogy

$$supp(I) \geq supp(I' \cup i'_1) + supp(I' \cup i'_2) + \cdots + supp(I' \cup i'_j) - (j-1)supp(I') \; .$$

31.1-2. Mennyi az érvényes asszociációs szabályok maximális száma, ha az elemek száma n?

31.2. Gyakori elemhalmazokat kinyerő algoritmusok

A keresési teret úgy képzelhetjük el, mint egy irányított gráfot, amelynek csúcsai az elemhalmazok, és az I_1-ből él indul I_2-be, amennyiben $I_1 \subset I_2$, és $|I_1| + 1 = |I_2|$. A keresési tér bejárásán mindig ezen gráf egy részének bejárását fogjuk érteni. Tehát például a keresési tér szélességi bejárása ezen gráf szélességi bejárását jelenti.

A következőkben bemutatjuk a három leghíresebb gyakori elemhalmazokat kinyerő (GYEK) algoritmust. Mindhárman az üres mintából indulnak ki. Az algoritmusok egy adott fázisában ***jelöltnek*** hívjuk azokat az elemhalmazokat, amelyek támogatottságát meg akarjuk határozni. Az algoritmus akkor ***teljes,*** ha minden gyakori elemhalmazt megtalál és ***helyes,*** ha csak a gyakoriakat találja meg.

Mindhárom algoritmus három lépést ismétel. Először jelölteket állítanak elő, majd meghatározzák a jelöltek támogatottságát, végül kiválogatják a jelöltek közül a gyakoriakat. Természetesen az egyes algoritmusok különböző módon járják be a keresési teret (az összes lehetséges elemhalmazt), állítják elő a jelölteket, és különböző módon határozzák meg a támogatottságokat.

Az általánosság megsértése nélkül feltehetjük, hogy az $\mathcal{I}$ elemein tudunk definiálni egy teljes rendezést, és a jelöltek, illetve a tranzakciók elemeit ezen rendezés szerint tároljuk. Más szóval az elemhalmazokat sorozatokká alakítjuk. Egy sorozat ℓ-elemű *prefixén* a sorozat első ℓ eleméből képzett részsorozatát értjük. A példákban majd, amennyiben a rendezésre nem térünk ki külön, az ábécé szerinti sorrendet használjuk. A GYEK algoritmusok általában érzékenyek a használt rendezésre. Ezért minden algoritmusnál megvizsgáljuk, hogy milyen rendezést célszerű használni annak érdekében, hogy a futási idő, vagy a memóriaszükséglet a lehető legkisebb legyen.

A jelölt-előállítás ***ismétlés nélküli,*** amennyiben nem állítja elő ugyanazt a jelöltet többféle módon. Ez a hatékonyság miatt fontos, ugyanis ismétléses jelölt-előállítás esetében minden jelölt előállítása után ellenőrizni kellene, hogy nem állítottuk-e elő már korábban. Ha ezt nem tesszük, akkor feleslegesen kötünk le erőforrásokat a támogatottság ismételt meghatározásánál. Mindhárom ismertetett algoritmusban a jelöltek előállítása ismétlés nélküli lesz, amit a rendezéssel tudunk garantálni.

Az algoritmusok pszeudokódjaiban GY-vel jelöljük a gyakori elemhalmazok halmazát, J-vel jelöltekét és $j.számláló$-val a j jelölt számlálóját. Az olvashatóbb kódok érdekében feltesszük, hogy minden számláló kezdeti értéke nulla, és az olyan halmazok, amelyeknek nem adunk kezdeti értéket (például GY), nem tartalmaznak kezdetben egyetlen elemet sem.

31.2.1. Az Apriori algoritmus

Az Apriori algoritmus az egyik legelső GYEK algoritmus. Szélességi bejárást valósít meg, ami azt jelenti, hogy a legkisebb mintából (ami az üres halmaz) kiindulva szintenként halad előre a nagyobb méretű gyakori elemhalmazok meghatározásához. A következő szinten (iterációban) az eggyel nagyobb méretű elemhalmazokkal foglalkozik. Az iterációk száma legfeljebb eggyel több, mint a legnagyobb gyakori elemhalmaz mérete.

A jelöltek definiálásánál a következő egyszerű tényt használja fel: *Gyakori elemhalmaz minden részhalmaza gyakori.* Az állítást indirekten nézve elmondhatjuk, hogy egy elemhalmaz biztosan nem gyakori, ha van ritka részhalmaza. Ennek alapján ne legyen jelölt azon elemhalmaz, amelynek van ritka részhalmaza. Az Apriori algoritmus ezért építkezik lentről. Egy adott iterációban csak olyan jelöltet veszünk fel, amelynek összes valódi részhalmazáról tudjuk, hogy gyakori. Az algoritmus onnan kapta a nevét, hogy az ℓ-elemű jelölteket a bemeneti sorozat ℓ-edik átolvasásának megkezdése előtt (a priori) állítja elő. Az ℓ-elemű jelöltek halmazát J_ℓ-lel, az ℓ-elemű gyakori elemhalmazokat pedig GY_ℓ-lel jelöljük.

Apriori($\mathcal{T}$, *min-supp*)

```
1  ℓ = 0
2  J_ℓ = {∅}
3  while |J_ℓ| ≠ 0
4        for minden t ∈ T ,            // Támogatottságok meghatározása.
5            for minden j ∈ J_ℓ, j ⊆ t
6                j.számláló = j.számláló +1
```

```
7        for minden j ∈ J_ℓ                    // Gyakori jelöltek kiválogatása.
8            if j.számláló ≥ min-supp
9                GY_ℓ = GY_ℓ ∪ {j}
10       GY = GY ∪ GY_ℓ
11       J_{ℓ+1} = Jelölt-előállítás(GY_ℓ)
12       ℓ = ℓ + 1
13   return GY
```

A kezdeti értékek beállítása után egy ciklus következik, amely akkor ér véget, ha nincsen egyetlen ℓ-elemű jelölt sem. A cikluson belül először meghatározzuk a jelöltek támogatottságát. Ehhez egyesével vesszük a tranzakciókat (4. sor), és azon jelöltek számlálóját növeljük eggyel, amelyeket tartalmaz a vizsgált tranzakció (5–6. sorok). Ha rendelkezésre állnak a támogatottságok, akkor a jelöltek közül kiválogathatjuk a gyakoriakat (7–9. sorok). A Jelölt-előállítás függvény az ℓ-elemű gyakori elemhalmazokból $(\ell+1)$-elemű jelölteket állít elő. Azok és csak azok az elemhalmazok lesznek jelöltek, amelyek minden részhalmaza gyakori.

A jelöltek előállítása során olyan ℓ-elemű, gyakori I_1, I_2 elemhalmaz párokat keresünk, amelyekre igaz, hogy

- I_1 lexikografikusan megelőzi I_2-t,
- I_1-ből a legnagyobb elem törlésével ugyanazt az elemhalmazt kapjuk, mintha az I_2-ből törölnénk a legnagyobb elemet.

Ha a feltételeknek megfelelő párt találunk, akkor képezzük a pár unióját, majd ellenőrizzük, hogy a kapott elemhalmaznak minden valódi részhalmaza gyakori-e. A támogatottság anti-monotonitása miatt szükségtelen az összes valódi részhalmazt megvizsgálni; ha mind az $\ell+1$ darab ℓ-elemű részhalmaz gyakori, akkor az összes valódi részhalmaz is gyakori. Az I_1, I_2 halmazokat a jelölt ***generátorainak*** szokás hívni.

31.2. példa. Legyenek a 3-elemű gyakori elemhalmazok a következők: $GY_3 = \{ABC, ABD, ACD, ACE, BCD\}$. Az ABC és ABD elemhalmazok megfelelnek a feltételnek, ezért képezzük az uniójukat. Mivel $ABCD$ minden háromelemű részhalmaza a GY_3-nak is eleme, az $ABCD$ jelölt lesz. Az ACD, ACE pár is megfelel a két feltételnek, de uniójuknak van olyan részhalmaza (ADE), amely nem gyakori. Az Apriori a következő iterációban tehát már csak egyetlen jelölt támogatottságát határozza meg.

A fenti módszer csak akkor alkalmazható, ha $\ell > 0$. Az egyelemű jelöltek előállítása egyszerű: minden egyelemű halmaz jelölt, amennyiben az üres elemhalmaz gyakori ($|\mathcal{T}| \geq$ *min-supp*). Ez összhangban áll azzal, hogy akkor

lehet egy elemhalmaz jelölt, ha minden részhalmaza gyakori.

Vizsgáljuk meg részletesebben az 5. sort. Adott egy tranzakció és ℓ-méretű jelöltek egy halmaza. Feladatunk meghatározni azon jelölteket, amelyek a tranzakció részhalmazai. Megoldhatjuk ezt egyszerűen úgy, hogy a jelölteket egyesével vesszük, és eldöntjük, hogy tartalmazza-e őket a tranzakció. Rendezett halmazban rendezett részhalmaz keresése elemi feladat. Ennek az egyszerű módszernek a hátránya, hogy sok jelölt esetén lassú, hiszen annyiszor kell a tranzakció elemein végighaladni, amennyi a jelöltek száma. A gyorsabb működés érdekében a jelölteket szófában célszerű tárolni.

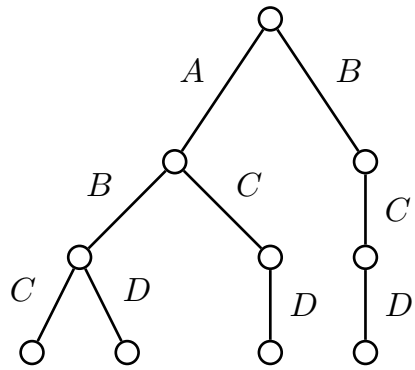
A ***szófa*** éleinek címkéi elemek lesznek. Minden csúcs egy elemhalmazt reprezentál, amelynek elemei a gyökérből a csúcsig vezető út éleinek címkéivel egyeznek meg. Feltehetjük, hogy az egy csúcsból induló élek, továbbá az egy úton található élek címkék szerint rendezve vannak (pl. legnagyobb elem az első helyen). A jelöltek számlálóit a jelöltet reprezentáló levélhez rendeljük. A 31.2. ábrán egy szófát láthatunk.

A t tranzakcióban az ℓ-elemű jelölteket úgy találjuk meg, hogy a jelölteket leíró fa gyökeréből kiindulva, rekurzív módon bejárunk bizonyos részfákat. Ha egy d szintű belső csúcshoz a tranzakció j-edik elemén keresztül jutunk, akkor azon élein keresztül lépünk eggyel mélyebb szintre, amelyeknek címkéje megegyezik a tranzakció j'-edik elemével, ahol $j < j' \leq |t| - \ell + d$ (ugyanis $\ell - d$ elemre még szükség van ahhoz, hogy levélbe érjünk). Ha ily módon eljutunk egy ℓ szintű csúcshoz, az azt jelenti, hogy a csúcs által reprezentált elemhalmazt tartalmazza t, így ennek a levélnek a számlálóját kell növelnünk eggyel.

A szófát prefix fának is szokták hívni, ami arra utal, hogy a közös prefixeket csak egyszer tárolja. Ettől lesz gyorsabb a szófás támogatottság-meghatározás a naiv módszernél. A közös prefixeket összevonjuk, és csak egyszer foglalkozunk velük.

A szófa nagy előnye a gyors támogatottság-meghatározás mellett, hogy a jelölt-előállítást is támogatja. Tudjuk, hogy két gyakori elemhalmaz akkor lesz generátor, ha a legnagyobb sorszámú elemük elhagyásával ugyanazt az elemhalmazt kapjuk, vagy más szavakkal, a két gyakori elemhalmaz $\ell - 1$ hosszú prefixei megegyeznek. A támogatottság-meghatározásában használt szófát felhasználhatjuk a következő iterációs lépés jelöltjeinek az előállítására, hiszen a szófa tárolja a jelölt-előállításhoz szükséges gyakori elemhalmazokat.

Az egész algoritmus alatt tehát egyetlen szófát tartunk karban, amely az algoritmus kezdetekor csak egy csúcsból áll (ez reprezentálja az üres halmazt). A támogatottság-meghatározás után töröljük azon leveleket, amelyek számlálója kisebb *min-supp*-nál. Az iterációs lépés végére kialakuló szófa alapján előállítjuk a jelölteket, amely során a szófa új, eggyel mélyebb szinten lévő



31.2. ábra. Az *ABC*, *ABD*, *ACD*, *BCD* jelölteket tároló szófa.

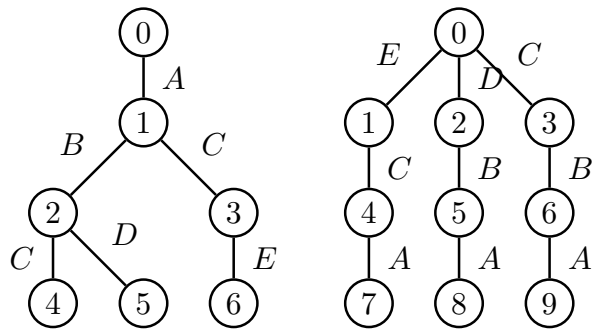
levelekkel bővül. A jelölt-előállítás során arra is lehetőségünk van, hogy az előző iterációban gyakorinak talált elemhalmazokat és azok számlálóit kiírjuk (a kimenetre vagy a háttértárolóra).

Szükségtelen tárolni azon csúcsokat, amelyekből az összes elérhető levelet töröltük. Ezek ugyanis lassítják a támogatottságok meghatározását (miközben szerepet nem játszanak benne) és feleslegesen foglalják a memóriát.

Gyorsíthatjuk az algoritmust, ha a bemeneti sorozatot nem az eredeti formájában dolgozzuk fel. Általában az adatbázis speciálisan formázott fájlban a háttértáron található. Az operációs rendszer a fájl egy blokkját bemásolja a memóriába és amikor használni akarjuk a tranzakciót, akkor át kell alakítani a feldolgozáshoz megfelelő formátumba (például egész értékeket tartalmazó vektorrá). Ha van elég hely a memóriában, akkor a tranzakció bent marad a memóriában, I/O művelet nem történik a következő iterációban, de az átalakítást ismét végre kell hajtani. Ezt az átalakítást megtakaríthatjuk a bemenet explicit tárolásával.

Sajnos a bemenet általában olyan nagy, hogy azt az eredeti formájában nem tudjuk tárolni. Erre nincs is szükség. Csökkenthetjük a memóriaigényt, ha csak a tranzakciók gyakori elemeit tároljuk (megszűrjük a tranzakciót), továbbá, az azonos tranzakciókat egyszer tároljuk, és egy számlálót rendelünk hozzájuk, amely megadja a tranzakció multiplicitását.

A szűrt tranzakciókat célszerű olyan adatstruktúrában tárolni, amelyet gyorsan fel lehet építeni (azaz gyorsan tudjuk beszúrni a szűrt tranzakciókat) és gyorsan végig tudunk menni a beszúrt elemeken. Alkalmazhatunk erre a



31.3. ábra. Példa: különböző rendezést használó, ugyanazokat az elemhalmazokat tároló szófák.

célra egy szófát, vagy egy piros-fekete fát, amelynek csúcsaiban egy-egy szűrt tranzakció található.

Vizsgáljuk meg, hogy az APRIORI mennyire érzékeny az elemeken definiált rendezésre. A jelöltek előállításánál a rendezést csak azért használtuk fel, hogy a jelölt-előállítás ismétlés nélküli legyen. Itt tetszőleges rendezést használhatunk, a kimenet (a jelöltek halmaza) független a rendezéstől. A jelölteket tároló szófa szerkezete azonban már nagyban függ a rendezéstől. Ezt a 31.3. ábrával is szemléltethetjük, ahol két olyan szófát láthatunk, amelyek a ABC, ABD, ACE elemhalmazokat tárolják. Az első szófa az ABC szerint csökkenő sorrendet használja ($C \prec B \prec A$), míg a második ennek ellenkezőjét.

Memóriaigény szempontjából az lenne a legkedvezőbb, ha azt a rendezést használnánk, amelyik a legkevesebb csúcsú szófát adná, ugyanis a szófa mérete egyenesen arányos a csúcsainak számával. Sajnos ennek meghatározása nehéz feladat.

31.1. tétel. *Legyen $\mathcal{I}$ egy elemhalmaz, $J \subseteq 2^{\mathcal{I}}$ és k pozitív egész szám. NP-teljes annak eldöntése, hogy van-e olyan rendezése $\mathcal{I}$-nek, amely esetén a J-t tartalmazó szófa csúcsainak száma nem több k-nál.*

A fentiek szerint a minimális méretű szófa meghatározása NP-nehéz feladat. Ennek ellenére egy nagyon egyszerű heurisztika a gyakorlatban rendkívül jól működik. Tudjuk, hogy a szófa az azonos prefixeket egyszer tárolja, azaz minél több a közös prefix, annál kisebb a szófa mérete. Akkor van a legnagyobb esélye annak, hogy két halmaznak közös a prefixe, ha a halmazt úgy alakítjuk át sorozattá, hogy a sorozat elején a leggyakoribb elemek szerepelnek, vagy más szavakkal a választott rendezés a gyakoriság szerint csökkenő

rendezéssel egyezik meg. Tulajdonképpen ennek a heurisztikának a helyességét sejteti az előző ábra is.

Amennyiben nem a kis memóriaigény a fontos, hanem a gyors futás, akkor a fenti heurisztika ellenkezőjét célszerű használni, azaz a teljes rendezés a gyakoriság szerint növekvő rendezéssel egyezik meg. Ebben az esetben a gyökérhez közeli éleken lesznek a ritkább elemek és a levelekhez közel a leggyakoribbak. Ez azt jelenti, hogy a támogatottság-meghatározás során először azt ellenőrizzük, hogy a tranzakció tartalmazza-e a jelölt ritkább elemeit. Ezen a teszten jóval kevesebb jelölt fog átmenni, mintha a gyakori elemek vizsgálatával kezdenénk. Jól szemlélteti ezt az előző ábra, amennyiben a vizsgált tranzakció az $\{ABFGH\}$ halmaz. Gyakoriság szerint csökkenő esetben a 0,1,2 sorszámú csomópontokat fogjuk bejárni a támogatott jelöltek keresés során, míg ellenkező esetben a támogatottság meghatározása már a gyökérben véget ér. A gyors futás érdekében több memóriát használunk.

Futási idő és memóriaigény

A GYEK feladat megadásakor elmondtuk, hogy már az eredmény kiírása – ami a futási időnek a része – az $|I|$-ben exponenciális lehet. A memóriaigényről is hasonló mondható el. Az $(\ell+1)$-elemű jelöltek előállításához szükségünk van az összes ℓ-elemű jelöltre, amelyek száma akár $\binom{|I|}{|I|/2}$ is lehet. Ezek a felső korlátok élesek is, hiszen $min - supp = 0$-nál minden elemhalmaz gyakori.

Az algoritmus indítása előtt tehát nem sokat tudunk mondani a futási időről. A futás során, azonban egyre több információt gyűjtünk, így felmerül a kérdés, hogy ezt fel tudjuk-e használni az algoritmus maradék futási idejének jóslására. Például, ha a gyakori elemek száma négy, akkor tudjuk, hogy a legnagyobb gyakori elemhalmaz mérete legfeljebb négy (azaz még legfeljebb háromszor olvassuk végig az adatbázist), az összes jelölt maximális száma pedig $\binom{4}{2}+\binom{4}{3}+\binom{4}{4}=11$. A következőkben megvizsgáljuk, hogy mit tudunk elmondani a jelöltek számáról és a maximális jelöltek méretéről, ha adottak az ℓ-elemű gyakori elemhalmazok (GY_ℓ).

A következő rész fontos fogalma a kanonikus reprezentáció lesz.

31.2. lemma. *Adott n és ℓ pozitív egészek esetében a következő felírás egyértelmű:*

$$n = \binom{m_\ell}{\ell} + \binom{m_{\ell-1}}{\ell-1} + \cdots + \binom{m_r}{r},$$

ahol $r \geq 1$, $m_\ell > m_{\ell-1} > \cdots > m_r$ és $m_j \geq j$ minden $j = r, r+1, \ldots, \ell$ számra.

Ezt a reprezentációt hívják ℓ-kanonikus reprezentációnak. Meghatározása nagyon egyszerű: m_ℓ-nek ki kell elégítenie a $\binom{m_\ell}{\ell} \leq n < \binom{m_\ell+1}{\ell}$ feltételt, $m_{\ell-1}$-nek a $\binom{m_{\ell-1}}{\ell-1} \leq n - \binom{m_\ell}{\ell} < \binom{m_{\ell-1}+1}{\ell-1}$ feltételt, és így tovább, amíg

$n - \binom{m_\ell}{\ell} - \binom{m_{\ell-1}}{\ell-1} - \cdots - \binom{m_r}{r}$ nulla nem lesz.

Legyen $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ elemek halmaza és GY_ℓ egy olyan $\mathcal{I}$ feletti halmazcsalád,[2] amelynek minden eleme ℓ-elemű. Az ℓ-nél nagyobb méretű $I \subseteq \mathcal{I}$ halmaz ***fedi*** a GY_ℓ-et, ha I minden ℓ-elemű részhalmaza eleme GY_ℓ-nek. Az összes lehetséges $(\ell+p)$-méretű GY_ℓ-et fedő halmazokból alkotott halmazcsaládot $J_{\ell+p}(GY_\ell)$-lel jelöljük. Nem véletlen, hogy ezt a halmazt ugyanúgy jelöltük, mint az Apriori algoritmus jelöltjeit, ugyanis az $(\ell+p)$-méretű jelöltek ezen halmazcsaládnak az elemei, és ha az algoritmus során minden jelölt gyakori, akkor az $(\ell+p)$-méretű jelöltek halmaza megegyezik $J_{\ell+p}(GY_\ell)$-lel.

A következő tétel megadja, hogy adott GY_ℓ esetén legfeljebb mennyi lehet a $J_{\ell+p}(GY_\ell)$ elemeinek száma.

31.3. tétel. *Ha*

$$|GY_\ell| = \binom{m_\ell}{\ell} + \binom{m_{\ell-1}}{\ell-1} + \cdots + \binom{m_r}{r}$$

ℓ-kanonikus reprezentáció, akkor

$$|J_{\ell+p}(GY_\ell)| \leq \binom{m_\ell}{\ell+p} + \binom{m_{\ell-1}}{\ell-1+p} + \cdots + \binom{m_s}{s+p},$$

ahol s a legkisebb olyan egész, amelyre $m_s < s+p$. Ha nincs ilyen egész szám, akkor $s = r-1$.

A fenti tétel a Kruskal–Katona tétel következménye, ezért a tételben szereplő felső korlátot a továbbiakban $KK_\ell^{\ell+p}(|GY_\ell|)$-el jelöljük.

31.4. tétel. *A 31.3. tételben szereplő felső korlát éles, azaz adott n, ℓ, p számokhoz mindig létezik GY_ℓ, amelyre $|GY_\ell| = n$, és $|J_{\ell+p}(GY_\ell)| = KK_\ell^{\ell+p}(|GY_\ell|)$.*

A kanonikus reprezentáció segítségével egyszerű éles felső becslést tudunk adni a legnagyobb jelölt méretére (jelölésben $maxsize(GY_\ell)$) is. Tudjuk, hogy $|GY_\ell| < \binom{m_\ell+1}{\ell}$, ami azt jelenti, hogy nem létezhet olyan jelölt, amelynek mérete nagyobb m_ℓ-nél.

31.5. következmény. *Amennyiben a $|GY_\ell|$ számnak az ℓ-kanonikus reprezentációjában szereplő első tag $\binom{m_\ell}{\ell}$, akkor $maxsize(GY_\ell) \leq m_\ell$.*

Az m_ℓ számot a továbbiakban $\mu_\ell(|GY_\ell|)$-el jelöljük. Ez az érték azt is megmondja, hogy mekkora jelöltméretnél válik nullává a felső korlát, azaz:

31.6. következmény. $\mu_\ell(|GY_\ell|) = \ell + \min\{p \mid KK_\ell^{\ell+p}(|GY_\ell|) = 0\} - 1$

[2] A H-t az $\mathcal{I}$ feletti ***halmazcsaládnak*** nevezzük, amennyiben $H \subseteq 2^{\mathcal{I}}$.

A maradék futási idő jóslására a következő állítás nyújt segítséget.

31.7. következmény. *Az összes lehetséges ℓ-nél nagyobb méretű jelölt száma legfeljebb*

$$KK_\ell^{\text{összes}}(|GY_\ell|) = \sum_{p=1}^{\mu_\ell(|GY_\ell|)} KK_\ell^{\ell+p}(|GY_\ell|) \ .$$

A fenti korlátok szépek és egyszerűek, mivel csak két paramétert használnak: az ℓ aktuális méretet és az ℓ-elemű gyakori elemhalmazok számát ($|GY_\ell|$). Ennél jóval többet tudunk. Nem csak a gyakori elemhalmazok számát ismerjük, hanem már pontosan meghatároztuk őket magukat is! Az új információ segítségével számos esetben jobb felső korlátot adhatunk. Például, ha a GY_ℓ-ben csak páronként diszjunkt elemhalmazok vannak, akkor nem állítunk elő jelölteket. A 31.3. tételben szereplő felső korlát azonban jóval nagyobb lehet nullánál. A következőkben bemutatjuk, hogyan lehet a meglévő felső korlátot az ℓ méretű gyakori elemhalmazok *struktúrájára* rekurzívan alkalmazni. Ehhez feltesszük, hogy egy teljes rendezést tudunk definiálni az $\mathcal{I}$ elemein, ami alapján tetszőleges elemhalmaznak meg tudjuk határozni a legkisebb elemét. Vezessük be a következő két jelölést:

$$GY_\ell^i = \{I - \{i\} \mid I \in GY_\ell, i = \min I\} \ ,$$

A GY_ℓ^i halmazt úgy kapjuk GY_ℓ-ből, hogy vesszük azon halmazokat, amelyek legkisebb eleme i, majd töröljük ezekből az i elemet.

Ezek után definiálhatjuk a következő rekurzív függvényt tetszőleges $p > 0$-ra:

$$KK_{\ell,p}^*(GY_\ell) = \begin{cases} \binom{|GY_\ell|}{p+1}, & \text{ha } \ell = 1, \\ \min\{KK_\ell^{\ell+p}(|GY_\ell|), \sum_{i\in\mathcal{I}} KK_{\ell-1,p}^*(GY_\ell^i)\}, & \text{ha } \ell > 1 \ . \end{cases}$$

A definícióból következik, hogy $KK_{\ell,p}^*(GY_\ell) \leq KK_\ell^{\ell+p}(|GY_\ell|)$, továbbá

31.8. tétel. $|J_{\ell+p}(GY_\ell)| \leq KK_{\ell,p}^*(GY_\ell)$.

Bizonyítás. A bizonyítás teljes indukción alapul, az $\ell = 1$ eset triviális. Tulajdonképpen csak az kell belátni, hogy

$$|J_{\ell+p}(GY_\ell)| \leq \sum_{i\in\mathcal{I}} KK_{\ell-1,p}^*(GY_\ell^i) \ .$$

Az egyszerűség kedvéért vezessük be a következő jelölést: $H \cup i = \{I \cup \{i\} \mid I \in H\}$, ahol H egy $\mathcal{I}$ feletti halmazcsalád. Vegyük észre, hogy $GY_\ell = \sum_{i\in\mathcal{I}} GY_\ell^i \cup$

i és $GY_\ell^i \cap GY_\ell^j = \emptyset$ minden $i \neq j$ elempárra. Azaz a GY_ℓ halmazcsalád egy partícióját képeztük.

Amennyiben $I \in J_{\ell+p}(GY_\ell)$, és I-nek legkisebb eleme i, akkor $I \setminus \{i\} \in J_{\ell-1+p}(GY_\ell^i)$, hiszen $I \setminus \{i\}$ minden $(\ell-1)$-elemű részhalmaza GY_ℓ^i-beli. Ebből következik, hogy

$$J_{\ell+p}(GY_\ell) \subseteq \bigcup_{i \in \mathcal{I}} J_{\ell-1+p}(GY_\ell^i) \cup i \ .$$

Abból, hogy az GY_ℓ^i halmazcsaládok páronként diszjunktak következik, hogy $J_{\ell-1+p}(GY_\ell^i) \cup i$ is páronként diszjunkt halmazcsaládok. Ebből következik az állítás, hiszen:

$$\begin{aligned} |J_{\ell+p}(GY_\ell)| &\leq |\bigcup_{i \in \mathcal{I}} J_{\ell-1+p}(GY_\ell^i) \cup i| \\ &= \sum_{i \in \mathcal{I}} |J_{\ell-1+p}(GY_\ell^i) \cup i| \\ &= \sum_{i \in \mathcal{I}} |J_{\ell-1+p}(GY_\ell^i)| \\ &\leq \sum_{i \in \mathcal{I}} KK^*_{\ell-1,p}(GY_\ell^i) \ , \end{aligned}$$

ahol az utolsó egyenlőtlenségnél az indukciós feltevést használtuk. ■

A páronként diszjunkt halmazok esete jó példa arra, hogy a minimum kifejezésben szereplő második tag kisebb lehet az elsőnél. Előfordulhat azonban az ellenkező eset is. Például legyen $GY_2 = \{AB, AC\}$. Könnyű ellenőrizni, hogy $KK_2^3(|GY_2|) = 0$, ugyanakkor a második tagban szereplő összeg 1-et ad. Nem tudhatjuk, hogy melyik érték a kisebb, így jogos a két érték minimumát venni.

Javíthatjuk a legnagyobb jelölt méretére, illetve az összes jelölt számára vonatkozó felső korlátokat is. Legyen $\mu_\ell^*(GY_\ell) = \ell + \min\{p | KK^*_{\ell+p}(GY_\ell) = 0\} - 1$ és

$$KK^*_{\text{összes}}(GY_\ell) = \sum_{p=1}^{\mu_\ell^*(GY_\ell)} KK^*_{\ell+p}(GY_\ell) \ .$$

31.9. következmény. *$maxsize(GY_\ell) \leq \mu_\ell^*(GY_\ell) \leq \mu_\ell(|GY_\ell|)$.*

31.10. következmény. *Az összes lehetséges ℓ-nél nagyobb méretű jelölt száma legfeljebb $KK^*_{összes}(GY_\ell)$ lehet, és $KK^*_{összes}(GY_\ell) \leq KK_\ell^{összes}(|GY_\ell|)$.*

A KK^* érték függ a rendezéstől. Például a $KK^*_{2,1}(\{AB, AC\})$ értéke 1, amennyiben a rendezés szerinti legkisebb elem A, és 0 bármely más esetben. Elméletileg meghatározhatjuk az összes rendezés szerinti felső korlátot,

és kiválaszthatjuk azt, amelyik a legkisebb értéket adja. Ez a megoldás azonban túl sok időbe telne. A szófa által használt rendezés szerinti felső korlátot viszonylag könnyen meghatározhatjuk. Ehhez azt kell látnunk, hogy a gyökér i címkéjű éléhez tartozó részfa levelei reprezentálják a GY^i_ℓ elemeit. A szófa egyetlen bejárásával egy egyszerű rekurzív módszer segítségével minden csúcshoz kiszámíthatjuk a $KK^*_{\ell-d,p}(GY^I_{\ell-d})$ és $KK^{\ell-d+p}_{\ell-d}(|GY^I_{\ell-d}|)$ értékeket, ahol d a csúcs mélységét jelöli, $GY^I_{\ell-d}$ pedig az adott csúcshoz tartozó részfa által reprezentált elemhalmazokat. A gyökérhez kiszámított két érték adja meg a KK és KK^* korlátokat.

Ha a maradék futási idő becslésére kívánjuk használni a fenti felső korlátot, akkor tudnunk kell, hogy a jelöltek támogatottságának meghatározása függ az Apriori algoritmusban felhasznált adatstruktúrától. Szófa esetében például egy jelölt előfordulásának meghatározásához el kell jutnunk a jelöltet reprezentáló levélhez, ami a jelölt méretével arányos lépésszámú műveletet igényel. A maradék futási idő pontosabb felső becsléséhez a $KK^*_{\ell+p}(GY_\ell)$ értékeket súlyozni kell $(\ell+p)$-vel.

31.2.2. Az Eclat algoritmus

Az Eclat az üres mintából indulva egy rekurzív, mélységi jellegű bejárást valósít meg. A rekurzió mélysége legfeljebb eggyel több, mint a legnagyobb gyakori elemhalmaz mérete. Az Apriori-val szemben mindig egyetlen jelöltet állít elő, majd ennek azonnal meghatározza a támogatottságát. Az $(\ell+1)$-elemű, P prefixű jelölteket, ahol $|P| = \ell - 1$ az ℓ-elemű, P prefixű gyakori elemhalmazokból állítja elő egyszerű páronkénti unióképzéssel.

Az algoritmus központi fogalma az ún. TID-halmaz. Egy elemhalmaz ***TID-halmazának*** (Transaction IDentifier) elemei azon bemeneti sorozatok azonosítói (sorszámai), amelyek tartalmazzák az adott elemhalmazt. Más szóval egy TID-halmaz a vertikális adatbázis egy megfelelő sora. Például $\langle AD, AC, ABCD, B, AD, ABD, D\rangle$ bemenet esetén az $\{A, C\}$ elemhalmaz TID-halmaza $\{1, 2\}$, amennyiben egy tranzakció azonosítója megegyezik a bemeneti sorozatban elfoglalt helyével, és a helyek számozását nullától kezdjük.

A TID-halmaz két fontos tulajdonsággal bír:

1. Az I elemhalmaz TID-halmazának mérete megadja az I támogatottságát.

2. Egy jelölt TID-halmazát megkaphatjuk a generátorainak TID-halmazaiból egy egyszerű metszetképzéssel.

Az Eclat pszeudokódja az alábbi.

ECLAT($\mathcal{T}$, *min-supp*)

```
1  for minden t ∈ T                ▷ Elemek támogatottságának meghatározása.
2      for minden i ∈ t
3          J_1 = J_1 ∪ {i}
4          i.számláló ← i.számláló + 1
5  for minden j ∈ J_1                          // Gyakori elemek meghatározása.
6      if j.számláló ≥ min-supp
7      GY_1 = GY_1 ∪ {j}
8  for i = 1 to |T|                // Gyakori elemek TID-halmazainak felépítése.
9      for minden j ∈ t_i ∩ GY_1
10         j.TID ← j.TID ∪ {i}
11 return GY_1∪ ECLAT-SEGÉD(GY_1, ∅, min-supp)
```

Először meghatározzuk a gyakori elemeket, majd felépítjük a gyakori elemek TID-halmazait. A későbbiekben nem használjuk a bemenetet, csak a TID-halmazokat. Az algoritmus lényege a ECLAT-SEGÉD rekurziós eljárás. Jelöljük a P prefixű, P-nél eggyel nagyobb méretű gyakori elemhalmazokból alkotott halmazcsaládot GY^P-vel. Nyilvánvaló, hogy $GY^{\emptyset} = GY_1$.

ECLAT-SEGÉD(GY^P, P, *min-supp*)

```
1  for minden gy ∈ GY^P
2      for minden gy' ∈ GY^P, gy ≺ gy'
3          j = gy ∪ gy'
4          j.TID = gy.TID ∩ gy'.TID
6          if |j.TID| ≥ min-supp
7              GY^gy = GY^gy ∪ {j}
8          if |GY^gy| ≥ 2
9              GY = GY ∪ GY^gy∪ ECLAT-SEGÉD(GY^gy, gy, min-supp)
10             else GY = GY ∪ GY^gy
11 return GY
```

Az ECLAT jelölt-előállítása megegyezik az APRIORI jelölt-előállításával, azzal a különbséggel, hogy nem ellenőrizzük az unióképzéssel kapott halmaznak minden részhalmazára, hogy gyakori-e (a mélységi bejárás miatt ez az információ nem is áll rendelkezésünkre). Látható, hogy az ECLAT abban is különbözik az APRIORI-tól, hogy egy jelölt előállítása után azonnal meghatározza a támogatottságát, mielőtt újabb jelöltet állítana elő. Nézzünk egy példát a keresési tér bejárására.

31.3. példa. Legyen $\mathcal{T} = \langle ACDE, ACG, AFGM, DM \rangle$ és *min-supp* = 2. Első lépésben meghatározzuk a gyakori elemeket: A, C, D, G, M, ami nem más, mint $GY^{\emptyset}$. Ezután előállítjuk és azonnal meg is határozzuk az (A,C), (A,D), (A,G), (A,M) párok unióját. Ezek közül csak az AC, AG halmazok gyakoriak. A következő rekurziós lépésben ennek a két halmaznak vesszük az unióját, állítjuk elő a TID-halmazát, amely alapján kiderül, hogy az ACG ritka, és a rekurzió ezen ága véget ér. Ezután a C elemnek vesszük az unióját a sorban utána következő elemekkel egyesével és így tovább.

Látnunk kell, hogy az Eclat legalább annyi jelöltet állít elő, mint az Apriori. A mélységi bejárás miatt ugyanis egy jelölt előállításánál nem áll rendelkezésünkre az összes részhalmaz. Az előző példa esetében például az {A,C,G} támogatottságát hamarabb vizsgálja, mint a {C,G} halmazét, holott ez utóbbi akár ritka is lehet. Ebben a tekintetben tehát az Eclat rosszabb az Apriori-nál, ugyanis több lesz a ritka jelölt.

Az Eclat igazi ereje a jelöltek támogatottságának meghatározásában van. A jelöltek TID-halmazainak előállítása egy rendkívül egyszerű és nagyon gyors művelet lesz. Emellett ahogy haladunk egyre mélyebbre a mélységi bejárás során, úgy csökken a TID-halmazok mérete, és ezzel a támogatottság meghatározásának ideje is. Ezzel szemben az Apriori-nál ahogy haladunk az egyre nagyobb méretű jelöltek felé, úgy nő a szófa mélysége, és lesz egyre lassabb minden egyes jelölt támogatottságának meghatározása.

A keresési tér bejárása függ a prefix definíciójától, amit az elemeken definiált rendezés határoz meg. Melyek lesznek azok a jelöltek, amelyek az Apriori-ban nem lennének jelöltek (tehát biztosan ritkák), illetve várhatóan melyik az a rendezés, amely a legkevesebb ilyen tulajdonságú halmazt adja? Ha egy elemhalmaz jelölt az Eclat algoritmusban, de az Apriori-ban nem, akkor van olyan részhalmaza, amely ritka. Amennyiben feltételezzük, hogy az elemek függetlenek, akkor azon részhalmaz előfordulásának lesz legkisebb a valószínűsége (és ezzel együtt az esélye annak, hogy ritka), amely a leggyakoribb elemet nem tartalmazza. A jelölt prefixe generátor, tehát gyakori, így akkor lesz a legnagyobb esélye annak, hogy minden részhalmaz gyakori, ha a prefix a leggyakoribb elemet nem tartalmazza. Az Eclat algoritmusnál a legkevesebb ritka jelöltet és így a legjobb futási időt tehát a gyakoriság szerint csökkenő rendezéstől várhatjuk.

31.4. példa. Ennek a gondolatmenetnek az illusztrálására nézzük a következő példát. Legyenek gyakori halmazok a következők: A, B, C, D, AB, AC, BC, AD, ABC, továbbá $supp(A) \prec supp(B) \prec supp(C) \prec supp(D)$. Amennyiben az Eclat algoritmus a gyakoriság szerint növekvő sorrendet használja, akkor az előállítás sorrendjében a következő halmazok lesznek jelöltek: A, B, C, D, AB, AC, AD, ABC, ABD, ACD, BC, BD, CD. Ugyanez gyakoriság szerint csökkenő sorrendnél D, C,

B, A, DC, DB, DA, CB, CA, CBA, BA. Az utóbbi esetben tehát négy ritka jelölt helyett (ABD, ACD, BD, CD) csak kettő lesz (CD, BD). Megjegyezzük, hogy ez a két elemhalmaz az APRIORI esetében is jelölt lesz. A gyakoriság szerint csökkenő esetben egyszer állítunk elő olyan háromelemű jelöltet, amelynek van olyan kételemű részhalmaza, amelyet nem vizsgáltunk. Ez a jelölt a CBA és a nem megvizsgált részhalmaz a BA. Mivel a részhalmaz éppen a leggyakoribb elemeket tárolja, ezért van nagy esélye annak, hogy gyakori (főleg ha hozzávesszük, hogy a jelölt két generátora, CB és CA is gyakori).

31.2.3. Az FP-growth algoritmus

Az FP-GROWTH algoritmus[3] egy mélységi jellegű, rekurzív algoritmus, a keresési tér bejárása tekintetében megegyezik az ECLAT-tal. A támogatottságok meghatározását az egyelemű gyakori halmazok meghatározásával, majd a bemenet ***szűrésével*** és ***vetítésével*** valósítja meg rekurzív módon. A bemenet szűrése azt jelenti, hogy az egyes tranzakciókból töröljük a bennük előforduló ritka elemeket. A $\mathcal{T}$ elemhalmaz P elemhalmazra vetítését (jelölésben $\mathcal{T}|P$) pedig úgy kapjuk, hogy vesszük a P-t tartalmazó tranzakciókat, majd töröljük belőlük a P-t. Például $\langle ACD, BCE, ABCE, BE, ABCE\rangle|B = \langle CE, ACE, E, ACE\rangle$. Az algoritmus pszeudokódja a következőkben olvasható.

FP-GROWTH($\mathcal{T}$, *min-supp*)

```
1  FP-GROWTH-SEGÉD(T, min-supp, ∅)
```

A segédeljárás harmadik paramétere (P) egy prefix elemhalmaz, az első paraméter pedig az eredeti bemenet P-re vetítése. Az eredeti bemenet $\emptyset$-ra vetítése megegyezik önmagával.

FP-GROWTH-SEGÉD(T, *min-supp*, P)

```
1  for minden t ∈ T                          ▷ A tranzakciókban előforduló
2      for minden i ∈ t        // elemek támogatottságának meghatározása.
3          J_1 = {i}
4          i.számláló = i.számláló + 1
```

[3] Az FP a **F**requent **P**attern rövidítése, ami miatt az algoritmust ***mintanövelő*** algoritmusnak is hívják. Ez az elnevezés azonban félrevezető, ugyanis szinte az összes GYEK algoritmus mintanövelő abban az értelemben, hogy egy új jelölt a generátorainak egyelemű bővítése, vagy más szóval növelése. Az FP-GROWTH sajátsága nem a jelöltek előállítása, hanem a jelöltek támogatottság-meghatározásának módja.

```
5  for minden j ∈ J_1                          // Gyakori elemek kiválogatása.
6      if j.számláló ≥ min-supp
7          GY_1 ← GY_1 ∪ {j}
8  T* = SZŰRÉS(T, GY_1)
9  for minden gy ∈ GY_1
10     T*|gy = VETÍTÉS(T*, gy)
11     GY = GY ∪ {P ∪ {gy}}
12     GY = GY ∪ FP-GROWTH(T*|gy, min-supp, P ∪ {gy})
13     T* = TÖRLÉS(T*, gy)
14 return GY
```

Egy rekurziós lépés három fő lépésből áll. Először meghatározzuk azon elemek támogatottságát, amelyek előfordulnak valamelyik tranzakcióban (1–4. sorok). Ezekből kiválasztjuk a gyakoriakat (5–7. sorok). Ezután minden gy gyakori elemet egyesével veszünk (9. sor). Meghatározzuk a gy-hez tartozó vetített bemenetet, majd meghívjuk az algoritmust rekurzívan a $\mathcal{T}|gy$ bemenetre. Törölnünk kell a gy elemet a $\mathcal{T}^*$-beli tranzakciók elemei közül (13. sor) annak érdekében, hogy egy jelöltet csak egyszer állítsunk elő.

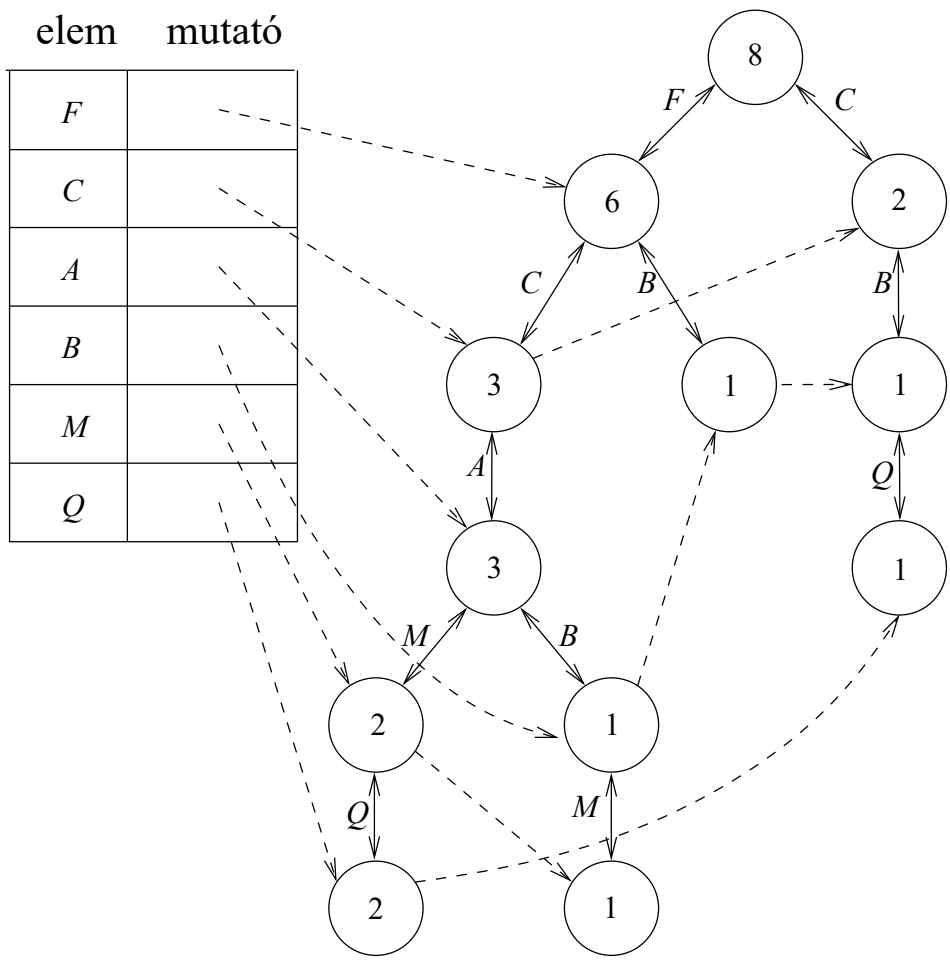
A jelöltek előállításának tekintetében az FP-GROWTH algoritmus a legegyszerűbb. Ha az I elemhalmaz gyakori, akkor a következő rekurziós szinten azon $I \cup j$ halmazok lesznek a jelöltek, ahol j az I-re vetített bemenetben előforduló elem és $I \cup j$ nem volt jelölt korábban. Tulajdonképpen az FP-GROWTH a nagy elemszámú jelöltek támogatottságának meghatározását visszavezeti három egyszerű műveletre: egyelemű gyakori elemhalmazok kiválogatása, szűrés és vetített bemenet előállítása.

A 9. sorban egyesével vesszük a gyakori elemeket. Ezt valamilyen rendezés szerint kell tennünk és ez a rendezés határozza meg, hogy milyen sorban járjuk be a keresési teret, milyen vetített bemeneteket állítunk elő és mely elemhalmazok lesznek a hamis jelöltek. AzECLAT-nál elmondottak itt is élnek; várhatóan abban az esetben lesz a hamis jelöltek száma minimális, amennyiben a prefixben a legritkább elemek vannak, azaz a 9. sorban gyakoriság szerint növekvő sorban vesszük az elemeket.

Az FP-GROWTH algoritmus szerves része az ***FP-fa,*** amelyben a szűrt bemenetet tároljuk. Az FP-fa segítségével könnyen előállíthatjuk a vetített bemeneteket, azokban könnyen meghatározhatjuk az elemek támogatottságát, amiből előállíthatjuk a vetített, majd szűrt bemenetet. Ezt a vetített és szűrt bemenetet szintén egy FP-fában tároljuk, amelyet ***vetített FP-fának*** hívunk.

Az FP-fa egy keresztélekkel és egy fejléc táblával kibővített szófa. Az élek

címkéi gyakori elemek. Az egyszerűbb leírás kedvéért egy (nemgyökér) csúcs címkéjén a csúcsba mutató él címkéjét értjük. Minden csúcs egy elemhalmazt reprezentál, amelynek elemei a gyökérből a csúcsig vezető út csúcsainak címkéivel egyeznek meg. Minden csúcshoz egy számlálót rendelünk. Ez a számláló adja meg, hogy a csúcs által reprezentált halmaz mennyi bemeneti (vagy vetített) elemhalmaznak a prefixe. Az azonos címkéjű csúcsok láncolt listaszerűen össze vannak kötve keresztirányú élekkel. A lánc legelső elemére mutat a fejléctáblának az adott eleméhez tartozó mutatója.



31.4. ábra. Az $\langle ACFMQ,\ ABCFM,\ BF,\ BCQ,\ ACFMQ,\ C,\ F,\ F\rangle$ szűrt bemenetet tároló FP-fa.

31.5. példa. Tegyük fel, hogy bemenetként a $\langle ACDFMQ, ABCFMO, BFO, BCKSQ, ACFMQ, CS, DFJ, FHI\rangle$ sorozat van adva, és $min\text{-}supp = 3$. A gyakori elemek: A, B, C, F, M, Q, amelyek támogatottsága rendre 3, 3, 5, 6, 3, 3. Ekkor a szűrt bemenetet ($\langle ACFMQ, ABCFM, BF, BCQ, ACFMQ, C, F, F\rangle$) reprezentáló FP-fa, amely gyakoriság szerint csökkenő sorrendet ($Q \prec M \prec B \prec A \prec C \prec F$) használ, a 31.5. ábrán látható

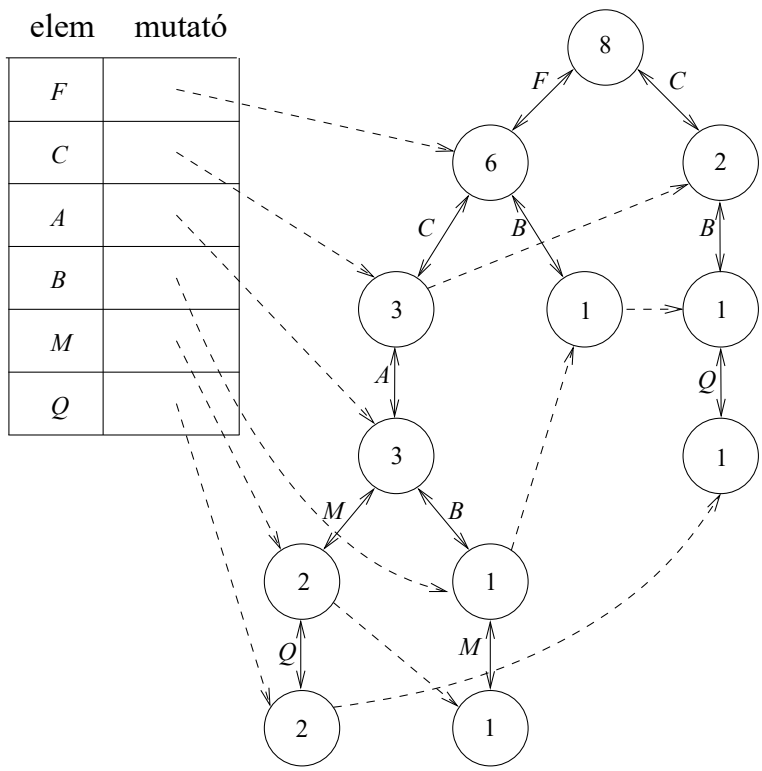
Egy FP-fát hasonló módon építünk fel, mint egy szófát. Különbség, hogy egy I elemhalmaz beszúrásánál nem csak az I-t reprezentáló levélnek a számlálóját növeljük eggyel, hanem minden olyan csúcsét, amelyet érintünk a beszúrás során (hiszen ezen csúcsokat reprezentáló halmazok az I prefixei). A keresztirányú éleket és a fejléctáblát is egyszerűen megkaphatjuk. Legyen a fejléctábla mutatóinak kezdeti értéke NIL. Amikor beszúrunk egy új, i címkéjű csúcsot, akkor két dolgot kell tennünk. Az új csúcs keresztél mutatója felveszi a fejléctábla i-hez tartozó bejegyzését, majd ezt a bejegyzést az új csúcs címére cseréljük. Ezzel tulajdonképpen olyan láncot készítünk, amelyben a csúcsok a beszúrási idejük szerint csökkenően vannak rendezve (az először beszúrt elem van leghátul) és a lista a fejléctáblában kezdődik.

Az FP-fa mérete – hasonlóan a szófa méretéhez – függ az elemeken definiált rendezéstől. Az FP-GROWTH algoritmus akkor lesz hatékony, ha a fa elfér a memóriában, ezért fontos lenne azt a rendezést használni, ami várhatóan a legkisebb fát eredményezi. Az APRIORI esetében már elmondtuk, hogy az a heurisztika, amely az elemek gyakoriság szerint csökkenő rendezését használja, általában kis méretű fát eredményez.

Egyszerű lesz a vetített bemenet előállítása és a szűrt bemenetből egy elem törlése, amennyiben a legritkább gyakori elemet (gy_r) vesszük először. Ez összhangban áll azzal, hogy a pszeudokód 9. sorában az elemeket gyakoriság szerint növekvő sorrendben vesszük. A gy_r csak levél címkéje lehet. Mivel a fából törölni fogjuk a gy_r címkéjű csúcsokat a rekurziós művelet után (13. sor), a következő elem is csak levél címkéje lesz.

Nézzük most meg, hogy amennyiben a szűrt bemenet egy FP-fában van tárolva, akkor hogyan kaphatjuk meg a gy_r elemre vett vetítésben az elemek támogatottságát. A fejléctábla gy_r eleméhez tartozó mutatóból kiindulva a keresztélek alkotta láncban pontosan azok a csúcsok vannak, amelyek gy_r-t tartalmazó bemeneti elemet reprezentálnak. Az egyes elemhalmazok előfordulását a gy_r címkéjű csúcsokhoz rendelt számláló adja meg, az elemeiket pedig a gyökérig felsétálva kaphatjuk. A lista utolsó csúcsának feldolgozása után rendelkezésünkre állnak a gy_r elemhez tartozó vetített bemenetben valahol előforduló elemek támogatottságai, amely alapján kiválogathatjuk a vetített bemenetben gyakori elemeket.

Ugyanilyen bejárással kaphatjuk meg a vetített, majd szűrt bemenetet



31.5. ábra. Az $\langle ACFMQ, ABCFM, BF, BCQ, ACFMQ, C, F, F\rangle$ szűrt bemenetet tároló FP-fa.

tartalmazó FP-fát. A fejléctáblából kiindulva végigmegyünk a láncolt lista elemein. A csúcs által reprezentált elemhalmazból töröljük a ritka elemeket, majd a kapott elemhalmazt beszúrjuk az új FP-fába. A kis memóriaigény érdekében a gyakoriság szerint csökkenő sorrendet használjuk. Ezt a sorrendet a vetített bemenet alapján állítjuk fel (lévén az új fa a vetített és szűrt bemenetet fogja tárolni), ami különbözhet az eredeti FP-fában alkalmazott rendezéstől.

31.6. példa. Folytassuk az előző példát és állítsuk elő a legritkább gyakori elemhez (Q) tartozó vetített és szűrt bemenetet. A fejléctábla Q eleméhez tartozó mutatóból kiindulva mindössze két csúcsot látogatunk meg, ami azt jelenti, hogy a vetített bemenet két különböző elemhalmazt tartalmaz: az $FCAM$-et kétszer, a CB-t egyszer. Ez alapján a vetített bemenetben egyetlen gyakori elem van, C. Ez a rekurziós ág nem folytatódik, hanem visszatér a QC gyakori elemhalmazzal. Az FP-fából törölhetjük a fejléctábla Q bejegyzéséhez tartozó mutatóból, keresztirányú élek segítségével elérhető csúcsokat. A következő vizsgált elem az M. Az M vetített bemenetében három gyakori elem van, és a vetített szűrt bemenet az FCA elemhalmazt tartalmazza háromszor. Ezt a vetített, szűrt bemenetet egy egyetlen útból álló FP-fa fogja reprezentálni. A többi FP-fa ugyanilyen egyszerűen megkapható.

Hatékonysági szempontból rendkívül fontos, hogy a rekurziót ne folytassuk, ha a vizsgált FP-fa egyetlen útból áll. A rekurzió helyett képezzük inkább az út által reprezentált elemhalmaz minden részhalmazát. A részhalmaz támogatottságát annak a csúcsnak a számlálója adja meg, amely a legmélyebben van a részhalmazt meghatározó csúcsok között.

Hasonlítsuk össze a három algoritmust. A ritka jelöltek számát vizsgálva Apriori a legjobb, vagy más szavakkal, Apriori tölti a legkevesebb idő felesleges elemhalmazok támogatottságának meghatározásával. A támogatottságok meghatározásának tekintetében Apriori az esetek többségében nem tud labdába rúgni a két riválisával szemben. Ennek oka az, hogy az Apriori semmilyen már kinyert tudást nem használ fel az új jelöltek támogatottságának meghatározása során. Ezzel szemben a másik két algoritmus ügyesen kihasználja a prefixek előfordulásáról megszerzett tudást. Az Eclat-nál rendelkezésre állnak a jelölt generátorainak TID-halmazai, az FP-growth esetében pedig a prefixre vetített bemenet.

31.2.4. Toivonen mintavételező algoritmusa

Az adatbányászati alkalmazásokban a feldolgozandó adathalmaz mérete óriási lehet, akár több terabájt nagyságú is. Habár a rendelkezésre álló erőforrások (memória, processzor teljesítmény) is egyre nagyobbak, az algoritmusok nem mindig adnak elfogadható időn belül megoldást. Sokszor a gyors válaszidő fontosabb, mint hogy minden gyakori elemhalmazt megtaláljunk. Ilyen ese-

tekben mintavételező-algoritmusokat célszerű használni.

Tegyük fel, hogy rendelkezésünkre áll a bemeneti sorozatnak egy akkora – nem feltétlenül összefüggő – részsorozata ($\mathcal{T}'$), amelyet könnyen fel tudunk dolgozni valamely, az előzőekben bemutatott algoritmusnak a segítségével. Jelöljük a $\mathcal{T}'$-ben gyakori elemhalmazok halmazát GY'-vel.

A GY' elemei nem feltétlenül egyeznek meg az eredeti sorozatban gyakori elemhalmazokkal. Ennek két oka van. Egyfelől lehet, hogy a $\mathcal{T}'$-ben gyakori elemhalmaz valójában ritka. Másfelől a $\mathcal{T}'$-ben ritka elemhalmaz lehet, hogy valójában gyakori.

Az első típusú hibát könnyű kiküszöbölni: tekintsük a GY' elemeit jelölteknek, és határozzuk meg a támogatottságokat most már a $\mathcal{T}$-ben. A hatékonyság miatt célszerű az elemhalmazokat az Apriori algoritmushoz hasonlóan egy szófában tárolni. Különbség lesz, hogy ekkor a szófában egyszerre vannak jelen különböző méretű elemhalmazok, így számlálót nem csak a levelekhez kell rendelni, hanem a belső pontokhoz is.

A második típusú hibát nem lehet ilyen egyszerűen kiküszöbölni. A hiba elkövetésének esélyét csökkenthetjük, ha csökkentjük *min-freq*-et. Ez azt a veszélyt rejti magában, hogy túl sok ritka elemhalmaz lesz gyakori a részsorozatban. A másik, talán még nagyobb probléma, hogy nem lehetünk biztosak abban, hogy minden gyakori elemhalmazt megtaláltunk.

Toivonen ezért a *min-freq* csökkentése helyett a következőt javasolta. Ne csak a GY' elemeinek támogatottságát határozzuk meg $\mathcal{T}$-ben, hanem az $NB(GY')$ elemeiét is. Egy elemhalmaz eleme $NB(GY')$-nek, amennyiben nem eleme GY'-nek, de minden valódi részhalmaza benne van GY'-ben.[4]

31.7. példa. Legyen $\mathcal{I} = \{A, B, C, D, E, F\}$ és $GY' = \{A, B, C, F, AB, AC, AF, CF, ACF\}$. Ekkor $NB(GY') = \{BC, BF, D, E\}$.

A következő állítás szerint bizonyos esetekben biztosak lehetünk abban, hogy nem veszítünk el jelöltet.

31.11. állítás. *Legyen $\mathcal{T}'$ a $\mathcal{T}$ bemeneti sorozat egy része. Jelöljük GY-vel a $\mathcal{T}$-ben, GY'-vel az $\mathcal{T}'$-ben gyakori elemhalmazokat és GY^*-gal azokat az $\mathcal{T}$-ben gyakori elemhalmazokat, amelyek benne vannak $GY' \cup NB(GY')$-ben ($GY^* = GY \cap (GY' \cup NB(GY'))$). Amennyiben*

$$NB(GY^*) \subseteq GY' \cup NB(GY')$$

teljesül, akkor $\mathcal{T}$-ben a gyakori elemhalmazok halmaza pontosan a GY^*, tehát $GY^* \equiv GY$.

[4] Az NB jelölést a „negative border” rövidítéséből kapjuk.

Bizonyítás. Indirekt tegyük fel, hogy létezik $I \in GY$, de $I \notin GY^*$, és a feltétel teljesül. Tekintsünk egy legkisebb méretű $I' \subseteq I$-t, amire $I' \in GY$ és $I' \notin GY^*$ (ilyen I'-nek kell lennie, ha más nem, ez maga az I halmaz). A GY^* definíciója miatt ekkor $I' \notin GY' \cup NB(GY')$. Az I' minimalitásából következik, hogy minden valódi részhalmaza eleme $GY' \cup NB(GY')$-nek és gyakori $\mathcal{T}$-ben. Ebből következik, hogy I' minden részhalmaza eleme GY^*-nak, amiből kapjuk, hogy $I' \in NB(GY^*)$. Ez ellentmond az $NB(GY^*) \subseteq GY' \cup NB(GY')$ feltételnek, hiszen van olyan elem, amely eleme a bal oldalnak, de nem eleme a jobb oldalnak. ■

Amennyiben a $\mathcal{T}'$-ben gyakori elemhalmazok meghatározásához az Apriori algoritmust használjuk, akkor az $NB(GY')$ elemeit nem kell külön előállítanunk. A ritka jelöltek halmaza ugyanis pontosan $NB(GY')$ lesz (ez a tulajdonság sem az Eclat, sem az FP-growth algoritmusra nem teljesül).

31.8. példa. Legyen $\mathcal{I} = \{A, B, C, D\}$ és $GY' = \{A, B, C\}$. Ekkor az $NB(GY')$ elemei az $\{AB, AC, BC, D\}$ halmazok. Tehát ennek a 7 elemhalmaznak fogjuk a támogatottságát meghatározni a teljes bemenetben. Ha például az A, B, C, AB halmazokat találjuk gyakorinak, akkor a tételbeli tartalmazási reláció fennáll, hiszen az $NB(\{A, B, C, AB\})$ összes eleme szerepel a 7 jelölt között. Nem mondhatjuk el ugyanezt, ha D-ről derül ki, hogy gyakori. Ekkor Toivonen algoritmusa jelenti, hogy nem biztos, hogy minden gyakori elemhalmazt megtalált.

Gyakorlatok

31.2-1. Mutassuk meg, hogy már a kételemű halmazokat tároló minimális méretű szófa megtalálása is NP-nehéz. *Útmutatás.* Használjuk fel azt, hogy egy gráf minimális lefogó élhalmazának meghatározása NP-nehéz feladat.

31.2-2. Mutassunk példát arra, amikor nem a gyakoriság szerint csökkenő sorrend adja a minimális méretű szófát.

31.2-3. Adjunk olyan módszert az egy- és kételemű jelöltek támogatottságának meghatározására, amely a szófát használó módszernél gyorsabb és kevesebb memóriát használ.

31.2-4. Legyen $GY_2 = \{AB, AC, CD, DE, EF, FG\}$. Határozzuk meg $KK_2^{2+1}(|GY_2|)$ és $KK_{2,1}^*(GY_2)$ értékeket.

31.2-5. Melyik rendezéstől várhatjuk a legkisebb KK^* felső korlátot abban az esetben, ha az $\mathcal{I}$ elemeinek előfordulása függetlenek egymástól.

31.2-6. Adjuk olyan bemeneti sorozatot és *min-supp* küszöböt, amelyek esetében az FP-growth és az Eclat különböző módon járja be a keresési teret.

Feladatok

31-1 Nem bővíthető gyakori elemhalmazok

Az I elemhalmaz ***nem bővíthető, gyakori*** elemhalmaz, mennyiben nem létezik olyan gyakori elemhalmaz, amelynek I valódi részhalmaza. A nem bővíthető, gyakori elemhalmazokat egy utószűréssel megkaphatjuk a gyakori elemhalmazokból. Amennyiben csak a nem bővíthető gyakori minták kinyerése a cél, akkor hatékonyabban megoldhatjuk a feladatot olyan algoritmussal, amely eleve csak ezeket a halmazokat határozza meg. Hogyan kell módosítani az Apriori, Eclat és FP-growth algoritmusokat, hogy csak a nem bővíthető, gyakori mintákat nyerje ki? *Útmutatás.* Gondoljuk meg, hogy melyek azok a gyakori bővíthető elemhalmazok, amelyek nem szabad törölnünk, mert még szükség van rájuk az algoritmusnak egy későbbi lépésében.)

31-2 Zárt elemhalmazok

Legyen az I elemhalmaz ***lezártja*** az a legnagyobb elemhalmaz, amelynek támogatottsága megegyezik I támogatottságával, és tartalmazza I-t részhalmazként. I lezártját jelölje $h(I)$.

a. Mutassuk meg, hogy a lezárás valóban egy lezárási operátor, tehát $I \subseteq h(I)$, idempotens, azaz $h(h(I)) = h(I)$ és monoton, más szóval $I \subseteq J$ esetén $h(I) \subseteq h(J)$.

b. Mutassuk meg, hogy minden maximális méretű gyakori elemhalmaz zárt is egyben.

c. A zártságot felhasználva hogyan tudjuk csökkenteni a jelöltek számát az Apriori, Eclat és FP-growth algoritmusokban?

d. Hogyan kell módosítani az Apriori, Eclat és FP-growth algoritmusokat, hogy eleve csak a gyakori, zárt mintákat nyerjék ki?

Megjegyzések a fejezethez

A gyakori elemhalmazok keresése először az asszociációs szabályok kinyerésének [9] részfeladataként merült fel 1993-ban. Egy évvel később Agrawal és Srikant publikálták az Apriori algoritmust [7], illetve tőlük függetlenül Mannila, Toivonen és Verkamo [248]. Az öt szerző végül egyesítette a két írást [6] (az optimális szófa NP-teljességét Comer és Sethi bizonyította a 3SAT problémára való visszavezetéssel [80]). A jelöltekre vonatkozó felső korlátokat Geerts, Goethals és Bussche bizonyították [129]. A probléma felbukkanása után az Apriori algoritmus és különböző módosulatai voltak

jellemzőek. A 90-es évek közepén még viszonylag kis memóriával rendelkeztek a számítógépek, így a legfőbb törekvés az algoritmusok I/O műveletei számának csökkentése volt. Emellett a jelöltek számát próbálták csökkenteni, mondván, hogy annál gyorsabb egy algoritmus, minél kevesebb időt tölt a hamis jelöltek támogatottságának meghatározásával.

Az ECLAT [382] és FP-GROWTH [169] algoritmusokat Zaki és Han által vezetett csoportok mutatták be. A két módszer rendkívül hatékonynak bizonyult és megcáfolták azt a korábbi hipotézist, hogy a sok ritka jelöltet előállító algoritmusok csak rosszak lehetnek. Ezzel, a mélységi bejárást megvalósító algoritmusok kiütötték a kutatások középpontjából a szélességi bejárást megvalósító algoritmusokat.

A GYEK algoritmusok nagy száma miatt 2003-ban megrendezték a GYEK implementációk első versenyét [146]. A rendezvény rengeteg tanulsággal és meglepetéssel szolgált. Sok új, gyorsnak beharangozott algoritmus meglehetősen rosszul szerepelt a teszteken, jócskán elmaradva még a legegyszerűbb algoritmustól, az APRIORI-tól is. A versenynek nem volt abszolút győztese, a különböző karakterisztikával rendelkező adatbázisokon más-más implementáció győzött. Az FP-GROWTH [300] és ECLAT [344] algoritmusok gyorsítása és egy új algoritmus, az LCM [340] azonban mindig az elsők között szerepelt. Az APRIORI [56, 57, 58] az alacsony memóriaigényével tűnt ki sok esetben. A versenyzők folyamatosan frissíthetik programjaikat, amelyek fél évenként összemérhetik erejüket. A legfrissebb eredmények és implementációk megtalálhatók a `http://fimi.cs.helsinki.fi` oldalon.

Nemcsak egyre gyorsabb algoritmusok születtek, hanem a minták típusa is egyre bonyolultabb lett. Elemhalmazokon túl kerestek gyakori sorozatokat, elemhalmazokat tartalmazó sorozatokat [8], epizódokat [249], Boole-formulákat [247], címkézett rendezett/nem rendezett gyökeres fákat [379] és gyakori részgráfokat [217], illetve feszített részgráfokat [189].

A nem bővíthető gyakori elemhalmazok fogalmát a *31-1* feladatban ismertettük. A nem bővíthető, gyakori elemhalmazok alapján tetszőleges elemhalmazról el tudjuk dönteni, hogy gyakori vagy sem, igaz, a pontos támogatottságokat nem tudjuk kikövetkeztetni. A nem bővíthető, gyakori elemhalmazok száma jóval kisebb lehet az összes gyakori elemhalmaz számánál, ezért sokszor csak ezeket nyerik ki. A legismertebb ilyen algoritmusok az FP-GROWTH* [157], MAFIA [65] és az AFOPT [232].

A gyakori elemhalmazoknak másik fontos részhalmaza a *zárt* gyakori elemhalmazok (pontos definíciót lásd *31-2*). A zárt elemhalmazokból meg tudjuk határozni a gyakori elemhalmazokat és azok pontos támogatottságát is. A zártság fogalmát Pasquier [277, 279, 278] és Zaki [381] vezette be egymástól függetlenül, és később számos algoritmus született (APRIORI-CLOSE [279],

CHARM [380], CLOSET [282], CLOSET+ [358], MAFIA [65]), amelyek csak a gyakori zárt elemhalmazokat nyerik ki.

Két adatbányászatról szóló könyvet fordítottak magyarra [5, 169]. Mindkét könyv bevezető jellegű, elsősorban a nagyközönség számára készült. Ugyancsak bevezető jellegű, de sok témával foglalkozó mű a Khosrow-Pour által szerkesztett enciklopédia [211]. Gyakori elemhalmazok kereséséről a társítási szabályokról szóló részekben olvashatunk. A [5] könyvet röviddel a GYEK feladat születése után adták ki, így nem csoda, hogy csak az APRIORI algoritmus szerepel benne. [169]-ban az APRIORI mellett részletesen írnak az FP-GROWTH-ról is. [55]-ban olvashatunk legtöbbet a gyakori minták kereséséről. Szerepel az írásban elemhalmazok keresésén kívül sorozatok, epizódok, címkézett fák és gráfok keresése is. Szó van még a munkában a GYEK feladat változatairól is: például zárt és nem bővíthető minták keresése, dinamikus GYEK, kényszerek kezelése, illetve GYEK változó támogatottsági küszöb esetén.

A szerző munkáját részben támogatta a T 042706 számú OTKA-szerződés.

32. Klaszterezés

A ***klaszterezés*** egy adathalmaz pontjainak, rekordjainak hasonlóság alapján való csoportosítását jelenti. A klaszterezés szinte minden nagyméretű adathalmaz ***leíró modellezésére*** alkalmas. A teljesség igénye nélkül néhány példa: csoportosíthatunk weboldalakat tartalmuk, webfelhasználókat böngészési szokásaik, kommunikációs és szociális hálózatok pontjait közösségeik, kémiai vegyületeket szerkezetük, géneket funkcióik, betegségeket tüneteik szerint.

A klaszterezés feladatával eleinte a statisztikában kezdtek foglalkozni, majd az adatbányászat keretében az igen nagy méretű adathalmazok klaszterezésének kérdései kerültek az előtérbe. Így a klaszterezés az adatbányászat egyik legrégebbi és talán leggyakrabban alkalmazott része. A klaszterezés, illetve osztályozás során az adatpontokat diszjunkt csoportokba, a későbbiekben használt nevükön klaszterekbe, illetve osztályokba soroljuk, azaz particionáljuk az adathalmaz elemeit. A klaszterezés célja, hogy az elemeknek olyan partícióját adjuk meg, amelyben a közös osztályba kerülő elempárok lényegesen hasonlóbbak egymáshoz, mint azok a pontpárok, melyek két különböző osztályba sorolódtak.

Klaszterezés során a megfelelő csoportok kialakítása nem egyértelmű feladat, hiszen a különböző adatok eltérő jelentése és felhasználása miatt adathalmazonként más és más szempontok szerint kell csoportosítanunk. További nehézséget jelent, hogy egy n darab adatpontot tartalmazó adathalmaznak Bell-számnyi, $O(e^{n \lg n})$ darab klaszterezése lehet, vagyis az adatpontok számában exponenciálisnál nagyobb méretű keresési térben kívánunk egy – több szempontból is – optimális klaszterezést megtalálni.

A klaszterezéshez hasonló feladat az osztályozás. A klaszterezéstől az osztályozás feladatát az különbözteti meg, hogy az osztályozás *felügyelt* (supervised), a klaszterezés pedig *felügyelet nélküli* (unsupervised) csoportosítás. Osztályozás esetén a választható osztályok valamilyen ábrázolással, mintával vagy modellel leírva előre adottak. Klaszterezés esetén előzetesen megadott osztályok nincsenek, az adatok maguk alakítják ki a klasztereket, azok határait.

Egy klaszterezési feladat megoldásához ismernünk kell a fellelhető algoritmusok lényeges tulajdonságait, illetve szükségünk van az eredményként kapott klaszterezés kiértékelésére, jóságának mérésére. Ezeket az alapvető elemeket tárgyalja a 32.2. alfejezet.

Mivel egy klaszterezés az adatpontok hasonlósági viszonyaiból indul ki, ezért az első fontos lépés az adatpontok páronkénti hasonlóságát lehető legjobban megragadó hasonlósági függvény kiválasztása. A különféle adattípusokra leggyakrabban használt távolság és hasonlóság függvények leírása a 32.3. alfejezetben található meg.

Nagy adathalmazok klaszterezése során gyakran találkozunk azzal a problémával, hogy az adatpontok távolságának, illetve hasonlóságának kiszámítása nem oldható meg elég gyorsan ahhoz, hogy az adott klaszterező algoritmus elfogadható futási időben véget érjen. Számos esetben e problémának az adatok sok (tízes nagyságrendnél nagyobb számú) attribútummal való jellemzése az oka. Ebben az esetben szoktuk az adatok dimenzióját nagynak tekinteni. A hasonlóság vagy távolság hatékony kiszámításának problémáját megoldhatja az adatok dimenzió-csökkentése, amely során az adathalmazt – az adatpontok közötti hasonlósági viszonyokat a legkevésbé torzítva – olyan formába alakítjuk, amelyen már hatékonyan tudjuk a kívánt klaszterező algoritmust futtatni. A dimenzió-csökkentés két módszeréről olvashatunk a 32.4. alfejezetben.

A klaszterező módszerek tárgyalása során bemutatunk három klaszterezési elvet és azok legjellemzőbb algoritmusait. A particionáló algoritmusok (32.5. alfejezet) és a hierarchikus módszerek (32.6. alfejezet) a főleg a régebb óta ismert, de ma is gyakran használt módszereket képviselik, míg a sűrűség alapú eljárások (32.7. alfejezet) az elmúlt évtized új elveire, eljárásaira mutatnak példát.

32.1. Alapok

32.1.1. A hasonlóság és távolság tulajdonságai

A klaszterezés általunk használt modelljeiben minden egyes elempár *hasonlósága* vagy *távolsága* ismert, illetve kiszámítható.

32.1. definíció. *Tetszőleges két x,y klaszterezendő elem esetén azok **hasonlóságát** $s(x, y)$ jelöli. Feltesszük, hogy $0 \leq s(x, y) \leq 1$ és $s(x, y) = s(y, x)$ minden x, y pontpárra, valamint $s(x, x) = 1$ minden x adatpontra.*

Hasonlóság helyett többször az x, y pontok $d(x, y)$-vel jelölt távolságát fogjuk használni.

32.2. definíció. *Egy $d(x, y)$ függvény **távolság** ha teljesülnek rá a következő tulajdonságok:*

1. *minden x, y esetén $0 \leq d(x, y) < \infty$,*

2. *minden x adatpontra $d(x,x)=0$,*

3. *szimmetrikus, azaz minden x,y esetén $d(x,y)=d(y,x)$.*

Időnként feltesszük, hogy d-re teljesül a *háromszög-egyenlőtlenség.*

32.3. definíció. *Egy d távolságra akkor teljesül a **háromszög-egyenlőtlenség**, ha minden x,y,z ponthármasra $d(x,z) \leq d(x,y)+d(y,z)$. Ha egy távolságra a háromszög-egyenlőtlenség teljesül, akkor azt **metrikának** hívjuk.*

A későbbiekben, hacsak külön nem említjük, nem követeljük meg a háromszög-egyenlőtlenség teljesülését. Számos esetben az előforduló hasonlóság és távolság összekapcsolható a $d(x,y)=1-s(x,y)$ vagy az $s(x,y)=1/(1+d(x,y))$ összefüggésekkel.

32.1.2. Mátrixábrázolások

Egy klaszterező eljárás bemenetét a szimmetrikus ***távolságmátrixszal*** is reprezentálhatjuk:

$$\begin{bmatrix} 0 & d(1,2) & d(1,3) & \cdots & d(1,n) \\ d(2,1) & 0 & d(2,3) & \cdots & d(2,n) \\ d(3,1) & d(3,2) & 0 & \cdots & d(3,n) \\ & \vdots & \vdots & \ddots & \vdots \\ d(n,1) & d(n,2) & d(n,3) & \cdots & 0 \end{bmatrix},$$

ahol $d(i,j)$ adja meg az i-edik és a j-edik elem távolságát és n az adatpontok száma. Ennek mintájára értelmezhető a ***hasonlóságmátrix*** is.

A gyakorlatban legtöbbször az n adatpont attribútumokkal van leírva, és a hasonlóságokat az attribútumok értékeiből valamilyen függvény segítségével számolhatjuk ki. Ha megadjuk a függvényt, akkor elvben felírhatjuk a megfelelő hasonlóságmátrixot.

Elképzelhető, hogy a mátrix elemeinek kiszámolása megoldható, de elemeinek száma olyan nagy, hogy a hasonlóságmátrix nem fér el a belső tárban. Ha a hasonlóságmátrix ritka, azaz ha csak lineárisan sok nullától különböző elem szerepel benne, akkor segítséget jelent a ***ritka mátrixos tárolási forma*** használata. Ekkor a mátrixnak csak a nullától különböző elemeit tároljuk.

32.2. A klaszterező algoritmusok jóságának kérdései

Egy „jó” klaszterező algoritmussal szemben sokféle követelményt támaszthatunk. Ebben a fejezetben először arra mutatunk példát, hogy egy klaszterező eljárás kimenetének *jóságát* hogyan lehet numerikusan mérni. A fejezet további részében megadjuk azokat a legfontosabb tulajdonságokat, amelyeket egy klaszterező eljárástól meg szeretnénk követelni, majd a *kívülálló* adatpontok problémáját tárgyaljuk. Végül a 32.4. tétel a klaszterezés elvi határaira mutat példát: nem létezik túl sok intuitív szempontnak *pontosan* megfelelő klaszterező algoritmust.

Ha egy klaszterező eljárás jóságát, eredményének pontosságát szeretnénk mérni, és rendelkezésünkre áll egy előre adott, pontosnak tartott minta klaszterezés, akkor a vizsgálandó algoritmus adta klaszterezést össze tudjuk hasonlítani a minta klaszterezéssel. Egy ilyen összehasonlítás alapja a ***tévesztésmátrix.*** Legyen a minta klaszterezés t darab osztálya $M_1, M_2, \ldots, M_t$, és az eredményként kapott klaszterezés s darab osztálya $K_1, K_2, \ldots, K_s$. A $t \times s$ méretű C tévesztésmátrix elemei legyenek $c_{i,j} = |M_i \cap K_j|$. Látható, hogy ha a klaszterező algoritmus pontosan a minta klaszterezést adja eredményképpen, akkor C (feltéve a klaszterek megfelelő címkézését) egy diagonális mátrix lesz.

Minden M_i, K_j klaszterpárra értelmezhetjük az

$$r_{i,j} = \frac{|M_i \cap K_j|}{|K_j|}$$

felidézés (recall) és a

$$p_{i,j} = \frac{|M_i \cap K_j|}{|M_i|}$$

pontosság (precision) mennyiségeket. Az így kapott $t \cdot s$ mutatószámot többféleképpen aggregálhatjuk. Az egyik elterjedt lehetőség, hogy a koordinátánkénti két érték harmonikus közepeinek összegzése,

$$2 \cdot \sum_{i,j} \frac{|M_i \cap K_j|}{|M_i| + |K_j|} ,$$

adja a két klaszterezés eltérését.

Alább felsorolunk olyan általános tulajdonságokat, amelyek döntően befolyásolják egy klaszterező algoritmus alkalmazhatóságát.

Skálázhatóság: az algoritmus tár és idő szükséglete agy adathalmazon, illetve magas dimenziós adatokon alkalmazva is kezelhető maradjon.

Adattípusok: többfajta adattípusra is működjön (numerikus, bináris, kategorikus és ezek keverékei).

Klaszter-geometria: ne preferáljon a kialakítható klaszterek között azok geometriai tulajdonságai alapján.

Robusztusság: ne legyen zajos adatokra érzékeny;

Sorrend-függetlenség: ne függjön az adatpontok beolvasási sorrendjétől.

Hasznos tulajdonsága lehet egy klaszterező algoritmusnak, hogy ha előre megadhatunk számára klaszterhatárokat, azaz a kimenete egy előre adott (kényszer) klaszterezés finomítása lesz.

Gyakorlati példák esetében gyakran találkozhatunk olyan adatpontokkal, amelyekhez kevés más pont hasonló. Az olyan adatpontokat, amelyek lényegesen különböznek szinte minden más adatponttól, ***kívülállóknak*** (outlier) nevezzük. A gyakorlatban előforduló kívülálló adatpontok egyik forrása mérési hiba, zaj. Ebben az esetben a kívülálló pontok elhagyása kívánatos. Ezzel szemben ha pontos adatok alapján bizonyul egy adatpont lényegesen különbözőnek az adathalmaztól, akkor ez az információ fontos lehet. Mindkét esetben előnyben részesítjük a kívülállókat elkülönítő klaszterező algoritmusokat. Végül célszerű lehet a klaszterezést a várhatóan homogénebb maradék adatpontokon újra elvégezni, mert a kívülállók kihagyásával a maradék adathalmazt leíró modell, klaszterezés minősége ugrásszerűen javulhat.

Egy klaszterező algoritmustól számos intuitíven többé-kevésbé indokolt formális tulajdonságot követelhetünk meg. Ilyen tulajdonságok például az alábbiak:

Skála-invariancia: ha minden elempár távolsága helyett annak az α-szorosát vesszük alapul (ahol $\alpha > 0$), akkor a klaszterező eljárás eredménye ne változzon meg.

Gazdagság: tetszőleges előre megadott klaszterezéshez létezzen olyan távolságmátrix, hogy a klaszterező eljárás az adott előre megadott partícióra vezessen.

Konzisztencia: tekintsük egy adott adathalmazon alkalmazott klaszterező algoritmus eredményét. Ha ezután a pontok közötti távolságokat úgy változtatjuk meg, hogy tetszőleges, azonos csoportban lévő elempárok között a távolság nem nő, illetve különböző csoportban lévő elempárok közötti távolság nem csökken, akkor az újonnan kapott távolságokra alkalmazott algoritmus az eredetivel megegyező csoportosítást adja.

Meglepő negatív eredmény a következő, legalább kételemű adathalmazokat klaszterező algoritmusokról szóló Kleinberg-tétel, amelyet bizonyítás nélkül közlünk.

32.4. tétel. *Nem létezik olyan klaszterező eljárás, ami egyszerre rendelkezik a skála-invariancia, gazdagság és konzisztencia tulajdonságokkal.*

32.3. Adattípusok és távolságfüggvények

Egy adathalmaz klaszterezésének sikere nagymértékben függ a megfelelő, az adatok hasonlóságát jól modellező hasonlósági, illetve távolság függvények megválasztásától. Adatpontokat alapvetően kétfajta attribútum típussal szokták leírni: *numerikus* és *kategorikus értékekkel*. Numerikus érték esetén az attribútumok valós számok, amelyeken aritmetikai műveleteket és rendezést alkalmazhatunk. Kategorikus adatok esetén csak az adatpontok megfelelő attribútumainak egyenlőségét vagy különbözőségét tudjuk felhasználni az adatpontok közötti távolságok definiálására.

32.3.1. Numerikus adatok

Numerikus attribútumok esetén a kapott vektorok távolságának mérésére a legfontosabb példa az L_p vagy más néven ***Minkowski-metrika.*** Legyen u, v két adatpont (illetve az őket leíró vektorok) és u_i, v_i az attribútumok (koordináták) értékei, ahol $i = 1, \ldots, d$. Ekkor

$$d_p(u, v) = \left(\sum_{i=1}^{d} |u_i - v_i|^p \right)^{1/p}$$

adja a két pont távolságát. A p paraméter értéke elvileg tetszőleges pozitív szám lehet. A legnépszerűbb a $p = 2$ értékhez tartozó, geometriailag és statisztikailag könnyen interpretálható ***euklideszi metrika.*** A $p = 1$ értékhez tartozó metrikát ***Manhattan-távolságnak*** nevezik. A $p \to \infty$ esetben kapjuk a

$$d_{\max}(u, v) = \max_{i \in \{1,\ldots,d\}} |u_i - v_i|$$

maximum távolságot.

Ha az adatpontokat leíró attribútumok nagysága helyett csak azok aránya lényeges, akkor az adatpontok távolságának mérésére a két attribútumvektor által bezárt szög koszinuszát megadó

$$d_{\cos}(u, v) = \frac{u^T v}{||u||_2 ||v||_2} = \frac{\sum_{i=1}^{d} u_i v_i}{\sqrt{\sum_{i=1}^{d} u_i^2 \sum_{i=1}^{d} v_i^2}}$$

koszinusz távolságot alkalmazhatjuk.

32.3.2. Bináris és kategorikus adatok

Kategorikus attribútumok esetén az egyező értéket felvett attribútum számát osztva az összes attribútum számával kapjuk a

$$s_R(u, v) = \frac{|\{i \mid u_i = v_i\}|}{d}$$

Rand-hasonlóságot (másnéven egyszerű illeszkedési hasonlóságot).

Bináris (0-1) értékű attribútumok esetén, amikor a hasonlóság szempontjából csak az 1 értéket felvett attribútumok megegyezése lényeges, a

$$s_J(u,v) = \frac{|\{i \mid u_i = 1 \text{ és } v_i = 1\}|}{|\{i \mid u_i = 1 \text{ vagy } v_i = 1\}|}$$

Jaccard-hasonlóságot használhatjuk.

Kettőnél több értékű kategorikus attribútumok átírhatók bináris attribútumokká úgy, hogy az attribútum minden lehetséges értékéhez hozzárendelünk egy bináris változót, amely 1 lesz, ha az attribútum felveszi az adott értéket, és 0 lesz különben.

Gyakorlatok

32.3-1. Legyen $G = (V, E)$ egy egyszerű gráf. Mutassuk meg, hogy a gráf pontjain értelmezett

$$d(u,v) = \begin{cases} 1, & \text{ha } (u,v) \in E; \\ 2, & \text{különben} \end{cases}$$

függvény metrika.

32.3-2. Legyen d egy metrika és t egy tetszőleges pozitív szám. Mutassuk meg, hogy a

$$d'(u,v) = \frac{d(u,v)}{d(u,v)+t}$$

függvény is metrika.

32.3-3. Mutassuk meg, hogy ha az s a

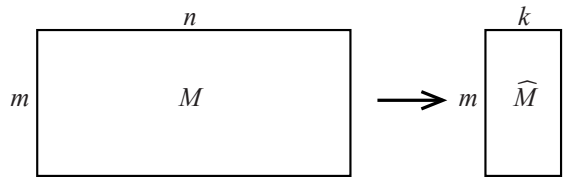
a. Rand-hasonlóságot,

b. Jaccard-hasonlóságot jelöli, akkor az $1 - s$ függvény metrika.

32.4. Dimenzió-csökkentés

A klaszterező algoritmusok alapvető építőeleme egy hatékonyan kiértékelhető, az adatpontok közötti viszonyokat hűen tükröző hasonlóság- vagy távolságfüggvény. A *dimenzió-csökkentés* módszere, melynek során az adatpontok túlságosan sok attribútummal való leírását kevesebb attribútumot használó új leírással helyettesítjük, a korábban bevezetett hasonlósági függvényeken több szempontból is javíthat. A továbbiakban az eredeti adathalmazt az $m \times n$-es M mátrixszal (m az adatpont, n az attribútumok száma), az új leírást pedig az $m \times k$ méretű $\widehat{M}$ mátrixszal (k darab új attribútum, lásd a 32.1. ábrát) reprezentáljuk.

A dimenzió-csökkentésnek eltérő céljai lehetnek. Egyik nyilvánvaló cél az



32.1. ábra. A dimenziócsökkentés célja, hogy az M mátrix mérete jelentősen csökkenjen, azaz $k \ll n$ legyen.

adatbázis tömörítése, a tárolandó adatok mennyiségének csökkentése. Másik lehetséges cél, hogy a csökkentés utáni $\widehat{m}$ mátrix sorai között hatékonyan tudjuk kiértékelni két adatpont hasonlóságát, illetve hogy megfelelő adatszerkezettel támogassuk az olyan lekérdezéseket, amelyek egy adott ponthoz kikeresik a hozzá leghasonlóbbakat. Végül a dimenzió-csökkentés céljai között megemlíthetjük a zajszűrést is. E mögött az a feltételezés áll, hogy az adatpontok egy alacsony dimenziójú térből származnak, és a dimenzió megnövekedését valamilyen véletlen zaj okozza. Az utóbbi esetben a dimenziócsökkentési eljárás után számított hasonlóság értékek pontosabbak lesznek, mint ha az eredeti M mátrixból számítanánk őket.

Az alábbiakban két példát mutatunk dimenzió-csökkentésre; mindkettő jelentősen csökkentheti egy adatbázis méretét. Az első példánál egy lineáris algebrai szempontból optimális dimenzió-csökkentést ismertetünk, melynek zajszűrési tulajdonságait több tapasztalati tény is mutatja. A második ismertetett módszer algoritmikus szempontból érdekes, segítségével nagyban növelhető a hasonlósági lekérdezések hatékonysága.

32.4.1. Szinguláris felbontás

A szinguláris felbontás az elméleti szempontból egyik legtöbbet vizsgált, klasszikus lineáris algebrai eszközöket használó dimenzió-csökkentési eljárás. Ennek alkalmazása után nyert $\widehat{M}$ mátrix soraiból jól közelíthető az euklideszi távolság, illetve az attribútumok vektoraiból számított skalári szorzattal mért hasonlóság. Utóbbi megegyezik a koszinusz mértékkel, ha a mátrix sorai normáltak. Ebben a pontban néhány jelölés és alapvető fogalom után definiáljuk a szinguláris felbontást, igazoljuk a felbontás létezését, majd megmutatjuk, hogy miként használható a felbontás dimenzió-csökkentésre. Megjegyezzük, hogy a szakasz nem mutat a gyakorlatban numerikus szempontból jól alkalmazható módszert a felbontás kiszámítására. Kisebb adathalmaz esetén általános lineáris algebrai programcsomag (Matlab, Octave, Maple) használata

javasolt, míg nagyobb adatbázisoknál az adatok sajátosságát kihasználó szinguláris felbontó program (SVDPack) használata ajánlott.

Egy $U \in \mathbb{R}^{n\times n}$ mátrixot ***ortogonálisnak*** nevezünk, ha oszlopai ortogonális rendszert alkotnak, azaz $U^TU = I_n$, ahol I_n az $n \times n$ méretű egységmátrixot, és U^T az U transzponáltját jelöli. Másképpen mondva U invertálható és U^{-1} inverzére $U^{-1} = U^T$ teljesül. Mátrix ortogonalitásának szemléletes tárgyalásához szükségünk lesz a vektorok hosszának általánosítására, a norma fogalmára. Egy $v \in \mathbb{R}^n$ vektor $\|v\|_2$-vel jelölt ***2-normáját*** a $\|v\|_2 = \sqrt{\sum_i v_i^2}$ egyenlőséggel definiáljuk. Egyszerűen látható, hogy $\|v\|_2{}^2 = v^Tv$ teljesül. A 2-norma általánosítása a tetszőleges $M \in \mathbb{R}^{m\times n}$ mátrix esetén értelmezett $\|M\|_F$ ***Frobenius-norma***, amelynek definíciója $\|M\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n M_{i,j}^2}$.

Visszatérve az ortogonalitás szemléletes jelentésére, egy ortogonális mátrix által reprezentált lineáris transzformációra úgy gondolhatunk, mint egy forgatásra, amely a vektorok hosszát nem változtatja. A szemlélet alapja, hogy tetszőleges $U \in \mathbb{R}^{n\times n}$ ortogonális mátrix és $x \in \mathbb{R}^n$ vektor esetén

$$\|Ux\|_2 = \|x\|_2$$

teljesül. Az azonosság az alábbi elemi lépésekből következik: $\|Ux\|_2^2 = (Ux)^T(Ux) = x^T(U^TU)x = x^Tx = \|x\|_2^2$. Hasonlóan belátható, hogy tetszőleges $X \in \mathbb{R}^{m\times n}$ mátrix esetén és $U \in \mathbb{R}^{m\times m}$ illetve $V \in \mathbb{R}^{n\times n}$ ortogonális mátrixok esetén igaz, hogy

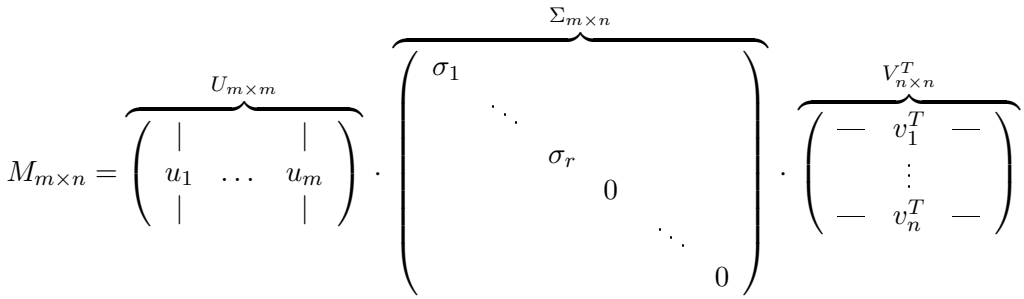
$$\left\|UXV^T\right\|_F = \|X\|_F \ .$$

A rövid bevezető után rátérünk a szinguláris felbontás definíciójára. Egy nem szükségszerűen négyzetes $M \in \mathbb{R}^{m\times n}$ mátrix ***szinguláris érték felbontásán*** (singular value decomposition, SVD) az olyan

$$M = U\Sigma V^T$$

szorzattá bontást értjük, ahol $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ ortogonális mátrixok, továbbá a Σ mátrix M-mel megegyező méretű és a bal felső sarokból 45°-ban lefele elhelyezkedő $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ pozitív számokat csupa 0 követ és a többi elem szintén 0. A σ_i számokat ***szinguláris értékeknek*** nevezzük, és a $\sigma_i = 0$ választással terjesztjük ki az $i > r$ esetre. A felbontásból látható, hogy $\mathrm{rang}(M) = \mathrm{rang}(\Sigma) = r$.

Az U és a V oszlopait ***bal-, illetve jobboldali szinguláris vektoroknak*** mondjuk. A jelölések áttekintése a 32.2. ábrán látható.



32.2. ábra. A szinguláris felbontás sematikus vázlata.

32.5. tétel. *Tetszőleges $M \in \mathbb{R}^{m\times n}$ mátrixnak létezik szinguláris érték felbontása, azaz léteznek $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ ortogonális mátrixok, melyekkel*

$$M = U\Sigma V^T,$$

ahol

$$\Sigma \in \mathbb{R}^{m\times n}, \qquad \Sigma = \begin{pmatrix} \Sigma_+ & 0 \\ 0 & 0 \end{pmatrix},$$

továbbá Σ^+ egy $r \times r$ méretű diagonális mátrix, amelynek főátlójában a $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ számok helyezkednek el sorrendben.

Bizonyítás. Az M^TM mátrix szimmetrikus, ezért ortogonális transzformációval diagonalizálható és sajátértékei valósak. Továbbá pozitív szemidefinit, mert tetszőleges $x \in \mathbb{R}^{n\times n}$ vektor esetén $x^TM^TMx = (Mx)^T(Mx) = \|Mx\|_2^2 \geq 0$, ezért a sajátértékek nem negatívak. A sajátértékek legyenek $\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_r^2 > 0$. Az ezekhez tartozó sajátvektorokból alkotott ortogonális mátrixot jelölje V, ekkor

$$V^TM^TMV = \begin{pmatrix} \Sigma_+^2 & 0 \\ 0 & 0 \end{pmatrix}.$$

A mátrixot két részre osztva $V = (V_r\, V_2)$, ahol $V_r \in \mathbb{R}^{n\times r}$ a pozitív sajátértékhez tartozó sajátvektorokat tartalmazza. Vagyis

$$V_r^TM^TMV_r = \Sigma_+^2\,.$$

Vezessük be az

$$U_r = MV_r\Sigma_+^{-1}$$

jelölést, ekkor

$$M = U_r \Sigma_+ V_r^T.$$

Az U_r vektorai ortogonális vektorrendszert alkotnak, ezt tetszőlegesen kiegészítve $U = (U_r\, U_2)$ ortogonális mátrixszá a

$$M = U \begin{pmatrix} \Sigma_+ & 0 \\ 0 & 0 \end{pmatrix} V^T$$

eredményt kapjuk. ■

Most megmutatjuk, hogy szinguláris felbontás segítségével hogyan lehet dimenzió-csökkentést végrehajtani. Emlékeztetünk rá, hogy az M mátrix n-dimenziós sorvektorai adatpontokat jellemeznek. Dimenzió-csökkentéskor az n attribútumot szeretnénk $k < n$ dimenziójú vektorokkal jellemezni úgy, hogy közben az adatpontok euklideszi távolsága vagy skaláris szorzattal mért hasonlósága csak kis mértékben változzon. A mátrixszorzás elemi tulajdonsága, hogy a szinguláris felbontás az alábbi formában is írható.

$$M = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T,$$

ahol $u_i v_i^T$ a bal- illetve a jobboldali szinguláris vektorokból képzett diádszorzat, azaz egy oszlop- és egy sorvektor szorzataként felírt $m \times n$ méretű 1-rangú mátrix. Látható, hogy az $u_i v_i^T$ diádok monoton csökkenő σ_i súllyal szerepelnek az összegben. Innen adódik az ötlet, hogy $k < r$ esetén csak az első k legnagyobb súlyú diád összegével közelítsük az M mátrixot. Azaz

$$M_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T = U_k \Sigma_k V_k^T,$$

ahol $U_k = (u_1\, u_2\, \ldots\, u_k)$ és $V_k = (v_1\, v_2\, \ldots\, v_k)$, valamit Σ_k egy $k \times k$ méretű diagonális mátrix, melynek főátlójában a $\sigma_1, \sigma_2, \ldots, \sigma_k$ értékek vannak. Könnyen látható, hogy M_k sorai egy k-dimenziós altérben helyezkednek el, hiszen $\operatorname{rang}(M_k) = \operatorname{rang}(\Sigma_k) = k$. Sokkal mélyebb eredmény a következő, melynek bizonyítását mellőzzük.

32.6. tétel. *Legyen M egy legalább k rangú mátrix és legyen M_k a fenti módon számított közelítése. Ha a közelítés hibáját Frobenius-normával mérjük, akkor a k-rangú mátrixok közül az M_k mátrix a lehető legjobban közelíti M-et, azaz*

$$\|M - M_k\|_F = \min_{N:\, \operatorname{rang}(N)=k} \|M - N\|_F .$$

Továbbá a közelítés hibája a σ_i szinguláris értékekkel kifejezhető:

$$\|M - M_k\|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^2} \,.$$

Az M_k mátrix sorai az M-éhez hasonlóan n méretűek, de most már egy k-dimenziós altérnek az elemei. Ennek az altérnek egy bázisát alkotják a V_k^T sorai, és az

$$M' = U_k \Sigma_k$$

mátrix k-dimenziós sorvektorai e bázisban fejezik ki az M_k sorait. Tehát a dimenzió-csökkentés eredménye, hogy az M mátrix n-dimenziós sorait a vetítés után az M' mátrix k-dimenziós soraival közelítjük. A V_k^T sorainak ortogonalitásából könnyen belátható, hogy az M_k, illetve az M' soraiból számított euklideszi távolságok és skaláris szorzatok is megegyeznek. Tehát a közelítés alatt torzítás kizárólag az M-ből M_k-ba történő vetítés során történik, melynek mértéke a 32.6. tétel alapján felülről becsülhető.

Gyakorlatok

32.4-1. *a.* Igazoljuk, hogy ha az u és v vektorokra $\|u\|_2 = \|v\|_2 = 1$ teljesül, akkor az uv^T diádszorzatra is fennáll, hogy $\left\|uv^T\right\|_F = 1$.

b. Bizonyítsuk be, hogy tetszőleges M mátrix esetén, ahol M szinguláris értékeit σ_i jelöli,

$$\|M\|_F = \sqrt{\sum_{i=1}^{r} \sigma_i^2} \,.$$

c. Lássuk be a 32.6. tétel második egyenlőségét.

32.4-2. Az M mátrix sorai által reprezentált adatpontok hasonlóságát skaláris szorzattal számítjuk. Az $M \cdot M^T$ mátrix *hasonlóságmátrix* lesz, mert elemei az egyes adatpontok közti hasonlóságok értékeivel egyeznek meg. Fejezzük ki az M' mátrixból számított hasonlóságmátrix Frobenius-normában mért hibáját a σ_i értékekből és k-ból, ahol a jelölések megegyeznek a 32.6. tételben használtakkal.

32.4-3. Legyen M egy $m \times n$ méretű mátrix, jelölje A az alábbi $(m+n) \times (m+n)$ méretű mátrixot.

$$A = \begin{pmatrix} 0 & M \\ M^T & 0 \end{pmatrix}$$

Az A mátrix szimmetrikus, ezért tudjuk, hogy sajátértékei valósak, és sajátvektorai ortogonális rendszert alkotnak. Azaz létezik az $A = WDW^T$ szorzattá bontás, ahol W ortogonális és D diagonális mátrix, melynek

főátlójában valós számok szerepelnek.

a. Igazoljuk, hogy A pozitív sajátértékei megegyeznek M szinguláris értékeivel.

b. Lássuk be, hogy ha $\lambda > 0$ sajátértéke A-nak, akkor $-\lambda$ is az.

c. Adjunk új bizonyítást a 32.5. tételre felhasználva, hogy az A-nak létezik az $A = WDW^T$ sajátérték-felbontása.

32.4.2. Ujjlenyomat alapú dimenzió-csökkentés

Ebben a pontban egy véletlen mintavételezésen alapuló, elemi eszközöket használó dimenzió-csökkentési eljárást tárgyalunk, melynek segítségével bináris attribútumok által jellemzett adatpontok Jaccard-hasonlóságát lehet közelíteni. Először ismertetjük, hogy miként lehet a dimenziót csökkenteni, majd a rövidebb vektorokból hasonlóságot számítani. Utána megmutatjuk a módszer igazi erejét, egy olyan hasonlóság-keresési eljárást, amely külső táron elhelyezett adatbázison is működik, lekérdezési időben korlátozott számú adatbázis-hozzáférést igényel, ezért különösen nagy adatbázisokon is alkalmazható.

Tegyük fel, hogy az $M \in \{0,1\}^{m \times n}$ mátrix m sora m különböző adatpontot jellemez egyenként n darab bináris attribútummal. Jelöljük $X(i)$-vel az i-edik sorban elhelyezkedő egyesek indexeinek halmazát, azaz

$$X(i) = \{\ j\ :\ M_{ij} = 1\ \}\ .$$

A továbbiakban feltesszük, hogy $X(i) \neq \emptyset$. Az i_1 és i_2 adatpontok $s_J(i_1, i_2)$ Jaccard-hasonlóságát a korábbi definíció alapján az alábbi formában is írhatjuk:

$$s_J(i_1, i_2) = \frac{|X(i_1) \cap X(i_2)|}{|X(i_1) \cup X(i_2)|}\ .$$

Most egy olyan eljárást ismertetünk, amely minden adatponthoz egy véletlen ujjlenyomatot rendel úgy, hogy az i_1 és i_2 ujjlenyomata pont $s_J(i_1, i_2)$ valószínűséggel egyezzen meg. Legyen σ az attribútumok $1, 2, \cdots, n$ indexeinek egy véletlen permutációja, amelyet az $n!$ darab permutáció közül egyenletes eloszlás szerint választunk ki. Az i-edik adatponthoz tartozó $u(i)$ *ujjlenyomatot* az alábbiak szerint választjuk meg:

$$u(i) = \operatorname*{argmin}_{j \in X(i)} \sigma(j)$$

azaz $u(i) = j$, ahol j az $M_{ij} = 1$ egyenlőséget kielégítő indexek közül a σ permutáció alkalmazása után a legkisebb értéket kapja.

32.7. tétel. *A tetszőleges i_1 és i_2 adatpontok ujjlenyomatai $s_J(i_1, i_2)$ valószínűséggel egyeznek meg, azaz*

$$\Pr\{u(i_1) = u(i_2)\} = s_J(i_1, i_2) \; .$$

Bizonyítás. Az $u(i_1) = u(i_2)$ egyenlőség pontosan akkor teljesül, ha

$$\operatorname*{argmin}_{j \in X(i_1) \cup X(i_2)} \sigma(j) \in X(i_1) \cap X(i_2).$$

Az utóbbi esemény valószínűsége $|X(i_1) \cap X(i_2)| \,/\, |X(i_1) \cup X(i_2)| = s_J(i_1, i_2)$, mivel bármelyik $X(i_1) \cup X(i_2)$-beli elem azonos valószínűséggel lesz a permutáció alkalmazása után minimális. ■

Az előző állítás alkalmazásával egy olyan algoritmust mutatunk, amely független minták átlagaként becsüli a hasonlóság-értékeket.

Először a dimenzió-csökkentő lépés során minden adatponthoz k darab ujjlenyomatot generálunk, ezzel az $M \in \{0,1\}^{m \times n}$ mátrixból egy $M' \in \{1, 2, \ldots, n\}^{m \times k}$ mátrixot készítünk az alábbi pszeudokód szerint. Fontos, hogy a $\sigma_1, \sigma_2, \ldots, \sigma_k$ véletlen permutációkat egymástól függetlenül válasszuk meg.

UJJLENYOMAT(M,k)

1 **for** $\ell \leftarrow 1$ **to** k
2 **do** $\sigma_\ell =$ az $1, 2, \cdots, n$ indexek egy véletlen permutációja
3 $M'_{i\ell} = \operatorname{argmin}_{j \,:\, M_{ij}=1} \sigma_\ell(j)$
4 **return** M'

Ha az ujjlenyomatokat már elkészítettük, és az i_1 és i_2 adatpontok hasonlóságát szeretnénk kiértékelni, akkor összeszámoljuk, hogy hány pozícióban egyezik meg az M' mátrix i_1-ik és i_2-ik sora, majd az egyezések számát osztjuk k-val, azaz a

$$\frac{\left|\{\ \ell \,:\, M'_{i_1\ell} = M'_{i_2\ell}\ \}\right|}{k}$$

mennyiséggel becsüljük az $s_J(i_1, i_2)$ hasonlóságot. Bizonyítás nélkül megemlítjük, hogy bármely rögzített $\delta > 0$ esetén annak a valószínűsége, hogy a becsült hasonlóság a valóditól legalább δ értékkel eltér, k növelésével exponenciálisan tart 0-hoz.

Megjegyezzük, hogy n elem véletlen permutációjának kiszámítása, tárolása és a permutáció által meghatározott rendezés hatékony lekérdezése nem könnyű feladat nagy n esetén. A gyakorlatban nem az összes lehetséges $n!$-féle permutáció közül választunk, hanem permutációk egy kisebb,

tömören reprezentálható családjából. Egy lehetséges megoldás, hogy választunk egy $p \gg n$ prímet. Ekkor a véletlen permutációt úgy kapjuk, hogy az a, b számokat egyenletesen véletlenül generáljuk az $[1, p-1]$ intervallumból, és az $j \in \{1, 2, \cdots, n\}$ indexek σ permutációbeli sorrendjét az $a \cdot j + b \pmod p$ értékek sorrendjével határozzuk meg. Rögzített p esetén a permutációk családjának mérete $(p-1)^2$, és egy permutáció tárolásához elég az a és b számokat tárolni. A módszer előnye, hogy a permutáció tömören reprezentálható; hátránya, hogy nem csak egy kisebb családból választ permutációt, továbbá a nagy egészekkel való aritmetika lassítja az ujjlenyomatok kiszámítását.

A szakasz befejezéseként egy egyszerű eljárást mutatunk a *hasonlóságkeresési* feladatra, melynek bemenete egy $i_q \in \{1, 2, \cdots, m\}$ adatpontként adott lekérdezés és egy $\alpha > 0$ küszöb, a kimenet pedig az olyan adatpontok halmaza, melyek az α küszöbnél hasonlóbbak i_q-hoz. Formálisan megfogalmazva tehát az $\{\ i\ :\ s_J(i_q, i) > \alpha\ \}$ elemeit keressük. Kézenfekvő, de nem hatékony megoldás, hogy az i_q adatpontot összehasonlítjuk az összes többivel az M' mátrix alapján. Ez a megoldás különösen sokáig tart, ha az M' mátrix csak a háttértáron fér el.

A következő algoritmusban nem követeljük meg, hogy M' mátrix a memóriában legyen. Az M' elemeit most egy adatbázisban tároljuk, amely a háttértárról az $M'[i]$ lekérdezés hatására betölti az i adatponthoz tartozó k darab ujjlenyomatot a memóriába. A hasonlósági lekérdezések előtt elkészítjük az I adatbázist is, amely az $I[\ell][j]$ lekérdezés hatására azon i indexeket adja vissza, amelyeknek ℓ-ik ujjlenyomata j. Az I adatbázis M'-ből úgy számítható, hogy M' elemeit ℓ és j szerint lexikografikusan rendezzük. Háttértáras rendezés alkalmazásával ez akkor is megtehető, ha M' nem fér el a memóriában.

A következő algoritmus egyszer fér hozzá M'-höz, k-szor az I-hez, és eközben megszámolja, hogy a többi adatpontnak az adott i_q-val hány közös ujjlenyomata van. Az egyes adatpontokkal közös ujjlenyomatok számát a H hash-táblában tároljuk azon adatpontok esetén, amelyekkel már találtunk közös ujjlenyomatot. Az alábbi pszeudokódban $H[i] = 0$, ha i nincs benne még a hash-táblában.

```
HASONLÓ(i_q, α)
1  u ← M'[i_q]
2  for ℓ ← 1 to k
```

```
3       for i ∈ I[ℓ][u_ℓ]
4           H[i] ← H[i] + 1
5   return { i : H[i]/k > α }
```

A dimenzió-csökkentés eredményeként egyrészt sikerült M helyett a k által szabályozható méretű M' mátrixba tömöríteni az adatbázist. Másrészt a hasonlóság keresési problémára is hatékonyan, mindössze $k+1$ tömbelem hozzáféréssel tudunk válaszolni.

Gyakorlatok

32.4-1. Javasoljunk olyan módszert véletlen ujjlenyomat generálására, hogy az i_1 és i_2 adatpontok esetén az ujjlenyomatok egybeesésének valószínűsége az

$$\frac{|X(i_1) \cap X(i_2)|}{|X(i_1)| \cdot |X(i_2)|}$$

érték legyen.

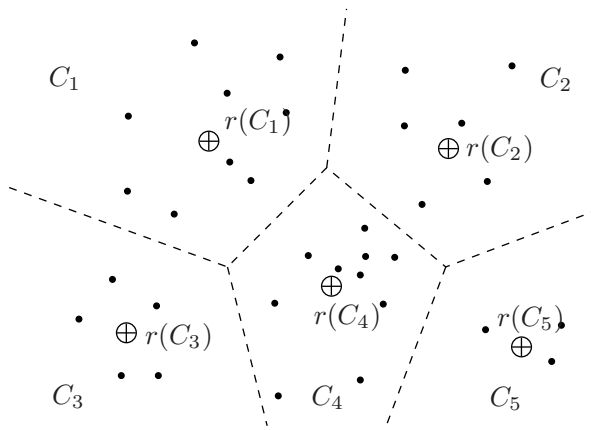
32.5. Particionáló klaszterező algoritmusok

A *particionáló algoritmusok* alapgondolata, hogy a megfelelő klaszterezést a pillanatnyi eredményként kapott klaszterezés folyamatos pontosításával iterálva érjük el. Az iteráció egy fázisában a klasztereket tömören reprezentáljuk úgy, hogy a reprezentáció segítségével a reprezentált klaszterek és az adatpontok közötti távolságokat értelmezni és hatékonyan számolni tudjuk. Ezek után az adatpontokat újból particionáljuk, minden adatpontot hozzárendelünk a hozzá legközelebb eső klaszterhez. Az algoritmus akkor ér véget, ha az iterációs lépés során nem (vagy csak alig) változik meg a partíció, illetve a klaszterek reprezentánsai.

Egy particionáló klaszterező algoritmus elvileg tetszőleges reprezentációból, illetve a hozzátartozó partícióból kiindulhat, de általában célszerű valamilyen egyszerű (klaszterező) heurisztikát alkalmazni a kezdeti klaszterreprezentánsok meghatározására vagy véletlenszerűen választani azokat. A kezdeti partíció megadásával az osztályok k száma is adott lesz. Ez az szám az itt bemutatandó particionáló klaszterező algoritmusok futása során nem változik.

32.5.1. k-közép

A k-KÖZÉP (k-means) algoritmus az egyik legrégebbi és legegyszerűbb klaszterező algoritmus. Feltesszük, hogy az adatpontok egy vektortérben



32.3. ábra. A k-KÖZÉP klaszterező algoritmus egy fázisa: a C_i klasztereket az $r(C_i)$ súlypontjaik reprezentálják.

helyezkednek el. Ekkor a klasztereket súlypontjukkal, középpontjukkal (innen az algoritmus elnevezése) reprezentáljuk, azaz az adott klasztert az adatpontjaihoz tartozó vektorok átlagával reprezentáljuk. Az algoritmus olyan C klaszterezést keres, ahol az adatpontoknak a klaszterük $r(C_i)$ reprezentánsától mért

$$E(C) = \sum_{i=1}^{k} \sum_{u \in C_i} d(u, r(C_i))^2$$

négyzetes távolságösszege minimális. Egy adott C klaszterezés esetén az $E(C)$ mennyiségre a későbbiekben a klaszterezés ***hibájaként*** fogunk hivatkozni.

Az algoritmus menete a következő. Legyen k a felhasználó által előre megadott klaszterszám. Ezután k darab tetszőleges elemet választunk az adatpontok vektorterében (célszerűen azok konvex burkában). Az iteráció első fázisában ezek a pontok reprezentálják a kezdeti k darab klasztert. Ezután minden adatpontot a legközelebbi reprezentánssal rendelkező klaszterhez sorolunk be. A besorolás során kialakult klaszterek új reprezentáns pontjai az új klaszterek középpontjai lesznek (lásd a 32.3. ábrát). A besorolás, középpont választás lépéseket addig iteráljuk, amíg a reprezentánsok rendszere változik. Akkor állunk meg, amikor a klaszterek elemei, illetve középpontjai már nem változnak meg az iterációs lépéstől.

Jelölje D az adatpontok halmazát egy euklideszi vektortérben. Legyen k az előre adott klaszterszám. A k-KÖZÉP algoritmus pszeudokódja a következő:

k-KÖZÉP(D, k)

1 $\{r(C_1), r(C_2), \ldots, r(C_k)\} \leftarrow$ reprezentánsok tetszőleges k elemű kezdeti halmaza
2 **while** az $r(C_i)$ reprezentánsok rendszere változik
3 keyfor $i \leftarrow 1$ **to** k
4 $r(C_i) \leftarrow \frac{1}{|C_i|} \sum_{u \in C_i} u$
5 **for** minden $u \in D$
6 u legyen az $\operatorname{argmin}_i d(u, r(C_i))$ indexű klaszterben
7 $C \leftarrow$ az új klaszterezés
8 **return** C

Az algoritmus jól alkalmazható, ha a „valódi” klaszterek konvex burkai diszjunktak. Fontos előnye, hogy egyszerűen megvalósítható. Futási ideje $O(nkt)$ darab távolságszámítás, ahol n az adatpontok, k a klaszterek és t az iterációk számát jelöli. Látható, hogy az algoritmus futási ideje nagyban függ az alkalmazott távolságszámítástól. A gyakorlatban az algoritmus fenti alapváltozata kis (tízes nagyságrendű) k és t értékeket igényel. Az algoritmus további előnye, hogy nem érzékeny az adatpontok sorrendjére és alkalmazható L_p terekben is.

A k-KÖZÉP algoritmus egyik hátránya, hogy az algoritmus a hiba *lokális minimumában* is megállhat, azaz előfordulhat, hogy egy újabb iteráció során nem változik meg a partíció, mégis létezik olyan klaszterezés, aminek hibája kisebb, mint az algoritmus által adott partícióé. Gyakorlatban a lokális optimumba kerülés esélyének csökkentése érdekében érdemes az algoritmust többször futtatni különböző kezdeti középpontokkal. Azt a klaszterezést fogadjuk el a többszöri futások végeredményei közül, amelyik hibája a legkisebb hibát adja.

Megmutatjuk, hogy a k-KÖZÉP algoritmus érzékeny a kívülálló pontokra. Ebből a célból bevezetjük a Voronoi-tartomány fogalmát. Egy ponthalmaz u pontjának ***Voronoi-tartománya*** álljon azon pontokból, amelyeknek u-tól vett távolsága nem nagyobb, mint bármely más ponthalmazbeli ponttól vett távolsága. A k-KÖZÉP algoritmus során a klaszterek a reprezentáló középpontok által kialakított konvex *Voronoi-tartományokba* esnek, ahogyan ez a 32.3. ábrán is látható. Mivel a klaszterek a Voronoi-tartományok belsejében alakulnak ki, ezért a k-KÖZÉP eredményében a klaszterek konvex burkai diszjunktak lesznek. Az algoritmus tehát a nyilvánvalóan kívülálló pontokat is besorolja valamelyik klaszterbe, és ezek a pontok a klaszter-középpont meghatározásakor indokolatlan torzító hatást okoznak.

További problémák az algoritmussal, hogy a megfelelő k klaszterszám meg-

találása többszöri futtatást igényelhet, az iterációk során szélsőséges klaszterméretek alakulhatnak ki és a végeredmény nagyban függ a kezdeti partíciótól.

A k-KÖZÉP egyik, a gyakorlatban jobb minőségű eredményt adó módosítása a következő. A pontokat egy rögzített sorrend szerint ciklikusan vizsgálva meghatározzuk a vizsgált ponthoz legközelebbi klaszter-középpontot, majd ha szükséges, átsoroljuk az adatpontot a hozzá közelebbi középpontú klaszterbe. Átsorolás esetén kiszámoljuk a két érintett klaszter új középpontjait. Az alapváltozathoz hasonlóan akkor állunk meg az algoritmussal, ha már nem változnak a klaszterek. Az eljárás pszeudokódja a következő.

k-KÖZÉP VÁLTOZAT(D, k)

1 $\{r(C_1), r(C_2), \ldots, r(C_k)\} \leftarrow$ reprezentánsok tetszőleges k elemű kezdeti halmaza
2 **while** $r(C_i)$ reprezentánsok változnak
3 **do for** minden $u \in D$ adatpontra egy rögzített sorrend szerint
4 **do** $h \leftarrow u$ klaszterének indexe
5 $j \leftarrow \operatorname{argmin}_i d(u, r(C_i))$
6 **if** $h \neq j$
7 **then** $C_j \leftarrow C_j \cup \{u\}$
8 $C_h \leftarrow C_h \setminus \{u\}$
9 $r(C_j) \leftarrow \dfrac{1}{|C_j|} \displaystyle\sum_{v \in C_j} v$
10 $r(C_h) \leftarrow \dfrac{1}{|C_h|} \displaystyle\sum_{v \in C_h} u$
11 **return** C

Meglepő módon euklideszi vektortérben adott pontok esetén a fenti k-KÖZÉP VÁLTOZAT futtatása ugyanannyi távolságszámítást igényel, mint az k-KÖZÉP alapváltozat. További algoritmus változathoz jutunk, ha a pszeudokód 3. sorát megváltoztatva a pontokat nem egy rögzített sorrend szerint, hanem véletlenül választva vizsgáljuk.

32.5.2. k-medoid

A k-KÖZÉP klaszterező algoritmus csak olyan adatpontok csoportosítására használható, ahol értelmezhető a klaszterek középpontja, például a pontok vektortérben vannak megadva. Így a k-KÖZÉP nem használható olyan alkalmazásokban, ahol az adatpontok nem attribútumokkal adottak, vagy amikor az adatpontokat leíró attribútumok között kategorikus értékek is szerepelnek. Ilyen helyzetekben a k-MEDOID algoritmus használható, mert futása során

csak az adatpontok közötti távolságok összehasonlítását használja.

A k-MEDOID módszer a klasztereket egy-egy, az adott klaszter elemeihez legközelebbi adatponttal, a ***medoiddal*** reprezentálja. A k-KÖZÉP módszerhez hasonlóan itt is egy távolságnégyzetek összegén alapuló függvényt minimalizálunk, amit az összes adatpont és klaszterük medoidjának távolságnégyzetét összegezve számolunk ki:

$$E(C) = \sum_{i=1}^{k} \sum_{u \in C_i} d(u, m_i)^2 \ ,$$

ahol m_i a C_i klaszter medoidja és d az adatpontokon értelmezett metrika.

A k-MEDOID algoritmus menete megegyezik a k-KÖZÉP menetével. Kezdetben választunk k darab tetszőleges adatpontot, ezek lesznek először a medoidok. Ezután elkezdődik az adatpontok medoidokhoz történő hozzárendelése, illetve az új medoidok választásából álló iteratív folyamat. Az iteráció során a k-KÖZÉP algoritmus módosításánál látott stratégiához hasonló módszert célszerű követni. Tekintsünk egy véletlen nem medoid v adatpontot és vizsgáljuk meg, hogy van-e olyan klaszter, amelyet a v pont jobban reprezentál, mint a klaszter jelenlegi medoidja (az új medoid nem feltétlenül a klaszter eleme, de klaszterezendő adatpont). Ezt a lépést a következőképpen valósíthatjuk meg. Az összes m_i medoidra meghatározzuk, hogy mennyivel változna a klaszterezés jósága, ha v átvenné az m_i szerepét. Az algoritmus menetét egyszerűsíti, hogy a hiba változására azok az elemek nincsenek hatással, amelyekhez létezik v-nél és m_i-nél is közelebbi medoid. Ha találunk olyan m_i-t, amellyel v-t cserélve $E(C)$ csökken, akkor javítunk a klaszterezésen; azt a cserét hajtjuk mindig végre, amellyel pillanatnyilag a legnagyobb csökkenést érjük el.

A pszeudokódból látható, hogy a számoláshoz csak az adatpontok egymástól mért távolságainak értékeit használjuk. Az iteráció egy fázisához $O(nk^2)$ távolság kiszámítása kell, mert minden potenciális m_i-ről v-re történő medoid csere esetén a hiba változásának kiszámolásához $n \times k$ darab pont-medoid pár távolságára van szükségünk. A teljes futási időt az határozza meg, hogy az adatpontok számának növekedésével lineárisan nő a fázisok szükséges száma. Fontos előnye a k-MEDOID módszernek, hogy a k-középhez képest kevésbé érzékeny a kívülálló pontokra.

A pszeudokód a következő:

k-MEDOID(D, k)

```
1  Adjunk meg k darab tetszőleges m_i ∈ D (i = 1,...,k)
                                   adatpontot.        // A első medoidok.
2  while C_i osztályok változnak
3        for minden u ∈ D adatpont
4             u-t tegyünk be az j = argmin_i d(u, m_i) indexű C_j klaszterbe.
5             v ← véletlen adatpont, ahol v ≠ m_i minden i-re
6             for i ← 1 to k
7                  Δ_i ← az E(C) hiba változása,
                   ha az m_i medoidot v-re cseréljük
8             if min_i Δ_i < 0
9                v legyen medoid m_j helyett, ahol j = argmin_i Δ_i
10            C ← az új klaszterezés
11 return C
```

Gyakorlatok

32.5-1. Mutassunk példát arra, hogy a k-KÖZÉP algoritmus által kimenetként adott klaszterezés hibája nem a hiba globális minimuma.

32.6. Hierarchikus eljárások

A *hierarchikus* eljárások onnan kapták a nevüket, hogy az elemeket egy hierarchikus adatszerkezetbe (fába, dendogramba, taxonómiába) rendezik el. Az adatpontok a fa leveleiben találhatók. A fa minden belső pontja egy klaszternek felel meg, amely azokat a pontokat tartalmazza, melyek a fában alatta találhatók. A gyökér az összes adatpontot tartalmazza.

32.6.1. Felhalmozó és lebontó módszerek

A hierarchikus klaszterezési módszereket két csoportra bonthatjuk: a ***felhalmozó*** (egyesítő, bottom-up) és ***lebontó*** (felosztó, top-down) módszerek. A lentről felfelé építkező *felhalmozó* eljárásoknál kezdetben minden elem különálló klaszter, majd az eljárás a legközelebbi klasztereket egyesíti, a hierarchiában egy szinttel feljebb újabb klasztert alakítva ki. A fentről lefelé építkező *lebontó* módszerek fordítva működnek: egyetlen, minden adatpontot tartalmazó klaszterből indulnak ki, amit kisebb klaszterekre particionálnak, majd ezeket is tovább bontják.

A kapott hierarchiából számos klaszterezést tudunk kinyerni, amennyiben például a futás során kialakult a fa egy-egy szintjén található pontokat tek-

intjük. Az így kinyert klaszterezések egymás finomításai lesznek. Általánosabban választhatjuk a fa gyökere és levelei közötti tetszőleges, tartalmazásra minimális elvágó ponthalmaz elemeit; az ezekhez rendelt klaszterek a ponthalmaz egy partícióját alkotják.

Általában tetszőleges hasonlóság vagy távolság fogalomra építve könnyen kidolgozható hierarchikus algoritmus, így bármilyen típusú adat klaszterezéséhez több rugalmas és többnyire hatékony módszerhez juthatunk.

A hierarchikus algoritmusok kulcslépése az egyesítendő vagy osztandó klaszterek kiválasztása. Miután egy *egyesítés* (*osztás*) megtörténik, minden további műveletet az új klasztereken végezzük el. Tehát egyik művelet sem fordítható meg, így egy rossz választás később nem javítható ki.

Nehézséget jelent annak eldöntése, hogy mennyire, milyen magas szintig épüljön fel a hierarchikus klaszterezés fája, illetve az elkészült dendogram melyik vágásával tudjuk a feladatot legjobban megoldó klaszterezést kiválasztani.

A legegyszerűbb felhalmozó hierarchikus klaszterező eljárás az alábbi. Legyen D az adatpontok halmaza, $d_*(C_i, C_j)$ két klaszter távolságát mérő függvény és $S(C)$ egy megállási szabály. Kezdetben minden pont különálló egyelemű klaszterhez tartozik. Keressük meg, majd egyesítsük a két, d_* szerint legközelebbi klasztert. Iteráljuk ezt a lépést, amíg $S(C)$ meg nem állítja az eljárást. Például $S(C)$ definiálható úgy, hogy az algoritmus akkor álljon meg, amikor a $|C|$ vagy d_* aktuális értéke elér egy előre adott küszöböt.

EGYSZERŰ-FELHALMOZÓ(D, S)

1 $C \leftarrow \{\{u\} | u \in D\}$
2 **while** $S(C)$ engedi
3 $\quad (a, b) \leftarrow \text{argmin}_{(i,j)}\, d_*(C_i, C_j)$
4 $\quad C \leftarrow C \cup \{C_a \cup C_b\} \setminus \{C_a, C_b\}$
5 **return** C

32.6.2. Klasztertávolságok mértékei

A d_* klaszterek távolságát mérő függvény megválasztásától függően különböző hierarchikus klaszterező algoritmusokat kapunk. Mivel a klaszterezés bemenete az adatpontok d távolságfüggvénye, ezen értékekből számíthatjuk d_*-ot. A továbbiakban d_* leggyakrabban használt megvalósításait mutatjuk be.

Amennyiben két klaszter d_{min} távolságát azok legközelebbi pontjainak távolságával definiáljuk, akkor ***legkisebb távolságot*** (single linkage) használó

eljárásról beszélünk. Ekkor

$$d_{min}(C_i, C_j) = \min_{\substack{u \in C_i \\ v \in C_j}} d(u, v) \ .$$

Általános esetben, távolságmátrixot használva az eljárás $O(n^2)$ összehasonlítást igényel.

A legkisebb távolságon alapuló algoritmusok alkalmasak jól elkülönülő, tetszőleges alakú klaszterek felfedezésére. Mivel a klaszterelemek minimális távolságát veszik figyelembe, hajlamosak azonban összekötni két klasztert, ha egy-egy elemük túl közel kerül egymáshoz. További hibája a módszernek, hogy érzékeny az kívülállókra.

Ha egy legkisebb távolságon alapuló klaszterzést gráfelméleti algoritmusként tekintünk úgy, hogy a gráf pontjai az adatpontok és az éleken adott költségeknek pedig a távolságok felelnek meg, akkor algoritmusunk a minimális költségű feszítőfát építő *Kruskal-algoritmussal* lesz azonos. A Kruskal-algoritmus megvalósításához nagyon hatékony adatstruktúrák ismertek. Amennyiben a távolságok *ritka mátrix formátumban* adottak, az algoritmus $O(e\alpha(e))$ futási idővel is megvalósítható, ahol e az élek száma és α az inverz Ackermann-függvényt[1] jelöli.

Ha a

$$d_{max}(C_i, C_j) = \max_{\substack{u \in C_i \\ v \in C_j}} d(u, v)$$

klasztertávolságot használjuk, akkor a ***legnagyobb távolságon*** (complete linkage) alapuló eljárást kapjuk. További klaszterek közötti távolságfüggvényt ad a

$$d_{avg}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{\substack{u \in C_i \\ v \in C_j}} d(u, v)$$

átlagos távolság.

Ha az adatpontok vektortérbeliek és d a vektortér metrikája, akkor d_* a klaszterközéppontok távolságaként értelmezhető:

$$d_{mean}(C_i, C_j) = d\left(\frac{1}{|C_i|} \sum_{u \in C_i} u, \frac{1}{|C_j|} \sum_{v \in C_j} v \right) \ .$$

[1] Az $A(n) = A(n, n)$ Ackermann-függvény a következő rekurzióval adható meg:

$$A(n, m) = \begin{cases} m + 1, & \text{ha } n = 0, \\ A(n-1, 1), & \text{ha } n > 0 \text{ és } m = 0, \\ A\left(n-1, A(n; m-1)\right), & \text{ha } n > 0 \text{ és } m > 0 \ . \end{cases}$$

Az $\alpha(e) = \min\{n : A(n) \geq e\}$ inverz Ackermann-függvény rendkívül lassan tart a végtelenbe, a gyakorlatban előforduló esetekben értéke legfeljebb 4 lesz.

További hagyományos klasztermetrika az átlagos távolsághoz hasonló ***Ward-távolság***

$$d_{Ward}(C_i, C_j) = \sum_{u,v \in C_i \cup C_j} d^2(u,v) - \left(\sum_{u,v \in C_i} d^2(u,v) + \sum_{u,v \in C_j} d^2(u,v) \right) .$$

Ward módszere a klasztereken belüli távolságnégyzetek összegének minimalizálását célozza. Azt a két klasztert egyesíti, amelyek a legkisebb négyzetes hibanövekedést okozzák.

A fenti klasztertávolságok mindegyike alkalmazható, mint az EGYSZERŰFELHALMOZÓ hierarchikus algoritmus d_* függvénye. Ehhez hasonlóan dolgozhatók ki a lebontó hirarchikus algoritmusok részletei.

Gyakorlatok

32.6-1. Legyen f monoton növő függvény. Mutassuk meg, hogy az $f(d_{min})$, illetve az $f(d_{max})$ klasztertávolságon alapuló hierarchikus klaszterező módszer esetén ugyanazt az eredményt kapjuk, mint a d_{min}, illetve a d_{max} klasztertávolság alkalmazása esetén.

32.6-2. Vektortérbeli adatpontok hierarchikus klaszterezését gyorsítják az úgynevezett CF-hármasok. Egy C klasztert egy $CF(C) = (s_0, \vec{s_1}, s_2)$ hármas jellemezzen, ahol $s_0 = |C|$ a klaszterben található adatpontok száma, $\vec{s_1} = \sum_{u \in C} u$ az adatpontok vektoriális összege és $s_2 = \sum_{u \in C} ||u||^2$ az adatpontok normáinak négyzetösszege. Mutassuk meg, hogy a két klaszter CF-hármasából kiszámítható a két klaszter

a. uniójának CF-hármasa,

b. átlagos távolsága,

c. Ward-távolsága.

32.6.3. A Rock algoritmus

A korábban tárgyalt klaszterező eljárások – a k-MEDOID kivételével – numerikus attribútumokkal rendelkező adatpontokra lettek kitalálva. A ROCK felhalmozó hierarchikus klaszterező algoritmus kifejezetten olyan kategorikus attribútumokkal jellemezhető adatok klaszterezésére szolgál, mint amilyenek a bináris attribútumokkal leírható *tranzakciók*. A ROCK az EGYSZERŰFELHALMOZÓ algoritmus sémáját követi. Specialitása a d_* klaszterek távolságát mérő függvény és az S megállási szabály megvalósításában van.

A ROCK algoritmus az adatpontok közötti *kapcsolat* fogalma köré épül fel. Legyen ε az algoritmus paramétere. Az u és v pontok ***kapcsolatát*** közös szomszédainak

$$\ell(u,v) = |N_\varepsilon(u) \cap N_\varepsilon(v)|$$

száma adja. Két klaszter ***kapcsolat alapú hasonlóságát*** az

$$\ell(C_i, C_j) = \sum_{\substack{u \in C_i \\ v \in C_j}} \ell(u, v)$$

mennyiséggel, egy klaszter ***belső kapcsolatát*** az

$$\ell(C) = \ell(C, C) = \sum_{u,v \in C} \ell(u, v)$$

mennyiséggel definiáljuk.

A Rock egyik alapötlete, hogy mivel az $\ell(C_i, C_j)$ mennyiség érzékeny a C_i, C_j klaszterek méretére, az algoritmus az ℓ függvény egy normalizált változatát használja a klaszterek hasonlóságának mérésére. A következőkben egy olyan $h(n_i, n_j)$ mennyiséget definiálunk, amely két, n_i, illetve n_j méretű „átlagos" klaszter kapcsolat alapú távolságát modellezi. A h függvény meghatározásához a következő észrevételt használjuk, amelynek bizonyítását az Olvasóra hagyjuk (32.6-2 gyakorlat).

32.8. állítás. *Legyen C_i és C_j két diszjunkt klaszter. Ekkor*

$$\ell(C_i \cup C_j) = \ell(C_i) + \ell(C_j) + \ell(C_i, C_j) \ .$$

Ennek analógiájára megköveteljük, hogy $h(n_i, n_j) = h(n_i + n_j) - h(n_i) - h(n_j)$ is igaz legyen, ahol $h(n_i)$ az „átlagos" n_i méretű klaszter belső kapcsolata.

A Rock algoritmusban a $h(n_i) = n^{1+2f(\varepsilon)}$ függvény modellezi az átlagos klaszter belső kapcsolatfüggvényét, ahol $f : [0, 1] \to [0, 1]$ alkalmasan választott függvény. Például, ha tranzakciós adatok Jaccard-hasonlósággal adottak, akkor az $f(\varepsilon) = (1 - \varepsilon)/(1 + \varepsilon)$ függvény javasolható. Így két klaszter hasonlóságát az

$$s_{\text{Rock}}(C_i, C_j) = \frac{\ell(C_i, C_j)}{h(n_i, n_j)}$$

alakban írjuk fel, ahol $n_i = |C_i|$ és $n_j = |C_j|$.

A Rock algoritmusnak, mint hierarchikus módszernek minden klaszterösszevonási lépés előtt $O(n^2)$ lehetséges klaszterpár közül kell választania. A hierarchia felépítéséhez $O(n)$ összevonás szükséges, így a naív megvalósítás futási ideje $O(n^3)$ lesz. Gyakorlatban az adatpontok ℓ kapcsolatának kiszámítása, illetve a klaszterek összevonásának lépése adatstruktúrák alkalmazásával gyorsítható, így az algoritmus futási ideje hatékonyan csökkenthető.

Gyakorlatok

32.6-1. Mutassuk meg, hogy két klaszter kapcsolat alapú távolsága kiszámolható a következő módon:

$$\ell(C_i, C_j) = \sum_{u \in D} |N_\varepsilon(u) \cap C_i| \cdot |N_\varepsilon(u) \cap C_j| \ ,$$

ahol az összegzés az adatpontok D halmazán történik.

32.6-2. Bizonyítsuk be az 32.8. állítást.

32.7. Sűrűség alapú eljárások

Síkban, térben vizualizált adatpontokra „szemmel ránézve" tetszőleges formájú klasztereket tudunk azonosítani, ha a klaszteren belüli pontok megfelelő méretű (sugarú) környezete jellemzően több pontot tartalmaz, mint a klaszter határán, illetve általában a klaszterek között találhatóké. Ugyanakkor a kívülálló pontok is felismerhetők ritkább környezetükről. Ez az észrevétel adja a sűrűség alapú klaszterező eljárások alapötletét.

Egy ***sűrűség alapú*** klaszterező módszer a következő elemekből épül fel. Először az adatpontokon megadunk egy reflexív és szimmetrikus szomszédsági relációt. Ez a reláció meghatározza az adatpontok *környezetét*. A gyakorlatban a relációt úgy választjuk meg, hogy az minél jobban igazodjon a környezet esetleg előre adott intuitív fogalmához. Második lépésben lokális tulajdonságok alapján minden adatpontról eldöntjük, hogy az egy klaszter *belső pontja* lesz-e, azaz részt vesz-e a klaszter kialakításában, kohéziójában. A belső pontok meghatározása független lehet a korábban definiált szomszédsági relációtól. A következő lépésben a belső pontokra bevezetjük a szomszédsági reláció tranzitív lezártját. Ez az új ekvivalencia reláció alakítja ki a belső pontok partícióját. A többi, nem belső adatpontot a szomszédsági reláció segítségével (nem feltétlenül egyértelműen és teljesen) hozzárendeljük a klaszterekhez, mint azok határpontjai. A kimaradt adatpontok adják a *kívülállók* halmazát.

Egy *metrikus térbeli* ponthalmaz esetén adott sugarú gömbbel értelmezhetjük a pontok környezetét, és egy küszöbnél nagyobb számosságú környezettel rendelkező pontokat tekinthetjük belső pontoknak. Az így kapott sűrűség alapú módszer előnye robusztussága az adatpontok (pontosabban távolságaik) perturbációjával szemben. Az algoritmus időkomplexitása lényegesen függ a környezetek generálásától, például az itt szubrutinként használható *legközelebbi adatpontot* kiválasztó operáció megvalósításától.

32.7.1. A Dbscan algoritmus

A Dbscan algoritmus a sűrűségen alapú klaszterező módszerek legfontosabb példája. A Dbscan, mint sűrűség alapú algoritmus a *környezet* és *belső pont* fogalmát az alábbi módon definiálja. Legyen D a klaszterezendő adatpontok halmaza egy d metrikával ellátva. Az adatpontokat $u, v, \ldots$ betűkkel jelöljük. Legyen ε egy előre adott paraméter. Az ε paraméter segítségével fogjuk meghatározni az algoritmusban használt környezet fogalmát.

32.9. definíció. *Egy $u \in D$ pont ε sugarú **környezete** legyen $N_\varepsilon(u) = \{v \in D | d(u,v) \leq \varepsilon\}$. Ekkor $N_\varepsilon(u)$ elemeit u **szomszédainak** hívjuk.*

Az algoritmus további paramétere az alkalmasan választott μ ***sűrűségi korlát.*** A következő definícióval a klaszterek belső pontjait határozzuk meg.

32.10. definíció. *Egy $u \in D$ pont **sűrűségén** szomszédainak $|N_\varepsilon(u)|$ számát értjük. Egy $u \in D$ pontot **magpontnak** nevezünk, ha $|N_\varepsilon(u)| \geq \mu$.*

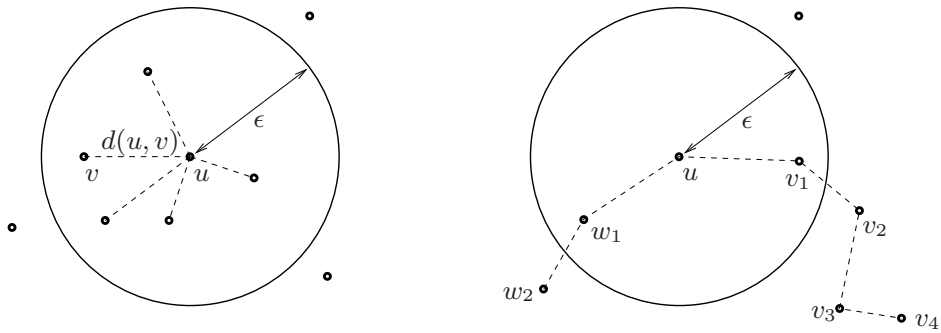
A klasztereket az alábbi definíciók segítségével fogjuk kialakítani.

32.11. definíció. *Egy $u \in D$ pont **közvetlenül elérhető** a $v \in D$ pontból, ha v magpont és u szomszédja v-nek. Egy u pont **elérhető** a v pontból, ha létezik a $v = v_1, v_2, \ldots, v_n, v_{n+1} = u$ $(v \geq 0)$ pontok olyan sorozata, hogy minden $i = 1, \ldots, n$ esetén v_{i+1} közvetlenül elérhető v_i-ből. Végül az u_1 és u_2 pontok **összekapcsoltak,** ha létezik olyan v pont, ahonnan u_1 és u_2 is elérhető.*

32.1. példa. A 32.4. ábra bal oldalán $|N_\varepsilon(u)| = 7$. Ha például $\mu = 5$, akkor az u magpont. Az ábrán az u-ból közvetlenül elérhető pontok szaggatott vonallal kapcsolódnak u-hoz. A jobb oldali ábrán u-ból elérhetők a $v_1, \ldots, v_4$, illetve a w_1, w_2 pontok, így v_4 és w_2 összekapcsoltak.

Az algoritmussal olyan C klasztereket szeretnénk előállítani, melyek egyrészt összefüggőek, azaz minden $u, v \in C$ pont összekapcsolt, másrészt maximálisak, azaz ha $u \in C$ és v elérhető u-ből, akkor $v \in C$ is teljesül. Azokat az adatpontokat, amelyek nem érhetők el egyik klaszterből sem (és így nem is magpontok), *kívülállóknak* tekintjük.

Az Dbscan algoritmus menete a következő. Válasszunk egy tetszőleges $u \in D$ pontot. Ha u magpont, akkor határozzuk meg az u-ból elérhető pontokat, ezek fogják a C_u klasztert alkotni. Ha u nem magpont, válaszunk egy új, még nem vizsgált pontot. Ha már nem tudunk új pontot választani, akkor az algoritmus véget ér. A kívülálló elemek D' halmazát azok a pontok fogják alkotni, amelyeket nem soroltunk egyik klaszterbe sem. Az algoritmus pszeudokódja a következő.



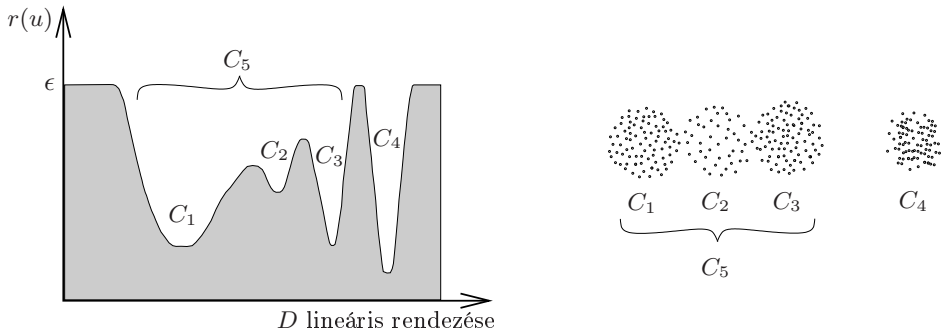
32.4. ábra. Illusztráció a 32.1. példához.

DBSCAN(D, ε, μ)

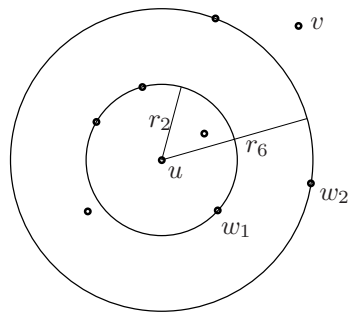
```
1  C ← ∅
2  D' ← ∅
3  while D ≠ ∅
4        u ← D egy tetszőleges eleme
5        if u magpont
6           C_u ← {v ∈ D ∪ D' | v elérhető u-ból}
7           C ← C ∪ {C_u}
8           D ← D \ C_u
9           D' ← D' \ C_u
10          else D' ← D' ∪ {u}
11                D ← D \ {u}
12 return C, D'
```

Látható, hogy minden pont környezetét elegendő egyszer megvizsgálni az algoritmus futása alatt. Egy környezet megvizsgálása általános esetben n távolság lekérdezését jelenti, ami $O(n^2)$-es futási időre vezet. Számos esetben (például alacsony dimenziós euklideszi vektortereken) megfelelő adatstruktúrával a környezetek kiszámítása $O(\log n)$ lépésben megoldható, ekkor a DBSCAN futási ideje $O(n \log n)$ lesz.

A DBSCAN algoritmus előnye, hogy tetszőleges alakú klasztereket képes felfedezni. Hátránya, hogy érzékeny a felhasználó által megadott két paraméterre (ε és μ). Amennyiben a klaszterekben található elemek sűrűsége eltérő, akkor még az sem biztos, hogy létezik olyan paraméterezés, amellyel a DBSCAN jó eredményt ad.



32.5. ábra. Az Optics algoritmus kimenetének egy lehetséges vizualizációja. A bal oldali függvényábrázoláson $r(u)$ lokális minimumai jelzik a klasztereket.



32.6. ábra. Illusztráció a 32.2. példához.

32.7.2. Az Optics algoritmus

A szintén a sűrűség fogalmára építő Optics eljárás a Dbscan ε és μ paraméterekre való érzékenységét küszöböli ki. Bár az Optics is használja ezeket a paramétereket, kimenete a Dbscan algoritmushoz képest kevésbé érzékeny értékükre. Az Optics algoritmus kimenetéből több klaszterezést is megkaphatunk. Az algoritmus listába rendezi az adatpontokat és mindnen egyes adatpontra meghatároz a pont környezetét jellemző bizonyos numerikus értékeket. Ezek segítségével azután hatékonyan megjeleníthetjük a teljes adathalmazt, illetve megkaphatjuk a Dbscan minden $\varepsilon' \leq \varepsilon$ értéknél kisebb paraméteréhez tartozó kimeneti klaszterezést és jellemezhetjük azok kapcsolatát.

Az algoritmus a Dbscan tárgyalásánál bevezetett fogalmakon túl a következő definíciókat használja. Egy $u \in D$ pont ***k*-sugara** az az r szám,

amelyre igaz, hogy legalább k olyan $w \in D \setminus \{u\}$ pont van, amelyre $d(u,w) \leq r$, és legfeljebb $k-1$ olyan $w \in D \setminus \{u\}$ pont van, amelyre $d(u,w) < r$. Ha egy előre megadott ε paraméter esetén $N_\varepsilon(u) < k$, akkor végtelennek tekintjük az u pont k-sugarát. Amikor a μ paraméter adott, akkor egy $u \in D$ pont μ-távolságát ***magsugárnak*** hívjuk. Egy $v \in D$ pont $u \in D$ magponttól tekintett ***elérhető távolsága*** pedig legyen $r_u(v) = \max\{u \text{ magsugara}, d(u,v)\}$.

32.2. példa. A 32.6. ábrán az u pont 2-távolsága r_2, a 6-távolsága pedig r_6. Ha $\mu = 6$, akkor a v pont u-tól mért elérhető távolsága $d(u,v)$; míg a w_1 és w_2 pontok u-tól mért elérhető távolsága r_6 magsugár lesz.

Az Optics algoritmus kimenete

- a D-beli adatpontok $\sigma : \{1, \ldots, |D|\} \to D$ lineáris rendezése;
- minden u adatponthoz annak $m(u)$ magsugara; valamint
- a pont környezetét jellemző $r(u)$ érték, amely egy alkalmasan választott másik ponttól tekintett elérhetőségi távolság lesz.

Az algoritmus fő ciklusa korábban nem vizsgált magpontot keres, majd a Bővít szubrutin során kiszámolja a magpontból elérhető klasztert. A Dbscan-től való lényeges eltérés az új klaszter bejárásában jelentkezik. Míg a Dbscan tetszőlegesen választ a közvetlenül elérhető pontokból, addig az Optics mindig a pillanatnyilag elérhető legnagyobb sűrűségű környezetből választ. Az algoritmus pszeudokódja a következő.

Optics(D, ε, μ)

```
1  s ← 0                    // A kimeneti sorrend mutatójának kezdeti értékadása.
2  while D ≠ ∅
3        do u ← D egy tetszőleges eleme
4           D ← D \ {u}
5           σ(u) ← s
6           s ← s + 1
7           r(u) ← ε                    // Az elérhető távolsághoz itt még
                                        // nincs referenciapont.
8           if u magpont
9              Bővít (D, ε, μ, s, u)
10 return σ,r
```

BŐVÍT$(D, \varepsilon, \mu, s, u)$

```
1  B ← N_ε(u) \ {u}                          // Bővítési halmaz kezdetben.
2  D ← D \ N_ε(u)
3  while B ≠ ∅
4        for minden v ∈ B
5            r_i(v) ← max (d(v, σ(i)), σ(i) magsugara)
                                  // A v pont aktuális elérhetőségi távolságai,
                                  // végtelen, ha σ(i) nem magpont.
6            r(v) ← min_{i=0,...,s−1} r_i(v)
7            w ← argmin_{v∈B} r(v)
8            σ(w) ← s
9            s ← s + 1
10           if w magpont
11           B ← B ∪ N_ε(w)
12           D ← D \ N_ε(w)
13           B ← B \ {w}
14 return D, s, σ, r
```

A BŐVÍT algoritmus 5. sorának megvalósítása során az $r(v)$ értékek minden, a σ sorrend végére kerülő új w adatpont környezetének lekérdezésével frissíthetőek. Így az OPTICS futása során egy adatpont környezetét legfeljebb kétszer kell lekérdezni, ezért a DBSCAN esetében kapott $O(n^2)$, illetve az adatstruktúrák alkalmazásával elérhető $O(n \lg n)$ futási idő itt is érvényes. Látható az is, hogy az algoritmus futási ideje az ε és μ paraméterek függvényében monoton módon változik. Az ε paraméter a B bővítési halmaz méretén keresztül (lásd a BŐVÍT algoritmus 3. sorát) döntően befolyásolja az egész OPTICS algoritmus futási idejét. Gyakorlati tapasztalatok szerint a μ paraméter értékének megváltoztatása az algoritmus kimenetét jellemzően csak kis (bár természetesen az adathalmaztól függő) mértékben változtatja meg.

Gyakorlatok

32.7-1. Az OPTICS algoritmus BŐVÍT szubrutinjában (a pszeudokód 6. sorában) definiált $r(v)$ érték miért lesz véges?

32.7-2. Mutassuk meg, hogy az OPTICS kimenetéből (a klaszterek határpontjaitól eltekintve) megkapható tetszőleges $\varepsilon' \leq \varepsilon$ paraméterrel futtatott DBSCAN által kimenetként adott klaszterezés.

Feladatok

32-1 A k-középpont probléma
Tekintsük egy d metrikával ellátott metrikus térben n pontot. Jelölje D a pontok halmazát. A k-középpont probléma megoldása a pontoknak az a k elemű S halmaza, amely minimalizálja a

$$\max_{u \in D} \min_{s \in S} d(u, s)$$

mennyiséget. Mutassuk meg, hogy a k-középpont probléma NP-nehéz! (*Útmutatás.* Vezessük vissza a gráfok domináns halmazának NP-nehéz problémáját a k-középpont problémára a 32.3-1 gyakorlatban szereplő metrika segítségével!)

32-2 Mintavételezés
Tegyük fel, hogy az adathalmaz egyforma klaszterekből áll. Mekkora véletlen mintát kell ahhoz venni, hogy legfeljebb 0.01 valószínűséggel veszítsünk el klasztert, azaz annak a valószínűsége, hogy a minta egyetlen egy elemet sem tartalmaz valamelyik klaszterből, legfeljebb 0.01 legyen?

Most tegyük fel, hogy az s méretű ($s = 1, 2, \ldots$) klaszterek száma egy hatványfüggvénnyel arányos, azaz $|\{C : |C| = s\}| \sim s^{-\alpha}$, ahol $\alpha > 1$. Mekkora minta szükséges, ha 0.01 valószínűséggel szeretnénk mintát kapni legalább a klaszterek feléből? (*Útmutatás.* Használjuk a Chernoff-egyenlőtlenséget!)

Megjegyzések a fejezethez

Az elmúlt évtizedekben számos monográfia készült a klaszterezés témakörében: Anderberg [18], Hartigan [174], Jain és Dubes [196], Kaufman és Rousseeuw [202]. A korábbi művek főleg a statisztika és a gépi tanulás szemszögéből vizsgálták a klaszterezést, az újabb összefoglalásokban jelenik meg az adatbányászati szemlélet. Frissebb összefoglalások a Jain, Murty és Flynn [195], Berkhin [44] áttekintő cikkek, illetve Han és Kamber [170] könyvének klaszterezésről szóló fejezete.

Ezzel a könyvvel egy időben jelennek meg a Khosrow-Pour [211], illetve a Wang [361] által szerkesztett enciklopédiák, amelyek gazdag anyagot tartalmaznak a klaszterezés gyakorlati kérdéseiről.

A Bell-számok pontos aszimptotikája megtalálható Lovász feladatgyűjteményének [236] első fejezetében. A klaszterezés korlátait mutató 32.4. tétel forrása Kleinberg [212].

A dimenzió-csökkentés keretében a 32.4.1. pontban ismertetett szinguláris felbontás a lineáris algebra egyik ismert, alapvető eszköze. A felbonthatóságról szóló 32.5. tétel bizonyításáról, illetve a felbontás további alkalmazásairól olvashatunk Rózsa [311] könyvében magyarul, illetve Golub és Van Loan

[150] művében. Utóbbi alkotás betekintést nyújt a felbontás kiszámítására használt numerikus módszerekbe is. A felbontással történő közelítés optimalitásáról szóló 32.6. tétel általánosításának bizonyítása, illetve mátrix normák és a szinguláris felbontás kapcsolatáról részletesen olvashatunk Stewart és Sun [330] monográfiájában. Szöveges dokumentumok dimenzió-csökkentésére először Deerwester, Dumais, Landauer, Furnas és Harshman [92] használták a felbontást, azóta számos cikk foglalkozik ezzel az alkalmazással.

A 32.4.2. pontban ismertetett ujjlenyomat alapú dimenzió-csökkentést szintén szöveges dokumentumokra Broder [62] vezette be. Az itt tárgyalt hasonlóság keresési eljárást és weboldalakra történő alkalmazását Haveliwala, Gionis és Indyk [176] tárgyalja részletesen.

A k-KÖZÉP algoritmus egyik korai megfogalmazása MacQueentől [243], a fejezetben már említett módosítás Hartigantól [174] származik. A módosítás futási idejét Berkhin és Becher [45] vizsgálta. A k-MEDOID eljárás PAM algoritmus néven Kaufmann és Rousseeuw [202] munkája. További particionáló algoritmus az adatokról valószínűségi modellt feltevő és ak-KÖZÉP algoritmus általánosításának tekinthető Lauritzentől [227] származó EM (Expectation-Maximization) algoritmus.

Néhány dimenziós vektortérben reprezentálható adatpontok és klaszterreprezentánsok esetében a *kd*-fa adatstruktúra jól használható az algoritmusok skálázására, lásd Moore és Pelleg [284]. A k-KÖZÉP algoritmus számos, például a klaszterszámot futás közben változtatni, vagy a kívülállók kezelésre képes variánsa is ismert (lásd [243] és [18]).

A hierarchikus klaszterezés módszerét Kaufmann és Rousseeuw [202] tárgyalja. A Kruskal-algoritmus útösszenyomáson alapuló, rendkívül hatékony adatstruktúrát használó megvalósításának részletei Rónyai, Ivanyos és Szabó könyvének [307] 6.6.3. pontjában olvashatóak. A ROCK algoritmust Guha, Rastogi és Shim [161] vezette be.

A DBSCAN algoritmus Ester, Kriegel, Sander és Xu [117], az OPTICS eljárás Ankerst, Breunig, Kriegel és Sanders [21] munkája.

A szerzők munkáját részben támogatta a T 042706 számú OTKA-szerződés és az ADATROSTA NKFP-2/0017/2002 számú projekt.

33. Lekérdezés átírás relációs adatbázisokban

Az első kötetben (lásd 12. fejezet) bevezettük a relációs adatbázisok alapfogalmait, többek között a relációs séma, reláció, reláció példány fogalmát. Az adatbázisokat tervezői oldalról közelítettük meg, a fő kérdés az volt, hogyan érhetjük el, hogy elkerüljük az adatok redundáns tárolását, illetve az adatbázis használata során fellépő különböző anomáliákat.

Jelen fejezetben a sémát adottnak tekintjük és megpróbáljuk a felhasználó kérdéseit minél gyorsabban és teljesebben megválaszolni. Ehhez először (33.1. alfejezet) áttekintjük az alapvető (elméleti) lekérdezési nyelveket, az ezek közti kapcsolatokat.

A fejezet második részében (33.2. alfejezet) a nézeteket tárgyaljuk. Informálisan, egy nézet nem más, mint egy lekérdezés eredménye. Bemutatjuk a nézetek kapcsolatát lekérdezések gyorsításával, a fizikai adatfüggetlenség biztosításával, valamint az adatok integrálásával.

A fejezet harmadik részében (33.3. alfejezet) lekérdezések átírásával foglalkozunk.

33.1. Lekérdezések

Tekintsük a budapesti mozihálózat adatbázisát. Tegyük fel, hogy a séma három relációból áll:

$$\textbf{PestiMűsor} = \{\mathit{Filmek}, \mathit{Mozik}, \mathit{Műsor}\} \ . \tag{33.1}$$

Az egyes relációk sémái a következőek:

$$\begin{array}{rcl} \mathit{Filmek} & = & \{\textbf{C}\mathit{ím}, \textbf{R}\mathit{endező}, \textbf{Sz}\mathit{ínész}\} \ , \\ \mathit{Mozik} & = & \{\textbf{M}\mathit{ozi}, \textbf{U}\mathit{tca}, \textbf{T}\mathit{elefon}\} \ , \\ \mathit{Műsor} & = & \{\textbf{M}\mathit{ozi}, \textbf{C}\mathit{ím}, \textbf{I}\mathit{dőpont}\} \ . \end{array} \tag{33.2}$$

Az egyes relációk példányainak lehetséges részletei a 33.1. ábrán láthatók. Tipikus felhasználói kérdések lehetnek:

33.1 Ki rendezte a “Kontroll”-t?

33.2 Listázzuk az összes olyan mozi nevét és címét, ahol Kuroszava filmet játszanak!

Filmek

C*ím*	**R***endező*	**Sz***ínész*
Kontroll	Antal Nimród	Csányi Sándor
Kontroll	Antal Nimród	Mucsi Zoltán
Kontroll	Antal Nimród	Pindroch Csaba
⋮	⋮	⋮
A vihar kapujában	Kuroszava Akira	Mifune Tosiro
A vihar kapujában	Kuroszava Akira	Kjó Macsiko
A vihar kapujában	Kuroszava Akira	Maszajuki Mori

Mozik

M*ozi*	**U***tca*	**T***elefon*
Bem	II., Margit krt. 5/b.	316-8708
Corvin Budapest Filmpalota	VIII., Corvin köz 1.	459-5050
Európa	VII., Rákóczi út 82.	322-5419
Művész	VI., Teréz krt. 30.	332-6726
⋮	⋮	⋮
Uránia Nemzeti Filmszínház	VIII., Rákóczi út 21.	486-3413
Vörösmarty	VIII., Üllői út 4.	317-4542

Műsor

M*ozi*	**C***ím*	**I***dőpont*
Bem	A vihar kapujában	19:00
Bem	A vihar kapujában	21:30
Uránia Nemzeti Filmszínház	Kontroll	18:15
Művész	A vihar kapujában	16:30
Művész	Kontroll	17:00
⋮	⋮	⋮
Corvin Budapest Filmpalota	Kontroll	10:15

33.1. ábra. A **PestiMűsor** adatbázis.

33.3 Adjuk meg azon rendezők nevét, akik szerepeltek valamelyik saját filmjükben!

Ezek a kérdések leképezéseket definiálnak a **PestiMűsor** adatbázis séma relációiból valamilyen másik sémába (jelen esetben egyetlen relációból álló sémákba). Formálisan meg kell különböztetnünk a ***lekérdezést*** és a ***lekérdezés függvényt.*** Az előbbi szintaktikus fogalom, az utóbbi pedig

leképezés a bemeneti sémához tartozó példányok halmazából a kimeneti séma példányainak halmazába, amit a lekérdezés határoz meg, valamilyen alkalmas szemantikus értelmezés szerint. Azonban az egyszerűség kedvéért mindkét fogalomra a „lekérdezés" szót használjuk, a környezetből mindig világos lesz, hogy éppen melyikről beszélünk.

33.1. definíció. *Az* $\mathbf{R}$ *bemeneti séma feletti* q_1 *és* q_2 *lekérdezések* ***ekvivalensek,*** *jelölésben* $q_1 \equiv q_2$*, ha ugyanaz a kimeneti sémájuk, és minden* $\mathbf{R}$*-hez tartozó* $\mathcal{I}$ *példányra* $q_1(\mathcal{I}) = q_2(\mathcal{I})$*.*

A fejezet további részében áttekintjük a legfontosabb lekérdezési nyelveket. Szükségünk lesz a lekérdezési nyelvek kifejező erejének összehasonlítására.

33.2. definíció. *Legyenek* $\mathcal{Q}_1$ *és* $\mathcal{Q}_2$ *lekérdezési nyelvek (a megfelelő értelmezéssel).* $\mathcal{Q}_2$ ***gazdagabb,*** *mint* $\mathcal{Q}_1$ *(* $\mathcal{Q}_1$ ***szűkebb,*** *mint* $\mathcal{Q}_2$*), jelölésben* $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$*, ha minden* q_1 $\mathcal{Q}_1$*-beli lekérdezéshez van* $q_2 \in \mathcal{Q}_2$*, amelyre* $q_1 \equiv q_2$*.* $\mathcal{Q}_1$ *és* $\mathcal{Q}_2$ ***ekvivalensek,*** *ha* $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$ *és* $\mathcal{Q}_1 \sqsupseteq \mathcal{Q}_2$*.*

33.1. példa. *Lekérdezés.* Tekintsük a 31.1. kérdést. Első közelítésben a következő megoldást találjuk:

> **if** léteznek a *Filmek*, *Mozik* és *Műsor* relációkban $(x_C, \text{"Kuroszava Akira"}, x_{Sz})$, (x_M, x_U, x_T) és (x_M, x_C, x_I) sorok
> **then** vegyük be a $(Mozi : x_M, Utca : x_U)$ sort az eredmény relációba.

$x_C, x_{Sz}, x_M, x_I, x_U, x_T$ különböző változókat jelölnek, amelyek az értékeiket a megfelelő attribútum értelmezési tartományából veszik fel. Ugyanazon változók használatával közvetetten jeleztük, hogy a különböző sorokban hol kell egyenlő értékeknek szerepelniük.

33.1.1. Konjunktív lekérdezések

A lekérdezések legegyszerűbb és legtöbb jó tulajdonsággal rendelkező fajtája. Három, egymással ekvivalens változatát ismertetjük, amelyek közül kettő logikai alapú, a harmadik pedig algebrai. A név a logikai változatbó lered, olyan elsőrendű kifejezéseken alapszik, amelyek csak egzisztenciális kvantorokat ($\exists$), valamint "és"-sel (konjunkcióval) összekötött atomi kifejezéseket tartalmaznak.

Datalog – szabály alapú lekérdezés

Az $(x_1, \ldots, x_m)$ sort ***szabad sornak*** nevezzük, ha az x_i-k változók vagy konstansok. A szabad sor a reláció példány egy sorának általánosítása. A 33.1. példában szereplő $(x_C, \text{"Kuroszava Akira"}, x_{Sz})$ sor szabad.

33.3. definíció. *Legyen* $\mathbf{R}$ *relációs adatbázis séma.* ***Szabály alapú konjunktív lekérdezésen*** *a következő alakú kifejezést értjük:*

$$val(u) \leftarrow R_1(u_1), R_2(u_2), \ldots, R_n(u_n)\ , \tag{33.3}$$

ahol $n \geq 0$, $R_1, \ldots, R_n$ $\mathbf{R}$*-beli reláció nevek,* val *olyan reláció név, ami nincs* $\mathbf{R}$*-ben;* $u, u_1, \ldots, u_n$ *szabad sorok. Minden* u*-ban előforduló változónak elő kell fordulnia* $u_1, \ldots, u_n$ *valamelyikében is.*

A szabály alapú konjunktív lekérdezéseket egyszerűbben csak ***szabályoknak*** is nevezzük. $val(u)$ a szabály ***feje,*** $R_1(u_1), \ldots, R_n(u_n)$ a szabály ***teste.*** $R_i(u_i)$-t ***(relációs) atomnak*** nevezzük. Feltesszük, hogy a fejben előforduló összes változó előfordul valamelyik testbeli atomban is.

Egy szabályt úgy tekinthetünk, mint valamilyen eszközt, ami megmondja, hogyan vezethetünk le újabb és újabb ***tényeket,*** azaz sorokat, a lekérdezés eredmény relációjába. Ha találunk a szabályban előforduló változóknak olyan értékeket, hogy minden $R_i(u_i)$ atom igaz (azaz a megfelelő sor az R_i relációban van), akkor a val relációba bevesszük az u sort. Mivel a fejben előforduló változók előfordulnak testbeli atomokban is, elérjük, hogy sohasem kell ***végtelen*** értelmezési tartományokkal foglalkozzunk, hiszen a változók csak az éppen lekérdezett példányban előforduló konstans értékeket vehetik fel. Formálisan, legyen $\mathcal{I}$ az $\mathbf{R}$ relációs séma feletti példány, q pedig (33.3)-mal adott lekérdezés. Jelölje $var(q)$ a q-ban előforduló változók halmazát, $dom(\mathcal{I})$ pedig az $\mathcal{I}$-beli konstansok halmazát. $\mathcal{I}$ q ***alatti képe***

$$q(\mathcal{I}) = \{\nu(u) | \nu\colon var(q) \to adom(\mathcal{I}) \text{és} \nu(u_i) \in R_i\ i = 1, \ldots, n\}\ . \tag{33.4}$$

$q(\mathcal{I})$ kiszámításának triviális módja, hogy valamely sorrendben végignézzük a lehetséges ν kiértékeléseket. Ennél hatékonyabb algoritmus is lehetséges, egyrészt a lekérdezés ekvivalens átalakításával, másrészt megfelelő indexek használatával. Fontos különbség a fejben és a testben szereplő atomok között, hogy az $R_1, R_2, \ldots, R_n$ relációkat adottnak, (fizikailag) tároltnak tekintjük, míg a val relációt nem, azt úgy gondoljuk, hogy a szabály segítségével számoljuk ki. Ebből adódik az elnevezés: R_i-k ***extenzionális relációk,*** val ***intenzionális reláció.***

Az $\mathbf{R}$ séma feletti q lekérdezés ***monoton,*** ha az $\mathcal{I}$ és $\mathcal{J}$ $\mathbf{R}$ feletti példányokra $\mathcal{I} \subseteq \mathcal{J}$-ből $q(\mathcal{I}) \subseteq q(\mathcal{J})$ következik. q ***kielégíthető,*** ha létezik olyan $\mathcal{I}$ példány, amelyre $q(\mathcal{I}) \neq \emptyset$. A következő egyszerű észrevétel bizonyítását az Olvasóra bízzuk (33.1-1 gyakorlat).

33.4. állítás. *Szabály alapú lekérdezések monotonok és kielégíthetőek.*

A 33.4. állítás rámutat a szabály alapú lekérdezések korlátaira. Például a *Mely mozikban játszanak csak Jancsó filmeket?* lekérdezés nyilvánvalóan nem monoton, tehát nem is fejezhető ki (33.3) formájú szabállyal.

Táblázatos lekérdezések

Ha a változók és konstansok közti különbséget nem vesszük figyelembe, akkor egy szabály teste a séma feletti példánynak is tekinthető. Ez a nézőpont elvezet a konjunktív lekérdezések táblázatos formájához, ami leginkább hasonlatos a Microsoft Access adatbázis-kezelő rendszer (QBE: Query By Example) szemléletes lekérdezéseihez.

33.5. definíció. *Az* $\mathbf{R}$ *séma feletti* ***tábla*** *az* $\mathbf{R}$ *séma feletti példány általánosítása, a sorokban konstansok mellett változók is előfordulhatnak. A* $(\mathbf{T}, u)$ *pár* ***táblázatos lekérdezés,*** *ha* $\mathbf{T}$ *egy tábla és* u *egy szabad sor, úgy, hogy minden* u*-ban előforduló változó előfordul* $\mathbf{T}$*-ben is. Az* u *szabad sor a táblázatos lekérdezés* ***összegzése.***

A $(\mathbf{T}, u)$ táblázatos lekérdezés u összegzés sora mutatja meg, hogy mely sorok alkotják a lekérdezés eredményét. Az eljárás lényege, hogy megpróbáljuk a $\mathbf{T}$ táblával adott mintát az adatbázisban megtalálni, és ha sikerül, akkor az u-nak megfelelő sort bevesszük az eredmény relációba. Pontosabban, $\nu\colon var(\mathbf{T}) \to dom(\mathcal{I})$ a $(\mathbf{T}, u)$ tábla ***beágyazása*** az $\mathcal{I}$ példányba, ha $\nu(\mathbf{T}) \subseteq \mathcal{I}$. A $(\mathbf{T}, u)$ táblázatos lekérdezés eredmény relációja mindazon $\nu(u)$ sorokból áll, amelyekre ν a $(\mathbf{T}, u)$ tábla beágyazása az $\mathcal{I}$ példányba.

33.2. példa. *Táblázatos lekérdezés.* Legyen $\mathbf{T}$ a következő tábla.

Filmek	*Cím*	*Rendező*	*Színész*
	x_C	"Kuroszava Akira"	x_{Sz}

Mozik	*Mozi*	*Utca*	*Telefon*
	x_M	x_U	x_T

Műsor	*Mozi*	*Cím*	*Időpont*
	x_M	x_C	x_I

A $(\mathbf{T}, \langle Mozi\colon x_M, Utca\colon x_U\rangle)$ táblás lekérdezés a bevezetés 33.2. kérdését válaszolja meg.

A táblázatos lekérdezések szintaxisa nagyon hasonló a szabály alapú lekérdezésekéhez. A későbbiekben hasznos lesz számunkra, hogy a táblázatos lekérdezések nyelvén könnyen megfogalmazható lesz annak feltétele, hogy két lekérdezés közül az egyik mikor tartalmazza a másikat.

Relációs algebra*

Az adatbázis relációkból áll, a relációk pedig sorok halmazai. A lekérdezések eredménye is adott attribútum halmazzal rendelkező reláció. Természetes gondolat, hogy a lekérdezés eredményét a bemeneti relációkból halmazalgebrai,

illetve a relációkon értelmezett egyéb műveletekkel fejezzük ki. A ***relációs algebra**** a következő műveleteket tartalmazza[1].

Kiválasztás: Az operáció alakja $\sigma_{A=c}$ vagy $\sigma_{A=B}$, ahol A és B attribútumok, c pedig konstans. A művelet alkalmazható minden olyan R relációra, amelyiknek A (és B) attribútuma, eredménye pedig értelemszerűen a *val* reláció, amelynek ugyanazok az attribútumai, mint R-nek, és azon R-beli sorokat tartalmazza, amelyekre igaz a kiválasztási feltétel.

Vetítés: Az operáció alakja $\pi_{A_1,\ldots,A_n}$, $n \geq 0$, ahol A_i-k különböző attribútumok. A művelet alkalmazható minden olyan R relációra, amelyik attribútumai közt minden A_i előfordul, eredménye pedig a *val* reláció, amelyik attribútum halmaza $\{A_1, \ldots, A_n\}$,

$$val = \{t[A_1, A_2, \ldots, A_n] | t \in R\} ,$$

azaz az R-beli sorok $\{A_1, A_2, \ldots, A_n\}$ attribútum halmazra való megszorításaiból áll.

Természetes összekapcsolás: Ezt a műveletet a k**??**. fejezetben definiáltuk. Jelölése $\bowtie$, bemenete kettő R_1, R_2 (vagy több) reláció, V_1, V_2 attribútum halmazokkal. Az eredmény reláció attribútum halmaza $V_1 \cup V_2$.

$$R_1 \bowtie R_2 = \{t \text{ sor } V_1 \cup V_2 \text{ felett } | \exists v \in R_1, \exists w \in R_2, t[V_1] = v \text{ és } t[V_2] = w\} .$$

Átnevezés: Attribútum átnevezés nem más, mint egy egy-egy értelmű leképezés a véges U attribútum halmazról az összes attribútumok halmazába. Az f attribútum átnevezés megadható az $(A, f(A))$ párok listájával, ahol $A \neq f(A)$, ezt általában $A_1 \ldots A_n \to B_1 \ldots B_n$ alakban írjuk, $f(A_i) = B_i$. Az ***átnevezés operátor*** δ_f, U feletti bemenetekről $f[U]$ feletti kimenetekre képez. Ha R az U feletti reláció, akkor

$$\delta_f(R) = \{v \ f[U] \text{ felett} | \exists u \in R, v(f(A)) = u(A), \forall A \in U\} .$$

Relációs algebra* lekérdezéseket a fenti műveletek véges sokszor ismételt alkalmazásával nyerünk a ***relációs algebrai alap lekérdezésekből,*** amelyek

Bemenet reláció: R.

Egyetlen konstans: $\{\langle A : a \rangle\}$, ahol a konstans, A attribútum név.

33.3. példa. *Relációs algebra* lekérdezés.* A bevezetés 33.2. kérdése relációs algebrai műveletekkel a következőképpen fejezhető ki

$$\pi_{Mozi,Utca}\left((\sigma_{Rendező=\text{“Kuroszava Akira”}}(Filmek) \bowtie Műsor) \bowtie Mozik\right) .$$

[1] A relációs algebra* a később bevezetett (teljes) relációs algebra ***monoton*** része.

A relációs algebra* lekérdezés által megvalósított leképezést a műveletifa szerinti indukcióval definiálhatjuk értelemszerűen. Könnyen látható (33.1-2 gyakorlat), hogy relációs algebra* használatával megadható olyan lekérdezés, ami nem elégíthető ki. Az ilyennel ekvivalens szabály alapú, illetve táblázatos lekérdezés nyilván nem létezik. Azonban igaz a következő.

33.6. tétel. *A szabály alapú lekérdezések, a táblázatos lekérdezések és a kielégíthető relációs algebra* lekérdezések nyelvei ekvivalensek.*

A tétel bizonyítása három fő lépésből áll:

1. Szabály alapú $\equiv$ Táblázatos.
2. Kielégíthető relációs algebra* $\sqsubseteq$ Szabály alapú és
3. Szabály alapú $\sqsubseteq$ Kielégíthető relációs algebra*.

Az első lépést az Olvasóra bízzuk (33.1-3 gyakorlat). A második lépéshez először be kell látni, hogy a szabály alapú lekérdezések nyelve zárt a lekérdezések egymásba ágyazására nézve. Pontosabban, legyen $\mathbf{R} = \{R_1, R_2, \ldots, R_n\}$ adatbázis, q lekérdezés $\mathbf{R}$ felett. Ha q eredmény relációja S_1, akkor egy következő lekérdezésben már S_1 is használható ugyanúgy, mint $\mathbf{R}$ tetszőleges extenzionális relációja. Így definiálhatjuk az S_2, majd annak segítségével az S_3, és így tovább, relációkat. Az S_i relációk ***intenzionális*** relációk. A P ***konjunktív lekérdezés program*** szabályok sorozata

$$\begin{array}{rcl} S_1(u_1) & \leftarrow & test_1 \\ S_2(u_2) & \leftarrow & test_2 \\ & \vdots & \\ S_m(u_m) & \leftarrow & test_m, \end{array} \tag{33.5}$$

ahol az S_i-k különbözőek, és nincsenek $\mathbf{R}$-ben. A $test_i$ szabály testben csak az $R_1, \ldots, R_n$ és $S_1, \ldots, S_{i-1}$ relációk fordulhatnak elő. P eredmény relációjának S_m-t tekintjük, kiszámítása a szabályok sorrend szerinti kiértékelésével történik. Nem nehéz látni, hogy a változók megfelelő átnevezésével P egyetlen szabállyal helyettesíthető, ahogy azt a következő példa mutatja.

33.4. példa. *Konjunktív lekérdezés program.* Legyen $\mathbf{R} = \{Q, R\}$, és tekintsük a következő konjunktív lekérdezés programot

$$\begin{array}{rcl} S_1(x,z) & = & Q(x,y), R(y,z,w) \\ S_2(x,y,z) & = & S_1(x,w), R(w,y,v), S_1(v,z) \\ S_3(x,z) & = & S_2(x,u,v), Q(v,z). \end{array} \tag{33.6}$$

Az (33.6) első két szabálya alapján S_2 felírható csak Q és R használatával

$$S_2(x,y,z) = Q(x,y_1), R(y_1,w,w_1), R(w,y,v), Q(v,y_2), R(y_2,z,w_2)\ . \tag{33.7}$$

Látható, hogy bizonyos változókat át kellett nevezni, hogy elkerüljük a különböző szabály testek nem kívánt egymásra hatását. A (33.7) kifejezést (33.6) harmadik szabályába írva S_2 helyére, és megfelelően átnevezve a változókat kapjuk

$$S_3(x,z) = Q(x,y_1), R(y_1,w,w_1), R(w,u,v_1), Q(v_1,y_2), R(y_2,v,w_2), Q(v,z)\ . \tag{33.8}$$

Ezek után elegendő a relációs algebra* műveleteit egyenként megvalósítani szabállyal.

$P \bowtie Q$: Jelölje $\overrightarrow{x}$ a P és Q közös attribútumainak megfelelő változók (konstansok) listáját, $\overrightarrow{y}$ a csak P-ben, míg $\overrightarrow{z}$ a csak Q-ban előforduló attribútumoknak megfelelő változókat (konstansokat). Ekkor a $val(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z}) \leftarrow P(\overrightarrow{x}, \overrightarrow{y}), Q(\overrightarrow{x}, \overrightarrow{z})$ szabály a $P \bowtie Q$ relációt adja eredményül.

$\sigma_F(R)$: Tegyük fel, hogy $R = R(A_1, \ldots, A_n)$. F a szelekciós feltétel, $A_i = a$ vagy $A_i = A_j$ alakú, ahol A_i, A_j attribútumok, a konstans. Ekkor

$$val(x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n) \leftarrow R(x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n)\ ,$$

illetve

$$val(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_{j-1}, y, x_{j+1}, \ldots, x_n) =$$
$$R(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_{j-1}, y, x_{j+1}, \ldots, x_n)$$

megfelelő szabály. Itt használjuk ki a relációs algebra* lekérdezés kielégíthetőségét. Ugyanis, a műveletek egymás utáni elvégzése során sohasem kapunk olyan kifejezést, amiben két különböző konstanst kellene egyenlővé tennünk.

$\pi_{A_{i_1},\ldots,A_{i_m}}(R)$: Ha $R = R(A_1, \ldots, A_n)$, akkor

$$val(x_{i_1}, \ldots, x_{i_m}) \leftarrow R(x_1, \ldots, x_n)$$

jó lesz.

$A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_n$: Az átnevezés relációs algebrai műveletet a megfelelő változók átnevezésével oldhatjuk meg, amint azt a 33.4. példában láttuk.

A harmadik lépés bizonyításához tekintsünk egy

$$val(\overrightarrow{x}) \leftarrow R_1(\overrightarrow{x_1}), R_2(\overrightarrow{x_2}), \ldots, R_n(\overrightarrow{x_n}) \tag{33.9}$$

szabályt. Az R_i relációk attribútumainak megfelelő átnevezésével elérhető, hogy csupa különböző attribútumnév szerepeljen. Ezek után képezhetjük az $R = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$ relációt, ami ténylegesen az R_i relációk direkt szorzata lesz, az attribútum nevek különbözősége miatt. A (33.9) szabályban szereplő konstansokat és változó egyezéseket megfelelő szelekciós operátorok alkalmazásával szimulálhatjuk. Végül a $val(\overrightarrow{x})$ reláció változóinak megfelelő attribútum halmazra való vetítéssel kapjuk a végeredményt.

33.1.2. Kiterjesztések

A konjunktív lekérdezések a lekérdezési nyelvek jól kezelhető fajtája. Azonban viszonylag szűk az általuk kifejezhető kérdések köre. Tekintsük a következőket.

33.4. Adjuk meg az olyan párokat, amelyeknek az első tagja rendezte a második tagot valamilyen filmben, valamint fordítva, a második tag is rendezte az elsőt valamilyen filmben.

33.5. Melyik moziban megy a "Szegénylegények" vagy a "Vihar kapujában"?

33.6 Melyek azok a Hitchcock filmek, amelyekben nem szerepelt Hitchcock maga?

33.7. Listázzuk azokat a filmeket, amelyek összes színésze szerepelt Jancsó Miklós valamelyik filmjében.

33.8. Ismeretes a „Színészlánc" játék. Az első játékos mond egy színészt, a sorban következő pedig egy olyat, aki vele egy filmben játszott. Ez így folytatódik, mindig olyan színészt kell mondani, aki az előzővel játszott egy filmben, és még nem szerepelt a játék folyamán. (Az nyer, aki utolsóként tudja még folytatni a láncot.) Listázzuk azokat a színészeket, akikhez „Latinovits Zoltán"-tól el lehet jutni a játék folyamán.

Egyenlőség atomok

A **33.4** kérdésre könnyen megadhatjuk a választ, ha a szabály testében a relációs atomokon kívül egyenlőséget is megengedünk

$$val(y_1, y_2) = Filmek(x_1, y_1, z_1), Filmek(x_2, y_2, z_2), y_1 = z_2, y_2 = z_1 \ . \quad (33.10)$$

Az egyenlőség megengedése két problémát is felvet. Az első, hogy a lekérdezés eredménye végtelen is lehet. Például a

$$val(x, y) = R(x), y = z \quad (33.11)$$

lekérdezés eredménye végtelen sok sor, hiszen az y és z változókat az R reláció nem korlátozza, így végtelen sok kiértékelés lehetséges, ami a szabály

testet kielégíti. Ezért a q szabály alapú lekérdezés ***tartomány-korlátozott,*** ha minden változó, melynek q testében előfordul, az előfordul valamely relációs atomban is.

A második probléma az, hogy egyenlőség atomok a szabály testben a testbeli feltétel kielégíthetetlenségét okozhatják, ellentétben a 33.4. állítással. Például a

$$val(x) \leftarrow R(x), x = a, x = b \tag{33.12}$$

szabály tartomány-korlátozott, viszont ha a és b különböző konstansok, akkor a válasz az üres reláció lesz. Könnyen eldönthető, hogy egy q egyenlőségeket is tartalmazó szabály alapú lekérdezés kielégíthető-e.

KIELÉGÍTHETŐ(q)

```
1  Számítsuk ki a q testében levő egyenlőségek tranzitív lezártját.
2  if Két különböző konstans egyenlő a tranzitivitás miatt
3     return "Nem elégíthető ki."
4            else return "Kielégíthető."
```

Igaz az is (33.1-4. gyakorlat), hogy ha a q egyenlőségeket is tartalmazó szabály alapú lekérdezés kielégíthető, akkor van vele ekvivalens q', amelyik egyenlőség nélküli.

Diszjunkció – egyesítés

A **33.5** kérdés nem fejezhető ki konjunktív lekérdezéssel, azonban ha az egyesítés műveletét hozzávesszük a relációs algebrához, akkor az így kiterjesztett algebrával már kifejezhető:

$$\pi_{Mozi}\left(\sigma_{Cím=\text{“Szegénylegények”}}(Műsor) \cup \sigma_{Cím=\text{“A vihar kapujában”}}(Műsor)\right) . \tag{33.13}$$

Szabály alapú lekérdezések is képesek a **33.5** kérdés kifejezésére, ha megengedjük, hogy több különböző szabálynak is ugyanaz a reláció legyen a fejében:

$$\begin{aligned} val(x_M) &\leftarrow Műsor(x_M, \text{“Szegénylegények”}, x_V) , \\ val(x_M) &\leftarrow Műsor(x_M, \text{“A vihar kapujában”}, x_V) . \end{aligned} \tag{33.14}$$

Ennek általánosítása a ***nemrekurzív datalog program.***

33.7. definíció. *Az* **R** *séma feletti* ***nemrekurzív datalog program***

$$\begin{aligned} S_1(u_1) &\leftarrow test_1 \\ S_2(u_2) &\leftarrow test_2 \\ &\vdots \\ S_m(u_m) &\leftarrow test_m \end{aligned} \tag{33.15}$$

szabályok halmaza, ahol $\mathbf{R}$*-beli reláció nem szerepel szabály fejben, ugyanaz a reláció több szabály fejében is szerepelhet, valamint létezik a szabályok olyan* $r_1, \ldots, r_m$ *sorrendje, hogy az* r_i *szabály fejében levő reláció nem fordul elő semelyik* r_j *szabály testében sem, ha* $j \leq i$*.*

A (33.15) nemrekurzív datalog program eredményének kiszámítása a (33.5) konjunktív lekérdezés program kiszámításához hasonló. A szabályokat a 33.7. definícióban szereplő sorrendben számoljuk, ha több szabály fejében ugyanaz a reláció szerepel, akkor a szabályok által adott sorok halmazainak egyesítését vesszük.

A $(\mathbf{T}_i, u)$ $i = 1, 2, \ldots, n$ táblázatos lekérdezések egyesítését $(\{\mathbf{T}_1, \ldots, \mathbf{T}_n\}, u)$ jelöli. Kiszámításához az egyes $(\mathbf{T}_i, u)$ lekérdezéseket külön-külön kiszámítjuk, majd az eredmények egyesítését vesszük. Igaz a következő tétel.

33.8. tétel. *Az egyetlen célrelációval rendelkező nemrekurzív datalog programok nyelve és a relációs algebra kiegészítve az unió műveletével ekvivalensek.*

A 33.8. tétel bizonyítása hasonló a 33.6. tétel bizonyításához, az Olvasóra bízzuk (33.1-5 gyakorlat). Megjegyezzük, hogy a táblázatos lekérdezések egyesítésének kifejező ereje gyengébb, aminek az az oka, hogy ugyanazt az összegző sort követeljük meg minden egyes táblázathoz. A

$$\begin{array}{l} val(a) \leftarrow \\ val(b) \leftarrow \end{array} \tag{33.16}$$

nemrekurzív datalog program nem valósítható meg táblázatos lekérdezések egyesítéseként.

Tagadás

A **33.6**. kérdés nyilvánvalóan nem monoton. Tegyük fel, hogy a *Filmek* relációban léteznek sorok Hitchcock *Psycho* című filmjéről, például ("Psycho","Hitchcock","A. Perkins"), ("Psycho","Hitchcock","J. Leigh"), ... , azonban nincs ("Psycho","Hitchcock","Hitchcock") sor. Ekkor a **33.6**. lekérdezés eredményében a ("Psycho") sor is szerepel. Azonban kicsit alaposabb kutatással kideríthető, hogy Hitchcock szerepel a *Psycho* című filmben, mint "egy ember cowboy kalapban". Ha ezek után hozzávesszük a *Filmek* relációhoz a ("Psycho","Hitchcock","Hitchcock") sort, akkor a **PestiMűsor** séma aktuális példánya bővebb lesz, viszont a **33.6** lekérdezés eredménye szűkebb.

Könnyű látni, hogy az eddigi fejezetekben tárgyalt lekérdezési nyelvek által megvalósított lekérdezések monotonok, azaz a **33.6**. lekérdezés nem valósítható meg nemrekurzív datalog programmal, vagy vele ekvivalens nyelvvel. Azonban, ha a relációs algebrai műveletek közé felvesszük a kivonás

$(-)$ operátort is, akkor alkalmas a **33.6**. típusú lekérdezések megvalósítására. Például,

$$\pi_{Cím}(\sigma_{Rendező=\text{“Hitchcock”}}(Filmek)) - \pi_{Cím}(\sigma_{Színész=\text{“Hitchcock”}}(Filmek)) \tag{33.17}$$

pontosan a **33.6.** lekérdezést valósítja meg. A (teljes) relációs algebrát tehát a $\{\sigma, \pi, \bowtie, \delta, \cup, -\}$ műveletek alkotják. A relációs algebra fontosságát az is mutatja, hogy Codd a $\mathcal{Q}$ lekérdezési nyelvet pontosan akkor nevezi ***relációsan teljesnek,*** ha minden q algebrai lekérdezéshez van $q' \in \mathcal{Q}$, hogy $q \equiv q'$.

Ha megengedünk szabály testekben ***negatív literálokat,*** azaz $\neg R(u)$ alakú atomokat, akkor az így kapott ***nemrekurzív datalog tagadással,*** jelölésben ***nr-datalog***$^\neg$ már relációsan teljes lesz.

33.9. definíció. *Nemrekurzív datalog*$^\neg$ *(nr-datalog*$^\neg$*) szabály a*

$$q: \quad S(u) \leftarrow L_1, L_2, \ldots, L_n \tag{33.18}$$

alakú szabály, ahol S egy reláció, u egy szabad sor, az L_i pedig ***literál,*** *azaz $R(v)$ vagy $\neg R(v)$ alakú kifejezés, $i = 1, \ldots, n$, v szabad sor. S nem fordul elő a szabály testében. A szabály* ***tartomány-korlátozott,*** *ha minden x változó, ami előfordul a szabályban, előfordul valamely* ***pozitív literálban*** *($R(v)$ alakú kifejezésben) is a szabály testében. Ha másképp nem jelezzük, akkor minden nr-datalog*$^\neg$ *szabályt tartomány-korlátozottnak tekintünk.*

A (33.18) szabály jelentése a következő. Legyen $\mathbf{R}$ relációs séma, amelyik tartalmazza a q testében szereplő összes relációt, továbbá legyen $\mathcal{I}$ $\mathbf{R}$ feletti példány. $\mathcal{I}$ ***q-szerinti képe***

$$\begin{aligned} q(\mathcal{I}) = \{\nu(u) |\ & \nu \text{ a változók kiértékelése és } i = 1, \ldots, n\text{-re} \\ & \nu(u_i) \in \mathcal{I}(R_i), \text{ ha } L_i = R_i(u_i) \text{ és} \\ & \nu(u_i) \notin \mathcal{I}(R_i), \text{ ha } L_i = \neg R_i(u_i)\}. \end{aligned} \tag{33.19}$$

Az $\mathbf{R}$ séma feletti ***nr-datalog***$^\neg$ ***program*** nr-datalog$^\neg$ szabályok

$$\begin{aligned} S_1(u_1) &= test_1 \\ S_2(u_2) &= test_2 \\ &\vdots \\ S_m(u_m) &= test_m \end{aligned} \tag{33.20}$$

halmaza, ahol $\mathbf{R}$-beli reláció nem szerepel szabály fejben, ugyanaz a reláció több szabály fejében is szerepelhet, valamint létezik a szabályok olyan $r_1, r_2, \ldots, r_m$ sorrendje, hogy az r_i szabály fejében levő reláció nem fordul elő semelyik r_j szabály testében sem, ha $j \leq i$.

A (33.20) nr-datalog$^\neg$ program eredményének kiszámítása az $\mathbf{R}$ séma $\mathcal{I}$ példányára alkalmazva megegyezik a (33.15) nemrekurzív datalog program esetén használt módszerrel, azzal a különbséggel, hogy az egyes nr-datalog$^\neg$ szabályokat (33.19) szerint értelmezzük.

33.5. példa. *Nr-datalog$^\neg$ program.* Tegyük fel, hogy minden filmnek, amelyik a *Filmekben* szerepel, egyetlen rendezője van. (Nem mindig teljesül a valóságban!) A **33.6** lekérdezést

$$val(x) = Filmek(x, \text{“Hitchcock”}, z), \neg Filmek(x, \text{“Hitchcock”}, \text{“Hitchcock”}) \tag{33.21}$$

nr-datalog$^\neg$ szabály valósítja meg. A **33.7.** lekérdezést pedig a

$$\begin{array}{rcl} Jancsó\text{-}színész(z) & \leftarrow & Filmek(x, \text{”Jancsó”}, z) \\ Nem\text{-}ez\text{-}a\text{-}válasz(x) & \leftarrow & Filmek(x, y, z), \neg Jancsó\text{-}színész(z) \\ Válasz(x) & \leftarrow & Filmek(x, y, z), \neg Nem\text{-}ez\text{-}a\text{-}válasz(x) \end{array} \tag{33.22}$$

nr-datalog$^\neg$ program válaszolja meg. Óvatosnak kell lennünk azonban nr-datalog$^\neg$ program írásakor. Ha a (33.22) program első két szabályát egybe vonnánk a 33.4 példához hasonlóan

$$\begin{array}{rcl} Rossz\text{-}nem\text{-}v(x) & = & Filmek(x, y, z), \neg Filmek(x', \text{”Jancsó”}, z), Filmek(x', \text{”Jancsó”}, z') \\ Válasz(x) & = & Filmek(x, y, z), \neg Rossz\text{-}nem\text{-}v(x), \end{array} \tag{33.23}$$

akkor (33.23) nem a **33.7.** lekérdezést adná, hanem (feltéve, hogy minden filmnek egy rendezője van) a következő lekérdezést.

33.9. Listázzuk azokat a filmeket, amelyek minden színésze az ***összes*** Jancsó filmben szerepelt.

Könnyen látható, hogy minden kielégíthető nr-datalog$^\neg$ program, amelyik tartalmaz egyenlőség atomokat is, helyettesíthető olyannal, amelyikben nem szerepelnek egyenlőség atomok. Továbbá igaz az alábbi állítás is.

33.10. állítás. *A kielégíthető (teljes) relációs algebra és az egyetlen cél relációval rendelkező kielégíthető nr-datalog$^\neg$ program ekvivalens lekérdezési nyelvek.*

Rekurzió

A **33.8**. kérdést az eddigi lekérdezési nyelvekkel nem tudjuk megfogalmazni. Szükségünk lenne valamilyen ***a priori*** információra arról, hogy legfeljebb milyen hosszú *színészlánc* képezhető a kiindulási színészből. Tegyük fel, hogy tudjuk, “Latinovits”-ból indulva legfeljebb 117 hosszú lánc képezhető. (Érdekes lenne tudni a tényleges értéket!) Ekkor a következő nemrekurzív

datalog program megadja a választ.

$$\begin{aligned}
\textit{Film-társ}(z_1, z_2) &= \textit{Filmek}(x, y, z_1), \textit{Filmek}(x, y, z_2), z_1 < z_2 \text{[2]} \\
\textit{Rész-válasz}_1(z) &= \textit{Film-társ}(z, \text{“Latinovits”}) \\
\textit{Rész-válasz}_1(z) &= \textit{Film-társ}(\text{“Latinovits”}, z) \\
\textit{Rész-válasz}_2(z) &= \textit{Film-társ}(z, y), \textit{Rész-válasz}_1(y) \\
\textit{Rész-válasz}_2(z) &= \textit{Film-társ}(y, z), \textit{Rész-válasz}_1(y) \\
&\vdots \quad \vdots \\
\textit{Rész-válasz}_{117}(z) &= \textit{Film-társ}(z, y), \textit{Rész-válasz}_{116}(y) \\
\textit{Rész-válasz}_{117}(z) &= \textit{Film-társ}(y, z), \textit{Rész-válasz}_{116}(y) \\
\textit{Latinovits-lánc}(z) &= \textit{Rész-válasz}_1(z) \\
\textit{Latinovits-lánc}(z) &= \textit{Rész-válasz}_2(z) \\
&\vdots \quad \vdots \\
\textit{Latinovits-lánc}(z) &= \textit{Rész-válasz}_{117}(z)
\end{aligned} \tag{33.24}$$

A (33.24) datalog program rekurzív, mivel a *Színészlánc-társ* reláció definíciójában önmagát használjuk. Tegyük fel, hogy ez értelmezhető (lásd később), ekkor a **33.8** kérdésre a választ megadja a

$$\textit{Latinovits-lánc}(x) = \textit{Színészlánc-társ}(x, \text{“Latinovits”}) \tag{33.25}$$

szabály.

33.11. definíció. *Az*

$$R_1(u_1) = R_2(u_2), R_3(u_3), \ldots, R_n(u_n) \tag{33.26}$$

kifejezés ***datalog szabály,*** *ha* $n \geq 1$, R_i*-k reláció nevek,* u_i*-k megfelelő hosszúságú szabad sorok. Minden* u_1*-beli változó elő kell forduljon* $u_2, \ldots u_n$ *valamelyikében is. A szabály feje* $R_1(u_1)$*, teste* $R_2(u_2), \ldots, R_n(u_n)$*. A* ***datalog program*** *(33.26) alakú szabályok véges halmaza. Legyen* P *datalog program. A* P*-ben szereplő* R *reláció* ***extenzionális,*** *ha csak szabály testekben fordul elő.* ***Intenzionális*** *a reláció, ha valamelyik szabály fejében előfordul.*

Ha ν a (33.26) szabályban szereplő változók valamely kiértékelése, akkor $R_1(\nu(u_1)) \leftarrow R_2(\nu(u_2)), R_3(\nu(u_3)), \ldots, R_n(\nu(u_n))$ a (33.26) szabály ***megvalósítása.*** Az ***extenzionális (adatbázis) séma*** P extenzionális relációiból áll, jelölésben $edb(P)$. $idb(P)$, az ***intenzionális (adatbázis) séma*** hasonlóképpen P intenzionális relációit tartalmazza. Jelölje $sch(P) = edb(P) \cup idb(P)$. A P datalog program szemantikus jelentése egy leképezés az

[2] Az egyenlőség atomokhoz hasonlóan használhatunk más összehasonlítási atomokat is. Itt a $z_1 < z_2$ azt biztosítja, hogy minden pár csak egyszer szerepel a felsorolásban.

$edb(P)$ feletti példányok halmazáról az $idb(P)$ feletti példányok halmazába. Ezt a jelentést modell-elméletileg, bizonyítás-elméletileg, illetve valamely leképezés fixpontjának segítségével lehet megadni. Mivel az első kettő lényegileg ekvivalens a harmadikkal, és tárgyalásuk túl messzire vezetne, ezért csak a fixpont alapú definícióval foglalkozunk.

A 33.11. definícióban nem használtunk negatív literálokat. Ennek fő oka, hogy általában a rekurzió és a tagadás együtt értelmetlen lehet. Azonban bizonyos esetekben szükségünk lehet negatív atomokra is, akkor majd speciálisan foglalkozunk a program értelmezésével.

Fixpont szemantika

Legyen P datalog program, $\mathcal{K}$ $sch(P)$ feletti példány. Az A ***tény,*** azaz konstansokból álló sor, $\mathcal{K}$ és P ***közvetlen következménye,*** ha vagy $A \in \mathcal{K}(R)$ valamely $R \in sch(P)$ relációra, vagy $A \leftarrow A_1, \ldots, A_n$ a P valamelyik szabályának megvalósítása és minden A_i $\mathcal{K}$-ban van. P ***közvetlen következmény operátora*** T_P az $sch(P)$ feletti példányok halmazából képez önmagára. $T_P(\mathcal{K})$ a $\mathcal{K}$ és P összes közvetlen következményéből áll.

33.12. állítás. *A T_P közvetlen következmény operátor monoton.*

Bizonyítás. Tegyük fel, hogy $\mathcal{I}$ és $\mathcal{J}$ $sch(P)$ feletti példányok, valamint $\mathcal{I} \subseteq \mathcal{J}$. Legyen A $T_P(\mathcal{I})$-be tartozó tény. Ha $A \in \mathcal{I}(R)$ valamely $R \in sch(P)$ relációra, akkor $\mathcal{I} \subseteq \mathcal{J}$ alapján $A \in \mathcal{J}(R)$ is teljesül. Ha pedig $A \leftarrow A_1, \ldots, A_n$ a P valamelyik szabályának megvalósítása és minden A_i $\mathcal{I}$-ben van, akkor $A_i \in \mathcal{J}$ teljesül. ∎

T_P definíciójából következik, hogy $\mathcal{K} \subseteq T_P(\mathcal{K})$. Innen, felhasználva a 33.12. állítást kapjuk, hogy

$$\mathcal{K} \subseteq T_P(\mathcal{K}) \subseteq T_P(T_P(\mathcal{K})) \subseteq \ldots \ . \tag{33.27}$$

33.13. tétel. *Minden $sch(P)$ feletti $\mathcal{I}$ példányhoz létezik egy egyértelmű minimális $\mathcal{I} \subseteq \mathcal{K}$ példány, ami a T_P* ***fixpontja,*** *azaz $\mathcal{K} = T_P(\mathcal{K})$.*

Bizonyítás. Jelölje $T_P^i(\mathcal{I})$ a T_P operátor i-szeres egymás utáni alkalmazását, és legyen

$$\mathcal{K} = \bigcup_{i=0}^{\infty} T_P^i(\mathcal{I}) \ . \tag{33.28}$$

(33.28) és T_P monotonitása alapján

$$T_P(\mathcal{K}) = \bigcup_{i=1}^{\infty} T_P^i(\mathcal{I}) \subseteq \bigcup_{i=0}^{\infty} T_P^i(\mathcal{I}) = \mathcal{K} \subseteq T_P(\mathcal{K}) \ , \tag{33.29}$$

azaz $\mathcal{K}$ fixpont. Könnyen látható, hogy minden olyan fixpont, ami tartalmazza $\mathcal{I}$-t, tartalmazza $T_P^i(\mathcal{I})$-t is minden i-re ($i = 1, \ldots$), azaz $\mathcal{K}$-t is. ∎

33.14. definíció. *A P datalog program* ***eredménye*** *az $edb(P)$ feletti $\mathcal{I}$ példányon T_P $\mathcal{I}$-t tartalmazó minimális fixpontja, jelölésben $P(\mathcal{I})$.*

Belátható, lásd 33.1-6 gyakorlat, hogy a (33.27)-beli lánc véges, azaz létezik olyan n, hogy $T_P(T_P^n(\mathcal{I})) = T_P^n(\mathcal{I})$. Ez az alapja a datalog program eredménye naiv kiszámítási módjának.

NAIV-DATALOG(P,$\mathcal{I}$)

1 $\mathcal{K} = \mathcal{I}$
2 **while** $T_P(\mathcal{K}) \neq \mathcal{K}$
3 $\quad\quad \mathcal{K} = T_P(\mathcal{K})$
4 **return** $\mathcal{K}$

Természetesen a NAIV-DATALOG eljárás nem optimális, hiszen minden egyes tényt, ami $\mathcal{K}$-ba belekerül, a **while** ciklus minden további végrehajtásánál újra kiszámol.

A FÉLIG-NAIV-DATALOG eljárás elve, hogy amennyire csak lehet, az éppen kiszámított új tényeket használja csak a **while** ciklus során, elkerülve ezzel a már ismert tények újraszámolását. Tekintsük a P datalog programot, $edb(P) = \mathbf{R}$, $idb(P) = \mathbf{T}$. A P-beli

$$S(u) = R_1(v_1), \ldots, R_n(v_n), T_1(w_1), \ldots, T_m(w_m) \tag{33.30}$$

szabályhoz, ahol $R_k \in \mathbf{R}$ és $T_j \in \mathbf{T}$, elkészítjük a következő szabályokat $j = 1, 2, \ldots, m$ és $i \geq 1$-re

$$temp_S^{i+1}(u) = \begin{array}{l} R_1(v_1), \ldots, R_n(v_n), \\ T_1^i(w_1), \ldots, T_{j-1}^i(w_{j-1}), \Delta_{T_j}^i(w_j), T_{j+1}^{i-1}(w_{j+1}), \ldots, T_m^{i-1}(w_m) \,. \end{array} \tag{33.31}$$

A $\Delta_{T_j}^i$ reláció a T_j i-edik iterációban történő változását jelöli. Az i-edik szint S-re vonatkozó szabályainak együttesét P_S^i jelöli (azaz (33.31) szabályokat $temp_S^{i+1}$-re, $j = 1, 2, \ldots, m$ esetén). Tegyük fel, hogy $T_1, T_2, \ldots, T_\ell$ az S *idb* relációt meghatározó szabályok testében szereplő *idb* relációk listája. Jelölje

$$P_S^i(\mathcal{I}, T_1^{i-1}, \ldots, T_\ell^{i-1}, T_1^i, \ldots, T_\ell^i, \Delta_{T_1}^i, \ldots, \Delta_{T_\ell}^i) \tag{33.32}$$

a (33.31) szabályok az $\mathcal{I}$ bemeneti példányra és a $T_j^{i-1}, T_j^i, \Delta_{T_j}^i$ *idb* relációkra való alkalmazásával kapott tényeket (sorokat). Az $\mathcal{I}$ bemeneti példány az $edb(P)$ relációinak aktuális értéke.

FÉLIG-NAIV-DATALOG(P,$\mathcal{I}$)

1 $P' =$ azon P-beli szabályok, amelyek testében nincs *idb* reláció
2 **for** $S \in idb(P)$
3 $\quad S^0 = \emptyset$
4 $\quad\quad \Delta_S^1 = P'(\mathcal{I})(S)$
5 $i = 1$
6 **repeat**
7 $\quad\quad$ **for** $S \in idb(P)$
8 $\quad$ // $T_1, \ldots, T_\ell$ az S-t meghatározó szabályokban előforduló *idb* relációk.
9 $\quad\quad\quad S^i = S^{i-1} \cup \Delta_S^i$
10 $\quad\quad\quad \Delta_S^{i+1} = P_S^i(\mathcal{I}, T_1^{i-1}, \ldots, T_\ell^{i-1}, T_1^i, \ldots, T_\ell^i, \Delta_{T_1}^i, \ldots, \Delta_{T_\ell}^i) - S^i$
11 $\quad\quad i = i + 1$
12 **until** $\Delta_S^i = \emptyset$ minden $S \in idb(P)$-re
13 $\quad\quad$ **for** $S \in idb(P)$
14 $\quad\quad S = S^i$
15 **return** $\mathcal{S}$

33.15. tétel. *A* FÉLIG-NAIV-DATALOG *helyesen számítja ki a P datalog program eredményét.*

Bizonyítás. i-szerinti teljes indukcióval belátjuk, hogy tetszőleges $S \in idb(P)$-re a 6–12. sorok ciklusának i-edik végrehajtása után az S^i értéke $T_P^i(\mathcal{I})(S)$, Δ_S^{i+1} pedig $T_P^{i+1}(\mathcal{I})(S) - T_P^i(\mathcal{I})(S)$-sel egyenlő. $T_P^i(\mathcal{I})(S)$ értelemszerűen a T_P közvetlen következmény operátor i-szeres alkalmazásával az $\mathcal{I}$ példányból kiindulva az S relációra kapott érték.

$i = 0$-ra a 4. sor pontosan $T_P(\mathcal{I})(S)$-t számítja ki minden $S \in idb(P)$-re. Az indukciós lépéshez mindössze azt kell látnunk, hogy $P_S^i(\mathcal{I}, T_1^{i-1}, \ldots, T_\ell^{i-1}, T_1^i, \ldots, T_\ell^i, \Delta_{T_1}^i, \ldots, \Delta_{T_\ell}^i) \cup S^i$ pontosan $T_P^{i+1}(\mathcal{I})(S)$-sel egyenlő, hiszen a 9–10. sorokban a FÉLIG-NAIV-DATALOG eljárás ennek használatával állítja elő S^i-t és Δ_S^{i+1}-et. Az indukciós feltétel szerint S^i értéke $T_P^i(\mathcal{I})(S)$, ehhez képest új sorokat csak úgy kaphatunk, ha valamely S-et meghatározó *idb* relációnak olyan sorait vesszük figyelembe, amelyek T_P legutolsó alkalmazásakor keletkeztek, ezek pedig szintén az indukciós feltétel miatt a $\Delta_{T_1}^i, \ldots, \Delta_{T_\ell}^i$ relációkban vannak.

A 12. sor feltétele pontosan azt jelenti, hogy minden $S \in idb(P)$ reláció változatlan marad a T_P közvetlen következmény operátor alkalmazása során, tehát az algoritmus megtalálta annak minimális fixpontját. Az pedig a 33.14. definíció szerint pontosan a P datalog program eredménye az $\mathcal{I}$ bemeneti példányra. ∎

A FÉLIG-NAIV-DATALOG eljárás ugyan sok felesleges számítást kiküszöböl, azonban bizonyos datalog programokon ez sem optimális (33.1-7. gyakorlat). Azonban a datalog program elemzésével és a számítás azon alapuló módosításával a legtöbb felesleges lépést meg lehet takarítani.

33.16. definíció. *Legyen P datalog program. P **előzmény gráfja** G_P a következő irányított gráf. Csúcshalmaza $idb(P)$ relációi, $R, R' \in idb(P)$ esetén (R, R') irányított él, ha létezik olyan szabály P-ben, amelynek feje R' és R a testében van. P **rekurzív,** ha G_P-ben van irányított kör. Az R és R' relációk **kölcsönösen rekurzívak,** ha G_P ugyanazon erősen összefüggő komponensébe esnek.*

A kölcsönös rekurzivitás ekvivalencia reláció $idb(P)$-be tartozó relációk halmazán. A JAVÍTOTT-FÉLIG-NAIV-DATALOG eljárás alapgondolata, hogy az $R \in idb(P)$ relációval "egyszerre" csak a vele kölcsönösen rekurzív relációkat kell számolni, minden más, R-t definiáló szabályban előforduló relációt már „előre" kiszámíthatunk, és *edb* relációnak tekinthetünk.

JAVÍTOTT-FÉLIG-NAIV-DATALOG(P,$\mathcal{I}$)

1 Határozzuk meg $idb(P)$ kölcsönös rekurzivitás szerinti ekvivalencia osztályait.
2 Készítsük el az $[R_1], [R_2], \ldots, [R_n]$ ekvivalencia osztályok listáját G_P topologikus rendezése szerint.
3 // Minden $i < j$-re teljesül, hogy G_P-ben nincs irányított út R_j-ből R_i-be.
4 **for** $i = 1$ **to** n
5 Használjuk a FÉLIG-NAIV-DATALOG eljárást az $[R_i]$-beli relációk kiszámítására, az $[R_j]$-beli relációkat *edb* relációkként kezelve $j < i$-re.

Mélységi keresés alkalmazásával az 1–2. sorok $O(v_{G_P} + e_{G_P})$ időben végrehajthatók, ahol v_{G_P} és e_{G_P} a G_P gráf csúcs-, illetve élszámát jelölik. Az eljárás helyességének bizonyítását az Olvasóra bízzuk (33.1-8 gyakorlat).

33.1.3. Bonyolultsági kérdések lekérdezések közti tartalmazásról

A jelen részben visszatérünk a konjunktív lekérdezésekhez. Lekérdezések eredményének számításakor a legköltségesebb feladat relációk természetes összekapcsolásának elvégzése. Különösen igaz ez, ha a közös attribútumokhoz nincs index megadva, és így csak TELJES-SORONKÉNTI-ÖSSZEKAPCSOLÁS eljárás alkalmazható.

TELJES-SORONKÉNTI-ÖSSZEKAPCSOLÁS(R_1, R_2)

```
1  S = ∅
2  for minden u ∈ R_1
3      for minden v ∈ R_2
4          if u és v összekapcsolható
5              S = S ∪ {u ⋈ v}
6  return S
```

Világos, hogy a TELJES-SORONKÉNTI-ÖSSZEKAPCSOLÁS futási ideje $O(|R_1| \times |R_2|)$. Nem mindegy tehát, hogy egy lekérdezést milyen sorrendben számítunk ki, hiszen az eljárás folyamán különböző méretű relációk természetes összekapcsoltjait kell képezni. Táblázatos lekérdezések esetén a ***homomorfizmus tétel*** lehetőséget ad a lekérdezés olyan átírására, amelyik kevesebb összekapcsolást használ, mint az eredeti.

Az $\mathbf{R}$ séma feletti q_1, q_2 lekérdezésekre q_2 ***tartalmazza*** q_1-t, jelben $q_1 \sqsubseteq q_2$, ha minden $\mathbf{R}$ feletti $\mathcal{I}$ példányra $q_1(\mathcal{I}) \subseteq q_2(\mathcal{I})$ teljesül. $q_1 \equiv q_2$ a 33.1. definíció értelmében pontosan akkor, ha $q_1 \sqsubseteq q_2$ és $q_1 \sqsupseteq q_2$. Szükségünk lesz a kiértékelések általánosítására. ***Helyettesítésen*** olyan leképezést értünk, amelyik a változók halmazából képez a változók és a konstansok halmazának egyesítésébe, és amelyiket konstansokra identitásként terjesztünk ki. Természetesen értelmezhető a helyettesítés kiterjesztése szabad sorokra, illetve táblázatokra.

33.17. definíció. *Legyen $q = (\mathbf{T}, u)$ és $q' = (\mathbf{T}', u')$ két táblázatos lekérdezés az $\mathbf{R}$ séma felett. A θ helyettesítés* ***homomorfizmus*** *q'-ről q-ra, ha $\theta(\mathbf{T}') = \mathbf{T}$ és $\theta(u') = u$.*

33.18. tétel. (homomorfizmus tétel). *Legyen $q = (\mathbf{T}, u)$ és $q' = (\mathbf{T}', u')$ két táblázatos lekérdezés az $\mathbf{R}$ séma felett. $q \sqsubseteq q'$ akkor és csak akkor, ha létezik homomorfizmus q'-ről q-ra.*

Bizonyítás. Tegyük fel először, hogy θ homomorfizmus q'-ről q-ra, és legyen $\mathcal{I}$ az $\mathbf{R}$ séma feletti példány. Legyen $w \in q(\mathcal{I})$. Ez pontosan akkor teljesül, ha létezik egy ν kiértékelés, amelyik a $\mathbf{T}$ táblát $\mathcal{I}$-be képezi és $\nu(u) = w$. Könnyen látható, hogy $\theta \circ \nu$ a $\mathbf{T}'$ táblát képezi $\mathcal{I}$-be és $\theta \circ \nu(u') = w$, azaz $w \in q'(\mathcal{I})$. Tehát $w \in q(\mathcal{I}) \Longrightarrow w \in q'(\mathcal{I})$, ami pontosan $q \sqsubseteq q'$-vel egyenértékű.

Másik oldalról, tegyük fel, hogy $q \sqsubseteq q'$. A bizonyítás gondolata, hogy q-t és q'-t is alkalmazzuk a $\mathbf{T}$ „példányra". q eredménye az u szabad sor, tehát q' eredménye szintén tartalmazza az u sort, vagyis létezik $\mathbf{T}'$ egy θ beágyazása $\mathbf{T}$-be, amelyik u'-t u-ra képezi. A gondolatmenet szabatossá tételéhez elkészítjük a $\mathbf{T}$-vel izomorf $\mathcal{I}_\mathbf{T}$ példányt.

Legyen V a $\mathbf{T}$-ben előforduló változók halmaza. Minden $x \in V$-hez rendeljük az a_x konstanst, ami különbözik a $\mathbf{T}$-ben illetve $\mathbf{T}'$-ben előforduló konstansoktól, valamint $x \neq x' \implies a_x \neq a_{x'}$. Legyen μ az a kiértékelés, amelyik $x \in V$-hez a_x-t rendeli, valamint legyen $\mathcal{I}_{\mathbf{T}} = \mu(\mathbf{T})$. μ bijekció V-ről $\mu(V)$-re, és $\mu(V)$-ben nem fordulnak elő $\mathbf{T}$-beli konstansok, ezért μ^{-1} jól definiált az $\mathcal{I}_{\mathbf{T}}$-ben előforduló konstansokon.

Világos, hogy $\mu(u) \in q(\mathcal{I}_{\mathbf{T}})$, tehát $q \sqsubseteq q'$ alapján $\mu(u) \in q'(\mathcal{I}_{\mathbf{T}})$ is teljesül. Azaz, létezik egy ν kiértékelés, ami a $\mathbf{T}'$ táblát beágyazza $\mathcal{I}_{\mathbf{T}}$-be úgy, hogy $\nu(u') = \mu(u)$. Könnyen látható, hogy $\nu \circ \mu^{-1}$ homomorfizmus q'-ről q-ra. ■

Lekérdezés optimalizálás tábla minimalizálással

A 33.6. tétel alapján táblázatos lekérdezések és a kielégíthető relációs algebrai lekérdezések (kivonás nélkül) ekvivalensek. A bizonyítás során kiderül, hogy a táblázatos lekérdezéssel ekvivalens relációs algebra kifejezés $\pi_{\overrightarrow{\mathbf{x}}}(\sigma_F(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_k))$ alakú, ahol k a tábla sorainak száma. Ebből következik, hogy ha a természetes összekapcsolások számát akarjuk minimalizálni, akkor az ekvivalens tábla sorainak számát kell a lehető legkisebbre csökkenteni.

A $(\mathbf{T}, u)$ táblázatos lekérdezés ***minimális,*** ha nincs olyan $(\mathbf{S}, v)$ lekérdezés, amelyik ekvivalens $(\mathbf{T}, u)$-val és $|\mathbf{S}| < |\mathbf{T}|$, azaz $\mathbf{S}$-nek kevesebb sora van. Meglepő, de igaz, hogy a $(\mathbf{T}, u)$-val ekvivalens minimális lekérdezés megkapható egyszerűen néhány sor elhagyásával $\mathbf{T}$-ből.

33.19. tétel. *Legyen $q = (\mathbf{T}, u)$ táblázatos lekérdezés. Van $\mathbf{T}'$ részhalmaza $\mathbf{T}$-nek, hogy $q' = (\mathbf{T}', u)$ minimális és ekvivalens $q = (\mathbf{T}, u)$-val.*

Bizonyítás. Legyen $(\mathbf{S}, v)$ minimális, q-val ekvivalens lekérdezés. A homomorfizmus tétel szerint létezik θ homomorfizmus q-ról $(\mathbf{S}, v)$-re, valamint λ $(\mathbf{S}, v)$-ről q-ra. Legyen $\mathbf{T}' = \theta \circ \lambda(\mathbf{S})$. Könnyen ellenőrizhető, hogy $(\mathbf{T}', u) \equiv q$ és $|\mathbf{T}'| \leq |\mathbf{S}|$. Azonban $(\mathbf{S}, v)$ minimális, így $(\mathbf{T}', u)$ is az. ■

33.6. példa. *Tábla minimalizálás alkalmazása.* Tekintsük az $\{A, B, C\}$ attribútum halmazú $\mathbf{R}$ séma feletti

$$q = \pi_{AB}\left(\sigma_{B=5}(R)\right) \bowtie \pi_{BC}\left(\pi_{AB}(R) \bowtie \pi_{AC}\left(\sigma_{B=5}(R)\right)\right) \tag{33.33}$$

relációs algebrai lekérdezést. A q-nak megfelelő táblás lekérdezés a következő $\mathbf{T}$ tábla:

R	A	B	C
	x	5	z_1
	x_1	5	z_2
	x_1	5	z
u	x	5	z

(33.34)

Olyan homomorfizmust keresünk, ami a $\mathbf{T}$ tábla néhány sorát $\mathbf{T}$ más soraira képezi, ezáltal mintegy „összehajtogatja"' a táblát. Az első sor nem hagyható el,

mert a homomorfizmus az u szabad soron azonosság, tehát x-t önmagának kell megfeleltesse. Hasonló a helyzet a harmadik sorral, mert z-nek is saját maga a képe minden homomorfizmusnál. Azonban a második sort ki lehet küszöbölni, x_1-t x-re, z_2-t z-re képezve. Tehát a $\mathbf{T}$-vel ekvivalens minimális tábla $\mathbf{T}$ első és harmadik sorát tartalmazza. Visszaírva algebrai lekérdezésre,

$$\pi_{AB}(\sigma_{B=5}(R)) \bowtie \pi_{BC}(\sigma_{B=5}(R)) \tag{33.35}$$

az eredmény. A (33.35) lekérdezés a (33.33) lekérdezéshez képest eggyel kevesebb összekapcsolás műveletet tartalmaz.

A következő tétel azt mondja ki, hogy táblák közti tartalmazás és ekvivalencia eldöntésének kérdése NP-teljes, következésképpen a tábla minimalizálás NP-nehéz feladat.

33.20. tétel. *Adott q és q' táblázatos lekérdezések esetén az alábbi döntési feladatok NP-teljesek:*

33.10. $q \sqsubseteq q'$*?*

33.11. $q \equiv q'$*?*

33.12. Tegyük fel, hogy q-ból néhány szabad sor elhagyásával kaptuk q'-t. Igaz-e ekkor, hogy $q \equiv q'$*?*

Bizonyítás. A Pontos fedés feladatot vezetjük vissza a különböző tábla feladatokra. A Pontos fedés feladat bemenete egy $X = \{x_1, \ldots, x_n\}$ halmaz, valamint részhalmazainak $\mathcal{S} = \{S_1, \ldots, S_m\}$ rendszere. Eldöntendő, hogy létezik-e olyan $\mathcal{S}' \sqsubseteq \mathcal{S}$, hogy az $\mathcal{S}'$-beli részhalmazok pontosan lefedik X-et (azaz, minden $x \in X$-hez pontosan egy $S \in \mathcal{S}'$ létezik, amelyre $x \in S$). A Pontos fedés ismert NP-teljes feladat.

Legyen $\mathcal{E} = (X, \mathcal{S})$ a Pontos fedés bemenete. Vázolunk egy konstrukciót, ami $\mathcal{E}$-hez táblázatos $q_\mathcal{E}, q'_\mathcal{E}$ lekérdezés párt készít polinomiális időben. Ezt a konstrukciót lehet aztán a különböző NP-teljességi állítások bizonyítására használni.

Az $\mathbf{R}$ séma attribútumai legyenek a páronként különböző $A_1, A_2, \ldots, A_n, B_1, \ldots, B_m$ attribútumok. $q_\mathcal{E} = (\mathbf{T}_\mathcal{E}, t)$ és $q'_\mathcal{E} = (\mathbf{T}'_\mathcal{E}, t)$ az $\mathbf{R}$ séma feletti táblázatos lekérdezések, mindkettő összegzése a $t = \langle A_1 : a_1, \ldots, A_n : a_n \rangle$ szabad sor, ahol $a_1, \ldots, a_n$ páronként különböző változók.

Legyenek $b_1, b_2, \ldots, b_m, c_1, c_2, \ldots c_m$ további páronként különböző változók. $\mathbf{T}_\mathcal{E}$ n sorból áll, X minden elemének megfelel egy. Az x_i elem sorában a_i áll az A_i attribútum oszlopában, b_j a B_j attribútum oszlopában minden olyan j-re, amelyre $x_i \in S_j$ teljesül. A $\mathbf{T}_\mathcal{E}$ n tábla többi helyén csupa különböző új változó áll.

Hasonlóan, $\mathbf{T}'_{\mathcal{E}}$ m sorból áll, $\mathcal{S}$ minden elemének megfelel egy. Az S_j részhalmaz sorában a_i áll az A_i attribútum oszlopában, minden olyan i-re, amelyre $x_i \in S_j$, valamint $c_{j'}$ a $B_{j'}$ attribútum oszlopában, minden $j' \neq j$-re. A $\mathbf{T}'_{\mathcal{E}}$ n tábla többi helyén csupa különböző új változó áll.

A 33.10. kérdés NP-teljessége következik abból, hogy X-nek akkor és csak akkor létezik pontos fedése $\mathcal{S}$-beli halmazokkal, ha $q'_{\mathcal{E}} \sqsubseteq q_{\mathcal{E}}$ teljesül. A bizonyítást, valamint a 33.11. és 33.12. kérdések NP-teljességének bizonyítását az Olvasóra bízzuk (33.1-9 gyakorlat). ■

Gyakorlatok

33.1-1. Bizonyítsuk be a 33.4. állítást, azaz hogy minden szabály alapú q lekérdezés monoton és kielégíthető. *Útmutatás.* A kielégíthetőség bizonyításához legyen a q lekérdezésben szereplő összes konstans halmaza K, $a \notin K$ pedig egy további konstans. Minden (33.3)-beli R_i reláció sémához készítsük el az összes olyan $(a_1, a_2, \ldots, a_r)$ sort, ahol $a_i \in K \cup \{a\}$, és r az R_i attribútumainak száma. Legyen $\mathcal{I}$ az így kapott példány. Lássuk be, hogy $q(\mathcal{I})$ nem üres.

33.1-2. Adjunk meg egy $\mathbf{R}$ relációs sémát és q relációs algebrai lekérdezést $\mathbf{R}$ felett, amelynek eredménye üres halmaz tetszőleges $\mathbf{R}$ feletti példányra.

33.1-3. Bizonyítsuk be, hogy a szabály alapú lekérdezések nyelve és a táblázatos lekérdezések nyelve ekvivalens.

33.1-4. Bizonyítsuk be, hogy minden szabály alapú q lekérdezés, amelyik egyenlőség atomokat is tartalmazhat, vagy ekvivalens a $q^{\emptyset}$ üres lekérdezéssel, vagy létezik egy q' szabály alapú lekérdezés, amely nem tartalmaz egyenlőség atomokat úgy, hogy $q \equiv q'$. Adjunk polinomiális algoritmust, ami egy adott egyenlőségeket is tartalmazó szabály alapú q lekérdezésről eldönti, hogy $q \equiv q^{\emptyset}$ teljesül-e, és ha nem, akkor elkészít egy q' szabály alapú lekérdezést, amely nem tartalmaz egyenlőség atomokat úgy, hogy $q \equiv q'$.

33.1-5. A 33.6. tétel bizonyításának gondolatát általánosítva bizonyítsuk be a 33.8.tételt.

33.1-6. Legyen P datalog program, $\mathcal{I}$ példány $edb(P)$ felett, $C(P, \mathcal{I})$ az $\mathcal{I}$-ben és P-ben szereplő konstansok (véges) halmaza. Legyen $\mathbf{B}(P, \mathcal{I})$ az alábbi $sch(P)$ feletti példány:

1. Minden $edb(P)$-beli R relációra az $R(u)$ tény $\mathbf{B}(P, \mathcal{I})$-ben van pontosan akkor, ha $\mathcal{I}$-ben van, valamint
2. minden $idb(P)$-beli R relációra minden $C(P, \mathcal{I})$-beli konstansokból képzett $R(u)$ tény $\mathbf{B}(P, \mathcal{I})$-ben van.

Bizonyítsuk be, hogy

$$\mathcal{I} \subseteq T_P(\mathcal{I}) \subseteq T_P^2(\mathcal{K}) \subseteq T_P^3(\mathcal{K}) \subseteq \cdots \subseteq \mathbf{B}(P, \mathcal{I}). \tag{33.36}$$

33.1-7. Adjunk példát olyan bemenetre, P datalog programra és $\mathcal{I}$ *edb* példányra, amelyre ugyanazt a sort a FÉLIG-NAIV-DATALOG ciklusának különböző végrehajtásai is előállítják.
33.1-8. Bizonyítsuk be, hogy a JAVÍTOTT-FÉLIG-NAÍV-DATALOG eljárás minden bemenetre véges időben megáll, és helyes eredményt ad. Mutassunk példát olyan bemenetre, amelyiken a JAVÍTOTT-FÉLIG-NAIV-DATALOG kevesebb sort számít ki többször, mint a FÉLIG-NAIV-DATALOG eljárás.
33.1-9.

1. Bizonyítsuk be, hogy a 33.20. tétel bizonyításában szereplő $q_{\mathcal{E}} = (\mathbf{T}_{\mathcal{E}}, t)$ és $q'_{\mathcal{E}} = (\mathbf{T}'_{\mathcal{E}}, t)$ táblázatos lekérdezésekre pontosan akkor létezik homomorfizmus $(\mathbf{T}_{\mathcal{E}}, t)$-ről $(\mathbf{T}'_{\mathcal{E}}, t)$-re, ha az $\mathcal{E} = (X, \mathcal{S})$ PONTOS FEDÉS feladatnak van megoldása.

2. Bizonyítsuk be, hogy a 33.11. és 33.12. kérdések eldöntése NP-teljes feladat.

33.2. Nézetek

Egy adatbázis rendszer felépítésének három fő szintje van:

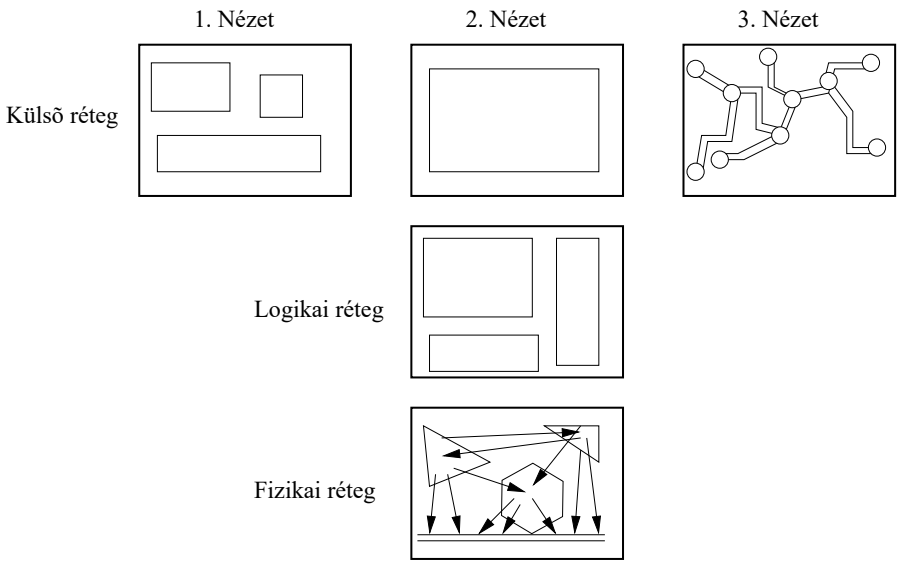
- fizikai réteg;
- logikai réteg;
- külső (felhasználói) réteg.

A szintek elválasztásának célja a fizikai adatfüggetlenség elérése, valamint a felhasználói kényelem. A **??**. ábrán a három nézet lehetséges felhasználói felületeket mutat: multirelációs, univerzális relációs, illetve grafikus felület.

A ***fizikai réteg*** a ténylegesen tárolt adatállományokat, a rájuk épített sűrű és ritka indexeket jelenti.

A ***logikai réteg*** elválasztása a fizikai rétegtől lehetővé teszi, hogy a felhasználó az adatok logikai összefüggéseire koncentráljon, ami sokkal jobban közelíti a modellezni kívánt valóságról alkotott képét. A logikai réteghez tartozik az adatbázis séma leírása a különböző integritási feltételekkel, függőségekkel. Ez az a szint, ahol az adatbázis adminisztrátorok kezelik a rendszert. A logikai és a fizikai réteg közti kapcsolatot az adatbáziskezelő program tartja fent.

A ***külső réteg*** és a logikai réteg elválasztásának célja, hogy a végfelhasználók az adatbázist a saját (szűk) igényeik és szempontjaik szerint lássák. Például egy banki adatbázis esetén a külső réteg egyik nagyon egyszerű nézete



33.2. ábra. Az adatbázis rendszerek felépítésének három szintje.

lehet a pénzkiadó automata, vagy egy jóval bonyolultabb nézete lehet kölcsön igénylés elbírálásához az ügyfél teljes banki hitel története.

33.2.1. Nézet, mint lekérdezés eredménye

Kérdés az, hogy a külső réteghez tartozó nézeteket hogyan adhatjuk meg. Ha egy relációs algebrai kifejezéssel adott lekérdezést úgy tekintünk, mint valamely formulát, amit majd reláció példányokra alkalmazunk, akkor kapjuk a ***nézetet***. A datalog szabályok jól szemléltetik a lekérdezések és nézetek közti különbséget. A szabályok által meghatározott relációkat ***intenzionálisnak*** neveztük, mivel ezek azok a relációk, amelyeknek nem kell külső tárolókon, extenzionálisan léteznüük, szemben az ***extenzionális*** relációkkal.

33.21. definíció. *Az* $\mathbf{R}$ *séma feletti valamely* $\mathcal{Q}$ *lekérdezési nyelven megadott* $\mathcal{V}$ *kifejezést* $\mathbf{R}$ *séma feletti* ***nézetnek*** *nevezünk.*

Természetesen datalog intenzionális relációkhoz hasonlóan nézeteket is használhatunk lekérdezések megadásakor, illetve újabb nézetek meghatározásakor.

33.7. példa. *SQL nézet.* Az SQL adatbázis-kezelő nyelvben az alábbi módon lehet nézetet megadni. Tegyük fel, hogy a **PestiMűsor** sémából számunkra érdekes adat csak annyi, hogy mikor és hol játszanak Kuroszava filmeket. A **Kuroszava-időpontok** nézetet a

KUROSZAVA-IDŐPONTOK

```
1  create view KuroszavaIdőpontok as
2          select Mozi, Időpont
3          from Filmek, Műsor
4          where Filmek.Cím=Műsor.Cím and Filmek.Rendező="Kuroszava Akira"
```

SQL utasítás definiálja. Relációs algebrai alakban a következő.

$$\textit{Kuroszava-időpontok(Mozi, Időpont)} = \tag{33.37}$$

$$\pi_{\textit{Mozi, Időpont}}(\textit{Mozik} \bowtie \sigma_{\textit{Rendező}=\text{"Kuroszava Akira"}}(\textit{Filmek})) \ . \tag{33.38}$$

Végül datalog szabállyal ugyanez

$$\textit{Kuroszava-időpontok}(x_M, x_I = \tag{33.39}$$

$$\textit{Mozik}(x_M, x_C, x_I), \textit{Filmek}(x_C, \text{"Kuroszava Akira"}, x_{Sz}) \ . \tag{33.40}$$

A KUROSZAVA-IDŐPONTOK utasítás 2. sora jelöli ki a használt szelekciós operátort, a 3. sor, hogy melyik két relációt kell összekapcsolni, végül a 4. sor feltétele mutatja meg, hogy természetes összekapcsolásról van szó, nem pedig direkt szorzatról.

Amennyiben a $\mathcal{V}$ nézetet már definiáltuk, akkor a további int akármilyen másik (extenzionális) reláció.

Nézetek használatának előnyei

- Adatok automatikus elrejtése: Olyan adatok, amelyek nem részei a használt nézetnek, nyilván nem is jelennek meg a felhasználó előtt, így azokat nem is olvashatja illetéktelenül, illetve nem is módosíthatja. Tehát azzal, hogy az adatbázis hozzáférést nézeteken keresztül engedjük meg, egyszerű, de hatékony biztonsági mechanizmust üzemeltetünk.
- Nézetek egyszerű „makró képességet” szolgáltatnak. A 33.7 példában definiált KuroszavaIdőpontok nézet használatával könnyedén meg tudjuk keresni, melyik moziban adnak délelőtt Kuroszava filmet:

$$\textit{KuroszavaDélelőtt}(\textit{Mozi}) = \textit{KuroszavaIdőpontok}(\textit{Mozi}, x_I), x_I < 12 \ . \tag{33.41}$$

  Természetesen a felhasználó beírhatná a kódba a *KuroszavaIdőpontok* definícióját közvetlenül, de a makró utasításokkal szoros hasonlatosságban, itt is a kényelmi szempontok az elsők.
- A nézetek lehetővé teszik, hogy ugyanazt az adatot különböző felhasználók különböző módon lássák ugyanabban az időben.

- Nézetek biztosítják a ***logikai adatfüggetlenséget.*** A logikai adatfüggetlenség lényege, hogy a felhasználók és programjaik védettek legyenek az adatbázis séma szerkezeti változtatásaitól. Ezt úgy lehet elérni, hogy a szerkezet-változtatás előtti relációkat mint nézetek definiáljuk a szerkezetváltoztatás utáni új sémában.
- A nézetek lehetővé teszik az ellenőrzött adatbevitelt is. SQL-ben a **create view** utasítás **with check option** záradéka ezt a célt szolgálja.

Materializált nézet

Előfordulhat, hogy valamely nézetet több különböző lekérdezésben is használunk. Ilyen esetekben hasznos lehet, ha nézet által meghatározott reláció(k) sorait nem kell minden esetben újra kiszámolnunk, hanem a nézet definíciójában szereplő lekérdezés eredményét tároljuk, és a további feldolgozásokkor csak beolvassuk. Az ilyen tárolt eredményt nevezzük ***materializált nézetnek.***

Gyakorlatok

33.2-1. Tekintsük az alábbi sémát:

$$Filmsztár(Név, Cím, Nem, SzülDátum)$$
$$FilmMogul(Név, Cím, Igazolvány\#, Vagyon)$$
$$Stúdió(Név, Cím, ElnökIg\#) \quad .$$

A *FilmMogul* reláció a filmszakma nagymenőinek (stúdió elnökök, producerek stb) adatait tartalmazza. Az egyes attribútumok nevei értelemszerűen megadják jelentésüket, illetve az *Igazolvány#* a mogul működési engedélyének száma, az *ElnökIg#* a stúdió elnöke működési engedélyének száma. Adjuk meg az alábbi nézeteket datalog szabállyal, relációs algebra ikifejezéssel, illetve SQL nyelven:

1. *GazdagMogul*: a legalább 100,000,000 forint vagyonú filmmogulok nevét, címét, igazolvány számát és vagyonát listázza.

2. *StúdióElnök*: az olyan filmmogulok nevét, címét, igazolvány számát listázza, akik stúdió elnökök is egyben.

3. *MogulSztár*: az összes olyan személy nevét, címét, igazolvány számát és vagyonát listázza, aki egyszerre filmsztár és filmmogul is.

33.2-2. A **PestiMűsor** séma felett adjuk meg az alábbi nézeteket datalog szabállyal, relációs algebrai kifejezéssel, illetve SQL nyelven:

1. *Marilyn*(*Cím*): Marilyn Monroe szereplésével készült filmek címét listázza.

2. *CorvinInfo*(*Cím*,*Időpont*,*Telefon*): a Corvin moziban játszott filmek címét, vetítési időpontjait, valamint a mozi telefonszámát listázza.

33.3. Lekérdezés átírás

A lekérdezések megválaszolása nézetek felhasználásával, más néven lekérdezések átírása nézetek felhasználásával a közelmúltban nagyon sok figyelmet és érdeklődést kiváltó feladattá vált. Ennek oka a feladat széles körű alkalmazhatósága a legkülönfélébb adatkezelési feladatokban: lekérdezés optimalizálásban, fizikai adatfüggetlenség megvalósításában, adat-, illetve információ-egyesítésnél, valamint adattárházak tervezésénél.

A feladat lényege a következő. Tegyük fel, hogy az $\mathbf{R}$ séma felett adott a Q lekérdezés, valamint $V_1, \ldots, V_n$ nézetek. Meg lehet-e válaszolni a Q lekérdezést pusztán a $V_1, \ldots, V_n$ nézetek eredményeinek ismeretében? Avagy, mi azon sorok legbővebb halmaza, amit a nézetek ismeretében meg tudunk határozni? Ha elérhetjük a nézeteket és az alapséma relációit is, melyik a legolcsóbb kiszámítási mód Q megválaszolására?

33.3.1. Motiváció

Mielőtt a lekérdezés átírás algoritmusait részletesen tárgyalnánk, néhány alkalmazás bemutatásával indokoljuk, hogy miért érdemes a kérdéssel foglalkozni. A példákhoz az alábbi **Egyetem** adatbázist használjuk:

$$\mathbf{Egyetem} = \{Tanár, Kurzus, Tanít, Felvett, Szakirány, Dolgozik, Témavezető\}\ . \tag{33.42}$$

Ahol az egyes relációk sémái a következőek:

$$\begin{aligned} Tanár &= \{TNév, Szakterület\} \\ Kurzus &= \{K\text{-}szám, Cím\} \\ Tanít &= \{TNév, K\text{-}szám, Félév, Véleményezés\} \\ Felvett &= \{Diák, K\text{-}szám, Félév\} \\ Szakirány &= \{Diák, Tanszék\} \\ Dolgozik &= \{TNév, Tanszék\} \\ Témavezető &= \{TNév, Diák\}\ . \end{aligned} \tag{33.43}$$

Feltesszük, hogy a tanárokat, diákokat és tanszékeket a nevük egyértelműen meghatározza, valamint a kurzusokat a számuk. A *Felvett* reláció sorai azt mutatják, hogy melyik diák melyik tárgyat melyik félévben vette fel, a *Szakirány* pedig azt, hogy melyik tanszéket választotta szakosodáskor (feltesszük az egyszerűség kedvéért, hogy egy tanszéken csak egy szakirány van meghirdetve).

Lekérdezés optimalizálás

Ha egy lekérdezés megválaszolásához szükséges számítások egy részét már előzőleg elvégeztük és tároltuk valamely materializált nézetben, akkor használhatjuk a nézetet a lekérdezés megválaszolásának meggyorsítására.

Tekintsük azt a lekérdezést, amelyik azokat a (*Diák,Cím*) párokat keresi, ahol a diák az adott doktorandusz tárgyat felvette, a tárgyat *adatbázis* szakterületű tanár tanítja (választható tárgyak K-száma legalább 400, ebből a doktorandusz tárgyak azok, amelyek K-száma legalább 500).

$$\begin{aligned} val(x_D, x_C) \quad \leftarrow \quad & Tanít(x_T, x_K, x_F, y_1),\ Tanár(x_T, \text{“adatbázis”}), \\ & Felvett(x_D, x_K, x_F),\ Kurzus(x_k, x_C),\ x_K \geq 500\ . \end{aligned} \tag{33.44}$$

Tegyük fel, hogy rendelkezésre áll az alábbi materializált nézet, amely a választható tárgyak regisztrációs adatait tartalmazza:

$$Választható(x_D, x_C, x_K, x_F) \leftarrow Felvett(x_D, x_K, x_F),\ Kurzus(x_K, x_C),\ x_K \geq 400. \tag{33.45}$$

A *Választható* nézet felhasználható (33.44) megválaszolására

$$\begin{aligned} val(x_D, x_C) \quad \leftarrow \quad & Tanít(x_T, x_K, x_F, y_1),\ Tanár(x_T, \text{“adatbázis”}), \\ & Választható(x_D, x_C, x_K, x_F),\ x_K \geq 500\ . \end{aligned} \tag{33.46}$$

(33.46) kiszámítása gyorsabb lesz, mint (33.44)-é, mivel a *Felvett* és a *Kurzus* természetes összekapcsolását már a *Választható* nézet elvégezte, valamint leválasztotta a kötelező tárgyakat (amelyek a regisztráció legnagyobb részét teszik ki a legtöbb egyetemen). Érdemes megjegyezni, hogy a *Választható* nézet annak ellenére használható, hogy ***szintaktikusan*** a (33.44) lekérdezés egyetlen részével sem egyezik meg.

Másfelől azonban előfordulhat, hogy az eredeti lekérdezést gyorsabban meg tudjuk válaszolni. Ha a *Felvett* és a *Kurzus* relációknak a *K-szám* attribútumon létezik indexük, viszont a *Választható*-hoz semmilyen index sincs felépítve, akkor gyorsabb lehet a (33.44) lekérdezést közvetlenül az adatbázis relációkból számítani. Az igazi kihívás tehát nem csak az, hogy eldöntsük egy materializált nézetről, hogy logikailag használható-e valamely lekérdezés megválaszolására, hanem alapos költség elemzést kell végeznünk, hogy mikor érdemes használni a létező nézeteket.

Fizikai adatfüggetlenség

A mai modern adatbázis rendszerek egyik alapelve az adatok logikai szerkezetének és fizikai tárolási módjának szétválasztása. A relációs adatbázis rendszerek esetében, eltekintve a relációk vízszintes vagy függőleges szétvágásától, még mindig lényegileg egy-az-egyhez megfeleltetés van a séma

relációi és az őket tartalmazó fizikai állományok között. Objektum-elvű rendszerek esetében a fizikai-logikai elválasztás elengedhetetlen, mivel a logikai séma jelentős redundanciát tartalmaz, ezért nem felel meg jó fizikai elhelyezésnek. Másik példa a fizikai adatfüggetlenség fontosságára az, amikor a logikai modellt mint közbenső réteget határozzuk meg azután, hogy a fizikai megjelentetést már eldöntöttük. Ez általános amikor XML adatokat tárolunk relációs adatbázisokban, illetve adat egyesítésnél. A STORED rendszer például XML adatokat tárol relációs adatbázisban úgy, hogy nézeteket használ az „XML→relációk" leképezés leírására.

A fizikai adatfüggetlenség fenntartására az egyik elterjedt módszer, hogy az adat tényleges fizikai tárolását a logikai séma feletti nézetek segítségével írjuk le. Például Tsatalos, Solomon és Ioannidis ÁTEU-kat (Általánosított Többszintű Elérési Utakat) használnak az adattárolás leírására.

Az ÁTEU leírja a tárolási szerkezet fizikai szervezését és indexeit. Az első kulcsszó (**as**) leírja a használt adatszerkezetet, amelyben a sorokat tárolják (B^+-fa, hasítási index stb). A leírás többi része a tartalmat adja meg, nézetek megadásához nagyon hasonlóan. A **given** és a **select** kulcsszavak leírják a rendelkezésre álló attribútumokat, ahol a **given** adja meg, hogy melyik attribútumok alapján indexeljük a struktúrát. A **where** kulcsszóval adott nézet definícióban infix jelölést használunk.

A 33.3. ábrán látható példában az A1 ÁTEU olyan (*Diák,Tanszék*) párokat tartalmaz, ahol a *Diák* a *Tanszéket* választotta szakosodáskor. Ezek a párok egy B^+ fával vannak indexelve a *Diák.név* attribútumon. Az A2 ÁTEU egy indexet tárol a diákok nevéből azon kurzusok *K-számaiba,* amelyeket felvettek. Az A3 ÁTEU a kurzusok *K-számaiból* azon *Tanszékekbe* mutató indexet tartalmaz, amely *Tanszékeknél* szakosodott diákok felvették az adott tárgyat.

Mivel az adatokat az ÁTEU-kban leírt adatszerkezetekben tároljuk, felmerül a kérdés, hogyan lehet ezeket az adatszerkezeteket felhasználni lekérdezések megválaszolására. Az ÁTEU-k logikai tartalmát nézetek írják le, ezért a lekérdezés megválaszolása pontosan az a feladat, hogy találjuk meg a lekérdezésnek egy olyan átírását, ami az adott nézeteket használja. Ha több átírás is lehetséges, akkor a legolcsóbb átírást keressük. Jegyezzük meg, hogy a lekérdezés optimalizálással szemben itt ***szükségszerűen*** használnunk kell a nézeteket, mivel az adatokat ÁTEU-kban tároljuk.

Tekintsük a következő lekérdezést, amelyik azon diákok nevét és a szakosodásnál választott tanszékét kérdezi, akik doktorandusz kurzust vettek fel.

$$val(Diák, Tanszék) = \tag{33.47}$$

$$Felvett(Diák, K\text{-}szám, y), Szakirány(Diák, Tanszék), K\text{-}szám \geq 500\ . \tag{33.48}$$

def.áteu A1 **as** b$^+$-fa **by**
given *Diák.Név*
select *Tanszék*
where *Diák* szakosodik *Tanszék*

def.áteu A2 **as** b$^+$-fa **by**
given *Diák.Név*
select *Kurzus.K-szám*
where *Diák* felvett *Kurzus*

def.áteu A3 **as** b$^+$-fa **by**
given *Kurzus.K-szám*
select *Tanszék*
where *Diák* felvett *Kurzus* **and** *Diák* szakosodik *Tanszék*

33.3. ábra. Az **Egyetem** tartomány ÁTEU-i.

A lekérdezést két módon is megválaszolhatjuk. Először, mivel a *Diák.név* egyértelműen meghatározza a diákot, vehetjük A1 és A2 természetes összekapcsolását, majd alkalmazhatunk egy kiválasztási operátort, amivel kiválasztjuk azokat a sorokat, amelyekre $K\text{-}szám \geq 500$, végül egy vetítéssel kiküszöbölhetjük a szükségtelen attribútumokat. Másik végrehajtási terv lehet, hogy A3-at és A2-t kapcsoljuk össze, majd elvégezzük a $K\text{-}szám \geq 500$ szelekciót. Ez utóbbi megoldás hatékonyabb lehet, mert A3 tartalmaz indexet a *K-szám* attribútumon, így a közbenső összekapcsolások sokkal gyorsabbak lehetnek.

Adategyesítés

Egy ***adategyesítési rendszer*** (más néven ***adatközvetítő rendszer***) célja, hogy ***egységes*** lekérdezési felületet biztosítson nagyszámú és eltérő szerkezetű adatforráshoz. Legfontosabb példák a vállalati integráció, több különböző webes forrás egyidejű lekérdezése, valamint elosztott tudományos kísérletek eredményének egyesítése.

Az egységes lekérdezési felület elérése érdekében az adategyesítési rendszer a felhasználó felé egy ***közvetített sémát*** mutat. A közvetített séma ***virtuális*** relációkból áll, abban az értelemben, hogy fizikai valóságukban ezeket a relációkat sehol nem tárolják ebben a formában. A közvetített sémát az adategyesítési alkalmazás szempontjából kell egyedileg megtervezni. Ahhoz, hogy a lekérdezéseket meg tudja válaszolni, a rendszernek tartalmaznia kell ***forrás leírásokat.*** Egy adatforrás leírása meghatározza a forrás tartalmát, a benne

megtalálható attribútumokat, valamint a forrás tartalmára vonatkozó integritási feltételeket. Az adatforrás leírás egyik elterjedt megközelítési módja, hogy a forrás tartalmát a közvetített séma feletti ***nézetként*** adjuk meg. Ez lehetővé teszi új adatforrások beillesztését, illetve az integritási feltételek megadását.

A lekérdezés megválaszolásához az adategyesítési rendszernek a közvetített sémában megfogalmazott lekérdezést le kell fordítania olyanra, amelyik közvetlenül az adatforrásokra hivatkozik. Mivel a források nézetként adottak, a fordítás azzal egyenértékű, hogy a lekérdezést nézetek segítségével válaszoljuk meg.

Példaként tekintsük az **Egyetem** sémát, mint a felhasználó felé közvetített sémát, azzal a különbséggel, hogy a *Tanít* és a *Kurzus* relációknak eggyel több attribútumuk van, nevezetesen az *Egyetem* attribútum, amelyik megmondja, hogy az adott tárgyat melyik egyetemen tanítják:

$$\begin{aligned} Kurzus &= \{K\text{-}szám, Cím, Egyetem\} \\ Tanít &= \{TNév, K\text{-}szám, Félév, Véleményezés, Egyetem\} \end{aligned} \tag{33.49}$$

Tegyük fel, hogy az alábbi két adatforrás áll rendelkezésre. Az első az összes „Adatbázisok"' című tárgyakat, és azok előadóját listázza az összes egyetemről. A következő nézet meghatározással írható le:

$$\begin{aligned} DBkurz(Cím, Tnév, K\text{-}szám, Egyetem) = Kurzus(&K\text{-}szám, Cím, Egyetem), \\ &Tanít(TNév, K\text{-}szám, Félév, \\ &Véleményezés, Egyetem), \\ &Cím = \text{“Adatbázisok”} \ . \end{aligned} \tag{33.50}$$

A második adatforrás a Budapesti Műszaki és Gazdaságtudományi Egyetem doktorandusz kurzusait adja meg, az alábbi módon.

$$\begin{aligned} BMEPhD(Cím, Tnév, K\text{-}szám, Egyetem) = Kurzus(&K\text{-}szám, Cím, Egyetem), \\ &Tanít(TNév, K\text{-}szám, Félév, \\ &Véleményezés, Egyetem), \\ &Egyetem = \text{“BME”}, K\text{-}szám \geq 500. \end{aligned} \tag{33.51}$$

Ha az adategyesítési rendszertől azt kérdezzük, hogy ki tanít *Adatbázisokat* a Műegyetemen, akkor az a lekérdezést egyszerűen megválaszolhatja a *DBkurz* relációra alkalmazott szelekciós operátorral:

$$Val(Tnév) = DBkurz(Cím, Tnév, K\text{-}szám, Egyetem), Egyetem = \text{"BME"} \ . \tag{33.52}$$

Azonban, ha azt szeretnénk tudni, hogy milyen választható (nem csak adatbázis) tárgyak léteznek a Műegyetemen, akkor az adategyesítési rendszer nem tudja a lekérdezés eredményéhez tartozó összes sort megtalálni,

mivel csak a (33.50) és a (33.51) források állnak a rendelkezésére. Ehelyett, a források alapján meghatározható legbővebb sorhalmazt keresheti meg. Azaz, meghatározhatja az összes műegyetemi választható *adatbázis* kurzust a *DBkurz* forrásból, valamint a doktorandusz tárgyakat a *BMEPhD* forrásból. Tehát a következő nemrekurzív datalog program megadja a legbővebb választ:

$$\begin{aligned} &val(Cím, K\text{-}szám) \leftarrow DBkurz(Cím, Tnév, K\text{-}szám, Egyetem), \\ &\qquad\qquad Egyetem = \text{"BME"}, K\text{-}szám \geq 400 \\ &val(Cím, K\text{-}szám) \leftarrow BMEPhD(Cím, Tnév, K\text{-}szám, Egyetem)\ . \end{aligned} \tag{33.53}$$

Vegyük észre, hogy azok a választható tárgyak, amelyek nem doktorandusz tárgyak, vagy nem adatbázis témájúak, nem szerepelnek az eredményben. A lekérdezés optimalizálás és fizikai adatfüggetlenség esetében ***ekvivalens*** lekérdezés átírást kellett találni, itt olyan lekérdezés kifejezést keresünk, amelyik a nézetekből kapható ***legbővebb*** eredményhalmazt találja meg.

Szemantikus gyorstárolás

Kliens-szerver felépítésű adatbázis elérés esetén a kliens által már letöltött adatok szemantikusan modellezhetők, mint bizonyos nézetek eredményei. Tehát a kliens gépen eltárolt adatokat nem mint fizikai adategységeket, lapokat, sorokat, hanem mint nézeteket tekintjük. Ekkor annak eldöntésére, hogy a kliens újabb lekérdezésének megválaszolásához mely további adatok letöltése szükséges, a szervernek azt a feladatot kell megoldania, hogy a kliens oldalon létező nézetek segítségével a lekérdezés mely részei válaszolhatók meg.

33.3.2. Átírás bonyolultsági kérdései

Ebben az alfejezetben a lekérdezés átírás ***elméleti*** bonyolultságát tárgyaljuk. Elsősorban konjunktív lekérdezésekkel foglalkozunk. Megkülönböztetjük a ***minimális*** és a ***teljes*** átírásokat. Belátjuk, hogy ha a lekérdezés konjunktív és a nézetek is konjunktív lekérdezések materializált eredményeként adottak, akkor az átírási probléma *NP-teljes,* feltéve, hogy sem a lekérdezés, sem a nézetek nem tartalmaznak összehasonlítási atomokat. A konjunktív lekérdezéseket szabály alakban tekintjük, ez a legkényelmesebb számunkra.

Tegyük fel, hogy Q lekérdezés adott a $\mathbf{R}$ séma felett.

33.22. definíció. *A Q' konjunktív lekérdezés a Q lekérdezés $\mathcal{V} = V_1, V_2, \ldots, V_m$ nézeteket használó* ***átírása,*** *ha*

- *Q és Q' ekvivalensek, és*
- *Q' egy, vagy több $\mathcal{V}$-beli literált tartalmaz.*

Azt mondjuk, hogy Q' ***lokálisan minimális,*** *ha Q'-ből nem hagyható el literál anélkül, hogy az ekvivalencia megsérülne. Az átírás* ***globálisan minimális,*** *ha nem létezik kevesebb literált használó átírás. (A literálok számába az összehasonlítási atomok $=, \neq, \leq, <$ nem számítanak bele!)*

33.8. példa. *Lekérdezés átírás.* Tekintsük a következő Q lekérdezést és V nézetet.

$$\begin{array}{lll} Q\colon q(X,U) & \leftarrow & p(X,Y), p_0(Y,Z), p_1(X,W), p_2(W,U) \\ V\colon v(A,B) & \leftarrow & p(A,C), p_0(C,B), p_1(A,D)\ . \end{array} \tag{33.54}$$

Q átírható V használatával:

$$Q'\colon q(X,U) \leftarrow v(X,Z), p_1(X,W), p_2(W,U)\ . \tag{33.55}$$

A V nézet a Q lekérdezés első két literálját helyettesíti. Vegyük észre, hogy a lekérdezés harmadik literálját is biztosan kielégíti a nézet, azonban nem hagyható el az átírásból, mert a D változó már nem szerepel V fejében, ezért ha a p_1 literált is elhagynánk, akkor a p_1 és p_2 közti természetes összekapcsolást már nem kényszerítené ki semmi.

Mivel az alkalmazások egy részében az adatbázis relációkat nem érjük el, csak a nézeteket, például az adategyesítés és adattárházak esetében, ezért szükségünk van a ***teljes átírás*** fogalmára.

33.23. definíció. *A Q lekérdezés $\mathcal{V} = V_1, V_2, \ldots, V_m$ nézeteket használó Q' átírása* ***teljes átírás,*** *ha Q' csak $\mathcal{V}$-beli literálokat és összehasonlítási atomokat tartalmaz.*

33.9. példa. *Teljes átírás.* Tegyük fel, hogy a 33.8 példában szereplő V nézet mellett még a

$$V_2\colon v_2(A,B) \leftarrow p_1(A,C), p_2(C,B), p_0(D,E) \tag{33.56}$$

nézet is adott. A Q lekérdezés teljesen átírható:

$$Q''\colon q(X,U) \leftarrow v(X,Z), v_2(X,U)\ . \tag{33.57}$$

Fontos látnunk, hogy ezt az átírást nem kaphatjuk meg ***lépésenként,*** először csak V-t használva, majd megpróbálni V_2-t beépíteni (vagy éppen a másik sorrendben), hiszen p_0 reláció a V_2-ben szereplő p_0 reláció nem szerepel Q'-ben. Tehát a teljes átírás megtalálásához a két nézet használatát egyszerre, ***párhuzamosan*** kell tekinteni.

A lekérdezés átírás megtalálása szoros kapcsolatban áll a lekérdezések közti tartalmazás eldöntésének feladatával. Ez utóbbit táblázatos

lekérdezésekre már a 33.1.3. pontban tárgyaltuk. Táblázatos lekérdezések közti homomorfizmus értelmezhető szabály alapú konjunktív lekérdezésekre is. Az egyetlen különbség, hogy ebben a fejezetben nem követeljük meg, hogy a szabály fejét a homomorfizmus a másik szabály fejére képezze. (A táblázatos lekérdezés összegző sorának a szabály feje felel meg.) A 33.20. tétel szerint annak eldöntése, hogy a Q_1 konjunktív lekérdezés tartalmazza-e Q_2 konjunktív lekérdezést, NP-teljes. Ez igaz marad akkor is, ha Q_2 összehasonlítási atomokat is tartalmaz. Azonban, ha Q_1 is tartalmaz összehasonlítási atomokat, akkor homomorfizmus létezése Q_1-ről Q_2-re csak elégséges feltételt ad a lekérdezések tartalmazására, amelyik ebben az esetben Π_2^p-teljes feladat. Ez utóbbi feladatosztály tárgyalása túlmutat a fejezet keretein, ezért nem részletezzük. A következő állítás szükséges és elégséges feltételt ad arra, hogy létezik-e a Q lekérdezésnek a V nézetet használó átírása.

33.24. állítás. *Tegyük fel, hogy Q és V konjunktív lekérdezések, amelyek tartalmazhatnak összehasonlítási atomokat is. Akkor és csak akkor létezik Q-nak V-t használó átírása, ha $\pi_\emptyset(Q) \subseteq \pi_\emptyset(V)$, azaz V vetítése az üres attribútum halmazra tartalmazza Q vetítését.*

Bizonyítás. Vegyük észre, hogy $\pi_\emptyset(Q) \subseteq \pi_\emptyset(V)$ ekvivalens az alábbi állítással: Ha V eredménye üres halmaz valamely adatbázis példányon, akkor Q eredménye is üres.

Tegyük fel először, hogy létezik átírás, azaz olyan Q-val ekvivalens szabály, amelynek testében V szerepel. Ha r olyan adatbázis példány, amelyiken V eredménye üres halmaz, akkor minden olyan szabály eredménye is üres, amelyiknek testében V szerepel.

Tegyük fel fordítva, hogy ha V eredménye üres halmaz valamely adatbázis példányon, akkor Q eredménye is üres. Legyen

$$\begin{array}{lcl} Q\colon q(\tilde{x}) & \leftarrow & q_1(\tilde{x}), q_2(\tilde{x}), \ldots, q_m(\tilde{x}) \\ V\colon v(\tilde{a}) & \leftarrow & v_1(\tilde{a}), v_2(\tilde{a}), \ldots, v_n(\tilde{a})\ . \end{array} \tag{33.58}$$

Legyen $\tilde{y}$ az $\tilde{x}$-beli változóktól diszjunkt változók listája. Ekkor a

$$Q'\colon q'(\tilde{x}) \leftarrow q_1(\tilde{x}), q_2(\tilde{x}), \ldots, q_m(\tilde{x}), v_1(\tilde{y}), v_2(\tilde{y}), \ldots, v_n(\tilde{y}) \tag{33.59}$$

lekérdezésre teljesül, hogy $Q \equiv Q'$. Világos, hogy $Q' \subseteq Q$. Másfelől, ha valamely r adatbázis példányra létezik az $\tilde{y}$ változóinak olyan kiértékelése, amelyik kielégíti V testét, akkor ezt rögzítve, az $\tilde{x}$-beli változók tetszőleges kiértékelésére pontosan akkor kapunk egy eredménysort Q-ban, amikor a rögzített $\tilde{y}$ kiértékeléssel együtt Q'-ben. ∎

A 33.24. állítás és a 33.20. tétel következménye az alábbi tétel.

33.25. tétel. *Legyen Q konjunktív lekérdezés, amelyik tartalmazhat összehasonlítási atomokat, $\mathcal{V}$ nézetek halmaza. Ha a $\mathcal{V}$-beli nézetek összehasonlítási atomokat nem tartalmazó konjunktív lekérdezéssel adottak, akkor NP teljes annak eldöntése, hogy létezik-e Q-nak $\mathcal{V}$-t használó átírása.*

A 33.25. tétel bizonyítását az Olvasóra bízzuk (33.3-1 gyakorlat).

A 33.24. állítás bizonyításában új változókat vezettünk be. Azonban, a következő lemma szerint erre nincs szükség. Másik fontos észrevétel, hogy elegendő az eredeti lekérdezésben szereplő adatbázis relációk egy részhalmazát tekinteni, amikor lokálisan minimális átírást keresünk, új adatbázis relációkat nem kell felhasználni.

33.26. lemma. *Legyen Q konjunktív lekérdezés, amelyben nem szerepelnek összehasonlítási atomok*

$$Q\colon q(\tilde{X}) \leftarrow p_1(\tilde{U}_1), p_2(\tilde{U}_2), \ldots, p_n(\tilde{U}_n)\ , \tag{33.60}$$

valamint legyen $\mathcal{V}$ nézetek halmaza, szintén összehasonlítási atomok nélkül.

1. *Ha Q $\mathcal{V}$-t használó lokálisan minimális átírása Q', akkor a Q'-ben szereplő adatbázis literálok halmaza izomorf a Q-beli adatbázis literálok egy részhalmazával.*

2. *Ha*

$$q(\tilde{X}) \leftarrow p_1(\tilde{U}_1), \ldots, p_n(\tilde{U}_n), v_1(\tilde{Y}_1), v_2(\tilde{Y}_2), \ldots v_k(\tilde{Y}_k) \tag{33.61}$$

a Q egy a nézeteket használó átírása, akkor létezik egy

$$q(\tilde{X}) \leftarrow p_1(\tilde{U}_1), \ldots, p_n(\tilde{U}_n), v_1(\tilde{Y'}_1), \ldots v_k(\tilde{Y'}_k) \tag{33.62}$$

átírás is, amelyre teljesül, hogy $\{\tilde{Y'}_1 \cup \cdots \cup \tilde{Y'}_k\} \subseteq \{\tilde{U}_1 \cup \cdots \cup \tilde{U}_n\}$, azaz az átírás nem vezet be új változókat.

A 33.26. lemma bizonyításának részleteit az Olvasóra hagyjuk (33.3-2 gyakorlat). A következő lemma alapvető fontosságú: A Q $\mathcal{V}$-t használó minimális átírása nem ***növelheti*** a literálok számát.

33.27. lemma. *Legyen Q konjunktív lekérdezés, $\mathcal{V}$ konjunktív lekérdezésekkel megadott nézetek halmaza, mindkettő összehasonlítási atomok nélkül. Ha Q teste p literált tartalmaz, és Q' a Q $\mathcal{V}$-t használó lokálisan minimális teljes átírása, akkor Q' legfeljebb p literált tartalmaz.*

Bizonyítás. A Q'-beli nézet literálok helyére írjuk be a definíciójukat, így kapjuk a Q'' lekérdezést. Legyen φ homomorfizmus Q testéből Q''-be. φ létezik a Homomorfizmus tétel (33.18. tétel) és $Q \equiv Q''$ alapján. A Q testében szereplő $l_1, l_2, \ldots, l_p$ literálok mindegyike legfeljebb egy nézet literál kifejtéséből kapott tagra képződik. Ha Q'-ben több, mint p nézet literál szerepel, akkor Q'' testében néhány nézet literál kifejtése diszjunkt φ képétől. Ezek a nézet literálok elhagyhatók Q'-ből, úgy, hogy az ekvivalencia nem változik. ■

A 33.27. lemma alapján a következő tétel mondható ki minimális átírások bonyolultságáról.

33.28. tétel. *Legyen Q konjunktív lekérdezés, $\mathcal{V}$ konjunktív lekérdezésekkel megadott nézetek halmaza, mindkettő összehasonlítási atomok nélkül. Tegyük fel, hogy Q testében p literál szerepel.*

1. *Annak eldöntése, hogy létezik-e Q-nak $\mathcal{V}$-t használó Q' átírása, amelyik legfeljebb k ($\leq p$) literált használ, NP-teljes.*
2. *Annak eldöntése, hogy létezik-e Q-nak $\mathcal{V}$-t használó Q' átírása, amelyik legfeljebb k ($\leq p$) adatbázis literált használ, NP-teljes.*
3. *Annak eldöntése, hogy létezik-e Q-nak $\mathcal{V}$-t használó teljes átírása, NP-teljes.*

Bizonyítás. Az első állítást bizonyítjuk, a másik kettő bizonyítása hasonló. A 33.27. és 33.26. lemmák alapján csak olyan átírásokat kell tekintenünk, amelyekben legfeljebb annyi literál szerepel, mint a lekérdezésben, a lekérdezés literáljainak egy részhalmazát tartalmazza, és nem használ új változókat. Egy ilyen átírást, valamint az ekvivalenciát bizonyító homomorfizmusokat polinomiális időben tudunk ellenőrizni, tehát a feladat NP-beli. Az NP-nehézség bizonyításához a 33.25. tételt használjuk. Adott Q lekérdezéshez és V nézethez legyen V' az a nézet, amelyiknek a feje megegyezik V fejével, a teste pedig Q és V testének konjunkciója. Könnyen látható, hogy pontosan akkor létezik V'-t használó átírás egyetlen literállal, amikor létezik V-t használó (korlátozások nélküli) átírás. ■

33.3.3. Gyakorlati algoritmusok

Ebben a fejezetben csak teljes átírásokkal foglalkozunk. Ez nem jelent igazi korlátozást, mert ha adatbázis literálokat is szeretnénk használni, akkor bevezethetünk olyan nézeteket, amelyek egy az egyben tükrözik az adatbázis relációkat. A 33.22. definícióban bevezetett ***ekvivalens*** átírás fogalma megfelelő, ha az átírás célja lekérdezés optimalizálás, illetve a fizikai adatfüggetlenség

biztosítása. Azonban, adategyesítési, illetve adattárház környezetben nem törekedhetünk arra, hogy az átírás ekvivalens legyen, mert nem feltétlenül áll rendelkezésre minden adat. Ezért bevezetjük a maximálisan tartalmazott átírás fogalmát, amelyik függ attól, melyik lekérdezési nyelvet használjuk, ellentétben az ekvivalens átírásokkal.

33.29. definíció. *Legyen Q lekérdezés, $\mathcal{V}$ nézetek halmaza, $\mathcal{L}$ pedig lekérdezési nyelv. Q-nak $\mathcal{V}$-t használó $\mathcal{L}$-re vonatkozó* ***maximálisan tartalmazott átírása*** *Q', ha*

1. *Q' az $\mathcal{L}$ nyelv olyan lekérdezése, amelyik csak a $\mathcal{V}$-beli nézeteket használja,*
2. *Q tartalmazza Q'-t,*
3. *ha $Q_1 \in \mathcal{L}$ lekérdezésre teljesül, hogy $Q' \sqsubseteq Q_1 \sqsubseteq Q$, akkor $Q' \equiv Q_1$.*

Lekérdezés optimalizálás materializált nézetek használatával

Mielőtt rátérünk arra, hogyan lehet egy hagyományos optimalizáló eljárást módosítani, hogy adatbázis relációk helyett materializált nézetekkel dolgozzon, át kell tekintenünk, mikor használható egy nézet valamely lekérdezés megválaszolásához. Lényegileg a V nézet használható a Q lekérdezéshez, ha a V definíciójában szereplő adatbázis relációk és a Q-ban szereplő relációk halmazainak közös része nem üres, és V olyan attribútumokat is kiválaszt, amelyeket Q is. Ezen kívül, ekvivalens átírás esetén, ha V-ben vannak összehasonlítási atomok olyan attribútumokra, amelyek Q-ban is szerepelnek, akkor a nézetnek logikailag ekvivalens, vagy gyengébb összehasonlítási feltételt kell alkalmaznia, mint a lekérdezés. Ha logikailag erősebb feltételt alkalmaz, akkor a nézet egy (maximálisan) tartalmazott átírás része lehet. Ezt legegyszerűbben egy példán keresztül világíthatjuk meg. Tekintsük a Q lekérdezést az **Egyetem** séma felett, amelyik az olyan tanár, diák, félév hármasokat sorolja fel, ahol a tanár a diák témavezetője és az adott félévben a diák a tanár valamelyik óráját felvette.

$$Q\colon q(\mathit{Tnév},\mathit{Diák},\mathit{Félév}) = \mathit{Felvett}(\mathit{Diák},\mathit{K\text{-}szám},\mathit{Félév}),\ \mathit{Témavezető}(\mathit{Tnév},\mathit{Diák}),\ \mathit{Tanít}(\mathit{Tnév},\mathit{K\text{-}szám},\mathit{Félév},x_V),\ \mathit{Félév} \geq \text{“2000ősz”}\ . \tag{33.63}$$

Az alábbi V_1 nézet használható Q megválaszolásánál, mivel ugyanazt az összekapcsolási feltételt használja a *Felvett* és *Tanít* relációkra, mint Q, amint azt az azonos nevű változók mutatják. Ezen kívül, V_1 kiválasztja a *Diák, Tnév, Félév* attribútumokat, ami szükséges ahhoz, hogy a *Témavezető* relációval

megfelelően összekapcsolhassuk, és a végeredménybe kiválaszthassuk a három attribútumot. Végül, a *Félév* > "1999ősz" összehasonlítási atom logikailag gyengébb feltétel, mint a Q-ban szereplő *Félév* ≥ "2000ősz".

$$V_1 : v_1(\mathit{Diák}, \mathit{Tnév}, \mathit{Félév}) = \mathit{Tanít}(\mathit{Tnév}, \mathit{K\text{-}szám}, \mathit{Félév}, x_V), \\ \mathit{Felvett}(\mathit{Diák}, \mathit{K\text{-}szám}, \mathit{Félév}), \mathit{Félév} > \text{"1999ősz"} . \tag{33.64}$$

A következő négy nézet mutatja, hogy V_1-t csak kicsit módosítva hogyan változik a felhasználhatóság.

$$V_2 : v_2(\mathit{Diák}, \mathit{Félév}) = \mathit{Tanít}(x_T, \mathit{K\text{-}szám}, \mathit{Félév}, x_V), \\ \mathit{Felvett}(\mathit{Diák}, \mathit{K\text{-}szám}, \mathit{Félév}), \mathit{Félév} > \text{"1999ősz"} . \tag{33.65}$$

$$V_3 : v_3(\mathit{Diák}, \mathit{Tnév}, \mathit{Félév}) = \mathit{Tanít}(\mathit{Tnév}, \mathit{K\text{-}szám}, x_F, x_V), \\ \mathit{Felvett}(\mathit{Diák}, \mathit{K\text{-}szám}, \mathit{Félév}), \mathit{Félév} > \text{"1999ősz"} . \tag{33.66}$$

$$V_4 : v_4(\mathit{Diák}, \mathit{Tnév}, \mathit{Félév}) = \mathit{Tanít}(\mathit{Tnév}, \ \mathit{K\text{-}szám}, \mathit{Félév}, x_V), \\ \mathit{Témavezető}(\mathit{Tnév}, x_D), \mathit{Tanár}(\mathit{Tnév}, x_{Sz}), \\ \mathit{Felvett}(\mathit{Diák}, \mathit{K\text{-}szám}, \mathit{Félév}), \mathit{Félév} > \text{"1999ősz"} . \tag{33.67}$$

$$V_5 : v_5(\mathit{Diák}, \mathit{Tnév}, \mathit{Félév}) = \mathit{Tanít}(\mathit{Tnév}, \ \mathit{K\text{-}szám}, \mathit{Félév}, x_V), \\ \mathit{Felvett}(\mathit{Diák}, \mathit{K\text{-}szám}, \mathit{Félév}), \mathit{Félév} > \text{"2001ősz"} . \tag{33.68}$$

A V_2 nézet majdnem ugyanaz, mint V_1, az egyetlen különbség, hogy nem választja ki a *Tnév* attribútumot a *Tanít* relációból. Az azonban szükséges a *Témavezető* relációval való természetes összekapcsoláshoz, valamint Q eredményében is szerepel. Így, ha használni akarjuk V_2-t az átíráshoz, akkor újra össze kell kapcsolni a *Tanít* relációval. Azonban, ha a *Felvett* és a *Tanít* relációk természetes összekapcsolása már csak kevés sort tartalmaz az eredeti relációk sorszámához képest, akkor megérheti az újabb összekapcsolás.

A V_3 nézetben a *Felvett* és a *Tanít* relációk összekapcsolása csak a *K-szám* attribútum szerint történik, a *Félév* és a x_F változók egyenlőségét nem követeli meg. Mivel az x_F attribútumot V_3 nem választja ki, ezért nem lehet később összekapcsolási feltételben alkalmazni, így V_3 használata nem jelent nyereséget.

A V_4 nézet csak olyan tanárokat vesz figyelembe, akiknek van legalább

egy szakterületük. Ezzel több feltételt alkalmaz, mint az eredeti Q lekérdezés, tehát nem alkalmazható ekvivalens átírásra, ha nem engedjük meg egyesítés és tagadás használatát is a lekérdezési nyelvben. Azonban, ha az adatbázisban van olyan integritási feltétel, hogy minden tanárnak van legalább egy szakterülete, akkor a lekérdezés optimalizálónak észre kell vennie, hogy V_4 használható.

Végül, V_5 összehasonlítási operátora erősebb feltételt használ, mint a Q-ban szereplő, így ekvivalens átíráshoz nem, csak (maximálisan) tartalmazott átíráshoz használható.

System-R stílusú optimalizálás

Mielőtt a hagyományos optimizálás változtatásait tárgyalnánk, röviden összefoglaljuk hogyan dolgozik a ***System-R stílusú optimizáló.*** A legjobb végrehajtási sorrendet alulról felfelé építkezve keresi meg. Első fázisban 1 méretű rész-lekérdezéseket tekint, azaz a lekérdezésben szereplő minden egyes relációs táblához megkeresi a legjobb elérési utat. Az n-edik fázisban N méretű végrehajtási terveket tekint, amelyeket kisebbek (k és $n-k$ méretűek) kombinációjával kap. Az eljárás akkor fejeződik be, ha olyan tervet talál, amelyik a lekérdezés összes relációját lefedi. Az eljárás hatékonysága abból adódik, hogy a végrehajtási terveket ***ekvivalencia osztályokra*** bontja, és minden osztályból csak egyetlen tervet tekint. Két terv ugyanabban az osztályban van, ha

- a lekérdezésben szereplő relációk közül ugyanazokat fedik le (tehát ugyanaz a méretük), valamint
- a választ ugyanabban a (lekérdezéstől függő) érdekes sorrendben adják.

A mi esetünkben az optimizáló a lekérdezés végrehajtási tervét nem adatbázis relációkra, hanem nézetekre alapozza. Ezért a szokásosan rendelkezésre álló meta-adatokon kívül (pl. statisztikák, indexek) az optimalizáló rendelkezésére állnak a nézeteket definiáló lekérdezés kifejezések is. Az alábbi változtatásokra van szükség.

A. A lekérdezés megválaszolásához használható nézeteket ki kell választani, a fentiekben vázolt feltételek alapján. A hagyományos optimizáló esetében ez triviális, hiszen egy adatbázis reláció pontosan akkor használható a lekérdezés megválaszolásához, ha szerepel a lekérdezésben.

B. Mivel a lekérdezés végrehajtás terve nézetek összekapcsolását jelenti, nem pedig adatbázis relációkét, a tervek nem oszthatók szépen ekvivalencia osztályokba, mint a hagyományos esetben, és így nem lehet őket méret szerint növekvő sorrendben végignézni. Ezért a következő módosítások szükségesek.

1. ***Megállási feltétel:*** Az eljárás megkülönbözteti a *részleges lekérdezés végrehajtási terveket* a *teljes lekérdezés végrehajtási tervektől.* A lehetséges összekapcsolási sorrendek végigtekintése akkor fejeződik be, ha már nincs több ellenőrizetlen részleges lekérdezés végrehajtási terv. Ezzel szemben a hagyományos optimizáló eljárás akkor ér éget, ha áttekintette azon ekvivalencia osztályokat, amelyek a lekérdezés összes relációját tartalmazzák.

2. ***Végrehajtási tervek elhagyása:*** A hagyományos optimizáló eljárás az *egy ekvivalencia osztályba* tartozó terveket hasonlítja össze páronként, és csak a legolcsóbbat tárolja el minden osztályból. A mi esetünkben *tetszőleges* két eddig előállított tervet hasonlít össze. A p tervet elhagyja, ha létezik olyan p' terv, amelyikre igaz, hogy

   (a) p' olcsóbb, mint p, és

   (b) p' legalább annyit hozzájárul a lekérdezés megválaszolásához, mint p. Ez lényegileg azt jelenti, hogy legalább annyi adatbázis relációt lefed, mint p, és legalább annyi szükséges attribútumot ki is választ.

3. ***Részleges tervek társítása:*** A hagyományos esetben, ha két részleges tervet társítunk egy nagyobb tervvé, akkor a hozzájuk tartozó összekapcsolási feltétel egyértelműen adott a lekérdezésben, az optimalizáló eljárás csak a leghatékonyabb megvalósítást kell megtalálja. A mi esetünkben azonban *a priori* – előzetesen – egyáltalán nem világos, hogy milyen összekapcsolási feltétel eredményez ekvivalens átírást. Tehát az optimalizáló eljárás több, különböző összekapcsolási feltételt kell megvizsgáljon. Szerencsére, a gyakorlatban a rendelkezésre álló meta-adatok lényegesen szűkítik a vizsgálandó feltételek körét. Például, nincs túl sok értelme megpróbálni összekapcsolni egy szöveg típusú attribútumot egy numerikus attribútummal. Hasonlóan az integritási feltételek is csökkenthetik a számba vehető összekapcsolások számát. Miután az összes lehetséges összekapcsolást végigvizsgálta, az optimalizáló azt is ellenőrzi, hogy a kapott terv még mindig a lekérdezés részleges megoldása-e.

Fentieket az alábbi összehasonlító táblázatban foglalhatjuk össze.

Hagyományos optimalizáló	Nézeteket használó optimalizáló
1. Fázis	***1. Fázis***
a) Keressük meg az összes lehetséges elérési utat.	a1) Keressük meg az összes nézetet, amelyik használható a lekérdezés megválaszolásához a2) Különböztessük meg a részleges és teljes terveket.
b) Hasonlítsuk össze a költségüket és tartsuk meg a legolcsóbbat.	b) Hasonlítsuk össze páronként a nézeteket. Ha valamelyik nem járul többel hozzá a lekérdezés megválaszolásához mint egy másik, és nem is olcsóbb annál, akkor hagyjuk el.
c) Ha a lekérdezésben egy reláció szerepel, **stop**.	c) Ha nincs részleges megvalósítási terv, **stop**.
2. Fázis	***2. Fázis***
Tekintsük az előző fázisban talált elérési utak összekapcsolásait, amelyek a lekérdezésnek megfelelőek, az összes lehetséges összekapcsolási eljárással.	a1) Tekintsük az előző fázisban talált részleges megoldások összekapcsolásait minden lehetséges összekapcsolási eljárással, minden lehetséges összekapcsolási feltételt alkalmazva. a2) Különböztessük meg a részleges és teljes terveket.
b) Hasonlítsuk össze a kapott összekapcsolási tervek költségét, és tartsuk meg a legolcsóbbat.	b) Ha valamelyik újonnan előállított megoldás nem használható a lekérdezés megválaszolásához, vagy valamelyik másik minden tekintetben jobb nála, akkor hagyjuk el.
c) Ha a lekérdezésben két reláció szerepel, **stop**.	c) Ha nincs részleges megvalósítási terv, **stop**.
3. Fázis	***3. Fázis***
⋮	⋮

Ekvivalens átírások másik módozata az átalakítási szabályok alkalmazása. A közös gondolat, hogy a hagyományos optimizáló átalakítási szabályaihoz hozzáveszik azt, hogy a lekérdezés valamely részét helyettesíteni lehet egy nézettel. Ezekkel részletesebben nem foglalkozunk.

A fentiekben tárgyalt optimalizáló eljárások elsősorban olyan helyzetekre készültek, amikor a szereplő nézetek száma nem nagy, legalábbis összehasonlítható az adatbázis relációk számával. Ezzel ellentétben az adategyesítés környezetben nagyszámú nézet kezelésére kell felkészülnünk, mivel minden egyes adatforrás egy-egy újabb nézetet jelent. Ezenkívül a nézetek bonyolult predikátumokat tartalmazhatnak, hiszen a céljuk, hogy az egyes adatforrások közti finom különbségeket leírják. További különbség, hogy az adategyesítési környezetben a nézetekről általában feltesszük, hogy nem teljesek, azaz nem tartalmazzák a definíciójukat kielégítő összes sort, csak azok egy részhalmazát. A továbbiakban ismertetünk néhány, az adategyesítés céljára kifejlesztett eljárást.

Vödör algoritmus

A vödör algoritmus célja, hogy a felhasználó közvetített sémában megfogalmazott lekérdezését átfogalmazza olyan lekérdezésre, amelyik közvetlenül

a rendelkezésre álló adatforrásokra hivatkozik. Feltesszük, hogy konjunktív lekérdezésekről van szó, melyekben lehetnek összehasonlítási atomok. A Q lekérdezés összehasonlítási atomjainak halmazát $C(Q)$-val jelöljük.

Mivel a lehetséges átírások száma exponenciális a lekérdezés méretében, ezért a vödör algoritmus fő gondolata, hogy a lehetséges átírások számát drasztikusan csökkenthetjük, ha a lekérdezés ***részcéljait*** – a benne szereplő relációs atomokat – egyenként tekintjük és meghatározzuk, mely nézetek használhatók a részcélokhoz külön-külön.

Az eljárás általános menete a következő. Először minden részcélhoz egy ***vödröt*** rendelünk, amelyik azon nézeteket tartalmazza, ahonnan a részcél sorait vehetjük. A második lépésben az összes olyan összekapcsolást tekintjük, amelyik minden vödörből tartalmaz egy nézetet, és ellenőrizzük, hogy az így kapott V konjunktív lekérdezés átírás szemantikusan helyes-e, azaz $V \sqsubseteq Q$ teljesül-e, vagy szemantikusan helyessé tehető-e összehasonlítási atomok hozzáadásával. Végül a megmaradó terveket minimalizáljuk a redundáns részcélok elhagyásával. Az alábbi VÖDÖR-KÉSZÍTŐ eljárás az első lépést hajtja végre. A bemenet adatforrások leírásának $\mathcal{V}$ halmaza, valamint Q konjunktív lekérdezés,

$$Q\colon Q(\tilde{X}) \leftarrow R_1(\tilde{X}_1), R_2(\tilde{X}_2), \ldots, R_m(\tilde{X}_m), C(Q) \tag{33.69}$$

formában.

VÖDÖR-KÉSZÍTŐ(Q,$\mathcal{V}$)

```
1  for i ← 1 to m
2      Vödör[i] = ∅
3      for minden V ∈ 𝒱 //      V: V(Ỹ)S_1(Ỹ_1), ..., S_n(Ỹ_n), C(V) formájú
4          for j = 1 to n
5              if R_i = S_j
6                  Legyen φ a V változóin következőképpen
                   definiált leképezés:
7                  if Ỹ_j k-adik változója y és y ∈ Ỹ
8                      φ(y) = x_k, ahol x_k az X̃_i k-adik változója
9                      else φ(y) egy új változó, ami nem szerepel
                            sem Q-ban sem V-ben
10                     Q'() = R_1(X̃_1), R_m(X̃_m), C(Q), S_1(φ(Ỹ_1)), ...,
                       S_n(φ(Ỹ_n)), φ(C(V))
11                     if KIELÉGÍTHETŐ^≥(Q')
12                         adjuk φ(V)-t Vödör[i]-hez.
13 return Vödör
```

A KIELÉGÍTHETŐ$^{\geq}$ eljárás a 33.1.2. pontban leírt KIELÉGÍTHETŐ eljárás kiterjesztése arra az esetre, ha egyenlőség atomok mellett egyenlőtlenség atomok is szerepelhetnek a szabály testében. A szükséges változtatás annyi, hogy minden olyan y változóra, amelyik egyenlőtlenség atomban szerepel, ellenőrizni kell, hogy az y-ra kirótt egyenlőtlenségek egyszerre teljesíthetőek-e.

A VÖDÖR-KÉSZÍTŐ eljárás polinomiális lépésszámú Q és $\mathcal{V}$ méretének függvényében. Valóban, a 3. és 4. sorok egymásba ágyazott ciklusának magja $n\sum_{V\in\mathcal{V}}|V|$-szer fut le. Az 5–12. sorok utasításai a 11. sor kivételével konstans sok lépést jelentenek. A 11. sor **if** utasításának feltételét polinomiális időben lehet ellenőrizni.

A VÖDÖR-KÉSZÍTŐ eljárás helyességének igazolásához nézzük meg, hogy milyen feltételekkel teszi be a V nézetet $Vödör_i$-be. Az 5. sorban ellenőrzi, hogy V-ben szerepel-e részcélként az R_i reláció. Ha nem, akkor nyilván V nem adhat használható információt a Q-beli R_i részcélhoz. Ha V-ben szerepel részcélként az R_i reláció, akkor a 8.–9. sorokban elkészíti azt a megfeleltetést, amelyet a változókra alkalmazva az S_j és R_i részcélok megfeleltethetők egymásnak a Q, illetve V fejében levő relációkkal összhangban. Végül a 12. sor ellenőrzi, hogy az így kapott változó megfeleltetésekkel az összehasonlítási atomok nem mondanak-e ellent.

A második lépésben, miután a vödröket a VÖDÖR-KÉSZÍTŐ eljárással elkészítette, a vödör algoritmus ***konjunktív lekérdezés átírások*** halmazát állítja elő. Minden átírás olyan konjunktív lekérdezés, amelyik minden vödörből tartalmaz pontosan egy tényezőt. Az algoritmus eredménye ezen konjunktív lekérdezés átírások egyesítése, hiszen a különböző átírások különböző sorokat adhatnak az eredményhez. Adott Q' konjunktív lekérdezés ***konjunktív lekérdezés átírás,*** ha

1. $Q' \sqsubseteq Q$, vagy

2. Q' kiegészíthető összehasonlítási atomokkal úgy, hogy az előző teljesüljön.

33.10. példa. *Vödör algoritmus.* Tekintsük a következő Q lekérdezést, amelyik azokat az x cikkeket listázza, amely cikkekhez létezik y cikk ugyanabban a témában, hogy x és y kölcsönösen hivatkoznak egymásra. Rendelkezésre áll három nézet V_1, V_2, V_3.

$$\begin{array}{lcl} Q(x) & = & idéz(x,y), idéz(y,x), uaTéma(x,y) \\ V_1(a) & = & idéz(a,b), idéz(b,a) \\ V_2(c,d) & = & uaTéma(c,d) \\ V_3(f,h) & = & idéz(f,g), idéz(g,h), uaTéma(f,g)\ . \end{array} \tag{33.70}$$

Első lépésben a VÖDÖR-KÉSZÍTŐ eljárással az alábbi vödröket állítjuk elő.

$idéz(x,y)$	$idéz(y,x)$	$uaTéma(x,y)$
$V_1(x)$	$V_1(x)$	$V_2(x)$
$V_3(x)$	$V_3(x)$	$V_3(x)$

(33.71)

A második lépésben a vödrök direkt szorzatának minden eleméből szerkeszt az algoritmus egy Q' konjunktív lekérdezést, és ellenőrzi, hogy Q tartalmazza-e Q'-t. Ha igen, akkor hozzáadja a válaszhoz.

Esetünkben megpróbálja V_1-t a többi nézettel összerakni, de így nem kap helyes eredményt. Ennek oka, hogy b nem szerepel V_1 fejében, így a Q-ban szereplő összekapcsolási feltételt – az x és y változók szerepelnek $uaTéma$ relációban is – nem tudja alkalmazni. Ezek után a V_3-t tartalmazó átírásokat tekinti, és észreveszi, hogy a V_3 fejében található változókat egyenlővé téve tartalmazott átírást kap. Végül, az algoritmus azt is megtalálja, hogy V_3-t és V_2-t kombinálva is átírást kap. Egyszerű további ellenőrzéssel kapjuk, hogy ez utóbbi átírás redundáns, V_2-t el lehet hagyni belőle. Tehát a vödör algoritmus eredménye a (33.70) lekérdezésre és nézetekre a (ténylegesen ekvivalens)

$$Q'(x) = V_3(x,x)\ . \tag{33.72}$$

A vödör algoritmus előnye, hogy jelentősen lecsökkenti az ellenőrizendő konjunktív átírás jelöltek számát. Ha az adatforrások alapvetően az összehasonlítási atomokban különböznek egymástól, akkor várhatóan a vödrök mérete kicsi lesz.

A vödör algoritmus fő hátránya éppen abban rejlik, amiben az előnye is. Semmilyen becslésünk nincs arra, hogy a vödrök direkt szorzatának a mérete mekkora lesz: lehet, hogy nagy. Továbbá az eljárás minden egyes lehetséges átírásra elvégez egy lekérdezés tartalmazás ellenőrzést, ami már akkor is NP-teljes, ha nincsenek összehasonlítási atomok.

Inverz szabályok

A vödör algoritmusnál általánosabban használható eljárás az ***inverz szabályok*** alkalmazása. Tetszőleges, negáció nélküli, de rekurziót megengedő datalog programmal adott lekérdezéshez megtalálja a maximálisan tartalmazott átírást polinomiális időben.

Az első kérdés az, hogy adott $\mathcal{P}$ datalog program és $\mathcal{V}$ konjunktív nézetek halmaza esetén létezik-e olyan $\mathcal{P}$-vel ekvivalens $\mathcal{P}_v$ datalog program, amelynek *edb* relációi a $\mathcal{V}$-beli $v_1, v_2, \ldots, v_n$ relációk. Sajnos azonban ez a kérdés algoritmikusan eldönthetetlen. Meglepő viszont az, hogy el tudjuk készíteni a lehető legjobb, maximálisan tartalmazott átírást. Abban az esetben, ha létezik $\mathcal{P}$-vel ekvivalens $\mathcal{P}_v$ datalog program, akkor az eljárásunk azt fogja előállítani, hiszen a maximálisan tartalmazott átírás tartalmazza $\mathcal{P}_v$-t is. Ez csak látszólagos mond ellent annak az állításnak, hogy az ekvivalens átírás algoritmikusan eldönthetetlen, hiszen az inverz szabályokkal előállított, maximálisan

tartalmazott átírásról nem tudjuk eldönteni, hogy ténylegesen ekvivalens-e.

33.11. példa. *Ekvivalens átírás.* Tekintsük a következő $\mathcal{P}$ datalog programot, ahol *él* és $fekete$ relációk *edb* relációk, egy G gráf éleit, illetve feketére színezett csúcsait tartalmazzák:

$$\begin{array}{lrcl} \mathcal{P}\colon & q(X,Y) & = & \acute{e}l(X,Z), \acute{e}l(Z,Y), fekete(Z) \\ & q(X,Y) & = & \acute{e}l(X,Z), fekete(Z), q(Z,Y)\ . \end{array} \tag{33.73}$$

Könnyen ellenőrizhető, hogy $\mathcal{P}$ a G gráf olyan útjainak (pontosabban sétáinak) végpontjait adja meg, amelyek minden belső pontja fekete. Tegyük fel, hogy csak az alábbi két nézet érhető el.

$$\begin{array}{rcl} v_1(X,Y) & = & \acute{e}l(X,Y), fekete(X) \\ v_2(X,Y) & = & \acute{e}l(X,Y), fekete(Y) \end{array} \tag{33.74}$$

v_1 a fekete kezdőpontú, v_2 a fekete végpontú éleket tárolja. Ekkor a $\mathcal{P}$ datalog programnak létezik ekvivalens $\mathcal{P}_v$ átírása, amelyik csak a v_1 és v_2 nézeteket használja *edb* relációként:

$$\begin{array}{lrcl} \mathcal{P}_v\colon & q(X,Y) & \leftarrow & v_2(X,Z), v_1(Z,Y) \\ & q(X,Y) & = & v_2(X,Z), q(Z,Y) \end{array} \tag{33.75}$$

Azonban, ha csak a v_1, vagy v_2 nézet érhető el, akkor nem lehetséges az ekvivalens átírás, mert csak olyan utakat kaphatunk, amelyeknek a kezdő, illetve végpontja fekete.

Az inverz szabály eljárás leírásához szükségünk lesz a ***datalog program,*** illetve a ***datalog szabály*** általánosítására, a ***Horn-szabályra.*** Ha a 33.11. definícióban szereplő (33.26) szabály u_i szabad soraiban a változók és konstansok mellett még ***függvény szimbólumokat*** is megengedünk, akkor ***Horn-szabályról*** beszélünk. Horn-szabályok halmazát ***logikai programnak*** nevezzük. Ebben az értelemben egy függvény szimbólum mentes logikai program lesz datalog program. A 33.11. definíció *edb*, *idb* fogalma logikai programra ugyanúgy értelmezhető.

Az inverz szabály eljárás két lépésből áll. Először olyan logikai programot készítünk, amelyik tartalmazhat függvény szimbólumokat. Azonban ezek a függvény szimbólumok nem szerepelnek rekurzív szabályokban, így a második lépében a logikai programot datalog programmá lehet alakítani.

33.30. definíció. *A*

$$v(X_1,\ldots,X_m) = v_1(\tilde{Y}_1),\ldots,v_n(\tilde{Y}_n) \tag{33.76}$$

szabállyal meghatározott v nézet v^{-1} ***inverze*** *Horn szabályok következő halmaza. Minden $v_i(\tilde{Y}_i)$ részcélnak megfelel egy szabály, amelyiknek teste a $v(X_1,\ldots,X_m)$ literál. A szabály feje $v_i(\tilde{Z}_i)$, ahol a $\tilde{Z}_i$-t $\tilde{Y}_i$-ból úgy kapjuk,*

hogy a (33.76) szabály fejében szereplő változókat meghagyjuk, ezen kívül minden, a fejben nem szereplő Y változó helyére pedig az $f_Y(X_1, \ldots, X_m)$ függvény szimbólumot írjuk. Különböző változókhoz különböző függvény szimbólumok tartoznak. A $\mathcal{V}$ nézet halmaz $\mathcal{V}^{-1}$ inverze a $\{v^{-1} : v \in \mathcal{V}\}$ halmaz, ahol a különböző nézetek inverzeiben különböző függvény szimbólumok szereplenek.

Az inverz definíció gondolata, hogy ha a v nézetben megjelenik a $(x_1, \ldots x_m)$ sor valamilyen $x_1, \ldots x_m$ konstansokkal, akkor minden, a fejben nem szereplő, y változónak van valamilyen kiértékelése, ami a szabály testét igazzá teszi. Ezt az „ismeretlen" kiértékelést jelöljük az $f_Y(X_1, \ldots, X_m)$ szimbólummal.

33.12. példa. *Nézetek inverze.* Legyen $\mathcal{V}$ az alábbi nézetek halmaza.

$$\begin{array}{lcl} v_1(X,Y) & = & \mathit{él}(X,Z), \mathit{él}(Z,W), \mathit{él}(W,Y) \\ v_2(X)) & = & \mathit{él}(X,Z)\ . \end{array} \tag{33.77}$$

Ekkor $\mathcal{V}^{-1}$ a következő Horn szabályokból áll.

$$\begin{array}{lcl} \mathit{él}(X, f_{1,Z}(X,Y)) & = & v_1(X,Y) \\ \mathit{él}(f_{1,Z}(X,Y), f_{1,W}(X,Y)) & = & v_1(X,Y) \\ \mathit{él}(f_{1,W}(X,Y), Y) & = & v_1(X,Y) \\ \mathit{él}(X, f_{2,Z}(X)) & = & v_2(X)\ . \end{array} \tag{33.78}$$

Ezek után $\mathcal{P}$ datalog programhoz és konjunktív nézetek $\mathcal{V}$ halmazához könnyű elkészíteni azt a logikai programot, amelyből majd azt a datalog programot kapjuk, ami $\mathcal{P}$ $\mathcal{V}$-t használó maximálisan tartalmazott átírása.

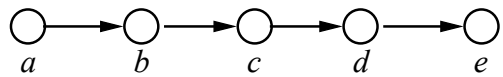
$\mathcal{P}$-ből töröljük az összes olyan szabályt, amelyikben olyan *edb* reláció szerepel, ami nem fordul elő $\mathcal{V}$-beli nézet definíciójában. Az így kapott $\mathcal{P}^-$ programhoz hozzávesszük a $\mathcal{V}^{-1}$ szabályait, és ezáltal nyerjük a $(\mathcal{P}^-, \mathcal{V}^{-1})$ logikai programot. Vegyük észre, hogy $\mathcal{P}$ megmaradt *edb* relációi a $(\mathcal{P}^-, \mathcal{V}^{-1})$ logikai programban *idb* relációk, mivel a $\mathcal{V}^{-1}$ szabályai fejében szerepelnek. Az *idb* relációk elnevezése tetszőleges, így átnevezhetjük őket, hogy ne egyezzen a nevük $\mathcal{P}$ *edb* relációinak nevével. Ezt azonban itt a könnyebb érthetőség kedvéért nem tesszük meg.

33.13. példa. *Logikai program.* Tekintsük az alábbi datalog programot, ami az *él* reláció tranzitív lezártját számítja ki.

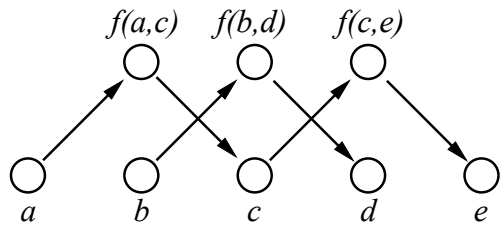
$$\begin{array}{lcl} \mathcal{P}: \quad q(X,Y) & = & \mathit{él}(X,Y) \\ q(X,Y) & = & \mathit{él}(X,Z), q(Z,Y) \end{array} \tag{33.79}$$

Tegyük fel, hogy csak a

$$v(X,Y) = \mathit{él}(X,Z), \mathit{él}(X,Y) \tag{33.80}$$



33.4. ábra. A G gráf.



33.5. ábra. A G' gráf.

materializált nézet érhető el, amelyik a kettő hosszúságú utak végpontjait tárolja. Ha csak ezt a nézetet használhatjuk, akkor a legtöbb, amit remélhetünk, hogy a páros hosszúságú utak végpontjait elő tudjuk állítani. Mivel $\mathcal{P}$ egyetlen *edb* relációja az *él*, ami szerepel v definíciójában, ezért $(\mathcal{P}^-, \mathcal{V}^{-1})$ logikai programot úgy kapjuk, hogy $\mathcal{P}$-hez hozzávesszük $\mathcal{V}^{-1}$ szabályait.

$$
\begin{array}{lll}
(\mathcal{P}^-, \mathcal{V}^{-1}): & q(X,Y) & = \quad \acute{e}l(X,Y) \\
& q(X,Y) & = \quad \acute{e}l(X,Z), q(Z,Y) \\
& \acute{e}l(X, f(X,Y)) & = \quad v(X,Y) \\
& \acute{e}l(f(X,Y),Y) & = \quad v(X,Y)\ .
\end{array}
\tag{33.81}
$$

A $\mathcal{P}$ datalog program *él edb* relációjának példánya legyen a 33.4. ábrán látható G gráf.

Ekkor $(\mathcal{P}^-, \mathcal{V}^{-1})$ három új konstanst vezet be, melyek $f(a,c)$, $f(b,d)$ és $f(c,e)$. A $\mathcal{V}^{-1}$ program *él idb* relációja a 33.5. ábrán látható G' gráfot adja.

$\mathcal{P}^-$ a G' gráf tranzitív lezártját számítja ki. Vegyük észre, hogy azok a párok a tranzitív lezártban, amelyek nem tartalmaznak egyet sem az új konstansokból, pontosan a G-beli páros hosszú utak végpontjai.

A 33.13 példában a $(\mathcal{P}^-, \mathcal{V}^{-1})$ logikai program eredményét például a Naiv-datalog eljárással számíthatjuk ki. Azonban logikai programokra általában nem igaz, hogy az eljárás véget ér. Tekintsük ugyanis a

$$
\begin{array}{lll}
q(X) & = & p(X) \\
q(f(X)) & = & q(X)
\end{array}
\tag{33.82}
$$

logikai programot. Ha a p *edb* reláció az a konstanst tartalmazza, akkor a program eredménye az $a, f(a), f(f(a)), f(f(f(a))), \ldots$ végtelen sorozat lesz. Ezzel ellentétben az inverz szabály eljárás által adott $(\mathcal{P}^-, \mathcal{V}^{-1})$ logikai program eredménye ***garantáltan*** véges, és a kiszámítási algoritmus véges időben

véget ér.

33.31. tétel. *Tetszőleges $\mathcal{P}$ datalog programra, konjunktív nézetek $\mathcal{V}$ halmazára és a nézetek tetszőleges véges példányaira teljesül, hogy a $(\mathcal{P}^-, \mathcal{V}^{-1})$ logikai programnak egyértelmű minimális fixpontja van, valamint a* Naiv-datalog *és* Félig-naiv-datalog *eljárások ezt a fixpontot adják eredményül.*

A 33.31. tétel bizonyításának lényege, hogy függvény szimbólumokat csak az inverz szabályok vezetnek be, amelyek azonban nem rekurzívak, így egymásba ágyazott függvény szimbólumokat tartalmazó tényezők nem keletkeznek. A bizonyítás részletezését az Olvasóra bízzuk (33.3-3 gyakorlat).

Még ha egy adatbázis *edb* relációiból indulunk is ki, egy logikai program eredményében lehetnek olyan sorok, amelyek függvény szimbólumokat tartalmaznak. Ezért bevezetünk egy szűrőt, ami eltávolítja a szükségtelen sorokat. Ha a $\mathcal{P}$ logikai program *edb* relációinak példánya a D adatbázis, akkor $\mathcal{P}(D)\downarrow$ jelöli azon $\mathcal{P}(D)$-beli sorok halmazát, amelyek nem tartalmaznak függvény szimbólumokat. Jelölje $\mathcal{P}\downarrow$ azt a programot, amelyik adott D példányra $\mathcal{P}(D)\downarrow$-t számítja ki. Az alábbi tétel bizonyítása meghaladja jelen fejezet kereteit.

33.32. tétel. *Tetszőleges $\mathcal{P}$ datalog programra, valamint konjunktív nézetek $\mathcal{V}$ halmazára teljesül, hogy $(\mathcal{P}^-, \mathcal{V}^{-1})\downarrow$ logikai program $\mathcal{P}$ $\mathcal{V}$-t használó maximálisan tartalmazott átírása. Továbbá $(\mathcal{P}^-, \mathcal{V}^{-1})$ előállítható $\mathcal{P}$ és $\mathcal{V}$ méretében polinomiális időben.*

A 33.32. tétel jelentése, hogy az az egyszerű eljárás, hogy a nézet definíciók inverzeit hozzáadjuk a datalog programhoz, olyan logikai programot eredményez, ami a lehető legjobban használja fel a nézeteket. Az, hogy $(\mathcal{P}^-, \mathcal{V}^{-1})$ előállítható $\mathcal{P}$ és $\mathcal{V}$ méretében polinomiális időben, könnyen látható, hiszen minden $v_i \in \mathcal{V}$ minden részcéljához egyetlen inverz szabályt kell elkészíteni.

Az átírási feladat teljes megoldásához szükséges azonban egy olyan datalog programot előállítani, amelyik ekvivalens a $(\mathcal{P}^-, \mathcal{V}^{-1})\downarrow$ logikai programmal. Ehhez adja a kulcsot az az észrevétel, hogy $(\mathcal{P}^-, \mathcal{V}^{-1})$-ben csak véges sok függvény szimbólum van, továbbá az alulról felfelé történő kiszámításban, mint a Naiv-datalog eljárás és változatai, egymásba ágyazott függvény szimbólumok nem jönnek létre. Megfelelő könyveléssel nyomon követhetjük a függvény szimbólumok megjelenését, anélkül, hogy ténylegesen előállítanánk azokat tartalmazó sorokat.

Az átalakítást alulról felfelé végezzük, a Naiv-datalog eljáráshoz hasonlóan. $\mathcal{V}^{-1}$-beli *idb* relációban megjelenő $f(X_1, \ldots, X_k)$ függvény szimbólumot a $X_1, \ldots, X_k$ változó listával helyettesítjük. Ugyanakkor az *idb* reláció nevet

meg kell jelölni, hogy tudjuk, az $X_1, \ldots, X_k$ lista a $f(X_1, \ldots, X_k)$ függvényhez tartozott. Ezzel új, „ideiglenes"' reláció neveket vezetünk be. Tekintsük a 33.13 példa (33.81) logikai programjában szereplő

$$él(X, f(X, Y)) = v(X, Y) \tag{33.83}$$

szabályt az

$$él^{\langle 1, f(2,3)\rangle}(X, X, Y) = v(X, Y) \tag{33.84}$$

szabállyal helyettesítjük. A $\langle 1, f(2, 3)\rangle$ jelölés értelmezése, hogy $él^{\langle 1, f(2,3)\rangle}$ első argumentuma megegyezik *él* első argumentumával, a második és harmadik argumentum $él^{\langle 1, f(2,3)\rangle}$-ben az f függvény szimbólummal együtt adja az *él* második argumentumát. Ha $(\mathcal{P}^-, \mathcal{V}^{-1})$ alulról felfelé kiszámítása során $\mathcal{P}^-$ valamelyik *idb* relációjának argumentumába függvény szimbólum kerülne, akkor egy új szabályt adunk a programhoz, a megfelelően megjelölt reláció nevekkel.

33.14. példa. *Logikai program átalakítása datalog programmá.* A 33.13. példa logikai programját az alábbi datalog programmá alakítja át a fent vázolt eljárás. A NAIV-DATALOG alulról felfelé végrehajtásának különböző fázisait vonalak határolják.

$$\begin{array}{lcl}
él^{\langle 1, f(2,3)\rangle}(X, X, Y) & \leftarrow & v(X, Y) \\
él^{\langle f(1,2), 3\rangle}(X, Y, Y) & \leftarrow & v(X, Y) \\
\hline
q^{\langle 1, f(2,3)\rangle}(X, Y_1, Y_2) & = & él^{\langle 1, f(2,3)\rangle}(X, Y_1, Y_2) \\
q^{\langle f(1,2), 3\rangle}(X_1, X_2, Y) & = & él^{\langle f(1,2), 3\rangle}(X_1, X_2, Y) \\
\hline
q(X, Y) & = & él^{\langle 1, f(2,3)\rangle}(X, Z_1, Z_2), q^{\langle f(1,2), 3\rangle}(Z_1, Z_2, Y) \\
q^{\langle f(1,2), f(3,4)\rangle}(X_1, X_2, Y_1, Y_2) & = & él^{\langle f(1,2), 3\rangle}(X_1, X_2, Z), q^{\langle 1, f(2,3)\rangle}(Z, Y_1, Y_2) \\
\hline
q^{\langle f(1,2), 3\rangle}(X_1, X_2, Y) & = & él^{\langle f(1,2), 3\rangle}(X_1, X_2, Z), q(Z, Y) \\
q^{\langle 1, f(2,3)\rangle}(X, Y_1, Y_2) & = & él^{\langle 1, f(2,3)\rangle}(X, Z_1, Z_2), q^{\langle f(1,2), f(3,4)\rangle}(Z_1, Z_2, Y_1, Y_2)
\end{array} \tag{33.85}$$

Az így kapott datalog program egyértelműen mutatja, hogy melyik argumentumokban keletkezhet függvény szimbólum az eredeti logikai programban. Azonban, bizonyos függvény szimbólumokat tartalmazó sorok sohasem eredményeznek függvény szimbólumokat nem tartalmazó sorokat a program eredményének kiszámítása során.

A p relációt ***fontosnak*** nevezzük, ha a 33.16. definícióban megadott előzmény gráfban[3] p-ből van irányított út a program q eredmény relációjába. Ha p nem fontos, akkor a program eredményének kiszámításához nincs szükség p-beli sorokra, így p elhagyható a programból.

[3] Itt az előzmény gráf definícióját ki kell terjesztenünk a datalog program *edb* relációira is.

33.15. példa. *Nem fontos relációk elhagyása.* A 33.14 példában kapott datalog program előzmény gráfjában a $q^{\langle 1,f(2,3)\rangle}$ és $q^{\langle f(1,2),f(3,4)\rangle}$ relációkból nincs irányított út a program q eredmény relációjába, ezért nem fontosak, azaz el lehet hagyni őket és a hozzájuk tartozó szabályokat. Az alábbi datalog programot kapjuk ezután:

$$\begin{array}{lcl} \acute{e}l^{\langle 1,f(2,3)\rangle}(X,X,Y) & \leftarrow & v(X,Y) \\ \acute{e}l^{\langle f(1,2),3\rangle}(X,Y,Y) & \leftarrow & v(X,Y) \\ q^{\langle f(1,2),3\rangle}(X_1,X_2,Y) & \leftarrow & \acute{e}l^{\langle f(1,2),3\rangle}(X_1,X_2,Y) \\ q^{\langle f(1,2),3\rangle}(X_1,X_2,Y) & \leftarrow & \acute{e}l^{\langle f(1,2),3\rangle}(X_1,X_2,Z), q(Z,Y) \\ q(X,Y) & \leftarrow & \acute{e}l^{\langle 1,f(2,3)\rangle}(X,Z_1,Z_2), q^{\langle f(1,2),3\rangle}(Z_1,Z_2,Y)\ . \end{array} \tag{33.86}$$

Még egy egyszerűsítő lépést hajthatunk végre, ami ugyan nem csökkenti a szükséges levezetések számát az eredmény kiszámítása során, viszont elkerül felesleges adat másolásokat. Ha p olyan reláció egy datalog programban, amelyet egyetlen szabály definiál, és annak az egyetlen szabálynak a testében csak egyetlen reláció áll, akkor p elhagyható a programból: Minden olyan szabályban, amelyiknek a testében p előfordul, p-t helyettesíthetjük a p-t definiáló szabály testében szereplő relációval, a változók megfelelő egyenlővé tétele után.

33.16. példa. *Adatmásolás elkerülése.* A 33.14 példában $\acute{e}l^{\langle 1,f(2,3)\rangle}$ és $\acute{e}l^{\langle f(1,2),3\rangle}$ relációkat egyetlen szabály határozza meg, amely szabályok testében egyetlen reláció szerepel. Ezért a (33.86) program tovább egyszerűsíthető.

$$\begin{array}{lcl} q^{\langle f(1,2),3\rangle}(X,Y,Y) & \leftarrow & v(X,Y) \\ q^{\langle f(1,2),3\rangle}(X,Z,Y) & \leftarrow & v(X,Z), q(Z,Y) \\ q(X,Y) & \leftarrow & v(X,Z), q^{\langle f(1,2),3\rangle}(X,Z,Y)\ . \end{array} \tag{33.87}$$

A fenti két egyszerűsítés eredményeként kapott datalog programot $(\mathcal{P}^-,\mathcal{V}^{-1})^{datalog}$-gal jelöljük. Világos, hogy $(\mathcal{P}^-,\mathcal{V}^{-1})$ és $(\mathcal{P}^-,\mathcal{V}^{-1})^{datalog}$ alulról felfelé történő kiszámítása közt kölcsönösen egyértelmű megfeleltetés létezik. Mivel a függvény szimbólumokat $(\mathcal{P}^-,\mathcal{V}^{-1})^{datalog}$-ban nyomon követjük, ezért tudjuk, hogy az eredményül kapott példány megegyezik $(\mathcal{P}^-,\mathcal{V}^{-1})$ eredményének függvény szimbólumot nem tartalmazó sorainak részhalmazával.

33.33. tétel. *Tetszőleges $\mathcal{P}$ datalog programra és konjunktív nézetek $\mathcal{V}$ halmazára, a $(\mathcal{P}^-,\mathcal{V}^{-1})\downarrow$ program ekvivalens a $(\mathcal{P}^-,\mathcal{V}^{-1})^{datalog}$ programmal.*

MiniCon

A vödör algoritmus hátránya alapvetően az, hogy a nézetek részcéljai közti kölcsönhatások nagy részét nem figyeli meg azáltal, hogy minden egyes

részcélt elkülönítve vizsgál. Így a vödrök sok használhatatlan nézetet tartalmazhatnak, és az algoritmus második fázisa nagyon költségessé válhat.

Az inverz szabály eljárás előnye a fogalmi egyszerűsége és modularitása. A nézetek inverzeit csak egyszer kell kiszámítani, utána már tetszőleges datalog programmal adott lekérdezéshez használhatóak. Azonban, az inverzek használatával elveszíthetjük azt az előnyt, hogy a nézet már kiszámított néhány szükséges összekapcsolást. A lekérdezés megválaszolása során ugyanis az inverz szabály eljárás lényegileg újra előállítja az adatbázis relációkat.

A MiniCon algoritmus az előző kettő hátrányait próbálja kiküszöbölni. A kulcs gondolata, hogy ahelyett, hogy a lekérdezés ***részcéljaihoz*** keresnénk átírásokat, azt vizsgáljuk, hogy a lekérdezés ***változói*** hogyan működnek együtt a rendelkezésre álló nézetekkel. A továbbiakban visszatérünk a konjunktív lekérdezésekhez, és a könnyebb érthetőség kedvéért csak olyan lekérdezéseket és nézeteket tekintünk, amelyek nem tartalmaznak konstansokat.

A MiniCon eljárás úgy kezdődik, mint a vödör algoritmus, azt vizsgálja, mely nézetek tartalmaznak a lekérdezés valamelyik részcéljának megfelelő részcélt. Azonban, amikor az algoritmus talál egy részleges leképezést a lekérdezés g részcéljáról valamelyik V nézet g_1 részcéljára, nézőpontot vált, és a lekérdezés változóit tekinti. Az algoritmus megvizsgálja a lekérdezés összekapcsolási feltételeit – amelyeket a változók többszörös előfordulásai határoznak meg – és megkeresi további részcélok olyan minimális halmazát, amit még a V részcéljaira kell képezni, feltéve, hogy g-t g_1-re képezzük. Részcéloknak ez a halmaza, valamint a leképezési információ együttese lesz a ***MiniCon leírás*** (MCL). A második fázisban az MCL-eket kapcsolja össze a MiniCon algoritmus. Az MCL-ek előállítása szükségtelenné teszi a vödör algoritmus legköltségesebb részének, az átírások és a lekérdezés közti tartalmazás ellenőrzésének végrehajtását, mert az MCL-ek előállítási szabálya biztosítja, hogy az összekapcsolásuk helyes eredményt ad.

Adott $\tau\colon Var(Q) \longrightarrow Var(V)$ leképezés esetén azt mondjuk, hogy a V nézet g_1 részcélja ***fedi*** a Q lekérdezés g részcélját, ha $\tau(g) = g_1$. $Var(Q)$, illetve $Var(V)$ a lekérdezés, valamint a nézet változóinak halmazát jelöli. Ahhoz, hogy belássuk egy átírásról, hogy csupa, a lekérdezés eredményéhez tartozó sort ad, meg kell adnunk egy homomorfizmust a lekérdezésről az átírásra. Egy MCL ezen homomorfizmus egy részletének tekinthető, így a részletek majd könnyen összekapcsolhatók lesznek.

A Q lekérdezés átírása a nézeteket használó konjunktív lekérdezések egyesítése. Lehet, hogy ezekben az egyes nézetek fejében szereplő változók közül néhány azonosítva lett, mint a 33.10 példában kapott (33.72) ekvivalens átírásban. Tehát célszerű bevezetnünk a ***fej homomorfizmus*** fogalmát. A

h: $Var(V) \longrightarrow Var(V)$ leképezés fej homomorfizmus, ha identitás a V fejében nem szereplő változókon, de azonosíthat fejben szereplő változókat. Minden x V fejében szereplő változóra $h(x)$ is V fejében szerepel, továbbá $h(x) = h(h(x))$. Ezek után megadhatjuk az MCL pontos definícióját.

33.34. definíció. *A Q lekérdezéshez a V nézet feletti **MiniCon leírás (MCL)** a $C = (h_C, V(\tilde{Y})_C, \varphi_C, G_C)$ négyes, ahol*

- *h_C fej homomorfizmus V-n,*
- *$V(\tilde{Y})_C$-t a h_C alkalmazásával kapjuk V-ből, azaz $\tilde{Y} = h_C(\tilde{A})$, ahol $\tilde{A}$ a V fejében szereplő változók halmaza,*
- *φ_C részleges leképezés $Var(Q)$-ról $h_C(Var(V)$-be,*
- *G_C a Q olyan részcéljainak egy halmaza, amelyeket valamelyik $H_C(V)$-beli részcél fed, a φ_C leképezéssel.*

Az alábbi állításon alapszik az MCL-eket előállító eljárás.

33.35. állítás. *Legyen C a V nézet feletti MiniCon leírás a Q lekérdezéshez. C csak akkor használható a Q nem redundáns átírásához, ha*

F1. *minden olyan x változóra, amelyik Q fejében van és φ_C értelmezési tartományában is, $\varphi_C(x)$ a $h_C(V)$ fejében található, valamint*

F2. *ha $\varphi_C(y)$ nem szerepel $h_C(V)$ fejében, akkor Q minden olyan g részcéljára, amelyik tartalmazza y-t, teljesül, hogy*

1.g minden változója szerepel φ_C értelmezési tartományában, és

2.$\varphi_C(g) \in h_C(V)$.

Az **F1.** feltétel lényegileg ugyanaz, mint a vödör algoritmus feltétele arra, mikor kerül bele egy nézet egy vödörbe. **F2.** jelentése, hogy ha az x változó szerepel a lekérdezés valamelyik összekapcsolási feltételében, amelyik feltételt a nézet nem teljesíti, akkor x-nek szerepelnie kell a nézet fejében, hogy az őt tartalmazó összekapcsolási feltétel később alkalmazható legyen valamilyen másik részcéllal az átírás folyamán. A MCL-KÉSZÍTŐ eljárás Q konjunktív lekérdezéshez és konjunktív nézetek $\mathcal{V}$ halmazához megadja a használható MiniCon leírásokat.

MCL-KÉSZÍTŐ($Q, \mathcal{V}$)

```
1  C ← ∅
2  for Q minden g részcéljára
3      do for V ∈ V
4          do for V minden v részcéljára
5              do Legyen h a legáltalánosabb fej homomorfizmus V-n, amelyikre van
                  φ leképezés, hogy φ(g) = h(v).
6                 if φ és h létezik
7                    then Adjuk C-hez azon C MCL-eket, amelyek megadhatók úgy, h
8                         (a) φ_C (h_C) a φ (h) kiterjesztése,
9                         (b) G_C a Q részcéljainak olyan minimális részhalmaza, mel
                              33.35. állítás teljesül, és
10                        (c) φ és h nem terjeszthető ki φ'_C és h'_C-re úgy, hogy (b) te
                              és a (b)-ben meghatározott G'_C-re G'_C ⊊ G_C.
11 return C
```

Tekintsük újra a 33.10 példa (33.70) lekérdezését és nézeteit. Az MCL-KÉSZÍTŐ eljárás először a lekérdezés *idéz*(x, y) részcélját tekinti. Nem állít elő MCL-t a V_1 nézethez, mivel a 33.35. állítás **F2.** feltétele megsérülne. Ugyanis V_1-nek az *uaTéma*(x, y) részcélt is fednie kellene a $\varphi(x) = a$, $\varphi(y) = b$ leképezésnél, hiszen b nincs V_1 fejében.[4] Ugyanezen ok miatt a lekérdezés többi részcéljánál sem készít MCL-t a V_1 nézethez. Valamilyen értelemben a MiniCon eljárás a vödör algoritmus második lépésének bizonyos részeit az MCL-KÉSZÍTŐ eljárásban elvégzi. Az alábbi táblázat mutatja az MCL-KÉSZÍTŐ eredményét.

$V(\tilde{Y})$	h	φ	G
$V_2(c, d)$	$c \to c, d \to d$	$x \to c, y \to d$	3
$V_3(f, f)$	$f \to f, h \to f$	$x \to f, y \to f$	1, 2, 3

(33.88)

Az MCL-KÉSZÍTŐ eljárás minimális olyan G_C részcél halmazt ad meg, ami teljesíti a 33.35 állítás feltételeit. Ez lehetővé teszi, hogy a MiniCon algoritmus második fázisában csak olyan MCL-eket kapcsoljunk össze, amelyek a lekérdezés részcéljainak ***páronként diszjunkt*** részhalmazait fedik.

33.36. állítás. *Adott Q lekérdezésre és nézetek $\mathcal{V}$, valamint MCL-ek $\mathcal{C}$ halmazára csak olyan $C_1, \ldots C_l$ MCL-ek kapcsolhatók össze Q nem redundáns átírásává, amelyekre teljesül, hogy*

F3. *$G_{C_1} \cup \cdots \cup G_{C_l}$ Q összes részcélját tartalmazza, és*

F4. *minden $i \neq j$-re $G_{C_i} \cap G_{C_j} = \emptyset$.*

[4] $\varphi(x) = b$, $\varphi(y) = a$ eset hasonló.

Az, hogy csak páronként diszjunkt MCL-eket érdemes összekapcsolni, jelentősen lecsökkenti a keresési teret. A MCL-ÖSSZEKAPCSOLÓ eljáráshoz még egy jelölést be kell vezetnünk. A C MCL φ_C leképezése Q változóinak egy egész halmazát képezheti $h_C(V)$ ugyanazon változójára. E halmaz egy tetszőlegesen választott reprezentánsát kiválasztjuk, arra ügyelve, hogy ha a halmazban van Q fejében található változó, akkor egy olyat. $EC_{\varphi_C}(x)$ jelöli annak a halmaznak a reprezentáns változóját, amelyikbe x tartozik. A C MiniCon leírást EC_{φ_C}-vel kiegészítve a $(h_C, V(\tilde{Y}), \varphi_C, G_C, EC_{\varphi_C})$ ötösként kezeljük. Ha a $C_1, \ldots, C_k$ MCL-eket akarjuk összekapcsolni, és valamilyen $i \neq j$-re $EC_{\varphi_{C_i}}(x) = EC_{\varphi_{C_i}}(y)$ és $EC_{\varphi_{C_j}}(y) = EC_{\varphi_{C_j}}(z)$ teljesül, akkor az összekapcsolással kapott konjunktív átírásban x, y és z is ugyanarra a változóra lesz leképezve. Jelölje S_C azt az ekvivalencia relációt Q változóin, amelynek osztályai a φ_C által ugyanarra a változóra képezett elemek, azaz $xS_Cy \iff EC_{\varphi_C}(x) = EC_{\varphi_C}(y)$. $\mathcal{C}$ az MCL-KÉSZÍTŐ eljárás eredményeként kapott MCL-ek halmaza.

MCL-ÖSSZEKAPCSOLÓ($\mathcal{C}$)

Válasz$\leftarrow \emptyset$
for $\{C_1, \ldots, C_n\} \subseteq \mathcal{C}$ amelyre $G_{C_1}, \ldots, G_{C_n}$ a Q részcéljainak partíciója
  do Definiáljuk a Ψ_i leképezést az $\tilde{Y}_i$-n a következőképpen:
    if Q-ban van x változó melyre $\varphi_i(x) = y$
      then $\Psi_i(y) = x$
      else $\Psi_i(y)$ az y egy új példánya
  legyen S az $S_{C_1} \cup \cdots \cup S_{C_n}$ tranzitív lezártja
                                  $\triangleright$ S ekvivalencia reláció Q változóin.
  az S minden osztályához jelöljünk ki egy reprezentánst.
  definiáljuk az EC leképezést a következőképpen:
  if $x \in Var(Q)$
    then $EC(x)$ az S x-t tartalmazó osztályának reprezentánsa
    else $EC(x) = x$
  legyen Q' a $Q'(EC(\tilde{X})) \leftarrow V_{C_1}(EC(\Psi_1(\tilde{Y}_1))), \ldots, V_{C_n}(EC(\Psi_n(\tilde{Y}_n)))$
  Válasz$\leftarrow$ *Válasz*$\cup \{Q'\}$
return *Válasz*

Igaz az alábbi tétel.

33.37. tétel. *Adott konjunktív Q lekérdezésre és konjunktív nézetek $\mathcal{V}$ halmazára a* MiniCon *algoritmus által előállított konjunktív lekérdezések egyesítése a Q $\mathcal{V}$-t használó, maximálisan tartalmazott átírása.*

A 33.37. tétel teljes bizonyítása meghaladja e fejezet kereteit. A *33-1* feladatban az Olvasóra bízzuk annak belátását, hogy az MCL-összekapcsoló eljárás eredményeként kapott konjunktív lekérdezések egyesítését Q tartalmazza.

Megjegyezzük, hogy a Vödör algoritmus, az Inverz-szabályok és a MiniCon algoritmus futási ideje legrosszabb esetben megegyező: $O(nmM^n)$, ahol n a lekérdezés részcéljainak száma, m a nézetek részcéljainak maximális száma és M a nézetek száma. Azonban gyakorlati futási eredmények azt mutatják, hogy nagyszámú nézet esetén (3–400 nézet) a MiniCon algoritmus lényegesen gyorsabb, mint a másik kettő.

Gyakorlatok

33.3-1. A 33.24. állítás és a 33.20.tétel felhasználásával bizonyítsuk be a 33.25.tételt.

33.3-2. Bizonyítsuk be a 33.26. lemma két állítását. *Útmutatás.* Az első állításhoz Q'-ben a $v_i(\tilde{Y}_i)$ nézetek helyére írjuk be a definíciójukat. Az így kapott Q'' lekérdezést minimalizáljuk a 33.19. tétel segítségével. A második bizonyítandó állításhoz használjuk a 33.24. állítást, amellyel bizonyítsuk be, hogy létezik h_i homomorfizmus a $v_i(\tilde{Y}_i)$ nézetet definiáló konjunktív leképezés testéből Q testébe. Lássuk be, hogy a $\tilde{Y}'_i = h_i(\tilde{Y}_i)$ választás megfelelő.

33.3-3. Bizonyítsuk be a 33.31. tételt, felhasználva, hogy a datalog programok minimális fixpontja egyértelmű.

Feladatok

33-1 MiniCon helyes

Bizonyítsuk be, hogy a MiniCon algoritmus helyes eredményt ad. *Útmutatás.* Elegendő belátni, hogy bármelyik, az MCL-összekapcsoló eljárás 14. sorában megadott Q' konjunktív lekérdezésre igaz, hogy $Q' \sqsubseteq Q$. Ez utóbbihoz készítsünk homomorfizmust Q-ról Q'-re.

33-2 $(\mathcal{P}^-, \mathcal{V}^{-1})\downarrow$ ***helyes***

Bizonyítsuk be, hogy a $(\mathcal{P}^-, \mathcal{V}^{-1})\downarrow$ logikai program eredményének minden sora benne van $\mathcal{P}$ eredményében. (A 33.32. tétel bizonyításának része.) *Útmutatás.* Legyen t olyan sor $(\mathcal{P}^-, \mathcal{V}^{-1})$ eredményében, amelyik nem tartalmaz függvény szimbólumot. Tekintsük t levezetési fáját. Ennek levelei nézet literálok, hiszen azok a $(\mathcal{P}^-, \mathcal{V}^{-1})$ program extenzionális relációi. Ha ezeket a leveleket elhagyjuk a levezetési fából, a maradék fa levelei már $\mathcal{P}$ *edb* relációi. Mutassuk meg, hogy az így kapott fa t levezetési fája a $\mathcal{P}$ datalog programban.

33-3 Datalog nézet

Ezzel a feladattal azt szeretnénk megindokolni, miért csak konjunktív nézet definíciókat tekintettünk. Legyen $\mathcal{V}$ nézetek halmaza, Q pedig lekérdezés. A nézetek adott $\mathcal{I}$ példánya esetén a t sor a Q lekérdezés ***biztos válasza,*** ha tetszőleges olyan $\mathcal{D}$ adatbázis példányra, amelyikre teljesül, hogy $\mathcal{I} \subseteq \mathcal{V}(\mathcal{D})$, $t \in Q(\mathcal{D})$ fenn áll.

a. Bizonyítsuk be, hogy ha a $\mathcal{V}$-beli nézetek datalog programmal vannak meghatározva, a Q lekérdezés konjunktív, és nem-egyenlőségi ($\neq$) atomokat tartalmazhat, az a kérdés, hogy a nézetek adott $\mathcal{I}$ példánya esetén egy t sor a Q lekérdezés biztos válasza-e, algoritmikusan eldönthetetlen. *Útmutatás.* Vezessük vissza rá a ***Post Megfeleltetési Problémát,*** ami a következő: Adott az $\{a, b\}$ ábécé feletti szavak két, $\{w_1, \ldots, w_n\}$ és $\{w'_1, \ldots, w'_n\}$ halmaza. A kérdés az, hogy létezik-e olyan $i_1, \ldots, i_k$ (ismétlések megengedettek) index sorozat, hogy

$$w_{i_1} w_{i_2} \cdots w_{i_k} = w'_{i_1} w'_{i_2} \cdots w'_{i_k} \ . \tag{33.89}$$

A Post Megfeleltetési Problémáról ismeretes, hogy algoritmikusan eldönthetetlen. Legyen a V nézet az alábbi datalog programmal adva:

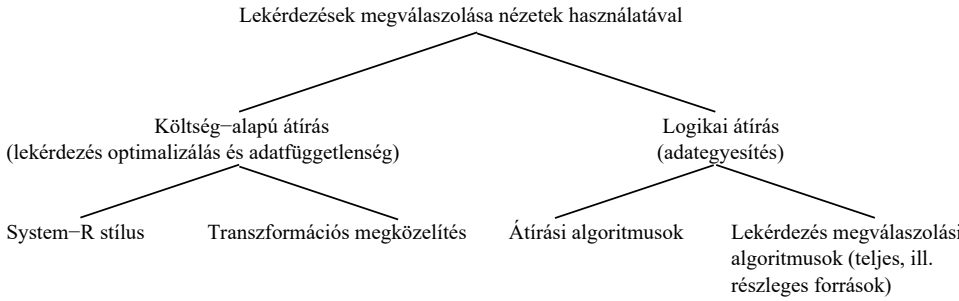
$$\begin{array}{lcl} V(0,0) & \leftarrow & S(e,e,e) \\ V(X,Y) & \leftarrow & V(X_0,Y_0), S(X_0,X_1,\alpha_1), \ldots, S(X_{g-1},Y,\alpha_g), \\ & & S(Y_0,Y_1,\beta_1), \ldots, S(Y_{h-1},Y,\beta_h) \\ & & \text{ahol } w_i = \alpha_1 \ldots \alpha_g \text{ és } \ w'_i = \beta_1 \ldots \beta_h \\ & & \text{egy szabály minden } i \in \{1,2,\ldots,k\}\text{-ra} \\ S(X,Y,Z) & \leftarrow & P(X,X,Y), P(X,Y,Z) \ . \end{array} \tag{33.90}$$

Legyen továbbá Q a következő konjunktív lekérdezés.

$$Q(c) \leftarrow P(X,Y,Z), P(X,Y,Z'), Z \neq Z' \ . \tag{33.91}$$

Lássuk be, hogy a V nézet $\mathcal{I}$ példánya esetén, melyre $\mathcal{I}(V) = \{\langle e, e\rangle\}$ és $\mathcal{I}(S) = \{\}$, a $\langle\rangle$ sor pontosan akkor lesz Q biztos válasza, ha a $\{w_1, \ldots, w_n\}$ és $\{w'_1, \ldots, w'_n\}$ halmazokra a Post Megfeleltetési Problémának *nincs* megoldása.

b. Az *a.* pontban leírt kiszámíthatatlansági eredménnyel ellentétben, ha $\mathcal{V}$ konjunktív nézetek halmaza, és a Q lekérdezés a $\mathcal{P}$ datalog programmal adott, akkor tetszőleges t sorról könnyen eldönthető, hogy a Q lekérdezés biztos válasza-e adott $\mathcal{I}$ nézet példány esetén. Bizonyítsuk be, hogy a $(\mathcal{P}^-, \mathcal{V}^{-1})^{datalog}$ datalog program pontosan a Q biztos válaszában található sorokat adja eredményül.



33.6. ábra. Lekérdezés átírás használatának osztályozása.

Megjegyzések a fejezethez

A „lekérdezések megválaszolása nézetek használatával" feladat kezelését több szempontból lehet osztályozni. A 33.6. ábrán az osztályozás vázlata látható.

A legfontosabb választóvonal a különböző munkák közt, hogy a céljuk adategyesítés, vagy pedig lekérdezés optimalizálás és a fizikai adatfüggetlenség elérése. A két megközelítés közti különbség kulcsa a lekérdezéseket megválaszoló nézeteket használó algoritmus kimenete. Az első esetben, adott Q lekérdezéshez és nézetek $\mathcal{V}$ halmazához az eljárás célja olyan Q' lekérdezés előállítása, amelyik a nézetekre hivatkozik, és ekvivalens Q-val, vagy Q tartalmazza Q'-t. A második esetben az eljárásnak tovább kell lépnie, és egy (remélhetőleg) optimális végrehajtási tervet is elő kell állítania a Q megválaszolására a nézetek (és esetleg adatbázis relációk) használatával. Ebben az esetben csak ekvivalens átírásokat tekinthetünk.

A hasonlóság a két megközelítés között az, hogy mindkettő alapkérdése, hogy egy átírás vajon ekvivalens-e, vagy tartalmazza-e a lekérdezés. Azonban, amíg logikai helyesség elegendő az adategyesítés nézőpontból, addig lekérdezés optimalizálási szempontból nem, ott a *legolcsóbb,* a nézeteket használó végrehajtási tervet kell megtalálni. A nehézségek abból adódnak, hogy az optimalizáló algoritmusok olyan nézeteket is figyelembe kell vegyenek, amelyek ugyan nem járulnak hozzá az átírás logikai helyességéhez, de csökkentik a végrehajtási terv költségét. Ezért az adategyesítési algoritmusok helyességének indoklása főként logikai, míg az optimizálóké logikai és költség-alapú is. Másfelől, adategyesítési problémakörben alapvető adottság a nézetek nagy száma, amelyek a különböző adatforrásoknak felelnek meg. Ezzel ellentétben, az optimalizálási feladatnál általában (de nem mindig!) feltesszük, hogy a nézetek száma a séma nagyságával összehasonlítható.

A lekérdezés optimalizálás téma munkái tovább oszthatók ***System-R stílusú,*** valamint transzformációs optimizálókra. Az előbbiekhez tartoznak

Chaudhuri, Krishnamurty, Potomianos és Shim [71]; Tsatalos, Solomon, és Ioannidis [347] munkái. Az utóbbihoz Florescu, Raschid, és Valduriez [127]; Bello és társai [40]; Deutsch, Popa és Tannen [101], Zaharioudakis és társai [378], valamint Goldstein és Larson [149] cikkei.

Az adategyesítési munkák közül átírási algoritmusokkal foglalkoznak Yang és Larson [373]; Levy, Mendelzon, Sagiv és Srivastava [229]; Qian [299]; valamint Lambrecht, Kambhampati és Ganaprakasam [221] cikkei. A vödör algoritmust Levy, Rajaraman és Ordille [228] vezette be. Az inverz szabályok eljárás Duschka és Genesereth [108, 107] munkája. A MiniCon algoritmust Pottinger és Halevy fejlesztették ki [292, 291].

Lekérdezés megválaszolási algoritmusokkal, illetve feladat bonyolultságával foglalkozik Abiteboul és Duschka [2]; Grahne és Mendelzon [156]; valamint Calvanese, De Giacomo, Lenzerini és Vardi [68].

A STORED rendszert Deutsch, Fernandez és Suciu dolgozták ki [100]. A szemantikus gyorstárolást Yang, Karlapalem és Li [374] tárgyalja. Az átírási feladat különböző kiterjesztéseivel [63, 128, 134, 219, 374] foglalkoznak.

A témakör összefoglalása található Abiteboul [1], Florescu, Levy és Mendelzon [126], Halevy [166, 167], valamint Ullman [351] munkáiban.

A fejezet témakörében magyar nyelven a datalog alapjai olvashatók Ullman és Widom [353] könyvében. Az NP-teljesség és algoritmikus eldönthetőség kérdéseit Ivanyos Gábor, Rónyai Lajos és Szabó Réka tankönyve tárgyalja [308]. Logikai programozásról magyar nyelven Peter Flach könyvében olvashatunk [125].

A szerzők munkáját részben támogatták a T034702, T037846T és a T042706 számú OTKA-szerződések.

34. 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 34.1. alfejezetben a statikus és dinamikus partícionálás, majd a 34.2. alfejezetben a lapozás legismertebb módszereit tekintjük át.

A 34.3. alfejezetben az operációs rendszerek történetének legnevezetesebb anomáliáit – a FIFO lapcserélési algoritmus, az átfedéses memória és a listás ütemező algoritmusok meglepő tulajdonságait – elemezzük.

Végül a 34.4. alfejezetben annak az optimalizálási feladatnak a közelítő algoritmusait tekintjük át, melyben adott méretű fájlokat kell minimális számú mágneslemezen elhelyezni.

34.1. 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 annál, 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 a partíciók dinamikusan keletkeznek, illetve szűnnek meg. Mindkét módszernek vannak mind előnyei, mind hátrányai, és megvalósításuk teljesen különböző algoritmusokat igényel. Ezeket tekintjük át a következőkben.

34.1.1. 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 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      if part[j] = 0
3          BETÖLT-LEGNAGYOBB(hely, helyigény, méret, j, 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       if vár[i] = IGAZ és max < helyigény[i] ≤ méret[p]
5           ind = i
6           max = helyigény[i]
7   if ind > 0
8      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 teljesíti 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 $\mathit{part}[k] = 0$, és amelyre van olyan pozitív $i \leq n$, hogy $\mathit{vár}[i] = \textsc{igaz}$ és $\mathit{helyigény}[i] \leq \mathit{méret}[k]$.

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 $\mathit{vár}[i] = \textsc{hamis}$, 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 $\mathit{part}[k] = 0$, és amelyre van olyan pozitív $i \leq n$, hogy $\mathit{vár}[i] = \textsc{igaz}$ és $\mathit{helyigény}[i] \leq \mathit{méret}[k]$, azaz nem hagytunk szabadon egyetlen olyan partíciót sem, amelyben lenne folyamat, ami belefér.

A Legnagyobb-befér 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 $\mathit{part}[j] = 0$. 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 $\mathit{part}[j] > 0$, 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 valamilyen 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.

34.1. 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      if part[j] == 0
3          BETÖLT-LEGNAGYOBB-VAGY-RÉGÓTA-VÁRAKOZÓ(hely, helyigény,
           küszöb, méret, j, 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. Ezenkívül 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: $(q + \mathit{küszöb})p$ 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       if vár[i] = IGAZ és helyigény[i] ≤ méret[p]
5          if (pont[i] > küszöb és pont[i] > pont[ind]) vagy helyigény[i] > max
6             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
```

34.2. példa. Az előző példánkban a 8 kB helyigényű folyamatnak $küszöb + 1 = k$ 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       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             if rend[rang[i]] < helyigény[i]
8             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 futási idő $\Theta((n+m)\lg m)$.

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      if part[i] = 0
3          BETÖLT-RÉGÓTA-VÁRAKOZÓ-VAGY-KISEBBE-NEM(hely, helyigény,
           méret, i, 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.

34.3. 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      if vár[i] és helyigény[i] ≤ méret[p]
5          if (pont[i] > küszöb és pont[i] > pont[ind]) vagy rang[i] > max
6              pont[ind] = pont[ind] + 1
7              ind = i
8              max = rang[i]
9              else  pont[i] = pont[i] + 1
10 part[p] = ind
11 hely[ind] = p
12 vár[ind] = HAMIS
```

34.1.2. 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 című partíció 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      if vár[j] = IGAZ
3          ∗-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*, p, *helyigény*, *köv*, *méret*, *part*, *vár*)

```
1  p = fej[P]
2  while vár[f] = IGAZ és p ≠ NIL
3        part[p] = 0 és méret[p] ≥ helyigény[f]
4        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: 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, utolsó, méret, helyigény, köv, part, vár*)

```
1  if utolsó[P] ≠ NIL
2     p = köv[utolsó[P]]
3     else p = fej[P]
4  while vár[f] = IGAZ és p ≠ utolsó[P]
5           if p = NIL
6           p = fej[P]
7           if part[p] = 0 és méret[p] ≥ helyigény[f]
8              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-ével, 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 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        if part[p] = 0 és helyigény[f] ≤ méret[p] < min
6           ind = p
7           min = méret[p]
8           p = köv[p]
9  if ind ≠ NIL
10    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 algoritmus egyetlen folyamat számára sem fog nagyobb partíciót kettévágni, mint amekkorát a meglévők közül minimálisan szükséges.

Nem mindig jó azonban, ha minden folyamatot a legkisebb olyan 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, *méret*, *fej*, *helyigény*, *köv*)

```
1  min = ∞
2  ind = NIL
3  p = fej[P]
4  while p ≠ NIL
5        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           ind = p
7           min = méret[p]
8           p = köv[p]
9  if ind ≠ NIL
10    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 $ind = \text{NIL}$, 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 $min < \text{KORLÁT}(helyigény[f])$ 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 $min < \text{KORLÁT}(helyigény[f]) \leq méret[p]$, 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 $min < méret[p] < \text{KORLÁT}(helyigény[f])$ 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*, *méret*, *helyigény*, *köv*, *part*, *vár*)

```
1  max = 0
2  ind = NIL
3  p = fej[P]
4  while p ≠ NIL
5        if part[p] = 0 és méret[p] ≥ helyigény[f] és méret[p] > max
6           ind = p
7           max = méret[p]
8           p = köv[p]
7           max = méret[p]
8           p = köv[p]
9  if ind ≠ NIL
10    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 feltételes 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, méret, helyigény, köv, part, vár*)

```
 1  max = 0
 2  ind = NIL
 3  p = fej[P]
 4  while p ≠ NIL
 5        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           ind = p
 7           max = méret[p]
 8           p = köv[p]
 9  if ind ≠ NIL
10     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

34.1-1. 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?

34.1-2. 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?

34.1-3. 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?

34.2. Lapcserélési algoritmusok

Mint már említettük, a mai számítógépek memóriája több szintből áll. 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 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.

34.2.1. 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 $g : M \times Q \times X \to M \times Q \times Y$ *átmenetfüggvényt* használjuk, ahol M a lehetséges memóriaállapotok halmaza.

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

$$g_{\mathrm{FIFO}}(S, q, x) = \begin{cases} (S, q, \epsilon), & \text{ha } x \in S\ , \\ (S \cup \{x\}, q', \epsilon), & \text{ha } x \notin S,\ |S| = k < m\ , \\ (S \setminus \{y_1\} \cup \{x\}, q", y_1), & \text{ha } x \notin S \mathrm{s} |S| = k = m\ , \end{cases} \quad (34.1)$$

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      laptábla[i, 1] = HAMIS
5  *-ELŐKÉSZÍT(laptábla)
6  for i = 1 to p                              // A program futtatása.
7      *-VÉGREHAJT(laptábla, i)
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($laptábla, t$),

```
1  if laptábla[r_t, 1] = IGAZ                  // A következő lap benn van.
2      NIL
3  if laptábla[r_t, 1] = HAMIS                 // A következő lap nincs benn.
4      laphiba = laphiba + 1
5      if foglalt < m                          // A fizikai memória nincs tele.
6          SORBA(Q, r_t)
7          behoz = foglalt
8          foglalt = foglalt + 1
9          if foglalt = m                      // A fizikai memória tele van.
10             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
17 SORBA(Q, r_t)
```

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 be-

olvassa: 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 – ezért 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: $q_0 = \langle\rangle$ és

$$g_{\text{LRU}}(S,q,x) = \begin{cases} (S, q''', \epsilon), & \text{ha } x \in S\ , \\ (S \cup \{x\}, q', \epsilon), & \text{ha } x \notin S,\ |S| = k < m\ , \\ (S \setminus \{y_1\} \cup \{x\}, q", y_1), & \text{ha } x \notin S \text{s} |S| = k = m\ , \end{cases} \tag{34.2}$$

ahol $q = \langle y_1, y_2, \ldots, y_k\rangle$, $q' = \langle y_1, y_2, \ldots, y_k, x\rangle$, $q'' = \langle y_2, y_3, \ldots, y_m, x\rangle$ és ha $x = y_k$, akkor $q''' = \langle y_1, y_2, \ldots, y_{k-1}, \ldots, y_{k+1} \ldots y_m, y_k\rangle$.

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      utolsó-hiv[r_t] = t
3  if laptábla[r_t, 1] = HAMIS                 // A következő lap nincs benn.
```

```
4       laphiba = laphiba + 1
5       if foglalt < m                          // A fizikai memória nincs tele.
6          behoz = foglalt
7          foglalt = foglalt + 1
8          if foglalt = m                       6/ A fizikai memória tele van.
9             kidob = r_{t-1}
10            for i = 0 to n - 1
11               if laptábla[i, 1] = IGAZ és utolsó-hiv[i] < utolsó-hiv[kidob]
12                  kidob = utolsó-hiv[i]
13            laptábla[kidob, 1] = 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     NIL
3  if laptábla[r_t, 1] = HAMIS                 // A következő lap nincs benn.
4     laphiba = laphiba + 1
5     foglalt < m                              // A fizikai memória nincs tele.
6     behoz = foglalt
7     foglalt = foglalt + 1
```

```
8     if foglalt = m                          // A fizikai memória tele van.
9        OPT-KIDOB(t, R)
10       laptábla[kidob, 1] = 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     keret[j] = HAMIS
4  s = t + 1                          // A lapkeretek védettségének meghatározása.
5  while s ≤ p és védve < m − 1 és laptábla[r_s, 1] = IGAZ és keret[laptábla[r_s, 2]] = HAMIS és
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       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*$[0\,..\,n-1]$ 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     gyakoriság[r_t] = gyakoriság[r_t] + 1
 3  if laptábla[r_t, 1] = HAMIS                         // A következő lap nincs benn.
 4     laphiba = laphiba + 1
 5     if foglalt < m                                   // A fizikai memória nincs tele.
 6        behoz = foglalt
 7        foglalt = foglalt + 1
 8        if foglalt = m                                // A fizikai memória tele van.
 9        kidob = r_{t-1}
10        for i = n - 1 downto 0
11            if laptábla[i, 1] = IGAZ és gyakoriság[i] ≤ gyakoriság[kidob]
12               kidob = utolsó-hiv[i]
13               laptábla[kidob, 1] = 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] = behoz
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 *piszkos* 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 és a négy üres sor létrehozását jelenti.

NRU-ELŐKÉSZÍT(n)

```
1  for i = 0 to n − 1
2      hivatkozott[j] = HAMIS
3      piszkos[j] = HAMIS
4  Q_0 = ∅
5  Q_1 = ∅
6  Q_2 = ∅
7  Q_3 = ∅
```

NRU-VÉGREHAJT($m, hivatkozott, piszkos, k, R, W$)

```
 1  if laptábla[r_t, 1] = IGAZ                  // A következő lap benn van.
 2     if W[r_t] = IGAZ
 3        piszkos[r_t] = IGAZ
 4  if laptábla[r_t, 1] = HAMIS                 // A következő lap nincs benn.
 5     laphiba = laphiba + 1
 6     if foglalt < m                           // A fizikai memória nincs tele.
 7        behoz = foglalt
 8        foglalt = foglalt + 1
 9        hivatkozott[r_t] = IGAZ
10        if W[r_t] = IGAZ
11        piszkos[r_t] = IGAZ
12        if foglalt = m                        // A fizikai memória tele van.
13           NRU-KIDOB(t, kidob)
14           laptábla[kidob, 1] = HAMIS
15           behoz = laptábla[kidob, 2]
16           if piszkos[kidob] = IGAZ
17              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     for i = 0 to n − 1
23         if hivatkozott[i] = hamis
24            piszkos[i] = hamis
```

A cserélendő lap kiválasztása azon alapul, hogy a fizikai memóriában lévő lapokat négy sorba (Q_0, Q_1, Q_2, Q_3) soroljuk.

NRU-kidob(*idő*)

```
1   for i = 0 to n − 1                              // Lapok osztályozása.
2       if hivatkozott[i] == hamis
3          / if / piszkos[i] == hamis
4                  Sorba(Q_1,i)
5                  else Sorba(Q_2,i)
6          else if piszkos[i] = hamis
7
    textscSorba(Q_3,i)
8                   else= Sorba(Q_4,i)
9   if Q_1 ≠ ∅                                      // Cserélendő lap kiválasztása.
10     kidob = Sorból(Q_1)
11     else if Q_2 ≠ ∅
12             kidob = Sorból(Q_2)
13             else if Q_3 ≠ ∅
14                     kidob = Sorból(Q_3)
15                 kidobelse Sorból(Q_4)
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 lap *hivatkozott* változójának értéke hamis, akkor a lapot kidobjuk. Ha viszont a *hiv* változójának értéke 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.

34.2.2. 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     WS-KIDOB(t)
 3     KIÍR(laptábla[kidob, 2], kidob)
 4     laptábla[r_t, 1] = IGAZ
 5     laptábla[r_t, 2] = kidob
 6  if t > h                                     // Benn marad-e r_{t-h} a memóriában?
 7     j = h - 1
 8     while r_j ≠ r_{t-h} és j < t
 9           j = j + 1
10           if j > t
11           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, mint 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($laptábla, t, d$)

```
1  számláló = 0                                          // Előkészítés.
2  for i = 1 to n
3      laptábla[i, 1] = HAMIS
4      hivatkozott[i] = HAMIS
5  for j = 1 to p                                        // Futtatás.
6      if laptábla[r_t, 1] == IGAZ
7          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 = 1 to n
12             if hivatkozott[i] == HAMIS
13                 laptábla[i, 1] = HAMIS
14                 hivatkozott[i] = HAMIS
```

Gyakorlatok

34.2-1. Tekintsük a következő hivatkozási sorozatot: $R = \langle 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6\rangle$. Hány laphiba fordul elő, ha a FIFO, LRU és OPT algoritmust alkalmazzuk k ($1 \leq k \leq 8$) lapkeretes fizikai memóriájú gépen?

34.2-2. Valósítsuk meg a FIFO algoritmust úgy, hogy a Q sor helyett egy mutatót kezelünk, amely mindig a fizikai memória következő, lapbetöltésre váró lapkeretére mutat.

34.2-3. 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?

34.2-4. Írjuk meg és elemezzük a MÁSODIK-ESÉLY, az ÓRA és a LIFO algoritmusok pszeudokódját.

34.2-5. Csökkenthető-e az NFU futási idejének nagyságrendje, ha nem minden laphiba után osztályozzuk a lapokat, hanem folyamatosan karbantartjuk a négy sort?

34.2-6. 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.

34.2-7. * 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.

34.3. 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.

34.3.1. 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 $R = \langle r_1, r_2, \ldots, r_p \rangle$ véges hivatkozási sorozatra az

$$E_P(R,m) = \frac{f_P(R,m)}{p} \,, \tag{34.3}$$

$R = \langle r_1, r_2, \ldots \rangle$ végtelen hivatkozási sorozatra pedig a

$$E_P(R,m) = \liminf_{k\to\infty} \frac{f_P(R_k,m)}{k} \tag{34.4}$$

módon definiálunk, ahol $R_k = \langle r_1, r_2, \ldots, r_k \rangle$.

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 $R = \langle 1,2,3,4,1,2,5,1,2,3,4,5 \rangle$ 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.

34.1. 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.

34.2. 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*

$$\frac{f(R,M)}{f(R,m)} > 2 - \epsilon \,. \tag{34.5}$$

Bélády, Nelson és Shedler a következőt sejtették.

34.3. sejtés. *Tetszőleges R hivatkozási sorozatra és $M > m \ge 1$ memóriaméretekre*

$$\frac{f_{\mathrm{FIFO}}(R,M)}{f_{\mathrm{FIFO}}(R,m)} \le 2 \,. \tag{34.6}$$

A sejtést például a következő példával cáfolhatjuk.

Legyen $m = 5$, $M = 6$, $n = 7$, $k \geq 1$ és $R = UV^k$, ahol $V = (1,2,3,4,5,6,7)^3$ és $U = \langle 1,2,3,4,5,6,7,1,2,4,5,6,7,3,1,2,4,5,7,3,6,2,1,4,7,3,6,2,5,7,3,6,2,5\rangle$.

Ha az U végrehajtási sorozatot $m = 5$ lapkeretet tartalmazó fizikai memóriával hajtjuk végre, akkor 29 laphiba következik be és a feldolgozás a $\langle 7,3,6,2,5\rangle$ vezérlő állapotot eredményezi. Ezután a V hivatkozási sorozat minden végrehajtása 7 további laphibát okoz és 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.

34.4. tétel. *Tetszőlegesen nagy L számhoz megadhatók olyan m, M és R paraméterek, melyekre*

$$\frac{f(R,M)}{f(R,m)} > L\ . \tag{34.7}$$

34.3.2. 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.

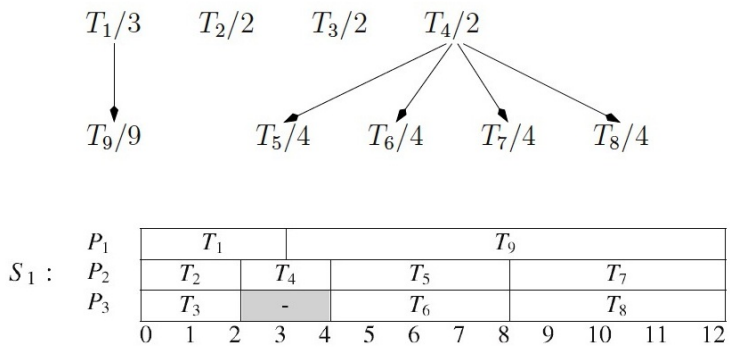
E. G. Coffman jr. 1976-ban leírta, hogy a p processzorszám csökkenése, az egyes programok végrehajtásához szükséges lépések t_i számának csökkenése, a megelőzési korlátozások enyhítése és a lista változtatása külön is anomáliát okozhat.

Legyen a programok végrehajtási időinek vektora $\mathbf{t}$, a megelőzési reláció $<$, a lista L és a programok közös listás végrehajtásához szükséges lépések száma p azonos processzoron $C(p,L,<,\tau)$. Az L' lista, a $<'\subseteq<$ megelőzési reláció, a $\mathbf{t}' \leq \mathbf{t}$ végrehajtási idő vektor vektor és a $p' \geq p$ processzorszám esetén a lépésszám legyen $C'(p',L',<',\tau')$.

Az anomália mértékét ezúttal a relatív futási idővel jellemezzük.

Először négy példát mutatunk az anomália különböző típusaira.

34.4. 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 34.1. ábra) $C_{\max}(S_1) = 12$, amiről könnyen belátható, hogy optimális érték.

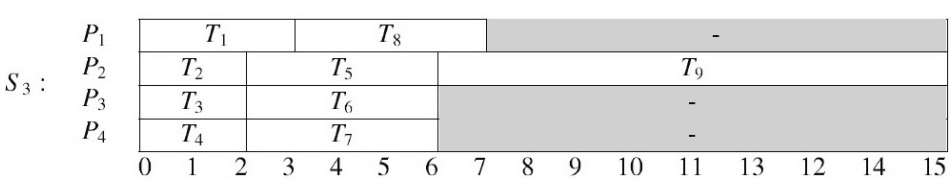


34.1. ábra. A τ_1 taszkrendszer, és annak optimális ütemezése.

34.5. 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 34.2. ábra) $C_{max}(S_2) = 14$.

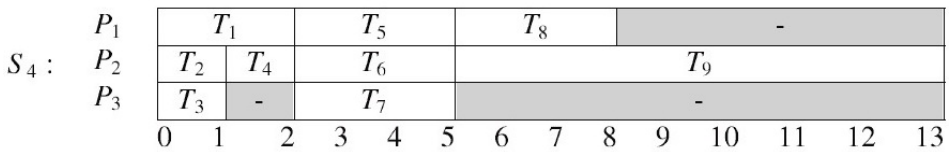
34.2. ábra. A τ_1 taszkrendszer ütemezése az L' lista esetén.

34.6. példa. Ütemezzük a τ_1 taszkrendszert az L listával $m' = 4$ processzorra. Az eredmény $C_{max}(S_3) = 15$ (lásd 34.3. ábra).



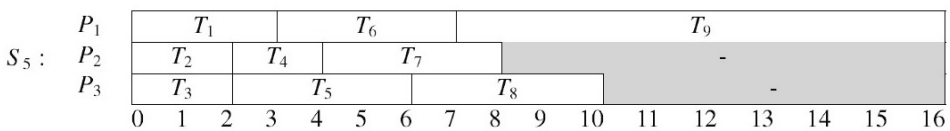
34.3. ábra. τ_1 ütemezése az L listával $m' = 4$ processzorra.

34.7. példa. Csökkentsük τ_1-ben a végrehajtási időket eggyel. Ütemezzük az így kapott τ_2 taszkrendszert az L listával $m = 3$ processzorra. Az eredmény: $C_{max}(S_4) = 13$ (lásd a 34.4. ábrát).



34.4. ábra. τ_2 ütemezése az L listával $m = 3$ processzorra.

34.8. 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 34.5. ábrán látható: $C_{\max}(S_5) = 16$.



34.5. ábra. A τ_3 taszkrendszer ütemezése $m = 3$ processzorra.

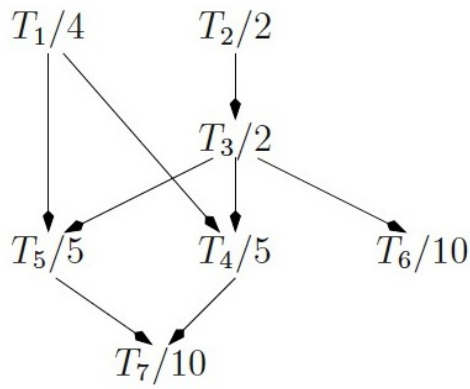
A következő példa azt mutatja, hogy a maximális befejezési idő növekedése nem csak a lista rossz megválasztása miatt következhet be. Legyen a τ taszkrendszer és annak S_{OPT} optimális ütemezése a 34.7. ábra szerinti. Ekkor $C_{max}(S_{\mathrm{OPT}}) = 19$

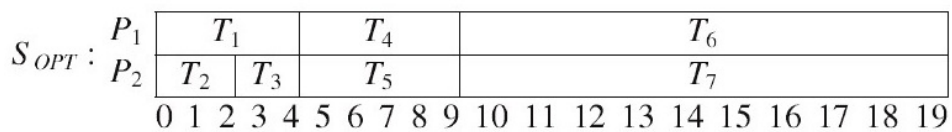
34.9. példa. Legyen a τ taszkrendszer és annak S_{OPT} optimális ütemezése a 34.7. ábra szerinti. Ekkor $C_{max}(S_{\mathrm{OPT}}) = 19$.

34.6. ábra. A τ taszkrendszer, és annak S_{OPT} ütemezése két processzoron.

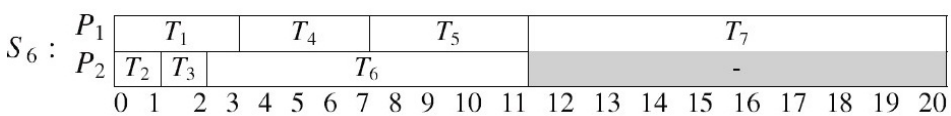
Könnyen belátható, hogy ha most a végrehajtási időket 1-gyel csökkentjük, akkor a kapott τ' taszkrendszeren a $C_{max}(S_6) = 20$ értéknél egyetlen listával sem érhetünk el jobbat (lásd a 34.11. ábrát).

Ezen példák után az ütemezési paraméterek hatását jellemző relatív korlátot adunk meg. Tegyük fel, hogy adott τ és τ' taszkrendszerekre $T' = T$, $<' \subseteq <$, $\mathbf{t}' \leq \mathbf{t}$. 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, illetve S' ütemezésekre legyen $C(S) = C$ és $C(S') = C'$.





34.7. ábra. A τ taszkrendszer, és anak S_{OPT} ütemezése két processzoron.



34.8. ábra. A τ' taszkrendszer optimális listás ütemezése.

34.5. tétel. (ütemezési korlát). *A fenti feltételek mellett*

$$\frac{C'}{C} \leq 1 + \frac{m-1}{m'} \,. \tag{34.8}$$

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: $A = \{t \in [0, C') \mid \text{a } t \text{ időpontban minden processzor foglalt}\}$, $B = [0, C') \setminus A$. 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 $x_1 = \sup\{x \mid x < s_{j_1} \text{ és } x \in B\}$, 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 (b 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

$$T_{j_r} <' T_{j_{r-1}} <' \cdots <' T_{j_1} \ , \tag{34.9}$$

továbbá minden $t \in B$ időpontban van olyan processzor, amelyik dolgozik, és éppen a lánc valamelyik elemét hajtja végre. Ebből következik, hogy

$$\sum_{\phi \in S'} t'(\phi) \le (m' - 1) \sum_{k=1}^{r} t'_{j_k} \ , \tag{34.10}$$

ahol ϕ-vel az üres periódusokat jelöltük, és így a bal oldali összeg az S'-ben lévő összes üres periódusra vonatkozik.

(34.9) és $<' \subseteq <$ alapján $T_{j_r} < T_{j_{r-1}} < \cdots < T_{j_1}$, és így

$$C \ge \sum_{k=1}^{r} t_{j_k} \ge \sum_{k=1}^{r} t'_{j_k} \ . \tag{34.11}$$

Mivel

$$mC \ge \sum_{i=1}^{n} t_i \ge \sum_{i=1}^{n} t'_i \ , \tag{34.12}$$

és

$$C' = \frac{1}{m'} \left(\sum_{i=1}^{n} t'_i + \sum_{\phi \in S'} t'(\phi) \right),$$

így (34.10), (34.11) és (34.12) alapján

$$C' \le \frac{1}{m'} \Big(mC + (m' - 1)C \Big) \ ,$$

S_7 :

T_1	T_{m+1}
T_2	T_{m+2}
⋮	⋮
T_{m-1}	T_{2m-1}
T_m	

34.9. ábra. Az $L = (T_1, \ldots, T_n)$ listához tartozó $S_7(\tau_4)$ ütemezés.

ahonnan $C'/C \leq 1 + (m-1)/m'$. ■

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.

34.10. példa. Ebben a példában a lista változik, $<$ üres, m tetszőleges. A végrehajtási idők a következők:

$$t_i = \begin{cases} 1, & \text{ha } i = 1, \ldots, m-1\ , \\ m, & \text{ha } i = m\ , \\ m-1, & \text{ha } i = m+1, \ldots, 2m-1\ . \end{cases}$$

Ha ezt a τ_4 taszkrendszert m processzorra az $L = (T_1, \ldots, T_{2m-1}n)$ listával ütemezzük, akkor 34.9. 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 34.10. ábrán látható $S_8(\tau_4)$ ütemezést kapjuk.

S_8 :

T_{m+1}				T_m
T_{m+2}				-
⋮				⋮
T_{2m-1}				-
T_1	T_2	...	T_{m-1}	-

34.10. ábra. Az L' listához tartozó $S_8(\tau_4)$ ütemezés.

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 $\leq$ jel jobb oldalán szereplő kifejezés nem csökkenthető.

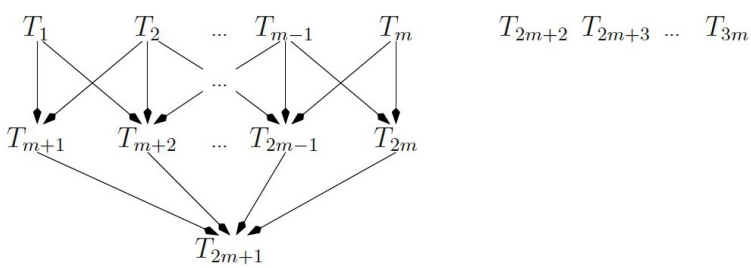
34.11. példa. Ebben a példában a végrehajtási időket csökkentjük. A lista mindkét esetben az $L = L' = \langle T_1, \ldots, T_{3m} \rangle$. Itt és a fejezet további részében ε tetszőleges kis pozitív számot jelöl. Az eredeti végrehajtási időket a $\mathbf{t} = (t_1, \ldots, t_{3n})$ vektor tartalmazza, ahol

$$t_i = \begin{cases} 2\varepsilon, & \text{ha } i = 1, \ldots, m\ , \\ 1, & \text{ha } i = m+1, \ldots, 2m\ , \\ m-1, & \text{ha } i = 2m+1, \ldots, 3m\ . \end{cases}$$

Az új végrehajtási idők

$$t'_i = \begin{cases} t_i - \varepsilon, & \textit{ha } i = 1, \ldots, m-1\ , \\ t_i, & \textit{ha } i = m, \ldots, 3m\ . \end{cases}$$

A τ_5, illetve a módosított τ'_5 taszkrendszer megelőzési gráfját 34.11. ábra mutatja, az $S_9(\tau_5)$ (optimális) ütemezés és az $S_{10}(\tau'_5)$ ütemezés pedig a 34.12. ábrán látható. Itt $C = C_{\max}(S_9(\tau_5)) = m+2\varepsilon$ és $C' = C_{\max}(S_{10}(\tau'_5) = 2m-1+\varepsilon$, 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.



34.11. ábra. τ_5 és τ'_5 azonos gráfja.

Itt $C = C_{\max}(S_9(\tau_5)) = m + 2\varepsilon$ és $C' = C_{\max}(S_{10}(\tau'_5) = 2m - 1 + \varepsilon$, 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.

34.12. példa. Ebben a példában a megelőzési korlátozásokat gyengítjük. A τ_6 taszkrendszer megelőzési gráfját a 34.13. ábra mutatja. A taszkok végrehajtási ideje pedig: $t_1 = \varepsilon$, $t_i = 1$, ha $i = 0, 1, \ldots, m^2 - m + 1$, 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 34.14. ábra tartalmazza. Az összes megelőzési korlátozás elhagyásával kapjuk τ_6-ból a τ'_6 taszkrendszert. A 34.15. ábra mutatja az $S_{12}(\tau'_6)$ ütemezést.

34.13. példa. Ezúttal a processzorok számát növeljük: m-ről m'-re. A τ_7 taszkrend-

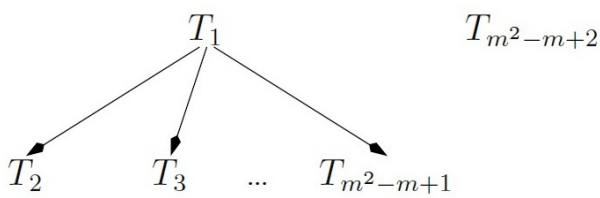
S_9 :

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}	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}	-

34.12. ábra. Az $S_9(\tau_5)$ és $S_{10}(\tau_5')$ ütemezések.



34.13. 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}

34.14. ábra. Az $S_{11}(\tau_6)$ optimális ütemezés.

szer gráfját a 34.16. ábra mutatja, a végrehajtási idők pedig

$$t_i = \begin{cases} \varepsilon, & \text{ha } i = 1, \ldots, m+1, \\ 1, & \text{ha } i = m+2, \ldots, mm' - m' + m + 1, \\ m', & \text{ha } i = mm' - m' + m + 2. \end{cases}$$

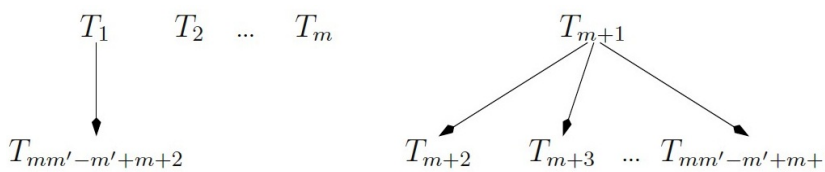
A taszkrendszer optimális ütemezését m, illetve m' processzoron a 34.17., illetve 34.18. ábra mutatja. A maximális befejezési időket összehasonlítva itt is a kívánt

S_{12} :

T_1	T_{m+1}	$\cdots$	T_{m^2-m+1}
T_2	T_{m+2}	$\cdots$	T_{m^2-m+2}
T_3	T_{m+3}	$\cdots$	-
$\vdots$	$\vdots$	$\ddots$	$\vdots$
T_{m-1}	T_{2m+1}	$\cdots$	-
T_m	T_{2m}	$\cdots$	-

34.15. ábra. Az $S_{12}(\tau_6')$ ütemezés.

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 $\lim_{\varepsilon\to 0} C'/C = 1 + (m-1)/m'$.



34.16. ábra. τ_7 rákövetkezési gráfja.

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

34.17. ábra. Az $S_{13}(\tau_7)$ optimális ütemezés ($a = mm' - m' + 3$, $b = a + 1$, $c = mm' - m' + m + 1$).

Ezeknek a példáknak a segítségével beláttuk a következő állítás helyességét.

34.6. tétel. (ütemezési korlát élessége). *A relatív sebességre adott (34.8) korlát az m, t, $<$ és L paraméterek mindegyikének változására nézve (különkülön is) aszimptotikusan éles.*

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	-

34.18. ábra. Az $S_{14}(\tau_7)$ optimális ütemezés ($a = mm' - 2m' + m + 2$, $b = m + 1$, $c = 2m + 2$, $d = mm' - 2m' + 2m + 2$, $e = m + m' + 1$, $f = mm' - m' + m + 1$).

34.3.3. 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 az 1, 2, ..., n számokkal jelöljük. Mindegyik törpe végtelen gombócsorozatot állít elő (a T_i törpe sorozatát G_i, a gombócsorozatok mátrixát G jelöli).

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

$$S_f(G) = \liminf_{t\to\infty} \frac{\sum_{i=1}^{t} h_i(f)}{t} \tag{34.13}$$

határértékkel definiáljuk.

Könnyen belátható, hogy ha $1 \le f \le g$, akkor a gombócevési sebességekre fennáll

$$f \le S_f(G) \le fn, \quad g \le S_g(G) \le gn\ , \tag{34.14}$$

a két óriás relatív gombócevési sebességére pedig

$$\frac{f}{gn} \leq \frac{S_f(G)}{S_g(G)} \leq \frac{fn}{g} \,. \tag{34.15}$$

Most megmutatjuk, hogy a *kis óriás* gombócevési sebessége akárhányszor nagyobb lehet, mint a *nagy óriásé*.

34.7. tétel. *Ha $r \geq 1$, $n \geq 3$, $g > f \geq 1$, akkor léteznek olyan G gombócsorozatok, amelyekre egyrészt*

$$\frac{S_g(G)}{S_f(G)} = \frac{g}{fn} \,, \tag{34.16}$$

másrészt

$$\frac{S_g(G)}{S_f(G)} = \frac{gn}{f} \,. \tag{34.17}$$

Bizonyítás. A természetes korlátokat megadó (34.15) egyenlőtlenségben szereplő alsó korlát élességének belátásához tekintsük az

$$G_1 = 1^f 2^{2f+1} 1^* \tag{34.18}$$

és

$$G_2 = 1^{f+1} (2\ 3\ \ldots\ n)^* \tag{34.19}$$

sorozatokat.

A felső korlát élességét a

$$G_1 = 1\ 2^{2f+1}\ 1^* \tag{34.20}$$

és

$$G_2 = 1^{f-1}\ 3^{2f}\ 1^f\ (2\ 3\ \ldots\ n)^* \tag{34.21}$$

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*.

34.3.4. 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

34.3-1. 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.

34.3-2. Adjunk meg olyan paramétereket, hogy a listás ütemezésnél a processzorszám növeléseker legalább másfélszeresére nőjön a maximális befejezési idő.

34.3-3. Adjunk meg olyan paramétereket, hogy a kis óriás gombócevési sebessége kétszerese legyen a nagy óriásénak.

34.4. 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 tankönyv *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.

34.4.1. 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 $\mathbf{h} = \langle h_1, h_2, \ldots h_n \rangle$ 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$(n, \mathbf{t})$

```
1  for i = 1 to n
2      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 az 1–2. sorokban lévő ciklusban oldjuk meg, akkor a helyigény $\Theta(1)$-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$(n, \mathbf{t})$

```
1  h[1] = t[1]
2  lemezszám = 1
3  for i = 2 to n
4      if h[lemezszám] + t[i] ≤ 1
5          h[lemezszám] = h[lemezszám] + t[i]
6          lemezszám = lemezszám + 1
7          h[lemezszám] = t[i]
8  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 3–7. sorokban lévő ciklusban oldjuk meg, akkor a helyfoglalás $\Theta(1)$-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**)

```
1  lemezszám = 1
2  for i = 1 to n
3      h[i] = 0
3  for i = 1 to n
4      k = 1
5      while t[i] + h[k] > 1
6              k = k + 1
7              h[k] = h[k] + t[i]
8              if k > lemezszám
9                  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**)

```
1  lemezszám = 1
2  for i = 1 to n
3      h[i] = 0
4  for i = 1 to n
5      szabad = 1.0
6      ind = 0
7      for k = 1 to lemezszám
8          if h[k] + t[i] ≤ 1 és 1 − h[k] − t[i] < szabad
9              ind = k
10             szabad = 1 − h[k] − t[i]
11     if ind > 0
12         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**)

```
1  lemezszám = 0
2  bind = 1
3  eind = n
4  while eind ≥ bind
5        if eind − bind ≥ 1
6           if t[bind] + t[eind] > 1
7              lemezszám = lemezszám + 2
8              h[lemezszám − 1] = t[bind]
9              h[lemezszám] = t[eind]
10             else lemezszám = lemezszám + 1
11                  h[lemezszám] = t[bind] + t[eind]
12          if eind = bind
13             lemezszám = lemezszám + 1
14             h[lemezszám] = t[eind]
15          bind = bind + 1
16          eind = eind − 1
17 return lemezszám
```

Ennek az algoritmusnak a helyigénye ($\Theta(n)$), futási ideje $\Theta(n)$. Ha azonban a fájlméretek beolvasását és a lemezszintek kivitelét online módon megoldjuk, a helyigény csak $\Theta(1)$.

A következő öt algoritmus két részből áll: először a végrehajtási idők alapján csökkenő sorrendbe rendezik a taszkokat (ennek a rendezésnek az eredménye a **s** vektor), majd a második részben a rendezett taszkokat ütemezik.

Rendező egyszerű algoritmus (NFD)

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 rendező mohó 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 rendező gazdaságos 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 rendező párosító 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 rendező gyors 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, **s**)

```
1  bind = 1
2  eind = n
4  lemezszám = 0
5  while eind ≥ bind
6        lemezszám = lemezszám + 1
7        h[lemezszám] = s[bind]
8        bind = bind + 1
9        while eind ≥ bind és h[lemezszám] + s[eind] ≤ 1
10             ind = eind
11             while ind > bind és h[lemezszám] + s[ind − 1] ≤ 1
12              ind = ind − 1
13              h[lemezszám] = h[lemezszám] + s[ind]
14              if eind > ind
15              for i = ind to eind − 1
```

```
16                s[i] = s[i + 1]
17                eind = eind − 1
18  return lemezszám
```

Ennek a programnak a helyigénye $\Theta(n)$, futásideje legrosszabb esetben $\Theta(n^2)$, a gyakorlatban azonban – egyenletes eloszlású végrehajtási idők esetén – körülbelül $\Theta(n \lg n)$.

34.4.2. 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 lemezre 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 (**E**conomic **P**ower) 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.

34.4.3. 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 $\mathbf{t}_1$ lista elemei kerültek. Mivel mellettük legfeljebb $1 - t_1$, illetve $1 - t_2$ összegű fájlok lehetnek, így azok helyfoglalásainak 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 lemezszámot).

A rövidítés után megmaradó listát – például a korábbi algoritmusok valamelyikével – természetesen még fel kell dolgozni.

34.4.4. 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.

	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	2	3	2	2	2	2	2

34.19. ábra. Lemezszámok összefoglalása.

34.4.5. Algoritmusok páronkénti összehasonlítása

Ha egy ütemezési (vagy más) probléma megoldására több algoritmust ismerünk, ak- kor 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 lemezszámot $\mathrm{A}(\mathbf{t})$-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.

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 34.19. ábrán foglaltuk össze.

A 34.19. 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

	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

34.20. ábra. Algoritmusok páronkénti összehasonlítása.

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 34.20. 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 34.20. ábra lemezszámainak elemzését, a 34.20. ábrát a 34.21. 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 hármat. 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

34.8. 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 $\mathbf{t_i} = \langle t_1, t_2, \ldots, t_i \rangle$ $(i = 1, 2, \ldots, n)$. 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 után vagy FF lemezszáma kisebb lesz, vagy pedig egyenlő lemezszám mellett FF utolsó lemezének 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 34.21. ábrán összesítjük.

	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

34.21. ábra. Algoritmusok páronkénti összehasonlításának eredményei.

34.4.6. Közelítő algoritmusok hibája

Két algoritmus (A és B) relatív hatékonyságát gyakran jellemzik a kiválasztott hatékonysági mérték értékeinek hányadosával, jelen esetben az $\mathrm{A}(\mathbf{t})/\mathrm{B}(\mathbf{t})$ relatív lemezszámmal. 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 D_i azon valós listák halmaza, amelyek i elemet tartalmaznak, és legyen D az összes valós lista halmaza, azaz

$$D = \cup_{i=1}^{\infty} D_i \ .$$

Legyen $\mathcal{A}_{lsz}$ az ***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 $\mathrm{A} \in \mathcal{A}_{lsz}$ algoritmusok halmaza, amelyekre $\mathrm{A}(\mathbf{t}) \geq \mathrm{OPT}(\mathbf{t})$ minden $\mathbf{t} \in D$ listára, és van olyan $\mathbf{t} \in D$ lista, amelyre $\mathrm{A}(\mathbf{t}) > \mathrm{OPT}(\mathbf{t})$. Legyen A_{becs} azon $\mathrm{E} \in \mathcal{A}_{lsz}$ algoritmusok halmaza, amelyekre $\mathrm{E}(\mathbf{t}) \leq \mathrm{OPT}(\mathbf{t})$ minden $\mathbf{t} \in D$ listára, és van olyan $\mathbf{t} \in D$ lista, amelyre $\mathrm{E}(\mathbf{t}) < \mathrm{OPT}(\mathbf{t})$.

Legyen F_n azon valós listák halmaza, amelyekre $\mathrm{OPT}(\mathbf{t}) = n$, 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 (A, 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:

$$R_{A,B,n} = \sup_{t \in F_n} \frac{\mathrm{A}(\mathbf{t})}{\mathrm{B}(\mathbf{t})} \ ,$$

$$R_{A,B} = \sup_{t \in F} \frac{\mathrm{A}(\mathbf{t})}{\mathrm{B}(\mathbf{t})} \ ,$$

$$R_{A,B,\infty} = \limsup_{n \to \infty} R_{A,B,n} \ .$$

Ezek a mennyiségek főleg akkor érdekesek, ha B $\in \mathcal{A}_{opt}$. Ilyenkor az egyszerűség kedvéért a jelölésekből elhagyjuk a B-t, és az A $\in \mathcal{A}$, illetve az E $\in \mathcal{A}$ algoritmusok hibafüggvényéről, hibájáról és aszimptotikus hibájáról beszélünk.

A NF fájlelhelyező algoritmus jellemző adatai ismertek.

34.9. tétel. *Ha* $\mathbf{t} \in F_n$, *akkor*

$$n = \mathrm{OPT}(\mathbf{t}) \leq \mathrm{NF}(\mathbf{t}) \leq 2\mathrm{OPT}(\mathbf{t}) - 1 = 2n - 1 \ . \tag{34.22}$$

Továbbá, ha $k \in \mathbb{Z}$, *akkor léteznek olyan* $\mathbf{u}_k$ *és* $\mathbf{v}_k$ *listák, melyekre*

$$k = \mathrm{OPT}(\mathbf{u}_k) = \mathrm{NF}(\mathbf{u}_k) \tag{34.23}$$

és

$$k = \mathrm{OPT}(\mathbf{v}_k) \ \textit{és} \ \mathrm{NF}(\mathbf{v}_k) = 2k - 1 \ . \tag{34.24}$$

Ebből az állításból adódik a NF fájlelhelyező algoritmus hibafüggvénye, abszolút hibája és aszimptotikus hibája.

34.10. következmény. *Ha* $n \in \mathbb{Z}$, *akkor*

$$R_{\mathrm{NF},n} = 2 - \frac{1}{n} \ , \tag{34.25}$$

továbbá

$$R_{\mathrm{NF}} = R_{\mathrm{NF},\infty} = 2 \ . \tag{34.26}$$

A FF és BF fájlelhelyező algoritmus legrosszabb esetére vonatkozik a következő állítás.

34.11. tétel. *Ha* $\mathbf{t} \in F_n$, *akkor*

$$\mathrm{OPT}(\mathbf{t}) \leq \mathrm{FF}(\mathbf{t}),\ \mathrm{BF}(\mathbf{t}) \leq 1.7\mathrm{OPT}(\mathbf{t}) + 2\ . \tag{34.27}$$

Továbbá, ha $k \in \mathbb{Z}$, *akkor léteznek olyan* $\mathbf{u}_k$ *és* $\mathbf{v}_k$ *listák, melyekre*

$$k = \mathrm{OPT}(\mathbf{u}_k) = \mathrm{FF}(\mathbf{u}_k) = \mathrm{BF}(\mathbf{u}_k) \tag{34.28}$$

valamint

$$k = \mathrm{OPT}(\mathbf{v}_k)\ \textit{és}\ \mathrm{FF}(\mathbf{v}_k) = \mathrm{BF}(\mathbf{v}_k) = \lfloor 1.7k \rfloor\ . \tag{34.29}$$

A FF algoritmusra egy erősebb felső korlát is érvényes.

34.12. tétel. *Ha* $\mathbf{t} \in F_n$, *akkor*

$$\mathrm{OPT}(\mathbf{t}) \leq \mathrm{FF}(\mathbf{t}) < 1.7\mathrm{OPT}(\mathbf{t}) + 1\ . \tag{34.30}$$

Ebből a két állításból adódik FF és BF aszimptotikus hibája, valamint hibafüggvényük jó becslése.

34.13. következmény. *Ha* $n \in \mathbb{Z}$, *akkor*

$$\frac{\lfloor 1.7n \rfloor}{n} \leq R_{\mathrm{FF},n} \leq \frac{\lceil 1.7n \rceil}{n} \tag{34.31}$$

és

$$\frac{\lfloor 1.7n \rfloor}{n} \leq R_{\mathrm{BF},n} \leq \frac{\lfloor 1.7n + 2 \rfloor}{n} \tag{34.32}$$

továbbá

$$R_{\mathrm{FF},\infty} = R_{\mathrm{BF},\infty} = 1.7\ . \tag{34.33}$$

Ha n osztható tízzel, akkor a (34.31) egyenlőtlenségben az alsó és felső határok megegyeznek, azaz ebben az esetben $1.7 = R_{FF,n}$.

Gyakorlatok

34.4-1. Példával bizonyítsuk, hogy a FF és BF algoritmusok abszolút hibája legalább 1.7.

34.4-2. Valósítsuk meg a FF és BF algoritmusok alapgondolatát úgy, hogy a futási idő $O(n \lg n)$ legyen.

34.4-3. Egészítsük ki a 34.21 ábrát.

Feladatok

34-1 Lapcserélési anomália elkerülése
Osztályozzuk a tárgyalt lapcserélési algoritmusokat aszerint, hogy biztosítják-e az anomália elkerülését.

34-2 Optimális lapcserélési algoritmus
Bizonyítsuk be, hogy minden A igény szerinti lapcserélési algoritmusra, m memóriaméretre és R hivatkozási tömbre teljesül, hogy $f_{\mathrm{A}}(m, R) \geq f_{\mathrm{OPT}}(m, R)$.

34-3 Anomália
Tervezzünk egy algoritmust (és valósítsuk is meg), amelynél előfordulhat, hogy egy adott feladatot $q > p$ processzoron megoldani tovább tart, mint $p > 1$ processzoron.

34-4 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 [327], valamint Tanenbaum és Woodhull [341] tankönyvei alapján tárgyaljuk.

A lapcserélési algoritmusok Mealy-automatával való definiálása Denning összefoglaló cikkén [98], Gécseg Ferenc és Peák István [144], Hopcroft, Motwani és Ullman [184], valamint Peák István [283] tankönyvein alapul.

A MIN algoritmus optimalitását Mihnovszkij és Shor 1965-ben [261], majd Mattson, Gecsei, Slutz és Traiger 1970-ben [250] bizonyították.

A FIFO lapcserélési algoritmusnál a gyakorlatban tapasztalt anomáliát először Bélády László [52] írta le 1966-ban, majd Shedlerrel közös dolgozatában [51] 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 [132, 133] 2002-ben megmutatták, hogy a nagy és kisebb memóriájú számítógépekben szükséges lapcserék számának hányadosa akármilyen nagy lehet.A FIFO és LRU algoritmusok különböző szempontok szerinti összehasonlító elemzése található Leah Epstein et al. cikkeiben [111, 114].

Ütemezési anomáliára példák szerepelnek Coffman [79], Iványi és Szmeljánszkij [193], valamint Roosta [310] könyvében, továbbá Lai és Sahni [220] cikkében. Relációs adatbázisok frissítési anomáliáira Codd [75, 77] hívta

fel a figyelmet.

Ennek a fejezetnek az anyaga több helyen is kapcsolódik az *Ütemezéselmélet* című fejezethez. Például az 34.5. 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 [190] cikkből származik.

Az $\mathrm{NF}(\mathbf{t}) \leq 2\mathrm{OPT}(\mathbf{t}) + 2$ korlát D. S. Johnson PhD értekezésében szerepelt [199], a pontos 34.11. tétel a [191, 192] cikkekben jelent meg. A FF-re és BF-re vonatkozó első felső korlát Johnson, Demers, Ullman, Garey és Graham [200] eredménye, amelyet előbb Xia és Tan [372], majd Németh [273] javított, végül Dósa és Sgall [104] adta meg a pontos felső korlátot. Az FFD-re és BFD-re vonatkozó felső korlát forrása [200], az NFD-re vonatkozó korláté pedig [27]. A fájlelhelyezési feladat NP-nehézségének bizonyítása – a részletösszeg feladatra való visszavezetéssel – szerepel az *Új algoritmusok* közelítő algoritmusokkal foglalkozó fejezetében. [82]. A fájlelhelyezési feladat különböző változataira vonatkozó friss eredmények találhatók Leah Epstein és társszerzőinek (mint Csirik János, Imreh Csanád, Asaf Levin) dolgozataiban [59, 87, 112, 113, 115]

A témakör magyar nyelvű tankönyvei közül Varga László munkáját [354], Kernighan és Pike [209], Kóczy és Kondorosi [207], Tanenbaum és Woodhull [341] művét, valamint Galambos Gábor [140] 2003-ban megjelent tankönyvét ajánljuk.

35. Relációs adatmodell tervezése

35.1. 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 $U \to V$ 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.

35.1. 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 35.1 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.

35.2. Funkcionális függőségek

Egy adott relációs sémában teljesülő funkcionális függőségek közül némelyek már a tervezési szakaszban ismeretesek, mások pedig ezen függőségek következményei. A 35.1 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**.

35.1. 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*

$$F^+ = \{U \to V \colon F \models U \to V\} .$$

35.2.1. 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.

35.2. 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 (35.2-2 gyakorlat).

35.2. 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.

35.3. 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 $Y = A_1A_2 \dots A_n$, 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 35.2. lemma egyesítés szabálya alapján $X \to Y$ következik.

Másfelől, tegyük fel, hogy $X \to Y$ következik az axiómákból. A 35.2. lemma szétvágási szabálya alapján $X \to A_i$ következik az Armstrong-axiómákból minden $1 \leq i \leq n$-re. Tehát $Y \subseteq X^+$. ∎

35.2.2. 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

$$F = \{A \to B_1, A \to B_2, \dots, A \to B_n\}$$

funkcionális függőség rendszert. Ekkor F^+ az összes olyan $A \to Y$ funkcionális függőségből fog állni, ahol $Y \subseteq \{B_1, B_2, \dots, B_n\}$. 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              if Y ⊆ X^(i)
8                  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.

35.4. 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 35.3. 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 ered-

ményül 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 35.3. 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      for minden A ∈ W
 4          adjuk (W, Z)-t az L_A listához
 5      i[W, Z] = 0
 6  for minden A ∈ R \ X
 7      for minden (W, Z)-re az L_A listán
 8          i[W, Z] ← i[W, Z] + 1
 9  for minden (W, Z) ∈ F
10      if i[W, Z] = 0
11          adjuk (W, Z)-t a J listához
12  X+ = X
13                                          // Kezdeti értékek beállításának vége.
14  while J nem üres
15          (U, W) ← fej(J)
16          töröljük (U, W)-t a J listából
17          for minden A ∈ Z \ X+
18          for minden (W, Z)-re az L_A listán
19              i[W, Z] = i[W, Z] − 1
20              if i[W, Z] = 0
21                  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 $O(n)$.

35.2.3. 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.

35.5. 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: $(G - \{X \to A\})^+ \subsetneqq G^+$,*

3. *G-beli funkcionális függőségek bal oldalai minimálisak: $X \to A \in G$-re $X \to A \in G$, $Y \subsetneqq X \Longrightarrow ((G - \{X \to A\}) \cup \{Y \to A\})^+ \neq G^+$.*

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      for minden A ∈ Y − X
4          G = G ∪ X → A        // Most már minden jobb oldal egyelemű.
5  for minden X = A ∈ G
6      while van B ∈ X : A ∈ (X − B)⁺(G)
7          X = X − B                    // Minden bal oldal minimális.
9  for minden X = A ∈ G
10     if A ∈ X⁺(G − {X = A})
11         G = G − {X = A} // Nincs több elhagyható funkcionális függőség.
```

A 2–4. sorok végrehajtása után minden G-beli funkcionális függőség jobb oldala egyelemű. A $G^+ = F^+$ egyenlőség a 35.2. 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.

35.6. 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 35.2-1. gyakorlatot). Tehát X-et már a 6–8. sorok végrehajtásakor csökkentenünk kellett volna. ■

35.2.4. Kulcsok

Az adatbázis tervezésekor igen fontos megtalálni azon attribútumhalmazokat, melyek egyértelműen meghatározzák az egyes rekordok adatait.

35.7. 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.

35.8. 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 (35.2-4. 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 $e' = e$.

35.9. 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.

35.10. 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 (35.2-5. 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 35.10. állításban definiált halmazok.
2  if U ∩ V = ∅                                  // R − V az egyetlen kulcs.
3     return R − V
4  if (R − V)^+ = R
5     return R − V                               // R − V az egyetlen kulcs.
6  for az U ∩ V attribútumainak minden A_1, A_2, ... A_h sorrendjére
7        K = (R − V) ∪ U
10       for i = 1 to h
11             Z = K − A_i
12             if Z^+ = R
13             K = Z
14             return K
```

Gyakorlatok

35.2-1. 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.

35.2-2. Bizonyítsuk be a 35.2. lemma egyesítési, pszeudotranzitivitási és szétvágási szabályait.

35.2-3. 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.

35.2-4. 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 $e' = e$.

35.2-5. Bizonyítsuk be a 35.10. állítást.

35.3. 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.

35.3. példa. Tekintsük a következő sémát:

$$\text{SZÁLLÍTÓ-INFO(SNÉV,SCÍM,ÁRU,ÁR)} \ .$$

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:

$$\text{SZÁLLÍTÓK(SNÉV,SCÍM)}$$
$$\text{SZÁLLÍT(SNÉV,ÁRU,ÁR)}$$

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.

35.11. 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* $s = \bowtie_{i=1}^{n} r_i$.

35.4. 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 35.12. 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.

35.3.1. 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.

35.12. 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 (35.3-7 gyakorlat).

35.3.2. 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      for j = 1 to |R|
4          if A_j ∈ R_i
5              T[i,j] = 0
6              else T[i,j] = i              // Kezdeti értékek beállításának vége.
7  S = T
8  repeat
9          T = S
11         for minden {X → Y} ∈ F
12         for i = 1 to |ρ| − 1
13             for ji + 1 to |R|
14                 if minden X-beli A_h-ra S[i,h] = S[j,h]
15                 EGYENLŐSÍT(i,j,S,Y)
16 until S = T
17 if van S-ben csupa 0 sor
18    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      if S[i,l] < S[j,l]
3          u = S[i,l]
4          v = S[j,l]
5          else u = S[j,l]
6          v = S[i,l]
7          for d = 1to k
8          if S[d,l] = v
9              S[d,l] ← u
```

35.5. 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 35.1(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 35.1(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 35.1(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 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.

35.13. 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

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)

35.1. ábra. KAPCSOLÁS-TESZT(R, F, ρ) alkalmazása. **(a)** Kiindulási tömb. **(b)** $A \leftarrow B$ és $B \leftarrow C$ alkalmazásának eredménye. **(c)** A végeredmény.

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 eljárást, 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

$$\{a_1 a_2 \dots a_n \mid (\exists b_{11}) \dots (\exists b_{kn}) \, (R(w_1) \wedge \dots \wedge R(w_k))\} \ , \tag{35.1}$$

ahol w_i a T i-edik sora. Ha T a kiindulási tábla, akkor a (35.1) 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 (35.1) 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 (35.1)-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:

$$\{a_1 a_2 \dots a_n \mid (\exists b_{11}) \dots (\exists b_{kn})\, (R(a_1 a_2 \dots a_n) \wedge \cdots)\} \ . \tag{35.2}$$

Világos, hogy ha (35.2)-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 (35.2) formula $m_\rho(r)$-t számolja ki. A 35.12. lemma 1. része szerint $r \subseteq m_\rho(r)$ teljesül, így ha r kielégíti F-t, akkor a (35.2) 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.

35.14. 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	$00\dots0$	$22\dots2$	$00\dots0$

(35.3)

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 KAPCSOLÁS-TESZT 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 $(R_1 \cap R_2) \to (R_1 - R_2)$. ■

35.3.3. 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 $\pi_\rho(F) = \bigcup_{i=1}^{k} \pi_{R_i}(F)$.

35.15. 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.

35.6. példa. Legyen $R = (\mathbf{V}\text{áros},\mathbf{U}\text{tca},\mathbf{I}\text{rányítószám})$ 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 35.14. 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      Z = X
3      repeat
4              W = Z
5              for i = 1 to k
6                  Z = Z ∪ (LINEÁRIS-LEZÁRÁS(R, F, Z ∩ R_i) ∩ R_i)
7              until Z = W
8      if Y ⊈ Z
9         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.

35.7. 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

$$\begin{aligned} Z &= \{D\} \cup \big((\{D\} \cap \{C, D\})^+ \cap \{C, D\}\big) \\ &= \{D\} \cup \big(\{D\}^+ \cap \{C, D\}\big) \\ &= \{D\} \cup (\{A, B, C, D\} \cap \{C, D\}) \\ &= \{C, D\}\,. \end{aligned}$$

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.

35.16. 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. ■

35.3.4. 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

35.17. 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 $A \not\subseteq X$ esetén X szuperkulcs.*

A BCNF fontos tulajdonsága, hogy megszünteti a redundanciát. Ez a következő tételen alapszik, melynek bizonyítását gyakorlatként az Olvasóra bízzuk (35.3-8 gyakorlat).

35.18. 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 35.18. 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 35.6. 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.

35.19. definíció. *Az (R, F) séma **harmadik normálformában** van, ha valahányszor $X \to A \in F^+$, akkor vagy X szuperkulcs, vagy A prím attribútum.*

A 35.3. 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 35.6 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 *35-4* 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 veszteségmentes;
- 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.

35.20. 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 35.20. lemma bizonyítása a természetes összekapcsolás asszociativitásán alapszik. A részleteket az Olvasóra bízzuk gyakorlatként (35.3-9 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 → A} ∈ G, ami megsérti BCNF-t
2     S1 = {XA}
3     S2 = S − A
4     G1 = π_S1(G)
5     G2 = π_S2(G)
6     return (NAIV-BCNF(S1, G1) ∪ NAIV-BCNF(S2, G2))
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.

35.21. 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 nemtriviális 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 35.21. 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 35.21. 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.

35.22. 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 (35.3-10 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 35.14. tétel szerint a szétvágás veszteségmentes összekapcsolású. Utána ezt az eljárást alkalmazzuk rekurzívan $R-A$-ra, amíg egy olyan sémához nem érünk, amelyik teljesíti a 35.21. lemma 2. pontjának feltételét. A 35.20. 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        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              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 csak 3NF-et követelünk 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 F valamely minimális fedését, G-t. Legyen $G = \{X_1 \to A_1, X_2 \to A_2, \ldots, X_k \to A_k\}$.

35.23. 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 $U \subsetneqq X_i$. 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, $Y \subsetneqq X_i$, 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 35.23. 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 (35.3-11 gyakorlat).

35.24. 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 35.23. 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 *35-4* 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.

35.3.5. Többértékű függőségek

Tekintsük a következő példát.

35.8. példa. A 35.1 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:

35.25. 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 35.8 példában Tá→→IT teljesül.

[1] Elegendő csak t_3 létezését megkövetelni, mert abból már t_4 létezése következik. Azonban a többértékű függőség szimmetriája így jobban látszik.

35.26. 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.

Vegyük észre, hogy ha $X \to Y$ teljesül, akkor $X \twoheadrightarrow Y$ is igaz. A 35.25. 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 (35.3-12 gyakorlat), a teljesség bizonyítása meghaladja e könyv kereteit. Az (A1)–(A8) axiómákon kívül a 35.2. lemma szabályai ugyanúgy érvényesek, mint mikor csak funkcionális függőségeket tekintettünk. További szabályokat ad a következő állítás.

35.27. 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 35.27. állítás bizonyítását gyakorlatként az Olvasóra hagyjuk (35.3-13 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.

35.28. 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 35.27. 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 35.27. állítás egyesítés szabálya alapján a partíció néhány blokkjának egyesítése többértékűen függ X-től. ∎

35.29. definíció. *A 35.28. 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.*

35.9. példa. Tekintsük a 35.1 és a 35.8 példákból már jól ismert

R(**T**anár,**Tá**rgy,**Te**rem,**D**iák,**J**egy,**I**dő)

sémát. A 35.8 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.

35.30. 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 $Y = A_1 A_2 \ldots A_k$, 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.

35.31. tétel. (Beeri). *Tegyük fel, hogy $A \notin X$ és X függőségi bázisát a 35.30. 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              if van Y ∈ S melyre Y ∩ W ≠ ∅ ∧ Y ∩ V = ∅
5                  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 (*35-5*. 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.

35.32. 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.

35.33. 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.

35.34. 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 35.21. lemma szerint.

Gyakorlatok

35.3-1. 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 $X \to Z$.

35.3-2. Bizonyítsuk be a 35.30. tételt, azaz lássuk be a következőt:

Legyen D funkcionális és többértékű függőségekből álló halmaz, és legyen $m(D) = \{X \twoheadrightarrow Y : X \twoheadrightarrow Y \in D\} \cup \{X \twoheadrightarrow A : A \in Y \text{ valamely } X \to Y \in D\text{-re}\}$. Ekkor igazak az alábbiak:

a. $D \models X \to Y \implies m(D) \models X \twoheadrightarrow Y$, és

b. $D \models X \twoheadrightarrow Y \iff m(D) \models X \twoheadrightarrow Y$.

Útmutató. A b. pont bizonyításához alkalmazzunk indukciót a levezetési szabályokon.

35.3-3. 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.

35.3-4. A 35.3-3 gyakorlat S sémáját szétvágjuk az RO, $BeBr$, $BeRM$ és BrI sémákra. Veszteségmentes kapcsolású ez a szétvágás?

35.3-5. Tegyük fel most, hogy a 35.3-3 gyakorlat S sémáját a $BeRM$, $BeBr$, RO és $BeRI$ sémákkal reprezentáljuk. Adjuk meg a 35.3-3 gyakorlatban szereplő függőségek a megadott részsémára való vetületeinek minimális fedőjét. Találjunk minimális fedőt a vetített függőségi halmaz184-803ok egyesítéséhez. Függőségőrző ez a szétvágás?

35.3-6. A 35.3-3. 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.

35.3-7. 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. $m_\rho(m_\rho(r)) = m_\rho(r)$.

35.3-8. 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 $A \notin Y$.

35.3-9. Bizonyítsuk be a 35.20. lemmát.

35.3-10. Tegyük fel, hogy $R_2 \subset R_1 \subset R$, és az R séma funkcionális függőségeinek halmaza F. Bizonyítsuk be, hogy $\pi_{R_2}(\pi_{R_1}(F)) = \pi_{R_2}(F)$.

35.3-11. 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.

35.3-12. Bizonyítsuk be, hogy az (A1)–(A8) axiómák ellentmondásmentesek a funkcionális és többértékű függőségek körében.

35.3-13. Vezessük le az (A1)–(A8) axiómákból a 35.27. állítás négy szabályát.

35.4. Általános függőségek

35.4.1. 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 35.33. tétel azt mondja ki, hogy a többértékű függőség ténylegesen ekvivalens azzal, hogy valamely szétvágás két részre veszteségmentes összekapcsolású. Ennek általánosítása az ***összekapcsolási függőség.***

35.35. 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).***

35.36. 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.

35.10. 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.

35.37. 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:

35.38. 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.*

35.4.2. 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 $\boldsymbol{(p,q)}$-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.

35.11. 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.

35.39. 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 35.2-1. gyakorlatban szereplő három tulajdonságán alapul. Elágazó függőségekre a három közül csak az első kettő teljesül általában.

35.40. lemma. *Legyen $0 < p \leq q$, 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

35.4-1. Bizonyítsuk be a 35.38. tételt.

35.4-2. Bizonyítsuk be a 35.40. lemmát.

35.4-3. Bizonyítsuk be, hogy ha $p = q$, akkor a 35.40. lemmában szereplő két tulajdonságon kívül az is igaz, hogy $C_{p,p}\left(C_{p,p}(X)\right) = C_{p,p}(X)$.

35.4-4. ***Lezárásnak*** nevezünk egy R séma részhalmazaiból a részhalmazaiba képező függvényt, ha a 35.40. lemma két tulajdonságát és a 35.4-3. 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.

35.4-5. Legyen C az a lezárás, melyre

$$C(X) = \begin{cases} X, & \text{ha } |X| < 2\,, \\ R & \text{egyébként}\quad . \end{cases}$$

Bizonyítsuk be, hogy C-hez *nincs* Armstrong-reláció az (n,n)-függőségek családjában, ha $n > 2$.

Feladatok

35-1 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.

35-2 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 35.6. 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.

35-3 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.

35-4 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.

35-5* 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 [75] vezette be 1970-ben. A funkcionális függőségeket 1972-ben megjelent [77] tanulmányában tárgyalta, axiomatizálásuk Armstrong nevéhez fűződik [24]. A logikai következmény feladatot funkcionális függőségekre Beeri és Bernstein [33], valamint Maier [245] 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 [246]. Beeri, Fagin és Howard bizonyították, hogy az (A1)–(A8) axiómarendszer ellentmondásmente és teljes. A funkcionális és többértékű függőségekre [36] Yu és Johnson [377] konstruáltak olyan relációs sémát, melyben $|F| = k^n$, és a kulcsok száma $k!$. Békéssy és Demetrovics [48] 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 [119, 120], valamint Beeri, Fagin, Dowd és Statman [35] vezették be és tanulmányozták.

A többértékű függőségeket Zaniolo [384], Fagin [118] és Delobel [93] egymástól függetlenül fedezték fel.

A normálformák szükségességét Codd vetette fel a frissítési anomáliák tanulmányozásakor [76, 77]. A Boyce-Codd normálformát [78] vezeti be. A harmadik normálforma általunk használt definícióját Zaniolo [385] adta meg. A negyedik normálformát Fagin [118] vezette be. A normálformákra bontás bonyolultsági kérdéseit Lucchesi és Osborn [238], Beeri és Bernstein [33], valamint Tsou és Fischer [348] vizsgálták.

A 35.30. és 35.31. tételek Beeri eredményei [34]. A 35.34. tétel Aho, Beeri és Ullman cikkéből [10] való. A 35.37. és a 35.38. tételek megtalálhatók Abiteboul, Hull és Vianu könyvében, [3] az összekapcsolási függőségek axiomatizálhatóságáról Petrov [286] mutatta ki, hogy nem lehetséges.

Az elágazó függőségeket Demetrovics, Katona és Sali vezette be, tanulmányozták Armstrong-relációk létezését és a minimális Armstrong-relációk méretét [94, 95, 96, 312]. Az elágazó függőségek nem axiomatizálhatósága Katona nem publikált eredménye (ICDT'92 Berlin, meghívott előadás).

Numerikus függőségek axiomatizálásának lehetőségét Grant és Minker tárgyalja [158, 159].

A fejezetben szereplő fogalmak jó összefoglalása és bevezetése található Abiteboul, Hull és Vianu [3], Ullman [352] és Thalheim [343] könyveiben.

A fejezet témakörében magyar nyelven Békéssy és Demetrovics [49], Gajdos Sándor [138], Garcia-Molina, Ullman és Widom [141, 353] könyveit, Benczúr András, Kiss Attila és Márkus Tibor cikkét [41], valamint Békéssy András és Demetrovics János tankönyvét [50] ajánljuk.

36. A szemantikus web alkalmazásai

A szemantikus web célja, hogy összekapcsolja és a számítógépek által értelmezhetővé tegye az Interneten elérhető adatokat. Ehhez biztosít egy olyan adatmodellt (az RDF-et), amely lehetővé teszi az adatok leírását és integrálását. A **??**. fejezetben láthattuk, hogyan épül fel formálisan az adatmodell, illetve hogyan lehet az RDF-ben reprezentált adatokat lekérdezni SPARQL segítségével. Ebben a fejezetben megvizsgáljuk, miként lehet elosztott módon kiértékelni azokat a SPARQL lekérdezéseket, melyek WHERE záradékukban csak hármas mintákat tartalmaznak. Erre azért van szükség, mert a leírás mélységétől függően egy-egy adathalmaz igen nagy méretű lehet. Például, ha egy több millió felhasználóból álló közösségi hálót akarunk RDF-ben reprezentálni, akkor ehhez legalább annyi hármast kell felhasználni, ahány él van a hálóban. A közösségi hálózatok reprezentálására két megoldást is megnézünk a 36.3. alfejezetben, ahol továbbá azt is megvizsgáljuk, hogyan lehet transzformációkat végezni a közösségi hálókon az RDF reprezentációjukat felhasználva. Emellett egy olyan SPARQL kiegészítést mutatunk be, ami lehetővé teszi a felhasználói csoportok kiválasztását, elemzését. Végül megnézzük, hogyan lehet speciálisan a közösségi hálókat lekérdezni a MapReduce módszerrel.

36.1. A SPARQL kiértékelése MapReduce módszerrel

A szemantikus web adathalmazai sokszor nagy méretűek. Minél pontosabban akarjuk leírni az adatokat, annál több hármasra van szükségünk. A jelenlegi keretrendszerek nehezen tudják kezelni a nagy méretű gyűjteményeket. Emiatt vált fontossá annak a vizsgálata, hogyan tudnánk gyorsan megválaszolni a SPARQL lekérdezéseket nagy adathalmazokon. Erre egy kézenfekvő megoldás a feldolgozás párhuzamosítása, elosztása több gépen. A Google által kifejlesztett MapReduce egy módszer arra, hogy bizonyos feladatokat több, klaszterbe szervezett gépen párhuzamosan oldjunk meg. Ehhez a feladatokat két fázisra kell osztani: a Map kulcs-érték párokká alakítja a bemenetet és továbbítja a Reduce fázisnak, a Reduce feldolgozza az azonos kulcshoz tartozó értékeket és előállítja a végeredményt. A Hadoop egy ingyenesen elérhető implementációja a MapReduce technikának, melyben a fent említett két fázist

együtt egy jobnak nevezzük. Ebben az alfejezetben azt vizsgáljuk, hogyan lehet felhasználni a SPARQL lekérdezések kiértékeléséhez a Hadoop jobokat.

36.1.1. Adatok tárolása

A Hadoop job bemenete fájloknak egy halmaza, ezért lehetőség van arra, hogy az adatokat szétvágva több kisebb fájlba szervezzük. Ha az összes adatot egy fájlban tároljuk, akkor a rendszer minden alkalommal végigolvassa a teljes adathalmazt. Ennél hatékonyabb megoldás, ha több kisebb fájlunk van, mert ekkor nem feltétlenül szükséges végigolvasni a teljes adathalmazt, csak a szükséges részeit. Az adatok hatékony particionálásával több cikk foglalkozik, ezek közül kettőt mutatunk be. Az első megközelítésben az állítmányuk szerint osztjuk szét a hármasokat, ezt állítmányalapú vágásnak nevezik. Mivel egy RDF adathalmaz jellemzően nem tartalmaz sokféle állítmányt, így az adatok nem aprózódnak el, azonban lekérdezésnél elegendő a lekérdezésben szereplő állítmányok fájljait feldolgozni. Ez a módszer éppen ezért akkor használható hatékonyan, ha az állítmány helyén nem szerepelnek változók. A másik megközelítés az elsőnek egy finomítása, ahol a hármasokat tovább bontjuk a tárgy alapján, ezért ezt állítmány-tárgy alapú vágásnak. Többféle felosztás létezik, az egyik legegyszerűbb, ha a tárgy típusa alapján vágunk, tehát az IRI és a literál típusú értékek külön fájlokba kerülnek.

Bemeneti fájl kiválasztása

A szükséges fájlok meghatározásához meg kell vizsgálnunk a felhasználó által írt lekérdezést. A WHERE záradékban található hármasok alapján választjuk ki a megfelelő fájlokat. Ha valamely hármasban az állítmány egy változó, akkor minden fájlt kiválasztunk és befejeződik a futás. Ha nincs olyan hármas, amelyben az állítmány egy változó, akkor a tárgyban szereplő elemeket vizsgáljuk tovább. Ha valamely tárgyban változó szerepel, akkor minden olyan fájlt kiválasztunk, amely az adott állítmányhoz tartozik. Azonban, ha a tárgy nem változó, akkor a tárgy típusa tovább szűkíti a szükséges fájlok halmazát, amennyiben a második megközelítés alapján osztottuk el az adatainkat.

36.1.2. GenerateBestPlan algoritmus

A következő részben bemutatunk egy algoritmust, ami előállítja a lekérdezési terveket egy adott lekérdezéshez. Mielőtt rátérnénk a lekérdezés kiértékelésének költségszámítására, bevezetjük a fontosabb definíciókat, melyekre a későbbiekben hivatkozni fogunk. A definíciók megértéséhez nézzük az alábbi lekérdezést.

36.1. példa. Az alábbi SPARQL lekérdezés a <http://www.University0.edu> IRI-

vel reprezentált egyetem tanszékeit és azok vezetőit adja eredményül. Az rdf:type megmondja egy erőforrásról, hogy mely osztályba tartozik. qnindent

```
1 SELECT ?X, ?Y WHERE {
2       ?X rdf:type ub:Chair .
3       ?Y rdf:type ub:Department .
4       ?X ub:worksFor ?Y .
5       ?Y ub:subOrganizationOf <http://www.University0.edu>
6 }
```

36.1. definíció. (Hármas-minta, TP)
*A **hármas-minta** alany, állítmány és tárgy elemek olyan rendezett hármasa, amelyek a SPARQL lekérdezés WHERE záradékában szerepelnek. Az elemek lehetnek változók vagy konstansok.*

A példában a 2., 3., 4., és 5. sorok egy-egy TP-t tartalmaznak.

36.2. definíció. (MapReduce join, MRJ)
*A **MapReduce** join kettő vagy több hármas minta egy változón keresztül történő összekapcsolása.*

A 2. és a 4. sorban szereplő TP-k az ?X változón, a 3. és az 5. sorban szereplők az ?Y változón keresztül vannak összekapcsolva, ezek tehát MRJ-k. Ugyanígy a 3. és a 4., valamint a 4. és az 5. sorban szereplők. Továbbá a 3., 4. és 5. sor TP-i egy három tagú MRJ-t alkotnak.

36.3. definíció. (Konfliktusos MapReduce join, CMRJ)
Két MapReduce joint konfliktusosnak nevezünk, ha van közös változójuk, de az összekapcsolási feltételben különböző változók szerepelnek.

36.4. definíció. (Nem-konfliktusos MapReduce join, NCMRJ)
*Ha nincs közös változójuk, vagy a közös változó szerepel az összekapcsolási feltételben, akkor **nem-konfliktusos MapReduce joinokról** beszélünk.*

Ez alapján a 2. és 4. illetve a 4. és 5. sorban szereplő TP-kből képzett MapReduce joinok konfliktusosak, hiszen a 2. és 4. az ?X-en keresztül, míg a 4. és 5. az ?Y-on keresztül kapcsolódik, de az ?Y egy közös változó. Ha ugyancsak a 2. és 4. sort nézzük, de most a 3. és 5. sorral, akkor nem-konfliktusos MapReduce joint kapunk, hiszen nincs közös változójuk.

36.5. definíció. (job, JB)
*Azt a folyamatot, amelyben egy vagy több MapReduce join található, **jobnak** nevezzük. A job bemenetként fájlokat kap és kimenetként fájlokat készít.*

A jobokat tovább csoportosíthatjuk aszerint, hogy a bennük szereplő MapReduce joinoknak vannak-e közös hármas-mintáik. Ez alapján megkülönböztetünk megvalósítható és nem-megvalósítható jobokat. A közös hármas-minták gondot okoznak, mert ilyen esetben nem egyértelmű, hogy a Map fázis során milyen kulcsot rendeljünk egy adott hármashoz.

36.6. definíció. (Megvalósítható job, FJ)
*Egy jobot **megvalósíthatónak** nevezünk, ha nincs egynél több MapReduce joinban szereplő hármas-mintája.*

36.7. definíció. (Nem-megvalósítható job, IFJ)
*Egy jobot **nem-megvalósíthatónak** nevezünk, ha van olyan hármas-mintája, amely több MapReduce joinban szerepel.*

Egy lekérdezés megválaszolásához több Hadoop job egymás utáni végrehajtása szükséges. A jobok sorozatát a lekérdezési terv határozza meg. A lekérdezési terveket szintén csoportosíthatjuk megvalósíthatóság szempontjából, a következő definícióknak megfelelően.

36.8. definíció. (Lekérdezés Terv, QP)
*Egy **SPARQL lekérdezéshez tartozó lekérdezési terv** Hadoop jobok olyan sorozata, amely helyesen megválaszolja a SPARQL lekérdezést.*

36.9. definíció. (Megvalósítható Lekérdezési Terv, FQP)
*Egy lekérdezési tervet **megvalósíthatónak** nevezünk, ha minden benne szereplő job megvalósítható.*

36.10. definíció. (Nem-megvalósítható Lekérdezési Terv, IQP)
*A **nem-megvalósítható lekérdezési terv** olyan lekérdezési terv, amelyben van legalább egy olyan job, amely nem-megvalósítható.*

Ahhoz, hogy egy lekérdezéshez előállítsuk az összes lehetséges megvalósítható lekérdezési tervet, olyan jobokat kell képezni, melyek megvalósíthatóak és meg kell határozni azok sorrendjét. Ehhez a következő, hármas mintákból képzett gráfok nyújtanak segítséget.

36.11. definíció. (Hármas-minta Gráf, TPG)
*A **hármas-minta gráf** $G_{TP} = (V_{TP}, E_{TP})$ egy irányítatlan élcímkézett gráf, amelynek csúcsai a lekérdezésben szereplő hármas-minták. Két csúcs között pontosan akkor vezet él, ha van közös változójuk, és ekkor az adott él a közös változók halmazával címkézett. Azaz egy $(u, v, c) \in E_{TP}$ él esetén $u, v \in V_{TP}$ csúcsok és $c = var(u) \cap var(v)$ él, ahol a var a TP-ben szereplő változók halmaza.*

Ez a hármas-minta gráf lesz az alapja a join gráfnak, amely alapján össze tudjuk állítani a megfelelő jobokat.

36.12. definíció. (JOIN GRÁF, JG)
A G_{TP} hármas-minta gráfból készített $G_J = (V_J, E_J)$ ***join gráf*** *egy irányítatlan élcímkézett gráf, amelynek minden $v \in V_J$ csúcsa egy $(u_{TP}, v_{TP}, c_{TP}) \in E_{TP}$ élt reprezentál, és az $u_J, v_J \in V_J$ csúcsok között pontosan akkor vezet él, ha a megfelelő élek a G_{TP} gráfban szomszédosak, és az élcímkékben szereplő változóhalmazok diszjunktak.*

A jobok összeállításánál a célunk olyan MapReduce joinok összeválogatása, melyek páronként nem-konfliktusosak. Továbbá a lekérdezési tervekben szereplő jobok függhetnek egymástól, az ilyen jobok futtatása sorban történik. A helyes sorrend a join gráf színezésével állítható elő a következő módon. Az alábbi algoritmus előállítja a lehetséges színezéseket és a hozzájuk tartozó lekérdezési terveket. A lekérdezési terveknek különböző szempontok alapján lehet értékelni a hatékonyságát, majd kiválasztani közülük a leghatékonyabbat. Ilyen szempontok lehetnek például a jobok száma, a jobok bemenetének vagy kimenetének a mérete.

Input. Query q.
Output. plan
GENERATEBESTPLAN

```
1 plans = CreateEmptyPriorityQueue()
2 tpg = CreateTriplePatternGraph(q)
3 jg = CreatejoinGraph(tpg)
4 jobs = {}
5 COLORGRAPH1(tpg,jg,0,jobs)
6 return getHeadFromPriorityQueue(plans)
```

A színezés pontonként történik a COLORGRAPH1 meghívásával, amely kezdetben megvizsgálja, hogy van-e az adott csúcsnak fehér színű szomszédja, ha nincs, akkor a fehérrel meghívja a ColorGraph2 metódust, mely a tényleges színezést végzi. Majd a szomszédok színétől függetlenül ugyanarra a csúcsra meghívja a ColorGraph2-t fekete színnel is.

Input. TPG tpg, JG jg, int i, Set jobs.
Output.
COLORGRAPH1

```
1 if !neighborHasColor(jg, i, WHITE) then
2       ColorGraph2(tpg, jg, i, WHITE, jobs)
3 ColorGraph2(tpg, jg, i, BLACK, jobs)
```

A ColorGraph2 algoritmus végzi a tényleges színezést. Az algoritmus megőrzi az adott csúcs régi színét, majd ellenőrzi, hogy ez lesz-e az utolsó színezés, ugyanis legfeljebb annyi jobunk lehet, ahány csúcsa van a join gráfnak. Ha nem, akkor rekurzívan meghívjuk a ColorGraph1 algoritmust, különben elkészítjük a jobot. Ezután ellenőrizzük, hogy a készülő job szerepele már a jobok halmazában. Ha nem, akkor hozzávesszük. Ezután megnézzük, hogy az összes join elvégezhető-e az eltárolt jobok segítségével, ha igen, akkor elkészítjük a lekérdezési tervet, különben újraépítjük a gráfokat a megfelelő TP-k összevonása után és rekurzívan meghívjuk a ColorGraph1-et. Az algoritmus végén visszaállítjuk a csúcs eredeti színét. Erre azért van szükség, hogy egy visszalépéses kereséssel az összes lekérdezési tervet elő tudjuk állítani.

Input. TPG tpg, JG jg, int i, Color color, Set jobs.
Output.
ColorGraph2

```
 1 v = Vj[i]
 2 prev_color = color[v]
 3 color[v] = color
 4  if  i < |vertices[jg]-1|
 5      ColorGraph1(tpg,jg,i+1, jobs)
 6 else job = createjob(vertices[jg])
 7       if  !detectDuplicatejob(job, tpg)
 8           jobs = jobs ∪ job
 9           if |joins[job]| == |vertices[jg]|
10              enqueuePlan(plans, createNewPlan(jobs))
11           else tpg_new = mergejoinedTriplePatterns(tpg, job)
12                jg_new = CreatejoinGraph(tpg_new)
13                ColorGraph1(tpg_new, jg_new, 0, jobs)
14                jobs = jobs\{job}
15 color[v] = prev_color
```

36.13. tétel. *A* GenerateBestPlan *nem állít elő megvalósíthatatlan tervet.*

Bizonyítás. Tegyük fel, hogy az algoritmus előállít megvalósíthatatlan tervet. A nem-megvalósítható lekérdezési terv definíciójából adódik, hogy ekkor lennie kell legalább egy megvalósíthatatlan jobnak, ahol a hármasok úgy vannak kiválasztva, hogy azok közül legalább egy több joinban is szerepel. Legyen $v \in V_{TP}$ egy csúcs a G_{TP}-ben, amely több joinban is szerepel. Az $e_1, e_2 \in E_{TP}$

élek legyenek a v-ben szomszédosak úgy, hogy az élcímkén szereplő változók halmaza diszjunkt. Az e_1, e_2 éleknek feleljenek meg a $v_1, v_2 \in V_J$ csúcsok a G_J-ben. Ahhoz, hogy ez a két csúcs egyszerre fusson le egy adott jobban, mindkettőnek fehérnek kell lennie, ami a színezés miatt nem lehetséges. Ugyanis ha egy csúcsnak van fehér szomszédja, akkor azt már nem színezhetjük fehérre. ■

36.14. tétel. GenerateBestPlan *előállítja az összes megvalósítható lekérdezést.*

Bizonyítás. Tegyük fel, hogy az algoritmus nem állít elő egy megvalósítható lekérdezési tervet. Ez más megfogalmazásban annyit jelent, hogy nem állít elő legalább egy megvalósítható jobot, azaz valamely érvényes színezésű join gráfot. Legyen $C = \{c_1, c_2, \ldots, c_n\}$ egy ilyen színezés, ahol $c_i \in \{BLACK, WHITE\}$ a megfelelő $v_i \in V_J$ csúcs színe, és $n = |V_J|$. Tegyük fel, hogy $\{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ csúcsok színe fehér egy megvalósítható lekérdezési tervben, ahol $k \leq n$. Ahhoz, hogy az algoritmus ne generálja ezt a tervet, léteznie kell egy (v_{i_m}, v_{i_n}) párnak, amely között van egy él, amely meggátolja, hogy mindkét csúcs egyszerre fehér legyen. De ha a C érvényes színezésben ez a két csúcs fehér, akkor a G_{TP}-ben nem lehet a megfelelő éleknek olyan közös csúcsuk, melyek változó listája nem diszjunkt. Vagyis a G_J-ben nem lehet köztük él, ami ellentmondás. ■

36.2. Alkalmazás közösségi hálók elemzésére

A Web 2.0 megjelenésével interaktívvá váltak a weblapok. Ekkortól számítva a felhasználók már nem csupán böngészik, hanem ők maguk is hozzájárulnak egy-egy oldal tartalmához. Például leírják a véleményüket az oldalon bemutatott termékről, kommenteket fűznek a hírekhez, címkéket rendelnek a tartalomhoz ami segíti azok kategorizálását. Sőt, mára egyre több olyan szolgáltatás érhető el, amely csak egy keretrendszert ad a felhasználóknak, amivel saját tartalmakat hozhatnak létre, tölthetnek fel, videót, fotót, receptet, személyes bejegyzéseket vagy bármi mást. Emellett megjelentek az úgynevezett közösségi oldalak (például Facebook, Twitter, LinkedIn), ahol a felhasználók ismerkedhetnek, megoszthatják egymással az információkat, tartalmakat, kapcsolatokat alakíthatnak ki. Ezek a kapcsolatok egy nagy közösségi hálót alkotnak.

Az ipar és a kutatók is felismerték, hogy mennyire értékesek az ilyen, interaktív oldalakon a felhasználók által létrehozott tartalmak. Ipari szempontból érdekesek lehetnek például egy adott termékre vonatkozó hozzászólások, melyekből a gyártó visszajelzéseket kap, megismerheti a vásárlók véleményét.

A sok pozitív vélemény elősegíti a termékek értékesítését. A közösségi hálókon megosztott adatok alapján célzott hirdetések, reklámok hozhatók létre. Tudományos szempontból pedig a kapcsolati hálók segítségével elemezhetővé válik a társadalmi kapcsolatoknak, a közösségeknek a kialakulása és szerveződése. A hálók különböző mérőszámokkal jellemezhetőek, szokták vizsgálni például a csomópontok fokszámának eloszlását, a háló átmérőjét, sűrűségét (azaz a lehetséges és a létező kapcsolatok arányát).

36.2.1. Közösségi hálózatok reprezentálása RDF segítségével

Napjainkban az Interneten egyre több, különböző típusú, eltérő témájú közösségi oldalt találunk, mint például a Facebook, Google+ vagy a LinkedIn. Ezek közös jellemzője, hogy a felhasználók különféle kapcsolatokat alakítanak ki egymás (barát, rokon, munkatárs) és a tartalmak (látta a filmet, olvasta a könyvet, megvette a terméket) között. A bevezetőben szó volt róla, hogy ezeknek a hálózatoknak az elemzése milyen előnyökkel jár. Nincsen kidolgozott szabvány azonban a hálók reprezentálására, a különböző oldalak eltérő módon, saját formátumokban tárolják azokat, megnehezítve az elemzők dolgát. Egyrészt a kifejlesztett algoritmusokat hozzá kell igazítani az éppen vizsgált háló tárolási módjához, másrészt a különböző forrásból származó hálózatokat nehéz integrálni.

A szemantikus web megoldást nyújt ezekre a problémákra, hiszen az RDF keretrendszer egyik fő célja az adatok integrálásának elősegítése. Ehhez egy olyan leíró nyelvet biztosít, ami elég flexibilis a különböző témájú és szerkezetű közösségi hálók reprezentálásához. Továbbá ontológiák segítségével definiálhatjuk a kapcsolatok lehetséges típusait, a felhasználók, illetve a tartalmak (közösen aktorok) lehetséges osztályait. Ilyen ontológiák már léteznek, a legismertebbek közülük a ***Friend of a Friend*** (FOAF) és a ***Semantically-Interlinked Online Communities*** (SIOC).

A közösségi hálózatokat alapvetően kétféleképpen lehet reprezentálni. Az első a hagyományos megközelítés, amikor a háló csúcsai az aktorok, az élek pedig a köztük lévő kapcsolatok. Az aktorokhoz tartozhatnak attribútumok, amik speciális csúcsként jelennek meg.

36.15. definíció. ***(közösségi háló)*** *A* ***közösségi háló*** *egy* $G = (V, E, C, cimke())$ *élcímkézett irányított gráf, ahol*

1. $V = A \cup M$ *a csúcsok halmaza, az aktorok* A *és a tőle diszjunkt attribútumok* M *halmazának az uniója. Az aktorok halmaza tovább van particionálva típusok szerint, vagyis* $A = \bigcup_t A_t$*, valamint az attribútumok is lehetnek típusosak.*

2. $E = E_{AA} \cup E_{AM}$ *az élek halmaza, az* $E_{AA} \subseteq A \times A$ *aktorok közötti és*

$E_{AM} \subseteq A \times M$ *aktorok és attribútumok közötti élek diszjunkt halmazának uniója,*

3. $C = C_{AA} \cup C_{AM}$ *az adott éltípushoz tartozó élcímkék halmaza,*

4. $cimke()$ *az élcímkéző függvény, és* $cimke(e)$ *az* e *él címkéjével egyenlő.*

Érdemes felhívni a figyelmet arra, hogy a definícióban az A halmaz particionálása, azaz osztályok szerinti szétosztása, valamint az élek címkéje megadható ontológiák segítségével.

A fent definiált gráfot ezután a következőképpen reprezentáljuk RDF hármasok segítségével.

36.16. definíció. ***(RDF reprezentáció)*** *Legyen* $G = (V, E, C, cimke())$ *egy közösségi hálózat adatmodellje. Ekkor*

1. *Minden* a *aktorhoz rendelünk egy egyértelmű* id_a *azonosítót.*

2. *Minden* a *aktorhoz hozzárendeljük a típusát, vagyis egy* $(id_a, tipusa, tipus_a)$ *hármast.* $H_T = \{(id_a, \text{'tipusa'}, tipus_a) \mid a \in A\}$.

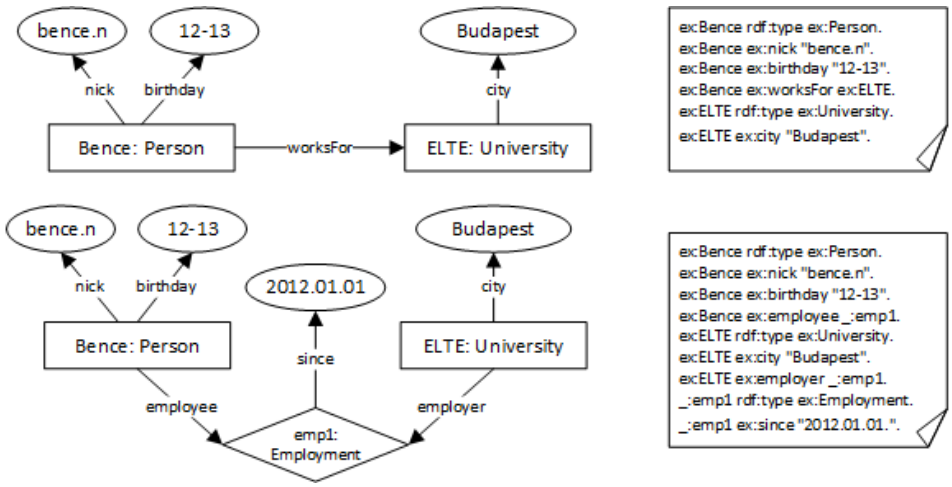
3. *Minden* $e = (a, b) \in E_{AA}$ *élhez, ahol* $a, b \in A$, *hozzárendelünk egy* $(id_a, cimke(e), id_b)$ *hármast.* $H_R = \{(id_a, cimke(e), id_b) \mid a, b \in A, (a, b) = e \in E_{AA}\}$.

4. *Minden* $e = (a, m) \in E_{AM}$ *élhez, ahol* $a \in A, m \in M$, *hozzárendelünk egy* $(id_a, cimke(e), m) \in H_M$ *hármast.* $H_M = \{(id_a, cimke(e), m) \mid a \in A, m \in M, (a, m) = e \in E_{AM}\}$.

Így a G ***közösségi hálózat RDF reprezentációja*** *a* $H = H_T \cup H_R \cup H_M$ *halmaz.*

A fent vázolt megközelítésnek az a hátránya, hogy a kapcsolatokhoz nem rendelhetünk attribútumokat. Ez bizonyos esetekben elengedhetetlen, például hogy mikor jött létre a kapcsolat, vagy ha az erősségét szeretnénk mérni. Egy másik hátrány, hogy ebben a szemléletben csak bináris kapcsolatokat lehet kifejezni. A második megközelítésben éppen ezért a kapcsolatokat is csúcsként reprezentáljuk, így azokhoz is rendelhetünk attribútumokat, illetve több aktor is részt vehet egy kapcsolatban. Az élek ebben az esetben a résztvevők szerepét jelölik.

36.17. definíció. ***(közösségi háló adatmodellje 2)*** *A* ***közösségi háló*** *egy* $G = (V, E, C, cimke())$ *élcímkézett irányított gráf, ahol*



36.1. ábra. Közösségi hálók első (felül) és második (alul) adatmodell változata.

1. *$V = A \cup M \cup R$ a csúcsok halmaza, az aktorok A, az attribútumok M és a relációk R diszjunkt halmazának az uniója. Az aktorok, attribútumok és relációk halmaza tovább van particionálva típusok szerint, ahogyan az előző definícióban láttuk.*

2. *$E = E_{AR} \cup E_{AM} \cup E_{RM}$ az élek halmaza, az $E_{AR} \subseteq A \times R$, $E_{AM} \subseteq A \times M$ és $E_{RM} \subseteq R \times M$ diszjunkt halmazoknak az uniója,*

3. *$C = C_{AR} \cup C_{AM} \cup C_{MR}$ az adott éltípushoz tartozó élcímkék halmaza,*

4. *$cimke()$ az élcímkéző függvény, és $cimke(e)$ az e él címkéjével egyenlő.*

A második adatmodell RDF reprezentációja hasonló az első adatmodelléhez. A reprezentáció részleteinek kidolgozását lásd a gyakorlatok között, amihez segítséget nyújt a 36.2. ábra. A felső az első megközelítést, az alsó a másodikat követi. A négyzetek jelölik az aktorokat az azonosítójukkal, az azonosító után kettősponttal elválasztva pedig a típus szerepel. Az ellipszisek az attribútumok. Az alsó hálóban a rombusz a relációt mint csúcsot jelöli. A hálók mellett az RDF reprezentáció látható.

Az előzőekben láthattunk két megoldást arra, hogyan lehet a közösségi hálókat RDF hármasokkal reprezentálni. Az első megoldás, ahol az aktorok a csúcsok és a relációk az élek, egyszerűbb és természetesebb, viszont a második modell, ahol a relációkat is csúcsokként reprezentáljuk, lehetőséget ad n-áris relációk kifejezésére, illetve attribútumok megadására a relációkhoz is. Mivel

a különböző közösségi hálók eltérő célt szolgálnak, előfordulhat, hogy egyiket az első, míg másikat a második modell szerint érdemes tárolni.

36.2.2. Közösségi hálók lekérdezése és transzformálása

A közösségi hálók gyakorlati felhasználása igen sokrétű. Többféle feladatot, lekérdezést szoktak megfogalmazni. A jellemző használati esetek a lekérdezéseket két nagy csoportba sorolják. Az első csoportba tartozó lekérdezések a hálónak, vagy annak egyes részeinek a struktúráját vizsgálják, és általában valamilyen mérőszámmal jellemzik azt. Ilyen lehet például a bevezetőben már említett fokszám eloszlás, átmérő vagy a sűrűség. A másik csoportba az adatok kezeléséért, transzformálásáért felelős lekérdezések tartoznak. Ezek közös jellemzője, hogy a kimenetük is egy háló, amit a bemenet transzformálásával kapunk. Ilyen feladatok jellemzően:

- *Csoportok kiválasztása.* Talán a leggyakoribb feladat, amikor a hálónak egy adott mintára illeszkedő részeit kell kiválogatni.
- *Attribútumok aktorrá alakítása.* Előfordulhat, hogy egy attribútumérték több aktornál is szerepel, esetleg egy attribútumhoz összetett értéket kell rendelni. Ezekben az esetekben szükséges lehet, hogy az attribútum értékét aktorrá alakítsuk, amit új objektumok létrehozásával érhetünk el.
- *Relációk összevonása, csoportosítása.* Gyakran szükséges adott típusú relációk összevonása, számlálása, aktoronkéni csoportosítása, vagyis az aggregálása.
- *Adott aktor teljes környezetének lekérdezése.* Egy adott felhasználó vizsgálatakor ki kell tudni nyerni a felhasználó összes kapcsolatát, a kapcsolatokhoz tartozó további aktorokkal együtt. Ez a feladat általánosítható olyan módon, hogy az adott felhasználó k-sugarú környezetét kérdezzük le.

A következőkben egy olyan nyelvet vizsgálunk meg, mely a fenti feladatok megoldását segíti. A ***Social Network Query and Transformation Language*** (röviden: SNQL) a gyakran előforduló lekérdezéshez szükséges kifejezőerő biztosítása mellett az alacsony adatbonyolultságra törekszik. Az SNQL elsősorban az adattranszformációs műveleteket támogatja. Ennek megfelelően a lekérdezések felépítése CONSTRUCT – WHERE – FROM szerkezetet követ. A FROM záradékban megadott közösségi háló lesz a lekérdezés bemenete, amelyből a WHERE záradékban megadott ***kiválasztási minta*** alapján megszűrt adatokat a CONSTRUCT záradékban megadott ***konstrukciós minta*** szerint szervezzük újra.

Az alábbi példában a 36.2. ábrán láthatóhoz hasonló szerkezetű hálóból a *University* típusú csúcsok *city* attribútumait alakítjuk át aktorokká, és a

második adatmodellt alkalmazva összekapcsoljuk a régi és az új aktorokat a megfelelő kapcslati csúcsokon keresztül.

36.2. példa. Attribútumok aktorrá alakítása

```
CONSTRUCT { (?U1, rdf:type, ex:University), (?U1, ex:locatedIn, ?R1),
            (?C1, ex:contains, ?R1), (?R1, rdf:type, ex:City)}
          IF ?R1=f(?U1,?C1)
WHERE     {(?U1, rdf:type, ex:University), (?U1, ex:city, ?C1)}
FROM      EmploymentNetwork
```

A példán megfigyelhetjük a konstrukciós minta és a kiválasztási minta felépítését, valamint a CONSTRUCT záradék IF feltételében szereplő kifejezést. Az IF kulcsszó után új változókat definiálhatunk a kiválasztási mintában definiált változók segítségével. Az f függvény egyedi azonosítót generál az új reláció csúcsoknak a relációban résztvevő aktorok alapján.

Az aggregációs feladatok megoldásához a nyelv az ***AGG*** kulcsszót biztosítja. Megadhatjuk az aggregálni kívánt változók nevét, az aggregáló függvényt és egy mintát, amely meghatározza az aggregálandó értékeket. Az alábbi példa hozzárendeli az emberekhez új attribútumként, hogy hány vállalatnál dolgoznak. (A példában az első adatmodellt követjük.)

36.3. példa. Csoportosítás és aggregálás

```
CONSTRUCT { (?P1, rdf:type, ex:Person), (?P1, ex:worksFor, ?U1),
            (?P1, ex:numberOfEmployer, ?DB)}
WHERE     AGG({?U1}, COUNT AS ?DB,
            {(?P1, rdf:type, ex:Person), (?P1, ex:worksFor, ?U1)} )
FROM      EmploymentNetwork
```

A következő feladat megoldásához, ahol egy adott csúcs környezetét kell lekérdeznünk tetszőleges mélységig, szükségünk van a tranzitív lezárt kiszámítására. A nyelv ehhez a ***TC*** függvényt vezeti be, melynek három paramétere van, a kezdőpont, a végpont és egy kiválasztási minta, ami tartalmazza ezt a két pontot. Emellett egy kezdőfeltételt kell megadnunk a WITH kulcsszó után. A 36.12. példában, az első adatmodellt használva, olyan embereket keresünk, akik *worksFor* reláción keresztül elérhetők *Bencétől* indulva, és összekötjük őket *Bencével* egy *ex:canReach* címkéjű éllel.

36.4. példa. Tranzitív lezárt

```
CONSTRUCT { (ex:Bence ex:canReach ?P2) }
WHERE     TC(?P1, ?P2,
              {(?P1, rdf:type, ex:Person), (?P1, ex:worksFor, ?U1),
               (?P2, rdf:type, ex:Person), (?P2, ex:worksFor, ?U1)}
               FILTER (?P1 != ?P2))
          WITH ?P1=ex:Bence
FROM      EmploymentNetwork
```

Az előző három példa jól szemlélteti a nyelv eszközkészletét, amellyel a fontosabb feladatok egyszerűen leírhatók. A pontos szintaktikai szabályok megtalálhatók a függelékben. A szintaktikai elemek megismerése után vizsgáljuk meg a nyelv szemantikáját. Legyen H egy közösségi háló és Q egy SNQL kérdés, jelölje $Q(H)$ a Q lekérdezés eredményét a H hálóra alkalmazva.

A lekérdezések kiértékelése két lépésből áll. Először az kiválasztási mintát átalakítjuk Datalog szabályokká, majd a konstrukciós minta alapján sorgeneráló függőségeket képzünk. A kiválasztási minta átalakítása a következő szabályok rekurzív alkalmazásával történik:

1. Minden $t = (s, p, o)$ hármast $t(s, p, o)$-ra fordítunk.
2. Egy $PATT = \{t_1, \ldots, t_n\}$ hármasokból álló mintához a $p(\bar{z}) \leftarrow \bigwedge_{i=1}^{n} t_i(s_i, p_i, o_i)$ szabályt rendeljük, ahol $\bar{z}$ a $PATT$-ban szereplő változókat jelöli.
3. $PATT_1$ AND $PATT_2 : p(\bar{z}) \leftarrow p_1(\bar{x}), p_2(\bar{y})$.
4. $PATT_1$ OR $PATT_2 : p(\bar{z}) \leftarrow p_1(\bar{x})$
   $p(\bar{z}) \leftarrow p_2(\bar{y})$.
5. $PATT_1$ AND-NOT $PATT_2 : p(\bar{z}) \leftarrow p_1(\bar{x}), \neg p_2(\bar{y})$.
6. $PATT_1$ FILTER $C : p(\bar{z}) \leftarrow p_1(\bar{x}), c(\bar{x})$.
7. $TC(u, v, PATT_1)$ WITH <startCondition> :
   $p(u, v) \leftarrow p_1(\ldots u \ldots v \ldots), startCond(\ldots u \ldots v \ldots)$
   $p(u, v) \leftarrow p_1(\ldots u \ldots v \ldots), p(w, v)$
8. $AGG(\bar{v}, agg, PATT_1) : p(\bar{v}, agg(\bar{y}) \leftarrow p_1(\bar{v}, \bar{y})$,
   ahol $\bar{x}$ a $PATT_1$ változóit jelöli, $\bar{v} \subseteq \bar{x}$, $\bar{y} = \bar{x} - \bar{v}$ és agg egy aggregáló függvény.

A második lépésben a konstrukciós minta alapján generálunk szabályokat. Ebben a lépésben felhasználjuk az előző lépés szabályait. Egy

CONSTRUCT trList IF eqList

mintából az eqList alapján $v_i = term_i$ és $term_i = term_l$ alakú kifejezéseket hozunk létre, ahol egy term lehet változó, konstans vagy függvény. Majd a minta és az egyenletek alapján

$$construct(v_1, \ldots, v_n) \leftarrow p(\bar{z}) \wedge \bigwedge_j eq_j$$

alakú szabályt képzünk. Végül az eredményt a trList alapján konstruáljuk a következőképpen:

$$H = \bigcup \{t(u_1, u_2, u_3) : \exists(..u_1..u_2..u_3..) \in construct \text{ and } t \text{ in } trList\}$$

36.5. példa. A 36.10. példában megadott lekérdezés átírása

$p(u_1, c_1)$	$\leftarrow t(u_1, \text{rdf:type}, \text{ex:University}),$ $t(u_1, \text{ex:city}, c_1)$
$construct(u_1, r_1, c_1)$	$\leftarrow ep(u_1, c_1), r_1 = f(u_1, c_1)$
$output(u_1, \text{rdf:type}, \text{ex:University})$	$\leftarrow construct(u_1, r_1, c_1)$
$output(u_1, \text{ex:locatedIn}, r_1)$	$\leftarrow construct(u_1, r_1, c_1)$
$output(c_1, \text{ex:contains}, r_1)$	$\leftarrow construct(u_1, r_1, c_1)$
$output(r_1, \text{rdf:type}, \text{ex:City})$	$\leftarrow construct(u_1, r_1, c_1)$

Az SNQL nyelv kidolgozásakor az egyik fő szempont az volt, hogy a gyakorlatban legtöbbször előforduló adattranszformációs feladatokat ki lehessen vele fejezni. A nyelv tervezésekor a Graphlog lekérdezőnyelvet és a másodrendű sorgeneráló függőségeket vették alapul.

36.18. állítás. *Legyen Q egy SNQL kérdés, H egy közösségi háló és $t = (s, p, o)$ egy RDF-hármas. Ekkor $t \in Q(H)$ eldöntése NLOGSPACE-ben van.*

A bizonyítás, amit most nem közlünk, a Graphlog és a másodrendű sorgeneráló függőségek vizsgálatán alapul.

A fenti példákból látszik, hogy az SNQL nyelv szerkezete és nyelvi elemei hasonlítanak a SPARQL CONSTRUCT típusú lekérdezéséhez. Mindkettő egy kiválasztási minta és egy konstrukciós minta alapján épül fel. Az SNQL annyiban tér el, hogy a kiválasztási mintában nem használhatunk bizonyos elemeket, amiket a SPARQL-ben igen, például az OPT és UNION kulcsszavakat. SNQL-ben az IF konstrukcióval új változókat képezhetünk, továbbá lehetőség van a tranzitív lezárt kiszámítására. Ezeket a funkciókat a SPARQL 1.1-ben pótolták, ahol a WHERE záradékban BIND kulcsszóval értéket rendelhetünk változókhoz, hasonlóan mint az IF konstrukcióban. Az 1.1-es verzióban továbbá útvonal kifejezéseket is definiálhatunk, vagyis megadhatunk egy reguláris kifejezést az élcímkék fölött, melyre a két csúcs közötti úton szereplő élek címkéinek illeszkedniük kell.

36.2.3. Csoportok kiválasztása hatékonyan

A közösségi hálók felhasználásának egyik gyakori feladata adott tulajdonságú csoportok kiválasztása, csoportok közötti kapcsolatok keresése. Egy közösségi háló vizsgálatakor kereshetünk például klikkeket, azaz olyan közösségeket, amelyben minden tag ismeri egymást, összekötőket, melyek összekapcsolnak különböző klikkeket vagy éppen egymástól teljesen független csoportokat is. Az előzőekben láttuk, hogyan lehet a közösségi hálókat RDF hármasokkal reprezentálni és azt, hogy milyen előnyei vannak ennek a rugalmas ábrázolási módnak. Ebben a fejezetben az RDF-hez kidolgozott SPARQL nyelv egy lehetséges kiegészítését vizsgáljuk meg, mely lehetővé teszi a csoportok kezelését. SPARQL-ben alapvetően a gráf csúcsaira és éleire vonatkozó konjunktív lekérdezéseket fogalmazhatunk meg. A javasolt kiegészítésben ezeket a konjunktív lekérdezéseket terjesztjük ki csúcshalmazokra.

36.19. definíció. ***(Reguláris útvonalkérdés)*** *Legyen $H = (V, E)$ egy irányított élcímkézett gráf. Jelöle a $a \xrightarrow{p_1 p_2 \ldots p_n} b$ az a és b csúcs közötti $p_1, \ldots, p_n$ élcímkékkel címkézett utat. Ekkor $Q^R \leftarrow R$ az R reguláris kifejezéssel definiált reguláris útvonalkérdés. Az $ans(Q^R, H)$ válasz azokat a csúcspárokat tartalmazza, amelyek a reguláris kifejezésre illeszkedő útvonallal össze vannak kötve, vagyis:*

$$ans(Q^R, H) = \left\{(a, b) \in V \times V \mid a \xrightarrow{p} b, ahol\ p \in L(R)\right\}$$

$L(R)$ az R reguláris kifejezés által generált nyelvet jelöli.

36.20. definíció. ***(Konjunktív reguláris útvonalkérdés)*** *A konjunktív reguláris útvonalkérdések a következő alakúak:*

$$Q^C(x_1, \ldots, x_n) \rightarrow y_1 R_1 y_2 \wedge \ldots \wedge y_{2m-1} R_m y_{2m},$$

ahol $x_1, \ldots, x_n, y_1, \ldots, y_m$ csúcsváltozók és $\{x_1, \ldots, x_n\} \subseteq \{y_1, \ldots, y_n\}$. Az $ans(Q^C, H)$ válasz azokat a $(v_1, \ldots v_n)$ H-beli csúcsokból képzett n-eseket tartalmazza, amelyekhez létezik olyan σ leképezés, melyre $\sigma(x_i) = v_i$, és $(\sigma(y_i), \sigma(y_{i+1})) \in ans(Q^R, H)$ minden $y_i R_i y_{i+1}$ term által definiált Q^R reguláris útvonalkérdésre.

Ahhoz, hogy a fenti definíciókat ki lehessen terjeszteni halmazokra, szükséges, hogy a halmazok bizonyos részeit ki tudjuk jelölni. Ehhez megadjuk a $\forall$ és a $\exists$ kvantoroknak a halmazokra vonatkozó kiterjesztését.

- Univerzális kvantor: $\forall_M = \{M\}$
- Egzisztenciális kvantor: $\exists_M = \{A \subseteq M \mid A \neq \emptyset\}$

- Számossági kvantor: $\exists_M(\odot n) = \{A \subseteq M \mid |A| \odot n\}$, ahol $\odot \in \{>, \geq, =, \leq, <\}$

A továbbiakban a görög Ξ vagy Ψ betűket úgy használjuk, hogy helyükre a fenti kvantorok valamelyike behelyettesíthető.

36.21. definíció. ***(Halmaz-csúcs reguláris útvonalkérdések)*** *$Q^{\Xi\bullet} \leftarrow R$ egy halmaz és egy csúcs közötti reguláris útvonalkérdés, ahol a Ξ kvantor meghatározza, hogy a halmaz hány elemének kell kapcsolódnia a $\bullet$-al jelölt csúcshoz úgy, hogy az útvonalon szereplő élek címkéje illeszkedjen az R reguláris kifejezésre. A válasz ennek megfelelően rendezett párokból áll, melynek első eleme egy halmaz, a második eleme pedig egy csúcs, vagyis:*

$$ans(Q^{\Xi\bullet}, H) = \left\{(A, b) \in 2^V \times V \mid a \xrightarrow{p} b, ahol\ a \in \Xi_A, p \in L(R)\right\}$$

A fordított irány, azaz egy csúcs és egy halmaz közötti reguláris útvonalkérdések, illetve a halmaz és halmaz közötti kérdések hasonlóan definiálhatóak.

36.22. definíció. ***(Halmaz-csúcs reguláris útvonalkérdés lezártja)*** *Legyen $Q^{\Xi\bullet}$ egy halmaz-csúcs reguláris útvonalkérdés, ekkor a **lezártja,** $\bar{Q}^{\Xi\bullet}$, tovább szűkíti az eredményhalmazt úgy, hogy csak azokat az A és b közötti útvonalakat engedi meg, amelyek A-ban maradnak, tehát*

$$ans(\bar{Q}^{\Xi\bullet}, H) = \{\ (A, b) \in 2^V \times V \mid a \xrightarrow{p} b, ahol\ a \in \Xi_A,\ p_1, \ldots, p_n = p \in L(R) \\ és\ \forall i \in \{1, \ldots, n-1\}\ a \xrightarrow{p_1 \ldots p_i} c => c \in A\}$$

36.23. definíció. ***(Halmaz méretét megszorító kérdés)*** *A $Q^{|.|} \leftarrow (from, to)$ kérdés egy unáris relációt ír le, ahol $from, to \in \mathbb{N}, from \leq to$ a halmaz méretének a minimumát és maximumát definiálja. A válasz azokból a halmazokból áll, melyek mérete megfelel a feltételeknek:*

$$ans(Q^{|.|}, H) = \left\{A \in 2^V \mid |A| \in \{from, \ldots, to\}\right\}$$

A fent definiált kérdéstípusokból konjunkcióval összetett kérdéseket lehet alkotni, hasonlóan a konjunktív reguláris útvonalkérdésekhez. A pontos definíció a következő.

36.24. definíció. ***(Konjunktív reguláris útvonalkérdés halmazokkal)*** *A halmazokat is megengedő konjunktív reguláris útvonalkérdés alakja*

$$Q^S(x_1, \ldots, x_n) \leftarrow \tilde{y}_1[(R_1)^{\Psi_1^1 \Psi_2^1}]y_2 \wedge \ldots \wedge \tilde{y}_{2m-1}[(R_m)^{\Psi_1^m \Psi_2^m}]y_2 m \\ \wedge Z_1[f_1, t_1] \wedge \ldots \wedge Z_l[f_l, t_l],$$

ahol $x_1, \ldots, x_n, y_1, \ldots, y_m$ *csúcs- vagy halmazváltozók és* $\{x_1, \ldots, x_n\} \subseteq \{y_1, \ldots, y_m\}$. $Z = \{Z_1, \ldots, Z_l\}$ *az* y_i*-k közötti összes halmazváltozó halmaza.* $A \sim$ *szimbólum vagy üres, vagy halmazok esetén lehet –, ami a lezártat jelenti.* $A\ \Psi$ *vagy egy kvantor, vagy a* $\bullet$ *szimbólum, amely egy egyszerű csúcsot jelöl. Az* R_i*-k reguláris kifejezések.*

Az $ans(Q^S, H)$ *azokat a* $(v_1, \ldots, v_n)$ *H-beli csúcsokból és halmazokból képzett rendezett n-eseket tartalmazza, melyekhez létezik olyan* σ *totális leképezés, hogy* $\sigma(x_i) = v_i$ *és* $(\sigma(y_i), \sigma(y_{i+1}))$ *szerepel a megfelelő részkifejezés eredményhalmazában.*

36.6. példa. Konjunktív reguláris útvonalkérdés halmazokkal

A példában egy olyan adathalmazt kérdezünk le, ahol a csúcsok *knows* címkéjű élekkel vannak összekötve, és feltesszük, hogy ez a reláció reflexív. Ekkor olyan (A, B) halmazpárokat keresünk, melyekre a következő feltételek teljesülnek:

1. Az A halmazban mindenki legfeljebb kettő hosszú úttal össze van kötve Bencével, akit az *ex:Bence* azonosító jelöl, és ezek az utak nem vezetnek ki A-ból.
2. A B halmazban mindenki ismer mindenkit.
3. A B halmazból mindenki ismer legalább három A halmazbeli személyt.
4. Az A halmaz hat, a B halmaz legalább négy, de legfeljebb öt elemű.

Ezeket a feltételeket fejezi ki az alábbi kérdés:

$$
\begin{aligned}
Q^S(A,B) \leftarrow\ & \\
& \bar{A}[(knows.knows?)^{\forall\bullet}]ex\!:\!Bence \\
\wedge\ & B[(knows)^{\forall\forall}]B \\
\wedge\ & B[(knows)^{\forall\exists(\geq 3)}]A \\
\wedge\ & A[6,6] \wedge B[4,5]
\end{aligned}
$$

Amit a következőképpen írhatunk át SPARQL lekérdezéssé:

```
SELECT ??A ??B
WHERE{
        ALL CLOSURE ??A knows\knows? ex:Bence.
        ALL ??B knows ALL ??B.
        ALL ??B knows SOME(≥ 3) ??A.
        FILTER ( ??A{6, 6}, ??B{4, 5})}
```

Az átírt lekérdezésből látszik, hogy néhány nyelvi elem bevezetésével hogyan fejezhetők ki a halmazokat is megengedő konjunktív reguláris útvonalkérdések. A halmaz változókat a ?? dupla kérdőjel jelzi. A reguláris kifejezésben szereplő, halmazokra vonatkozó kvantorok közül az univerzális kvantort az ALL kulcsszóval, az egzisztenciálisat a SOME-mal, míg a számossági kvantort szintén a SOME kulcsszóval jelöljük, amely után zárójelben feltüntettük a megszorítást. A halmaz lezárását a CLOSURE jelzi. Továbbá, a halmazok elemszámára vonatkozó feltételeket a FILTER után írjuk a fent látható alakban.

Az átírás után vizsgáljuk meg a halmazokat is megengedő reguláris útvonalkérdések bonyolultságát.

36.25. állítás. *Legyen Q egy kérdés, H egy N csúcsú élcímkézett háló, $\bar{v}$ csúcsváltozók és $\bar{V}$ halmazváltozók rendezett n-ese. Ekkor $(\bar{v}, \bar{V}) \in Q(H)$ eldöntése PTIME-ban van.*

Az állítás bizonyításhoz elég látni, hogy a csúcs- és halmazváltozók – jelöljük őket c-vel és C-vel – száma rögzített a kérdésben. Továbbá, minden halmaz alulról és felülről korlátos a megfelelő megszorító kérdés miatt. Legyen K a felső korlátok maximuma. Ekkor a kérdés megválaszolásához legfeljebb N^{c+K*C} esetet kell ellenőrizni, ami a háló méretében polinomiális.

Függelék: az SNQL nyelv szintaxisa

```
<query>                  ::= CONSTRUCT <construct-pattern>
                             WHERE <ext-patt>
                             FROM  <list-of-networks>
<construct-pattern>      ::= <list-of-pattern-triples>[IF <list-of-eqs>]
<list-of-eqs>            ::= <var>= <expr>[ AND <var>= <expr>]*
<ext-patt>               ::= <list-of-pattern-triples>
                           | (<ext-patt>) AND (<ext-patt>)
                           | (<ext-patt>) OR (<ext-patt>)
                           | (<ext-patt>) AND-NOT (<ext-patt>)
                           | (<ext-patt>) FILTER (<condition>)
                           | TC(<var>, <var>, <ext-patt>)
                             WITH <condition>
                           | AGG({<var>[, <var>]*},
                                 <agg-func>, <ext-patt>)
<list-of-pattern-triples> ::= {<pattern-triple>[, (<pattern-triple>)]*}
<pattern-triple>         ::= (<term>, <term>, <term>)
<list-of-networks>       ::= <network>[, <network>]*]
<network>                ::= <network-id>  |  <list-of-instance-triple>
```

<list-of-instance-triple> ::= {<instance-triple>[, <instance-triple>]*}
<instance-triple> ::= (<constant>, <constant>, <constant>)
<expr> ::= <term> | <func>
<term> ::= <var> | <constant>
<func> ::= func(<expr>[, <expr>]*)
<agg-func> ::= <*literal*>
<condition> ::= <*logical-expression*>
<var> ::= ?<*literal*> | $<*literal*>
<constant> ::= <*literal*>

Gyakorlatok

36.2-1. Írjuk fel a 36.42. definícióban leírt adatmodellhez tartozó RDF reprezentáció formális definícióját.

36.2-2. A 36.46. definíció alapján írjuk fel a csúcs-halmaz és halmaz-halmaz reguláris útvonal kérdések és a hozzájuk tartozó válaszok formális definícióját.

36.2.4. Közösségi hálózatok lekérdezése MapReduce módszerrel

A szemantikusan leírt közösségi adatok lekérdezésének legegyszerűbb MapReduce szemléletű módszere alaptáblák előállítására épül. A táblák azokat a hármasokat tartalmazzák, amelyek a lekérdezés egy adott feltételére teljesülnek. Ezt a szakaszt nevezzük SELECTION résznek. A SELECTION szakasz végigfut az adatokon, és kiválasztja azokat az értékeket, amelyek megfelelnek egy adott változónak.

36.7. példa. Az alábbi SPARQL lekérdezés a Budapest Bár iránt érdeklődő felhasználóknak azokat a barátait adja vissza, akik Budapesten laknak és akikkel egy 2013. május 2–nál újabb képen mindketten meg vannak jelölve.

```
SELECT ?Friend WHERE {
        ?User hasInterest "Budapest Bár" .
        ?Friend locatedAt "Budapest" .
        ?Photo laterThan "2013-05-02" .
        ?User friendOf ?Friend .
        ?Photo taggedBy ?User .
        ?Photo taggedBy ?Friend
}
```

A példában egy lekérdezés feltételeit láthatjuk. Ez egy tipikus lekérdezése a közösségi hálóknak, ami a célzott reklámok kiválasztására szolgál. A

lekérdezésben vesszük azokat a felhasználókat, akik egy bizonyos területről vannak és egy közeli ismerősük érdeklődik egy bizonyos termék iránt. A kiválasztási szakaszban ekkor hat alaptábla fog elkészülni: hasInterest(?User), locatedAt(?Friend), laterThan(?Photo), friendOf(?User?Friend), taggedBy(?User?Photo) és a taggedBy(?Friend?Photo). Ezeket a táblákat egy join szakasz fogja feldolgozni. Ez a megoldás nem hatékony, mivel vannak benne felesleges MapReduce job-ok. Ennek oka, hogy a hagyományos SQL join operátor és a SPARQL lekérdezés kiértékelése különbözik. A join esetében az egyes összekapcsolásoknál az alaptáblák száma korlátozott, emiatt több jobra van szükségünk.

SELECTION rész mellett a join rész sem hatékony. A jelenlegi megoldás a kiválasztásokat MapReduce folyamatokkal oldja meg, és az elkészült alaptáblákat használja későbbiekben az összekapcsolásra. Ennél egy hatékonyabb megoldás, ha kihasználjuk a MapReduce rugalmasságát, és a SELECTION részt beolvasztjuk a join részbe, ezzel csökkentve a felesleges jobok számát.

Többszörös összekapcsolás

Legyen adott egy SPARQL lekérdezésünk a következő négy alaptáblával: t1(?X?Y), t2(?X?Y), t3(?X?Z), t4(?X?Z). Azonos sémájú alaptáblák a különböző kapcsolatok miatt lehetségesek. Az SQL alapú megoldás előbb kiértékeli az ?X?Y és ?X?Z táblákat, majd összekapcsolja az eredményeket ?X alapján. Minden join művelethez egy MapReduce jobra van szükség. Hatékonyabb, ha összekapcsoljuk az összes alaptáblát ?X alapján, és szűrjük párhuzamosan ?Y és ?Z változókra. Ehhez a megoldáshoz mindössze csak egy jobra van szükségünk. Ezt a megoldást nevezzük multiple-join-with-filter metódusnak.

A művelet szintaxisa a következő:

Multiple join [tables] on [join Key]
Filter on [Filter Key]
Dump Result to [MT]

Vegyük a korábbi 36.15-es példánkat. Miután összekapcsoltuk a hasInterest(?User), friendOf(?User?Friend) és a taggedBy(?UserPhoto) táblákat a ?User alapján, a következő lépés az így kapott ?User?Friend?Photo tábla összekapcsolása a ?Friend és a ?Friend?Photo táblával a ?Friend változón keresztül. A multiple-join-with-filter művelettel ez a következőképpen néz ki:

Multiple join [?Friend, ?Friend?Photo,
?User?Friend?Photo] on [?Friend]
Filter on [?Photo]
Dump Result to [?User?Photo]

Ebben a példában a ?Friend változót használtuk join kulcsnak. Ezzel egy időben a ?Photo-ra kapott különböző értékeket leszűrjük a ?User?Friend?Photo és a ?Friend?Photo alapján. A multiple-join-with-filter a ?Photo változót használja szűrő kulcsnak.

Kiválasztó összekapcsolás

A rugalmas MapReduce programozás megengedi, hogy egyszerre generáljunk alaptáblákat és végezzünk join műveletet. Ennek eredményeként nem kell annyi job a kiválasztási szakaszban. Ezt a hibrid réteget nevezzük primitív select-join műveletnek, ahol a kiválasztás be lett integrálva az összekapcsolásba. A művelet során két csoportot alkalmazunk, ahol egy csoport hármasokat tartalmazó halmaz. Az első csoport a kiválasztó csoport, mely az alaptáblákat generálja. A másik csoport az összekapcsolási csoport, amely a join műveleteket végzi. A select-join szintaxisa a következő:

Select on [triplets A]
Multiple join [triplets B] on [join Key]
Filter on [Filter Key]
Dump Result to [ST]

A select-join működését az alábbi példa szemlélteti.

36.8. példa. Az alábbi SPARQL lekérdezés olyan hozzászólásokat ad vissza, melyek 2012. január 1-nél újabb keltezésűek, és azokat sorba rendezi a rájuk vonatkozó megjelölések száma szerint csökkenően.

```
SELECT ?Post WHERE {
        ?Post rdf:type Post .
        ?Post laterThan "2012-01-01" .
        ?Post tag ?Tag .
} GROUP BY ?Tag
ORDER BY DESC (count(?Post))
```

A lekérdezés a legtöbb post-tal rendelkező aktív témákat kérdezi le 2012 óta. Mivel csak egy megosztott változó van (?Post), ezért elég csak egy összekapcsolás ahhoz, hogy megkapjuk az értékét. Ez megoldható egyetlen MapReduce jobbal, ha a kiválasztást az összekapcsolással együtt végezzük el. Az összes feltétel az összekapcsolási csoportba kerül és a kiválasztási csoportba nem kerül semmi, vagyis nincs szükségünk alaptábla generálásra. Kijelenthetjük, hogy minden olyan SPARQL lekérdezés, amelyben egy változó van megosztva, lefuttatható egy MapReduce jobbal,

amely teljesen kihagyja a köztes adatokat. Ezek a lekérdezések jellemzőek a közösségi hálók lekérdezésében, ezeket a lekérdezéseket nevezzük ***csillag alakú lekérdezéseknek***.

Lekérdezésterv generálás

Az algoritmus egymás után dolgozza fel a hármasokban lévő változókat. Minden lépésben mindig a leggyakrabban megosztott változó lesz a join kulcs. Az algoritmus indít egy SQL join-t vagy egy multiple-join-with-filter jobot a változóra. A kiválasztási rész mindig benne van az első joinban. A multiple-join-with-filter függ attól, hogy van-e érvényes szűrő kulcs. Ha megszámoljuk a nem join kulcsváltozókat és találunk olyan változót, amely egynél többször szerepel a hármasokban, akkor azt használhatjuk szűrő kulcsnak. Ha két változó számossága megegyezik, akkor az algoritmus önkényesen választ a két változó közül. Az utolsó eredmény generálása után az algoritmus visszaadja a MapReduce jobokból álló lekérdezési tervet.

Input. Egy SPARQL lekérdezés Q (hármasok halmaza).
Output. MapReduce folyamat W.
MapReduceWorkflowGeneration

```
 1 v = mostFrequentVariable(Q)
 2 tp1 = triplets in Q not sharing v
 3 tp2 = triplets in Q sharing v
 4 fk = v-n kívüli változók amelyek egynél több tp2-beli hármasban
                    szerepelnek
 5 job ="Select on"+ tp1 +"Multiple join"
                    + tp2 +"join on" + v
                    + "Filter on" + fk + "Dump Result to" + ST
 6 W.append(job)
 7 while ST.size() > 1 do
 8                  VA = correspondingVariables(ST)
 9                  v = mostFrequentVariable(VA)
10                  tb = tables in ST sharing v
11                  fk = v-n kívüli változók amelyek egynél több tb-beli
                                táblában szerepelnek
12                  job = "Multiple join" + tb + "join on" + v +
                    "Filter on" + fk + "Dump Result to" + MT
13                  W.append(job)
14                  remove tb from ST
15                  add MT to ST
16 return W
```

Gyakorlatok

36.2-1. Generáljunk lekérdezési tervet a MapReduceWorkflowGeneration algoritmussal az alábbi lekérdezésre.

```
1 SELECT ?X, ?Y1, ?Y2, ?Y3
2 WHERE {
3       ?X rdf:type ub:Professor .
4       ?X ub:worksFor <http://www.Department0.University0.edu> .
5       ?X ub:name ?Y1 .
6       ?X ub:emailAddress ?Y2 .
7       ?X ub:telephone ?Y3 .
8 }
```

36.2-2. Generáljunk lekérdezési tervet a MapReduceWorkflowGeneration algoritmussal az alábbi lekérdezésre.

```
1 SELECT ?X, ?Y, ?Z
2 WHERE {
3       ?X rdf:type ub:Student .
4       ?Y rdf:type ub:Department .
5       ?X ub:memberOf ?Y .
6       ?Y ub:subOrganizationOf <http://www.University0.edu> .
7       ?X ub:emailAddress ?Z
8 }
```

36.2-3. Generáljunk lekérdezési tervet a MapReduceWorkflowGeneration algoritmussal az alábbi lekérdezésre.

```
1 SELECT ?X, ?Y, ?Z
2 WHERE {
3       ?X rdf:type ub:Student .
4       ?Y rdf:type ub:Faculty .
5       ?Z rdf:type ub:Course .
6       ?X ub:advisor ?Y .
7       ?Y ub:teacherOf ?Z .
8       ?X ub:takesCourse ?Z
9 }
```

Feladatok

36-1 Költségbecslés

Próbáljunk meg aszimptotikus felső becslést adni a GenerateBestPlan algoritmus futási idejére, a hármas-minta gráf csúcsainak és éleinek a függvé-

nyében.

36-2 Lekérdezés bonyolultság
Legyen H egy rögzített n csúcsú élcímkézett közösségi háló. Adjuk meg a $(\bar{v}, \bar{(V)}) \in Q(H)$ döntési probléma bonyolultsági osztályát adott Q, halmazokat is megengedő konjunktív reguláris útvonal kérdésre (36.49. definíció), ahol $\bar{v}$ csúcsváltozók és $\bar{V}$ halmazváltozók rendezett n-ese.

Megjegyzések a fejezethez

A szemantikus web elméleti és gyakorlati problémáiról az első magyar nyelvű tankönyveket Gottdank Tibor [153] és Szeredi Péter [335] írta. A SPARQL lekérdezések MapReduce [91] technikával történő kiértékelésének megoldását írják le Husain és társai [188]. A cikkben szó esik az adatok tárolásáról és a lekérdezéshez szükséges jobok előállításáról. Bemutatnak egy algoritmust, amely elkészíti a lekérdezés kiértékeléséhez szükséges futási tervet.

A közösségi hálózatok RDF reprezentálásával főként a [316] cikk foglalkozik, amely a reprezentálás bevezetése mellett egy lekérdezőnyelvet is bemutat, amelyből a szerzők később megalkották a [317] cikkben bemutatott SNQL nyelvet a hálók lekérdezésére, transzformálására. A nyelv a Graphlog [81] és a másodrendű sorgeneráló függőségek [121] előnyeit egyesíti. A közösségi hálózatok elemzésének egyik alapja a felhasználói csoportok vizsgálata. A SPARQL-ben [173] a lekérdezések alapja a pontokra és a közöttük lévő élekre vonatkozó mintaillesztés, amit a [231] cikkben leírt módon kiegészíthetünk felhasználói csoportok és a csoportok közötti összeköttetések megadásával.

A fentebb említett MapReduce technika közösségi hálózatok lekérdezésére történő alkalmazását Liu, Yin és Gao [233] mutatták be. Megoldásuk a relációs táblák összekapcsolásán alapszik.

A szemantikus web technológiák számos további gyakorlati alkalmazásban kerülnek felhasználásra. Matuszka, Gombos és Kiss [251] cikkükben egy beltéri navigációs rendszert ismertetnek, mely a lehetséges útvonalak kiszámításához RDF reprezentációt és következtetéseket használ fel. Gombos és társai egy általános célú virtuális obszervatóriumot ismertetnek szemantikus adatbázisok felett [151]. Zsigmondi és Kiss a [388]-ben bevezették a szemantikus helyettesítő karakter fogalmát, amellyel általánosítható a szöveges keresés. Matuszka a szemantikus web kiterjesztett valósággal kapcsolatos alkalmazásáról számolt be a [251] publikációban.

Jelen kutatást a FuturICT.hu nevű, TÁMOP-4.2.2.C-11/1/KONV-2012-0013 azonosítószámú projekt támogatta az Európai Unió és az Európai Szociális Alap társfinanszírozása mellett.

Bemeneti fájl kiválasztása

A szükséges fájlok meghatározásához meg kell vizsgálnunk a felhasználó által írt lekérdezést. A WHERE záradékban található hármasok alapján választjuk ki a megfelelő fájlokat. Ha valamely hármasban az állítmány egy változó, akkor minden fájlt kiválasztunk és befejeződik a futás. Ha nincs olyan hármas, amelyben az állítmány egy változó, akkor a tárgyban szereplő elemeket vizsgáljuk tovább. Ha valamely tárgyban változó szerepel, akkor minden olyan fájlt kiválasztunk, amely az adott állítmányhoz tartozik. Azonban, ha a tárgy nem változó, akkor a tárgy típusa tovább szűkíti a szükséges fájlok halmazát, amennyiben a második megközelítés alapján osztottuk el az adatainkat.

36.2.5. GenerateBestPlan algoritmus

A következő részben bemutatunk egy algoritmust, ami előállítja a lekérdezési terveket egy adott lekérdezéshez. Mielőtt rátérnénk a lekérdezés kiértékelésének költségszámítására, bevezetjük a fontosabb definíciókat, melyekre a későbbiekben hivatkozni fogunk. A definíciók megértéséhez nézzük az alábbi lekérdezést.

36.9. példa. Az alábbi SPARQL lekérdezés a <http://www.University0.edu> IRI-vel reprezentált egyetem tanszékeit és azok vezetőit adja eredményül. Az rdf:type megmondja egy erőforrásról, hogy mely osztályba tartozik.

```
1 SELECT ?X, ?Y WHERE {
2       ?X rdf:type ub:Chair .
3       ?Y rdf:type ub:Department .
4       ?X ub:worksFor ?Y .
5       ?Y ub:subOrganizationOf <http://www.University0.edu>
6 }
```

36.26. definíció. (Hármas-minta, TP)
*A **hármas-minta** alany, állítmány és tárgy elemek olyan rendezett hármasa, amelyek a SPARQL lekérdezés WHERE záradékában szerepelnek. Az elemek lehetnek változók vagy konstansok.*

A példában a 2., 3., 4., és 5. sorok egy-egy TP-t tartalmaznak.

36.27. definíció. (MapReduce join, MRJ)
*A **MapReduce** join kettő vagy több hármas minta egy változón keresztül történő összekapcsolása.*

A 2. és a 4. sorban szereplő TP-k az ?X változón, a 3. és az 5. sorban szereplők az ?Y változón keresztül vannak összekapcsolva, ezek tehát MRJ-k. Ugyanígy a 3. és a 4., valamint a 4. és az 5. sorban szereplők. Továbbá a 3., 4. és 5. sor TP-i egy három tagú MRJ-t alkotnak.

36.28. definíció. (Konfliktusos MapReduce join, CMRJ)
Két MapReduce joint konfliktusosnak nevezünk, ha van közös változójuk, de az összekapcsolási feltételben különböző változók szerepelnek.

36.29. definíció. (Nem-konfliktusos MapReduce join, NCMRJ)
Ha nincs közös változójuk, vagy a közös változó szerepel az összekapcsolási feltételben, akkor ***nem-konfliktusos MapReduce joinokról*** *beszélünk.*

Ez alapján a 2. és 4. illetve a 4. és 5. sorban szereplő TP-kből képzett MapReduce joinok konfliktusosak, hiszen a 2. és 4. az ?X-en keresztül, míg a 4. és 5. az ?Y-on keresztül kapcsolódik, de az ?Y egy közös változó. Ha ugyancsak a 2. és 4. sort nézzük, de most a 3. és 5. sorral, akkor nem-konfliktusos MapReduce joint kapunk, hiszen nincs közös változójuk.

36.30. definíció. (job, JB)
Azt a folyamatot, amelyben egy vagy több MapReduce join található, ***jobnak*** *nevezzük. A job bemenetként fájlokat kap és kimenetként fájlokat készít.*

A jobokat tovább csoportosíthatjuk aszerint, hogy a bennük szereplő MapReduce joinoknak vannak-e közös hármas-mintáik. Ez alapján megkülönböztetünk megvalósítható és nem-megvalósítható jobokat. A közös hármasminták gondot okoznak, mert ilyen esetben nem egyértelmű, hogy a Map fázis során milyen kulcsot rendeljünk egy adott hármashoz.

36.31. definíció. (Megvalósítható job, FJ)
Egy jobot ***megvalósíthatónak*** *nevezünk, ha nincs egynél több MapReduce joinban szereplő hármas-mintája.*

36.32. definíció. (Nem-megvalósítható job, IFJ)
Egy jobot ***nem-megvalósíthatónak*** *nevezünk, ha van olyan hármas-mintája, amely több MapReduce joinban szerepel.*

Egy lekérdezés megválaszolásához több Hadoop job egymás utáni végrehajtása szükséges. A jobok sorozatát a lekérdezési terv határozza meg. A lekérdezési terveket szintén csoportosíthatjuk megvalósíthatóság szempontjából, a következő definícióknak megfelelően.

36.33. definíció. (Lekérdezés Terv, QP)
Egy ***SPARQL lekérdezéshez tartozó lekérdezési terv*** *Hadoop jobok olyan sorozata, amely helyesen megválaszolja a SPARQL lekérdezést.*

36.34. definíció. (Megvalósítható Lekérdezési Terv, FQP)
Egy lekérdezési tervet ***megvalósíthatónak*** *nevezünk, ha minden benne szereplő job megvalósítható.*

36.35. definíció. (Nem-megvalósítható Lekérdezési Terv, IQP)
*A **nem-megvalósítható lekérdezési terv** olyan lekérdezési terv, amelyben van legalább egy olyan job, amely nem-megvalósítható.*

Ahhoz, hogy egy lekérdezéshez előállítsuk az összes lehetséges megvalósítható lekérdezési tervet, olyan jobokat kell képezni, melyek megvalósíthatóak és meg kell határozni azok sorrendjét. Ehhez a következő, hármas mintákból képzett gráfok nyújtanak segítséget.

36.36. definíció. (Hármas-minta Gráf, TPG)
*A **hármas-minta gráf** $G_{TP} = (V_{TP}, E_{TP})$ egy irányítatlan élcímkézett gráf, amelynek csúcsai a lekérdezésben szereplő hármas-minták. Két csúcs között pontosan akkor vezet él, ha van közös változójuk, és ekkor az adott él a közös változók halmazával címkézett. Azaz egy $(u, v, c) \in E_{TP}$ él esetén $u, v \in V_{TP}$ csúcsok és $c = var(u) \cap var(v)$ él, ahol a var a TP-ben szereplő változók halmaza.*

Ez a hármas-minta gráf lesz az alapja a join gráfnak, amely alapján össze tudjuk állítani a megfelelő jobokat.

36.37. definíció. (join Gráf, JG)
*A G_{TP} hármas-minta gráfból készített $G_J = (V_J, E_J)$ **join gráf** egy irányítatlan élcímkézett gráf, amelynek minden $v \in V_J$ csúcsa egy $(u_{TP}, v_{TP}, c_{TP}) \in E_{TP}$ élt reprezentál, és az $u_J, v_J \in V_J$ csúcsok között pontosan akkor vezet él, ha a megfelelő élek a G_{TP} gráfban szomszédosak, és az élcímkékben szereplő változóhalmazok diszjunktak.*

A jobok összeállításánál a célunk olyan MapReduce joinok összeválogatása, melyek páronként nem-konfliktusosak. Továbbá a lekérdezési tervekben szereplő jobok függhetnek egymástól, az ilyen jobok futtatása sorban történik. A helyes sorrend a join gráf színezésével állítható elő a következő módon. Az alábbi algoritmus előállítja a lehetséges színezéseket és a hozzájuk tartozó lekérdezési terveket. A lekérdezési terveknek különböző szempontok alapján lehet értékelni a hatékonyságát, majd kiválasztani közülük a leghatékonyabbat. Ilyen szempontok lehetnek például a jobok száma, a jobok bemenetének vagy kimenetének a mérete.

Input. Query q.
Output. plan
GenerateBestPlan

```
1 plans = CreateEmptyPriorityQueue()
2 tpg = CreateTriplePatternGraph(q)
```

```
3 jg = CreatejoinGraph(tpg)
4 jobs = {}
5 ColorGraph1(tpg,jg,0,jobs)
6 return getHeadFromPriorityQueue(plans)
```

A színezés pontonként történik a ColorGraph1 meghívásával, amely kezdetben megvizsgálja, hogy van-e az adott csúcsnak fehér színű szomszédja, ha nincs, akkor a fehérrel meghívja a ColorGraph2 metódust, mely a tényleges színezést végzi. Majd a szomszédok színétől függetlenül ugyanarra a csúcsra meghívja a ColorGraph2-t fekete színnel is.

Input. TPG tpg, JG jg, int i, Set jobs.
Output.
ColorGraph1

```
1 if !neighborHasColor(jg, i, WHITE) then
2       ColorGraph2(tpg, jg, i, WHITE, jobs)
3 ColorGraph2(tpg, jg, i, BLACK, jobs)
```

A ColorGraph2 algoritmus végzi a tényleges színezést. Az algoritmus megőrzi az adott csúcs régi színét, majd ellenőrzi, hogy ez lesz-e az utolsó színezés, ugyanis legfeljebb annyi jobunk lehet, ahány csúcsa van a join gráfnak. Ha nem, akkor rekurzívan meghívjuk a ColorGraph1 algoritmust, különben elkészítjük a jobot. Ezután ellenőrizzük, hogy a készülő job szerepele már a jobok halmazában. Ha nem, akkor hozzávesszük. Ezután megnézzük, hogy az összes join elvégezhető-e az eltárolt jobok segítségével, ha igen, akkor elkészítjük a lekérdezési tervet, különben újraépítjük a gráfokat a megfelelő TP-k összevonása után és rekurzívan meghívjuk a ColorGraph1-et. Az algoritmus végén visszaállítjuk a csúcs eredeti színét. Erre azért van szükség, hogy egy visszalépéses kereséssel az összes lekérdezési tervet elő tudjuk állítani.

Input. TPG tpg, JG jg, int i, Color color, Set jobs.
Output.
ColorGraph2

```
1 v = Vj[i]
2 prev_color = color[v]
3 color[v] = color
4 if i < |vertices[jg]-1| then
5       ColorGraph1(tpg,jg,i+1, jobs)
6 else
```

```
 7       job = createjob(vertices[jg])
 8       if !detectDuplicatejob(job, tpg) then
 9           jobs = jobs ∪ job
10           if |joins[job]| == |vertices[jg]| then
11               enqueuePlan(plans, createNewPlan(jobs))
12           else
13               tpg_new = mergejoinedTriplePatterns(tpg, job)
14               jg_new = CreatejoinGraph(tpg_new)
15               ColorGraph1(tpg_new, jg_new, 0, jobs)
16           jobs = jobs\{job}
17 color[v] = prev_color
```

36.38. tétel. *A* GenerateBestPlan *nem állít elő megvalósíthatatlan tervet.*

Bizonyítás. Tegyük fel, hogy az algoritmus előállít megvalósíthatatlan tervet. A nem-megvalósítható lekérdezési terv definíciójából adódik, hogy ekkor lennie kell legalább egy megvalósíthatatlan jobnak, ahol a hármasok úgy vannak kiválasztva, hogy azok közül legalább egy több joinban is szerepel. Legyen $v \in V_{TP}$ egy csúcs a G_{TP}-ben, amely több joinban is szerepel. Az $e_1, e_2 \in E_{TP}$ élek legyenek a v-ben szomszédosak úgy, hogy az élcímkén szereplő változók halmaza diszjunkt. Az e_1, e_2 éleknek feleljenek meg a $v_1, v_2 \in V_J$ csúcsok a G_J-ben. Ahhoz, hogy ez a két csúcs egyszerre fusson le egy adott jobban, mindkettőnek fehérnek kell lennie, ami a színezés miatt nem lehetséges. Ugyanis ha egy csúcsnak van fehér szomszédja, akkor azt már nem színezhetjük fehérre. ■

36.39. tétel. GenerateBestPlan *előállítja az összes megvalósítható lekérdezést.*

Bizonyítás. Tegyük fel, hogy az algoritmus nem állít elő egy megvalósítható lekérdezési tervet. Ez más megfogalmazásban annyit jelent, hogy nem állít elő legalább egy megvalósítható jobot, azaz valamely érvényes színezésű join gráfot. Legyen $C = \{c_1, c_2, \ldots, c_n\}$ egy ilyen színezés, ahol $c_i \in \{BLACK, WHITE\}$ a megfelelő $v_i \in V_J$ csúcs színe, és $n = |V_J|$. Tegyük fel, hogy $\{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ csúcsok színe fehér egy megvalósítható lekérdezési tervben, ahol $k \leq n$. Ahhoz, hogy az algoritmus ne generálja ezt a tervet, léteznie kell egy (v_{i_m}, v_{i_n}) párnak, amely között van egy él, amely meggátolja, hogy mindkét csúcs egyszerre fehér legyen. De ha a C érvényes színezésben ez a két csúcs fehér, akkor a G_{TP}-ben nem lehet a megfelelő éleknek olyan közös csúcsuk, melyek változó listája nem diszjunkt. Vagyis a G_J-ben nem lehet köztük él, ami ellentmondás. ■

36.3. Alkalmazás közösségi hálók elemzésére

alma A Web 2.0 megjelenésével interaktívvá váltak a weblapok. Ekkortól számítva a felhasználók már nem csupán böngészik, hanem ők maguk is hozzájárulnak egy-egy oldal tartalmához. Például leírják a véleményüket az oldalon bemutatott termékről, kommenteket fűznek a hírekhez, címkéket rendelnek a tartalomhoz ami segíti azok kategorizálását. Sőt, mára egyre több olyan szolgáltatás érhető el, amely csak egy keretrendszert ad a felhasználóknak, amivel saját tartalmakat hozhatnak létre, tölthetnek fel, videót, fotót, receptet, személyes bejegyzéseket vagy bármi mást. Emellett megjelentek az úgynevezett közösségi oldalak (például Facebook, Twitter, LinkedIn), ahol a felhasználók ismerkedhetnek, megoszthatják egymással az információkat, tartalmakat, kapcsolatokat alakíthatnak ki. Ezek a kapcsolatok egy nagy közösségi hálót alkotnak.

Az ipar és a kutatók is felismerték, hogy mennyire értékesek az ilyen, interaktív oldalakon a felhasználók által létrehozott tartalmak. Ipari szempontból érdekesek lehetnek például egy adott termékre vonatkozó hozzászólások, melyekből a gyártó visszajelzéseket kap, megismerheti a vásárlók véleményét. A sok pozitív vélemény elősegíti a termékek értékesítését. A közösségi hálókon megosztott adatok alapján célzott hirdetések, reklámok hozhatók létre. Tudományos szempontból pedig a kapcsolati hálók segítségével elemezhetővé válik a társadalmi kapcsolatoknak, a közösségeknek a kialakulása és szerveződése. A hálók különböző mérőszámokkal jellemezhetőek, szokták vizsgálni például a csomópontok fokszámának eloszlását, a háló átmérőjét, sűrűségét (azaz a lehetséges és a létező kapcsolatok arányát).

36.3.1. Közösségi hálózatok reprezentálása RDF segítségével

Napjainkban az Interneten egyre több, különböző típusú, eltérő témájú közösségi oldalt találunk, mint például a Facebook, Google+ vagy a LinkedIn. Ezek közös jellemzője, hogy a felhasználók különféle kapcsolatokat alakítanak ki egymás (barát, rokon, munkatárs) és a tartalmak (látta a filmet, olvasta a könyvet, megvette a terméket) között. A bevezetőben szó volt róla, hogy ezeknek a hálózatoknak az elemzése milyen előnyökkel jár. Nincsen kidolgozott szabvány azonban a hálók reprezentálására, a különböző oldalak eltérő módon, saját formátumokban tárolják azokat, megnehezítve az elemzők dolgát. Egyrészt a kifejlesztett algoritmusokat hozzá kell igazítani az éppen vizsgált háló tárolási módjához, másrészt a különböző forrásból származó hálózatokat nehéz integrálni.

A szemantikus web megoldást nyújt ezekre a problémákra, hiszen az RDF keretrendszer egyik fő célja az adatok integrálásának elősegítése. Ehhez egy

olyan leíró nyelvet biztosít, ami elég flexibilis a különböző témájú és szerkezetű közösségi hálók reprezentálásához. Továbbá ontológiák segítségével definiálhatjuk a kapcsolatok lehetséges típusait, a felhasználók, illetve a tartalmak (közösen aktorok) lehetséges osztályait. Ilyen ontológiák már léteznek, a legismertebbek közülük a ***Friend of a Friend*** (FOAF) és a ***Semantically-Interlinked Online Communities*** (SIOC).

A közösségi hálózatokat alapvetően kétféleképpen lehet reprezentálni. Az első a hagyományos megközelítés, amikor a háló csúcsai az aktorok, az élek pedig a köztük lévő kapcsolatok. Az aktorokhoz tartozhatnak attribútumok, amik speciális csúcsként jelennek meg.

36.40. definíció. ***(közösségi háló)***

A ***közösségi háló*** *egy $G = (V, E, C, cimke())$ élcímkézett irányított gráf, ahol*

1. *$V = A \cup M$ a csúcsok halmaza, az aktorok A és a tőle diszjunkt attribútumok M halmazának az uniója. Az aktorok halmaza tovább van particionálva típusok szerint, vagyis $A = \bigcup_t A_t$, valamint az attribútumok is lehetnek típusosak.*

2. *$E = E_{AA} \cup E_{AM}$ az élek halmaza, az $E_{AA} \subseteq A \times A$ aktorok közötti és $E_{AM} \subseteq A \times M$ aktorok és attribútumok közötti élek diszjunkt halmazának uniója,*

3. *$C = C_{AA} \cup C_{AM}$ az adott éltípushoz tartozó élcímkék halmaza,*

4. *$cimke()$ az élcímkéző függvény, és $cimke(e)$ az e él címkéjével egyenlő.*

Érdemes felhívni a figyelmet arra, hogy a definícióban az A halmaz particionálása, azaz osztályok szerinti szétosztása, valamint az élek címkéje megadható ontológiák segítségével.

A fent definiált gráfot ezután a következőképpen reprezentáljuk RDF hármasok segítségével.

36.41. definíció. ***(RDF reprezentáció)*** *Legyen $G = (V, E, C, cimke())$ egy közösségi hálózat adatmodellje. Ekkor*

1. *minden a aktorhoz rendelünk egy egyértelmű id_a azonosítót.*

2. *Minden a aktorhoz hozzárendeljük a típusát, vagyis egy $(id_a, tipusa, tipus_a)$ hármast. $H_T = \{(id_a, \text{'tipusa'}, tipus_a) \mid a \in A\}$.*

3. *Minden* $e = (a, b) \in E_{AA}$ *élhez, ahol* $a, b \in A$, *hozzárendelünk egy* $(id_a, cimke(e), id_b)$ *hármast.* $H_R = \{(id_a, cimke(e), id_b) \mid a, b \in A, (a, b) = e \in E_{AA}\}$.

4. *Minden* $e = (a, m) \in E_{AM}$ *élhez, ahol* $a \in A, m \in M$, *hozzárendelünk egy* $(id_a, cimke(e), m) \in H_M$ *hármast.* $H_M = \{(id_a, cimke(e), m) \mid a \in A, m \in M, (a, m) = e \in E_{AM}\}$.

Így a G ***közösségi hálózat RDF reprezentációja*** *a* $H = H_T \cup H_R \cup H_M$ *halmaz.*

A fent vázolt megközelítésnek az a hátránya, hogy a kapcsolatokhoz nem rendelhetünk attribútumokat. Ez bizonyos esetekben elengedhetetlen, például hogy mikor jött létre a kapcsolat, vagy ha az erősségét szeretnénk mérni. Egy másik hátrány, hogy ebben a szemléletben csak bináris kapcsolatokat lehet kifejezni. A második megközelítésben éppen ezért a kapcsolatokat is csúcsként reprezentáljuk, így azokhoz is rendelhetünk attribútumokat, illetve több aktor is részt vehet egy kapcsolatban. Az élek ebben az esetben a résztvevők szerepét jelölik.

36.42. definíció. ***(közösségi háló adatmodellje 2)*** *A* ***közösségi háló*** *egy* $G = (V, E, C, cimke())$ *élcímkézett irányított gráf, ahol*

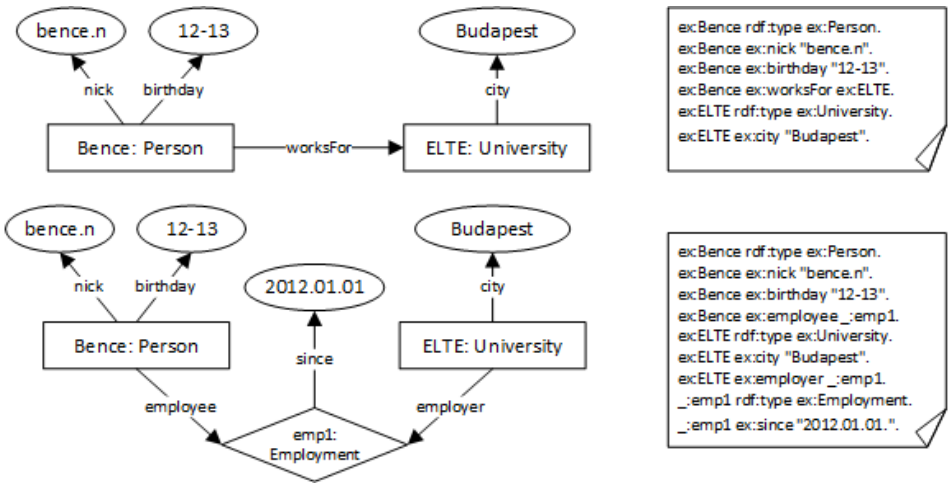
1. $V = A \cup M \cup R$ *a csúcsok halmaza, az aktorok* A, *az attribútumok* M *és a relációk* R *diszjunkt halmazának az uniója. Az aktorok, attribútumok és relációk halmaza tovább van particionálva típusok szerint, ahogyan az előző definícióban láttuk.*

2. $E = E_{AR} \cup E_{AM} \cup E_{RM}$ *az élek halmaza, az* $E_{AR} \subseteq A \times R$, $E_{AM} \subseteq A \times M$ *és* $E_{RM} \subseteq R \times M$ *diszjunkt halmazoknak az uniója,*

3. $C = C_{AR} \cup C_{AM} \cup C_{MR}$ *az adott éltípushoz tartozó élcímkék halmaza,*

4. $cimke()$ *az élcímkéző függvény, és* $cimke(e)$ *az* e *él címkéjével egyenlő.*

A második adatmodell RDF reprezentációja hasonló az első adatmodelléhez. A reprezentáció részleteinek kidolgozását lásd a gyakorlatok között, amihez segítséget nyújt a 36.2. ábra. A felső az első megközelítést, az alsó a másodikat követi. A négyzetek jelölik az aktorokat az azonosítójukkal, az azonosító után kettősponttal elválasztva pedig a típus szerepel. Az ellipszisek az attribútumok. Az alsó hálóban a rombusz a relációt mint csúcsot jelöli. A hálók mellett az RDF reprezentáció látható.



36.2. ábra. Közösségi hálók első (felül) és második (alul) adatmodell változata.

Az előzőekben láthattunk két megoldást arra, hogyan lehet a közösségi hálókat RDF hármasokkal reprezentálni. Az első megoldás, ahol az aktorok a csúcsok és a relációk az élek, egyszerűbb és természetesebb, viszont a második modell, ahol a relációkat is csúcsokként reprezentáljuk, lehetőséget ad n-áris relációk kifejezésére, illetve attribútumok megadására a relációkhoz is. Mivel a különböző közösségi hálók eltérő célt szolgálnak, előfordulhat, hogy egyiket az első, míg másikat a második modell szerint érdemes tárolni.

36.3.2. Közösségi hálók lekérdezése és transzformálása

A közösségi hálók gyakorlati felhasználása igen sokrétű. Többféle feladatot, lekérdezést szoktak megfogalmazni. A jellemző használati esetek a lekérdezéseket két nagy csoportba sorolják. Az első csoportba tartozó lekérdezések a hálónak, vagy annak egyes részeinek a struktúráját vizsgálják, és általában valamilyen mérőszámmal jellemzik azt. Ilyen lehet például a bevezetőben már említett fokszám eloszlás, átmérő vagy a sűrűség. A másik csoportba az adatok kezeléséért, transzformálásáért felelős lekérdezések tartoznak. Ezek közös jellemzője, hogy a kimenetük is egy háló, amit a bemenet transzformálásával kapunk. Ilyen feladatok jellemzően:

- *Csoportok kiválasztása.* Talán a leggyakoribb feladat, amikor a hálónak egy adott mintára illeszkedő részeit kell kiválogatni.
- *Attribútumok aktorrá alakítása.* Előfordulhat, hogy egy attribútumérték több aktornál is szerepel, esetleg egy attribútumhoz összetett értéket kell rendelni. Ezekben az esetekben szükséges lehet, hogy az attribútum

értékét aktorrá alakítsuk, amit új objektumok létrehozásával érhetünk el.

- *Relációk összevonása, csoportosítása.* Gyakran szükséges adott típusú relációk összevonása, számlálása, aktoronkéni csoportosítása, vagyis az aggregálása.
- *Adott aktor teljes környezetének lekérdezése.* Egy adott felhasználó vizsgálatakor ki kell tudni nyerni a felhasználó összes kapcsolatát, a kapcsolatokhoz tartozó további aktorokkal együtt. Ez a feladat általánosítható olyan módon, hogy az adott felhasználó k-sugarú környezetét kérdezzük le.

A következőkben egy olyan nyelvet vizsgálunk meg, mely a fenti feladatok megoldását segíti. A ***Social Network Query and Transformation Language*** (röviden: SNQL) a gyakran előforduló lekérdezéshez szükséges kifejezőerő biztosítása mellett az alacsony adatbonyolultságra törekszik. Az SNQL elsősorban az adattranszformációs műveleteket támogatja. Ennek megfelelően a lekérdezések felépítése CONSTRUCT – WHERE – FROM szerkezetet követ. A FROM záradékban megadott közösségi háló lesz a lekérdezés bemenete, amelyből a WHERE záradékban megadott ***kiválasztási minta*** alapján megszűrt adatokat a CONSTRUCT záradékban megadott ***konstrukciós minta*** szerint szervezzük újra.

Az alábbi példában a 36.2. ábrán láthatóhoz hasonló szerkezetű hálóból a *University* típusú csúcsok *city* attribútumait alakítjuk át aktorokká, és a második adatmodellt alkalmazva összekapcsoljuk a régi és az új aktorokat a megfelelő kapcslati csúcsokon keresztül.

36.10. példa. Attribútumok aktorrá alakítása

```
CONSTRUCT { (?U1, rdf:type, ex:University), (?U1, ex:locatedIn, ?R1),
              (?C1, ex:contains, ?R1), (?R1, rdf:type, ex:City)}
            IF ?R1=f(?U1,?C1)
WHERE       {(?U1, rdf:type, ex:University), (?U1, ex:city, ?C1)}
FROM        EmploymentNetwork
```

A példán megfigyelhetjük a konstrukciós minta és a kiválasztási minta felépítését, valamint a CONSTRUCT záradék IF feltételében szereplő kifejezést. Az IF kulcsszó után új változókat definiálhatunk a kiválasztási mintában definiált változók segítségével. Az f függvény egyedi azonosítót generál az új reláció csúcsoknak a relációban résztvevő aktorok alapján.

Az aggregációs feladatok megoldásához a nyelv az ***AGG*** kulcsszót biztosítja. Megadhatjuk az aggregálni kívánt változók nevét, az aggregáló függvényt és egy mintát, amely meghatározza az aggregálandó értékeket. Az alábbi példa hozzárendeli az emberekhez új attribútumként, hogy hány vál-

lalatnál dolgoznak. (A példában az első adatmodellt követjük.)

36.11. példa. Csoportosítás és aggregálás

```
CONSTRUCT { (?P1, rdf:type, ex:Person), (?P1, ex:worksFor, ?U1),
            (?P1, ex:numberOfEmployer, ?DB)}
WHERE     AGG({?U1}, COUNT AS ?DB,
            {(?P1, rdf:type, ex:Person), (?P1, ex:worksFor, ?U1)} )
FROM      EmploymentNetwork
```

A következő feladat megoldásához, ahol egy adott csúcs környezetét kell lekérdeznünk tetszőleges mélységig, szükségünk van a tranzitív lezárt kiszámítására. A nyelv ehhez a ***TC*** függvényt vezeti be, melynek három paramétere van, a kezdőpont, a végpont és egy kiválasztási minta, ami tartalmazza ezt a két pontot. Emellett egy kezdőfeltételt kell megadnunk a WITH kulcsszó után. A 36.12. példában, az első adatmodellt használva, olyan embereket keresünk, akik *worksFor* relación keresztül elérhetők *Bencétől* indulva, és összekötjük őket *Bencével* egy *ex:canReach* címkéjű éllel.

36.12. példa. Tranzitív lezárt

```
CONSTRUCT { (ex:Bence ex:canReach ?P2) }
WHERE     TC(?P1, ?P2,
            {(?P1, rdf:type, ex:Person), (?P1, ex:worksFor, ?U1),
             (?P2, rdf:type, ex:Person), (?P2, ex:worksFor, ?U1)}
             FILTER (?P1 != ?P2))
          WITH ?P1=ex:Bence
FROM      EmploymentNetwork
```

Az előző három példa jól szemlélteti a nyelv eszközkészletét, amellyel a fontosabb feladatok egyszerűen leírhatók. A pontos szintaktikai szabályok megtalálhatók a függelékben. A szintaktikai elemek megismerése után vizsgáljuk meg a nyelv szemantikáját. Legyen H egy közösségi háló és Q egy SNQL kérdés, jelölje $Q(H)$ a Q lekérdezés eredményét a H hálóra alkalmazva.

A lekérdezések kiértékelése két lépésből áll. Először az kiválasztási mintát átalakítjuk Datalog szabályokká, majd a konstrukciós minta alapján sorgeneráló függőségeket képzünk. A kiválasztási minta átalakítása a következő szabályok rekurzív alkalmazásával történik:

1. Minden $t = (s, p, o)$ hármast $t(s, p, o)$-ra fordítunk.
2. Egy $PATT = \{t_1, \ldots, t_n\}$ hármasokból álló mintához a $p(\bar{z}) \leftarrow \bigwedge_{i=1}^{n} t_i(s_i, p_i, o_i)$ szabályt rendeljük, ahol $\bar{z}$ a $PATT$-ban szereplő változókat jelöli.

3. $PATT_1$ AND $PATT_2 : p(\bar{z}) \leftarrow p_1(\bar{x}), p_2(\bar{y})$.
4. $PATT_1$ OR $PATT_2 : p(\bar{z}) \leftarrow p_1(\bar{x})$
   $p(\bar{z}) \leftarrow p_2(\bar{y})$.
5. $PATT_1$ AND-NOT $PATT_2 : p(\bar{z}) \leftarrow p_1(\bar{x}), \neg p_2(\bar{y})$.
6. $PATT_1$ FILTER $C : p(\bar{z}) \leftarrow p_1(\bar{x}), c(\bar{x})$.
7. $TC(u, v, PATT_1)$ WITH <startCondition> :
   $p(u, v) \leftarrow p_1(\ldots u \ldots v \ldots), startCond(\ldots u \ldots v \ldots)$
   $p(u, v) \leftarrow p_1(\ldots u \ldots v \ldots), p(w, v)$
8. $AGG(\bar{v}, agg, PATT_1) : p(\bar{v}, agg(\bar{y}) \leftarrow p_1(\bar{v}, \bar{y})$,
   ahol $\bar{x}$ a $PATT_1$ változóit jelöli, $\bar{v} \subseteq \bar{x}$, $\bar{y} = \bar{x} - \bar{v}$ és agg egy aggregáló függvény.

A második lépésben a konstrukciós minta alapján generálunk szabályokat. Ebben a lépésben felhasználjuk az előző lépés szabályait. Egy

CONSTRUCT trList IF eqList

mintából az eqList alapján $v_i = term_i$ és $term_i = term_l$ alakú kifejezéseket hozunk létre, ahol egy term lehet változó, konstans vagy függvény. Majd a minta és az egyenletek alapján

$$construct(v_1, \ldots, v_n) \leftarrow p(\bar{z}) \wedge \bigwedge_j eq_j$$

alakú szabályt képzünk. Végül az eredményt a trList alapján konstruáljuk a következőképpen:

$$H = \bigcup \{t(u_1, u_2, u_3) : \exists(..u_1..u_2..u_3..) \in construct \text{ and } t \text{ in } trList\}$$

36.13. példa. A 36.10. példában megadott lekérdezés átírása

$p(u_1, c_1)$	$\leftarrow t(u_1, \text{rdf:type}, \text{ex:University}),$ $t(u_1, \text{ex:city}, c_1)$
$construct(u_1, r_1, c_1)$	$\leftarrow ep(u_1, c_1), r_1 = f(u_1, c_1)$
$output(u_1, \text{rdf:type}, \text{ex:University})$	$\leftarrow construct(u_1, r_1, c_1)$
$output(u_1, \text{ex:locatedIn}, r_1)$	$\leftarrow construct(u_1, r_1, c_1)$
$output(c_1, \text{ex:contains}, r_1)$	$\leftarrow construct(u_1, r_1, c_1)$
$output(r_1, \text{rdf:type}, \text{ex:City})$	$\leftarrow construct(u_1, r_1, c_1)$

Az SNQL nyelv kidolgozásakor az egyik fő szempont az volt, hogy a gyakorlatban legtöbbször előforduló adattranszformációs feladatokat ki lehessen vele fejezni. A nyelv tervezésekor a Graphlog lekérdezőnyelvet és a másodrendű sorgeneráló függőségeket vették alapul.

36.43. állítás. *Legyen Q egy SNQL kérdés, H egy közösségi háló és $t = (s, p, o)$ egy RDF-hármas. Ekkor $t \in Q(H)$ eldöntése NLOGSPACE-ben van.*

A bizonyítás, amit most nem közlünk, a Graphlog és a másodrendű sorgeneráló függőségek vizsgálatán alapul.

A fenti példákból látszik, hogy az SNQL nyelv szerkezete és nyelvi elemei hasonlítanak a SPARQL CONSTRUCT típusú lekérdezéséhez. Mindkettő egy kiválasztási minta és egy konstrukciós minta alapján épül fel. Az SNQL annyiban tér el, hogy a kiválasztási mintában nem használhatunk bizonyos elemeket, amiket a SPARQL-ben igen, például az OPT és UNION kulcsszavakat. SNQL-ben az IF konstrukcióval új változókat képezhetünk, továbbá lehetőség van a tranzitív lezárt kiszámítására. Ezeket a funkciókat a SPARQL 1.1-ben pótolták, ahol a WHERE záradékban BIND kulcsszóval értéket rendelhetünk változókhoz, hasonlóan mint az IF konstrukcióban. Az 1.1-es verzióban továbbá útvonal kifejezéseket is definiálhatunk, vagyis megadhatunk egy reguláris kifejezést az élcímkék fölött, melyre a két csúcs közötti úton szereplő élek címkéinek illeszkedniük kell.

36.3.3. Csoportok kiválasztása hatékonyan

A közösségi hálók felhasználásának egyik gyakori feladata adott tulajdonságú csoportok kiválasztása, csoportok közötti kapcsolatok keresése. Egy közösségi háló vizsgálatakor kereshetünk például klikkeket, azaz olyan közösségeket, amelyben minden tag ismeri egymást, összekötőket, melyek összekapcsolnak különböző klikkeket vagy éppen egymástól teljesen független csoportokat is. Az előzőekben láttuk, hogyan lehet a közösségi hálókat RDF hármasokkal reprezentálni és azt, hogy milyen előnyei vannak ennek a rugalmas ábrázolási módnak. Ebben a fejezetben az RDF-hez kidolgozott SPARQL nyelv egy lehetséges kiegészítését vizsgáljuk meg, mely lehetővé teszi a csoportok kezelését. SPARQL-ben alapvetően a gráf csúcsaira és éleire vonatkozó konjunktív lekérdezéseket fogalmazhatunk meg. A javasolt kiegészítésben ezeket a konjunktív lekérdezéseket terjesztjük ki csúcshalmazokra.

36.44. definíció. ***(Reguláris útvonalkérdés)*** *Legyen $H = (V, E)$ egy irányított élcímkézett gráf. Jelöle a $a \xrightarrow{p_1 p_2 \ldots p_n} b$ az a és b csúcs közötti $p_1, \ldots, p_n$ élcímkékkel címkézett utat. Ekkor $Q^R \leftarrow R$ az R reguláris kifejezéssel definiált reguláris útvonalkérdés. Az $ans(Q^R, H)$ válasz azokat a csúcspárokat tartalmazza, amelyek a reguláris kifejezésre illeszkedő útvonallal össze vannak kötve, vagyis:*

$$ans(Q^R, H) = \left\{ (a, b) \in V \times V \mid a \xrightarrow{p} b, \text{ahol } p \in L(R) \right\}$$

$L(R)$ az R reguláris kifejezés által generált nyelvet jelöli.

36.45. definíció. ***(Konjunktív reguláris útvonalkérdés)*** *A konjunktív reguláris útvonalkérdések a következő alakúak:*

$$Q^C(x_1,\ldots,x_n) \to y_1 R_1 y_2 \wedge \ldots \wedge y_{2m-1} R_m y_{2m},$$

ahol $x_1,\ldots,x_n,y_1,\ldots,y_m$ csúcsváltozók és $\{x_1,\ldots,x_n\} \subseteq \{y_1,\ldots,y_n\}$. Az $ans(Q^C,H)$ válasz azokat a $(v_1,\ldots v_n)$ H-beli csúcsokból képzett n-eseket tartalmazza, amelyekhez létezik olyan σ leképezés, melyre $\sigma(x_i) = v_i$, és $(\sigma(y_i),\sigma(y_{i+1})) \in ans(Q^R,H)$ minden $y_i R_i y_{i+1}$ term által definiált Q^R reguláris útvonalkérdésre.

Ahhoz, hogy a fenti definíciókat ki lehessen terjeszteni halmazokra, szükséges, hogy a halmazok bizonyos részeit ki tudjuk jelölni. Ehhez megadjuk a $\forall$ és a $\exists$ kvantoroknak a halmazokra vonatkozó kiterjesztését.

- Univerzális kvantor: $\forall_M = \{M\}$
- Egzisztenciális kvantor: $\exists_M = \{A \subseteq M \mid A \neq \emptyset\}$
- Számossági kvantor: $\exists_M(\odot n) = \{A \subseteq M \mid |A| \odot n\}$, ahol $\odot \in \{>,\geq,=,\leq,<\}$

A továbbiakban a görög Ξ vagy Ψ betűket úgy használjuk, hogy helyükre a fenti kvantorok valamelyike behelyettesíthető.

36.46. definíció. ***(Halmaz-csúcs reguláris útvonalkérdések)*** *$Q^{\Xi\bullet} \leftarrow R$ egy halmaz és egy csúcs közötti reguláris útvonalkérdés, ahol a Ξ kvantor meghatározza, hogy a halmaz hány elemének kell kapcsolódnia a $\bullet$-al jelölt csúcshoz úgy, hogy az útvonalon szereplő élek címkéje illeszkedjen az R reguláris kifejezésre. A válasz ennek megfelelően rendezett párokból áll, melynek első eleme egy halmaz, a második eleme pedig egy csúcs, vagyis:*

$$ans(Q^{\Xi\bullet},H) = \left\{(A,b) \in 2^V \times V \mid a \xrightarrow{p} b, ahol\ a \in \Xi_A, p \in L(R)\right\}$$

A fordított irány, azaz egy csúcs és egy halmaz közötti reguláris útvonalkérdések, illetve a halmaz és halmaz közötti kérdések hasonlóan definiálhatóak.

36.47. definíció. ***(Halmaz-csúcs reguláris útvonalkérdés lezártja)*** *Legyen $Q^{\Xi\bullet}$ egy halmaz-csúcs reguláris útvonalkérdés, ekkor a* ***lezártja,*** *$\bar{Q}^{\Xi\bullet}$, tovább szűkíti az eredményhalmazt úgy, hogy csak azokat az A és b közötti útvonalakat engedi meg, amelyek A-ban maradnak, tehát*

$$ans(\bar{Q}^{\Xi\bullet},H) = \{\ (A,b) \in 2^V \times V \mid a \xrightarrow{p} b, ahol\ a \in \Xi_A,\ p_1,\ldots,p_n = p \in L(R)$$
$$és\ \forall i \in \{1,\ldots,n-1\}\ a \xrightarrow{p_1\ldots p_i} c => c \in A\}$$

36.48. definíció. ***(Halmaz méretét megszorító kérdés)*** *A $Q^{|.|} \leftarrow (from, to)$ kérdés egy unáris relációt ír le, ahol $from, to \in \mathbb{N}, from \leq to$ a halmaz méretének a minimumát és maximumát definiálja. A válasz azokból a halmazokból áll, melyek mérete megfelel a feltételeknek:*

$$ans(Q^{|.|}, H) = \left\{A \in 2^V \mid |A| \in \{from, \ldots, to\}\right\}$$

A fent definiált kérdéstípusokból konjunkcióval összetett kérdéseket lehet alkotni, hasonlóan a konjunktív reguláris útvonalkérdésekhez. A pontos definíció a következő.

36.49. definíció. ***(Konjunktív reguláris útvonalkérdés halmazokkal)*** *A halmazokat is megengedő konjunktív reguláris útvonalkérdés alakja*

$$Q^S(x_1, \ldots, x_n) \leftarrow \tilde{y}_1[(R_1)^{\Psi_1^1 \Psi_2^1}]y_2 \wedge \ldots \wedge \tilde{y}_{2m-1}[(R_m)^{\Psi_1^m \Psi_2^m}]y_2 m \\ \wedge Z_1[f_1, t_1] \wedge \ldots \wedge Z_l[f_l, t_l],$$

ahol $x_1, \ldots, x_n, y_1, \ldots, y_m$ csúcs- vagy halmazváltozók és $\{x_1, \ldots, x_n\} \subseteq \{y_1, \ldots, y_m\}$. $Z = \{Z_1, \ldots, Z_l\}$ az y_i-k közötti összes halmazváltozó halmaza. $A \sim$ szimbólum vagy üres, vagy halmazok esetén lehet –, ami a lezártat jelenti. A Ψ vagy egy kvantor, vagy a $\bullet$ szimbólum, amely egy egyszerű csúcsot jelöl. Az R_i-k reguláris kifejezések.

Az $ans(Q^S, H)$ azokat a $(v_1, \ldots, v_n)$ H-beli csúcsokból és halmazokból képzett rendezett n-eseket tartalmazza, melyekhez létezik olyan σ totális leképezés, hogy $\sigma(x_i) = v_i$ és $(\sigma(y_i), \sigma(y_{i+1}))$ szerepel a megfelelő részkifejezés eredményhalmazában.

36.14. példa. Konjunktív reguláris útvonalkérdés halmazokkal

A példában egy olyan adathalmazt kérdezünk le, ahol a csúcsok *knows* címkéjű élekkel vannak összekötve, és feltesszük, hogy ez a reláció reflexív. Ekkor olyan (A, B) halmazpárokat keresünk, melyekre a következő feltételek teljesülnek:

1. Az A halmazban mindenki legfeljebb kettő hosszú úttal össze van kötve Bencével, akit az *ex:Bence* azonosító jelöl, és ezek az utak nem vezetnek ki A-ból.

2. A B halmazban mindenki ismer mindenkit.

3. A B halmazból mindenki ismer legalább három A halmazbeli személyt.

4. Az A halmaz hat, a B halmaz legalább négy, de legfeljebb öt elemű.

Ezeket a feltételeket fejezi ki az alábbi kérdés:

$$
\begin{aligned}
Q^S(A,B) \leftarrow\ & \\
& \bar{A}[(knows.knows?)^{\forall\bullet}]ex\!:\!Bence \\
\wedge\ & B[(knows)^{\forall\forall}]B \\
\wedge\ & B[(knows)^{\forall\exists(\geq 3)}]A \\
\wedge\ & A[6,6] \wedge B[4,5]
\end{aligned}
$$

Amit a következőképpen írhatunk át SPARQL lekérdezéssé:

```
SELECT ??A ??B
WHERE{
        ALL CLOSURE ??A knows\knows? ex:Bence.
        ALL ??B knows ALL ??B.
        ALL ??B knows SOME(≥ 3) ??A.
        FILTER ( ??A{6, 6}, ??B{4, 5})}
```

Az átírt lekérdezésből látszik, hogy néhány nyelvi elem bevezetésével hogyan fejezhetők ki a halmazokat is megengedő konjunktív reguláris útvonalkérdések. A halmaz változókat a ?? dupla kérdőjel jelzi. A reguláris kifejezésben szereplő, halmazokra vonatkozó kvantorok közül az univerzális kvantort az ALL kulcsszóval, az egzisztenciálisat a SOME-mal, míg a számossági kvantort szintén a SOME kulcsszóval jelöljük, amely után zárójelben feltüntettük a megszorítást. A halmaz lezárását a CLOSURE jelzi. Továbbá, a halmazok elemszámára vonatkozó feltételeket a FILTER után írjuk a fent látható alakban.

Az átírás után vizsgáljuk meg a halmazokat is megengedő reguláris útvonalkérdések bonyolultságát.

36.50. állítás. *Legyen Q egy kérdés, H egy N csúcsú élcímkézett háló, $\bar{v}$ csúcsváltozók és $\bar{V}$ halmazváltozók rendezett n-ese. Ekkor $(\bar{v},\bar{V}) \in Q(H)$ eldöntése PTIME-ban van.*

Az állítás bizonyításhoz elég látni, hogy a csúcs- és halmazváltozók – jelöljük őket c-vel és C-vel – száma rögzített a kérdésben. Továbbá, minden halmaz alulról és felülről korlátos a megfelelő megszorító kérdés miatt. Legyen K a felső korlátok maximuma. Ekkor a kérdés megválaszolásához legfeljebb N^{c+K*C} esetet kell ellenőrizni, ami a háló méretében polinomiális.

Függelék: az SNQL nyelv szintaxisa

<query> ::= CONSTRUCT <construct-pattern>

```
                                WHERE <ext-patt>
                                FROM  <list-of-networks>
<construct-pattern>      ::= <list-of-pattern-triples>[IF <list-of-eqs>]
<list-of-eqs>            ::= <var>= <expr>[ AND <var>= <expr>]*
<ext-patt>               ::= <list-of-pattern-triples>
                           | (<ext-patt>) AND (<ext-patt>)
                           | (<ext-patt>) OR (<ext-patt>)
                           | (<ext-patt>) AND-NOT (<ext-patt>)
                           | (<ext-patt>) FILTER (<condition>)
                           | TC(<var>, <var>, <ext-patt>)
                             WITH <condition>
                           | AGG({<var>[, <var>]*},
                                 <agg-func>, <ext-patt>)
<list-of-pattern-triples> ::= {<pattern-triple>[, (<pattern-triple>)]*}
<pattern-triple>         ::= (<term>, <term>, <term>)
<list-of-networks>       ::= <network>[, <network>]*]
<network>                ::= <network-id> | <list-of-instance-triple>
<list-of-instance-triple> ::= {<instance-triple>[, <instance-triple>]*}
<instance-triple>        ::= (<constant>, <constant>, <constant>)
<expr>                   ::= <term> | <func>
<term>                   ::= <var> | <constant>
<func>                   ::= func(<expr>[, <expr>]*)
<agg-func>               ::= <literal>
<condition>              ::= <logical-expression>
<var>                    ::= ?<literal> | $<literal>
<constant>               ::= <literal>
```

Gyakorlatok

36.3-1. Írjuk fel a 36.42. definícióban leírt adatmodellhez tartozó RDF reprezentáció formális definícióját.

36.3-2. A 36.46. definíció alapján írjuk fel a csúcs-halmaz és halmaz-halmaz reguláris útvonal kérdések és a hozzájuk tartozó válaszok formális definícióját.

36.3.4. Közösségi hálózatok lekérdezése MapReduce módszerrel

A szemantikusan leírt közösségi adatok lekérdezésének legegyszerűbb MapReduce szemléletű módszere alaptáblák előállítására épül. A táblák azokat a hármasokat tartalmazzák, amelyek a lekérdezés egy adott feltételére teljesülnek. Ezt a szakaszt nevezzük SELECTION résznek. A SELECTION szakasz

végigfut az adatokon, és kiválasztja azokat az értékeket, amelyek megfelelnek egy adott változónak.

36.15. példa. Az alábbi SPARQL lekérdezés a Budapest Bár iránt érdeklődő felhasználóknak azokat a barátait adja vissza, akik Budapesten laknak és akikkel egy 2013. május 2–nál újabb képen mindketten meg vannak jelölve.

```
SELECT ?Friend WHERE {
        ?User hasInterest "Budapest Bár" .
        ?Friend locatedAt "Budapest" .
        ?Photo laterThan "2013-05-02" .
        ?User friendOf ?Friend .
        ?Photo taggedBy ?User .
        ?Photo taggedBy ?Friend
}
```

A példában egy lekérdezés feltételeit láthatjuk. Ez egy tipikus lekérdezése a közösségi hálóknak, ami a célzott reklámok kiválasztására szolgál. A lekérdezésben vesszük azokat a felhasználókat, akik egy bizonyos területről vannak és egy közeli ismerősük érdeklődik egy bizonyos termék iránt. A kiválasztási szakaszban ekkor hat alaptábla fog elkészülni: hasInterest(?User), locatedAt(?Friend), laterThan(?Photo), friendOf(?User?Friend), taggedBy(?User?Photo) és a taggedBy(?Friend?Photo). Ezeket a táblákat egy join szakasz fogja feldolgozni. Ez a megoldás nem hatékony, mivel vannak benne felesleges MapReduce job-ok. Ennek oka, hogy a hagyományos SQL join operátor és a SPARQL lekérdezés kiértékelése különbözik. A join esetében az egyes összekapcsolásoknál az alaptáblák száma korlátozott, emiatt több jobra van szükségünk.

SELECTION rész mellett a join rész sem hatékony. A jelenlegi megoldás a kiválasztásokat MapReduce folyamatokkal oldja meg, és az elkészült alaptáblákat használja későbbiekben az összekapcsolásra. Ennél egy hatékonyabb megoldás, ha kihasználjuk a MapReduce rugalmasságát, és a SELECTION részt beolvasztjuk a join részbe, ezzel csökkentve a felesleges jobok számát.

Többszörös összekapcsolás

Legyen adott egy SPARQL lekérdezésünk a következő négy alaptáblával: t1(?X?Y), t2(?X?Y), t3(?X?Z), t4(?X?Z). Azonos sémájú alaptáblák a különböző kapcsolatok miatt lehetségesek. Az SQL alapú megoldás előbb kiértékeli az ?X?Y és ?X?Z táblákat, majd összekapcsolja az eredményeket ?X alapján. Minden join művelethez egy MapReduce jobra van szükség. Hatékonyabb, ha összekapcsoljuk az összes alaptáblát ?X alapján, és szűrjük párhuzamosan ?Y és ?Z változókra. Ehhez a megoldáshoz mindössze csak egy jobra van szükségünk. Ezt a megoldást nevezzük multiple-join-with-filter metódusnak.

A művelet szintaxisa a következő:

Multiple join [tables] on [join Key]
Filter on [Filter Key]
Dump Result to [MT]

Vegyük a korábbi 36.15-es példánkat. Miután összekapcsoltuk a hasInterest(?User), friendOf(?User?Friend) és a taggedBy(?UserPhoto) táblákat a ?User alapján, a következő lépés az így kapott ?User?Friend?Photo tábla összekapcsolása a ?Friend és a ?Friend?Photo táblával a ?Friend változón keresztül. A multiple-join-with-filter művelettel ez a következőképpen néz ki:

Multiple join [?Friend, ?Friend?Photo,
?User?Friend?Photo] on [?Friend]
Filter on [?Photo]
Dump Result to [?User?Photo]

Ebben a példában a ?Friend változót használtuk join kulcsnak. Ezzel egy időben a ?Photo-ra kapott különböző értékeket leszűrjük a ?User?Friend?Photo és a ?Friend?Photo alapján. A multiple-join-with-filter a ?Photo változót használja szűrő kulcsnak.

Kiválasztó összekapcsolás

A rugalmas MapReduce programozás megengedi, hogy egyszerre generáljunk alaptáblákat és végezzünk join műveletet. Ennek eredményeként nem kell annyi job a kiválasztási szakaszban. Ezt a hibrid réteget nevezzük primitív select-join műveletnek, ahol a kiválasztás be lett integrálva az összekapcsolásba. A művelet során két csoportot alkalmazunk, ahol egy csoport hármasokat tartalmazó halmaz. Az első csoport a kiválasztó csoport, mely az alaptáblákat generálja. A másik csoport az összekapcsolási csoport, amely a join műveleteket végzi. A select-join szintaxisa a következő:

Select on [triplets A]
Multiple join [triplets B] on [join Key]
Filter on [Filter Key]
Dump Result to [ST]

A select-join működését az alábbi példa szemlélteti.

36.16. példa. Az alábbi SPARQL lekérdezés olyan hozzászólásokat ad vissza, melyek 2012. január 1-nél újabb keltezésűek, és azokat sorba rendezi a rájuk

vonatkozó megjelölések száma szerint csökkenően.

```
SELECT ?Post WHERE {
        ?Post rdf:type Post .
        ?Post laterThan "2012-01-01" .
        ?Post tag ?Tag .
} GROUP BY ?Tag
ORDER BY DESC (count(?Post))
```

A lekérdezés a legtöbb post-tal rendelkező aktív témákat kérdezi le 2012 óta. Mivel csak egy megosztott változó van (?Post), ezért elég csak egy összekapcsolás ahhoz, hogy megkapjuk az értékét. Ez megoldható egyetlen MapReduce jobbal, ha a kiválasztást az összekapcsolással együtt végezzük el. Az összes feltétel az összekapcsolási csoportba kerül és a kiválasztási csoportba nem kerül semmi, vagyis nincs szükségünk alaptábla generálásra. Kijelenthetjük, hogy minden olyan SPARQL lekérdezés, amelyben egy változó van megosztva, lefuttatható egy MapReduce jobbal, amely teljesen kihagyja a köztes adatokat. Ezek a lekérdezések jellemzőek a közösségi hálók lekérdezésében, ezeket a lekérdezéseket nevezzük ***csillag alakú lekérdezéseknek***.

Lekérdezésterv generálás

Az algoritmus egymás után dolgozza fel a hármasokban lévő változókat. Minden lépésben mindig a leggyakrabban megosztott változó lesz a join kulcs. Az algoritmus indít egy SQL join-t vagy egy multiple-join-with-filter jobot a változóra. A kiválasztási rész mindig benne van az első joinban. A multiple-join-with-filter függ attól, hogy van-e érvényes szűrő kulcs. Ha megszámoljuk a nem join kulcsváltozókat és találunk olyan változót, amely egynél többször szerepel a hármasokban, akkor azt használhatjuk szűrő kulcsnak. Ha két változó számossága megegyezik, akkor az algoritmus önkényesen választ a két változó közül. Az utolsó eredmény generálása után az algoritmus visszaadja a MapReduce jobokból álló lekérdezési tervet.

Input. Egy SPARQL lekérdezés Q (hármasok halmaza).
Output. MapReduce folyamat W.
MapReduceWorkflowGeneration

```
1 v = mostFrequentVariable(Q)
2 tp1 = triplets in Q not sharing v
3 tp2 = triplets in Q sharing v
4 fk = v-n kívüli változók amelyek egynél több tp2-beli hármasban
                    szerepelnek
```

```
5 job =“Select on”+ tp1 +“Multiple join”
                + tp2 +“join on” + v
                + “Filter on” + fk + “Dump Result to” + ST
6 W.append(job)
7 while ST.size() > 1 do
8               VA = correspondingVariables(ST)
9               v = mostFrequentVariable(VA)
10              tb = tables in ST sharing v
11              fk = v-n kívüli változók amelyek egynél több tb-beli
                                    táblában szerepelnek
12              job = “Multiple join” + tb + “join on” + v +
                “Filter on” + fk + “Dump Result to” + MT
13              W.append(job)
14              remove tb from ST
15              add MT to ST
16 return W
```

Gyakorlatok

36.3-1. Generáljunk lekérdezési tervet a MapReduceWorkflowGeneration algoritmussal az alábbi lekérdezésre.

```
1 SELECT ?X, ?Y1, ?Y2, ?Y3
2 WHERE {
3       ?X rdf:type ub:Professor .
4       ?X ub:worksFor <http://www.Department0.University0.edu> .
5       ?X ub:name ?Y1 .
6       ?X ub:emailAddress ?Y2 .
7       ?X ub:telephone ?Y3 .
8 }
```

36.3-2. Generáljunk lekérdezési tervet a MapReduceWorkflowGeneration algoritmussal az alábbi lekérdezésre.

```
1 SELECT ?X, ?Y, ?Z
2 WHERE {
3       ?X rdf:type ub:Student .
4       ?Y rdf:type ub:Department .
5       ?X ub:memberOf ?Y .
6       ?Y ub:subOrganizationOf <http://www.University0.edu> .
7       ?X ub:emailAddress ?Z
8 }
```

36.3-3. Generáljunk lekérdezési tervet a MapReduceWorkflowGener-

ATION algoritmussal az alábbi lekérdezésre.

```
1 SELECT ?X, ?Y, ?Z
2 WHERE {
3       ?X rdf:type ub:Student .
4       ?Y rdf:type ub:Faculty .
5       ?Z rdf:type ub:Course .
6       ?X ub:advisor ?Y .
7       ?Y ub:teacherOf ?Z .
8       ?X ub:takesCourse ?Z
9 }
```

Feladatok

36-1 Költségbecslés

Próbáljunk meg aszimptotikus felső becslést adni a GENERATEBESTPLAN algoritmus futási idejére, a hármas-minta gráf csúcsainak és éleinek a függvényében.

36-2 Lekérdezés bonyolultság

Legyen H egy rögzített n csúcsú élcímkézett közösségi háló. Adjuk meg a $(\bar{v}, \bar{(V)}) \in Q(H)$ döntési probléma bonyolultsági osztályát adott Q, halmazokat is megengedő konjunktív reguláris útvonal kérdésre (36.49. definíció), ahol $\bar{v}$ csúcsváltozók és $\bar{V}$ halmazváltozók rendezett n-ese.

Megjegyzések a fejezethez

A szemantikus web elméleti és gyakorlati problémáiról az első magyar nyelvű tankönyveket Gottdank Tibor [153] és Szeredi Péter [335] írta. A SPARQL lekérdezések MapReduce [91] technikával történő kiértékelésének megoldását írják le Husain és társai [188]. A cikkben szó esik az adatok tárolásáról és a lekérdezéshez szükséges jobok előállításáról. Bemutatnak egy algoritmust, amely elkészíti a lekérdezés kiértékeléséhez szükséges futási tervet.

A közösségi hálózatok RDF reprezentálásával főként a [316] cikk foglalkozik, amely a reprezentálás bevezetése mellett egy lekérdezőnyelvet is bemutat, amelyből a szerzők később megalkották a [317] cikkben bemutatott SNQL nyelvet a hálók lekérdezésére, transzformálására. A nyelv a Graphlog [81] és a másodrendű sorgeneráló függőségek [121] előnyeit egyesíti. A közösségi hálózatok elemzésének egyik alapja a felhasználói csoportok vizsgálata. A SPARQL-ben [173] a lekérdezések alapja a pontokra és a közöttük lévő élekre vonatkozó mintaillesztés, amit a [231] cikkben leírt módon

kiegészíthetünk felhasználói csoportok és a csoportok közötti összeköttetések megadásával.

A fentebb említett MapReduce technika közösségi hálózatok lekérdezésére történő alkalmazását Liu, Yin és Gao [233] mutatták be. Megoldásuk a relációs táblák összekapcsolásán alapszik.

A szemantikus web technológiák számos további gyakorlati alkalmazásban kerülnek felhasználásra. Matuszka, Gombos és Kiss [252] cikkükben egy beltéri navigációs rendszert ismertetnek, mely a lehetséges útvonalak kiszámításához RDF reprezentációt és következtetéseket használ fel. Gombos és társai egy általános célú virtuális obszervatóriumot ismertetnek szemantikus adatbázisok felett [151]. Zsigmondi és Kiss [388]-ben bevezették a szemantikus helyettesítő karakter fogalmát, amellyel általánosítható a szöveges keresés. Matuszka a szemantikus web kiterjesztett valósággal kapcsolatos alkalmazásáról számolt be a [251] publikációban.

Jelen kutatást a FuturICT.hu nevű, TÁMOP-4.2.2.C-11/1/KONV-2012-0013 azonosítószámú projekt támogatta az Európai Unió és az Európai Szociális Alap társfinanszírozása mellett.

37. A szemantikus web lekérdező nyelvei

A szemantikus web lényege, hogy az Interneten fellelhető információkat a számítógépek is értelmezni tudják, ezáltal a weben szétszórva tárolt adatok könnyebben összekapcsolhatóvá, hatékonyabban kereshetővé válnak. Az összekapcsolt adatokból következtetések útján új információk nyerhetők ki. A szemantikus web ehhez nyújt egy szabványosított adatmodellt és hozzá tartozó lekérdezőnyelvet. Az RDF (Resource Description Framework) adatmodell lényege, hogy a rendelkezésünkre álló információt alany-állítmány-tárgy formában megfogalmazott állítások halmazaként adjuk meg. Az ilyen állítások felfoghatók irányított élekként is, amelyek az alannyal címkézett csúcsból vezetnek a tárggyal címkézett csúcsba, az állítmánynak megfelelő élcímkével. Az RDF adatok lekérdezésére szolgáló SPARQL lekérdezőnyelv ezért az adatokban való keresést gráfminta-illesztési feladatra vezeti vissza.

Ahhoz, hogy a gépi feldolgozás kivitelezhetővé váljon, szükséges, hogy az azonos fogalmak ugyanazt jelentsék, ugyanaz az azonosító tartozzon hozzájuk a különböző adatbázisokban. Erre a problémára adnak megoldást a szókészletek és az ontológiák. A szókészletek tartalmazzák a fogalmakat azonosító IRI-k (Internationalized Resource Identifier) megfeleltetését a valós világ fogalmaival, míg az ontológiák a szókészleteket, valamint a definiált fogalmak közötti kapcsolatokat, szabályokat, megszorításokat. Számos elméleti és kísérleti eredményeket tartalmazó adatbázis érhető el az Interneten különböző tudományterületekről, például számítástudomány, biológia, kémia. A LOD (Linked Open Data) felhő az ilyen publikus adathalmazokat kapcsolja össze egy nagy összefüggő gyűjteménnyé. A LOD-felhő olyan szemantikus formátumú (RDF-ként tárolt) adathalmazokat tartalmaz, melyek legalább ezer sorból állnak, és legalább ötven ponton kapcsolódnak más LOD felhőbeli adathalmazhoz. Ezek a megkötések lehetővé teszik az információkinyerést a különböző tudományterületeken felhalmozódott tudásbázisokból.

37.1. Szemantikus adatok

37.1.1. RDF reprezentáció

Tekintsünk három diszjunkt halmazt, a B-t (üres csúcsok, melyekhez nem tartozik IRI), az L-et (literálok), és az I-t (IRI-k), illetve a BLI, BI és

LI rövidítést, amelyet a B, L és az I uniójának jelölésére használunk. A BLI rövidítést más néven RDF termnek is nevezzük. A megszokott konvenció szerint idézőjeleket használunk a literálok jelölésére (például: "János", "25") és az "_:" prefixet az üres csúcsokhoz. Egy $(v_1, v_2, v_3) \in BI \times I \times BLI$ RDF hármas összekapcsolja a v_1 alanyt a v_2 tárggyal a v_3 állítmányon keresztül. Egy RDF adatbázis (más néven RDF dokumentum) ilyen hármasok véges halmaza.

37.1.2. SPARQL lekérdező nyelv

Szintaxis

Legyen V a BLI-beli különböző változók halmaza. A változókat egy vezető kérdőjellel különböztetjük meg, például $?X$ vagy $?nev$. Elsőként a szűrőfeltételek absztrakt szintaxisát adjuk meg.

37.1. definíció. *Legyen $?X, ?Y \in V$ és $c, d \in LI$, ekkor a szűrőfeltételeket rekurzívan a következőképp definiáljuk. A $?X = c$, $?X = ?Y$, $c = d$, $bound(?X)$, $isIRI(?X)$, $isLiteral(?X)$, valamint $isBlank(?X)$ atomi szűrőfeltételek. Ezt követően, ha R_1, R_2 szűrőfeltételek, akkor $\neg R_1$, $R_1 \wedge R_2$ és $R_1 \vee R_2$ is szűrőfeltételek. A $vars(R)$ jelöli az R szűrőkifejezésben előforduló változók halmazát.*

A következőekben bemutatjuk a kifejezések egy absztrakt szintaxisát.

37.2. definíció. *Egy **SPARQL kifejezés** a következőek szerint épül fel rekurzív módon:*

1. *a $t \in IV \times IV \times LIV$ hármas egy kifejezés,*
2. *ha Q_1, Q_2 kifejezések, és R egy szűrőfeltétel, akkor Q_1 Filter R, Q_1 Union Q_2, Q_1 Opt Q_2, valamint Q_1 And Q_2 is kifejezés.*

A hivatalos W3C ajánlás négy különböző lekérdezéstípust ad meg, ezek a Select, Ask, Construct és a Describe lekérdezések. A továbbiakban csak a Select és Ask lekérdezésekkel foglalkozunk. A Select lekérdezések megadják a lekérdezés feltételének megfelelő változóhelyettesítések halmazát, míg az Ask lekérdezések logikai lekérdezések, amelyek akkor és csak akkor térnek vissza *igaz* értékkel, ha létezik egy vagy több eredmény, egyébként *hamis* eredményt adnak vissza. A jelölések egyszerűsítése érdekében eltekintünk a kapcsos zárójelek alkalmazásától a Select operátorban megjelenő változók halmazánál, például $\text{Select}_{?X,?Y}(Q)$-t írunk $\text{Select}_{\{?X,?Y\}}(Q)$ helyett.

37.3. definíció. *Legyen Q egy SPARQL kifejezés, és legyen $S \subset V$ változók véges halmaza. Egy* ***SPARQL* SELECT *lekérdezés*** *egy* $\text{SELECT}_S(Q)$ *formájú kifejezés. Egy* ***SPARQL* ASK *lekérdezés*** *egy* $\text{ASK}(Q)$ *formájú kifejezés.*

Szemantika

Kétféle alternatív szemantikát vezetünk be a SPARQL kiértékeléshez, a halmaz és a multihalmaz szemantikát. A halmaz alapú szemantikát a szakirodalom cikkeiben található elméleti vizsgálatokban használják, míg a multihalmaz szemantika a hivatalos W3C ajánlásokat követi.

A halmaz alapú SPARQL szemantika. A SPARQL kiértékelési folyamat középpontjában az úgynevezett leképezés áll, amely kifejezi a változók dokumentumra való leképezését a kiértékelés során. Formálisan egy $\mu : V \to BLI$ leképezés egy parciális függvény, amely a V-beli változók részhalmazáról képez az RDF termekre. $\mathcal{M}$-el jelöljük az összes leképezés univerzumát. A μ leképezés értelmezési tartománya a $dom(\mu)$, amely a V-nek egy olyan részhalmaza, amelyen μ értelmezett. Két leképezés, μ_1, μ_2 kompatibilisek, jelölés szerint $\mu_1 \sim \mu_2$, ha megegyeznek az összes közös változón, azaz $\mu_1(?X) = \mu_2(?X)$ minden $?X \in dom(\mu_1) \cap dom(\mu_2)$.

Felüldefiniáljuk a *vars* függvényt (amelyet korábban a szűrőfeltételeknél vezettünk be), és jelöljük ezután $vars(t)$-vel (ahol t egy RDF hármas) az összes, t hármasban szereplő változót. Továbbá, jelölje $\mu(t)$ az összes olyan hármast, amelyet úgy kapunk, hogy az összes t-beli $?X \in dom(\mu) \cap vars(t)$ változót lecseréljük $\mu(?X)$-re.

37.1. példa. Tekintsünk három leképezést, $\mu_1 := \{?X \mapsto a\}$, $\mu_2 := \{?X \mapsto b, ?Y \mapsto c\}$ és $\mu_3 := \{?X \mapsto a, ?Z \mapsto d\}$. Könnyű látni, hogy $dom(\mu_1) = \{?X\}$, $dom(\mu_2) = \{?X, ?Y\}$ és $dom(\mu_3) = \{?X, ?Z\}$. Továbbá megfigyelhető, hogy $\mu_1 \sim \mu_3$, de μ_1 és μ_2, illetve μ_2 és μ_3 nem kompatibilisek, azaz $\mu_1 \nsim \mu_2$ és $\mu_2 \nsim \mu_3$. Legyen $t_1 := (e, ?X, ?Y)$, ekkor $var(t_1) = \{?X, ?Y\}$, és $\mu_2(t_1) = (e, b, c)$.

A következőekben definiáljuk a szűrőfeltételek szemantikáját.

37.4. definíció. *Adott μ leképezés, R, R_1, R_2 szűrőfeltételek, $?X, ?Y$ változók, illetve $c, d \in LI$ esetén μ* ***kielégíti* R*-t***, *jelölés szerint $\mu \models R$ akkor és csak akkor, ha a következő feltételek valamelyike teljesül R-re:*

- *$bound(?X)$ formában van, és $?X \in dom(\mu)$.*
- *$c = d$ formában van, és c megegyezik d-vel.*
- *$?X = c$ formában van, $?X \in dom(\mu)$ és $\mu(?X) = c$.*
- *$?X = ?Y$ formában van, $\{?X, ?Y\} \subseteq dom(\mu)$ és $\mu(?X) = \mu(?Y)$ fennáll.*
- *$\neg R_1$ formában van, és nem igaz, hogy $\mu \models R_1$.*
- *$R_1 \vee R_2$ formában van, és $\mu \models R_1$ vagy $\mu \models R_2$.*

- $R_1 \wedge R_2$ *formában van,* $\mu \models R_1$ *és* $\mu \models R_2$.
- $isIRI(?X), isLiteral(?X)$ *vagy* $isBlank(?X)$ *formában van, és az* $?X$ *típusa ennek megfelelően IRI, literál vagy üres csúcs.*

Egy SPARQL kifejezés vagy lekérdezés D dokumentum feletti megoldása leképezések halmazával írható le, ahol minden egyes leképezés egy lehetséges választ reprezentál. A SPARQL lekérdezés szemantikájának definiálásához a leképezéshalmazok kompakt algebráját hívjuk segítségül.

37.5. definíció. *Legyenek* $\Omega, \Omega_l, \Omega_r$ *leképezéshalmazok,* R *jelölje a szűrőfeltételt és* $S \subset V$ *legyen a változók véges halmaza. Definiáljuk az összekapcsolás* ($\bowtie$)*, unió* ($\cup$)*, különbség* ($\setminus$)*, baloldali külső összekapcsolás* ($\bowtie$)*, projekció* (π) *és szelekció* (σ) *algebrai operátorokat a következőképp:*

- $\Omega_l \bowtie \Omega_r := \{\mu_l \cup \mu_r \mid \mu_l \in \Omega_l, \mu_r \in \Omega_r : \mu_l \sim \mu_r\}$
- $\Omega_l \cup \Omega_r := \{\mu \mid \mu \in \Omega_l \text{ vagy } \mu \in \Omega_r\}$
- $\Omega_l \setminus \Omega_r := \{\mu_l \in \Omega_l \mid \forall \mu_r \in \Omega_r : \mu_l \nsim \mu_r\}$
- $\Omega_l \bowtie \Omega_r := (\Omega_l \bowtie \Omega_r) \cup (\Omega_l \setminus \Omega_r)$
- $\pi_S(\Omega) := \{\mu_1 \mid \exists \mu_2 : \mu_1 \cup \mu_2 \in \Omega \wedge dom(\mu_1) \subseteq S \wedge dom(\mu_2) \cap S = \emptyset\}$
- $\sigma_R(\Omega) := \{\mu \in \Omega \mid \mu \models R\}$

A továbbiakban ezekre az algebrai operátorokra mint ***SPARQL halmazalgebrára*** *hivatkozunk.*

A SPARQL SELECT és ASK lekérdezések kiértékelésére egy szemantikát vezetünk be, és definiáljuk a $[\![.]\!]_D$ függvényt, amely átfordítja azt halmazalgebrává.

37.6. definíció. *(****SPARQL halmazszemantika****). Legyen* D *egy RDF dokumentum,* t *egy hármas,* Q, Q_1, Q_2 *SPARQL kifejezések,* R *egy szűrőfeltétel és* $S \subset V$ *a változók halmaza. Ekkor:*

- $[\![t]\!]_D := \{\mu \mid dom(\mu) = vars(t) \text{ and } \mu(t) \in D\}$
- $[\![Q_1 \text{ AND } Q_2]\!]_D := [\![Q_1]\!]_D \bowtie [\![Q_2]\!]_D$
- $[\![Q_1 \text{ OPT } Q_2]\!]_D := [\![Q_1]\!]_D \bowtie [\![Q_2]\!]_D$
- $[\![Q_1 \text{ UNION } Q_2]\!]_D := [\![Q_1]\!]_D \cup [\![Q_2]\!]_D$
- $[\![Q \text{ FILTER } R]\!]_D := \sigma_R([\![Q]\!]_D)$
- $[\![\text{SELECT}_S(Q)]\!]_D := \pi_S([\![Q]\!]_D)$
- $[\![\text{ASK}(Q)]\!]_D := \neg(\emptyset = [\![Q]\!]_D)$

37.2. példa. Tekintsük a következő SPARQL SELECT lekérdezést:
$Q_1 := \text{SELECT}_{?SZ,?E}(((?SZ, kor, ?K) \text{ OPT } (?SZ, email, ?E)) \text{ FILTER } (?K ='' 26''))$
A lekérdezés kigyűjti az összes 26 éves személyt ($?SZ$), és opcionálisan (amennyiben létezik) a hozzájuk tartozó e-mail címeket ($?E$). Tegyük fel továbbá, hogy az adatbázisunk a következő:
$D := \{(SZ1, kor,'' 26''), (SZ2, kor,'' 25''), (SZ3, kor,'' 26''),$
$(SZ3, email,'' mail@m.com'')\}$.
Könnyű belátni, hogy $[\![Q_1]\!]_D = \{\{?SZ \mapsto SZ1\}, \{?SZ \mapsto SZ3, ?E \mapsto'' mail@m.com''\}\}$.

Halmazszemantikáról a multihalmaz szemantikára. A továbbiakban áttekintjük a kapcsolódó multihalmaz szemantikát, amit úgy kapunk, hogy a leképezések halmaza helyett a leképezések multihalmazát vesszük. A multihalmaz szemantika így abban különbözik, hogy egy leképezés többször is előfordulhat a kiértékelés során. Formálisan, a multihalmaz szemantikát leképezések multihalmazával reprezentáljuk, amely mindegyik leképezéshez multiplicitást rendel.

37.7. definíció. *Egy* ***multihalmaz*** *egy (Ω, m) páros, ahol az Ω egy leképezéshalmaz és $m : \mathcal{M} \to \mathbb{N}_0$ egy totális függvény, $m(\mu^+) \geq 1$ minden $\mu^+ \in \Omega$-ra és $m(\mu^-) = 0$ minden $\mu^- \notin \Omega$ esetén. Legyen adott $\mu^+ \in \Omega$, ekkor az $m(\mu^+)$ a μ^+* ***multiplicitása*** *Ω-ban, valamint azt mondjuk, hogy μ^+ $m(\mu^+)$-szor fordul elő Ω-ban.*

A multihalmaz szemantikát könnyen formalizálhatjuk a 37.5. definícióban leírt algebrai operátorok adaptálásával úgy, hogy multihalmazokkal dolgozunk, és figyelembe vesszük a halmaz elemeinek multiplicitását. Tekintsük példaként a következő unió operátort multihalmazok felett: $(\Omega_l, m_l) \cup (\Omega_r, m_r)$, melynek eredménye $(\Omega_l \cup \Omega_r, m)$, ahol $m(\mu) = m_l(\mu) + m_r(\mu)$ minden $\mu \in \mathcal{M}$-re. Ezt az algebrát ***SPARQL multihalmaz-algebrának*** nevezzük. A multihalmaz szemantika pontos definíciója közvetlenül megkapható a 37.6. definíció első szabályának (t hármas esete) módosításával úgy, hogy multihalmazzal tér vissza halmaz helyett. A $[\![.]\!]^+_D$ függvényt használjuk azon leképezések multihalmazának jelölésére, amelyek során a SPARQL kifejezést vagy lekérdezést multihalmaz szemantikával értékeltük ki.

37.3. példa. Legyen $Q := (?X, c, c) \text{ UNION } (c, c, ?X)$, a dokumentum pedig $D := \{(c, c, c)\}$, valamint $\mu := \{?X \mapsto c\}$. Ekkor $[\![Q]\!]^+_D = (\{\mu\}, m)$, ahol $m(\mu) := 2$, és $m(\mu') := 0$ minden $\mu' \in \mathcal{M} \setminus \{\mu\}$ esetén.

37.2. A SPARQL lekérdezőnyelv bonyolultsági kérdései

A következő alfejezetben a SPARQL különböző résznyelveihez tartozó kiértékelési problémák bonyolultságával foglalkozunk. Kiderül, hogy a kérdés gyakorlati szempontból hamar kezelhetetlenné, NP-teljessé válik. Az opcionalitást lehetővé tevő OPT művelet bevezetése ráadásul tovább növeli a komplexitást általában PSPACE-teljessé téve a problémát. Az opcionalitás megjelenésének tehát megvan a maga ára, ráadásul a bevezetés mögött meghúzódó intuitív elképzelés sem minden esetben teljesül maradéktalanul. Azt várnánk ugyanis, hogy opcionális részkérdések alkalmazásával úgy szerezhetünk plusz információkat, hogy azokat az adatokat nem kell kihagynunk a megoldásból, ahol ez a többlet nem áll rendelkezésre. Az alfejezet második felében két szintaktikai megszorítás kerül bevezetésre, amelyek egyrészt polinom időben ellenőrizhetőek a lekérdezés méretében, másrészt biztosítják a fenti intuíció teljesülését. Az ily módon kapott ún. jól tervezett lekérdezések kiértékelési bonyolultsága szintén érdekes probléma, ráadásul az SQL lekérdezések optimalizációjánál alkalmazható egyszerű átírási technikák megfelelői működnek ebben a környezetben is, míg általános esetben ezek nem bizonyulnak használhatónak.

Amikor a kiértékelési kérdés bonyolultságáról beszélünk, formálisan a következő problémát szeretnénk megoldani:

INPUT: D RDF gráf, P SPARQL lekérdezés és μ leképezés
OUTPUT: $\mu \in [\![P]\!]_D$

Azaz, adott D RDF gráf, P SPARQL lekérdezés és μ leképezés esetén arra vagyunk kíváncsiak, hogy a μ leképezés eleme-e a D fölött végrehajtott P lekérdezés eredményhalmazának. Vegyük észre, hogy itt maga a lekérdezés is eleme a bemenetnek, a probléma ezen változatára a lekérdezőnyelv *összetett kiértékelési bonyolultságaként* szoktak hivatkozni, szemben a *kiértékelés adatbonyolultságával* , ahol a lekérdezés nem része a bemenetnek. Mindazonáltal a fejezetben egy adatbonyolultságra vonatkozó eredménnyel is találkozunk majd.

37.2.1. AND és FILTER műveletek

Amikor az iménti kiértékelési bonyolultságra vonatkozó feladatban a P lekérdezés csupán AND és FILTER műveleteket tartalmaz, a probléma relatíve egyszerű. Először meg kell nézni, hogy a P-ben szereplő t hármasokra teljesüle, hogy ha a t-beli változók helyére behelyettesítjük azok μ által adott értékeit, akkor a D-ben szereplő RDF hármast kapunk-e. Ha valamelyik hármasra ez

a feltétel nem áll fenn, akkor μ egész biztosan nincs benne a P lekérdezés D fölött vett eredményhalmazában. Ellenkező esetben P kifejezésfáján alulról felfelé haladva meg kell néznünk, hogy a változók μ szerint vett helyettesítései mellett a FILTER szűrések feltételei teljesülnek-e. Ha igen, μ eleme az előbbi eredményhalmaznak, különben pedig nem.

A következő tétel bizonyítható a fenti gondolatmenet formális leírásával, ami az elmondottak alapján már semmiféle nehézséget nem rejt magában:

37.8. tétel. *A csak* AND *és* FILTER *műveleteket tartalmazó SPARQL lekérdezések összetett kiértékelési bonyolultsága* $O(|P||D|)$*, ahol a korábbi jelölések alkalmazásával* P *a SPARQL lekérdezést,* D *pedig azt az RDF adathalmazt jelöli, amely fölött* P*-t kiértékeljük.*

37.2.2. AND, FILTER és UNION műveletek

Hasonlóan az ítéletlogika kielégíthetőségi problémájához, ahol a diszjunkció megjelenésének köszönhetően a feladat NP-teljessé válik, az UNION művelet a SPARQL esetén is NP-teljessé teszi az összetett kiértékelési kérdést.

37.9. tétel. *Ha a SPARQL lekérdezésekben csak az* AND *és* UNION *műveleteket engedjük meg, akkor az összetett kiértékelés probléma NP-teljessé válik.*

Bizonyítás. Csak az NP-nehézséget bizonyítjuk. A kérdésünket a halmazlefedési feladatra vezetjük vissza, amely közismerten NP-teljes. Itt adott

$$U = \{u_1, \ldots, u_n\}$$

alaphalmaz, $S_1, \ldots, S_k$ U feletti részhalmazok és l pozitív egész esetén arra vagyunk kíváncsiak, ki lehet-e választani legfeljebb l darabot $S_1, \ldots, S_k$ halmazok közül úgy, hogy ezek uniója U-val legyen azonos.

A visszavezetéshez a $D = \{(c, c, c)\}$ egyetlen RDF hármasból álló gráfot használjuk. Az $S_i = \{u_1, \ldots, u_m\}$ halmazt a

$$P_{S_i} := (c, c, ?U_1)\,\text{AND}\ldots\text{AND}\,((c, c, ?U_m))$$

lekérdezés reprezentálja. P_{S_i} D fölött nyilvánvalóan azt a leképezést eredményezi, amelyik mindegyik $?U_j$ változóhoz a c-t rendeli ($1 \leq j \leq m$). Az S_i részhalmazok S halmazát, $S = \{S_1, \ldots, c, S_k\}$, a

$$P_s := P_{S_1}\,\text{UNION}\ldots\text{UNION}\,P_{S_k}$$

lekérdezés szimulálja. Ennek D fölött vett eredményébe értelemszerűen azok a leképezések tartoznak, amelyek valamelyik P_{S_i} D feletti eredményhalmazába

beletartoznak. Mindennek ismeretében, ha megvizsgáljuk a

$$P = P_S \,\text{And}\, \ldots \,\text{And}\, P_S$$

lekérdezést, ahol P_S pontosan l alkalommal jelenik meg, akkor könnyen látható, hogy fenti halmazlefedési feladat akkor és csak akkor ad igaz eredményt, ha a $\mu := \{?U_1 \to c, \ldots, ?U_n \to c,\}$ leképezés eleme a P D fölött vett kiértékelésének. ■

Annak bizonyítását, hogy a kizárólag And és Union műveleteket tartalmazó SPARQL lekérdezések esetén az összetett kiértékelési bonyolultság NP-ben van, feladatként adjuk fel, hasonlóan az alábbi állítások bizonyításához.

37.10. tétel. *(i) A kizárólag* Union *és* Filter *műveleteket tartalmazó SPARQL lekérdezések esetén az összetett kiértékelési probléma polinom időben megoldható. (ii) Emellett az* And, Union *és* Filter *műveletek estén az iménti probléma NP-teljes marad, azaz a* Filter *művelet bevezetése nem növeli már a bonyolultságot.*

Bizonyítás. Megtalálható a [295] cikkben. ■

37.2.3. Az Opt művelet

Intuitíven, adott D RDF adathalmaz $P := (P_1 \,\text{Opt}\, P_2)$ lekérdezés esetén egy μ leképezés akkor és csak akkor tartozik a P D felett vett eredményhalmazába, (i) ha μ a $P_1 \,\text{And}\, P_2$ D fölött vett eredményhalmazának eleme, jelöléssel $\mu \in [\![P_1 \,\text{And}\, P_2]\!]_D$, vagy (ii) ha μ eleme a P_1 D feletti eredményhalmazának, $\mu \in [\![P_1]\!]_D$, ugyanakkor nem kompatibilis egyetlen olyan μ' leképezéssel sem, amely P_2 D felett vett eredményhalmazához tartozik, $\mu' \in [\![P_2]\!]_D$. A témakörben járatosak azonnal érzékelhetik, hogy a definíció a kvantorok alternálásának lehetőségét rejti magában, hiszen az (i) feltétel tulajdonképpen egy egzisztenciális kvantort tartalmaz, míg a (ii) feltétel egy univerzálisat. A változók alternálásának lehetősége pedig azért érdekes, mert a $\exists x_1 \forall x_2 \ldots Q_n x_n \phi$ zárt formula igazságértékének eldöntése PSPACE-teljes feladat, ahol ϕ olyan ítéletlogikai formula, melyben csak az $x_1, \ldots, cx_n$ logikai változók fordulnak elő, emellett Q_n az egzisztenciális kvantort jelöli, ha n páratlan, és az univerzális kvantort különben. Így az a sejtésünk támadhat, hogy az Opt művelet bevezetése tovább növeli a kiértékelési feladat bonyolultságát és PSPACE nehézzé teszi. A [295] cikkben bebizonyították a feltételezés helyességét, pontosabban a következő lemmát:

37.11. lemma. *Azon SPARQL lekérdezések esetében, melyek kizárólag az* Opt *műveletet tartalmazzák, az összetett kiértékelési feladat bonyolultsága PSPACE-nehéz.*

EVAL(μ: lekérdezés, P SPARQL lekérdezés, D RDF gráf)
case:
  P a t hármas:
    if a μ értelmezési tartománya a t-beli változók halmaza
      és $\mu(t)$ D-nek eleme **then return** *true*
    return *false*
  P P_1 FILTER R alakú:
    if EVAL(μ, P_1, D)= *true* és μ eleget tesz R feltételeinek
      then return *true*
    return *false*
  P P_1 UNION R alakú:
    IF EVAL(μ, P_1, D)= *true* vagy EVAL(μ, P_2, D)= *true*
      then return *true*
    return *false*
  P P_1 AND R alakú:
    for each μ_1, μ_2 leképezésre, ahol μ_i P_i változóihoz D értékeit rendeli
    hozzá $(i = 1, 2)$
      if EVAL(μ_1, P_1, D)= *true* és EVAL(μ_2, P_2, D)= *true* és $\mu = \mu_1 \cup \mu_2$
        then return *true*
    return *false*
  P P_1 OPT R alakú:
    if EVAL(μ, P_1 AND P_2, D)= *true* **then return** *true*
    if EVAL(μ, P_1 D)= *true* **then**
      for each μ' leképezésre, amely P_2 változóihoz D elemeit rendeli
        if EVAL(μ', P_2 D)= *true* és μ kompatibilis μ'-vel **then return** *false*
      return *true*
    return *false*

37.1. ábra. AZ AND, UNION, OPT ÉS FILTER MŰVELETEKET TARTALMAZÓ SPARQL LEKÉRDEZÉSEKET KIÉRTÉKELŐ EVAL ALGORITMUS.

A másik oldalról viszont a PSPACE-beliség, azaz annak megmutatása, hogy az előző lemmában szereplő lekérdezések kiértékelése polinom tárat használó Turing-géppel megoldható, már-már triviális. Sőt, ennél erősebb állítás is bizonyítható.

37.12. lemma. *Az* AND, UNION, OPT *és* FILTER *műveleteket tartalmazó SPARQL lekérdezések összetett kiértékelési feladata polinomiális méretű tárban megoldható.*

Bizonyítás. Könnyű belátni – hiszen csak a műveletek definícióit kell alkalmazni a megfelelő részeknél –, hogy a 37.1. ábrán szereplő algoritmus valóban a kiértékelési problémát válaszolja meg azon P SPARQL lekérdezések esetében, melyek AND, UNION, OPT és FILTER műveleteket tartalmazhatnak. Más szavakkal eldönti, hogy a további bemenetként adott D RDF gráf és μ leképezés esetén $\mu \in [\![P]\!]_D$ teljesül-e.

Emellett, mivel P változóinak halmaza $O(\log|P|)$, és D $O(\log|D|)$ nagyságú tárban már elfér, egy olyan leképezés, amely P változóihoz D elemeit rendeli, tárolható $O(|P|\log|P|\log|D|)$ nagyságú helyen, hiszen egy változó és a neki megfelelő érték elfér $O(\log|P|\log|D|)$ nagyságú tárterületen, az ilyen párosok száma pedig P változóinak darabszámával egyezik meg. Így az EVAL algoritmus egyszeri meghívásának eredménye polinom méretű tárhelyen biztosan kiszámítható. Az viszont ismételten nyilvánvalóan igaz, hogy az EVAL algoritmus legfeljebb $|P|$ alkalommal hívhatja meg önmagát. Összességében tehát a teljes algoritmus eredménye polinom tárhelyen kiszámítható. ■

37.13. következmény. *Az* AND, UNION, OPT *és* FILTER *műveleteket tartalmazó SPARQL lekérdezések kiértékelési feladatának adatbonyolultsága LOG-SPACE-beli.*

Bizonyítás. Emlékezzünk vissza, hogy a kiértékelési feladat esetén a kiértékelendő SPARQL lekérdezés méretét konstansnak tekintjük. A 37.12. lemma bizonyításánál beláttuk, hogy P, D, μ bemenetekkel az EVAL algoritmus $O(|P|^2\log(|P|)\log(|D|))$ tárhelyen megvalósítható, ahol P egy SPARQL lekérdezés – AND, UNION, OPT és FILTER műveletekkel –, D egy RDF gráfot, míg μ egy leképezést jelöl. Ha P méretét konstansnak vesszük, az iménti képlet szerint az algoritmus $O(\log(|D|))$ helyen végrehajtható. ■

A 37.11. és a 37.12. lemmákból pedig egyszerűen adódik a következő tétel.

37.14. tétel. *Bármely olyan SPARQL nyelvcsalád esetében, melynél az* OPT *művelet mellett az* AND, UNION, FILTER *műveletek közül néhány használata megengedett, az összetett kiértékelési probléma PSPACE-teljes.*

Bizonyítás. Megtalálható a [295] cikkben. ■

37.2.4. Az unió-normálforma

A P_1 és P_2 SPARQL lekérdezéseket akkor nevezzük ***ekvivalensnek***, ha minden D RDF gráf és μ leképezés mellett $\mu \in [\![P_1]\!]_D$ akkor és csak akkor teljesül, ha $\mu \in [\![P_2]\!]_D$ is teljesül. Nem nehéz belátni, a részletek leírása viszont hosszadalmas, hogy minden AND, FILTER, UNION, OPT műveleteket használó

SPARQL lekérdezés átalakítható olyan lekérdezéssé, ami olyan SPARQL lekérdezések uniójából áll, melyekben már nem szerepel a UNION művelet. Formálisan:

37.15. tétel. *Legyen P tetszőleges SPARQL lekérdezés, amely* AND, FILTER, UNION, OPT *műveleteket tartalmazhat. Ekkor létezik P-vel ekvivalens*

$$P' = P_1 \text{ UNION } \ldots \text{ UNION } P_k$$

lekérdezés, ahol P_i olyan SPARQL lekérdezés, melyben már nem szerepel a UNION *művelet* $(1 \leq i \leq k)$. *P'-t P* unió normálformájának *nevezzük.*

Bizonyítás. Megtalálható a [295] cikkben. ∎

37.16. tétel. *Ha egy* AND, UNION *és* FILTER *műveleteket tartalmazó P lekérdezés unió normálformában van, azaz unió normálformája megegyezik P-vel, akkor az összetett kiértékelési feladat $O(|P||D|)$ időben megoldható.*

Bizonyítás. Az állítás a 37.8. tétel egyszerű következménye. ∎

A tétel azt sugallja, hogy az unió normálforma akár exponenciális méretűvé is duzzadhat az eredeti mérethez képest, hiszen a 37.10. tétel értelmében az összetett kiértékelési probléma az AND, UNION és FILTER műveleteket tartalmazó SPARQL lekérdezéseknél általános esetben NP-teljes, míg unió normálformájú lekérdezés esetén polinom időben megválaszolható a lekérdezés méretében.

37.2.5. Jól tervezett SPARQL lekérdezések

Az OPT (optional) művelet segítségével a SPARQL nyelv megalkotói azt a lehetőséget akarták megteremteni, hogy ha egy információ elérhető, kerüljön bele a végeredménybe, ugyanakkor annak hiánya ne jelentse azon adatok elhagyását, ahol nem rendelkezünk az iménti többlet információval. Ez az intuíció azonban bizonyos, speciális lekérdezések esetén nem teljesül.

37.4. példa. Vegyük ugyanis a következő példát:

$$P = ((?\text{X, cím, Tüskevár}) \text{ OPT } ((?\text{Y, cím, Vuk}) \text{ OPT } (?\text{X, kiadó, ?Z}))),$$

ahol D egy könyvek adatait tartalmazó RDF gráf (ebben a szakaszban a literálok a jobb olvashatóság kedvéért nincsenek idézőjelek közé téve):

(K_1, cím, Harmonia Caelestis),	(K_4, kiadó, Európa)
(K_2, cím, Tüskevár),	(K_1, szerző, Esterházy Péter)
(K_3, cím, Bádogdob),	(K_4, szerző, Bodon Ádám)
(K_4, cím, Sinistra körzet),	(K_3, ár, 3500)
(K_2, kiadó, Móra),	(K_4, ár, 2600)

Itt $[\![P]\!]_D = \{(?X \to K_2)\}$, annak ellenére, hogy a K_2 azonosítójú könyv esetén ismerjük a kiadót is. A (?X, kiadó, ?Z) opcionális rész azonban a (?Y, cím, Vuk) hármashoz tartozik, aminek viszont üres halmaz az eredménye a D fölött, ily módon az ((?Y, cím, Vuk) OPT (?X, kiadó, ?Z)) részkérdés eredménye is üres halmaz. Más szavakkal: a plusz információt szolgáltató kérdésrészlet (?X, kiadó, ?Z) nem a (?X, cím, Tüskevár) hármas után következett közvetlen, aminek eredményéhez a további adatokat szolgáltathatta volna, s így a „közbeékelődő" hármas el tudta „rontani" a várt eredményt.

A példa mögött egy általánosabb jelenség húzódik meg, a problémát az okozhatja, ha a P lekérdezés egy $P' = (P_1 \,\text{OPT}\, P_2)$ részkérdésénél az $?X$ változó a P_2-ben és P'-n kívül is előfordul, P_1-ben azonban nem. A jól tervezettséget az ilyen mintázatok lehetőségeinek megszüntetése jelenti majd. Ám emellett egy másik hibalehetőségre is oda kell figyelni. Egy Q SPARQL lekérdezést *biztonságosnak* nevezünk, ha minden $(P \,\text{FILTER}\, R)$ részkérdésére teljesül, hogy az R feltételben csak olyan változók fordulnak elő, amelyek P-ben is megjelennek.

37.17. definíció. *Egy* AND, OPT, FILTER *műveleteket tartalmazó, azaz unió mentes P SPARQL lekérdezés* jól tervezett, *ha (i) P biztonságos, (ii) P minden $P' = (P_1 \,\text{OPT}\, P_2)$ részkérdése esetében, ha egy változó szerepel P_2-ben és P'-n kívül is, akkor P_1-ben is elő kell fordulnia.*

Könnyen belátható, hogy a jól tervezettség ellenőrzése a lekérdezés méretéhez képest polinom időben megoldható.

A következő megfigyelés szintén a „jól tervezettség", mint tulajdonság bevezetésének jogosságát indokolja. Intuitíven ugyanis, ha a P' SPARQL lekérdezés a P lekérdezésből néhány opcionális részkérdés törlésével keletkezik, akkor azt várnánk, hogy a P' tetszőleges D RDF gráf fölötti vett eredménye nem tartalmaz több adatot, mint a P végeredménye ugyanezen D RDF gráf fölött. Másképpen fogalmazva az opcionális részek csak bővíthetik a megoldást. Ez azonban általában nem teljesül. Vegyük például a következő egyszerű adatgráfot:

$$D := \{(1, a, 1),\ (2, a, 2),\ (3, a, 3)\}$$

és a

$$P := ((?X, a, 1) \,\text{OPT}\, ((?Y, a, 2) \,\text{OPT}\, (?X, a, 3)))$$

lekérdezést. Itt a P D fölötti kiértékelése egyetlen leképezést tartalmaz: $[\![P]\!]_D = \{(?X \to 1)\}$. Ugyanakkor, a $P' = ((?X, a, 1) \,\text{OPT}\, ((?Y, a, 2))$ lekérdezés esetén, melyet az $(?X, a, 3)$ opcionális rész törlésével kapunk P-ből,

$$[\![P']\!]_D = \{(?X \to 1, ?Y \to 2)\},$$

azaz ha P'-höz hozzávesszük az imént kidobott opcionális részt, információt veszítünk, ahelyett, hogy nőne az ismeretünk.

A jól tervezett lekérdezések esetében az iménti anomáliával egészen biztosan nem találkozhatunk. Formálisan: azt mondjuk, hogy P' lekérdezés P SPARQL lekérdezés *redukáltja*, ha P-ből megkaphatjuk a $(P_1 \,\textsc{Opt}\, P_2)$ részkérdés P_1-gyel történő helyettesítésével. Például a

$$P' = t_1 \,\textsc{And}\, (t_2 \,\textsc{Opt}\, t_3)$$

lekérdezés a $P' = t_1 \,\textsc{And}\, (t_2 \,\textsc{Opt}\, (t_3 \,\textsc{Opt}\, t_4))$ lekérdezés redukáltja. A reláció reflexív, tranzitív lezártját $\trianglelefteq$ jelöli. Ennek értelmében például $P \trianglelefteq (t_1 \,\textsc{And}\, t_2)$ teljesül.

37.18. tétel. *Legyen P jól tervezett SPARQL lekérdezés. Ekkor tetszőleges P' SPARQL lekérdezésre, ahol $P \trianglelefteq P'$ fennáll, $[\![P']\!]_D \subseteq [\![P]\!]_D$ is teljesül, azaz az opcionális részek hozzáadása csak bővítheti az eredményt.*

Bizonyítás. Megtalálható a [295] cikkben. ∎

A jól tervezett SPARQL lekérdezések kiértékelési problémájának bonyolultságáról a következő mondható el.

37.19. tétel. *(i) Az* And *és* Opt *műveleteket tartalmazó, jól tervezett SPARQL lekérdezések összetett kiértékelési problémája coNP-teljes. (ii) A* Filter *művelet alkalmazási lehetősége nem növeli a bonyolultságot.*

37.20. következmény. *A $P = P_1 \,\textsc{Union}\, \ldots \,\textsc{Union}\, P_k$ unió formában lévő SPARQL lekérdezések esetében, ahol minden P_i jól tervezett $(1 \leq i \leq k)$, az összetett kiértékelési probléma coNP-teljes.*

Más szavakkal, a jól tervezettség „csökkenti” a kiértékelési probléma bonyolultságát. A tulajdonság további előnye, hogy nagymértékű optimalizációt is lehetővé tesz. Az SQL lekérdezések esetében tudniillik jelentős időt lehet megtakarítani, ha a szűrő feltételeket minél hamarabb végrehajtjuk. Például az `Autó(id, márka, ár)` és `Tulaj(id, autó_id, név, kor)` táblák esetében a

```
SELECT név
FROM Autó a, Tulaj
WHERE a.id = autó_id AND ár > 2000000;
```

lekérdezést, ami a 2 millió forintnál drágább kocsik tulajdonosának nevét adja meg, úgy érdemes végrehajtani, hogy először az Autó táblából elhagyjuk a 2 milliónál olcsóbb vagy pontosan ennyibe kerülő autókat és csupán a maradékot kapcsoljuk össze a Tulaj táblával ahelyett, hogy rögtön az Autó és Tulaj tábla összekapcsolásával kezdenénk.

Hasonló optimalizálási lehetőség a jól tervezett SPARQL lekérdezések esetében is adódik.

37.21. tétel. *Legyenek P_1, P_2 és P_3 SPARQL lekérdezések és R egy beépített feltétel. Tekintsük az alábbi átírási szabályokat:*

1. $((P_1 \,\textsc{Opt}\, P_2) \,\textsc{Filter}\, R) \to ((P_1 \,\textsc{Filter}\, R) \,\textsc{Opt}\, P_2)$,
2. $((P_1 \,\textsc{And}\, (P_2 \,\textsc{Opt}\, P_3) \to ((P_1 \,\textsc{And}\, P_2) \,\textsc{Opt}\, P_3)$,
3. $((P_1 \,\textsc{Opt}\, P_2) \,\textsc{And}\, P_3) \to ((P_1 \,\textsc{And}\, P_3) \,\textsc{Opt}\, P_2)$.

Ekkor P jól tervezett SPARQL lekérdezések esetében, ha P'-t az iménti szabályok alkalmazásával kapjuk P-ből, akkor P' a P-vel ekvivalens SPARQL lekérdezés, ráadásul jól tervezett.

A bizonyítás ismételten csak nem nehéz, de hosszadalmas. Kivéve a 3. szabály esetében, hiszen az a 2. szabályból megkapható az And művelet kommutativitásának felhasználásával.

Az átírási szabályok hasznát könnyű belátni. A $(P_1 \,\textsc{Opt}\, P_2)$ tetszőleges D RDF gráf fölötti kiértékelésekor a P_1 D fölötti kiértékelésekor kapott leképezések száma biztosan nem csökken. Ugyanakkor az And és Filter műveletek alkalmazása csökkentheti az eredmény méretét. Ebből az következik, hogy ezeket a műveleteket érdemes minél hamarabb végrehajtani, a 37.21. tétel szabályai pedig pontosan ezt teszik lehetővé.

Érdemes észrevenni, hogy az iménti szabályok általános esetben ismételten csak nem teljesülnek. Tekintsük ugyanis újfent a

$$D := \{(1, a, 1), (2, a, 2), (3, a, 3)\}$$

RDF gráfot és a

$$P := ((?X, a, 1) \,\textsc{And}\, ((?Y, a, 2) \,\textsc{Opt}\, (?X, a, 3)))$$

lekérdezést. Itt P a D fölött az üres halmazt eredményezi. A 2. szabály alkalmazásával kapott

$$P' := (((?X, a, 1) \,\textsc{And}\, (?Y, a, 2)) \,\textsc{Opt}\, (?X, a, 3))$$

lekérdezés esetén az eredmény a $(?X \to 1, ?Y \to 2)$ leképezést tartalmazó halmaz, ami nyilvánvalóan különbözik az üres halmaztól.

Gyakorlatok

37.2-1. Lássuk be a 37.10. tétel (i) állítását, miszerint a UNION és FILTER műveleteket tartalmazó SPARQL lekérdezéseknél az összetett kiértékelési probléma polinom időben megoldható.

37.2-2. Adjunk meg egy olyan algoritmust, ami a bemenetként kapott AND, UNION, OPT, FILTER műveleteket tartalmazó tetszőleges SPARQL lekérdezést unió-normálformára hozza.

37.2-3. Ellenőrizzük, hogy a 37.21. tétel állításai multihalmaz szemantika mellett teljesülnek-e.

37.2-4. Adjunk meg egy RDF gráfot és két olyan SPARQL lekérdezést, amelyekre rendre nem teljesülnek a 37.21. tétel (1) és (3) átírási szabályai.

37.3. A SPARQL optimalizálása

A SPARQL – az adatbázisok világában jól ismert SQL-hez hasonlóan – deklaratív nyelv: a lekérdezések (adott szemantika mellett) pontosan meghatározzák, hogy az adott dokumentumon kiértékelve milyen eredményt kell kapnunk. A kiértékelés módja, a végrehajtás lépései azonban nincsenek a lekérdezésben meghatározva: ezek előállítása az adatbázismotor feladata. Egy lekérdezést általában többféleképpen is meg lehet fogalmazni (a jelentés változtatása nélkül), és minden megfogalmazáshoz is többféle kiértékelés tartozhat, akár nagyságrendben eltérő hatékonysággal. Az optimalizálás feladata, hogy a bemenetként kapott lekérdezéshez találjon egy olyan átfogalmazást és/vagy kiértékelési stratégiát, amellyel a feladat minél hatékonyabban oldható meg. A lekérdezés-optimalizálás nem egzakt feladat, nem cél minden esetben a létező legjobb alternatíva megtalálása, hiszen ez bizonyos esetekben költségesebb is lehet, mint maga a lekérdezés végrehajtása.

A jelen alfejezetben tárgyalt eredmények a [321] cikkből származnak. Ennek keretében az optimalizálás két fajtájával foglalkozunk: a 37.3.1. szakasz az algebrai optimalizálásról szól, melynek során a halmaz (vagy multihalmaz) szemantika algebrai operátorainak bizonyos tulajdonságait használjuk ki arra, hogy az operátorok átrendezésével, cseréjével, vagy elhagyásával hatékonyabban kiértékelhető kifejezést kapjunk. A 37.3.2. szakasz pedig a szemantikus optimalizálásról szól, ahol az adathalmazra (dokumentumra) vonatkozó többlet ismeretet használunk ki a végrehajtás költségeinek javítására.

37.3.1. Algebrai optimalizálás

Az algebrai optimalizálás során egy SPARQL algebrai kifejezéshez keresünk olyan kifejezéseket, amelyeket kiértékelve az eredetivel egyező eredményt kapunk, azért, hogy ezek közül kiválaszthassuk azt, amelyik várhatóan a leggyorsabban végrehajtható.

37.22. definíció. *Legyen A_1 és A_2 két halmaz (vagy multihalmaz) algebrai kifejezés. Azt mondjuk, hogy A_1 és A_2 kifejezések* ***ekvivalensek*** *(jelölés: $A_1 \equiv A_2$), ha minden D dokumentumon ugyanazt az eredményt adják.*

Az optimalizálandó kifejezéssel ekvivalens kifejezéseket ún. átírási szabályok egymás utáni alkalmazásával keressük. Egy átírási szabály két kifejezés (a szabály bal és jobb oldala) közötti ekvivalenciát mond ki. Mivel az ekvivalencia szimmetrikus, a szabályok bal és jobb oldala felcserélhető, azaz mindkét irányban alkalmazhatóak. A szabályokban szereplő kifejezések általában egyszerű felépítésűek, így elég általánosak ahhoz, hogy sok helyzetben alkalmazhatóak legyenek. A halmaz- és multihalmaz szemantika esetén más-más átírási szabályok léteznek. A továbbiakban a halmazszemantika átírási szabályairól beszélünk, azonban a későbbiekben látni fogjuk, hogy a legtöbb szabály (esetleg kis módosítással) átvihető a multihalmaz szemantikára is.

A szabályok tárgyalása előtt szükséges még két függvény bevezetése, amelyek a kifejezésekben szereplő változókat osztályozzák statikusan aszerint, hogy tetszőleges dokumentumon vett eredményben kapnak-e minden esetben értéket, vagy sem. Ezt statikusan csak közelítőleg vizsgáljuk: bevezetünk egy alsó- és egy felső korlátot a mindig kötött változók halmazára. Az alsó korlátot a $cVars$ (certain variables) függvény adja.

37.23. definíció. *Legyen A egy SPARQL halmazalgebrai kifejezés. Ekkor a $cVars(A)$ értéke az alábbi rekurzív módon számítható ki:*

- *Ha $A = [\![t]\!]_D$, akkor $cVars(A) = vars(t)$.*
- *Ha $A = A_1 \bowtie A_2$, akkor $cVars(A) = cVars(A_1) \cup cVars(A_2)$.*
- *Ha $A = A_1 \cup A_2$, akkor $cVars(A) = cVars(A_1) \cap cVars(A_2)$.*
- *Ha $A = A_1 \setminus A_2$, akkor $cVars(A) = cVars(A_1)$.*
- *Ha $A = \pi_S(A_1)$, akkor $cVars(A) = cVars(A_1) \cap S$.*
- *Ha $A = \sigma_R(A_1)$, akkor $cVars(A) = cVars(A_1)$.*

Megjegyzés: a fenti definícióban az $A = A_1 \ltimes A_2$ eset nincs követlenül feltüntetve, ugyanis ez a 37.5. definíció alapján átírható az $\bowtie$, $\cup$, és $\setminus$ operátorokat használó kifejezéssé.

A $cVars(A)$ alsó korlát jellegét az adja, hogy az ebben szereplő változók az eredmény minden leképezésében biztosan kötöttek. Ezt fogalmazza meg az alábbi állítás.

37.24. állítás. *Legyen A egy halmazalgebrai kifejezés és jelölje Ω_A a leképezések halmazát, amiket az A kifejezés D dokumentumon való kiértékelésével kaptunk. Ekkor ha $?X \in cVars(A)$, akkor $\forall \mu \in \Omega_A : ?X \in dom(\mu)$.*

Hasonlóan bevezethetünk a kötött változókra egy felső korlátot is, ezt a $pVars$ (possible variables) függvény adja meg.

37.25. definíció. *Legyen A egy SPARQL halmazalgebrai kifejezés. Ekkor a $pVars(A)$ értéke az alábbi rekurzív módon számítható ki:*

- *Ha $A = [\![t]\!]_D$, akkor $pVars(A) = vars(t)$.*
- *Ha $A = A_1 \bowtie A_2$, akkor $pVars(A) = pVars(A_1) \cup pVars(A_2)$.*
- *Ha $A = A_1 \cup A_2$, akkor $pVars(A) = pVars(A_1) \cup pVars(A_2)$.*
- *Ha $A = A_1 \setminus A_2$, akkor $pVars(A) = pVars(A_1)$.*
- *Ha $A = \pi_S(A_1)$, akkor $pVars(A) = pVars(A_1) \cap S$.*
- *Ha $A = \sigma_R(A_1)$, akkor $pVars(A) = pVars(A_1)$.*

Megjegyzés: a $pVars$ definíciója nagyon hasonló a $cVars$ definíciójához, az egyetlen különbség az, hogy $A_1 \cup A_2$ esetnél a jobb oldalon $\cap$ helyett $\cup$ szerepel.

Hasonlóan az előbbiekhez, itt is egy állítással igazoljuk a felső korlát jelleget.

37.26. állítás. *Legyen A egy halmazalgebrai kifejezés, és jelölje Ω_A a leképezések halmazát, amelyeket az A kifejezés D dokumentumon való kiértékelésével kaptunk. Ekkor, ha $\forall \mu \in \Omega_A : ?X \in dom(\mu)$, akkor $?X \in pVars(A)$.*

A szükséges fogalmak ismertetése után most nézzük az átírási szabályokat, a legegyszerűbbekkel kezdve.

37.27. tétel. *Legyenek A, A_1, A_2 és A_3 tetszőleges halmazalgebrai kifejezések. Ekkor az alábbi átírási ekvivalencia szabályok érvényesek.*

$$A \cup A \equiv A \tag{37.1}$$

$$A \setminus A \equiv \emptyset \tag{37.2}$$

$$(A_1 \cup A_2) \cup A_3 \equiv A_1 \cup (A_2 \cup A_3) \tag{37.3}$$

$$(A_1 \bowtie A_2) \bowtie A_3 \equiv A_1 \bowtie (A_2 \bowtie A_3) \tag{37.4}$$

$$A_1 \cup A_2 \equiv A_2 \cup A_1 \tag{37.5}$$

$$A_1 \bowtie A_2 \equiv A_2 \bowtie A_1 \tag{37.6}$$

$$(A_1 \cup A_2) \bowtie A_3 \equiv (A_1 \bowtie A_3) \cup (A_2 \bowtie A_3) \tag{37.7}$$

$$A_1 \bowtie (A_2 \cup A_3) \equiv (A_1 \bowtie A_2) \cup (A_1 \bowtie A_3) \tag{37.8}$$

$$(A_1 \cup A_2) \setminus A_3 \equiv (A_1 \setminus A_3) \cup (A_2 \setminus A_3) \tag{37.9}$$

$$(A_1 \cup A_2) ⟕ A_3 \equiv (A_1 ⟕ A_3) \cup (A_2 ⟕ A_3) \tag{37.10}$$

$$(A_1 \setminus A_2) \setminus A_3 \equiv (A_1 \setminus A_3) \setminus A_2 \tag{37.11}$$

$$(A_1 \setminus A_2) \setminus A_3 \equiv A_1 \setminus (A_2 \cup A_3) \tag{37.12}$$

$$A_1 \setminus A_2 \equiv A_1 \setminus (A_1 \bowtie A_2) \tag{37.13}$$

Bizonyítás. Megtalálható a [321] cikkben. ■

A fenti ekvivalenciák tekinthetőek a SPARQL halmazalgebra alaptörvényeinek is, mivel az operátorok alapvető tulajdonságait (kommutativitás, asszociativitás, disztributivitás) fejezik ki.

A következő csoportba a projekció átírásával kapcsolatos szabályok tartoznak.

37.28. tétel. *Legyenek A, A_1 és A_2 tetszőleges halmazalgebra kifejezések, S, S_1, $S_2 \subset V$ változók halmazai, és legyen R egy szűrőfeltétel. Továbbá legyen $S'' = pVars(A_1) \cap pVars(A_2)$, valamint $S' = S \cup S''$. Ekkor az alábbi átírási szabályok teljesülnek.*

$$\pi_{pVars(A) \cup S}(A) \equiv A \tag{37.14}$$

$$\pi_S(A) \equiv \pi_{S \cap pVars(A)}(A) \tag{37.15}$$

$$\pi_S(\sigma_R(A)) \equiv \pi_S(\sigma_R(\pi_{S \cup vars(R)}(A))) \tag{37.16}$$

$$\pi_{S_1}(\pi_{S_2}(A)) \equiv \pi_{S_1 \cap S_2}(A) \tag{37.17}$$

$$\pi_S(A_1 \cup A_2) \equiv \pi_S(A_1) \cup \pi_S(A_2) \tag{37.18}$$

$$\pi_S(A_1 \bowtie A_2) \equiv \pi_S(\pi_{S'}(A_1) \bowtie \pi_{S'}(A_2)) \tag{37.19}$$

$$\pi_S(A_1 \setminus A_2) \equiv \pi_S(\pi_{S'}(A_1) \setminus \pi_{S''}(A_2)) \tag{37.20}$$

$$\pi_S(A_1 ⟕ A_2) \equiv \pi_S(\pi_{S'}(A_1) ⟕ \pi_{S'}(A_2)) \tag{37.21}$$

Bizonyítás. Megtalálható a [321] cikkben. ∎

A projekcióra vonatkozó szabályok jelentőségét az adja, hogy egy jó adatbázismotor az optimalizálás során feltehetően olyan kifejezéseket részesítene előnyben, amelyekben a projekció minél hamarabb történik meg, így már a végrehajtás korai szakaszában csökken az adatmennyiség.

A fenti szabályok szemléletes jelentése a következő. A 37.14. és a 37.15. szabály azt fejezi ki, hogy a projekció szempontjából érdekes változók a $pVars(A)$ halmazból kerülnek ki. Az előbbi szabály szerint lényegében nem projektáltunk semmit, ha a projekcióban minden $pVars(A)$-beli változót megtartunk, míg az utóbbi szabály szerint a projekcióból a $pVars(A)$-n kívül eső változókat el is hagyhatjuk. A 37.16. szabály szerint szűrések esetén beiktathatunk egy előzetes projekciót, ha a szűrés után amúgy is ez következne, mindössze arra kell figyelnünk, hogy a szűrőfeltétel változóit meghagyjuk. A 37.17. szabály az ismételt projekció átírását teszi lehetővé egyetlen projekcióvá. A 37.18–37.21. szabályok a projekció és a bináris operátorok kapcsolatáról szólnak. Ezek alapján a projekciók bevihetők az operandusokra egy esetleg bővebb változóhalmazra való vetítésként. A változóhalmaz bővítésére azért van szükség, hogy a bináris műveletek értelmesek maradjanak, azaz ne tüntessük el olyan változók értékeit, amelyek ezáltal megváltoztatnák az összekapcsolás, vagy kivonás eredményét.

A szűrések átírását a következő csoportba tartozó szabályok teszik lehetővé.

37.29. tétel. *Legyenek A, A_1 és A_2 tetszőleges halmazalgebra kifejezések, R, R_1, R_2 pedig szűrőfeltételek. Ekkor az alábbi átírási szabályok teljesülnek.*

$$\sigma_{R_1 \wedge R_2}(A) \equiv \sigma_{R_1}(\sigma_{R_2}(A)) \tag{37.22}$$

$$\sigma_{R_1 \vee R_2}(A) \equiv \sigma_{R_1}(A) \cup \sigma_{R_2}(A) \tag{37.23}$$

$$\sigma_{R_1}(\sigma_{R_2}(A)) \equiv \sigma_{R_2}(\sigma_{R_1}(A)) \tag{37.24}$$

$$\sigma_{bound(?X)}(A) \equiv A, \ \textit{ha} \ ?X \in cVars(A) \tag{37.25}$$

$$\sigma_{bound(?X)}(A) \equiv \emptyset, \ \textit{ha} \ ?X \notin pVars(A) \tag{37.26}$$

$$\sigma_{\neg bound(?X)}(A) \equiv \emptyset, \ \textit{ha} \ ?X \in cVars(A) \tag{37.27}$$

$$\sigma_{\neg bound(?X)}(A) \equiv A, \ \textit{ha} \ ?X \notin pVars(A) \tag{37.28}$$

$$\sigma_R(A_1 \cup A_2) \equiv \sigma_R(A_1) \cup \sigma_R(A_2) \tag{37.29}$$

$$\sigma_R(A_1 \setminus A_2) \equiv \sigma_R(A_1) \setminus A_2 \tag{37.30}$$

Ha minden $?X \in vars(R)$-re: $?X \in cVars(A_1) \vee ?X \notin pVars(A_2)$, akkor:

$$\sigma_R(A_1 \bowtie A_2) \equiv \sigma_R(A_1) \bowtie A_2 \tag{37.31}$$

$$\sigma_R(A_1 \ltimes A_2) \equiv \sigma_R(A_1) \ltimes A_2 \tag{37.32}$$

Bizonyítás. Megtalálható a [321] cikkben. ∎

A szűrésekre vonatkozó szabályok motivációja hasonló a projekciónál említetthez: ha a szűrések felbontásával és átrendezésével elérhetjük, hogy azok a kifejezésben beljebbre kerüljenek, akkor már a végrehajtás korai fázisában csökken az adatmennyiség, amivel a továbbiakban dolgoznunk kell. A 37.22–37.24. szabályok a szűrőfeltételek felbontását és átrendezését teszik lehetővé. A 37.25–37.28. szabályok a változók kötöttségére vonatkozó szűrések elhagyhatóságát fogalmazzák meg azokban az esetekben, amikor a kérdéses változóról statikusan is eldönthetjük a kötöttséget. A 37.29–37.32. szabályok a szűrés és a bináris operátorok kapcsolatát mutatják meg. Ezek szerint a szűrés is bevihető az operandusokra, csupán az összekapcsolásoknál kell ügyelni arra, hogy a szűrőfeltétel értelmes maradjon csak az egyik operandusra értve is, és ez ne befolyásolja az összekapcsolás eredményét.

Végezetül nézzünk két olyan szabályt, amely bizonyos atomi szűrések elhagyhatóságát mutatja meg. Ezek lényege, hogy a szűrés által megkövetelt egyenlőség teljesülését a szűrés elhagyása után úgy fejezzük ki, hogy az egyenlőségnek megfelelő helyettesítéseket elvégezzük az algebrai kifejezés törzsében. Ehhez nézzük először a behelyettesítés fogalmát.

37.30. definíció. *Legyen A egy halmazalgebrai kifejezés, amely csak az $\cup$ és $\bowtie$ operátorok, valamint a $[\![t]\!]_D$ formájú hármasok használatával épül fel, továbbá legyenek $?X, ?Y \in cVars(A)$ változók. Ekkor $A\frac{?Y}{?X}$ azt a kifejezést jelöli, amit úgy kapunk A-ból, hogy az $?X$ minden előfordulását $?Y$-al helyettesítjük. Hasonlóan, jelölje $A\frac{c}{?X}$ egy c konstans* ***behelyettesítését*** *$?X$ helyére A-ban.*

A szűrőfeltételek behelyettesíthetőségét az alábbi tétel mondja ki.

37.31. tétel. *Legyen A egy – a 37.30. definíció feltételeit teljesítő – halmazalgebrai kifejezés, és legyenek $?X, ?Y \in cVars(A)$ változók. Ekkor az alábbi átírási szabályok teljesülnek.*

$$\pi_{S\setminus\{?X\}}(\sigma_{?X=?Y}(A)) \equiv \pi_{S\setminus\{?X\}}(A\tfrac{?Y}{?X}) \tag{37.33}$$

$$\pi_{S\setminus\{?X\}}(\sigma_{?X=c}(A)) \equiv \pi_{S\setminus\{?X\}}(A\tfrac{c}{?X}) \tag{37.34}$$

Bizonyítás. Megtalálható a [321] cikkben. ∎

A szabályok megfogalmazásában látható, hogy a behelyettesítés csak akkor végezhető el, ha a behelyettesítéssel eltüntetett változó a következő lépésben – egy projekció miatt – amúgy is eltűnt volna. Ellenkező esetben a változó eltüntetésével a kifejezés többi része értelmetlenné válhat. Ez a gyakorlatban nem jelent túl szigorú megkötést, mivel a 37.28. tétel átírási szabályaival gyakran a nem szigorúan ilyen alakú kifejezéseket is a megfelelő alakra tudjuk hozni.

Az eddigiek során a halmazszemantikához tartozó algebrai kifejezések átírásáról volt szó. A következőekben megmutatjuk, hogyan vihetőek át az átírási szabályok a multihalmaz szemantikára. Először ki kell terjesztenünk a $cVars$ és $pVars$ függvényeket a multihalmaz-algebrai kifejezésekre is. Ehhez mindössze annyit kell tennünk, hogy a rekurzív definíciókban az alapesetet $(A = [\![t]\!]_D)$ kicseréljük a multihalmaz szemantikánál használt megfelelőjére $(A = [\![t]\!]^+_D)$, míg minden más változatlanul marad.

Ezek után kimondhatjuk, hogy a fent bevezetett átírási szabályok közül majdnem mindegyik átvihető multihalmaz szemantikára is.

37.32. tétel. *A 37.27., 37.28., 37.29. és 37.31. tételekben tárgyalt átírási szabályok a 37.1. és 37.23. szabály kivételével mind teljesülnek a multihalmaz szemantikában is.*

Bizonyítás. Megtalálható a [321] cikkben. ∎

A fenti elméleti eredmények már biztos alapot nyújtanak egy SPARQL lekérdezés-optimalizáló motor elkészítéséhez. Egy ilyen motor fejlesztése azonban többet igényel az átírási szabályok beépítésén kívül (például költségbecslés, optimalizálási stratégia, heurisztikák stb.), erre most nem térünk ki.

37.3.2. Szemantikus optimalizálás

Az algebrai optimalizálás során csak a kifejezéseket alkotó operátorok általános tulajdonságait használtuk ki arra, hogy hatékonyabban kiértékelhető formára hozzuk a lekérdezéseket. A szemantikus optimalizálásnál ezzel szemben szükségünk van az adathalmazra vonatkozó többletinformációkra, megszorítások formájában. Ezek birtokában a lekérdezésen olyan átalakításokat tudunk végrehajtani, amelyek általános esetben nem tekinthetők ekvivalens átalakításnak, azonban a megszorításoknak eleget tevő adathalmaz felett nem változtatják meg a lekérdezés jelentését. A szemantikus optimalizálás tárgyalásakor erősen építünk a relációs adatbázisok körében ismert alapfogalmakra, mint a konjunktív lekérdezések, megszorítások, és a chase algoritmus. A következőkben ezek rövid összefoglalása, és a szemantikus adatbázisok világára történő átfogalmazása következik.

37.33. definíció. *Egy **konjunktív lekérdezés** egy $q : ans(\bar{x}) \leftarrow \varphi(\bar{x}, \bar{y})$ alakú kifejezés, ahol φ relációs atomok konjunkciója, és $\bar{x}$ minden változója előfordul φ-ben is. A q konjunktív lekérdezés szemantikája I adatbázispéldányon: $q(I) = \{\bar{a} \mid I \models \exists \bar{y} \varphi(\bar{a}, \bar{y})\}$.*

A szemantikus adatbázisoknál az adathalmazunkat egyetlen speciális ternáris relációként fogjuk fel, ezt T-vel jelöljük. A SPARQL-ben a csak AND operátort és esetleg egy külső projekciót tartalmazó kifejezések felelnek meg a konjunktív lekérdezéseknek (projekciót nem tartalmazó kifejezés esetén odaképzelhetünk egy $pVars$-ra történő projekciót a jelentés megváltoztatása nélkül). Jelölje ezen kifejezések osztályát $\mathcal{A}^\pi$. A SPARQL kifejezések és a konjunktív lekérdezések közötti kapcsolatot az alábbi definíció teremti meg.

37.34. definíció. *Legyen $S \subset V$, és $Q \in \mathcal{A}^\pi$ az alábbi alakú SPARQL kifejezés:*

$$Q = \text{SELECT}_S((s_1, p_1, o_1) \text{ AND } \ldots \text{ AND } (s_n, p_n, o_n))$$

Ekkor $cq(Q)$ jelölje a Q-nak megfelelő konjunktív lekérdezést, amely az alábbi alakú:

$$q : ans(\bar{s}) \leftarrow T(s_1, p_1, o_1), \ldots, T(s_n, p_n, o_n),$$

ahol $\bar{s}$ pontosan az S-beli változókat tartalmazza. Az ellenkező irányú fordítást jelölje $cq^{-1}(q) = Q$, amely csak akkor van definiálva, ha Q érvényes SPARQL lekérdezés, azaz minden i-re $(s_i, p_i, o_i) \in UV \times UV \times LUV$.

A szemantikus optimalizálásnál kihasznált többlet információ az adathalmazra vonatkozó megszorítások formájában jelenik meg. Ezen megszorítások forrása többféle is lehet: lehetnek a felhasználó által megadottak, egy ontológiában rögzítettek vagy az adathalmaz alapján automatikusan kivontak. A megszorítások általában tetszőleges elsőrendű logikai mondatok lehetnek. Speciális esetként megkülönböztethetünk sor-generáló függőségeket, és egyenlőség-generáló függőségeket, melyek alakja rendre $\forall \bar{x}(\varphi(\bar{x}) \rightarrow \exists \bar{y} \psi(\bar{x}, \bar{y}))$ és $\forall \bar{x}(\varphi(\bar{x}) \rightarrow x_i = x_j)$.

A ***chase algoritmus*** a relációs adatbázisoknál jól ismert módszer a szemantikus optimalizációra. Bemenetként egy q konjunktív lekérdezést, és sor- valamint egyenlőség-generáló függőségek egy Σ halmazát kapja. A q lekérdezés törzsét adatbázis-előforduláskén értelmezve iteratívan megjavítja a függőségeket sértő törzset (ez jelentheti sor hozzáadását, vagy változók egybeolvasztását). Végül kimenetként olyan lekérdezést kapunk (jelölje ezt q^Σ), amelynek törzse már kielégíti a függőségeket, és amely ekvivalens q-val minden olyan adathalmazon, ami eleget tesz Σ-nak, azaz $q \equiv_\Sigma q^\Sigma$.

A SPARQL lekérdezések szemantikus optimalizálásához a chase módszeren alapuló ***C&B (chase and backchase) algoritmust*** használjuk. A C&B algoritmus bemenetként kap egy q konjunktív lekérdezést, valamint függőségek egy Σ halmazát, kimenetként pedig q-val (Σ-nak eleget tevő adathalmazokon) ekvivalens, minimális lekérdezéseket ad. Az algoritmus két fázisa a chase és a backchase névre hallgat. Elsőként a chase fázisban lefuttatjuk a chase algoritmust q-ra, így kapunk egy másik u konjunktív lekérdezést, amely feltehetően nagyobb, mint q (hiszen a sor-generáló függőségek kielégítése miatt sorokat adtunk hozzá). A backchase fázisban u-ból kiindulva keressük a minimális megfelelőjét úgy, hogy u-ból sorokat elhagyva alkérdéseket képzünk, és az alkérdéseket chase-elve ellenőrizzük, hogy ekvivalensek maradtak-e u-val (és így q-val is) Σ felett. Így megkaphatjuk a q-val ekvivalens, és (sorok számát tekintve) minimális konjunktív lekérdezések halmazát, amit $cb_\Sigma(q)$-val jelölünk. Megjegyzés: a chase algoritmus nem minden esetben terminál, a C&B algoritmus pedig pontosan akkor terminál, ha a mögöttes chase algoritmus is.

A fogalmak tisztázása után készen állunk a csak AND operátort és projekciót tartalmazó SPARQL résznyelv szemantikus optimalizálásának ismertetésére. Az optimalizálás menete a következő: a lekérdezést a cq függvény segítségével átírjuk konjunktív lekérdezéssé, ezen lefuttatjuk a C&B algoritmust, majd az eredményként kapott minimális lekérdezések visszaírásával megkapjuk az optimalizált SPARQL lekérdezéseket. A művelet helyességét az alábbi tétel mondja ki.

37.35. tétel. *Legyen Q egy $\mathcal{A}^\pi$ osztálybeli SPARQL lekérdezés, és legyen Σ sor- és egyenlőség-generáló függőségek egy halmaza. Ekkor ha $cb_\Sigma(cq(Q))$ létezik, továbbá $q \in cb_\Sigma(cq(Q))$ és $cq^{-1}(q)$ is létezik, akkor $cq^{-1}(q) \equiv_\Sigma Q$.*

Bizonyítás. Megtalálható a [321] cikkben. ∎

Feladatok

37-1 Összetett kiértékelési probléma

A 37.9. tétellel kapcsolatban bizonyítsuk be, hogy az AND és UNION műveleteket tartalmazó SPARQL lekérdezések esetében az összetett kiértékelési probléma NP-beli.

37-2 Jól tervezettség eldönthetősége

M utassuk meg, hogy tetszőleges P SPARQL lekérdezés jól tervezettsége – AND, UNION, OPT, FILTER műveletekkel – eldönthető $O(|P|^2)$ időben.

37-3 Jól tervezettség bonyolultsága
P róbáljuk meg bebizonyítani, hogy jól tervezett SPARQL lekérdezések esetén az összetett kiértékelési probléma coNP-teljes (37.19. tétel).

Megjegyzések a fejezethez

A szemantikus web vízióját és alapötleteit Tim Berners-Lee, James Hendler és Ora Lassila 2001-es cikke [46] foglalja össze. Az RDF adatmodell [213] 2004 óta, a SPARQL lekérdező nyelv [297] pedig 2008 óta hivatalos W3C ajánlás, azaz szabványosított web-technológia. A módszertan és az eszközök népszerűségét a LOD-felhőbe [47] tartozó számos publikusan is elérhető szemantikus adathalmaz mutatja.

A fogalmak formális tárgyalásához bevezetett halmazszemantika a [295] és [20] cikkekben található elméleti vizsgálatokon alapszik, míg a multihalmaz szemantika a [297] és [294] cikkekben található hivatalos W3C ajánlásokat követi. A SELECT és ASK lekérdezések kiértékeléséhez használt kompozíciós szemantika bevezetése követi a [296] cikkben találhatóakat. A SPARQL bonyolultsági kérdéseivel foglalkozó 37.2. alfejezet a [296] és [321] cikkekre épít; az optimalizálásról szóló 37.3. alfejezeta [321] cikk eredményeit tárgyalja. A szemantikus optimalizálásnál használt alapfogalmakról bővebben a következő cikkekből érdemes tájékozódni: chase algoritmus [37], függőségek (megszorítások) [42], C&B algoritmus [102].

Jelen kutatást a FuturICT.hu nevű, TÁMOP-4.2.2.C-11/1/KONV-2012-0013 azonosítószámú projekt támogatta az Európai Unió és az Európai Szociális Alap társfinanszírozása mellett.

38. 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.

38.1. 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.

38.1.1. 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 for-

dul 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

$$T_1 \circ T_2 = S \tag{38.1}$$

teljesül, akkor

$$w(T_1) + w(T_2) = w(S) \ , \tag{38.2}$$

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

$$\delta(A, B) = \min\{w(T) | T(A) = B\} \ . \tag{38.3}$$

Ha w-ről feltesszük, hogy

$$w(a \leftarrow b) = w(b \leftarrow a) \ , \tag{38.4}$$

$$w(a \leftarrow a) = 0 \ , \tag{38.5}$$

$$w(b \leftarrow a) + w(c \leftarrow b) \geq w(c \leftarrow a) \tag{38.6}$$

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:

```
# -     - #
- #     # -  ,
```

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

$$\begin{aligned} w(\alpha^*(A_i, B_j)) = \quad & \min\{w(\alpha^*(A_{i-1}, B_j)) + w(- \leftarrow a_i) \ ; \\ & w(\alpha^*(A_i, B_{j-1})) + w(b_i \leftarrow -) \ ; \\ & w(\alpha^*(A_{i-1}, B_{j-1})) + w(b_i \leftarrow a_i)\} \ . \end{aligned} \tag{38.7}$$

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:

$$d_{0,0} = 0 \ , \tag{38.8}$$

$$d_{i,0} = \sum_{k=1}^{i} w(- \leftarrow a_k) \ , \tag{38.9}$$

$$d_{0,j} = \sum_{l=1}^{j} w(b_l \leftarrow -) \ . \tag{38.10}$$

A táblázat belsejének a kitöltése a (38.7) 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.

38.1.2. 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 $w(\alpha^*(A_k, B_j))$, $0 \leq k < i$-re és minden $w(\alpha^*(A_i, B_l))$, $0 \leq l < j$-re. Az előbbiekhez hasonlóan belátható, hogy

$$\begin{aligned} w(\alpha^*(A_i, B_j)) = \quad & \min\{w(\alpha^*(A_{i-1}, B_{j-1})) + w(b_j \leftarrow a_i)\ , \\ & \min_{0 \leq k < i}\{w(\alpha^*(A_k, B_j)) + g_{i-k}\}\ , \\ & \min_{0 \leq l < j}\{w(\alpha^*(A_i, B_l)) + g_{j-l}\}\}\ . \end{aligned} \tag{38.11}$$

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:

$$d_{0,0} = 0\ , \tag{38.12}$$

$$d_{i,0} = g_i\ , \tag{38.13}$$

$$d_{0,j} = g_j\ . \tag{38.14}$$

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.

38.1.3. Gotoh algoritmusa affin résbüntetéssel

Egy résbüntető függvényt ***affin függvénynek*** hívunk, ha

$$g_k = ku + v, \quad u \geq 0, \;\; v \geq 0 \; . \tag{38.15}$$

Ilyen résbüntető függvényt alkalmaz a Gotoh-algoritmus, amelynek a futási ideje $\Theta(nm)$. Emlékeztetőül, az előbbi gyors algoritmusban

$$d_{i,j} = \min\{d_{i-1,j-1} + w(b_j \leftarrow a_i);\; p_{i,j};\; q_{i,j}\} \; , \tag{38.16}$$

ahol

$$p_{i,j} = \min_{1 \leq k < i} \{d_{i-k,j} + g_k\} \; , \tag{38.17}$$

$$q_{i,j} = \min_{1 \leq l < j} \{d_{i,j-l} + g_l\} \; . \tag{38.18}$$

Az algoritmus a következő átindexelésen alapul:

$$\begin{aligned} p_{i,j} &= \min\{d_{i-1,j} + g_1,\; \min_{2 \leq k < i} \{d_{i-k,j} + g_k\}\} \\ &= \min\{d_{i-1,j} + g_1,\; \min_{1 \leq k < i-1} \{d_{i-1-k,j} + g_{k+1}\}\} \\ &= \min\{d_{i-1,j} + g_1,\; \min_{1 \leq k < i-1} \{d_{i-1-k,j} + g_k\} + u\} \\ &= \min\{d_{i-1,j} + g_1,\; p_{i-1,j} + u\} \; . \end{aligned} \tag{38.19}$$

És hasonlóan

$$q_{i,j} = \min\{d_{i,j-1} + g_1,\; q_{i,j-1} + u\} \; . \tag{38.20}$$

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.

38.1.4. 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

$$35,03 - 6,88 \log d + 17,02 \log q \ , \tag{38.21}$$

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      q_1[i, j] = d[i, 0] + g[j]
3      b[i, j] = 0
4  for j = 1 to m
5      q[i, j] = q_1[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 = j_1 to m                                  // Belső ciklus.
9          if q_1[i, j_1] > d[i, j] + g[j_1 − j]
10             q_1[i, j_1] = d[i, j] + g[j_1 − j]
11             b[i, j_1] = 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.

38.1. lemma. *Legyen j az aktuális cella. Ha*

$$d_{i,j} + g_{j_1-j} \geq q_1[i, j_1] \ , \tag{38.22}$$

akkor minden $j_2 > j_1$-re

$$d_{i,j} + g_{j_2-j} \geq q_1[i, j_2] \ . \tag{38.23}$$

Bizonyítás. A feltételből következik, hogy van olyan $k < j < j_1 < j_2$, melyre

$$d_{i,j} + g_{j_1-j} \geq d_{i,k} + g_{j_1-k} \ . \tag{38.24}$$

Adjunk az egyenlőtlenség mindkét oldalához $(g_{j_2-k} - g_{j_1-k})$-t:

$$d_{i,j} + g_{j_1-j} + g_{j_2-k} - g_{j_1-k} \geq d_{i,k} + g_{j_2-k} \ . \tag{38.25}$$

A konkáv résbüntető függvény tulajdonságából

$$g_{j_2-j} - g_{j_1-j} \geq g_{j_2-k} - g_{j_1-k} \ , \tag{38.26}$$

és ebből átrendezve és a (38.25) egyenletet felhasználva :

$$d_{i,j} + g_{j_2-j} \geq d_{i,j} + g_{j_1-j} + g_{j_2-k} - g_{j_1-k} \geq d_{i,k} + g_{j_2-k} \ , \tag{38.27}$$

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.

38.2. 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 balról jobbra haladva blokkonként csökkennek.*

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      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        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        pn[i] = 0
10       b[i,0] = j
11       else if j + 1 < m
12             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         (pn[i], b[i,Y]) = (Y, j)
15         else E = p[i,Y]
17       if Y < pn[i]
18         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 $O(nm(\lg n + \lg m))$.

38.1.5. 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***:

$$S(a,b) = \log\left(\frac{\Pr\{a,b\}}{\Pr\{a\}\Pr\{b\}}\right) , \tag{38.28}$$

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:

$$d_{0,0} = d_{i,0} = d_{0,j} = 0 . \tag{38.29}$$

A dinamikus programozási táblázat kitöltése lineáris résbüntetés mellett:

$$d_{i,j} = \max\{0;\, d_{i-1,j-1} + S(a_i, b_j), d_{i-1,j} + g;\, d_{i,j-1} + g\}\ . \tag{38.30}$$

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.

38.1.6. 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 szekven-

ciák 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 38.5 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.

38.1.7. 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 algoritmusa,* 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,

$$\min_j \left\{ w(\alpha^*(A_{[|A|/2]}, B_j)) + w(\alpha^*(A^{[|A|/2]}, B^j)) \right\} , \tag{38.31}$$

és a számolásból adódik, hogy az optimális illesztésben $A_{[|A|/2]}$ B_j azon prefixével van illesztve, melyre

$$w(\alpha^*(A_{[|A|/2]}, B_j)) + w(\alpha^*(A^{[|A|/2]}, B^j)) \tag{38.32}$$

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 (38.31) képletben minimális értéket kapunk. Így a teljes számolási igény

$$nm \times \left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right) = \Theta(nm)\ . \tag{38.33}$$

38.1.8. 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

$$d_{i,j} + |(n-i)-(m-j)|g \le t\ , \tag{38.34}$$

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, illetve Ukkonen algoritmusainak. Azok az algoritmusok szintén teszt értéket tartalmazó egyenlőtlenségeket használtak, de Fickett algoritmusában az egyenlőtlenség

$$d_{i,j} \leq t \ , \tag{38.35}$$

míg Ukkonen algoritmusában az egyenlőtlenség

$$|i-j|g + |(n-i)-(m-j)|g \leq t \tag{38.36}$$

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

$$SP_l = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} d(p_{i,l}, p_{j,l}) \ , \tag{38.37}$$

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

$$d_{i_1,i_2,\ldots,i_n} + \sum_{j=1}^{k-1} \sum_{l=j}^{k} w_{j,l}(i_j, i_l) \leq t \;, \tag{38.38}$$

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

38.1-1. Mutassuk meg, hogy egy n és egy m hosszúságú szekvencia esetén a lehetséges illesztések száma

$$\sum_{i=0}^{\min(n,m)} \frac{(n+m-i)!}{(n-i)!(m-i)!i!} \;.$$

38.1-2. Adjunk meg egy olyan értékelést és szekvenciák olyan sorozatát, amelyekre az optimális illesztések száma exponenciálisan nő a szekvenciák hosszával.

38.1-3. Adjuk meg Hirschberg algoritmusát többszörös szekvenciaillesztésre.

38.1-4. Adjuk meg Hirschberg algoritmusát affin résbüntető függvények esetén.

38.1-5. Adjuk meg a Smith-Waterman algoritmust affin résbüntetésekre.

38.1-6. Adjuk meg a Spouge-féle saroklevágási technikát affin résbüntetésekre.

38.1-7. 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.

38.2. Algoritmusok fákon

Az alább ismertetésre kerülő algoritmusok gyökereztetett fákon dolgoznak. A dinamikus programozás a gyökereztetett részfákon való visszavezetéssel történik. Mint látni fogjuk, nem csak optimális eseteket határozhatunk meg, hanem ugyanakkora futási idő alatt algebrai kifejezéseket is meghatározhatunk.

38.2.1. 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

$$m_{r,\omega} = \sum_{t \in D(r)} \min_{\sigma \in \Sigma} \{m_{t,\sigma} + \delta_{\omega,\sigma}\} \ , \tag{38.39}$$

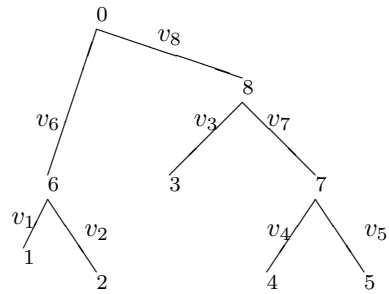
ahol a $D(r)$ r gyerekeinek a halmaza, Σ a lehetséges karakterek halmaza, $\delta_{\omega,\sigma}$ pedig 1 ha $\omega = \sigma$ és 0 egyébként. A (38.39) képlet helyessége abból adódik, hogy a megvizsgáltuk az összes lehetőséget arra vonatkozóan, hogy r gyerekeihez milyen karaktereket rendelhetünk, és 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.

38.2.2. 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 (38.1. 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



38.1. ábra. A fa, amin Felsenstein algoritmusát bemutatjuk. Az élekre írt v-k az éleken eltelt evolúciós időt jelölik.

ismerjük, ezért minden lehetséges állapotra összegezni kell:

$$L = \sum_{s_0}\sum_{s_6}\sum_{s_7}\sum_{s_8} \pi_{s_0} \times f_{s_0 s_6}(v_6) \times f_{s_6 s_1}(v_1) \times f_{s_6 s_2}(v_2) \times f_{s_0 s_8}(v_8) \times f_{s_8 s_3}(v_3) \times f_{s_8 s_7}(v_7) \times f_{s_7 s_4}(v_4) \times f_{s_7 s_5}(v_5) \quad (38.40)$$

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

$$L = \sum_{s_0} \pi_{s_0} \left\{ \sum_{s_6} f_{s_0 s_6}(v_6)[f_{s_6 s_1}(v_1)][f_{s_6 s_2}(v_2)] \right\} \times \left\{ \sum_{s_8} f_{s_0 s_8}(v_8)[f_{s_8 s_3}(v_3)] \left(\sum_{s_7} f_{s_8 s_7}(v_7)[f_{s_7 s_4}(v_4)][f_{s_7 s_5}(v_5)] \right) \right\} \quad (38.41)$$

szorzatra, aminek a számítási igénye jóval kisebb. Vegyük észre, hogy a (38.41) 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

38.2-1. Adjunk algoritmust a súlyozott takarékossági elv kis problémájára, azaz amikor az egyes változásokat súlyozzuk, és a változások súlyainak az összegét akarjuk minimalizálni.

38.2-2. 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.

38.2-3. Adott szekvenciákra és törzsfára adjuk meg a Maximum Likelihood evolúciós történetet, azaz a fa belső csúcsain azokat a szekvenciákat, amelyre a likelihood maximális.

38.2-4. Írjuk fel a takarékossági elv kis problémáját a (38.40) képlethez hasonló alakban (csak az összegzések helyett minimumok szerepeljenek), és mutassuk meg, hogy Sankoff algoritmusa tulajdonképpen Felsenstein algoritmusához hasonló átrendezésen alapszik.

38.2-5. 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

$$\begin{array}{ll} \cap_{t\in D(r)} C_t, & \text{ha} \quad \cap_{t\in D(r)} C_t \neq \emptyset\,, \\ \cup_{t\in D(r)} C_t & \text{egyébként} \quad . \end{array}$$

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 tetszőleges karaktert választunk. Mutassuk meg, hogy így egy legtakarékosabb történethez jutunk. Mennyi lesz ezen algoritmus futási ideje?

38.2-6. Mutassuk meg, hogy a 38.2.1. pontban megadott algoritmussal minden lehetséges legtakarékosabb evolúciós történetet megkaphatunk. Adjunk példát olyan legtakarékosabb történetre, amelyet Fitch algoritmusával nem kaphatunk meg.

38.3. 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.

38.3.1. 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 38.3-3 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_ω^i-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

$$\Pr_{max}\{A_{k+1},\, j\} = \max_i(\Pr_{max}\{A_k,\, i\}\, m_{i,j}\pi^j_{a_{k+1}})\ . \tag{38.42}$$

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

$$\Pr_{max}\{A\} = \Pr_{max}\{A, END\} = \max_i(\Pr_{max}\{A, i\}\, m_{i,END}) \;, \tag{38.43}$$

ahol $\Pr_{max}\{A\}$ a legvalószínűbb út valószínűsége. Ha nincsen végállapot, akkor

$$\Pr_{max}\{A\} = \max_i(\Pr_{max}\{A, i\}) \;. \tag{38.44}$$

Ha van *START* állapot, akkor a folyamat szükségképpen a *START* állapotból indul, úgyhogy $\Pr_{max}\{A_0, START\} = 1$. Ha nincs start állapot, akkor

$$\Pr_{max}\{A_1, j\} = p_j\pi^j_{a_1} \;, \tag{38.45}$$

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

$$\Pr\{A_{k+1}, j\} = \sum_i \Pr\{A_k, i\}\, m_{i,j}\pi^j_{a_{k+1}} \;. \tag{38.46}$$

Az *END* állapot nem emittál, így az algoritmus $\Pr\{A\}$-t a következőképen számolja ki:

$$\Pr\{A\} = \Pr\{A, END\} = \sum_i \Pr\{A, i\}\, m_{i,END} \;. \tag{38.47}$$

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 $\Pr\{a_k\text{-t az } i \text{ állapot emittálta} \mid \text{a folyamat az } A \text{ szekvenciát emittálta}\}$ 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

$$\Pr\left\{A^k,\, i\right\} = \sum_j (\Pr\left\{A^{k+1},\, j\right\} m_{i,j} \pi^j_{a_{k+1}}) \,. \tag{38.48}$$

Jelöljük a $\Pr\{a_k\text{-t az } i \text{ állapot emittálta} \mid \text{a folyamat az } A \text{ szekvenciát emittálta}\}$ valószínűséget $\Pr\{a_k = i|A\}$-val.

$$\Pr\{a_k = i|A\}\Pr\{A\} = \Pr\{A \wedge a_k = i\} = \Pr\{A_k,\, i\}\Pr\left\{A^k,\, i\right\} \,, \tag{38.49}$$

amiből:

$$\Pr\{a_k = i|A\} = \frac{\Pr\{A_k,\, i\}\Pr\left\{A^k,\, i\right\}}{\Pr\{A\}} \,, \tag{38.50}$$

ami éppen a keresett valószínűség.

38.3.2. 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 $W_v \to W_y W_z$ 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 $W_v \to W_y W_z$ 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:

$$\alpha(i,i,v) = e_v(a_i) \ , \tag{38.51}$$

minden i-re és v-re. A fő rekurzió:

$$\alpha(i,j,v) = \sum_{y=1}^{M} \sum_{z=1}^{M} \sum_{k=i}^{j-1} \alpha(i,k,y) t_v(y,z) \alpha(k+1,j,z) \ , \tag{38.52}$$

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:

$$\beta(1,L,1) = 1 \ , \tag{38.53}$$

$$\beta(1,L,v) = 0 \quad \texttt{ha } v \neq 1 \ . \tag{38.54}$$

A fő rekurzió:

$$\begin{aligned}\beta(i,j,v) &= \sum_{y=1}^{M}\sum_{z=1}^{M}\sum_{k=1}^{i-1}\alpha(k,i-1,z)t_y(z,v)\beta(k,j,y) + \\ &\quad \sum_{y=1}^{M}\sum_{z=1}^{M}\sum_{k=j+1}^{L}\alpha(j+1,k,z)t_y(v,z)\beta(i,k,y)\ . \end{aligned} \tag{38.55}$$

A (38.55) 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:

$$\alpha_{\max}(i,j,v) = \max_{y}\max_{z}\max_{i\le k\le j-1}\alpha_{\max}(i,k,y)t_v(y,z)\alpha_{\max}(k+1,j,z)\ . \tag{38.56}$$

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,

$$\frac{\alpha(i,j,v)\beta(i,j,v)}{\alpha(1,L,1)} \tag{38.57}$$

adja meg.

Gyakorlatok

38.3-1. 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.

38.3-2. 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.

38.3-3. 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.

38.3-4. 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.

38.3-5. 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.

38.3-6. 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.

38.3-7. 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.

38.4. 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.

38.4.1. 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

$$\sum_{n \in V_A} s(c(n)) \tag{38.58}$$

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 egyféleképpen 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 sokféleképpen 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 összeilleszt\H{o}dik, 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.

38.4.2. 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:

$$C(M_1, M_2) = \sum_s \Pr_{M_1}\{s\} \Pr_{M_2}\{s\}, \tag{38.59}$$

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,

$$\begin{aligned} C(M_1, M_2) &= \sum_s \left(\sum_{p_1 \in M_1, e(p_1)=s} \Pr_{M_1}\{p_1\} \right) \left(\sum_{p_2 \in M_2, e(p_2)=s} \Pr_{M_2}\{p_2\} \right) \\ &= \sum_{p_1 \in M_1, p_2 \in M_2, e(p_1)=e(p_2)} \Pr_{M_1}\{p_1\} \Pr_{M_2}\{p_2\} . \end{aligned} \tag{38.60}$$

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

$$\begin{aligned} C(M_1, M_2) &= \sum_{p_1 \in M_1, p_2 \in M_2, e(p_1)=e(p_2)} m_{x_1, END_1} m_{x_2, END_2} \Pr_{M_1}\{\bar{p}_1\} \Pr_{M_2}\{\bar{p}_2\} \\ &= \sum_{x_1, x_2} m_{x_1, END_1} m_{x_2, END_2} C(x_1, x_2) , \end{aligned} \tag{38.61}$$

ahol $m_{x,END}$ az x-ből az END állapotba ugrás valószínűsége, valamint

$$C(x_1, x_2) = \sum_{[x_1] \in M_1, [x_2] \in M_2, e([x_1])=e([x_2])} \Pr_{M_1}\{[x_1]\} \Pr_{M_2}\{[x_2]\} . \tag{38.62}$$

$C(x_1, x_2)$-t meg lehet adni a következő képlettel is:

$$C(x_1, x_2) = \sum_{y_1, y_2} m_{y_1, x_1} m_{y_2, x_2} C(y_1, y_2) \sum_{\sigma \in \Sigma} \Pr\{\sigma | x_1\} \Pr\{\sigma | x_2\} , \tag{38.63}$$

ahol $\Pr\{\sigma | x_i\}$ annak a valószínűsége, hogy az x_i állapot σ-t bocsátotta ki. A (38.63) képlet egy lineáris egyenletrendszert definiál az összes x_1 és x_2 kibocsátó állapotokra. A kezdeti feltételek:

$$C(START_1, START_2) = 1 , \tag{38.64}$$

$$C(START_1, x_2) = 0, \quad x_2 \neq START_2 , \tag{38.65}$$

$$C(x_1, START_2) = 0, \quad x_1 \neq START_1 . \tag{38.66}$$

Azonban a dinamikus programozás a szokásostól eltérően nem egy táblázat kitöltésével, hanem a (38.63) 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

38.4-1. Adjuk meg két fa lokális hasonlóságát, ami a két fa leginkább hasonló részfái illesztésének az értéke. Részfán most a fa tetszőleges összefüggő részét érjük.

38.4-2. ***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.

38.4-3. 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.

38.4-4. 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.

38.4-5. 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.

38.5. 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 távolságokra teljesül az additív metrika, akkor a szomszédok egyesítése visszaadja ezt a metrikát.

Mindkét bizonyítás esetében szükségünk lesz az alábbi lemmára:

38.3. 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

$$x + y = d_{i,j} \,, \tag{38.67}$$

$$x + z = d_{i,k} \,, \tag{38.68}$$

$$y + z = d_{k,l} \tag{38.69}$$

egyenletrendszer adja meg, aminek egyetlen megoldása van, mivel az

$$\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{vmatrix} \tag{38.70}$$

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, 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

$$d_{i,k} - d_{j,k} = d_{i,u} - d_{j,u} = 2d_{i,u} - d_{i,j} \ , \tag{38.71}$$

míg a második fán

$$d_{i,k} - d_{j,k} = d_{i,u_1} - d_{j,u_1} = 2d_{i,u_1} - d_{i,j} \ , \tag{38.72}$$

ami ellentmond annak, hogy $d_{i,u} \neq d_{i,u_1}$. ∎

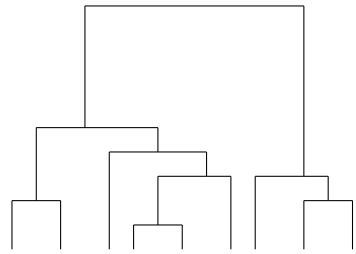
38.5.1. Osztályozó algoritmusok

38.4. definíció. *Egy metrikát* ***ultrametrikának*** *nevezünk, ha bármely i, j és k csúcsokra*

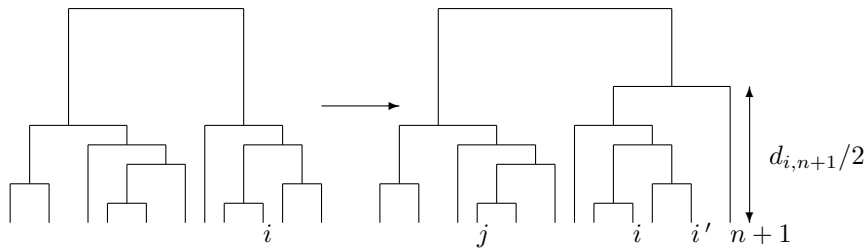
$$d_{i,j} \leq \max\{d_{i,k},\, d_{j,k}\} \tag{38.73}$$

Könnyen belátható (lásd 38.5-1 gyakorlat), hogy egy ultrametrikában bármely három csúcs között levő három távolság vagy mind azonos, vagy közülük kettő azonos, a harmadik pedig ennél kisebb.

38.5. 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.*



38.2. ábra. Egy dendrogram.



38.3. ábra. Az $(n+1)$-edik levél bekötése a dendrogramba.

Bizonyítás. A 38.3. 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 38.2. ábra). A tétel bizonyítása a levelek, mint objektumok száma szerinti indukcióval történik. Kettő objektum esetében nyílván meg tudjuk konstruálni a dendrogramot. Ha n levélre 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 38.3. ábra).

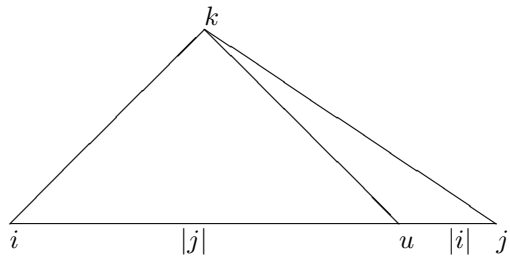
Ez a dendrogram helyesen ábrázolja az összes csúcsnak az $(n+1)$-edik levéltől vett távolságát. Ugyanis az összes olyan i' csúcsra, amely az $n+1$-ik levél bekötésének helye alatt helyezkedik el, $d_{i,i'} \leq d_{i,n+1}$, és az ultrametrikus tulajdonságból, valamint $d_{i,n+1}$ minimalitásából következik, 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 dendrog 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 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 38.4. 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 tu-



38.4. ábra. $d_{u,k}$ számolása a CENTROID módszer szerint.

lajdonságúak, 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.

38.5.2. Szomszédok egyesítése

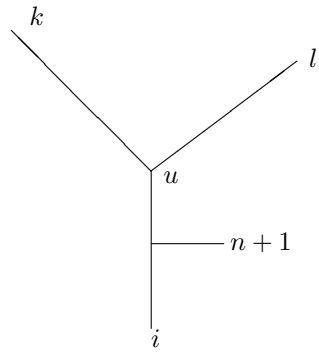
38.6. definíció. *Egy metrikát* ***additív*** *vagy* ***négycsúcs metrikának*** *nevezünk, ha bármely i, j, k és l csúcsára*

$$d_{i,j} + d_{k,l} \leq \max\{d_{i,k} + d_{j,l},\, d_{i,l} + d_{j,k}\}\,. \tag{38.74}$$

38.7. tétel. *Ha objektumok egy véges halmazán definiált metrika additív, akkor pontosan egy olyan fa létezik, amely ezt reprezentálja.*

Bizonyítás. A 38.3. 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 (38.67–38.69) 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



38.5. ábra. Az $(n+1)$-edik levél gyökereztetése additív fa megkonstruálásához.

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 38.5. ábra).

Az $(n+1)$-edik levél bekötése i-ből nézve az u csúcson innen van, ha

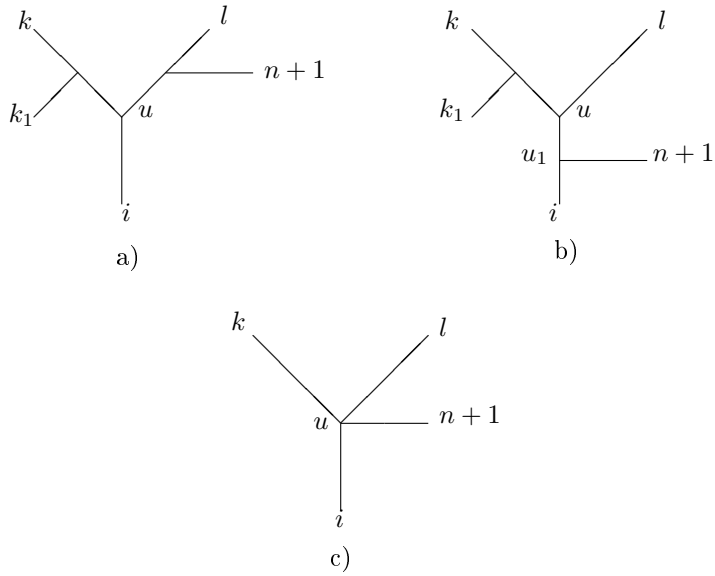
$$d_{i,n+1} + d_{k,l} < d_{i,k} + d_{n+1,l} \; . \tag{38.75}$$

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 (38.6/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 (38.67–38.69) egyenletek alapján számolunk ki. Ha az $(n+1)$-edik levelet az 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



38.6. ábra. Néhány fa topológia a 38.7. tétel bizonyításához.

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 38.6/a ábra). Az $(n+1)$-edik levél helyzetéből és az additív metrika tulajdonságából adódóan

$$d_{k,n+1} + d_{i,l} = d_{i,n+1} + d_{k,l} \ , \tag{38.76}$$

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

$$d_{k,n+1} - d_{k,u} = d_{i,n+1} - d_{i,u} \ . \tag{38.77}$$

Ugyanígy minden olyan k_1 levélre, melyet nem választ el k-tól az u csúcs, fennáll a

$$d_{k_1,n+1} + d_{i,l} = d_{i,n+1} + d_{k_1,l} \tag{38.78}$$

egyenlőség. Ez az additív metrikából és a

$$d_{k,k_1} + d_{l,n+1} < d_{k,n+1} + d_{k_1,l} \tag{38.79}$$

egyenlőtlenségből adódik, ez utóbbi pedig levezethető a

$$d_{k,k_1} + d_{i,l} < d_{k_1,l} + d_{k,i} \tag{38.80}$$

és

$$d_{l,n+1} + d_{k,i} < d_{i,l} + d_{k,n+1} \tag{38.81}$$

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 megkeresésében adódóan az ábrán i lehet egy korábbi u.)

Ha az $(n+1)$-edik levél bekötése az u-t az i-vel összekötő szakaszon van (38.6/b ábra), akkor $d_{u,n+1}$ definíciójából adódóan $d_{l,n+1}$-et is helyesen ábrázolja a fa. A

$$d_{k,n+1} + d_{i,l} = d_{k,i} + d_{l,n+1} \tag{38.82}$$

egyenlőségből egyszerűen levezethető, hogy

$$d_{k,n+1} = d_{k,u} + d_{u,n+1} \ , \tag{38.83}$$

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 (38.6/c ábra), akkor a

$$d_{i,n+1} + d_{k,l} = d_{k,i} + d_{l,n+1} = d_{k,n+1} + d_{j,i} \tag{38.84}$$

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 38.7. 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 (38.75) 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:

$$v_i = \sum_{j=1}^{n} d_{i,j} \ . \tag{38.85}$$

Ezután megkeressük azt a csúcspárt, melyre

$$s_{i,j} = (n-2)d_{i,j} - v_i - v_j \tag{38.86}$$

minimális. Az i és a j csúcsokból az új, u belső csúcsig húzott élek hossza

$$e_{i,u} = \frac{d_{i,j}}{2} - \frac{v_i - v_j}{2n-4} \ , \tag{38.87}$$

illetve

$$e_{j,u} = \frac{d_{i,j}}{2} - e_{i,u} \tag{38.88}$$

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:

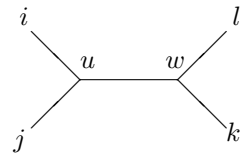
$$d_{k,u} = \frac{d_{k,i} + d_{k,j} - d_{i,j}}{2} \ . \tag{38.89}$$

38.8. tétel. *Ha a bemenő csúcsokon megadott d metrika additív metrika, akkor a* Szom- szédok-egyesítése *algoritmus visszaadja azt a fát, mely ezt a metrikát reprezentálja.*

Bizonyítás. A 38.7. 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 (38.86) egyenletben szereplő definíciót, az $s_{i,j} < s_{i,k}$ egyenlőtlenséget átrendezve kapjuk, hogy

$$\sum_{l \neq i,j} (d_{i,j} - d_{i,l} - d_{j,l}) - 2d_{i,j} - \sum_{m \neq j,k} (d_{j,k} - d_{j,m} - d_{k,m}) + 2d_{j,k} < 0 \tag{38.90}$$



38.7. ábra. Az i, j, k és l csúcsok elhelyezkedése, amennyiben i-t és j-t egyetlen u belső csúcs választja el.

Amennyiben $l = m \neq i, j, k$, a szummákból kapjuk, hogy

$$(d_{i,j} - d_{i,l} - d_{j,l}) - d_{j,k} + d_{j,l} + d_{k,l} = 2d_{w,l} - 2d_{u,l} < 0 \tag{38.91}$$

(lásd még 38.7. 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 (38.90) egyenlőtlenség fennáll.

Ezután a 38.8. 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

$$s_{k,l} - s_{i,j} > 0 \,. \tag{38.92}$$

Ezt átrendezve kapjuk, hogy

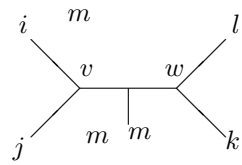
$$\sum_{m_1 \neq k,l} (d_{k,l} - d_{m_1,k} - d_{m_1,l}) - 2d_{k,l} - \sum_{m_2 \neq i,j} (d_{i,j} - d_{m_2,i} - d_{m_2,k}) + 2d_{i,j} > 0 \,. \tag{38.93}$$

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

$$d_{k,l} - d_{m,k} - d_{m,l} - d_{i,j} + d_{m,i} + d_{m,k} \,. \tag{38.94}$$

Ha m az i-t j-vel összekötő úton kapcsolódik az i, j, k és l levelek által kifeszített részfához, akkor a (38.94) kifejezés mindig negatív lesz (lásd 38.8. ábra). Nevezzük ezen m csúcsokat I. esetnek.

Ha m a v és w közötti úton kötődik be, akkor a (38.94) 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



38.8. ábra. Az m csúcsok lehetséges elhelyezkedése a fán.

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 38.8 tételt. ■

Gyakorlatok

38.5-1. 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.

38.5-2. Mutassuk meg, hogy minden ultrametrika egyben additív metrika is.

38.5-3. Adjunk példát olyan metrikára, amely nem additív.

38.5-4. Mutassuk meg, hogy minden additív metrika euklideszi.

38.5-5. 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.

38.5-6. Mutassuk meg, hogy a csúcsok számának négyzetével arányos időben eldönthető, hogy egy metrika additív-e, illetve ultrametrikus-e.

38.6. 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.

38.6.1. 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 gén-

ből 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.

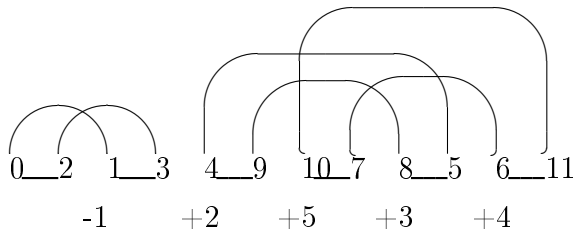
Ha feltesszük, hogy csak inverziók történtek, akkor meg tudunk adni egy $\Theta(n^2)$ idejű algoritmust, amely meghatároz egy olyan minimális mutációsorozatot, amely az egyik genomot a másikba transzformálja, sőt, a szükséges mutációk száma $\Theta(n)$ időben eldönthető, ahol n a gének száma.

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 permutációba. 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-



38.9. ábra. A -1, $+2$, $+5$, $+3$, $+4$ előjeles permutáció ábrázolása nem előjeles permutációként, és a permutáció töréspontgráfja.

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 abszolút értéke 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 38.9. 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.

38.9. tétel. *Legyen adva egy π előjeles permutáció. Inverziók egy optimális*

sorozata, amely ezt a permutációt rendezi,

$$b_\pi - c_\pi + h_\pi + f_\pi \tag{38.95}$$

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 (38.95) 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.

38.6.2. 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*** nukleinsavleolvasásnak, 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

38.6-1. Mutassuk meg, hogy ha egy permutáció erőd, akkor legalább három szupergát van benne.

38.6-2. Legalább hány elemből kell egy erődnek állnia?

Feladatok

38-1 Konkáv Smith–Waterman

Adjuk meg a Smith-Waterman algoritmust konkáv résbüntetésekre.

38-2 Konkáv Spouge

Adjuk meg Spouge algoritmusát konkáv résbüntetésekre.

38-3 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.

38-4 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.

38-5 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.

38-6 Rejtett Markov-modellek együttes kibocsátási valószínűsége
Ki lehet számolni ezt a valószínűséget $O((n_1 n_2)^2)$ időben?

38-7 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 [270]. Tetszőleges résbüntető függvényre Waterman és munkatársai adtak algoritmust [363]. Gotoh algoritmusa affin résbüntető függvényekre 1982-ben jelent meg [152], a konkáv résbüntető függvény ötlete Watermantól származik [359], mellyel később Miller és Myers [263], valamint Galil és Giancarlo [263] 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 [345].

A többszörös szekvenciaillesztés ötlete Sankofftól származik [318], mely a bioinformatika egyik központi feladata lett [162]. Bebizonyították, hogy a többszörös szekvenciaillesztés NP-teljes probléma [360], 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 [183], de mára számos bioinformatikai algoritmusban szerepel, például Doublescan [259]. A szekvenciaillesztés további algoritmikai elemzéséről kimerítően ír Gusfield [162].

A sztochasztikus nyelvtanok és bioinformatikai alkalmazásuk a központi témája Durbin, Eddy, Krogh és Mitchison népszerű könyvének, melyet számos egyetem bioinformatika oktatásában alapkönyvként használnak [106]. Formális nyelvekkel, nyelvtanokkal foglalkozó magyar nyelvű tankönyv Bach Iván [26] és Fülöp Zoltán [130] műve, valamint [82].

Az osztályozó algoritmusok összehasonlításáról részletesen olvashatunk Podani János könyvében [289]. A filogenetika és a filogenetikai algoritmusok iránt érdeklődők figyelmébe ajánljuk Felsenstein [123], valamint Semple és Steel [323] könyveit.

Kevéssé olvasmányos, de sok információt tartalmaz a genomátrendeződési algoritmusokról Pevzner könyve [287].

Stephen „String searching” című könyvében részletesen foglalkozik a legrövidebb szuperstring problémájával. A könyvben számos kiváló referenciát és az algoritmusok részletes leírásait is megtaláljuk [328].

Az alábbi megjegyzések a téma iránt kifejezetten érdeklődőknek szólnak.

Két string edit távolságán a beszúrások és törlések minimális számát értjük. Két string edit távolságának a kiszámolására van $\Theta(ln^2)$-nél gyorsabb módszer, amely a „négy orosz gyorsítása” néven híresült el, habár a négy egykori szovjetúnió-beli szerző közül csak egy volt orosz nemzetiségű [23]. 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 $a_i = b_j$. Van olyan, ezt a gráfot használó 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 [187].indexHunt, J. W. 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 [370]. A Unix `diff` utasításában használt algoritmus szintén diagonális kiterjesztésen alapul [262], 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 [215]. Landau és Vishkin módosított algoritmusa minden olyan illesztést megtalál M-ben, amely legfeljebb k helyen tér el P-től [224]. 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 Kececioglutól és munkatársaitól származik [208].. A módszert kiterjesztették tetszőleges résbüntető függvényre is [13]. Az egész értékű programozással kapcsolatos algoritmusokról írt áttekintő cikket

Lancia [223]. A DiAlign szintén nem dinamikus programozáson alapul [265].

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 [326].

Egy adott fa topológiára a Maximum Likelihood címkézés polinomiális időben megadható [298]. 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 [4].

A két legelterjedtebb, rejtett Markov-modellek alapján működő szekvenciaillesztő programcsomag a SAM [4] és a HMMER (http://hmmer.wustl.edu/). Rejtett Markov-modell alapján működő genomannotáló programot fejlesztett ki Pedersen és Hein [281], páros rejtett Markov-modellt használ szintén genomannotálásra a DoubleScan [259], (http://www.sanger.ac.uk/Software/analysis/doublescan/) és a Projector [260], (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 [148], 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 RNSek másodlagos térszerkezetét határozza meg [214]. Knudsen and J. 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 [239, 240].Nem ismert azonban, hogy erre a modellre a legvalószínűbb kibocsátás megkeresésére hasonló gyorsítás megadható-e.

Az RNS-ek Zuker-Tinoco [346] modellje térszerkezeti elemeken definiál szabadenergiát, és egy adott másodlagos térszerkezethez a térszerkezet részeihez rendelt szabadenergiák összegét rendeli. A Zuker-Sankoff-algoritmus [389] $\Theta(l^4)$ 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 [255]. Egy speciális esetben mind az optimális térszerkezet, mind a partíciófüggvény számolása felgyorsítható $\Theta(l^3)$ időre is, továbbra is $\Theta(l^2)$ memóriát használva [242].indexLyngso, R. Tinoco and O. and Uhlenbeck M. 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 [12, 241, 306, 350].indexLyngso, R. Rejtett Markov-modellek és sztochasztikus környezetfüggetlen nyelvtanok együttes kibocsátási valószínűségeinek meghatározásával foglalkoztak Jagota és munkatársai [194]. 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 állapotainak száma. A szerzők megadtak egy iterációt, mely bizonyítottan konvergál a pontos megoldáshoz, egy iterációs lépés számolási igénye $\Theta(V^3 n^5)$. 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 [269]. 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 [178].

Atteson matematikai definíciót adott a törzsfakészítő módszerek jóságára, és kimutatta, hogy bizonyos definíciók alapján a SZOMSZÉDOK EGYESÍTÉSE a lehető legjobb módszer [25].

Négy fajra három fa topológiát lehet készíteni, amelyeket quarteteknek hívunk. Ha egy fára ismerjük az összes quartet topológiáját, 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 [116]. 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 [172]. Transzlokációs átmérővel és a probléma általánosításával foglalkozik Pisanti és Sagot [288].

A permutációk rendezésének általánosított problémája a minimális generáló szó keresése egy véges csoportban. Ez a probléma bizonyítottan NP-teljes [197]. 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 [74]. Érdekességként megjegyezzük, hogy a Microsoft tulajdonosa, Bill Gates is foglalkozott permutációk rendezésével, speciálisan a prefix inverziókkal [143]. A könyvfejezetben csak a legfontosabb témákról írtunk, számos kevésbé fontos, de nagyon érdekes témakört kénytelenek voltunk kihagyni. Ilyen témakörök például a rekombináció, a pedigré elemzés, a karakteralapú törzsfakészítő módszerek, a részleges emésztés, a fehérjecsavarás, DNS chipek, a tudásábrázolás, a biokémiai hálózatok. Ennek fényében Donald Knuth szavaival [103] 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."

39. 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.

39.1. 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.

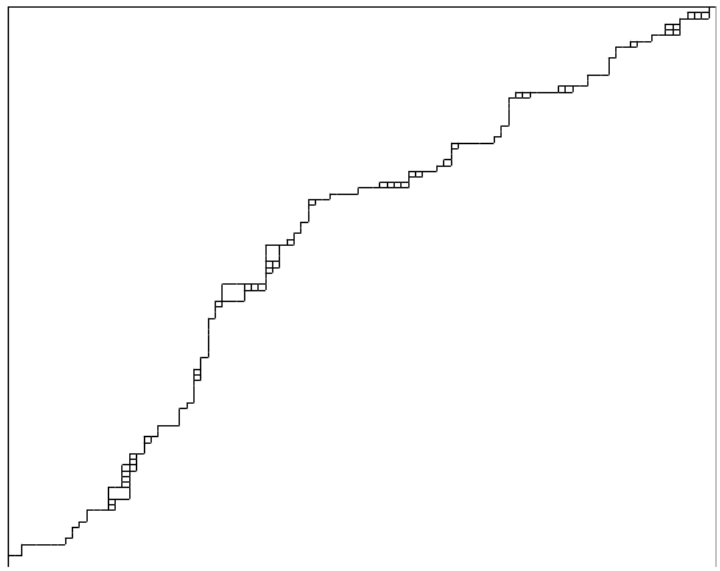
A több választási lehetőséget kínáló rendszerek különösen hasznosak lehetnek azoknál a valós idejű problémáknál, amikor alapjában véve elegendő idő áll rendelkezésre a teljes megoldás kiszámolására, de a feladat bizonyos paraméterei ismeretlenek vagy fuzzy jellegűek. Ezek konkrét értékét csak egy meglehetősen 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 39.1. 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 ecsettel rajzoltak. (A 39.2. 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” értelem-



39.1. ábra. 1000 legrövidebb út egy 100×100-as rács-gráfban, egymásra nyomtatott megjelenítésben.

ben. 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.

39.1.1. 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 okoznak, 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 keresőmotorok. A legtöbb esetben az internetes keresőmotorok rengeteg találattal térnek vissza, amelyeket egy átlagos felhasználó nem tud, és nem is akar végignézni. Ezért az ilyen keresőmotorok 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 meg lehet takarítani. Az utóbbi években ezek a gravitáció által segített röppályák egyre bonyolultabbak lettek, és 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 röppály 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 Élőpont 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 vasútál-

lomáson, 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 észrevegye az ilyen trükköket, 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

39.1-1. 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.

39.2. Több lehetséges megoldás előállítása

39.2.1. 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, amelyekből néhányat 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 $T = (i_1, \ldots, i_p, i_q, i_{q-1}, \ldots, i_{p+1}, i_{q+1}, \ldots, i_N)$
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 $T = (i_1, i_2)$
3 **for** $t = 2$ **to** $N - 1$
4 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 $T = (T(1), \ldots, T(s), i_{t+1}, T(s+1), \ldots, T(t))$
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 39.2.6. 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 39.1.1. 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 in-

terfészekkel. Valójában azonban mindegyik négy független program kernel és adatbázis valamelyikén alapult.

39.2.2. 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.

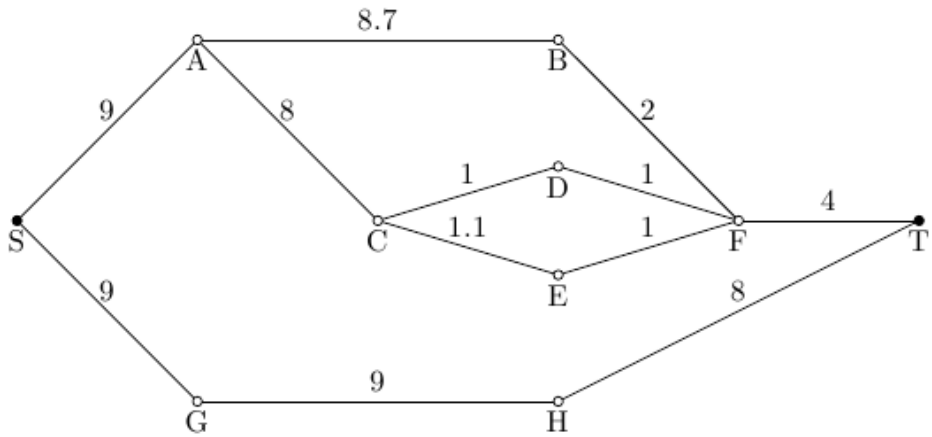
(ii) R_2-nek nem szabad nagyon hasonlítania az eredeti megoldásra. Ellenkező esetben nem beszélhetünk *valódi* alternatíváról. Az úgynevezett *mikro mutációk* esetén nagy az esélye annak, hogy az összes egymáshoz hasonló megoldásjelölt ugyanazzal a gyenge ponttal rendelkezik. A példánkban a „hasonlóság 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      if e R_1-hez tartozik
4          ŵ(e) ← w(e) · (1 + ε)
5          else ŵ(e) = w(e)
```



39.2. ábra. A39.1, 39.2 és 39.6 példákhoz tartozó gráf.

6 keressük meg az s-ből t-be vezető R_2 legrövidebb utat a $\widehat{G} = (V, E, \widehat{w})$ 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.

39.1. példa. Adott egy $G = (V, E)$ gráf súlyozott élhosszakkal. A 39.2. ábrán az élek hosszát a mellékük 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 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* $\widehat{w}(e) \leftarrow aw(e) + c$

39.2. példa. Adott a 39.1 példából már ismert $G = (V, E)$ gráf (lásd a 39.2. 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.

39.1. 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.

39.2.3. Példák $\sum$-típusú problémákra

- Legrövidebb út probléma
- Hozzárendelési probléma
- Utazóügynök probléma
- Hátizsák probléma
- Sorozat csoportosítási probléma

39.3. 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.

39.2.4. A büntető módszer absztrakt megfogalmazása $\sum$-típusú problémákra

39.2. 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.

39.4. példa. Adjuk meg a formális definícióját a 39.1 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     B = {B_0}
5          E = (0)
6          Border = ∅
        // B_0 minimalizálja a w és p függvényeket és minden ε-ra optimális
7     else 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               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                  Border[i] ← 1
16                  else 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.

39.5. 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 $0 \le p \le 1$ paramétert ***hiba valószínűségnek***, a $c \ge 1$ 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.

$$\phi_{\varepsilon_i} = \frac{\min\{\hat{w}(S_0), \hat{w}(S_{\varepsilon_i})\}}{\hat{w}(S_0)} \ (i = 1, \ldots, 30).$$

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 39.3. 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 39.3. 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.

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.)

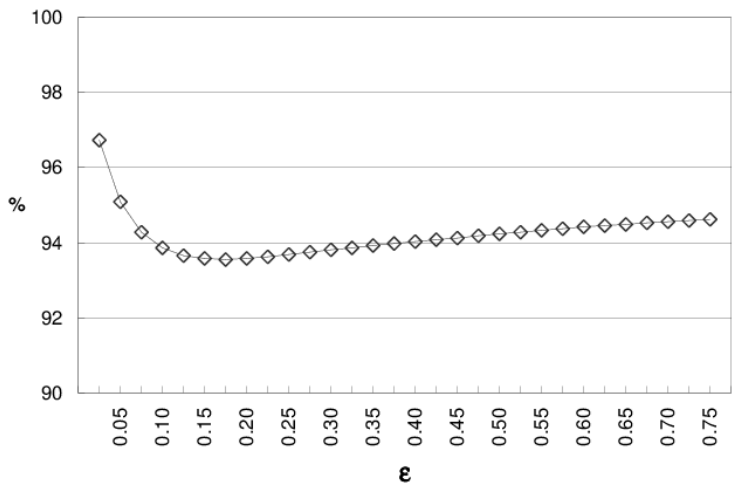
A büntetéses megoldások monotonitási tulajdonságai

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.



39.3. ábra. $\overline{\phi}_{\varepsilon_i}$ értékei $\varepsilon_1 = 0,025,\ \varepsilon_2 = 0,050,\ \ldots,\ \varepsilon_{30} = 0,750$-re 25×25-ös rácson.

39.3. 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 39.2. 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 $0 \leq \delta < \varepsilon$.

B_δ és B_ε definíciójából adódóan a következő egyenlőtlenségek teljesülnek.

1. $\varepsilon < \infty$ esetén
$$w(B_\varepsilon) + \varepsilon \cdot p(B_\varepsilon) \leq w(B_\delta) + \varepsilon \cdot p(B_\delta) \ , \tag{39.1}$$
$$w(B_\varepsilon) + \delta \cdot p(B_\varepsilon) \geq w(B_\delta) + \delta \cdot p(B_\delta) \ . \tag{39.2}$$
(39.2)-t kivonva (39.1)-ből a következőt kapjuk:
$$(\varepsilon - \delta) \cdot p(B_\varepsilon) \leq (\varepsilon - \delta) \cdot p(B_\delta) \qquad | : (\varepsilon - \delta) > 0$$
$$\Leftrightarrow \quad p(B_\varepsilon) \leq p(B_\delta) \ . \tag{39.3}$$
$\varepsilon = \infty$ esetén (39.3)) egyenlőtlenség közvetlenül következik a B_∞ definíciójából.
2. Vonjuk ki (39.3)-at (39.2)-ből megszorozva δ-val, ekkor a következőt kapjuk:
$$w(B_\varepsilon) \geq w(B_\delta) \ . \tag{39.4}$$
3. Vonjuk ki (39.3)-at (39.4)-ből, ekkor a következőt kapjuk:
$$w(B_\varepsilon) - p(B_\varepsilon) \geq w(B_\delta) - p(B_\delta) \ .$$
4. (39.2)-ből az $\varepsilon > \delta \geq 0$ egyenlőtlenség felhasználásával a
$$w(B_\delta) + \delta \cdot p(B_\delta) \leq w(B_\varepsilon) + \delta \cdot p(B_\varepsilon) \leq w(B_\varepsilon) + \varepsilon \cdot p(B_\varepsilon)$$
$$\Rightarrow \quad w(B_\varepsilon) + \varepsilon \cdot p(B_\varepsilon) \geq w(B_\delta) + \delta \cdot p(B_\delta) \tag{39.5}$$

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 ($i \neq j$).

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 $p_1(B) = \varepsilon \cdot w(B \cap S_0)$ 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** $j = 2$ **to** k
5 $\quad p_j(B) = \varepsilon \cdot w(B \cap S_0) + \varepsilon \cdot w(B \cap S_1) + \cdots + \varepsilon \cdot w(B \cap S_{j-1})$
6 $\quad$ 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:

$$5^* \qquad p_j(B) = \varepsilon \cdot w(B \cap (S_0 \cup S_1 \cup \cdots \cup S_{j-1}))$$

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.

39.6. példa. Vegyük ismét a 39.2. 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.

39.2.5. 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:

$$\min \sum_{e \in E} w(e) \cdot x_e, \text{feltéve, hogy}$$

$$\sum_{j \in S(s)} x_{(s,j)} - \sum_{j \in P(s)} x_{(j,s)} = 1 \quad \text{kimenő feltétel az } s \text{ kezdőpontra vonatkozóan,}$$

$$\sum_{j \in S(t)} x_{(t,j)} - \sum_{j \in P(t)} x_{(j,t)} = -1 \quad \text{bemenő feltétel a } t \text{ végpontra vonatkozóan,}$$

$$\sum_{j \in S(i)} x_{(i,j)} - \sum_{j \in P(i)} x_{(j,i)} = 0 \quad \text{minden további } i \in V \backslash \{s, t\} \text{pontra}$$

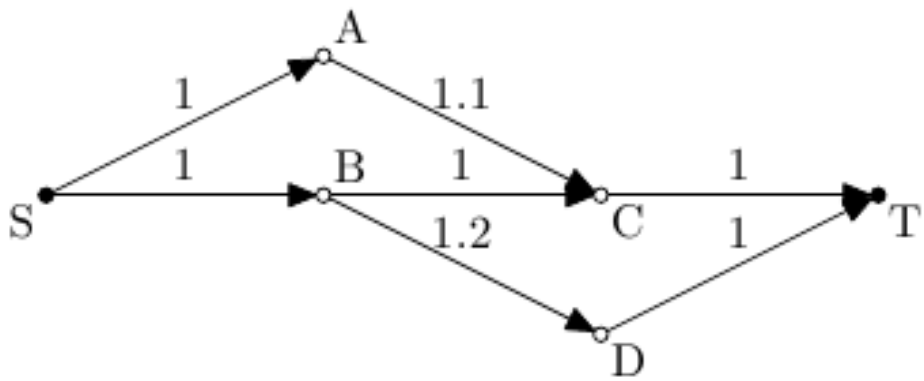
Kirchhoff-feltételek a belső pontokra

$$0 \leq x_e \leq 1 \text{ minden } e \in E\text{-re .}$$

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.

39.7. példa. Vegyük a 39.4. ábrán látható gráfot. A legrövidebb útvonal problémához tartozó lineáris programozási feladat most hat egyenlőség feltételt tartalmaz



39.4. ábra. Példa gráf az LP-büntető módszerhez.

(minden csomópontra egyet), és hét egyenlőtlenség párt (minden élre egy párt).

$$\begin{aligned}
&\min(x_{SA}+x_{SB}+x_{BC}+x_{CT}+x_{DT})\cdot 1 + x_{AC}\cdot 1.1 + x_{BD}\cdot 1.2 \\
\text{feltéve, hogy } & x_{SA}+x_{SB}=1\ , \\
& x_{CT}+x_{DT}=1\ , \\
& x_{SA}-x_{AC}=0\ , \\
& x_{SB}-x_{BC}-x_{BD}=0\ , \\
& x_{AC}+x_{BC}-x_{CT}=0\ , \\
& x_{BD}-x_{DT}=0\ , \\
& 0\le x_{SA},x_{SB},x_{AC},x_{BC},x_{BD},x_{CT},x_{DT}\le 1\ .
\end{aligned}$$

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 $LP_{\text{2-rövid-út}}$ 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
$$\sum_{j\in S(s)} x_{(s,j)} + y_{(s,j)} - \sum_{j\in P(s)} x_{(j,s)} + y_{(j,s)} = 2 \quad \text{feltétel az } s \text{ kezdőpontra vo}$$
$$\sum_{j\in S(t)} x_{(t,j)} + y_{(t,j)} - \sum_{j\in P(t)} x_{(j,t)} + y_{(j,t)} = -2 \quad \text{feltétel a } t \text{ végpontra vonat}$$
$$\sum_{j\in S(i)} x_{(i,j)} + y_{(i,j)} - \sum_{j\in P(i)} x_{(j,i)} + y_{(j,i)} = 0 \quad \text{Kirchhoff-feltételek}$$
$$\text{minden további pontra } i \in$$

$$0 \le x_e, y_e \le 1 \text{ minden } e \in E\text{-re} .$$

39.8. példa. Tekintsük ismét a 39.4. 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.

$$\begin{aligned}
\min \quad & (x_{SA}+x_{SB}+x_{BC}+x_{CT}+x_{DT})\cdot 1 + x_{AC}\cdot 1.1 + x_{BD}\cdot 1.2 \\
& +\big[(y_{SA}+y_{SB}+y_{BC}+y_{CT}+y_{DT})\cdot 1 + y_{AC}\cdot 1.1 + y_{BD}\cdot 1.2\big]\cdot(1+\varepsilon) \\
\text{feltéve, hogy } & x_{SA}+y_{SA}+x_{SB}+y_{SB} = 2 , \\
& x_{CT}+y_{CT}+x_{DT}+y_{DT} = 2 , \\
& x_{SA}+y_{SA}-x_{AC}-y_{AC} = 0 , \\
& x_{SB}+y_{SB}-x_{BC}-y_{BC}-x_{BD}-y_{BD} = 0 , \\
& x_{AC}+y_{AC}+x_{BC}+y_{BC}-x_{CT}-y_{CT} = 0 , \\
& x_{BD}+y_{BD}-x_{DT}-y_{DT} = 0 , \\
& 0 \le x_{SA}, x_{SB}, x_{AC}, x_{BC}, x_{BD}, x_{CT}, x_{DT}, y_{SA}, y_{SB}, y_{AC}, y_{BC}, y_{BD}, y_{CT}, y_{DT} \le 1 .
\end{aligned}$$

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?

39.4. 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

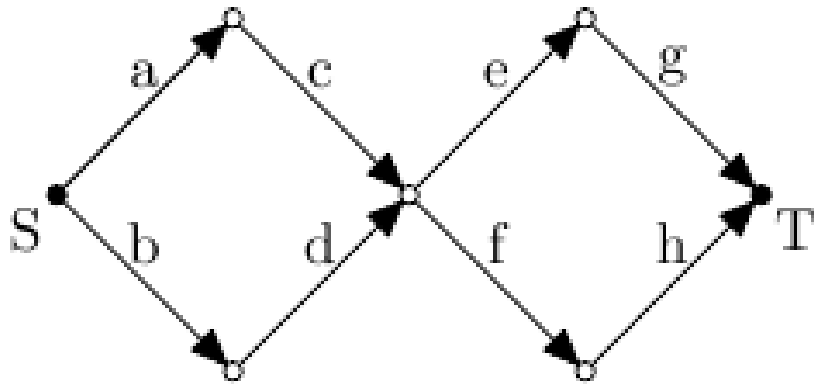
1. $E_1 \cap E_2 = \emptyset$, $E_1 \cap E_3 = \emptyset$ *és* $E_2 \cap E_3 = \emptyset$,
2. $x_e = 1$, $y_e = 0$ *minden* $e \in E_1 \cup E_2$,
3. $x_e = 1$, $y_e = 1$ *minden* $e \in E_3$,
4. $x_e = 0$, $y_e = 0$ *minden* $e \notin E_1 \cup E_2 \cup E_3$.
5. $E_1 \cup E_3$ *egy* P_1 *s-ből t-be vezető 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.*
6. *Nem létezik olyan* (Q_1, Q_2) *útvonal pár, amelyik jobb lenne* (P_1, P_2)*-nél, azaz*

$$w(P_1) + w(P_2) + \varepsilon \cdot w(P_1 \cap P_2) \leq w(Q_1) + w(Q_2) + \varepsilon \cdot w(Q_1 \cap Q_2),$$
minden (Q_1, Q_2) *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 39.5. 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 39.4. 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 39.4. 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



39.5. ábra. Példa két útvonal nem egyértelmű dekompozíciójára.

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:

$$\min f(x^0, \ldots, x^{k-1}) := \sum_{e \in E} \sum_{j=0}^{k-1} (1 + j \cdot \varepsilon) \cdot w(e) \cdot x_e^j$$

vagy

$$\min f(x^0, \ldots, x^{k-1}) := \sum_{e \in E} \sum_{j=0}^{k-1} (1 + \varepsilon)^j \cdot w(e) \cdot x_e^j \,.$$

- 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.

39.2.6. Büntető módszer heurisztikák alkalmazásával

A 39.2.2. 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.

39.9. példa. Egy jól ismert heurisztika az utazóügynök problémára a lokális keresés kettős cserével (lásd a 39.2.1. 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 39.5 példában már ismertetetthez hasonló kísérletet végeztek el az utazóügynök problémára 25 véletlenül kiválasztott ponttal az egységnégyzetben. A 39.6. ábra az arányosított átlagokat mutatja az $\varepsilon_0 = 0.000$, $\varepsilon_1 = 0.025$, $\ldots$, $\varepsilon_{30} = 0.750$ 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.

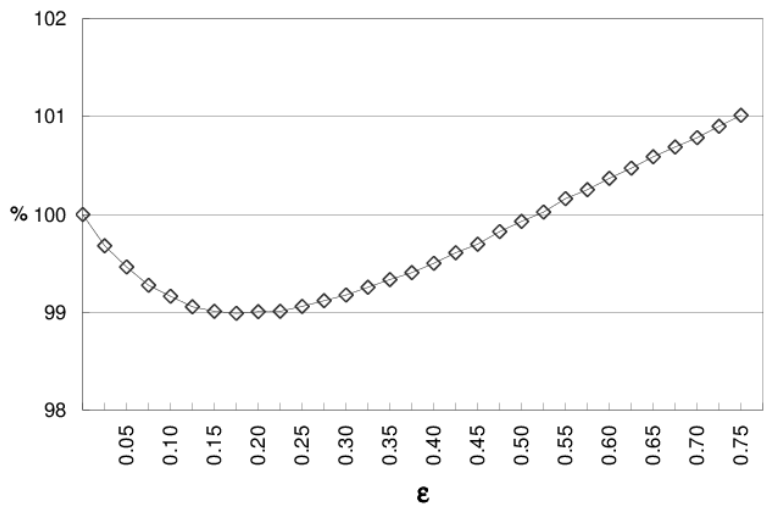
További kísérletek azt is kimutatták, hogy az ε^* optimális paraméter a probléma méretének növekedésével csökken.

Gyakorlatok

39.2-1. A következő, utazóügynök problémára vonatkozó programozási gyakorlat segít abban, hogy jobban átérezzük a lokális optimumok nagy változatosságát. Generáljunk véletlenszerűen 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.

39.2-2. Adjuk meg ugyanazokat a kulcsszavakat különböző internetes keresőmotoroknak. Hasonlítsuk össze a találati listákat, és azok változatosságát.

39.2-3. Formalizáljuk az utazóügynök problémát egy $\sum$-típusú proble-



39.6. ábra. $\overline{\phi}_{\varepsilon_i}$ az $\varepsilon_0 = 0$, $\varepsilon_1 = 0.025$, $\ldots$, $\varepsilon_{30} = 0.750$ értékekre 25×25-ös rácson.

maként.

39.2-4. Bizonyítsuk be a 469. oldalon található megjegyzésben foglalt állítást.

39.2-5. Hogyan néz ki a $p(e)$ büntető függvény additív büntetés esetén, mint például a 39.2 példában?

39.2-6. Bizonyítsuk be a 471. oldalon levő (1) és (2) tulajdonságokat.

39.2-7. Alkalmazzuk az OSZD MEG ÉS FEDD LE algoritmust a 39.2 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.

39.2-8. Keressünk olyan $\varepsilon > 0$ büntető paramétert a 476. oldalon levő 39.6 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.

39.3. 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.

39.3.1. 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        határozzuk meg f Max_d maximumát V_d-n
5        if Max_d > Opt
6           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.

39.3.2. 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.

39.3.3. 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.

39.3.4. 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.

39.3.5. 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

39.3-1. 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 39.2.1. pontban) és számoljuk össze, hogy melyik él hányszor fordul elő ebben a száz megoldásban. Definiáljunk egy K küszöböt (például $K = 30$) és rögzítsük 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” [285]. Sameith [314] technikai jelentésében számos kísérletet, leírást és elemzést találunk a büntető módszerre, különböző összeg típusú problémák, dimenziók és hibaszélességek esetére. A 39.3. tétel bizonyítása először [15]-ben jelent meg. Az e-kereskedelemben a több választási lehetőséget kínáló rendszereket gyakran *tanácsadó rendszereknek* nevezik – mint például Resnick és Varian cikkében [301] – szem előtt tartva a vevőket, akik számára az őket érdeklő termékeket ki kell listázni. Érthető okokból a kereskedelmi keresőmotorok és az e-cégek titokban tartják a kivonatoló algoritmusaikat. A 39.2.5. pontban említett áramlástani tétel megtalálható Ahuja, Magnanti és Orlin könyvében [11]. Egy útvonaltervező programot forgalmaz az AND cég [19]. Műholdas felvételeken alapuló útvonaltervezéssel foglalkozik Berger [43] diplomamunkája.

A BigBlackCell nevű szoftver Grosse és Schwarz munkája [160].

A genetikus algoritmusokról például Goldberg [147] tekinthető jó tankönyvnek. Az interaktív evolúciót és a generatív tervezést Banzhaf [29] tárgyalja. A több feltételes döntéshozatallal több cikk is részletesen foglalkozik, az egyik alapmű Gal, Stewart és Hanne könyve [139].

Althöfer könyvében [17] a 3-agy történeti hátteréről és a versenysakkban elért sikereiről olvashatunk. A 3-agy és Juszupov nagymester közötti mérkőzésről [17] számol be. [14] általános tájékoztatást ad arról az esetről, amikor több számítógép javaslatát használva javítjuk a játékerőt. [16] 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. [180] a sakkprogramok és más bonyolulta játékok technikai hátterét mutatja be.

M. Zuker és D. H. Turner által írt programok értékes online 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 [234], a kapcsolódó biológiai fogalmak megismeréséhez pedig Podani János könyvét [289] ajánljuk.

40. 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 fejezetben csak a látható alakzatok meghatározásával foglalkozunk, az alakzatok színét ismertnek tételezzük fel.

Először áttekintjük, hogyan írhatjuk le az alakzatokat számokkal, majd megismerkedünk a fényképezési folyamat algoritmusaival.

40.1. 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.

40.1. 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:

$$\vec{v}_1 \cdot \vec{v}_2 = |\vec{v}_1| \cdot |\vec{v}_2| \cdot \cos\alpha, \qquad \text{ahol } \alpha \text{ a } \vec{v}_1 \text{ és } \vec{v}_2 \text{ vektorok által bezárt szög .}$$

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:

$$\vec{v}_1 \times \vec{v}_2 = \vec{v}, \qquad \text{ahol } \vec{v} \text{ merőleges } \vec{v}_1, \vec{v}_2\text{-re, és } |\vec{v}| = |\vec{v}_1| \cdot |\vec{v}_2| \cdot \sin\alpha \ .$$

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.

40.1.1. 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

40.1-1. Igazoljuk, hogy a Descartes-koordináták és a pontok egy-egyértelmű kapcsolatban állnak.

40.1-2. Bizonyítsuk be, hogy ha a bázisvektorok egységnyi hosszúak és egymásra páronként merőlegesek, akkor $(x_1, y_1, z_1)\cdot(x_2, y_2, z_2) = x_1x_2+y_1y_2+z_1z_2$.

40.1-3. Igazoljuk, hogy a skaláris szorzás az összeadásra nézve disztributív.

40.2. Ponthalmazok leírása egyenletekkel

A koordinátarendszerek lehetőséget adtak pontok számokkal történő megadására. Egy koordinátákra megfogalmazott feltételrendszerrel pedig egy teljes ponthalmazt azonosíthatunk. A feltételrendszer általában egyenlet vagy egyenlőtlenség, amelyet kielégítő koordináta-hármasokhoz tartozó pontokat mondjuk a ponthalmazhoz tartozónak.

40.2.1. 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

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***	$r^2-z^2-(R-\sqrt{x^2+y^2})^2$

40.1. ábra. Néhány – origó középpontú testet definiáló – implicit függvény.

hagyja az elfajuló részeket.

A 40.1. ábra bemutatja a gömböt, téglatestet és tóruszt definiáló implicit függvényt.

40.2.2. 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:

$$(\vec{r}-\vec{r}_0)\cdot\vec{n}=0, \qquad n_x\cdot x+n_y\cdot y+n_z\cdot z+d=0\ , \tag{40.1}$$

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:

$$x=x(u,v),\quad y=y(u,v),\quad z=z(u,v),\quad u\in[u_{\min},u_{\max}],\quad v\in[v_{\min},v_{\max}]\ .$$

A felület paraméteres egyenleteiből az implicit egyenletet úgy származtathatjuk, hogy a három egyenletből kiküszöböljük az u,v szabad paramétereket. A 40.2. ábra bemutatja a gömböt, hengert és kúpot definiáló paraméteres formát.

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***	$R\cdot(1-v)\cdot\cos 2\pi u$	$R\cdot(1-v)\cdot\sin 2\pi u$	$h\cdot v$

40.2. ábra. Néhány – felületet definiáló – paraméteres forma, ahol $u,v\in[0,1]$.

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

$$\vec{r}(\alpha,\beta,\gamma)=\alpha\cdot\vec{p}_1+\beta\cdot\vec{p}_2+\gamma\cdot\vec{p}_3,\qquad\text{ahol}\quad\alpha,\beta,\gamma\geq 0\quad\text{és}\quad\alpha+\beta+\gamma=1\ .$$

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:

$$\vec{r}(u,v)=u\cdot\vec{p}_1+v\cdot\vec{p}_2+(1-u-v)\cdot\vec{p}_3,\qquad\text{ahol}\quad u,v\geq 0\quad\text{és}\quad u+v\leq 1\ .$$

40.2.3. 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 skaláregyenletnek 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.

A 40.3. ábra bemutatja az ellipszist, csavarvonalat és szakaszt definiáló paraméteres formát.

Figyeljük meg, hogy a felületeken úgy is felvehetünk görbéket, hogy az

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 ***csv***	$R \cdot \cos 2\pi t$	$R \cdot \sin 2\pi t$	$h \cdot t$
(x_1, y_1, z_1) és (x_2, y_2, z_2) közötti ***sz***	$x_1(1-t) + x_2 t$	$y_1(1-t) + y_2 t$	$z_1(1-t) + z_2 t$

40.3. ábra. Néhány – görbét definiáló – paraméteres forma, ahol $t \in [0,1]$ (ell = ellipszis, csv = csavarvonal, sz = szakasz.

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:***

$$\vec{r}(t) = r_0 + \vec{v} \cdot t, \qquad t \in (-\infty, \infty)\ . \tag{40.2}$$

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:***

$$\vec{r}(t) = \vec{r}_1 + (\vec{r}_2 - \vec{r}_1) \cdot t = \vec{r}_1 \cdot (1-t) + \vec{r}_2 \cdot t, \qquad t \in [0,1]\ . \tag{40.3}$$

Ezen definíció szerint a szakasz pontjai a végpontjainak ***konvex-kombinációi.***

40.2.4. 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 (40.1) egyenlettel összevetve megállapíthatjuk, hogy a felület Taylor-soros közelítésével kapott sík normálvektora éppen

$$\vec{n} = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right) = \mathrm{grad} f \, . \tag{40.4}$$

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 (40.2) 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

$$\vec{n} = \frac{\partial \vec{r}}{\partial u} \times \frac{\partial \vec{r}}{\partial v} \, . \tag{40.5}$$

40.2.5. 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 $\vec{r} = \vec{r}(t)$ $(t_0 \leq t \leq t_m)$ 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:

$$\sum_{i=0}^{m} B_i(t) = 1 \ .$$

Adott t-re a görbe pontját ezen mechanikai rendszer súlypontjaként értelmezzük:

$$\vec{r}(t) = \frac{\sum_{i=0}^{m} B_i(t) \cdot \vec{r}_i}{\sum_{i=0}^{m} B_i(t)} = \sum_{i=0}^{m} B_i(t) \cdot \vec{r}_i \ .$$

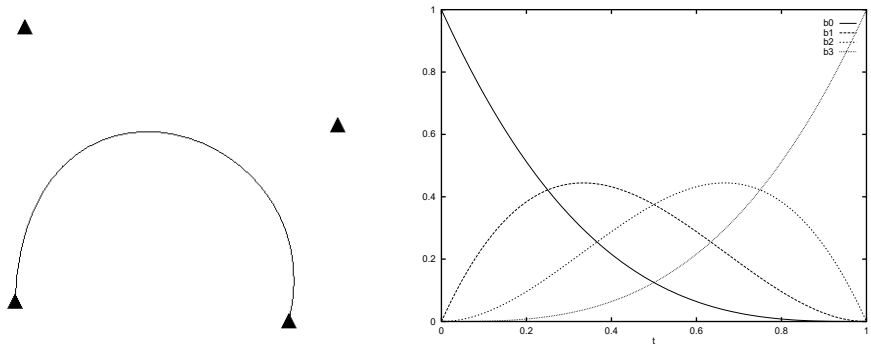
Figyeljük meg, hogy azért érdemes mindig egységnyi tömegű homokot szétosztani, mert ekkor a tört nevezője egységnyi lesz. A vezérlőpontokat kis mágnesekként is elképzelhetjük, amelyek a bázisfüggvényeknek megfelelő erővel vonzzák a görbét. Ha a bázisfüggvények nem negatívak, akkor ez az erő sohasem lehet 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

[1] Definíciószerűen egy pontrendszer konvex burka az a minimális konvex halmaz, amely tartalmazza a pontrendszert.



40.4. ábra. Négy vezérlőpont által definiált Bézier-görbe és bázisfüggvényei ($m = 3$).

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 ($i = 0, 1, \ldots, m$):

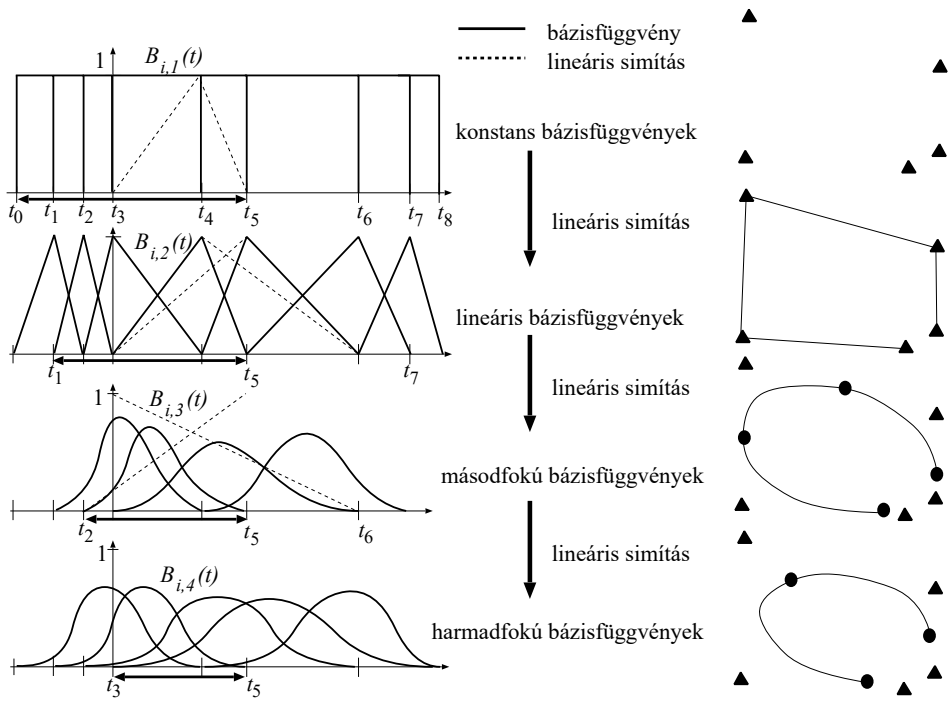
$$B_{i,m}^{\text{Bezier}}(t) = \binom{m}{i} \cdot t^i \cdot (1-t)^{m-i} \ . \tag{40.6}$$

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 40.4. 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:

$$\mathbf{t} = [t_0, t_1, \ldots, t_{m+k}], \quad t_0 \leq t_1 \leq \cdots \leq t_{m+k} \ .$$



40.5. 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 (40.5. 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 (40.5. 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 $t_{i+1} \leq t < t_{i+2}$ 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.

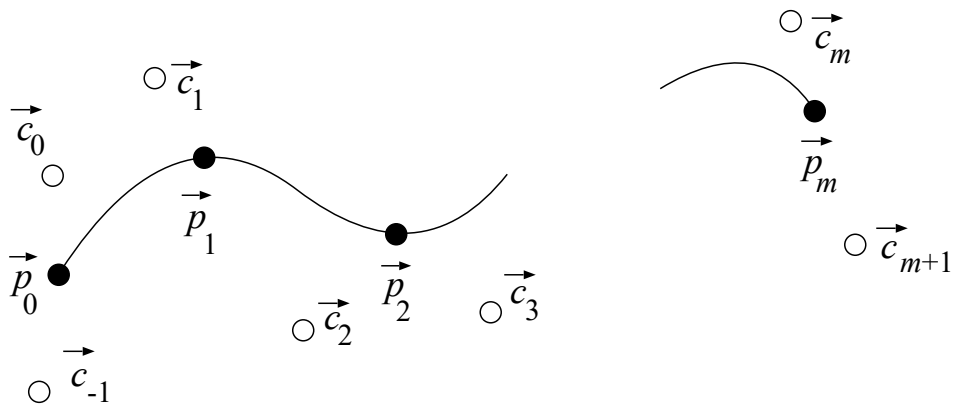
A másodrendű bázisfüggvények elsőfokú polinomok. A bázisfüggvények fokszámát, azaz a görbe rendjét, az ismertetett eljárás rekurzív alkalmazásával tetszőleges szintig növelhetjük. A következő rend bázisfüggvényei az alábbi módon függenek az előzőtől:

$$B_{i,k}^{\mathrm{BS}}(t) = \frac{(t - t_i)B_{i,k-1}^{\mathrm{BS}}(t)}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - t)B_{i+1,k-1}^{\mathrm{BS}}(t)}{t_{i+k} - t_{i+1}}, \qquad \text{ha } k > 1\ .$$

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 ***CoxdeBoor algoritmussal*** számíthatjuk:

[2] Akkor mondjuk, hogy a bázisfüggvény egy intervallumba lóg, ha értéke ebben az intervallumban nem azonosan zérus.



40.6. 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.

B-SPLINE$(i, k, t, \mathbf{t})$

```
1  if k = 1                                          // Triviális eset vizsgálata.
2     if t_i ≤ t < t_{i+1}
3        return 1
4        else return 0
5  if t_{i+k-1} − t_i > 0
6     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     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
```

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 (40.6. 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*** (**N**on**U**niform **R**ational **B**-**S**pline) 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:

$$\vec{r}(t) = \frac{\sum_{i=0}^{m} w_i B_i^{\mathrm{BS}}(t) \cdot \vec{r}_i}{\sum_{j=0}^{m} w_j B_j^{\mathrm{BS}}(t)} = \sum_{i=0}^{m} B_i^{\mathrm{NURBS}}(t) \cdot \vec{r}_i \; .$$

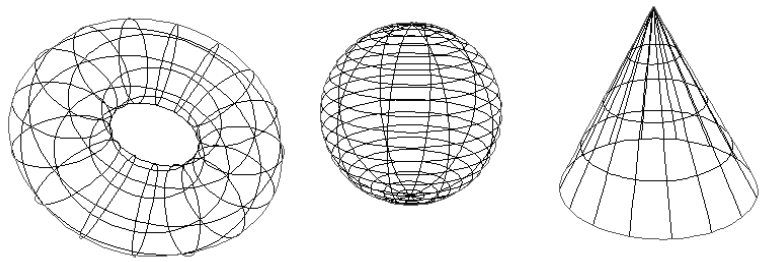
A B-spline és a NURBS bázisfüggvények közötti kapcsolat tehát a következő:

$$B_i^{\mathrm{NURBS}}(t) = \frac{w_i B_i^{\mathrm{BS}}(t)}{\sum_{j=0}^{m} w_j B_j^{\mathrm{BS}}(t)} \; .$$

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.

40.2.6. Felületmodellezés

A parametrikus felületek kétváltozós $\vec{r}(u, v)$ függvényekkel adhatók meg. A függvény közvetlen megadása helyett véges számú $\vec{r}_{ij}$ vezérlőpontot veszünk



40.7. ábra. Felületek izoparametrikus – azaz különböző u és v értékek rögzítésével kapott – vonalai.

fel, amelyeket a bázisfüggvényekkel súlyozva kapjuk meg a felületet leíró függvényeket:

$$\vec{r}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \vec{r}_{ij} \cdot B_{ij}(u,v) \ . \tag{40.7}$$

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:

$$\vec{r}_v(u) = \sum_{i=0}^{n} B_i(u) \cdot \vec{r}_i \ , \tag{40.8}$$

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 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 (40.8) 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.

40.2.7. 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 (40.8. 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 (40.8. 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.

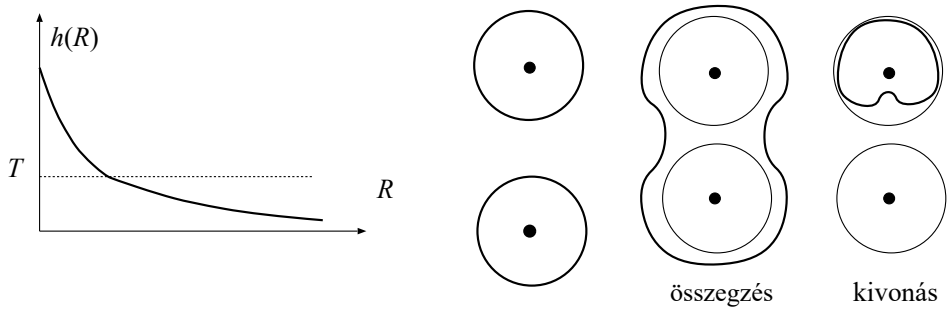
40.2.8. Konstruktív tömörtest geometria

A testmodellezés másik irányzata, a ***konstruktív tömörtest geometria*** az összetett testeket primitív testekből halmazműveletek (egyesítés, metszet, különbség) ismételt alkalmazásával építi fel (40.9. és 40.10. ábra). A primitív testek általában a téglatestet, a gömböt, a gúlát, a kúpot, a félteret stb. foglalják magukban, amelyek implicit függvényei ismertek.

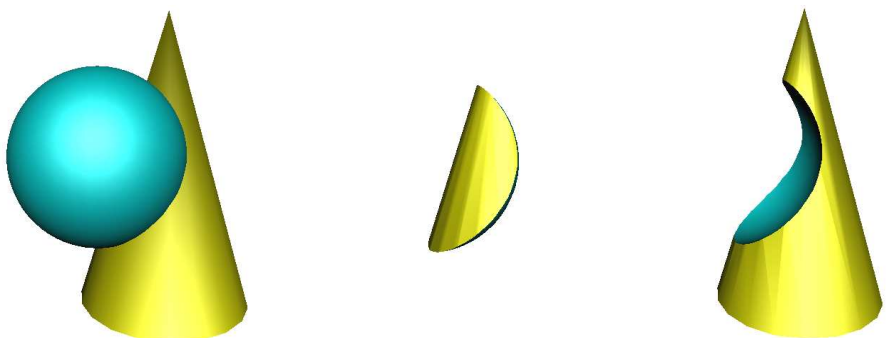
A halmazművelet eredményének implicit függvényét a résztvevő testek implicit függvényeiből kifejezhetjük:

- 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



40.8. ábra. A hatás a távolsággal csökken. Az azonos előjelű hatásfüggvények által leírt gömbök összeolvadnak, a különböző előjelűek csökkentik egymás hatását.



40.9. 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 ($\min(f,-g)$).

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

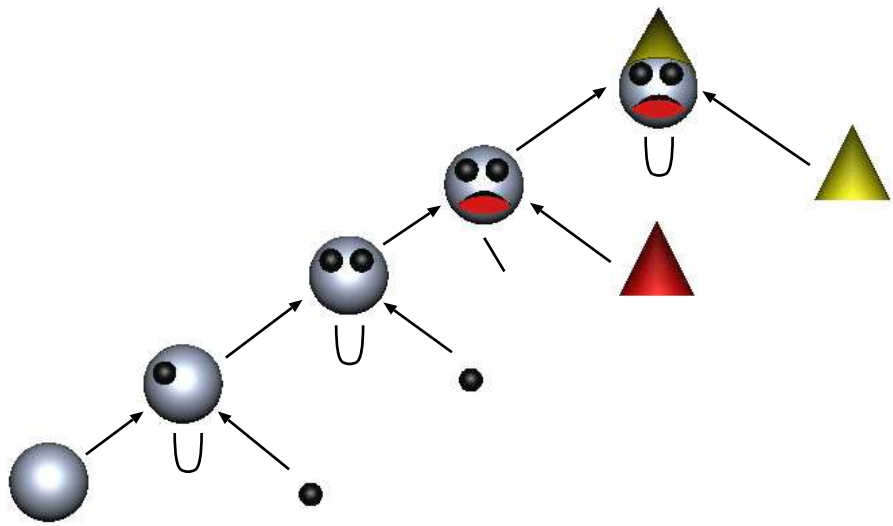
$$f^{morph}(x,y,z) = t \cdot f_1(x,y,z) + (1-t) \cdot f_2(x,y,z)$$

egyenlettel állíthatunk elő.

Gyakorlatok

40.2-1. Írjuk fel a tórusz paraméteres egyenletét.

40.2-2. Bizonyítsuk be, hogy a 4 vezérlőpontra illeszkedő, [0,0,0,0,1,1,1,1]



40.10. 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).

csomóvektorú, negyedrendű B-spline egy Bézier-görbe.

40.2-3. Adjuk meg a lobogó zászló és az egy pontból induló hullámfelület paraméteres egyenleteit, és a normálvektoraikat is egy tetszőleges pontban.

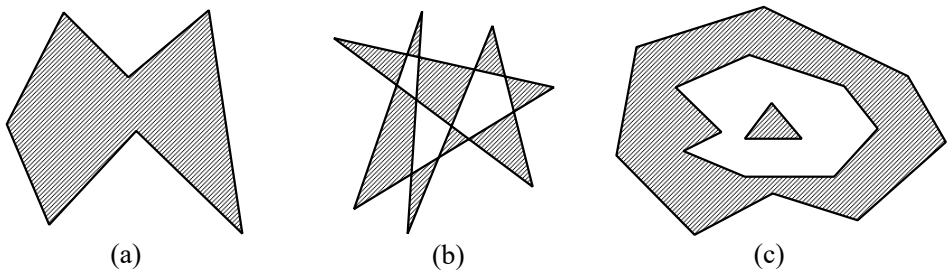
40.2-4. Bizonyítsuk be, hogy a Bézier-görbe érinti az első két és utolsó két vezérlőpont közötti szakaszokat.

40.2-5. Vezessük le a másod-, harmad- és negyedrendű B-spline görbe bázisfüggvényeinek képletét.

40.2-6. 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.

40.3. Geometriai feldolgozó és tesszellációs algoritmusok

A 40.2. 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 is-



40.11. ábra. A sokszögek fajtái. **(a)** egyszerű; **(b)** nem egyszerű, egyszeresen összefüggő; **(c)** többszörösen összefüggő.

mertetünk, 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.

40.3.1. Sokszög és poliéder

40.2. 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.*

40.3. definíció. *Egy sokszög* ***egyszeresen összefüggő,*** *ha egyetlen zárt töröttvonal határolja (40.11. ábra).*

40.4. definíció. *Egy sokszög* ***egyszerű,*** *ha egyszeresen összefüggő, és a határoló töröttvonal nem metszi saját magát (40.11(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.

40.5. 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.

40.3.2. 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.

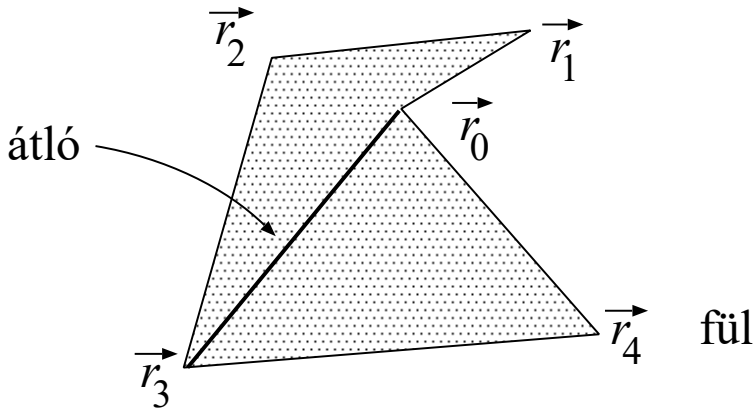
40.3.3. Egyszerű sokszögek háromszögekre bontása

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:

40.6. definíció. *Egy sokszög **átlója** egy, a sokszög két csúcsát összekötő olyan szakasz, amely teljes egészében a sokszög belsejében van (40.12. 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



40.12. ábra. A sokszög átlója és füle.

lehet például a jelölt középpontja. Egy pontról a 40.4.1. pontban tárgyalt eljárással dönthető el, hogy egy sokszög belsejében van-e.

40.7. definíció. *A sokszög egy csúcsa* ***fül,*** *ha az adott csúcsot megelőző és követő csúcsokat összekötő szakasz a sokszög átlója (40.12. ábra).*

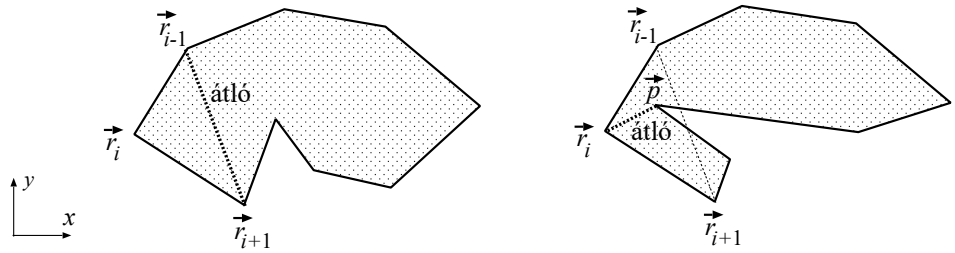
Nyilván csak azok a csúcsok lehetnek fülek, amelyekben a belső szög 180 foknál kisebb. Az ilyen csúcsokat ***konvex csúcsoknak*** nevezzük, a nem konvexeket pedig ***konkáv csúcsoknak.***

Az egyszerű sokszögekre kimondhatjuk a következő alapvető tételeket:

40.8. 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 $\vec{r}_{i+1}$ (40.13. 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 (40.13. 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 (40.13. 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. ■

40.9. tétel. *Egy egyszerű sokszög az átlóival mindig háromszögekre bontható. Ha a sokszög csúcsainak száma n, akkor a keletkező háromszögek száma $n-2$.*



40.13. ábra. Az átló létezésének bizonyítása egyszerű sokszögre.

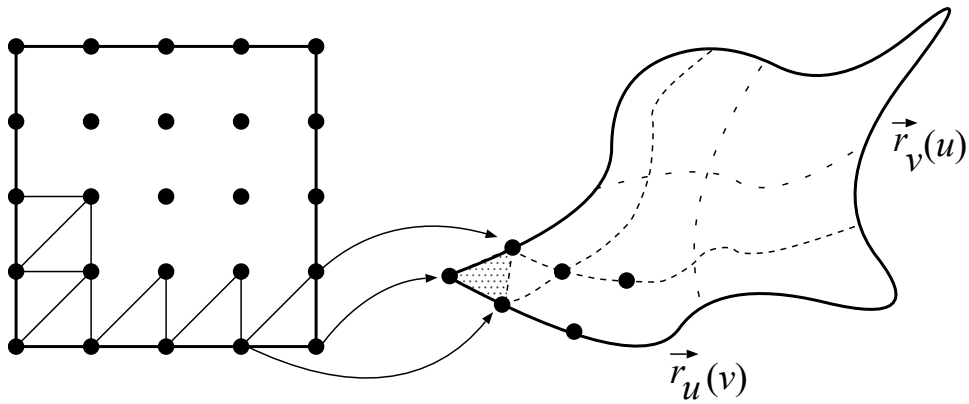
Bizonyítás. A tétel igazolásához teljes indukciót használunk. Az állítás $n = 3$ esetre, azaz egyetlen háromszögre nyilván igaz. Tegyük fel, hogy az állítás minden m $(m = 3, \ldots, n-1)$ csúcsú sokszögre igaz, és vegyünk egy n szöget. Az előző tétel szerint az n szögnek van átlója, tehát felbonthatjuk 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 fül-e. 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.



40.14. ábra. Paraméteres felületek tesszellációja.

40.10. tétel. *Egy egyszerű, legalább négy csúcsú sokszögnek mindig van legalább két, nem szomszédos, így egymástól függetlenül levágható, füle.*

Bizonyítás. A 40.9. 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. ■

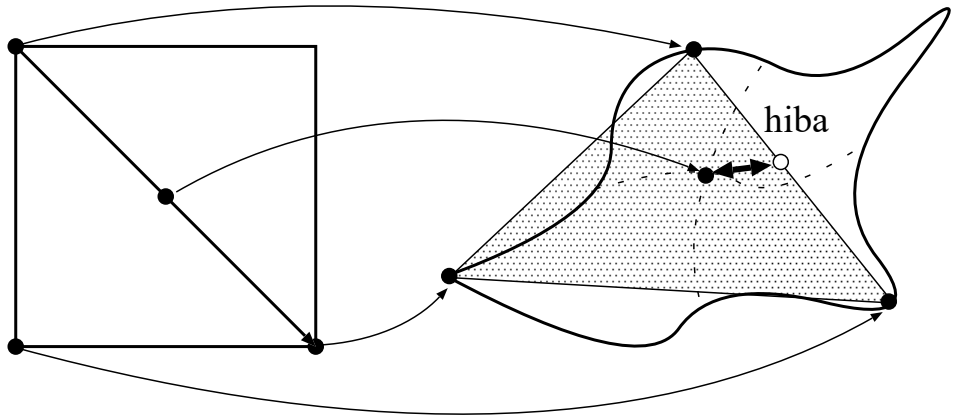
A két fül tétel miatt az algoritmusunk $O(n)$ lépésben mindig talál levágandó fület, amelynek eltávolításával a sokszög csúcsainak száma eggyel csökken, így az eljárás $O(n^2)$ lépésben befejeződik.

40.3.4. Paraméteres felületek tesszellációja

A paraméteres felületek a paramétertartomány egy $[u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}]$ téglalapját képezik le a felület pontjaira.

A tesszelláció elvégzéséhez a paramétertéglalapot háromszögesítjük. A paraméter háromszögek csúcsaira alkalmazva a paraméteres egyenletet, ép-

[3] A levél olyan csúcs, amelyhez pontosan egy él kapcsolódik.



40.15. ábra. A tesszellációs hiba becslése.

pen 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

$$[u_i, v_j] = \left[u_{\min} + (u_{\max} - u_{\min})\frac{i}{N}, v_{\min} + (v_{\max} - v_{\min})\frac{j}{M} \right]$$

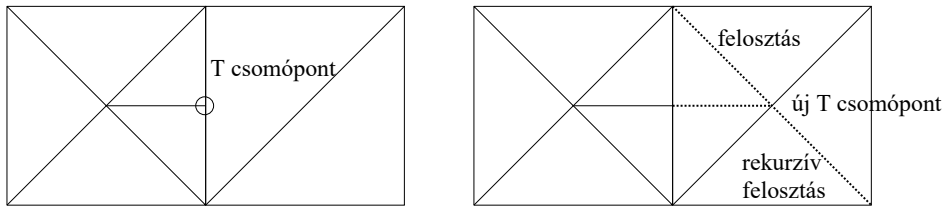
párokból kapott pontok közül az $\vec{r}(u_i, v_j)$, $\vec{r}(u_{i+1}, v_j)$, $\vec{r}(u_i, v_{j+1})$ ponthármasokra, illetve az $\vec{r}(u_{i+1}, v_j)$, $\vec{r}(u_{i+1}, v_{j+1})$, $\vec{r}(u_i, v_{j+1})$ ponthármasokra háromszögeket illeszt.

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 háromszög élfelező pontjaival, azaz képezzük a következő távolságot (40.15. 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.

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,



40.16. ábra. T csomópontok és kiküszöbölésük erőszakos felosztással.

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.

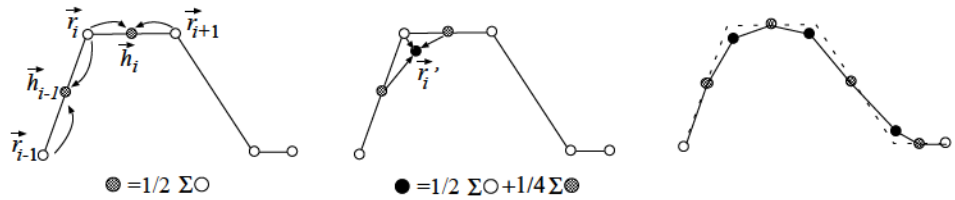
Az adaptív felbontásnál előfordulhat, hogy egy közös élre illeszkedő két háromszög közül az egyiket az élfelező ponton átmenő súlyvonallal felbontjuk, de a másikat nem, így a felbontás után az egyik oldalon lévő háromszög nem illeszkedik a másik oldalon lévő két másikhoz, azaz a felületünk kilyukad. Az ilyen problémás élfelező pontokat ***T csomópontnak*** nevezzük (40.16. ábra).

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.

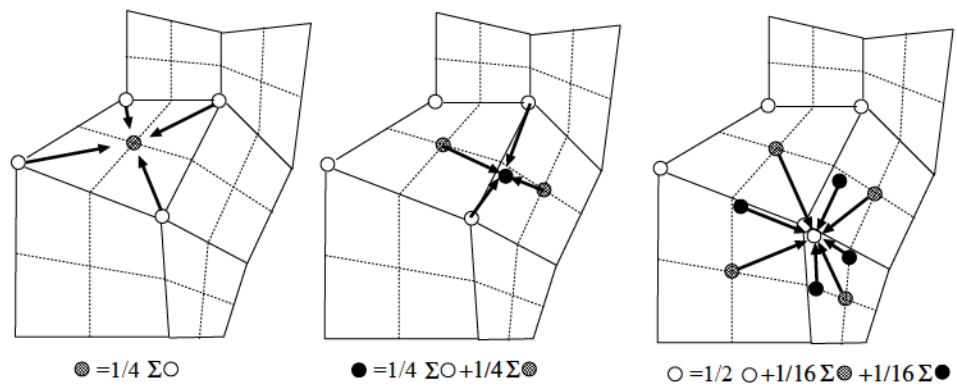
40.3.5. Töröttvonal és felület simítás, felosztott görbék és felületek

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.

Tekintsük először az $\vec{r}_0, \ldots, \vec{r}_m$ töröttvonalat. Egy látszólag simább töröttvonalhoz jutunk a következő, a vezérlőpontokat megduplázó eljárással (40.17. 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



40.17. ábra. Felosztott görbe létrehozása: minden lépésben felezőpontokat veszünk fel, majd az eredeti csúcspontokat a szomszédos felezőpontok és a csúcspont súlyozott átlagára helyezzük át.



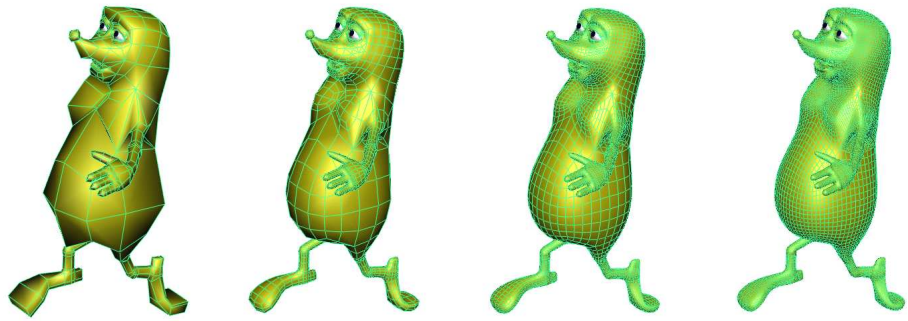
40.18. ábra. A Catmull-Clark-felosztás egy lépése. Először a lappontokat számítjuk, majd az élfelezők környezetében veszünk fel átlagolással egy pontot, végül pedig az eredeti csúcspontokat módosítjuk a környezet átlagának megfelelően.

lévő felezőpontok közé kerüljenek, az alábbi súlyozással:

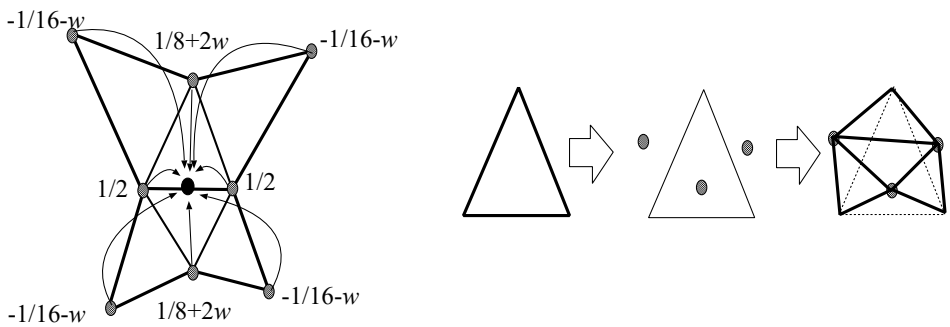
$$\vec{r}_i{}' = \frac{1}{2}\vec{r}_i + \frac{1}{4}\vec{h}_{i-1} + \frac{1}{4}\vec{h}_i = \frac{3}{4}\vec{r}_i + \frac{1}{8}\vec{r}_{i-1} + \frac{1}{8}\vec{r}_{i+1} .$$

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.

Az eljárás közvetlenül kiterjeszthető háromdimenziós hálókra, amelynek eredménye a ***Catmull-Clark felosztott felület.*** Induljunk ki egy háromdimenziós négyszöghálóból (40.18. 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



40.19. ábra. Az eredeti háló, valamint egyszer, kétszer és háromszor felosztott változatai.



40.20. ábra. Az új élpont meghatározása és a háromszög pillangó felosztása.

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 (40.19. 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.

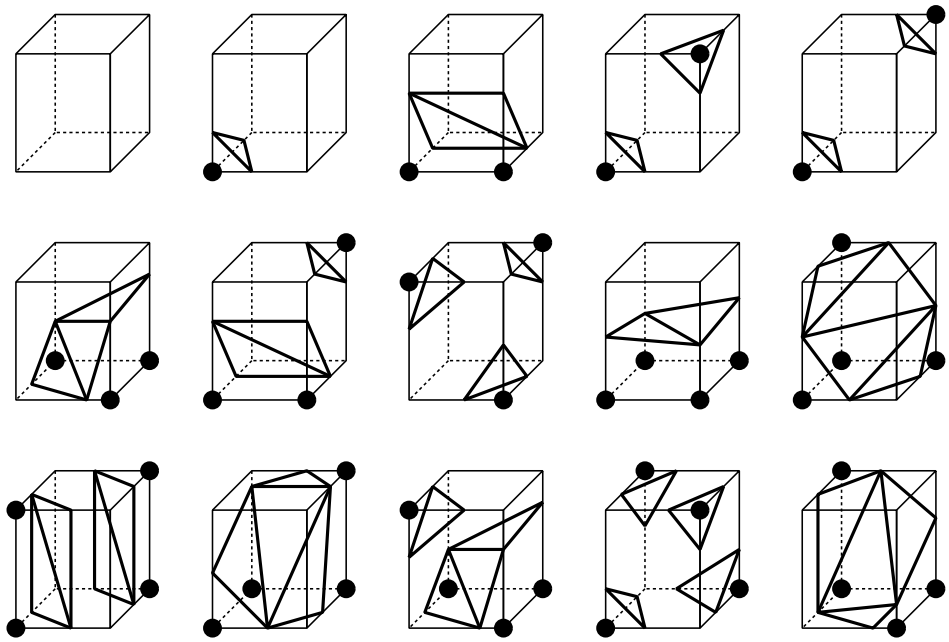
A Catmull-Clark-felosztással kapott felület általában nem megy át az eredeti háló csúcspontjain. Ezt a hátrányt küszöböli ki a háromszöghálókon működő ***pillangó felosztás.*** A pillangó felosztás a háromszögek élfelező pontjainak közelébe egy-egy új élpontot helyez, majd az eredeti háromszögeket négy új háromszöggel váltja fel. Az új háromszögek csúcsai egyrészt az eredeti háromszög csúcsai, másrészt a módosított élfelező pontjai (40.20. 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.

40.3.6. 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 (40.4. 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



40.21. 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.

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:

$$\vec{r}_i = \vec{r}_1 \cdot \frac{f(\vec{r}_2)}{f(\vec{r}_2) - f(\vec{r}_1)} + \vec{r}_2 \cdot \frac{f(\vec{r}_1)}{f(\vec{r}_2) - f(\vec{r}_1)} \ ,$$

$$\vec{n}_i = \mathrm{grad} f(\vec{r}_1) \cdot \frac{f(\vec{r}_2)}{f(\vec{r}_2) - f(\vec{r}_1)} + \mathrm{grad} f(\vec{r}_2) \cdot \frac{f(\vec{r}_1)}{f(\vec{r}_2) - f(\vec{r}_1)} \ .$$

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 (40.21. 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

40.3-1. Igazoljuk a két fül tételt teljes indukcióval.

40.3-2. Készítsünk adaptív görbetesszellációs algoritmust.

40.3-3. Bizonyítsuk be, hogy a Catmull-Clark felosztási módszer a B-spline görbéhez, illetve felülethez konvergál.

40.3-4. 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.

40.3-5. Készítsünk olyan masírozó kocka algoritmust, amely nem igényli az implicit függvény gradiensét a mintavételi pontokban, hanem azt is az implicit függvény mintáival becsli.

40.4. 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 (40.6. pont) építhetjük, így ebben a fejezetben csak a pont tartalmazással, a poliéderek közötti diszkrét idejű ütközésfelismeréssel, és végül néhány egyszerű alakzat vágásával foglalkozunk.

40.4.1. 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 (40.1) egyenlet alapján az

$$(\vec{r} - \vec{r}_0) \cdot \vec{n} \geq 0, \qquad n_x \cdot x + n_y \cdot y + n_z \cdot z + d \geq 0 \tag{40.9}$$

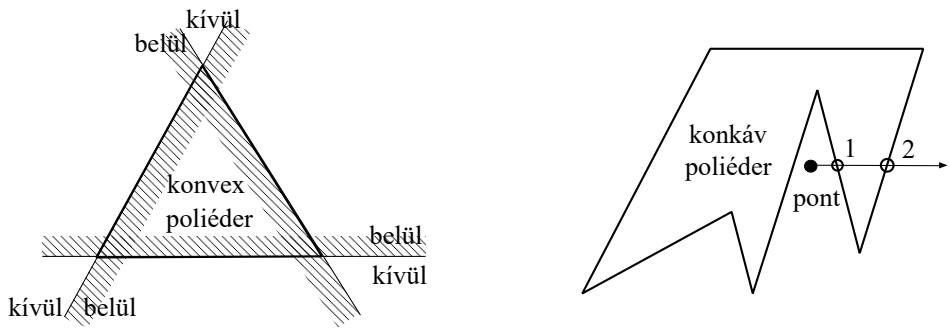
egyenlőtlenséggel adhatjuk meg, ahol a határoló sík normálvektora „befelé” mutat.

Konvex poliéder

Bármely konvex poliéder előállítható a lapjaira illeszkedő síkok által határolt félterek metszeteként (40.22. 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.

Konkáv poliéder

A 40.22. 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 40.6. szakaszban a sugárkövetéshez kidolgozott eljárások használhatók). Ha páratlan számú metszéspontot számolunk össze, akkor a poliéder belsejében, egyébként pedig azon kívül van



40.22. 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.

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 (40.23. 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:

$$\begin{aligned}((\vec{b}-\vec{a})\times(\vec{p}-\vec{a}))\cdot\vec{n} &\geq 0\,,\\ ((\vec{c}-\vec{b})\times(\vec{p}-\vec{b}))\cdot\vec{n} &\geq 0\,,\\ ((\vec{a}-\vec{c})\times(\vec{p}-\vec{c}))\cdot\vec{n} &\geq 0\,.\end{aligned} \tag{40.10}$$

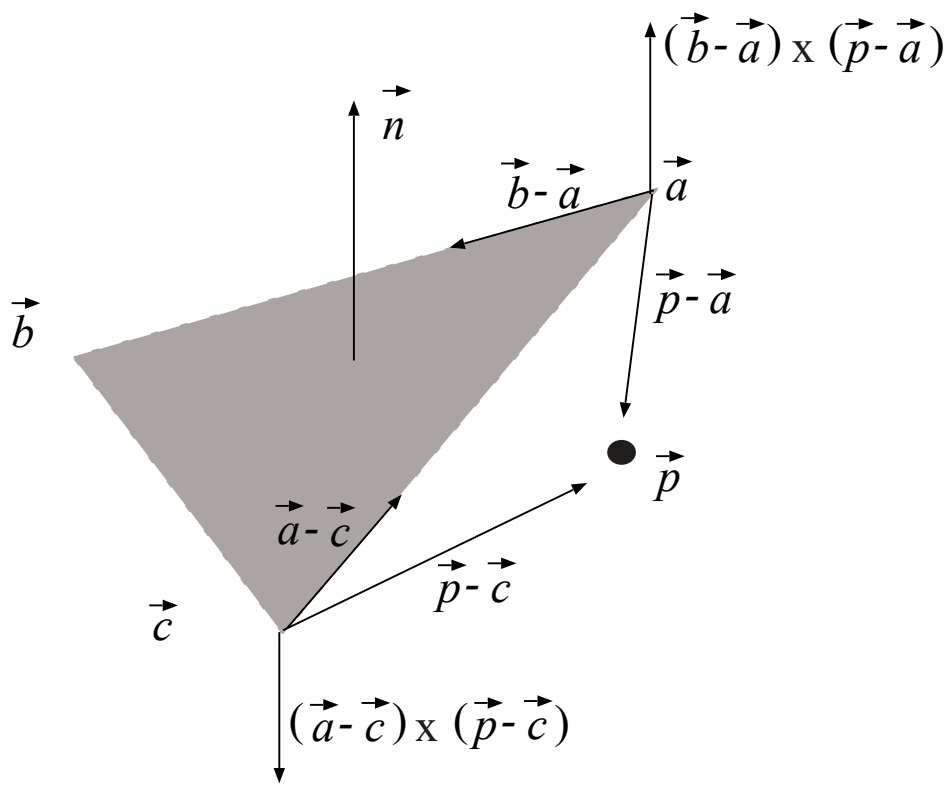
A vizsgálat robusztus, azaz akkor is helyes eredményt ad, ha a numerikus pontatlanságok miatt a $\vec{p}$ pont nincs pontosan a háromszög síkján, csak a síkra merőleges háromszög alapú hasábban.

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 oldalára 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.

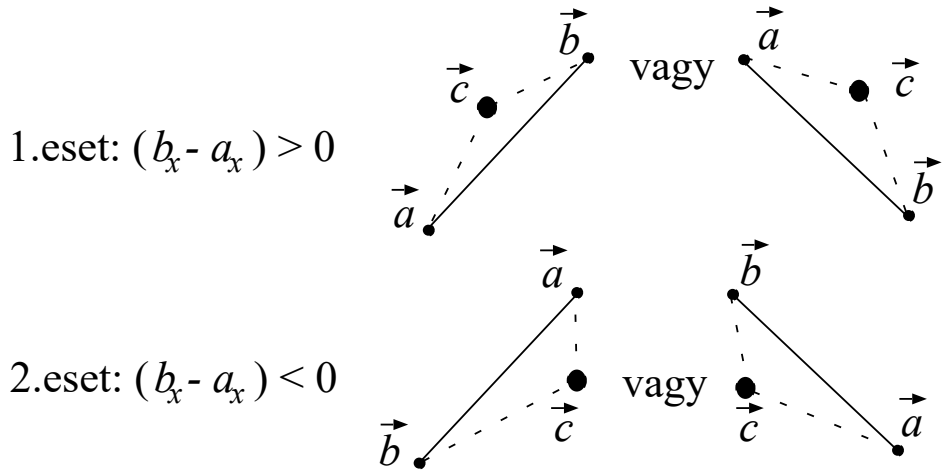
Először átalakítjuk a csúcsok sorrendjét úgy, hogy $\vec{a}$-ból $\vec{b}$-be haladva a $\vec{c}$ pont mindig a bal oldalon helyezkedjen el. Ehhez először vizsgáljuk meg az $\vec{ab}$ egyenes egyenletét:

$$\frac{b_y-a_y}{b_x-a_x}\cdot(x-b_x)+b_y=y\,.$$

A 40.24. ábra alapján a $\vec{c}$ pont akkor van az egyenes bal oldalán, ha



40.23. 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.



40.24. 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.

$x = c_x$-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:

$$(b_y - a_y) \cdot (c_x - b_x) < (c_y - b_y) \cdot (b_x - a_x) \ .$$

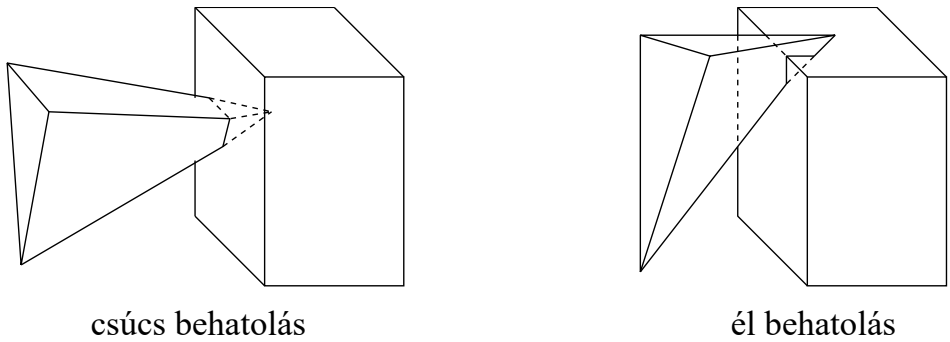
A második esetben a meredekség nevezője negatív. A $\vec{c}$ pont akkor van az egyenes bal oldalán, ha $x = c_x$-nél c_y az egyenes alatt van:

$$\frac{b_y - a_y}{b_x - a_x} \cdot (c_x - b_x) + b_y > c_y \ .$$

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 $\vec{bc}$



40.25. 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.

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:

$$\begin{aligned}(b_y - a_y) \cdot (p_x - b_x) &\leq (p_y - b_y) \cdot (b_x - a_x)\ , \\ (c_y - b_y) \cdot (p_x - c_x) &\leq (p_y - c_y) \cdot (c_x - b_x)\ , \\ (a_y - c_y) \cdot (p_x - a_x) &\leq (p_y - a_y) \cdot (a_x - c_x)\ .\end{aligned} \tag{40.11}$$

40.4.2. 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 (40.25. 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.

A csúccsal történő ütközésen kívül előfordulhat, hogy két poliéder élei a másikba hatolnak anélkül, hogy csúcsaik a másik belsejébe kerülnének (40.25. ábra jobb oldala). Ennek felismeréséhez az egyik poliéder összes élét össze kell vetni a másik poliéder összes lapjával. Egy él és lap tekintetében a (40.9) 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 (40.6.2. 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.

40.4.3. 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 (40.1) 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:

$$(\vec{r}_1 \cdot (1-t_i) + \vec{r}_2 \cdot t_i - \vec{r}_0) \cdot \vec{n} = 0, \quad \Longrightarrow \quad t_i = \frac{(\vec{r}_0 - \vec{r}_1) \cdot \vec{n}}{(\vec{r}_2 - \vec{r}_1) \cdot \vec{n}} .$$

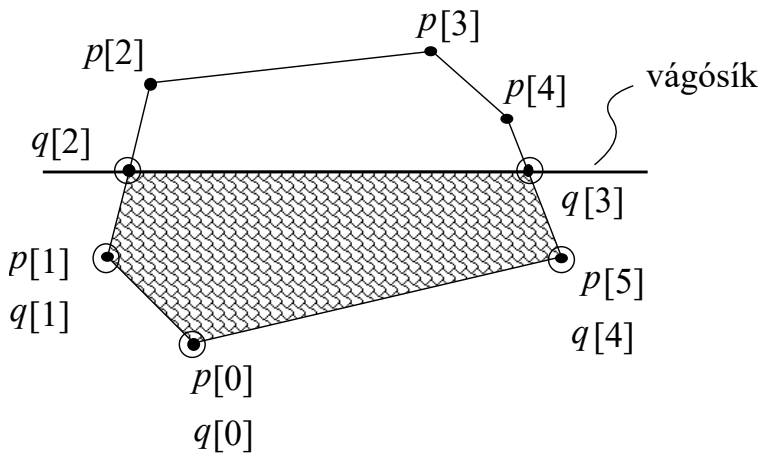
A t_i paramétert a szakasz egyenletébe visszahelyettesítve a metszéspont koordinátáit is előállíthatjuk.

Sokszögek vágása féltérre

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 (40.26. ábra).

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:***



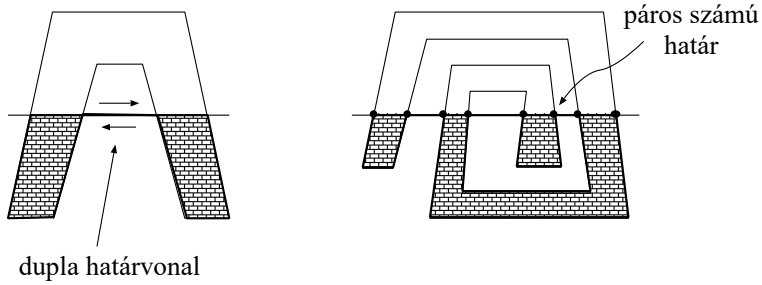
40.26. ábra. A $\vec{p}[0], \ldots, \vec{p}[5]$ sokszög vágása, amelynek eredménye a $\vec{q}[0], \ldots, \vec{q}[4]$ sokszög. Az eredménysokszög csúcsai az eredeti sokszög belső csúcsai és az éleknek a vágósíkkal képzett metszéspontjai.

SUTHERLAND-HODGEMAN-POLIGONVÁGÁS($\mathbf{p}$)

```
1  m = 0
2  for i = 0 to n − 1
3      if p[i] belső pont
4          q[m] = p[i]              // Az i-edik csúcs része a vágott poligonnak.
5          m = m + 1
6          if p[i + 1] külső pont
7              q[m] = METSZÉS-VÁGÓSÍKKAL(p[i], p[i + 1])
8              m = m + 1
9              else if p[i + 1] belső pont
10                 q[m] = METSZÉS-VÁGÓSÍKKAL(p[i], p[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 (40.27. 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.

A páros számú extra él azonban nem okoz problémát a továbbiakban, ha a sokszög belső pontjait a következő elv alapján határozzuk meg: a kérdéses



40.27. ábra. Konkáv sokszögek vágásakor a széteső részeket páros számú élek tartják össze.

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 külön-külön kell végrehajtani.

Szakaszok és poligonok vágása konvex poliéderre

Miként korábban megállapítottuk, egy konvex poliéder előállítható a lapjaira illeszkedő síkok által határolt félterek metszeteként (40.22. 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.

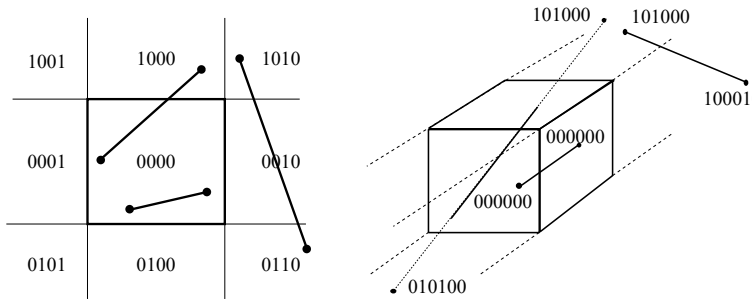
Szakaszok vágása AABB-re

A képszintézis algoritmusokban különösen fontos szerepet kap egy speciális típusú konvex poliéder, az AABB.

40.11. 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



40.28. 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.

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 feleslegessé 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 (40.28. ábra). Egy pont $C[0], \ldots, C[5]$ kódbitjei:

$$C[0] = \begin{cases} 1, & x \le x_{min}, \\ 0 & \text{egyébként.} \end{cases} \quad C[1] = \begin{cases} 1, & x \ge x_{max}, \\ 0 & \text{egyébként.} \end{cases} \quad C[2] = \begin{cases} 1, & y \le y_{min}\ , \\ 0 & \text{egyébként}\ . \end{cases}$$

$$C[3] = \begin{cases} 1, & y \ge y_{max}, \\ 0 & \text{egyébként.} \end{cases} \quad C[4] = \begin{cases} 1, & z \le z_{min}, \\ 0 & \text{egyébként.} \end{cases} \quad C[5] = \begin{cases} 1, & z \ge z_{max}\ , \\ 0 & \text{egyébként}\ . \end{cases}$$

Nyilvánvalóan a 000000 kóddal rendelkező pontok a vágási tartományban, a többiek pedig azon kívül találhatók (40.28. 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 leg-

egyszerűbben 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  C1 = az r1 végpont kódja // Kódbitek az egyenlőtlenségek ellenőrzésével.
2  C2 = az r2 végpont kódja
3  while IGAZ
4        if C1 == 0 és C2 == 0
5           return IGAZ              // Triviális elfogadás: van belső szakasz.
6           if C1 & C2 ≠ 0
7              return HAMIS          // Triviális eldobás: nincs belső szakasz.
8           f = a legelső bit indexe, amelyen a C1 és a C2 különbözik
9           ri = az (r1, r2) szakasz és az f indexű sík metszéspontja
10          Ci = az ri metszéspont kódja
11          if C1[f] == 1
12          r1 = ri
13          C1 = Ci                  // r1 van az f-edik sík rossz oldalán.
14          else r2 = ri
15               C2 = Ci             // r2 van az f-edik sík rossz oldalán.
```

Gyakorlatok

40.4-1. Tegyünk javaslatokat a poliéder-poliéder ütközésfelismerés négyzetes időbonyolultságának csökkentésére.

40.4-2. Készítsünk CSG-fa adatszerkezethez pont tartalmazást vizsgáló al-

goritmust.

40.4-3. Készítsünk algoritmust, amely egy sokszöget egy másik, akár konkáv sokszögre vág.

40.4-4. Készítsünk algoritmust, amely egy poliéder befoglaló gömbjét, illetve befoglaló AABB-jét kiszámítja.

40.4-5. Adjunk algoritmust a síkban két háromszög ütközésének felismeréséhez.

40.4-6. Általánosítsuk a Cohen-Sutherland szakaszvágó módszert konvex poliéder vágási tartományra.

40.4-7. Dolgozzuk ki a szakaszt gömbre vágó módszert.

40.5. 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

$$\{\vec{r}\,' : f(\mathcal{T}^{-1}(\vec{r}\,')) \geq 0\}\ , \tag{40.12}$$

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

$$\vec{r}\,'(u, v) = \mathcal{T}(\vec{r}(u, v))\ . \tag{40.13}$$

Hasonlóan egy $\vec{r} = \vec{r}(t)$ egyenletű görbe transzformáltjának egyenlete:

$$\vec{r}\,'(t) = \mathcal{T}(\vec{r}(t))\ . \tag{40.14}$$

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.

40.5.1. 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 pon-

tjait 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 (40.29. 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.

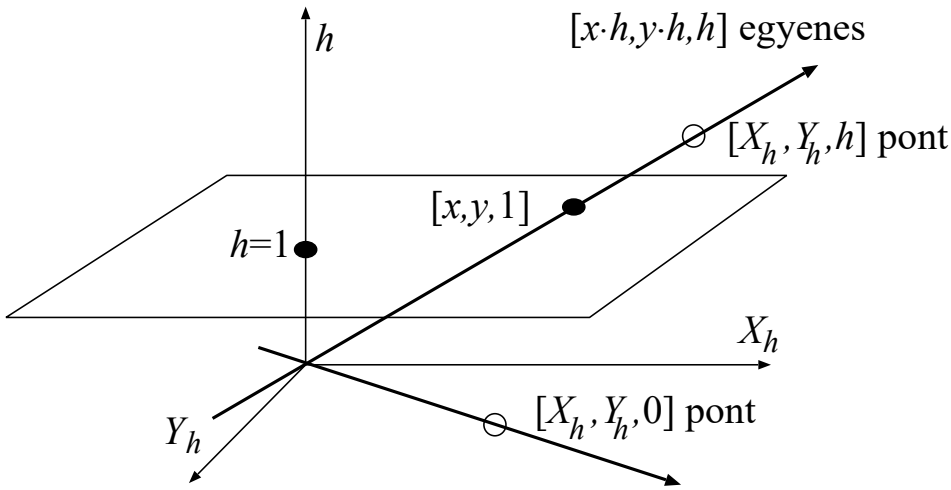
Az origón átmenő egyenes pontjait a $[t \cdot X_h, t \cdot Y_h, t \cdot h]$ paraméteres formában adhatjuk meg a t paraméter függvényében. Amennyiben a P pont a sík közönséges pontja, az egyenes nem párhuzamos a $h = 1$ síkkal (azaz $h \neq 0$). Az egyenes az $[X_h/h, Y_h/h, 1]$ pontban metszi a síkot, 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.

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 térnek 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:

$$X_h = x, \qquad Y_h = y, \qquad h = 1\,. \tag{40.15}$$



40.29. 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.

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:***

$$\begin{aligned} X_h(t) &= X_h^1 \cdot (1-t) + X_h^2 \cdot t \,, \\ Y_h(t) &= Y_h^1 \cdot (1-t) + Y_h^2 \cdot t \,, \\ h(t) &= h^1 \cdot (1-t) + h^2 \cdot t \,. \end{aligned} \tag{40.16}$$

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 ($h \neq 0$):

$$x = \frac{X_h}{h}, \qquad y = \frac{Y_h}{h}, \qquad z = \frac{Z_h}{h} \ . \tag{40.17}$$

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:

$$\begin{aligned} X_h(t) &= X_h^1 \cdot (1-t) + X_h^2 \cdot t \ , \\ Y_h(t) &= Y_h^1 \cdot (1-t) + Y_h^2 \cdot t \ , \\ Z_h(t) &= Z_h^1 \cdot (1-t) + Z_h^2 \cdot t \ , \\ h(t) &= h^1 \cdot (1-t) + h^2 \cdot t \ . \end{aligned} \tag{40.18}$$

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 (40.1) 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 (40.17) 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:

$$n_x \cdot X_h + n_y \cdot Y_h + n_z \cdot Z_h + d \cdot h = 0 \ , \tag{40.19}$$

vagy mátrixos alakban:

$$[X_h, Y_h, Z_h, h] \cdot \begin{bmatrix} n_x \\ n_y \\ n_z \\ d \end{bmatrix} = 0 \ . \tag{40.20}$$

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.

40.5.2. 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:

$$[X'_h, Y'_h, Z'_h, h'] = [X_h, Y_h, Z_h, h] \cdot \mathbf{T} \ . \tag{40.21}$$

40.12. 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.

40.13. 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:

$$\begin{aligned}[X'_h(t), Y'_h(t), Z'_h(t), h'(t)] &= [X_h(t), Y_h(t), Z_h(t), h(t)]\cdot\mathbf{T}\\ &= [X_h^1, Y_h^1, Z_h^1, h^1]\cdot\mathbf{T}\cdot(1-t) + [X_h^2, Y_h^2, Z_h^2, h^2]\cdot\mathbf{T}\cdot t\\ &= [X_h^{1'}, Y_h^{1'}, Z_h^{1'}, h^{1'}]\cdot(1-t) + [X_h^{2'}, Y_h^{2'}, Z_h^{2'}, h^{2'}]\cdot t\ ,\end{aligned}$$

ahol $[X_h^{1'}, Y_h^{1'}, Z_h^{1'}, h^{1'}]$ az $[X_h^1, Y_h^1, Z_h^1, h^1]$ pont, $[X_h^{2'}, Y_h^{2'}, Z_h^{2'}, h^{2'}]$ 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.

40.14. 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:

$$[x', y', z'] = [x, y, z] \cdot \mathbf{A} + [p_x, p_y, p_z] \,, \tag{40.22}$$

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 (40.22) egyenlet egy 4×4-es mátrixművelettel is leírható, miután a Descartes-koordinátákról áttérünk homogén koordinátákra a negyedik homogén koordinátát 1-nek választva:

$$[x', y', z', 1] = [x, y, z, 1] \cdot \begin{bmatrix} A_{11} & A_{12} & A_{13} & 0 \\ A_{21} & A_{22} & A_{23} & 0 \\ A_{31} & A_{32} & A_{33} & 0 \\ p_x & p_y & p_z & 1 \end{bmatrix} = [x, y, z, 1] \cdot \mathbf{T}\,. \quad (40.23)$$

Az affin transzformációk körén belül további speciális eset a távolságtartó transzformáció, amelyet ***egybevágósági transzformációnak*** nevezünk. Az egybevágósági transzformációk egyben szögtartóak is.

40.15. 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

40.5-1. 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.

40.5-2. A homogén koordináták, mint algebrai alap felhasználásával igazoljuk

a projektív geometria egy axiómáját, miszerint két különböző egyenes mindig egy pontban metszi egymást.

40.5-3. Bizonyítsuk be a súlypont analógia alapján, hogy a homogén lineáris transzformációk egy háromdimenziós szakaszt szakaszba visznek át.

40.5-4. Hogyan változtatja meg egy affin transzformáció egy test térfogatát?

40.5-5. Írjuk fel a $\vec{p}$ vektorral eltoló transzformáció mátrixát.

40.5-6. Igazoljuk a Rodrigues-képletet.

indexRodrigues-keplet@Rodrigues-képlet

40.5-7. Egy $f(\vec{r}) \geq 0$ egyenlőtlenséggel leírt test $\vec{v}$ sebességgel egyenletesen mozog. Írjuk fel a test pontjait leíró egyenlőtlenséget a t időpillanatban.

40.5-8. Igazoljuk, hogy ha az $\mathbf{A}$ mátrix sorai egymásra merőleges egységvektorok, akkor az affin transzformáció egyben egybevágósági transzformáció is. Mutassuk meg, hogy ilyen mátrixokra $\mathbf{A}^{-1} = \mathbf{A}^T$.

40.5-9. Írjuk fel azon homogén lineáris transzformáció mátrixát, amely a teret $\vec{c}$ középponttal az $\vec{n}$ normálvektorú, $\vec{r}_0$ helyvektorú síkra vetíti.

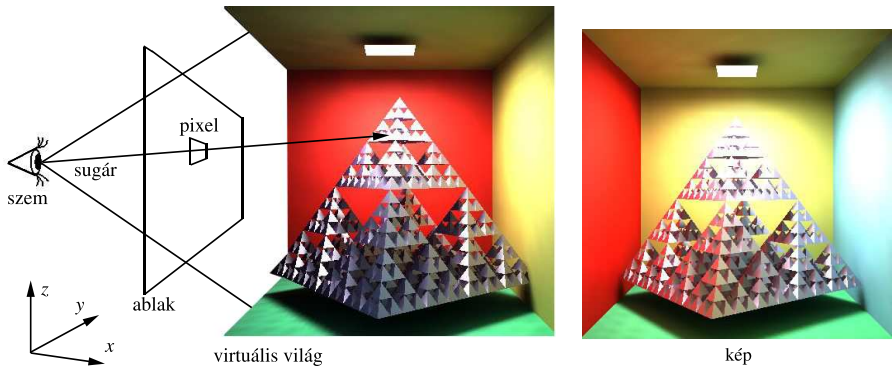
40.5-10. Mutassuk meg, hogy 5 tárgypont-képpont pár egyértelműen azonosít egy homogén lineáris transzformációt, ha az 5 pont közül semelyik négy sincs egy síkon.

40.6. 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 (40.30. 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



40.30. ábra. Képszintézis sugárkövetéssel.

azzal a felülettel ütközik, amelyet a pont pályáját leíró sugár először metsz.

A sugarat a következő egyenlettel adjuk meg:

$$\vec{ra}y(t) = \vec{s} + \vec{v} \cdot t, \quad (t > 0) \ , \tag{40.24}$$

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:

[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.

SUGÁR-ELSŐ-METSZÉSPONT($\vec{s}, \vec{v}$)

```
1  t = t_max                          // Inicializálás a tér legnagyobb méretére.
2  for minden o objektumra
3      t_o = SUGÁR-FELÜLET-METSZÉSPONT(s, v)
                                      // Negatív, ha nincs metszéspont.
4      if 0 ≤ t_o < t                 // Az új metszéspont közelebbi-e?
5          t = t_o                    // A legközelebbi metszés sugárparamétere.
6          o_metszett ← o             // A legközelebb metszett objektum.
7  if t < t_max                       // Volt egyáltalán metszéspont?
8      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.

40.6.1. 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 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:

$$\vec{n} \cdot (\vec{r} - \vec{a}) = 0 \ . \tag{40.25}$$

A sugár és a sík közös pontját megkaphatjuk, ha a sugár egyenletét ((40.24) egyenlet) behelyettesítjük a sík egyenletébe ((40.25) 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 40.4.1. 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 (40.4.3 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.

40.6.2. 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

[5] t_{max} = a kamerával együtt értendő színtér átmérője.

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 objektummal 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.

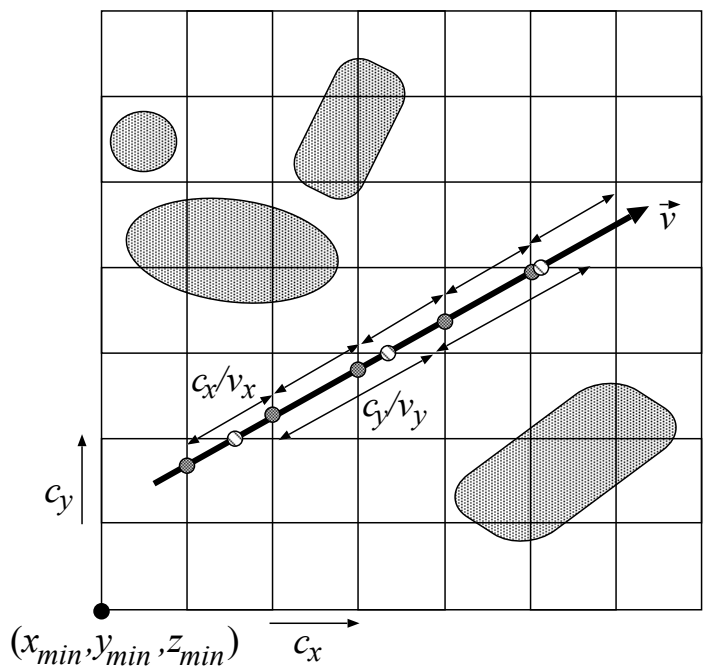
A naiv sugárkövetés az objektumok számával lineárisan növekvő időt igényel. A befoglaló keretek alkalmazása után az algoritmus továbbra is lineáris. A lineáris tag együtthatója viszont várhatóan kisebb.

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 (40.31. ábra), amit a teljes objektumteret befoglaló AABB felosztásával kapunk.

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 (40.4.3. 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



40.31. 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.

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      c cella objektumlistája = üres
4      for minden o objektumra // A cellába lógó objektumok regisztrálása.
5          if a c cella és az o objektum AABB-je egymásba lóg
6              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. Sugárfelület 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 (40.31. 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     t_x = (x_min + (X + 1) · c_x − s_x)/v_x
3     else if v_x < 0
4             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     t_y = (y_min + (Y + 1) · c_y − s_y)/v_y
8     else if v_y < 0
9             t_y = (y_min + Y · c_y − s_y)/v_y
10            else t_y = t_max
11 if v_z > 0
12    t_z = (z_min + (Z + 1) · c_z − s_z)/v_z
13    else if v_z < 0
14            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     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           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           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(s)
2  (t_x, t_y, t_z) =
     // SZABÁLYOS-RÁCS-SUGÁRPARAMÉTER-KEZDETI-ÉRTÉK(s, v, X, Y, Z)
3  while X, Y, Z a rács belsejében van
4        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           t_o =SUGÁR-FELÜLET-METSZÉS(s, v, o)   // Negatív, ha nincs.
8           if 0 ≤ t_o < t                    // Az új metszéspont közelebbi-e?
9              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                x = s + v · t // A metszéspont helye a sugár egyenletéből.
13                return t, x, 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:

$$T = T_o + N_I \cdot T_I + N_S \cdot T_S, \tag{40.26}$$

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.

40.16. 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*

$$\Pr\{N(A) = k\} = \frac{(\rho V(A))^k}{k!} \cdot e^{-\rho V(A)} ,$$

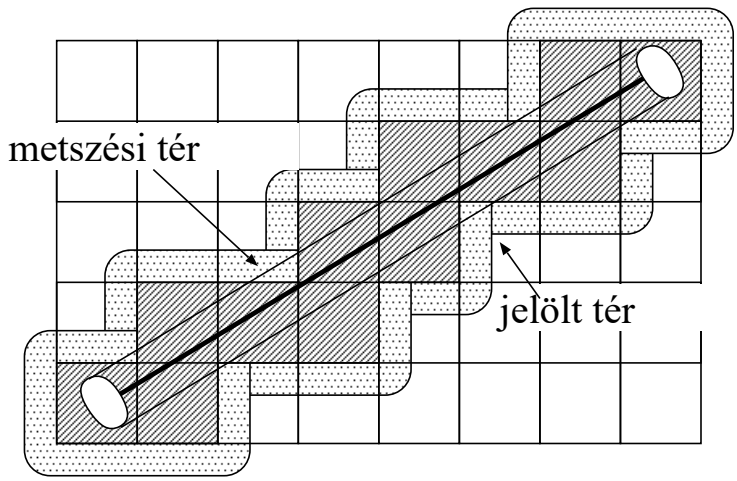
  és egy $V(A)$ térfogatban található mintapontok számának várható értéke $\rho V(A)$,
- *$A_1, A_2, \ldots, A_n$ diszjunkt halmazokra az $N(A_1), N(A_2), \ldots, N(A_n)$ valószínűségi változók függetlenek.*

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 40.32. 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



40.32. 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.

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.

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:

$$\mathrm{E}\left[N_I\right] = \frac{1}{s}\,. \tag{40.27}$$

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:

$$\frac{r^2\pi}{\sqrt{3}c^2+6cr+r^2\pi} \leq s \leq \frac{r^2\pi}{c^2+4cr+r^2\pi} .$$

A (40.27) egyenlet szerint, a metszési kísérletek várható száma ezen valószínűség reciproka:

$$\frac{1}{\pi}\left(\frac{c}{r}\right)^2+\frac{4}{\pi}\frac{c}{r}+1 \leq \mathrm{E}\left[N_I\right] \leq \frac{\sqrt{3}}{\pi}\left(\frac{c}{r}\right)^2+\frac{6}{\pi}\frac{c}{r}+1 . \tag{40.28}$$

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:

$$\mathrm{E}\left[N_S\right] = \int_0^\infty \mathrm{E}\left[N_S|t^*=t\right]\cdot p_{t^*}(t)\ dt ,$$

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:

$$\Pr\left\{t^*<t\right\} = 1-e^{-\rho r^2\pi t} .$$

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:

$$\mathrm{E}\left[N_S\right] \approx \int_0^\infty \left(\frac{t}{c}+1\right) \cdot \rho r^2 \pi \cdot e^{-\rho r^2 \pi t}\, dt = \frac{1}{c\rho r^2 \pi} + 1\,. \tag{40.29}$$

Például, ha a cellaméret az objektummérettel összevethető ($c = 2r$), és a cellában lévő gömbközéppontok várható száma $0,1$, akkor $\mathrm{E}\left[N_S\right] \approx 14$. 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 (40.28) és (40.29) egyenletek alapján a c cellamérettel 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:

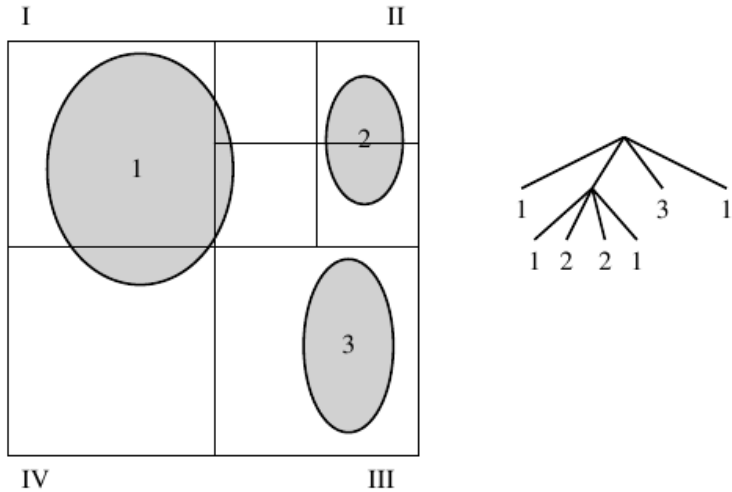
- Kezdetben foglaljuk az objektumainkat egy-egy koordinátatengelyekkel párhuzamos oldalú dobozba, azaz AABB-be, majd határozzuk meg az objektumonkénti AABB-k közös, befoglaló AABB-jét is. Ez lesz az oktális fa gyökere, és egyben a rekurzió kiindulópontja, azaz az első feldolgozandó cella.
- Ha az aktuális 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*** (40.33. 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          t_o =SUGÁR-FELÜLET-METSZÉS(s⃗, v⃗)      // Negatív, ha nincs.
8          if 0 ≤ t_o < t                  // Az új metszéspont közelebbi-e?
9          t = t_o              // A legközelebbi metszés sugárparamétere.
10         o_metszett = o                 // A legközelebb metszett objektum.
```



40.33. 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.

```
11      if t < t_ki                          // Volt metszéspont a cellában ?
12          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 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

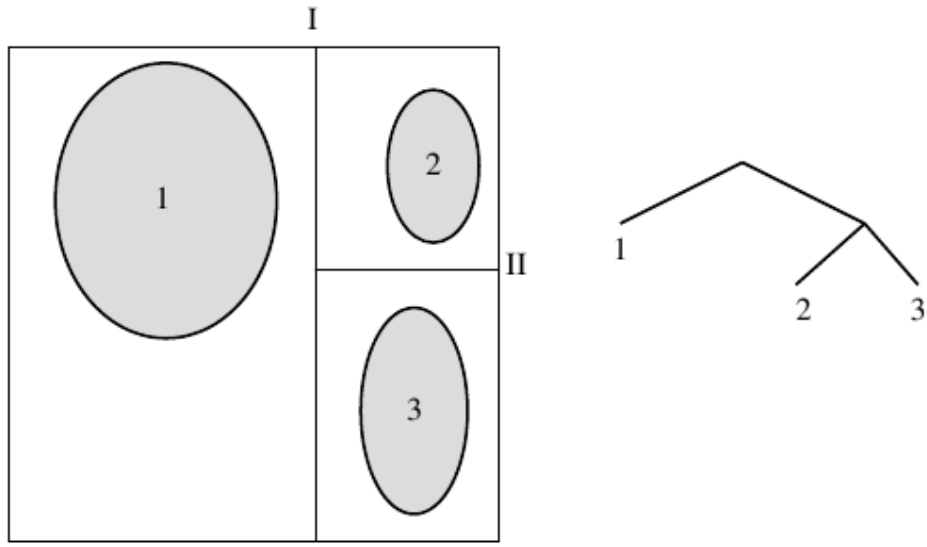
SUGÁR-ELSŐ-METSZÉSPONT-OKTÁLIS-FÁVAL algoritmusban ε-t). A kicsivel továbblendített sugárparamétert visszahelyettesítve a sugáregyenletbe, egy, a következő cellában lévő pontot (az algoritmusban a $\vec{q}$ pontot) kapunk. A cellát a pont alapján ismét a CELLA-KERESÉS-OKTÁLIS-FÁBAN algoritmussal azonosíthatjuk.

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 (40.28) 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.

A kd-fában a felezősíkot többféleképpen elhelyezhetjük:



40.34. ábra. A kd-fa. A „sok” objektumot tartalmazó cellát rekurzívan egy valamely koordinátatengelyre merőleges síkkal két cellára bontjuk.

- 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:

40.17. 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     return
3  cella.bal és cella.jobb befoglalódoboza = cella befoglalódoboza
4  if koordináta = x
5     cella.jobb.min.x = cella felezősíkja x irányban
6     cella.bal.max.x = cella felezősíkja x irányban
7     else if koordináta = y
8              cella.jobb.min.y = cella felezősíkja y irányban
9               cella.bal.max.y = cella felezősíkja y irányban
10    else if koordináta = z
11             cella.jobb.min.z = cella felezősíkja z irányban
12             cella.bal.max.z = cella felezősíkja z irányban
13 for a cella o objektumaira
14       if az o objektum a cella.bal befoglaló dobozában van
15          adjuk az o objektumot a cella.bal listájához
16          if az o objektum a cella.jobb befoglaló dobozában van
17             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

[6] Attól függően, hogy a sugár kezdőpontja az összes test közös befoglaló dobozán belül van-e vagy sem.

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 kilépési 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.

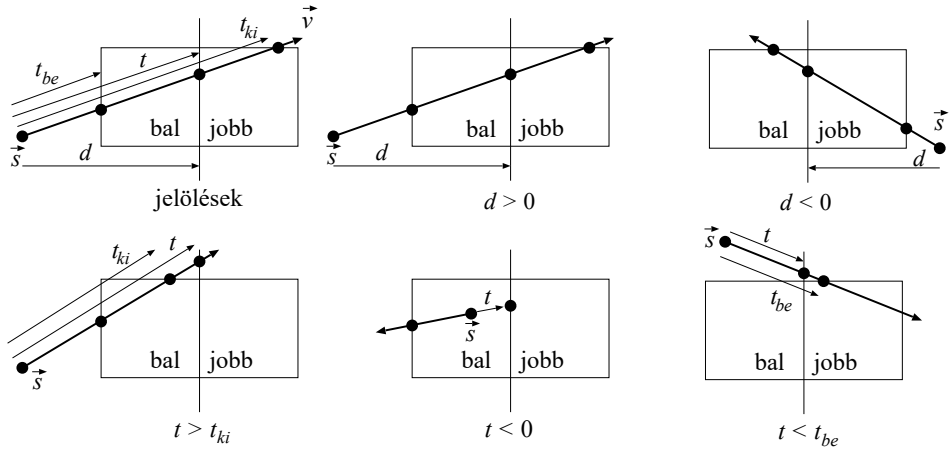
Ezt a hátrányt úgy küszöbölhetjük ki, hogy a meglátogatandó cellákat egy veremtárba tesszük, és mindig csak addig lépünk vissza, amíg szükséges. Így minden belső csúcsot és levelet csak egyszer látogatunk meg. Amikor a sugár egy olyan belső csúcshoz ér, amelynek két gyerekcsúcsa van, eldöntjük, hogy 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 40.35. ábrán követhetjük végig:

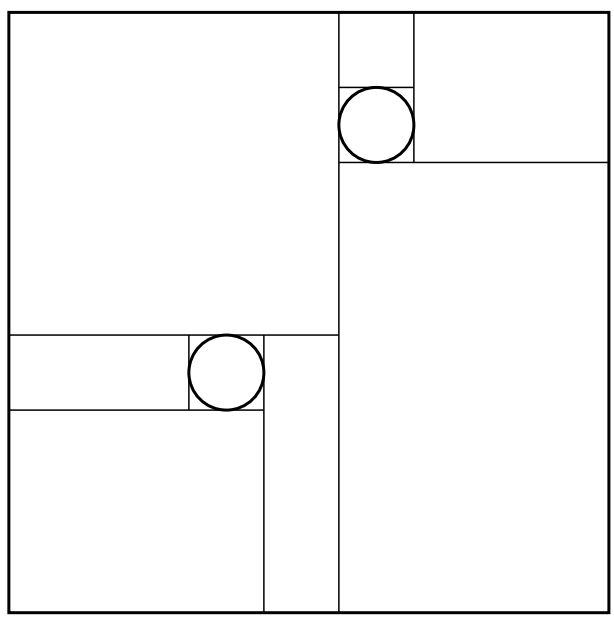
SUGÁR-ELSŐ-METSZÉSPONT-KD-FÁVAL($gyökér, \vec{s}, \vec{v}$)

```
1  (t_be, t_ki) = SUGÁR-AABB-METSZÉS(s, v, gyökér)
                                          // Metszés a tér-AABB-jével.
2  if nincs metszéspont
3     „nincs metszéspont”
4  VEREMBE(gyökér, t_be, t_ki)
5  keywhile a verem nem üres              // Amíg a fát be nem jártuk.
6           VEREMBŐL(cella, t_be, t_ki)
7           while cella nem levél
8                 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                   (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                        cella = közeli       // Csak a közeli cellát metszi.
16                        else if t < t_be
17                                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                t_o =SUGÁR-FELÜLET-METSZÉS(s, v) // Negatív, ha nincs.
24                if t_be ≤ t_o < t          // Az új metszéspont közelebbi-e?
25                   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                   x = s + v · t
                                  // A metszéspont helye a sugár egyenletéből.
29                   return t, x, o_metszett // A metszéspontot megtaláltuk.
30 return -1                                 // Nincs metszéspont.
```

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 (40.36. ábra).



40.35. 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.



40.36. ábra. Kd-fa alapú térpartícionálás az üres terek levágásával.

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 40.17. 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:

$$s = \frac{4r^2\pi}{6a^2} = \frac{\pi}{6} .$$

A metszési kísérletek várható száma tehát:

$$\mathrm{E}\left[N_I\right] = \frac{6}{\pi} \approx 1.91 .$$

A valószínűségi modell szerint a kd-fa igényli a legkevesebb sugár-felület metszési kísérletet.

Gyakorlatok

40.6-1. Bizonyítsuk be, hogy a valószínűségi modellben minden olyan sugárkövető algoritmus konstans időben fut, amely az objektumokat a sugár kezdőpontjától számított távolság sorrendjében dolgozza fel.

40.6-2. Készítsünk sugár-felosztott felület metszéspontszámító algoritmust.

40.6-3. Készítsünk sugár-B-spline felület metszéspontszámító algoritmust.

40.6-4. Készítsünk metszéspontszámító algoritmust CSG modellekhez – feltételezve, hogy a primitív testekre a metszéspontszámítás már működik.

40.6-5. Készítsünk metszéspontszámító algoritmust transzformált testekhez feltételezve, hogy az eredeti testre a metszéspontszámítás már működik (segítség: transzformáljuk a sugarat).

40.7. 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.

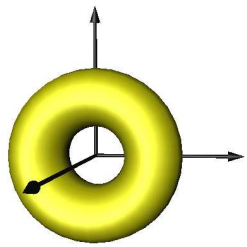
4. Feleslegesen nem számolnak, ezért a *vágás* során eltávolítják azon geometriai elemeket, amelyek a képen nem jelennének meg.

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 (**??**. szakasz).

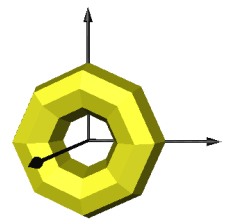
Az inkrementális képszintézis lépéseit a 40.37. ábrán láthatjuk. A virtuális világ objektumait azok egy referenciahelyzetében adjuk meg, a további műveletek miatt sokszögekkel közelítjük a felületeket, majd a modellezési transzformációval a virtuális világ koordinátarendszerébe helyezzük el őket. A virtuális 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, ame-

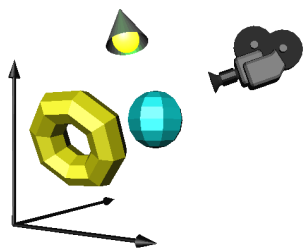
[7] Bár a Bézier és B-Spline görbék és felületek az affin transzformációkra, a NURBS pedig még a homogén lineáris transzformációkra is invariáns, de a vágás ezen görbék és felületek típusát is megváltoztatja.



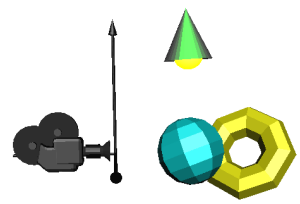
(a) Modellezés



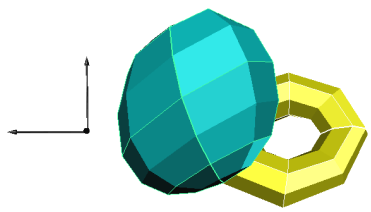
(b) Tesszelláció



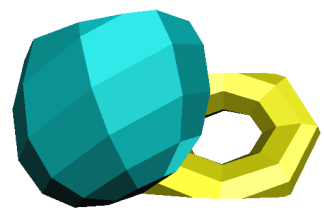
(c) Modellezési transzformáció



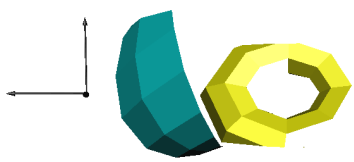
(d) Kamera transzformáció



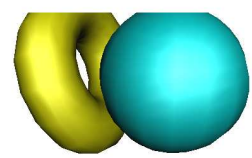
(e) Perspektív transzformáció



(f) Vágás

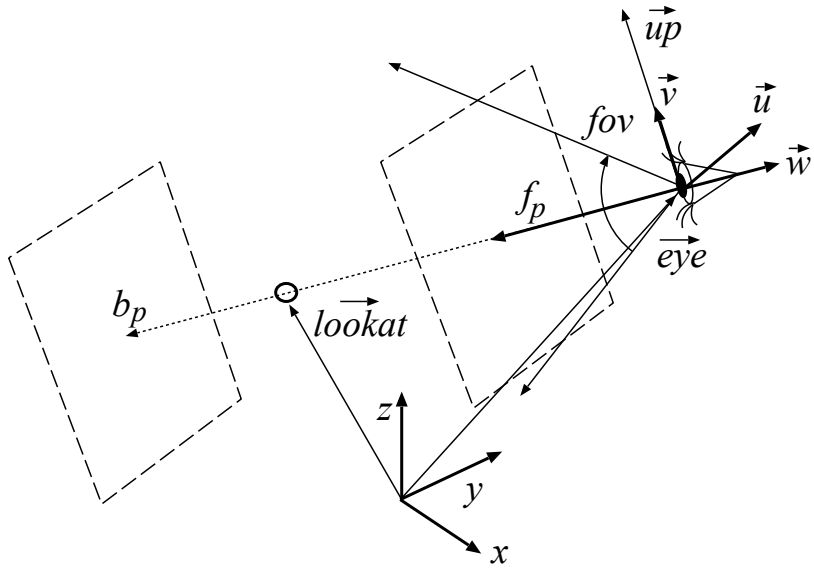


(g) Takarás



(h) Vetítés és színezés

40.37. 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.



40.38. 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).

lynek elemeit külön tárgyaljuk. A tényleges transzformációt viszont egyetlen 4×4-es mátrixszorzással valósítjuk meg, ahol a transzformációs mátrix az elemi transzformációk mátrixainak a szorzata.

40.7.1. 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 (40.38. 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 $\vec{w} = (w_x, w_y, w_z)$ nézeti irányba mutató egységvektorokat a

következő módon határozhatjuk meg:

$$\vec{w} = \frac{e\vec{y}e - l\vec{oo}kat}{|e\vec{y}e - l\vec{oo}kat|}, \qquad \vec{u} = \frac{u\vec{p} \times \vec{w}}{|u\vec{p} \times \vec{w}|}, \qquad \vec{v} = \vec{w} \times \vec{u}\ .$$

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:

$$[x', y', z', 1] = [x, y, z, 1] \cdot \mathbf{T}_{kamera} = [x, y, z, 1] \cdot \mathbf{T}_{eltol} \cdot \mathbf{T}_{forgat}\ , \qquad (40.30)$$

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:

$$[u_x, u_y, u_z, 1] \cdot \mathbf{T}_{forgat} = [\vec{u} \cdot \vec{u}, \vec{u} \cdot \vec{v}, \vec{u} \cdot \vec{w}, 1] = [1, 0, 0, 1].$$

40.7.2. A normalizáló transzformáció

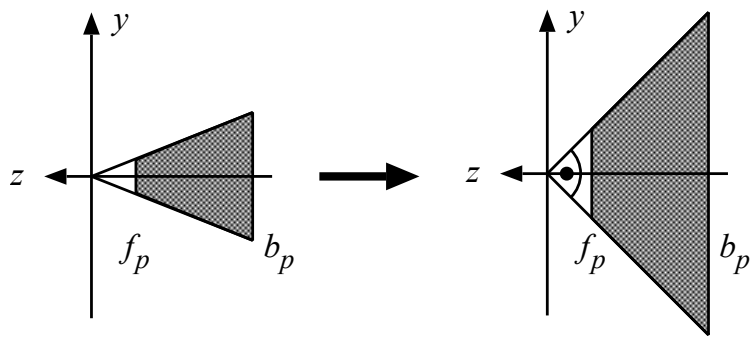
A további műveletekhez normalizáljuk a látható pontokat tartalmazó, a szem és az ablak által meghatározott gúlát oly módon, hogy a gúla csúcsában a nyílásszög 90 fok legyen.

A normalizálás egy egyszerű skálázás:

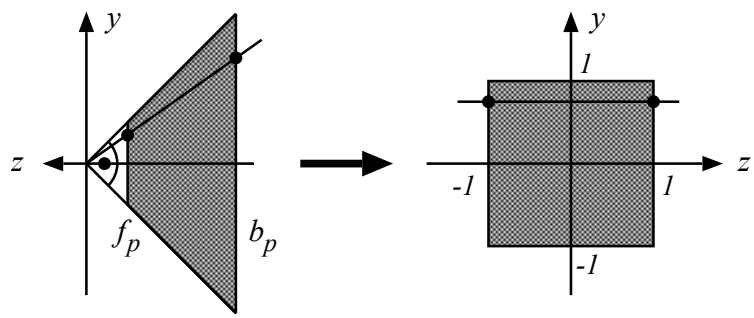
$$\mathbf{T}_{norm} = \begin{bmatrix} 1/(\tan(fov/2) \cdot aspect) & 0 & 0 & 0 \\ 0 & 1/\tan(fov/2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

40.7.3. 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.



40.39. ábra. A normalizáló transzformáció a látószöget 90 fokra állítja.



40.40. 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 normalizáló transzformáció után a képszintézisben résztvevő pontok tartománya egy szimmetrikus csonka gúla (40.39. 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 (40.40. ábra).

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 transzformációt 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 40.40. 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:

$$[m_x \cdot f_p, m_y \cdot f_p, -f_p, 1] \cdot \mathbf{T}_{persp} = [m_x, m_y, -1, 1] \cdot \lambda \ ,$$

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:

$$[m_x \cdot f_p, m_y \cdot f_p, -f_p, 1] \cdot \mathbf{T}_{persp} = [m_x \cdot f_p, m_y \cdot f_p, -f_p, f_p] \ . \qquad (40.31)$$

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 (40.31) 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 (40.5.2. 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.

40.7.4. 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 textitiátfordulási probléma kiküszöbölése miatt a vágást a homogén osztás előtt kell végrehajtani. A homogén koordinátás vágási határokat a képernyőkoordinátarendszerben megfogalmazott, egy AABB-t definiáló feltételek visszatranszformálásával kaphatjuk meg. A homogén osztás után a belső pontok

kielégítik a következő egyenlőtlenségeket:

$$-1 \le X = X_h/h \le 1, \qquad -1 \le Y = Y_h/h \le 1, \qquad -1 \le Z = Z_h/h \le 1\ . \tag{40.32}$$

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 (40.32) egyenlőtlenségeket h-val, így eljutunk a vágási tartomány homogén koordinátás leírásához:

$$-h \le X_h \le h, \qquad -h \le Y_h \le h, \qquad -h \le Z_h \le h\ . \tag{40.33}$$

A pontok vágása triviális feladat, hisz a homogén koordinátás alakjukra csak ellenőrizni kell, hogy teljesülnek-e a (40.33) 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 (40.33) egyenlőtlenségeket.

A Descartes-koordinátákkal dolgozó vágási algoritmusokkal a 40.4.3. pontban foglalkoztunk. Az ott megismert módszerek alkalmazhatók homogén koordinátákra is, azzal a különbséggel, hogy most a (40.33) 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 $X_h \le h$ félteret (a többi féltérre a vizsgálat teljesen hasonló). Három esetet kell megkülönböztetni:

1. Ha a szakasz mindkét végpontja belső pont, azaz $X_h^1 \le h^1$ és $X_h^2 \le h^2$, 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 (40.19) egyenletet, a vágósík pontjai pedig kielégítik az $X_h = h$ egyenletet, a metszéspont t_i paraméterét a

következőképpen határozhatjuk meg:

$$X_h(t_i) = h(t_i) \quad \Longrightarrow \quad X_h^1 \cdot (1-t_i) + X_h^2 \cdot t_i = h^1 \cdot (1-t_i) + h^2 \cdot t_i \quad \Longrightarrow$$

$$t_i = \frac{X_h^1 - h^1}{X_h^1 - X_h^2 + h^2 - h^1}$$

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 $I^1 \cdot (1-t_i) + I^2 \cdot t_i$.

40.7.5. 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.

40.7.6. 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

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:

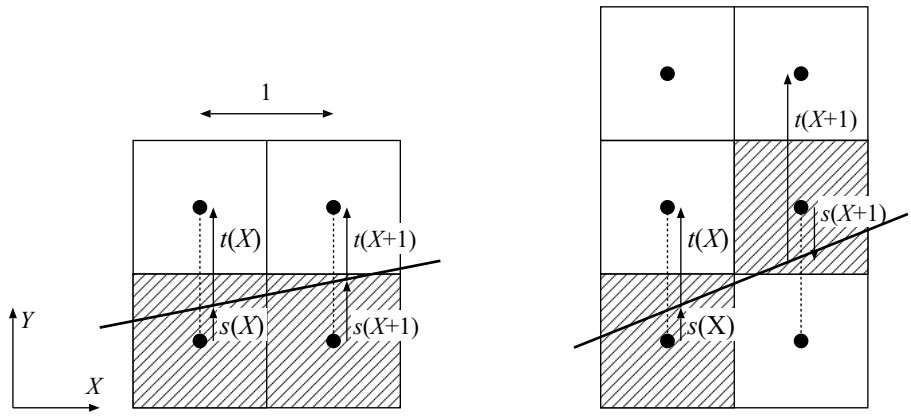
$$y = m \cdot X + b, \qquad \text{ahol} \quad m = \frac{Y_2 - Y_1}{X_2 - X_1}, \quad \text{és} \quad b = Y_1 - X_1 \cdot \frac{Y_2 - Y_1}{X_2 - X_1} \; , \tag{40.34}$$

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:



40.41. 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.

DDA-SZAKASZRAJZOLÁS$(X_1, Y_1, X_2, Y_2, \textit{szín})$

```
1  m = (Y_2 − Y_1)/(X_2 − X_1)
2  y = Y_1
3  for X = X_1 to X_2
4      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.***

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 (40.41. 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:

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*$)$

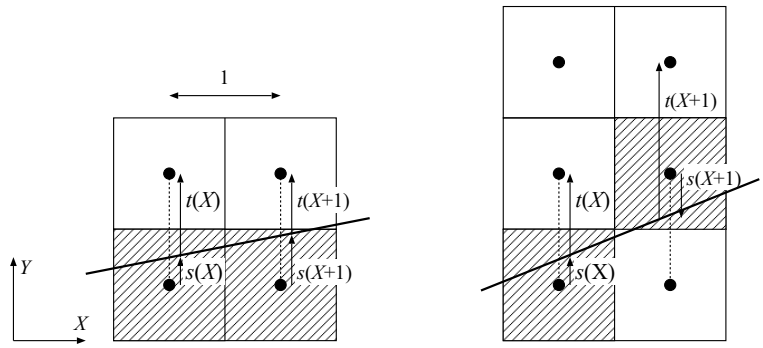
```
1  ΔX = X2 − X1
2  ΔY = Y2 − Y1
3  (dE+, dE−) = (2(ΔY − ΔX), 2ΔY)
4  E = −ΔX
5  Y = Y1
6  for X = X1 to X2
7      if E ≤ 0
8          E = E + dE−          // Döntési változó nempozitív,
                                // maradunk a pixelsorban.
9          E = E + dE+          // Döntési változó pozitív,
                                // 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 töröttvonallal, azaz több csúcstömbbel adhatjuk 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



40.42. ábra. Poligonkitöltés. A sokszög belsejében lévő pixeleket vízszintes pásztánként keressük meg.

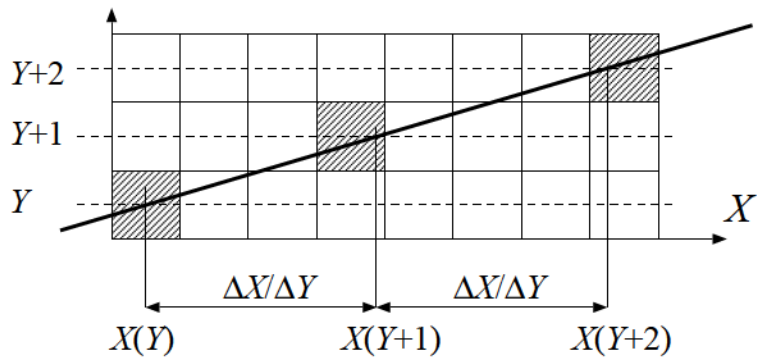
negyedik pont közötti, általában a $(2i+1)$-edik és $(2i+2)$-edik pont közötti pixeleket (40.42. ábra). Ez a módszer azokat a pontokat színezi ki, amelyeket ha végtelen távolból közelítünk meg, akkor páratlan számúszor kell átlépnünk a poligon határán.

A pászták és az élek metszéspontjainak kiszámítását a következő megfigyelésekkel gyorsíthatjuk:

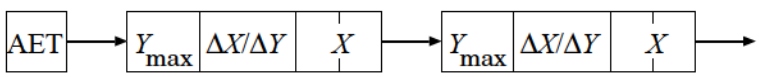
1. Egy él és a pászta között csak akkor keletkezhet metszéspont, ha a pászta Y koordinátája az él minimális és maximális Y koordinátája között van, ezért csak ezekre érdemes a metszéspontot kiszámítani. Az ilyen éleket ***aktív éleknek*** nevezzük. Az implementációhoz létre kell hoznunk az ***aktív él 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, nem-egész összeadással (40.43. ábra).

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



40.43. ábra. A pászta és az élek közötti metszéspont inkrementális számítása. Az X koordináta mindig az egyenes meredekségének a reciprokával nő.



40.44. ábra. Az aktív él lista szerkezete.

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.

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, amelyekmaximális 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      for a poligon összes él-ére            // Aktívvá váló élek az AET-be.
3          if él.ymin == Y
4              AETBE-RAK(él)
5          for minden él-re az AET-ben        // Az aktív létet befejező élek
                                              // törlése az AET-ből.
6              if él.ymax ≤ Y
7                  AETBŐL-TÖRÖL(él)
8          AET-RENDEZÉS                       // Rendezés X szerint.
9          for minden második egymást követő (él1, él2) párra az AET-ben
10             for X = KEREKÍT(él1.x) to KEREKÍT(él2.x)
11                 PIXEL-ÍRÁS(X, Y, szín)
11         for minden él-re az AET-ben        // Inkrementális elv.
12             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.

40.7.7. 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 ***mélység-puffernek.***

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.

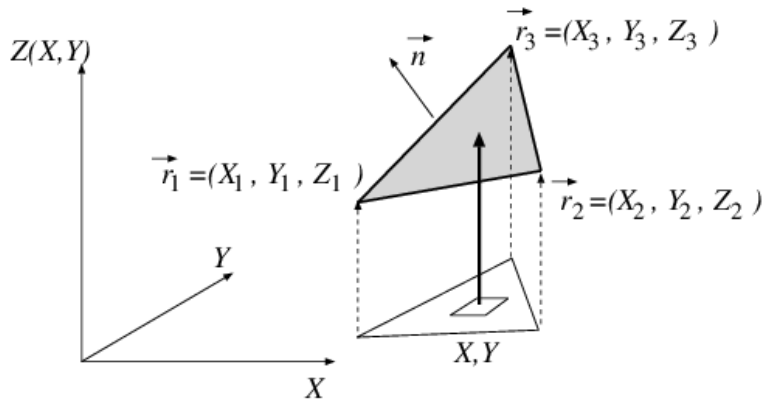
Amint a kitöltés során egy pixelhez érünk, kiszámítjuk a felületi pont Z koordinátáját és összehasonlítjuk a z-bufferben lévő értékkel. Ha az ott található érték kisebb, akkor a már feldolgozott háromszögek között van olyan, amelyik az aktuális háromszöget ebben a pontban takarja, így az aktuális háromszög 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      PIXEL-ÍRÁS(p, háttér-szín)
3      z-buffer[p] = a legnagyobb ábrázolható érték
4  for minden o háromszögre                               // Rajzolás.
5      for az o háromszög vetületének minden p pixelére
6          Z = az o háromszög p-re vetülő pontjának Z koordinátája
7          if Z < z-buffer[p]
8              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



40.45. 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.

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 (40.45. ábra).

Az X, Y pixel címből a megfelelő Z koordinátát a háromszög síkjának az egyenletéből számíthatjuk ((40.1) egyenlet), amely szerint a Z koordináta az X, Y koordináták lineáris függvénye. A háromszög síkjának az egyenlete:

$$n_X \cdot X + n_Y \cdot Y + n_Z \cdot Z + d = 0, \qquad \text{ahol } \vec{n} = (\vec{r}_2 - \vec{r}_1) \times (\vec{r}_3 - \vec{r}_1) \quad \text{és} \quad d = -\vec{n} \cdot \vec{r}_1 \ . \tag{40.35}$$

A háromszög balállású, illetve jobbállású voltát a normálvektor Z koordinátájának előjele 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.

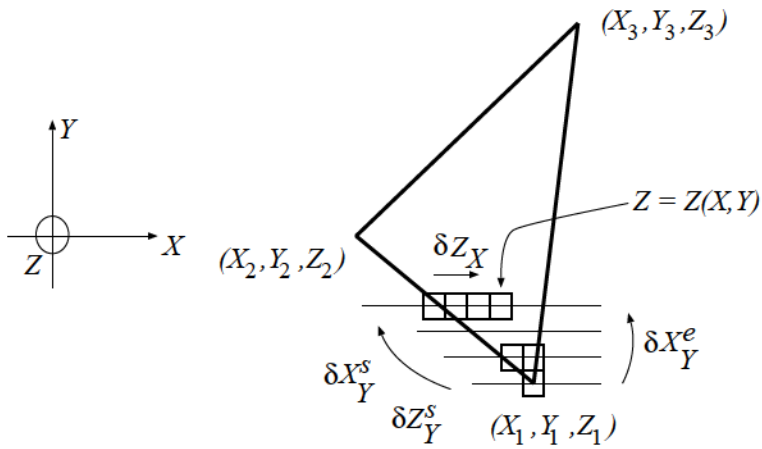
A háromszög síkjának az egyenletéből a $Z(X, Y)$ függvény:

$$Z(X, Y) = -fracn_X \cdot X + n_Y \cdot Y + dn_Z \ . \tag{40.36}$$

Az inkrementális elv felhasználásával ezen képlet jelentősen egyszerűsíthető:

$$Z(X+1, Y) = Z(X, Y) - \frac{n_X}{n_Z} = Z(X, Y) + \delta Z_X \ . \tag{40.37}$$

Mivel a δZ_X paraméter állandó az egész háromszögre, csak egyszer kell kiszámítani. Egyetlen pásztán belül a Z koordináta kiszámítása tehát egyetlen

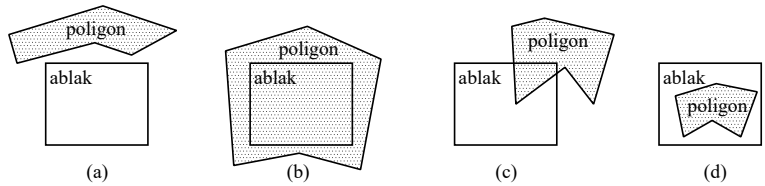


40.46. ábra. Inkrementális Z érték számítás egy balállású háromszögre.

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 (40.46. 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 $\vec{n} = ((X_2, Y_2, Z_2) - (X_1, Y_1, Z_1)) \times ((X_3, Y_3, Z_3) - (X_1, Y_1, Z_1))$
// Normálvektor.
2 $\delta Z_X = -n_X / n_Z$ // Z inkremens, ha X eggyel nő.
3 $(\delta X_Y^s, \delta Z_Y^s, \delta X_Y^e) =$
$((X_2 - X_1)/(Y_2 - Y_1), (Z_2 - Z_1)/(Y_2 - Y_1), (X_3 - X_1)/(Y_3 - Y_1))$
4 $(X_{bal}, X_{jobb}, Z_{bal}) = (X_1, X_1, Z_1)$
5 **for** $Y = Y_1$ **to** Y_2
6 $Z = Z_{bal}$
7 **for** $X =$ KEREKÍT(X_{bal}) **to** KEREKÍT(X_{jobb}) // Egy pászta rajzolása.
8 **if** $Z <$ *z-buffer*$[X, Y]$ // Takarás vizsgálat.
9 PIXEL-ÍRÁS$(X, Y, szín)$
10 *z-buffer*$[X, Y] = Z$
11 $Z = Z + \delta Z_X$
12 $(X_{bal}, X_{jobb}, Z_{bal}) = (X_{bal} + \delta X_Y^s, X_{jobb} + \delta X_Y^e, Z_{bal} + \delta Z_Y^s)$
// Következő pászta.



40.47. ábra. Poligon-ablak relációk: **(a)** különálló; **(b)** körülvevő; **(c)** metsző; **(d)** tartalmazott.

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 [155]. 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 40.47. 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 alkalmazzuk a kétdimenziós szakaszvágó al-

goritmust (40.4.3. 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)

```
1  if X1 ≠ X2 vagy Y1 ≠ Y2                    // Az ablak a pixelnél nagyobb?
2     if legalább egy él esik az ablakba      // Ablakfelezés és rekurzió.
3        WARNOCK-ALGORITMUS(X1, Y1, (X1 + X2)/2, (Y1 + Y2)/2)
4        WARNOCK-ALGORITMUS(X1, (Y1 + Y2)/2, (X1 + X2)/2, Y2)
5        WARNOCK-ALGORITMUS((X1 + X2)/2, Y1, X2, (Y1 + Y2)/2)
6        WARNOCK-ALGORITMUS((X1 + X2)/2, (Y1 + Y2)/2, X2, Y2)
7        return      // Triviális eset: az (X1, Y1, X2, Y2) téglalap homogén.
8  poligon = az ((X1 + X2)/2, (Y1 + Y2)/2) pixelben látható poligon
9  if nincs poligon
10    (X1, Y1, X2, Y2) téglalap kitöltése háttér színnel
11    else (X1, Y1, X2, Y2) 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.

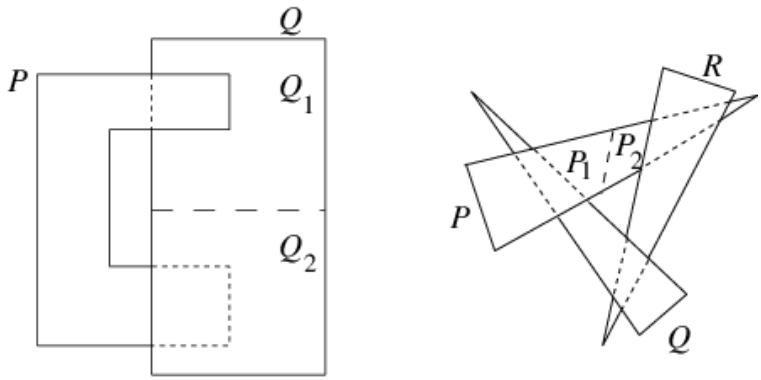
A felsorolt feltételek bármelyike elégséges feltétel, amelyek ellenőrzése a sorrendnek megfelelően egyre nehezebb, ezért az ellenőrzéseket a fenti sorrendben végezzük el.

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.

Előfordulhat, hogy ez az algoritmus végtelen ciklusba kerül. Például ha két poligon egymást kölcsönösen takarja (40.48. 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



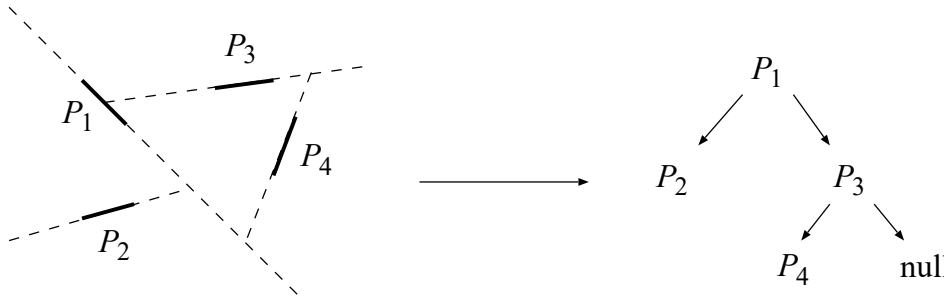
40.48. ábra. Ciklikus takarás, amelyet úgy oldhatunk fel, hogy az egyik sokszöget a másik síkjával kettévágunk.

40.6.2. 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 (40.49. 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     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
```



40.49. 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

40.7-1. Készítsük el a Bresenham-algoritmus teljes, mind a 8 síktartományt kezelő változatát.

40.7-2. 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.

40.7-3. Készítsük el a z-buffer algoritmus teljes változatát, amely mind balállású, mind pedig jobbállású háromszögekre működik.

40.7-4. 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.

40.7-5. 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.

40.7-6. Alkalmazzuk a BSP-fát diszkrét idejű ütközésfelismeréshez.

40.7-7. Alkalmazzuk a BSP-fát a sugárkövető eljárás térpartícionáló adatszerkezeteként.

Feladatok

40-1 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.

40-2 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.

40-3 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 [165] kiváló könyvében, a projektív geometriáról általában Maxwell [253, 254] és Coxeter [85] műveiben, a számítógépes grafikai alkalmazásáról pedig Herman Iván [182] és Krammer Gergely [216] cikkeiben olvashatunk. A görbék és felületek modellezésével a számítógépes geometriai tervezés (CAD, CAGD) foglalkozik, amelyet átfogóan Gerald Farin [122], valamint Rogers és Adams [309] tárgyalnak. A geometriai modellek mérési eredményekből történő felépítése a *mérnöki visszafejtés* [356] területe. Az implicit felületek tanulmányozásához Bloomenthal művét [54] ajánljuk.

Ezt az irodalomjegyzéket HBibTEXsegítségével állítottuk össze. A dokumentumokat elsősorban a szerzők neve (első szerző, második szerző stb.),

másodsorban a megjelenés éve, harmadsorban pedig a dokumentumok címe alapján rendeztük.

Az aláhúzás azt jelzi, hogy az aláhúzott szövegrészre kattintva a megfelelő honlapra juthatunk. A hivatkozás végén lévő kék számok pedig azt mutatják, mely oldalakon van hivatkozás az adott dokumentumra.

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 [371]. 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 *Algoritmusok* [82] című könyv 33. fejezete is foglalkozik, további információk Preparata és Shamos [293],a [333] cikk alapján mutattuk be. valamint Marc de Berg műveiben [89, 90] találhatók. Az 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 [90, 322, 383], bár Chazelle [72] egy optimális, lineáris idejű algoritmus elvét is kidolgozta. A *két fül tétel* idézett bizonyítása Joseph O'Rourke-től származik [274]. A háromszöghálókon működő pillangó felosztást Dyn és társai [109], javasolták. A *Sutherland-Hodgeman-poligonvágást* pedig a [333] cikk alapján mutatjuk be.

Az ütközésfelismerés a számítógépes játékok [338] 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 [198]. A felosztott felületekről sok hasznos információt kaphatunk Catmull és Clark eredeti cikkéből [70], Warren és Weimer könyvéből [362], valamint Brian Sharp ismertetőiből [325, 324]. A pillangófelosztást Dyn, Gregory és Levin [109] javasolta. A sugárkövetés elveivel Glassner [145] könyvében ismerkedhetünk meg. A *3D szakaszrajzoló algoritmust* Fujimoto és társai [137] cikke alapján tárgyaljuk. A sugárkövető algoritmusok bonyolultságát számos publikációban vizsgálták. Bebizonyosodott, hogy N objektumra a sugárkövetési feldatot $O(\lg N)$ időben meg lehet oldani [89, 339], 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 gyakor-

latban 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 [267, 339] 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 [177] olvashatunk, egy konkrét, optimalizált megvalósítás pedig Szécsi László cikkében [334] található.

A virtuális világ valószínűségi modelljében használt Poisson-pontfolyamat ismertetése megtalálható például Karlin és Taylor [201], valamint Lamperti [222] könyveiben.

Az alkalmazott cellafelezési módszer Havrantól [177] származik. Az idézett integrálgeometriai tétel megtalálható Santaló [319] könyvében.

A négyfa és az oktálisfa geoinformatikai alkalmazásait a kötet **??**. fejezete tárgyalja.

Az inkrementális képszintézis algoritmusaival Jim Blinn foglalkozik részletesen [53], C++ nyelvű megvalósítást a [337] könyvben találhatunk, valamint más általános számítógépes grafika könyvekhez is fordulhatunk [131, 338]. A láthatósági algoritmusok összehasonlító elemzését például a [333, 336] művekben találjuk meg.

A Bresenham-algoritmus forrása [61]. 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 [272] javasolták. A BSP-fa felépítésére Fuchs [136] 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 [124] olvashatunk.

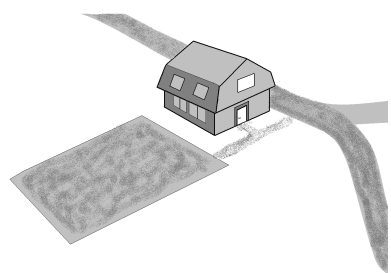
41. Térinformatika

A térinformatika (vagy geoinformatika) a térképészet és az informatika határán kialakult tudományterület.

Ebben a fejezetben a legfontosabb térinformatikai fogalmakat és módszereket mutatjuk be: a 41.1. alfejezetben az adatmodellekkel, a 41.2. alfejezetben a térbeli indexeléssel, a 41.3. alfejezetben a digitális szűrési eljárásokkal, és végül a 41.4. alfejezetben a mintavételezéssel foglalkozunk.

41.1. 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 41.1. ábrán a mintaterület egy nem térképszerű ábrázolását látjuk, amelynek vektoros, majd a raszteres reprezentációját is ismertetjük.

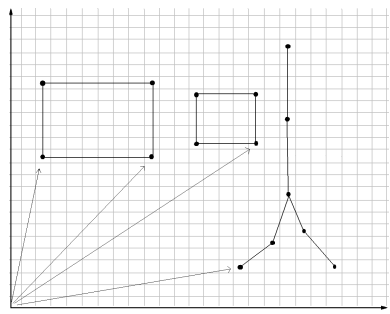


41.1. ábra. A mintaterület madártávlatból.

41.1.1. 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 41.2. ábrán az előbbi terület vektoros modellje látható.



41.2. ábra. A mintaterület vektoros modelljel.

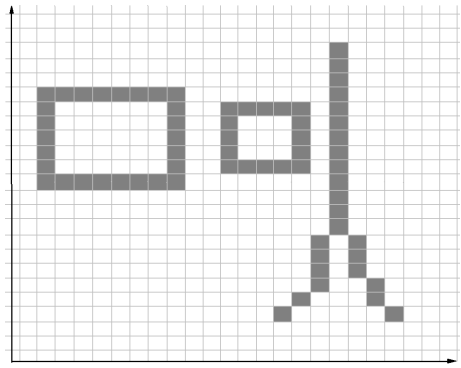
41.1.2. A raszteres modell

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 41.3. ábrán a mintaterület raszteres modelljét láthatjuk.

A raszteres térképek legnagyobb előnye a felszín rendkívül részletgazdag leírása. Hátrányuk a nagy erőforrásigény, és a bonyolult objektumok felépítésének, valamint a relációs adatkapcsolatok létrehozásának nehézsége.

Gyakorlatok

41.1-1. Hasonlítsuk össze a vektoros és raszteres adatmodellt.



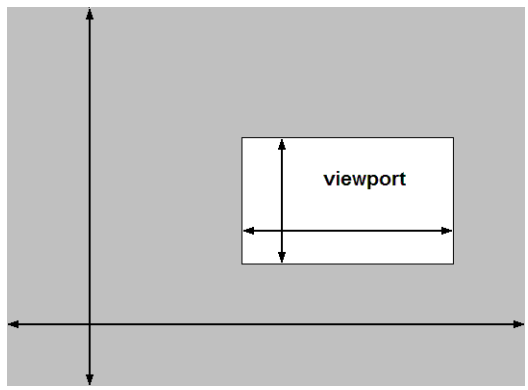
41.3. ábra. A mintaterület raszteres modelljel.

41.2. 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 (41.4. 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 segít-

séget 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.



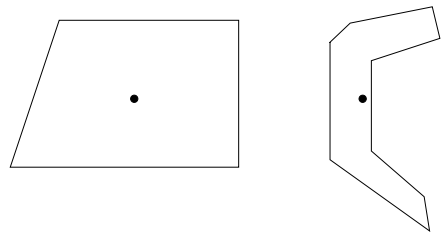
41.4. ábra. A viewport és az adatbázis térbeli kiterjedésének viszonya.

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 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.

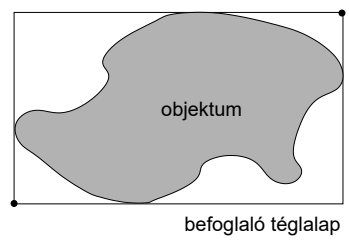
Még mielőtt rátérnénk néhány térbeli indexelési algoritmus bemutatására, ismerkedjünk meg a centroid és a befoglaló téglalap fogalmával. A ***centroid*** egy olyan pont, amely a poligon belsejében van, és alkalmas a poligon térbeli elhelyezkedésének ábrázolására. Konvex poligonok esetében a centroid kézenfekvő definíciója a poligon súlypontja, míg konkáv esetben egy súlyvonalra illeszkedő belső pont (konkrét definíciója többféleképp is lehetséges és elterjedt) (41.5. ábra).

A ***befoglaló téglalap*** (MBR) az a legkisebb területű téglalap, melynek oldalai párhuzamosak a koordináta-tengelyekkel, valamint teljes egészében tartalmazza a kérdéses objektumot (41.6. 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



41.5. ábra. Konvex és konkáv poligonok centroidjai.l.



41.6. ábra. A befoglaló téglalap..

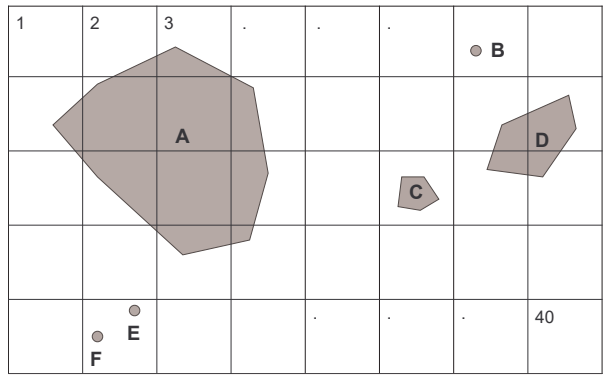
tárolunk – a vektoros adatmodellnek megfelelően. Az objektumok lehetnek nulladimenziósak, vagyis egy darab ponttal reprezentáltak, lehetnek egydimenziósak, mint például utak, folyók vagy általában a vonalláncok, vagy kétdimenziósak, például poligonok. A fent bevezetett két fogalom a poligonok, vonalláncok, valamint általában síkbeli objektumok helyettesítésére való. Azért használjuk őket, mert a térbeli indexeléskor ezen egyszerű objektumokkal reprezentáljuk az időnként rendkívüli összetettségű térbeli objektumokat.

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.

41.2.1. Grid index

Illesszünk a síkbeli objektumainkat tartalmazó síkrészre egy szabályos négyzethálót, ahogy a 41.7. 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 cel-



41.7. ábra. Különböző méretű objektumok egy adott rácsállandójú négyzethálón..

lába 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 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.

cella azonosító	objektum azonosító
7	B
22	C
34	F
34	E

objektum azonosító	egyéb attribútumok
A	
B	
C	
D	
E	
F	

41.8. ábra. Az index tábla és az objektumok jellemzőit tartalmazó tábla..

A 41.9. ábra az index tábla felépítését szemlélteti a 41.7. 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.

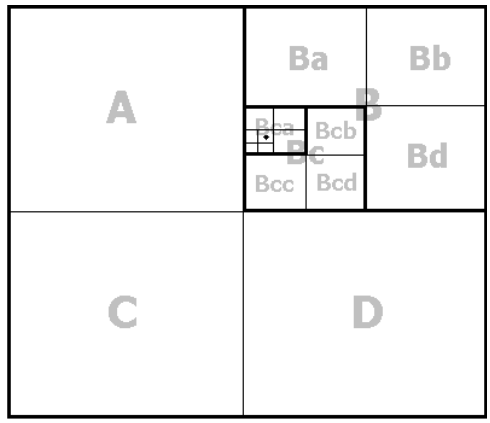
Amint a 41.7. ábrán megfigyelhető, a cellák mérete nem mindig van összhangban az objektum méretével. Ha túl nagyok a cellák, akkor túl sok objektum esik egy adott cellába, ami szélsőséges esetben nem gyorsít a keresésen. Ha túl kicsik a cellák, akkor a nagyobb objektumok túl sok cellát fednek le, ami 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.

41.2.2. Négy-fa

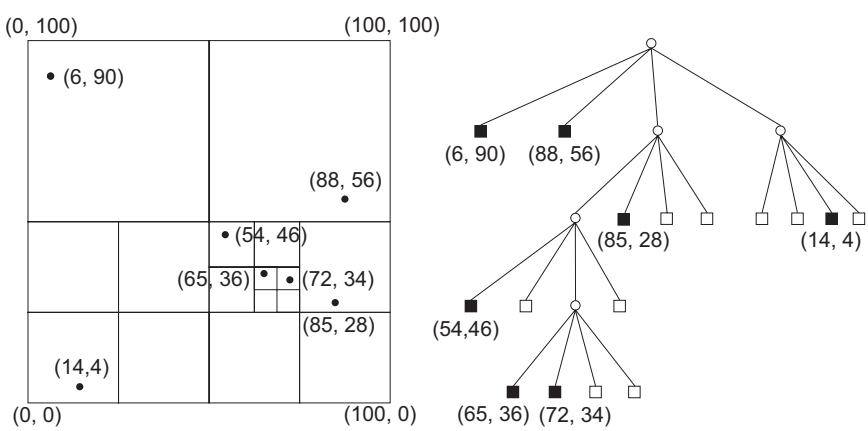
A ***négy-fa*** egy hierarchikus adatszerkezet, amely az oszd-meg-és-uralkodj elvhez hasonló rekurzív felbontás elvén alapszik. A sokféle négy-fa változatból mi most az úgynevezett pont-tartomány négy-fát vázoljuk fel. Az indexelés alapja az adott síktartomány rekurzív felosztása négy síknegyedre (41.10. ábra).

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



41.9. ábra. Az index tábla és az objektumok jellemzőit tartalmazó tábla..

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 41.10. ábra egy hét objektumot tartalmazó négy-fát szemléltet, feltüntetve az objektumok koordinátáit.



41.10. ábra. Hét objektumot tartalmazó pont-tartomány négy-fa...

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 négyfában 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 paraméter), 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     if ¬ üres[n] ∧x1 ≤ koordX[n] ≤ x2 ∧ y1 ≤ koordY[n] ≤ y2
3        return n        // Megtaláltunk egy viewporton belüli objektumot.
4     else if felosztásX[n] > x1∧ felosztásY[n] < y1        // Gyerekek
                rekurzív bejárása.
5           NÉGY-FA-KERES(NW[n], x1, y1, x2, y2)
6     if felosztásX[n] < x2∧ felosztásY[n] < y1
7        NÉGY-FA-KERES(NE[n], x1, y1, x2, y2)
8     if felosztásX[n] < x2∧ felosztásY[n] > y2
9        NÉGY-FA-KERES(SE[n], x1, y1, x2, y2)
10    if felosztásX[n] > x1∧ felosztásY[n] > y2
11       NÉGY-FA-KERES(SW[n], x1, y1, x2, y2)
```

A fa mérete és alakja fontos a keresés, beszúrás és törlés műveletek

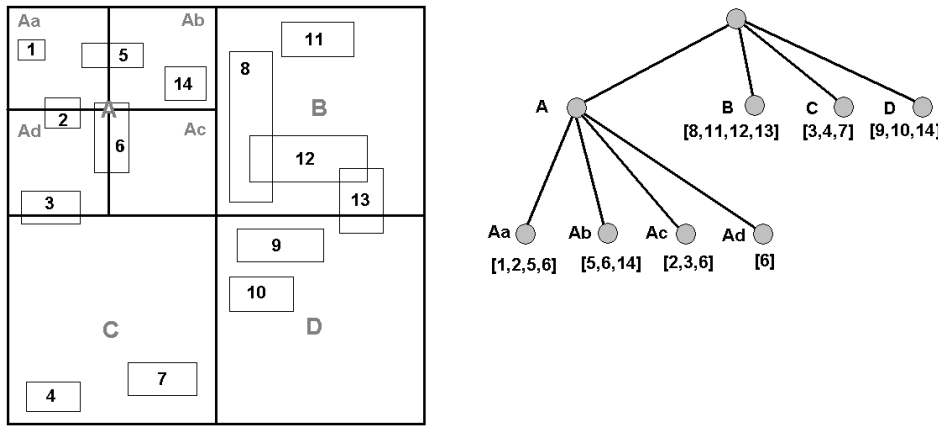
hatékonyságának szempontjából. A pont-tartomány négy-fa szerkezete a fix térfelosztás miatt független a beszúrások sorrendjétől, alakja és mérete viszont függ a beszúrt objektumoktól. A fa műveleteinek futásideje alapvetően a fa mélységétől függ. A fa minimális mélysége M objektum esetén $\lceil \log_4(M-1) \rceil$, 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 legalább d, valamint a tartományunk kiterjedése kezdetben r, akkor a fa mélységére adható egy $\lceil \lg(\sqrt{2}(r/d)) \rceil$ 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 41.11. 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.



41.11. ábra. Négy-fa és a poligonok befoglaló négyszögei...

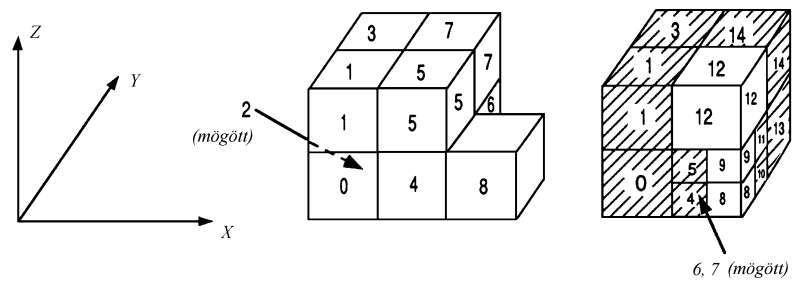
41.2.3. Nyolc-fa

A négy-fa háromdimenziós kiterjesztése a ***nyolc-fa.*** Speciális esetekben használatos, mint például háromdimenziós objektumok (geológiai képződmények, földalatti olajtárolók stb.) leírására. A 41.12. 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 41.12. 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.



41.12. ábra. A nyolc-fa térfelosztása...

Gyakorlatok

41.2-1. Bővítsük a **??**. ábrán szereplő négy-fát egy $(60, 38)$ és egy $(67, 30)$ koordinátájú objektummal.

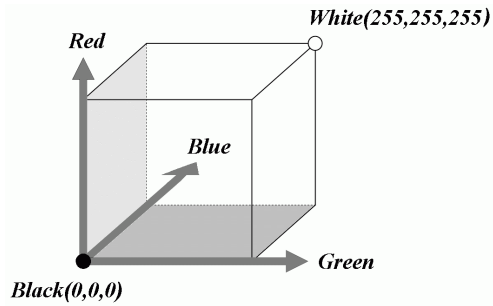
41.2-2. Lássuk be, hogy adott koordinátájú pont keresése a négy-fában $O(h)$ időben megoldható, ahol h a fa mélysége.

41.3. Digitális szűrési eljárások

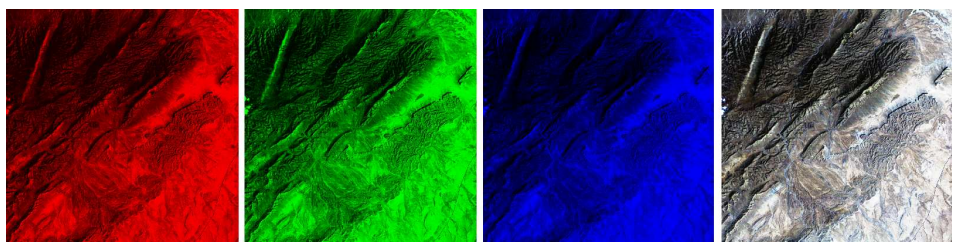
A térinformatika egy másik fontos területe a digitális képek, űrfotók feldolgozása. A képi adatokat kezelő szoftverek számos képfeldolgozó, szűrő algoritmuson alapuló eljárást tartalmaznak. Ezen eljárások közül mutatunk be néhány fontosabbat, mint például alul- és felülvágó szűrők, élmegőrző, éldetektáló 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.

41.3.1. 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



41.13. ábra. Az RGB színkocka..



41.14. ábra. Egy LANDSAT műhold felvétele: az RGB sávok és a sávokból összeállított kép..

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 41.13. ábra.

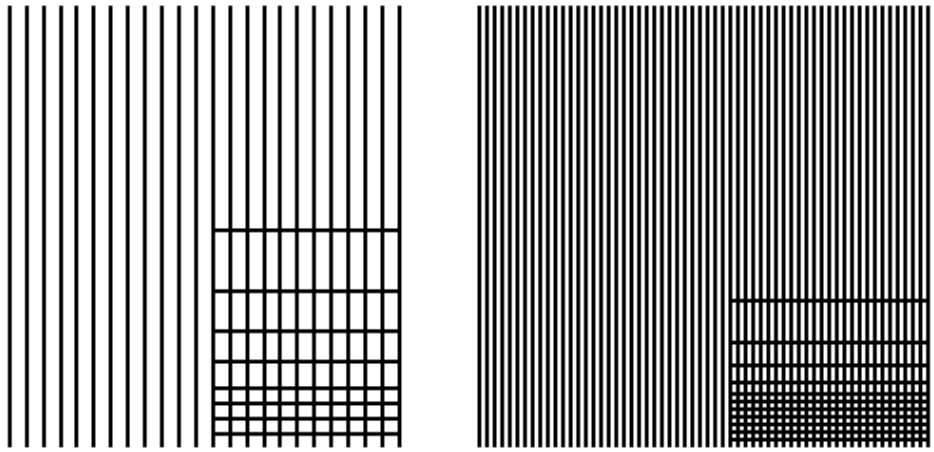
Bármely szín előállítható e három alapszín kombinációjából, a következő módon:

$$colour = a \cdot Red + b \cdot Green + c \cdot Blue \ ,$$

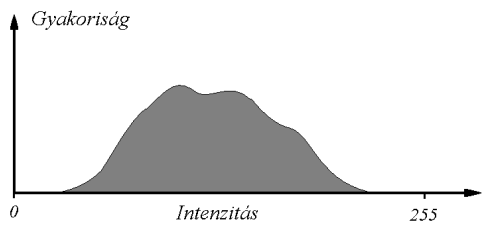
ahol a, b, c az egyes alapszínek intenzitása.

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 41.14. 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



41.15. ábra. A térbeli frekvencia szemléletes jelentése..



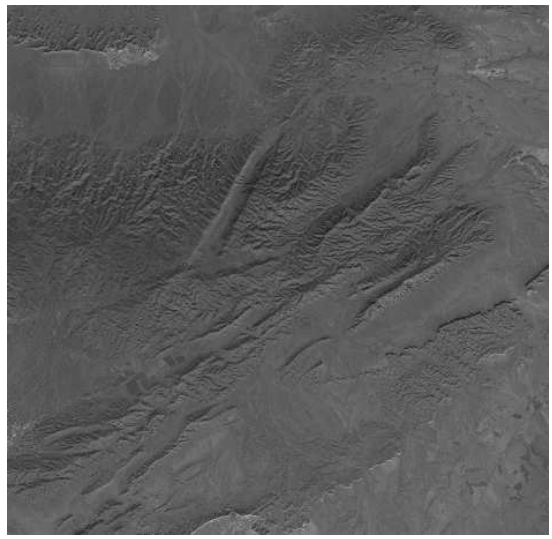
41.16. ábra. Egy kép intenzitás értékeinek eloszlása (hisztogramja)..

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 41.15. ábra a térbeli frekvencia fogalmát szemlélteti.

41.3.2. 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 41.16. 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 meg-



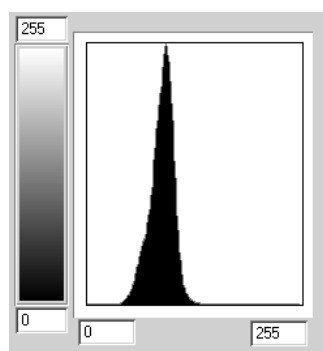
41.17. ábra. Az eredeti SPOT felvétel..

növeltük a pixelek közti intenzitás különbséget, ezáltal kontrasztosabbá tettük a képet.

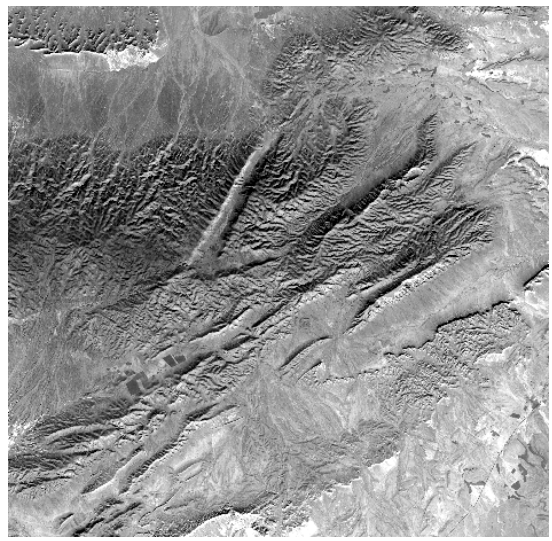
Tekintsünk egy valós esetet, mégpedig a 41.17. ábrát, mely egy francia SPOT műhold által készített kép.

A kép hisztogramját a 41.18. 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 41.19. ábra mutatja.



41.18. ábra. Az eredeti SPOT felvétel..



41.19. ábra. A hisztogram kiegyenlítéssel megnövelt kontrasztú kép..

41.3.3. 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:

$$S(f) = \int_{-\infty}^{\infty} s(t)e^{-2\pi ift}dt \ ,$$

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.***

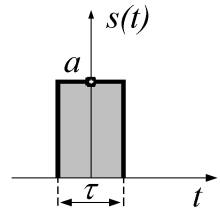
Írjuk fel a következő kifejezést, amelyet inverz Fourier-transzformációnak is neveznek:

$$s(t) = \int_{-\infty}^{\infty} S(f)e^{2\pi ift}dt \ .$$

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.



41.20. ábra. A négyszögimpulzus..

41.3.4. 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 (41.20. ábra), amely a következő:

$$s(t) \;=\; \begin{cases} a, & \text{ha } |t| < \tau/2\,, \\ 0 & \text{egyébként}\quad . \end{cases}$$

Végezzük el a függvény Fourier-transzformációját:

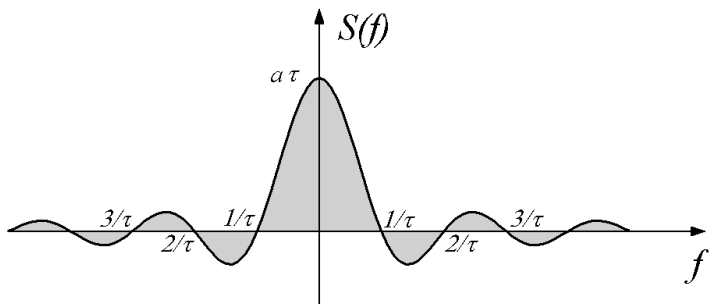
$$S(f) = \int_{-\infty}^{\infty} s(t) e^{-2\pi i f t} dt = a \int_{-\tau/2}^{\tau/2} e^{-2\pi i f t} dt = a\tau \frac{\sin \pi f \tau}{\pi f \tau}\ .$$

Ezt a függvényt szinusz kardinálisz függvénynek is nevezik és *sinc*-vel jelölik (41.21. 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 számos vonatkozásban megkönnyíti a digitális adatrendszerek leírását. Definíció szerint legyen δ egy olyan függvény (klasszikus értelemben nem az, hanem úgynevezett disztribúció), melyre $\delta(t) = 0$ ha $t \neq 0$,



41.21. ábra. A négyszög impulzus Fourier-transzformáltja a sinc függvény..

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

$$\int_{-\infty}^{\infty} \delta(t - t_0)f(t)dt = f(t_0) \ .$$

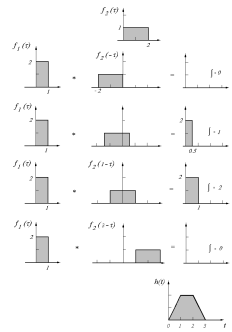
Ez a kiválasztási képesség, vagyis a $\delta(t_0)$ az $f(t)$ függvényből kiválasztotta a t_0 -hoz tartozó értéket.

A következőkben vizsgáljuk meg egy Dirac-impulzus és egy Dirac-impulzus sorozat Fourier-transzformáltját. Jelölje F a Fourier-transzformációt, amely a definíciója alapján

$$F\{\delta(t)\} = \int_{-\infty}^{\infty} \delta(t)e^{-2\pi ift} = 1 \ .$$

Nem az origóban lévő Dirac-impulzus Fourier-transzformáltja pedig

$$F\{\delta(t - a)\} = \int_{-\infty}^{\infty} \delta(t - a)e^{-2\pi ift} = e^{-2\pi ifa} \ ,$$



41.22. ábra. Két négyszög függvény konvolúciója az idő tartományba..

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:

$$F\left\{\sum_{k=-\infty}^{\infty} \delta(t - k\tau)\right\} = \frac{1}{\tau} \sum_{k=-\infty}^{\infty} \delta(f - k/\tau) \ .$$

41.3.5. Konvolúció

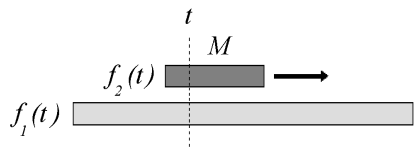
Legyenek f_1 és f_2 folytonos függvények. Jelölje ***konvolúciójukat*** $h = f_1 * f_2$, melyet a következő kifejezés definiál:

$$h(t) = f_1(t) * f_2(t) = \int_{-\infty}^{\infty} f_1(\tau) f_2(t - \tau) d\tau \ .$$

A 41.22. 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 41.23. ábra szemlélteti.

Az eddigiekben csak egydimenziós függvényekkel foglalkoztunk. Könnyen általánosítható a konvolúció fogalma kétdimenziós függvényekre is, mint amilyen a digitális kép.



41.23. ábra. A konvolúció szemléletes jelentése..

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.

41.3.6. 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 szűrőegyüttthatókat, vagyis a kernelbe töltendő szűrőegyüttthatókat. A digitális konvolúció tehát a szűrések végrehajtásának egyik lehetséges módja. Ebben az esetben a szűrést az idő tartományban végezzük a következő módon:

$$g'(t) = g(t) * s(t) \ ,$$

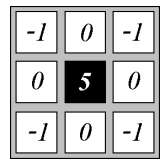
ahol $g(t)$ az eredeti adatrendszer az idő tartományban, $g'(t)$ a szűrt adatrendszer és $s(t)$ a kernel.

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.

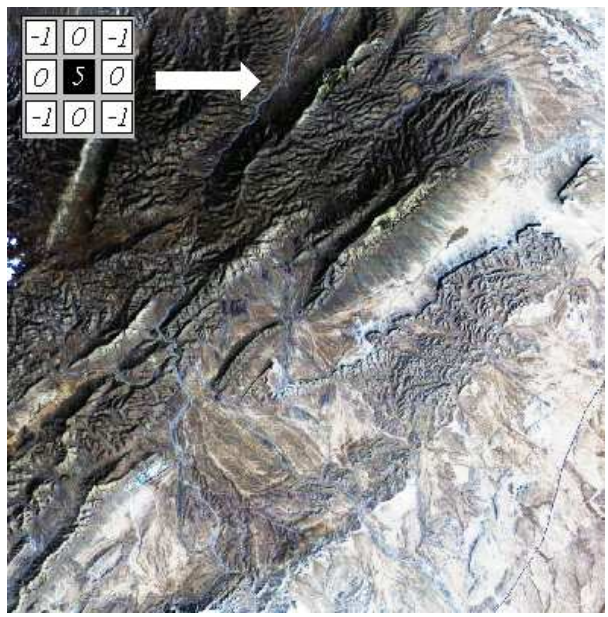
$$\begin{aligned} G(f) &= F\{g(t)\} \ , \\ G'(f) &= G(f)S(f) \ , \\ g'(t) &= F^{-1}\{G'(f)\} \ , \end{aligned}$$

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



41.24. ábra. A 3×3 pixeles kernel..



41.25. ábra. Az erősen felnagyított kernel mozgása a képen (LANDSAT képrészlet)..

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 41.25. ábra a feldolgozás folyamatát szemlélteti egy háromszor hármas – erősen felnagyított – kernellel.

KONVOLÚCIÓ(F, H, K, M)

```
1  N = K^2                                  ▷ Beállítjuk a normalizációs konstanst.
2  for x = 0 to M − K
3      for y = 0 to M − K                   // Sorra vesszük az eredeti kép pontjait.
4          for i = 0 to K − 1
5              for j = 0 to K − 1           // Sorra vesszük a kernel pontjait.
6                  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.

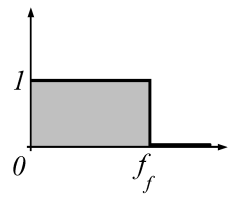
Az N normalizációs konstans elméletileg tetszőleges (nullától különböző) értéket felvehet, az algoritmus most a kernel pontjainak számával normalizál. A feldolgozás során bejárjuk az eredeti $M \times M$ méretű F kép minden – nem képhatáron lévő – pontját, melyek száma $(M - K + 1)^2$, mindig feldolgozva 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 $O((M - K + 1)^2 K^2)$.

Felülvágó szűrő

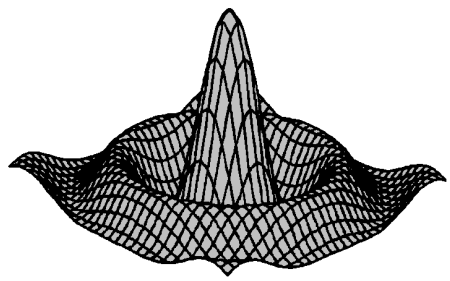
Akkor használunk felülvágó szűrőt, amikor a frekvencia tartományban egy bizonyos felső határfrekvenciánál (f_f) nagyobb frekvenciákat 0-val szorzunk, és a nála kisebbeket 1-gyel. A 41.26. ábra mutatja az ideális felülvágás átviteli függvényét. Ami a frekvencia tartományban szorzás, az idő tartományban konvolúció.

A négyszög függvényről tudjuk, hogy Fourier-transzformáltja a *sinc* függvény. A 41.27. 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.



41.26. ábra. A felülvágó szűrő átviteli függvénye (Fourier-transzformáltja a *sinc* függvény)...

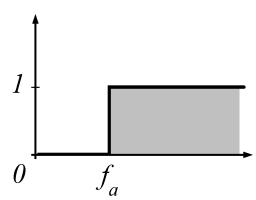


41.27. ábra. A kétdimenziós *sinc* függvény, mint az ideális felülvágás kernel függvénye..

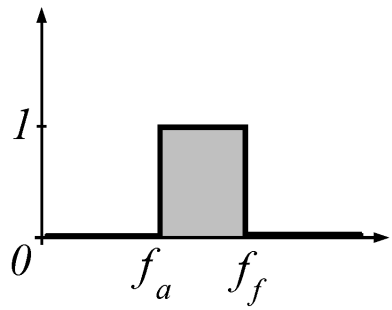
Alulvágó szűrő

Az alulvágó szűrőt akkor alkalmazzuk, ha az alacsony frekvenciás jeleket el akarjuk tüntetni a képről. Átviteli függvénye a 41.28. 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 határfrekvenciájú felülvágóhoz tartozó *sinc* függvényt levonjuk 1-ből, akkor az ugyanazon (alsó) határfrekvenciájú alulvágó szűrő szűrőegyüttható it kapjuk meg.



41.28. ábra. Az ideális alulvágás átviteli függvénye...



41.29. ábra. Az ideális sávszűrő átviteli függvénye..

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

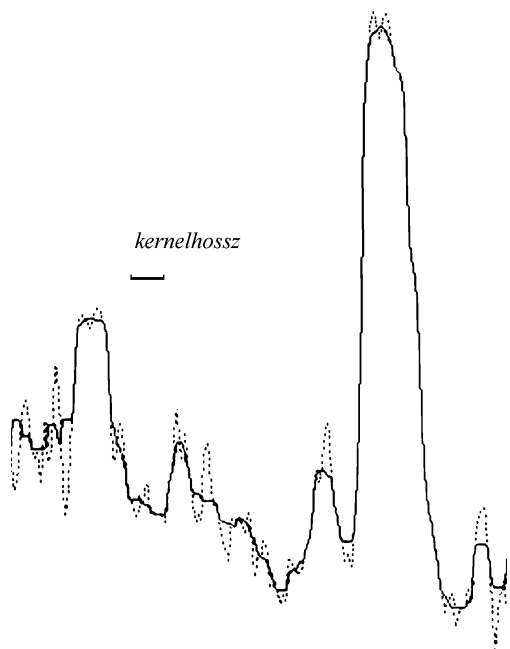
A sávszűrő egy olyan szűrő, amely egy adott alsó határfrekvencia alatt és egy felső határfrekvencia felett nem enged át. Átviteli függvénye a 41.29. ábrán látható. Kerneljét úgy kaphatjuk meg, hogy kiszámítjuk az f_f felső határfrekvenciához, majd az f_a alsó határfrekvenciához tartozó *sinc* függvényeket, 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.

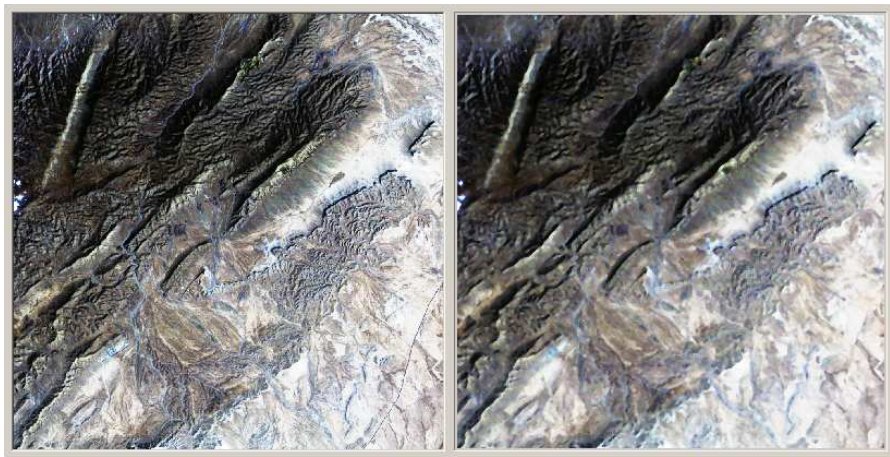
Élmegőrző szűrők

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. A 41.30. 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



41.30. ábra. Egy egydimenziós időfüggvény (szaggatott vonal) és medián-szűrt változata (folytonos vonal)..



41.31. ábra. Egy egydimenziós időfüggvény (szaggatott vonal) és medián-szűrt változata (folytonos vonal)..

rendezettek. Nem éleken azonban erőteljesen simít. A simítás mértéke a kernel hosszától függ, annál jobban simít, minél hosszabb. A 41.31. 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 menynyire 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

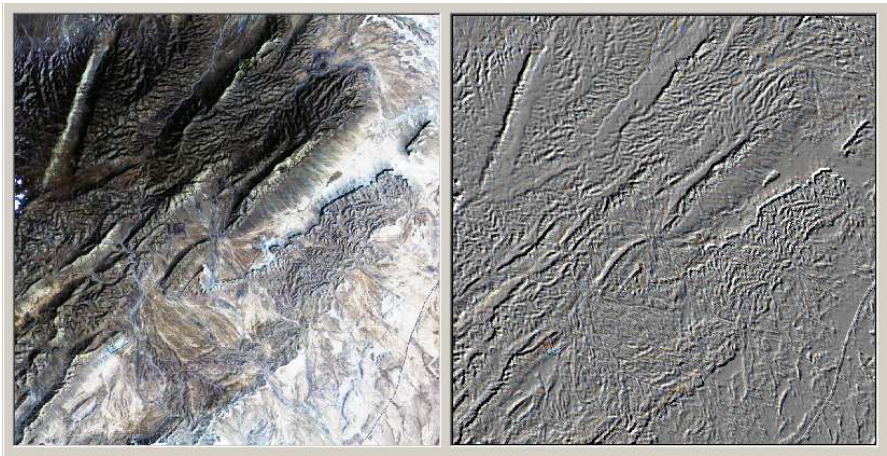
$$g_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\pi\sigma^2}}$$

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 egyenlő a jel és a Gauss-függvény deriváltjának a konvolúciójával. Ezek alapján az éldetektálás algoritmusa a következő.



41.32. ábra. A bal oldali kép az eredeti, a jobb oldali az emboss-szűrt változat..

ÉLDETEKTÁLÁS(f, g')

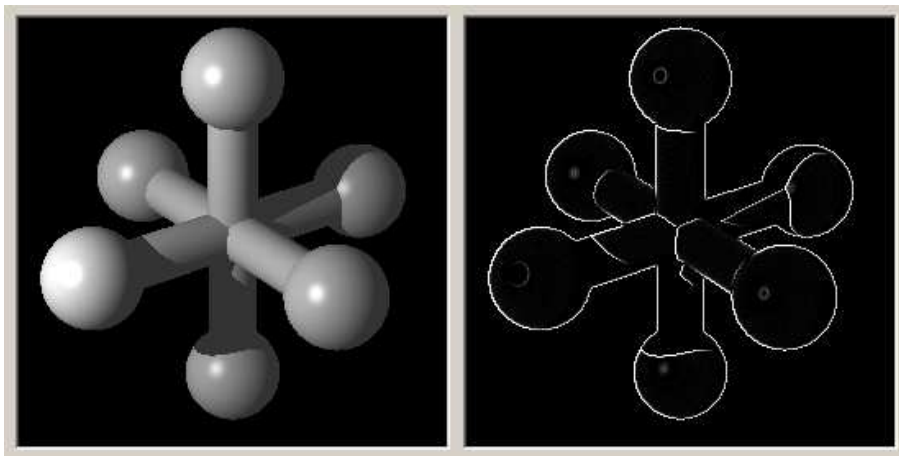
1 konvolváljuk f-et g'-vel
2 számítsuk ki h abszolút értékét
3 definiáljuk éleknek mindazokat a helyeket, ahol a h abszolút értéke meghalad egy előre meghatározott küszöbértéket

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 a 41.32. ábra.

Az éldetektálók egy másik, gyakran alkalmazott eljárása a ***Laplace-szűrés.*** Hatása hasonlít az emboss szűrőhöz, de nem tüntet ki irányokat.



41.33. ábra. A bal oldali az eredeti kép, a jobb oldali a Laplace-szűrő változata...

Mindenféle irányú éleket kiemel, 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 41.33. ábrán.

Érdekes összehasonlítani a kétféle éldetektor hatását a szintetikus képre. A 41.34. ábrán a szintetikus képet és az emboss-szűrt változatát láthatjuk.

Gyakorlatok

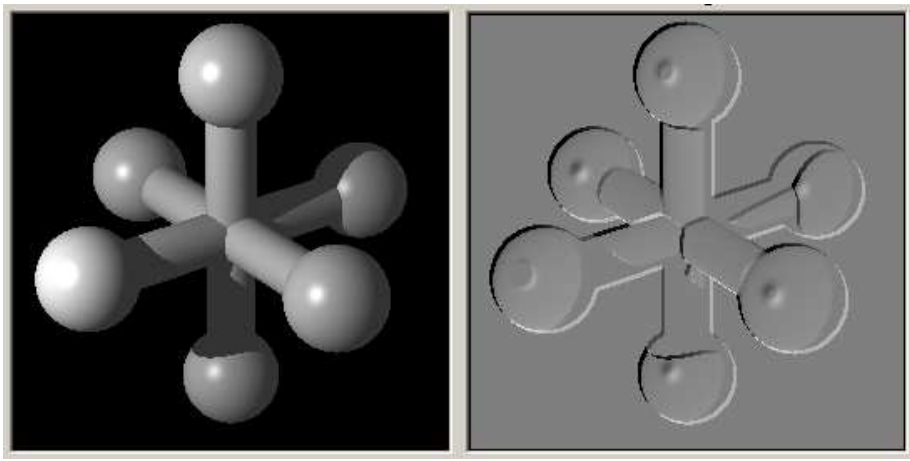
41.3-1. Bizonyítsuk be, hogy igaz a következő kifejezés:

$$(f * g)' = f * g' .$$

41.3-2. 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.

41.4. 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

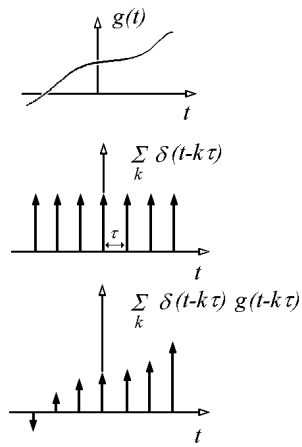


41.34. ábra. A bal oldali az eredeti kép, a jobb oldali az emboss-szűrt változat.

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 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.

41.4.1. Mintavételi tétel

Legyen τ az úgynevezett mintavételi távolság. Ábrázoljuk a $g(t)$ időfüggvényt és a mintavételezés eszközét, a Dirac-impulzusok sorozatát, majd



41.35. ábra. A mintavételezés eredménye egy olyan impulzus sorozat, melynek tagjai az eredeti függvénynek a mintavételi helyen felvett értékei...

a mintavételezés eredményét, a digitalizált időfüggvényt (41.36. 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

$$\tau \leq \frac{1}{2f_f} \ . \tag{41.1}$$

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 41.1. egyenlőtlenség. Ha túl kicsire választjuk a mintavételi távolságot, akkor feleslegesen sűrűn mintavételezett adatrendszert kapunk, ha viszont túl nagyra, mellyel nem áll fenn a 41.1. 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.

41.4.2. 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 41.1. 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 41.1. 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.

41.4.3. Két tipikus mintavételi probléma

A következőkben bemutatunk két, a térinformatikában tipikus mintavételezési feladatot, melyek megoldása során jól használhatók a mintavételről elmondottak.

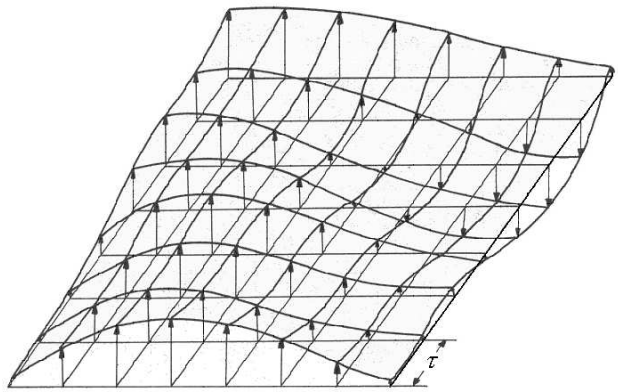
Digitális fénykép

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 (41.36. ábra).

A 41.37. ábrán egy valóságos domborzati modellt láthatunk, amelynek felbontóképessége körülbelül 5 méter, amivel a mintavételi tétel miatt 10



41.36. ábra. A τ rácsállandójú Dirac-δ sorozattal mintavételezett felszín...

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.

Gyakorlatok

41.4-1. Hogyan célszerű megválasztani a τ mintavételi távolságot?

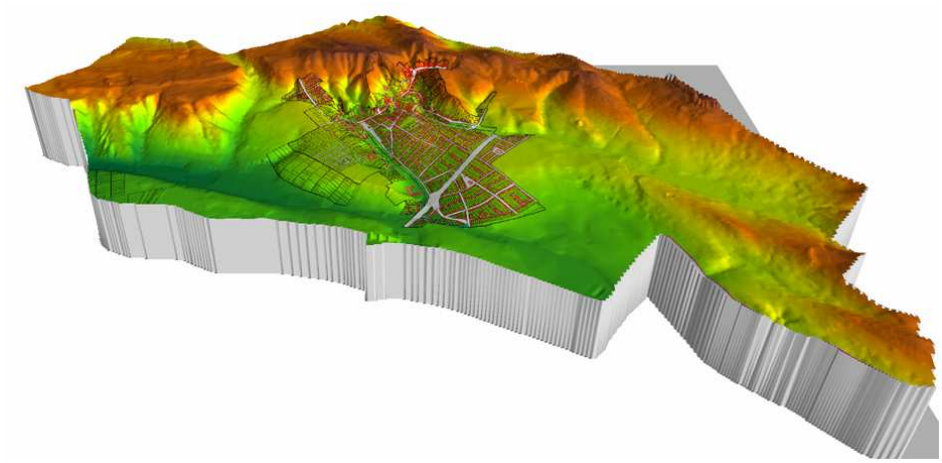
41.4-2. Elemezzük, hogy mit történik az adatrendszerrel és a spektrummal, ha τ_1 mintavételi távolságról áttérünk a τ_2 mintavételi távolságra, abban az esetben, ha

- $\tau_1 < \tau_2$ (ritkítás);
- $\tau_1 > \tau_2$ (sűrítés).

Feladatok

41-1 A négy-fa további műveletei

Az előzőekben felvázoltuk a pont-tartomány négy-fa egy lekérdező műveletét.



41.37. ábra. Egy település domborzati modelljének perspektivikus megjelenítése északi irányból...

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).

41-2 Konvolúció RGB színkódok esetében

A színes képek pontjainak RGB kódjait a KONVOLUCIO algoritmus összeadást és szorzást használó művelete nem megfelelően kezeli. Készítsük fel az algoritmust a színes képek kezelésére.

41-3 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.

41-4 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?

41-5 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 [226], valamint Maguire és társai [244] könyvei részletesen tárgyalják a különböző térinformatikai adatmodelleket. A centroid és a befoglaló négyszög fogalmát is bevezetik, aminek jelentőségét [303] több helyen ki is fejti. Hatvany Csaba [175] és Stephens [329] grafikus programozással foglalkozó műveikben részletesen foglalkoznak a grafikus adatok és a megjelenítés kérdéseivel. Rigaux és társai [303] elméleti alapművükben részletesen foglalkoznak a térbeli indexelés problematikájával és összevetik a [82] tankönyvben is részletesen tárgyalt B-fa működésével. [235] és [303] összehasonlítják a grid index és a rekurzív térfelosztási algoritmusok hatékonyságát. A négy-fa és más kapcsolódó adatszerkezetek részletes elemzésével foglalkozik Samet [315] cikke. [226], [235] és [303] 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 **??**. alfejezet.

[175] és [329] 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 [302] elméleti alapmű, amely széleskörű áttekintést nyújt az űrfotók, a távérzékelés problémaköréről az észleléstől a feldolgozáson át az értelmezésig. Duncan művéből [105] jó áttekintést kapunk a komplex függvénytanról, amely nélkül nem érthető a Fourier-analízis. [86] érthető formában vezeti be az Olvasót a disztribúcióelmélet fogalmaiba. Tárgyalja a Dirac-δ-t, a Heaviside-féle lépcsőfüggvényt és sok más érdekességet, amelyek nélkül a jelfeldolgozás és mintavételezés nehezen lenne megérthető. [82], [257] és [258] részletesen ismertetik a Fourier-transzformációt, sok példával, alapos elméleti áttekintéssel. Meskó Attila tankönyveiben [257, 258] 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 [99, 185], Kertész Ádám [210], Márkus Béla [266], valamint Márton Mátyás és társai [268] könyve.

42. 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* (42.1. alfejezet) és a *hibaelemzés* (42.2. alfejezet) alapjaira, a *lineáris algebra* (42.3. alfejezet) alapvető módszereire és a *matematikai szoftverekre, szoftverkönyvtárakra* (42.4. alfejezet) vonatkoznak.

42.1. Lebegőpontos aritmetika és hibaelemzés

42.1.1. 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 ismert, 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 hibakorlátú 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:

$$\begin{aligned}
\delta(a+b) &= \delta a + \delta b, \qquad \frac{\delta(a+b)}{|a+b|} = \max\left\{\frac{\delta a}{|a|}, \frac{\delta b}{|b|}\right\} \quad (ab>0)\ , \\
\delta(a-b) &= \delta a + \delta b, \qquad \frac{\delta(a-b)}{|a-b|} = \frac{\delta a + \delta b}{|a-b|} \quad (ab>0)\ , \\
\delta(ab) &\approx |a|\,\delta b + |b|\,\delta a, \qquad \frac{\delta(ab)}{|ab|} \approx \frac{\delta a}{|a|} + \frac{\delta b}{|b|} \quad (ab\neq 0)\ , \\
\delta(a/b) &\approx \frac{|a|\,\delta b + |b|\,\delta a}{|b|^2}, \qquad \frac{\delta(a/b)}{|a/b|} \approx \frac{\delta a}{|a|} + \frac{\delta b}{|b|} \quad (ab\neq 0)\,.
\end{aligned}$$

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.

42.1. 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. $4,84\times 10^2$-szeres. 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:

$$\delta\left(f\left(a\right)\right) \approx \left|f'(a)\right| \delta a, \quad f : \mathbb{R} \to \mathbb{R} \ ,$$
$$\delta\left(f\left(a\right)\right) \approx \sum_{i=1}^{n} \left|\frac{\partial f(a)}{\partial x_i}\right| \delta a_i, \quad f : \mathbb{R}^n \to \mathbb{R} \ .$$

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):

$$c(f,a) = \frac{|f'(a)|\,|a|}{|f(a)|}, \quad f : \mathbb{R} \to \mathbb{R} \ ,$$
$$c(F,a) = \frac{\|a\|\,\|F'(a)\|}{\|F(a)\|}, \quad F : \mathbb{R}^n \to \mathbb{R}^m \ .$$

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.

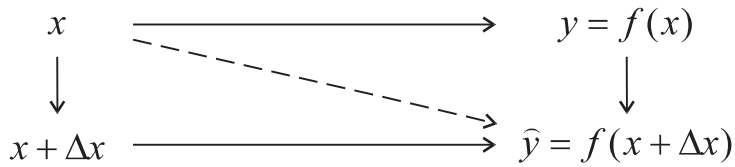
Mátrixok kondíciószámát is mint függvény kondíciószám felső korlátját vezethetjük be. Definiáljuk az $F : \mathbb{R}^n \to \mathbb{R}^n$ leképezést az $Ay = x$ egyenletrendszer megoldásával, azaz legyen $F(x) = A^{-1}x$ ($A \in \mathbb{R}^{n\times n}$, $\det(A) \neq 0$). Ekkor $F' \equiv A^{-1}$ és

$$c(F,a) = \frac{\|a\|\,\left\|A^{-1}\right\|}{\|A^{-1}a\|} = \frac{\|Ay\|\,\left\|A^{-1}\right\|}{\|y\|} \leq \|A\|\,\left\|A^{-1}\right\| \quad (Ay = a) \ .$$

A jobboldali felső korlátot az A ***mátrix kondíciószámának*** hívjuk. Ez a korlát pontos, mert létezik olyan $a \in \mathbb{R}^n$, hogy $c(F,a) = \|A\|\,\left\|A^{-1}\right\|$.

42.1.2. Direkt és inverz hibák

Vizsgáljuk egy $f\left(x\right)$ függvényérték kiszámítását. Ha az $\hat{y}$ közelítést számoljuk a pontos $y = f\left(x\right)$ érték helyett, akkor a ***direkt hiba*** $\Delta y = \hat{y} - y$. Ha egy



42.1. ábra. Direkt hiba és inverz hiba.

$x + \Delta x$ 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 42.1. 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

$$\frac{\delta \hat{y}}{|y|} \leq c(f, x) \frac{\delta \hat{x}}{|x|} \tag{42.1}$$

közelítő egyenlőtlenség fejezi ki. Szavakban megfogalmazva:

$$\text{relatív direkt hiba} \leq \text{kondíciószám} \times \text{relatív inverz hiba} \quad .$$

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.

42.2. példa. Vizsgáljuk az $f(x) = \log x$ függvényt. Ennek kondíciószáma $c(f, x) = c(x) = 1/|\log x|$, amely $x \approx 1$ esetén nagy. Tehát az $x \approx 1$ értékekre a relatív direkt hiba nagy lesz.

42.1.3. 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.

42.1. definíció. (a lebegőpontos számok halmaza).

$$F(\beta,t,L,U)=\left\{\pm m\times\beta^e \mid \frac{1}{\beta}\le m<1,\ m=0.d_1d_2\dots d_t,\ L\le e\le U\right\}\cup\{0\}\ , \tag{42.2}$$

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

$$m=0.d_1d_2\dots d_t=\frac{d_1}{\beta}+\frac{d_2}{\beta^2}+\dots+\frac{d_t}{\beta^t} \tag{42.3}$$

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 (42.3) felírást használva az $F=F(\beta,t,L,U)$ halmazt

megadhatjuk az

$$F = \left\{ \pm \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_t}{\beta^t} \right) \beta^e \mid \ L \le e \le U \right\} \cup \{0\} \tag{42.4}$$

alakban is, ahol $0 \le d_i \le \beta - 1$ $(i = 1, \ldots, t)$ és $1 \le d_1$.

42.3. 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 $m \in \left[1/\beta, 1 - 1/\beta^t\right]$, ugyanis

$$\frac{1}{\beta} \le m = \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_t}{\beta^t} \le \frac{\beta - 1}{\beta} + \frac{\beta - 1}{\beta^2} + \cdots + \frac{\beta - 1}{\beta^t} = 1 - \frac{1}{\beta^t} .$$

Fenti összefüggés segítségével könnyen igazolható az ábrázolható számok nagyságrendjét jellemző következő tétel.

42.2. 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

42.3. tétel. *Ha $x \in G$, akkor*

$$fl(x) = x(1+\varepsilon), \quad |\varepsilon| \le u \ .$$

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

$$m_1\beta^e \le x \le m_2\beta^e \ ,$$

és vagy $1/\beta \le m_1 < m_2 \le 1 - \beta^{-t}$, 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

$$|fl(x) - x| \le \frac{|m_2 - m_1|}{2}\beta^e = \frac{\beta^{e-t}}{2} \ .$$

Ebből következik

$$\frac{|fl(x) - x|}{|x|} \le \frac{|fl(x) - x|}{m_1\beta^e} \le \frac{\beta^{e-t}}{2m_1\beta^e} = \frac{\beta^{-t}}{2m_1} \le \frac{1}{2}\beta^{1-t} = u \ .$$

Fennáll tehát, hogy $fl(x) - x = \lambda x u$, ahol $|\lambda| \le 1$. Ezt átrendezve kapjuk, hogy

$$fl(x) = x(1 + \lambda u) \ .$$

Mivel $|\lambda u| \le u$, 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          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):

$$fl(a \square b) = (a \square b)\,(1+\varepsilon)\,, \quad |\varepsilon| \le u \quad (a, b \in F)\ . \tag{42.5}$$

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 (42.5) 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

$$\begin{array}{rccccccc} & 2 & \times & 0 & . & 1 & 0 & 0 \\ - & 2 & \times & 0 & . & 0 & 1 & 1 & 1 \end{array}\ .$$

Ha a számítást négy értékes jegyre végezzük, akkor az eredmény

$$\begin{array}{rccccccc} & 2^1 & \times & 0 & . & 1 & 0 & 0 \\ - & 2^1 & \times & 0 & . & 0 & 1 & 1 & 1 \\ \hline & 2^1 & \times & 0 & . & 0 & 0 & 0 & 1 \end{array}\ ,$$

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:

$$\begin{array}{rccccccc} & 2^1 & \times & 0 & . & 1 & 0 & 0 \\ - & 2^1 & \times & 0 & . & 0 & 1 & 1 \\ \hline & 2^1 & \times & 0 & . & 0 & 0 & 1 \end{array}\ ,$$

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

$$fl\,(x \pm y) = x\,(1+\alpha) \pm y\,(1+\beta)\,, \quad |\alpha|\,, |\beta| \le u\ , \tag{42.6}$$

$$fl\,(x \square y) = (x \square y)\,(1+\delta)\,, \quad |\delta| \le u, \quad \square = *, / \tag{42.7}$$

összefüggések teljesülnek.

Tegyük fel a továbbiakban, hogy van ellenőrző jegy a kivonásnál és teljesül a (42.5) feltevés. Vezessük be a következő jelöléseket:

$$|z| = [|z_1|, \ldots, |z_n|]^T \quad (z \in \mathbb{R}^n) \ , \tag{42.8}$$

$$|A| = [|a_{ij}|]_{i,j=1}^{m,n} \quad \left(A \in \mathbb{R}^{m\times n}\right) \ , \tag{42.9}$$

$$A \le B \Leftrightarrow a_{ij} \le b_{ij} \quad \left(A, B \in \mathbb{R}^{m\times n}\right) \ . \tag{42.10}$$

Igazolhatók az alábbi eredmények, ahol E az aktuális művelet hibáját (hibamátrixát) jelöli:

$$\left|fl\left(x^T y\right) - x^T y\right| \le 1.01nu\,|x|^T\,|y| \quad (nu \le 0.01) \ , \tag{42.11}$$

$$fl(\alpha A) = \alpha A + E \quad (|E| \le u\,|\alpha A|) \ , \tag{42.12}$$

$$fl(A + B) = (A + B) + E \quad (|E| \le u\,|A + B|) \ , \tag{42.13}$$

$$fl(AB) = AB + E \quad \left(|E| \le nu\,|A|\,|B| + O\left(u^2\right)\right) \ . \tag{42.14}$$

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.

42.4. 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        s = s + x_i
4  return s
```

42.5. 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       t = s
5       y = x_i + e
6       s = t + y
7       e (t − s) + y
8  return s
```

42.1.4. A lebegőpontos aritmetikai szabvány

Az ANSI/IEEE Std 754-1985 bináris ($\beta = 2$) *lebegőpontos 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 (42.5) 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

42.1-1. 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 hibakorlátait* 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:

a. a bemeneti adatoknak csak az abszolút korlátjait használva a δR_e-t és ezután a $\delta R_e/R_e$ relatív korlátot;

b. a bemeneti adatoknak csak a relatív korlátjait használva a $\delta R_e/R_e$ relatív korlátot és ezután a δR_e abszolút korlátot;

c. az eredő ellenállást $R_e = F(R_1, R_2)$ kétváltozós függvényként tekintve.

42.1-2. 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:

a. $1/\left(1+\sqrt{2}\right)^6$;

b. $99-70\sqrt{2}$.

Magyarázzuk is meg, miért.

42.1-3. Tekintsük az alapműveleteket kétváltozós függvényeknek, azaz $f(x,y)=x\square y$, ahol $\square$ a $+,-,*,/$ műveletek valamelyike.

a. Vezessük le az alapműveletek *hibakorlátjait,* mint kétváltozós függvényekét.

b. Adjuk meg ezen függvények *kondíciószámát.* Mikor rosszul kondicionáltak ezek a függvények?

c. 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.

d. 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.

42.1-4. 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 C konkrét értékének felhasználása nélkül.

42.1-5. Tekintsünk egy $F(\beta,t,L,U)$ *lebegőpontos számhalmazt.*

a. Mutassuk meg, hogy bármelyik aritmetikai alapművelet eredményezhet *aritmetikai túlcsordulást.*

b. Mutassuk meg, hogy bármelyik művelet okozhat alulcsordulást is.

42.1-6. Mutassuk meg, hogy a következő kifejezések *numerikusan instabilak* $x\approx 0$ esetén:

a. $(1-\cos x)/\sin^2 x$;

b. $\sin(100\pi+x)-\sin(x)$;

c. $2-\sin x-\cos x-e^{-x}$.

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.*

42.1-7. Hány eleme van az $F=F(\beta,t,L,U)$ halmaznak? Ezek között hány *szubnormális szám* található?

42.1-8. 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.

42.2. Lineáris egyenletrendszerek

A lineáris egyenletrendszerek általános alakja m egyenlet és n ismeretlen esetén:

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1j}x_j + \cdots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{i1}x_1 + \cdots + a_{ij}x_j + \cdots + a_{in}x_n &= b_i \\ &\vdots \\ a_{m1}x_1 + \cdots + a_{mj}x_j + \cdots + a_{mn}x_n &= b_m\,. \end{aligned} \tag{42.15}$$

Az egyenletrendszert megadhatjuk a tömörebb

$$Ax = b \tag{42.16}$$

formában is, ahol

$$A = [a_{ij}]_{i,j=1}^{m,n} \in \mathbb{R}^{m\times n},\ x \in \mathbb{R}^n,\ b \in \mathbb{R}^m\,.$$

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.

42.2.1. Lineáris egyenletrendszerek megoldásának közvetlen módszerei

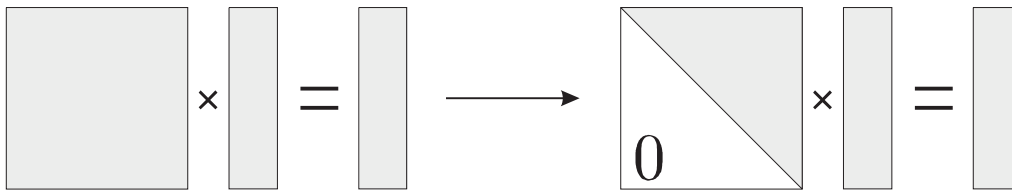
Ebben a pontban a Gauss- és Cholesky-módszert, valamint az LU-felbontást mutatjuk be.

Háromszögmátrixú egyenletrendszerek

42.4. 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:

$$\begin{bmatrix} * & * & \cdots & \cdots & * \\ 0 & * & & & \vdots \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & * & * \\ 0 & \cdots & \cdots & 0 & * \end{bmatrix}.$$



42.2. ábra. Gauss-elimináció.

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 42.2. ábra) két fázisból áll:

I. Azonos átalakításokkal az $Ax = b$ egyenletrendszert felső háromszög alakúra hozzuk. háromszög alakúra hozzuk. (A 42.2. 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 együtt-

hatómátrix főátló alatti elemeit 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      // {pivotálás esetén főelemkiválasztás, majd sor-, illetve oszlopcsere}
4      for i = k + 1 to n
5          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  // (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} $(k \leq j \leq n)$ 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

$$|a_{kk}| = \max_{k \leq i \leq n} |a_{ik}| \ .$$

Teljes főelemkiválasztás: A k-adik lépésben az a_{ij} $(k \leq i, j \leq n)$

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

$$|a_{kk}| = \max_{k \le i,j \le n} |a_{ij}| \ .$$

Megjegyezzük, hogy oszlopcsere esetén változócsere is történik.

Jól illusztrálja a pivotálás jelentőségét a következő

42.6. 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.

42.5. 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:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \quad (1 \le i \le n) \ .$$

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.

42.6. 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

42.7. 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.

42.8. 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.

42.9. definíció. *A négyzetes P mátrixot **permutációmátrixnak** nevezzük, ha minden sorában és oszlopában pontosan egy darab egyes 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

42.10. 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      határozzuk meg a t indexet,
       amelyre |A[P[t], k]| = max_{k≤i≤n} |A[P[i], k]|
5      if k < t
6          cseréljük fel a P[k] és a P[t] komponensek értékét
7          for i = k + 1 to n
8              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     s = 0
12     for j = 1 to i − 1
13         s = s + A[P[i], j] * x[j]
14         x[i] = b[P[i]] − s
15 for i = n downto 1
16     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

$$l_{kk} = (a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2)^{1/2},$$
$$l_{ik} = (a_{ik} - \sum_{j=1}^{k-1} l_{ij} l_{kj})/l_{kk} \quad (i = k+1, \ldots, n) \ .$$

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       a_kk = (a_kk − Σ_{j=1}^{k−1} a_kj^2)^{1/2}
4       for i = k + 1  to  n
5            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, amelynek együttható mátrixa sávmátrix.

42.11. definíció. *Az $A \in \mathbb{R}^{n \times n}$ mátrix **sávmátrix** p alsó sávszélességgel és q felső sávszélességgel, ha teljesül, hogy*

$$a_{ij} = 0, \quad \text{ha } i > j + p \text{ vagy } i + q < j \ .$$

A sávot, amelynek elemei lehetnek nemnullák, azon a_{ij} elemek definiálják, amelynek indexeire teljesül, hogy $i - p \leq j \leq i + q$, vagy ekvivalens módon

$j - q \leq i \leq j + p$. 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       for i = k + 1  to  min {k + p, n}
3            a_ik = a_ik/a_kk
4            for j = k + 1  to  min {k + q, n}
5                 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       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 $\quad b_i = \left(b_i - \sum_{j=i+1}^{\min\{i+q,n\}} u_{ij} b_j\right) / u_{ii}$
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** $i = 1$ **to** n
2 $\quad$ **for** $j = \max\{1, i-p\}$ **to** $i-1$
3 $\quad\quad a_{ij} = \left(a_{ij} - \sum_{k=\max\{1,i-p\}}^{j-1} a_{ik} a_{jk}\right) / a_{jj}$
4 $\quad\quad a_{ii} \left(a_{ii} - \sum_{k=\max\{1,i-p\}}^{i-1} a_{ik}^2\right)^{1/2}$
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.

42.2.2. 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

$$x_i = Gx_{i-1} + b \quad (i = 1, 2, \ldots)$$

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        i = i + 1
4        for l = 1 to L
5             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,

$$M = [M_{ij}]_{i,j=1}^{n}, \quad M_{ij} = \begin{cases} a_{ij}, \text{ ha } i,j \in S_l \text{ valamely } l \in \{1, \ldots, n\} \text{ értékre} \ , \\ 0 \quad \text{egyébként} \ . \end{cases}$$

Megmutatható, hogy a nemátfedő blokk Jacobi-multifelbontásárai 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

$$\widetilde{M}_l = \left[\widetilde{M}_{ij}^{(l)}\right]_{i,j=1}^{n}, \quad \widetilde{M}_{ij}^{(l)} = \begin{cases} a_{ij}, & \text{ha } i,j \in T_l \ , \\ a_{ii}, & \text{ha } i = j \ , \\ 0 & \text{egyébként} \quad , \end{cases}$$

$$\widetilde{N}_l = \widetilde{M}_l - A \ ,$$

$$\widetilde{e}_l = \left[\widetilde{e}_{ij}^{(l)}\right]_{i,j=1}^{n}, \quad E_{ii}^{(l)} = 0, \text{ ha } i \notin T_l$$

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

42.12. 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*

$$\rho\left(\widetilde{H}\right) \leq \rho\left(H\right) < 1 \ ,$$

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.

42.2.3. 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 $r = r\left(y\right) = Ay - b$ mennyiséget ***reziduális hibának*** nevezzük. Az elméleti megoldás esetén $r\left(x\right) = 0$, a közelítő megoldás esetén pedig

$$r\left(\hat{x}\right) = A\hat{x} - b = A\left(\hat{x} - x\right) = A\Delta x \ .$$

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, illetve 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

$$A\hat{x} = b + \Delta b \tag{42.17}$$

egyenletrendszert oldjuk meg. Legyen $\hat{x} = x + \Delta x$ és vizsgáljuk a két megoldás eltérését.

42.13. tétel. *Ha A nemszinguláris és $b \neq 0$, akkor*

$$\frac{\|\Delta x\|}{\|x\|} \leq \operatorname{cond}(A) \frac{\|\Delta b\|}{\|b\|} = \operatorname{cond}(A) \frac{\|r(\hat{x})\|}{\|b\|}, \tag{42.18}$$

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.

42.7. 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 42.13. 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.

42.8. 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

$$(A + \Delta A)\,\hat{x} = b \tag{42.19}$$

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.

42.14. 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.

42.15. 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*

$$\frac{\|r(\hat{x})\|}{\|b\|} \leq \frac{\operatorname{cond}(A)\frac{\|\Delta A\|}{\|A\|}}{1 - \operatorname{cond}(A)\frac{\|\Delta A\|}{\|A\|}}\,. \tag{42.20}$$

Ha A rosszul kondicionált, akkor a 42.15. tétel állítása nem teljesül.

42.9. 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

$$(A + \Delta A)\,\hat{x} = b + \Delta b \tag{42.21}$$

egyenletrendszert oldjuk meg. Igaz a következő

42.16. tétel. *Ha A nemszinguláris,* $\operatorname{cond}(A)\frac{\|\Delta A\|}{\|A\|} < 1$ *és* $b \neq 0$, *akkor*

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\operatorname{cond}(A)\left(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|}\right)}{1 - \operatorname{cond}(A)\frac{\|\Delta A\|}{\|A\|}} . \tag{42.22}$$

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 42.16. 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 $\|\Delta A\| < 1/\left\|A^{-1}\right\|$ 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:

$$\frac{1}{\operatorname{cond}(A)} = \min_{A+\Delta A \text{ szinguláris}} \frac{\|\Delta A\|}{\|A\|} . \tag{42.23}$$

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.

42.17. 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 42.13–42.16. tételeket összerakjuk, akkor a következő eredményt kapjuk.

42.18. 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 42.16. 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.

42.19. 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*

$$\|\hat{x} - x\|_\infty \leq \frac{\varepsilon \left\| \left|A^{-1}\right| (S\,|x| + s) \right\|_\infty}{1 - \varepsilon \left\| |A^{-1}|\, S \right\|_\infty} \ . \tag{42.24}$$

Ha $e = 0$ $(s = 0)$, $S = |A|$ és

$$k_r\,(A) = \left\| \left|A^{-1}\right| |A| \right\|_\infty < 1 \ , \tag{42.25}$$

akkor a

$$\|\hat{x} - x\|_\infty \leq \frac{\varepsilon k_r\,(A)}{1 - \varepsilon k_r\,(A)} \tag{42.26}$$

becslést kapjuk. A $k_r\,(A)$ mennyiséget ***Skeel-féle normának*** nevezzük, noha a szokásos értelemben nem norma. A Skeel-féle norma kielégíti a

$$k_r\,(A) \leq \mathrm{cond}_\infty\,(A) = \|A\|_\infty \left\|A^{-1}\right\|_\infty \tag{42.27}$$

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

$$D = \left\{ \Delta A \in \mathbb{R}^{n\times n} : |\Delta A| \leq \delta A \right\}, \quad G = \left\{ \Delta b \in \mathbb{R}^n : |\Delta b| \leq \delta b \right\} \ .$$

42.20. 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*

$$|r\,(\hat{x})| = |A\hat{x} - b| \leq \delta A\,|\hat{x}| + \delta b \ . \tag{42.28}$$

A tétel alkalmazásához nem kell a kondíciószám ismerete. A gyakorlatban δA és δb elemei a gépi epszilonnal arányosak.

42.21. tétel. (Wilkinson). *Az $Ax = b$ egyenletrendszer Gauss-módszerrel lebegőpontos aritmetikában kapott $\hat{x}$ közelítő megoldása kielégíti az*

$$(A + \Delta A)\,\hat{x} = b \tag{42.29}$$

egyenletrendszert, ahol

$$\|\Delta A\|_\infty \leq 8n^3\rho_n \|A\|_\infty u + O(u^2), \tag{42.30}$$

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 42.18. tétel alapján a -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 42.21. és a 42.16. tételek felhasználásával ($\Delta b = 0$ eset) a direkt hibára az alábbi közelítő becslést kapjuk:

$$\frac{\|\Delta x\|_\infty}{\|x\|_\infty} \leq 8n^3\rho_n \mathrm{cond}_\infty(A)\, u \ . \tag{42.31}$$

Ez az ökölszabály helyességét támasztja alá a Gauss-módszer esetén.

42.10. 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 $\mathrm{cond}\,(A)_\infty \approx 3,15 \times 10^{12}$, 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

$$H_n = \left[\frac{1}{i+j-1}\right]_{i,j=1}^{n} \tag{42.32}$$

Hilbert-mátrix, amelyre $\mathrm{cond}_2(H_n) \approx e^{3.5n}$, 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

$$(RAC)\, y = (Rb) \tag{42.33}$$

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

$$\hat{D} = \mathrm{diag}\left(\frac{1}{\left\|a_1^T\right\|_2}, \ldots, \frac{1}{\left\|a_n^T\right\|_2}\right)$$

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

$$\mathrm{cond}_2\left(\hat{D}A\right) \leq \sqrt{n} \min_{D \in D_+} \mathrm{cond}_2(DA) \ ,$$

ahol $D_+ = \{\mathrm{diag}(d_1, \ldots, d_n) \mid d_1, \ldots, d_n > 0\}$. 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.

42.11. 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 $\mathrm{cond}_\infty(A) = 12$. 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 $\mathrm{cond}_\infty(RAR) = 32/\epsilon$. 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

$$\tilde{A}x = (MA)\,x = Mb = \tilde{b} \tag{42.34}$$

alakba, ahol M olyan, hogy $\mathrm{cond}\left(M^{-1}A\right)$ 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

42.22. tétel. (Auchmuty). *Jelölje $\hat{x}$ az $Ax = b$ egyenletrendszer valamilyen módon kiszámított közelítő megoldását. Ekkor igaz, hogy*

$$\|x - \hat{x}\|_2 = \frac{c\,\|r(\hat{x})\|_2^2}{\|A^T r(\hat{x})\|_2},$$

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

$$C_2(A) = \frac{1}{2}\left(\operatorname{cond}_2(A) + \frac{1}{\operatorname{cond}_2(A)}\right) \leq \operatorname{cond}_2(A) \ .$$

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

$$\|x - \hat{x}\|_2 \lessapprox 0.5 \dim(A) \, \|r(\hat{x})\|_2^2 \, / \, \left\|A^T r(\hat{x})\right\|_2 \tag{42.35}$$

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:

$$\left\|A^{-1}\right\| \approx \frac{\|w\|}{\|y\|} \quad \left(\leq \left\|A^{-1}\right\|\right) \ . \tag{42.36}$$

Minthogy

$$\frac{\|w\|}{\|y\|} = \frac{\left\|A^{-1}\left(A^{-T}d\right)\right\|}{\|A^{-T}d\|} \ ,$$

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 $\operatorname{cond}(A)$ és a közelítő megoldás hibája már könnyen becsülhető (vö. a 42.16. 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.

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 = 1
2  x_1 = x̂
3  d̂ = ∞
4  while ‖d̂‖ / ‖x_k‖ > tol
5        r = Ax_k − b
6        számítsuk ki az Ad = r egyenletrendszer d̂
         közelítő megoldását LU-módszerrel
7        x_{k+1} = x_k − d̂
8        k = 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

$$(A + \Delta A)\hat{x} = b + \Delta b, \quad |\Delta A| \leq \eta |A| , \quad |\Delta b| \leq \eta |b| \ .$$

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

42.23. tétel. (Skeel). *Ha* $k_r\left(A^{-1}\right)\sigma\left(A,x\right) \leq c_1 < 1/\epsilon_M$, *akkor elég nagy* k *esetén fennáll, hogy*

$$(A+\Delta A)\,x_k = b+\Delta b, \quad |\Delta A| \leq 4\eta\epsilon_M\,|A|\,, \quad |\Delta b| \leq 4\eta\epsilon_M\,|b|\ . \tag{42.37}$$

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 $\|\hat{x}-x\| \leq q\,\|x\|$ $(q<1)$. Igazolták, hogy az iteratív javítás még egyszeres pontosságú lebegőpontos aritmetikában is javítja a közelítő megoldás pontosságát és a ϕ módszert gyengén stabillá teszi.

Gyakorlatok

42.2-1. Bizonyítsuk be a 42.7. tételt.

42.2-2. 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?

42.2-3. 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.

42.2-4. 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?

42.2-5. 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.

42.2-6. Tekintsük az $Ax = b$ egyenletrendszert, amelynek egy közelítő megoldása legyen $\hat{x}$.

a. Vezessünk le az $\hat{x}$ -ra hibakorlátot, ha az $(A+E)\hat{x} = b$ pontosan teljesül és sem A, sem $A + E$ nemszinguláris.

b. Adjunk (ha lehetséges) A elemeire ***relatív hibakorlátot***, melyen belül maradva nem változik a megoldás egyik komponensének sem az egész része, ha

$$A = \begin{bmatrix} 10 & 7 & 8 \\ 7 & 5 & 6 \\ 8 & 6 & 10 \end{bmatrix}, \quad b = \begin{bmatrix} 25 \\ 18 \\ 24 \end{bmatrix} .$$

42.3. 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.

42.24. 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*

$$Ax = \lambda x \; . \tag{42.38}$$

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

$$\phi(\lambda) = \det(A - \lambda I) = \det\left(\begin{bmatrix} a_{11} - \lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} - \lambda & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} - \lambda \end{bmatrix}\right) = 0 \ . \tag{42.39}$$

A (42.39) 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 (42.39) egyenletben. A különböző sajátértékekhez tartozó sajátvektorok lineárisan függetlenek.

A sajátértékek nagyságát és geometriai elhelyezkedését jellemzi a következő két tétel.

42.25. tétel. *Legyen λ az A mátrix tetszőleges sajátértéke. Tetszőleges indukált mátrixnormában fennáll, hogy $|\lambda| \leq \|A\|$.*

42.26. 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 (42.39) 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.

42.27. 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.*

42.28. 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.

42.29. 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.

42.12. 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 42.29. 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 $\approx 0,1051$. 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.

42.30. tétel. (Bauer-Fike). *Legyen $A \in \mathbf{C}^{n\times n}$ diagonalizálható mátrix, azaz létezzen olyan X mátrix, hogy $X^{-1}AX = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$. Jelölje μ az $A+\delta A$ mátrix sajátértékét. Ekkor*

$$\min_{1\le i\le n} |\lambda_i - \mu| \le \operatorname{cond}_2(X)\, \|\delta A\|_2 \ . \tag{42.40}$$

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 $\operatorname{cond}_2(X)$ 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.

42.13. 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.

42.14. 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.

42.31. 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.

42.3.1. 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 Miesestől 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        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 42.14 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

$$\|E_k\|_2 = \frac{\|r_k\|_2}{\left\|v^{(k)}\right\|_2} = \frac{\left\|Av^{(k)} - \gamma_k v^{(k)}\right\|_2}{\left\|v^{(k)}\right\|_2} \le \epsilon$$

kilépési feltétellel szokás leállítani. Ha a hatványmódszert szimultán alkalmazzuk az A^T mátrixra és $w_k = \left(A^T\right)^k w_0$, akkor

$$\nu\left(\lambda_1\right) \approx \frac{\left\|w^{(k)}\right\|_2 \left\|v^{(k)}\right\|_2}{\left|w_k^T v_k\right|}$$

a λ_1 kondíciószámának (lásd a 42.31. 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, illetve 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 negyedik sorát a következő utasításra cseréljük:

$$\text{oldjuk meg az } Az^{(k)} = v^{(k-1)} \text{ egyenletrendszert } z^{(k)}\text{-ra.}$$

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

$$|\lambda_t - \mu|^{-1} > |\lambda_i - \mu|^{-1} \quad (i \neq t)\ .$$

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:

42.32. definíció. *Egy $Q \in \mathbb{R}^{n\times n}$ mátrix **ortogonális,** ha $Q^TQ = I$.*

42.33. 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 $Ax = QRx = b \Leftrightarrow Rx = Q^Tb$ 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 $L\{a_1, \ldots, a_m\} = \left\{\sum_{j=1}^m \lambda_j a_j \mid \lambda_j \in \mathbb{R}, æ = 1, \ldots, m\right\}$ 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-Schmidteljárás 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 = 1 to m
2      for i = 1 to k − 1
3          r_ik = a_k^T a_i
4          a_k = a_k − r_ik a_i
5          r_kk = ‖a_k‖_2
6          a_k = a_k / r_kk
7  return a_1, ..., 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      r_kk = ‖a_k‖_2
3      a_k = a_k / r_kk
4      for j = k + 1 to m
5          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 $A_{k+1} = Q_k^{-1}A_kQ_k = Q_k^TA_kQ_k$ sorozatot, melynek alsó háromszögrésze diagonálmátrixhoz konvergál; főátlóbeli elemei tehát az A sajátértékeihez. A QR-módszernek nevezett eljárásban Q_k az $A_k = Q_kR_k$-felbontásában szereplő ortogonális tényező. Ekkor $A_{k+1} = Q_k^T(Q_kR_k)Q_k = R_kQ_k$.

QR-MÓDSZER(A)

1 $k = 1$
2 $A_1 = A$
3 **while** *kilépési feltétel* == HAMIS
4 számítsuk ki az $A_k = Q_kR_k$ felbontást
5 $A_{k+1} = R_kQ_k$
6 $k = k + 1$
7 **return** A_k

Igaz a következő tétel.

42.34. 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

$$\begin{bmatrix} \times & & & & & & & \times \\ 0 & \ddots & & & & & & \\ & & \times & & & & & \\ 0 & & 0 & * & \cdots & * & & \\ & & & \vdots & & \vdots & & \\ & & & * & \cdots & * & & \\ 0 & & & & & 0 & \times & \\ & & & & & & & \ddots \\ 0 & & & & & & 0 & \times \end{bmatrix} \tag{42.41}$$

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.

42.35. definíció. *Egy $A \in \mathbb{R}^{n\times n}$ mátrix **felső Hessenberg-alakú,** ha*

$$A = \begin{bmatrix} a_{11} & & & \cdots & & a_{1n} \\ a_{21} & & & & & \\ 0 & a_{32} & & & & \vdots \\ \vdots & 0 & \ddots & & & \\ & & \ddots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}.$$

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.

42.36. 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 $\lambda_1 - \sigma, \lambda_2 - \sigma, \ldots, \lambda_n - \sigma$, 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)

$H_1 = U^{-1}AU$ (H_1 felső Hessenberg-alakú)
$k = 1$
while *kilépési feltétel* = HAMIS
    számítsuk ki a $H_k - \sigma_k I = Q_k R_k$ felbontást
    $H_{k+1} = R_k Q_k + \sigma_k I$
    $k = k + 1$
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

42.3-1. 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?

42.3-2. 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.

42.3-3. Alkalmazzuk az *eltolásos* QR-módszert az előző gyakorlat mátrixára egy rögzített $\sigma_i = \sigma$ értékkel.

42.4. 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.

42.4.1. 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       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

$$A = A + xy^T \quad \left(A \in R^{m \times n},\ x \in R^m,\ y \in R^n\right)$$

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       A[i,:] = A[i,:] + x[i] y^T
4  return A
```

A „:” jelölés az összes megengedett indexet jelöli. Esetünkben az $1 \leq j \leq n$ 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       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

$$z = y + Ax \quad \left(x \in R^n,\ y \in R^m,\ A \in R^{m \times n}\right)$$

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      for j = 1 to n
7          for k = 1 to r
8              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      for k = 1 to r
7          for i = 1 to m
8              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 = textitsorok[A]
2  n = oszlopok[B]
3  C[1 : m, 1 : n] == 0
4  for j1 to n
5      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      for j = 1 to n
7          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.

42.4.2. 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, DERIVE és MATEMATICA nevű szoftvereket is.

A MATLAB

A közelítő megoldás iteratív javítása

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 függ-

vényt 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

42-1 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.

42-2 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.

42-3 Kétszeres szóhosszúság

Tekintsünk olyan kétszeres szóhosszúságú aritmetikai rendszert, amelyben minden $2t$ jeggyel ábrázolt szám két, 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.

a. Gondoljunk ki algoritmust két ilyen kétszeres szóhosszúságú szám összeadására, feltéve, hogy azok pozitívak.

b. Ha az ábrázolás megköveteli, hogy a számok mindkét felének legyen előjele, akkor módosítsuk az algoritmust úgy, hogy képes legyen mind a po-

zitív, mind a negatív számok helyes összeadására és az összeg mindkét részének azonos előjele legyen.
Feltehetjük, hogy a teljes összeg nem okoz túlcsordulást.

42-4 Auchmuty-tétel

Írjunk MATLAB programot az Auchmuty-féle (lásd 42.22. tétel) hibabecslésre és végezzük el a következő vizsgálatokat.

a. 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 $\|y_i\|_\infty \leq \beta$. 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 mennyiségeket grafikusan is ábrázoljuk. Javasolt értékek: $n \leq 200$, $\beta = 200$, $N = 40$.

b. Vizsgáljuk meg a kondíciószám és a méret hatását.

c. Végezzük el az a), b) feladatokat a LINPACK és BLAS programcsomagok segítségével.

42-5 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.

42-6 Konzisztens norma

Legyen $\|A\|$ konzisztens norma és tekintsük az $Ax = b$ egyenletrendszert.

a. Igazoljuk, hogy ha $A + \Delta A$ szinguláris, akkor $\mathrm{cond}(A) \geq \|A\| \,/\, \|\Delta A\|$.

b. Mutassuk meg, hogy „2”-es norma esetén (i)-ben az egyenlőség áll fenn, ha $\Delta A = -bx^T/(b^t x)$ és $\left\|A^{-1}\right\|_2 \|b\|_2 = \left\|A^{-1}b\right\|_2$.

c. Az (i)-t felhasználva adjunk alsó korlátot a $\mathrm{cond}_\infty(A)$-ra, ha

$$A = \begin{bmatrix} 1 & -1 & 1 \\ -1 & \varepsilon & \varepsilon \\ 1 & \varepsilon & \varepsilon \end{bmatrix} .$$

42-7 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 *Choleskyfelbontás* 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: $|f_{ij}| \le e_{ij}$, 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.*

42-8 Bauer–Fike-tétel

Legyen

$$A = \begin{bmatrix} 10 & 10 & & & & \\ & 9 & 10 & & & \\ & & 8 & 10 & & \\ & & & \ddots & \ddots & \\ & & & & 2 & 10 \\ \varepsilon & & & & & 1 \end{bmatrix} .$$

a. Vizsgáljuk meg a *sajátértékek* megváltozását az $\varepsilon = 10^{-5}, 10^{-6}, 10^{-7}, 0$ esetén.

b. Vizsgáljuk meg a Bauer-Fike tétel becslését az $A = A(0)$ mátrixhoz képest.

42-9 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: $N = 10$, $5 \leq n \leq 200$.
Hogyan alakulnak az eredmények a kondíciószám függvényében? Tapasztalunk-e függést az n rendszámtól? Ábrázoljuk grafikusan a maximális eltéréseket és a Baue-Fike-tétel becslését. r

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) [97]. 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 [332] könyvben is szerepel.

Az iterációs módszerek elméletének és alkalmazásainak kitűnő összefoglalását tartalmazzák Young [376], illetve Hageman és Young könyvei [164]. A téma szoftverelvű áttekintését adják Barett, Berry és társaik [30]. Itt hívjuk fel a figyelmet Andreas Frommer könyvére is, amely az iteratív módszerek párhuzamosítására is kitér [135]. A QR-módszer konvergenciájára vonatkozó 42.34. tételnél lényegesen jobb konvergencia eredmények is ismertek. Számos, a QR-módszerrel rokon eljárás ismert a sajátérték feladat megoldására (lásd például [364, 366]. 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 (42.41)-ben megmutatott alak – pontos számítások mellett, amint azt Batterson kimutatta [32]:

$$\begin{bmatrix} 0.83116322648071935 & -0.34924697025783983 & 0.06972435460743861 \\ 0.35613892213531548 & 0.86568478391818912 & 0.34924697025783983 \\ 0 & -0.35613892213531548 & 0.83116322648071935 \end{bmatrix}.$$

(a kerekítési hibák miatt, paradox módon, mégis tapasztalható igen lassú konvergencia).

Irodalomjegyzék

[1] S. Abiteboul. Querying semi-structured data. In F. Afrati, P. Kolaitis (szerk.), *Proceedings of ICDT'97* Lecture Notes in Computer Science 1186. kötete. Springer-Verlag, 1997, 1–18. 236

[2] S. Abiteboul, O. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM-Press, 1998, 254–263. 236

[3] S. Abiteboul, R. Hull, V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995. 334

[4] 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. 454

[5] P. Adriaans, D. Zantinge. *Data Mining*. Addison-Wesley Longman, 1996. Magyarul: *Adatbányászat*. Panem, Budapest, 2002. 145

[6] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, 307–328. o., 1996. 143

[7] R. Agrawal, R. Srikant. Fast algorithms for mining association rules. In J. B. Bocca, M. Jarke, Carlo Zaniolo (szerk.), *Proceedings of the 20th International Conference Very Large Data Bases, VLDB*, 487–499. o. Morgan Kaufmann Publishers, 1994. 143

[8] R. Agrawal, R. Srikant. Mining sequential patterns. In P. S. Yu, A. L. P. Chen (szerk.), *Proceedings of the 11th International Conference on Data Engineering, ICDE*, 3–14. o. IEEE Computer Society, 1995. 144

[9] R. Agrawal, T. Imielinski, A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman, S. Jajodia (szerk.), *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 207–216. o., 1993. 143

[10] A. Aho, C. Beeri, J. D. Ullman. The theory of joins in relational databases. *ACM Transactions on Database Systems*, 4(3):297–314, 1979. 334

[11] R. K. Ahuja, T. L. Magnanti, J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993. 488

[12] T. Akutsu. Dynamic programming algorithms for RNA secondary prediction with pseudoknots. *Discrete Applied Mathematics*, 104:45–62, 2000. 455

[13] 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. 453

[14] I. Althöfer. Improved game play by multiple computer hints. *Theoretical Computer Science*, 313:315–324, 2004. 489

[15] 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. 488

[16] I. Althöfer, J. de Koningand J. Lieberum, S. Meyer-Kahlen, T. Rolle, J. Sameith. Five visualisations of the k-best mode. *ICCA Journal*, 26:182–189, 2003. 489

[17] I. Althöfer. List-3-Hirn vs. grandmaster Yussupov – report on a very experimental match. *ICCA Journal*, 21:52–60 and 131–134, 1998. 489

[18] M. Anderberg. *Cluster Analysis for Applications*. Morgan Kaufmann Publishers, 1973. 177, 178

[19] AND Software GmbH. Car routing program "Route Germany", 1997. 488

[20] R. Angles, C. Gutierrez. The expressive power of SPARQL. In *The Semantic Web - ISWC 2008*. Springer, 114–129, 2008. 405

[21] M. Ankerst, M. Breunig, H. Kriegel, J. Sander. Optics: ordering points to identify the clustering structure. In *SIGMOD'99: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, 49–60. o., 1999. 178

[22] M. Arenas, L. Libkin. A normal form for XML documents. In *Proceedings of the 21st Symposium on Principles of Database Systems*, 85–96. o., 2002. 115

[23] V. L. Arlazano, E. A. Dinic, M. A. Kronrod, I. A. Faradzev. On economic construction of the transitive closure of a directed graph. *Doklady Academii Nauk SSSR*, 194:487–488, 1970. 453

[24] W. Armstrong. Dependency structures of database relationships. In *Proceedings of IFIP Congress*, 580–583. o., 1974. 333

[25] K. Atteson. The performance of the neighbor-joining method of phylogeny reconstruction. *Algorithmica*, 25(2/3):251–278, 1999. 456

[26] I. Bach. *Formális nyelvek.* Typotex, 2001. 452

[27] B. Baker. A tight asymptotic bound for next-fit decreasing bin-packing. *SIAM Journal on Algebraic and Discrete Methods*, 2(2):147–152, 1981. 294

[28] B. Balkenhol, S. Kurtz. Universal data compression based on the Burrows–Wheeler transform: theory and practice. *IEEE Transactions on Computers*, 49(10):1043–1053–953, 2000. 65

[29] W. Banzhaf. Interactive evolution. In T. Back, D. B. Fogel, Z. Michalewicz, T. Baeck Baeck (szerk.), *Handbook of Evolutionary Computation*. IOP Press, 1997. 489

[30] 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. 687

[31] C. F. Bates. *XML in Theory and Praxis*. John Wiley & Sons, 2003. Magyarul: *XML. Elmélet és gyakorlat*. Panem, 2004. 116

[32] S. Batterson. Convergence of the shifted QR algorithm on 3×3 normal matrices. , 58, 1990. 688

[33] 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. 333, 334

[34] C. Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Transactions on Database Systems*, 5:241–259, 1980. 334

[35] C. Beeri, M. Dowd. On the structure of armstrong relations for functional dependencies. *Journal of ACM*, 31(1):30–46, 1984. 333

[36] 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. 333

[37] C. Beeri, M. Y. Vardi. A proof procedure for data dependencies. *Journal of the ACM (JACM)*, 31(4):718–741, 1984. 405

[38] T. C. Bell, I. H. Witten, J. G. Cleary. Modeling for text compression. *Communications of the ACM*, 21:557–591, 1989. 65

[39] T. C. Bell, I. H. Witten, J. G. Cleary. *Text Compression*. Prentice Hall, 1990. 65, 66

[40] R. Bello, K. Dias, A. Downing, J. Freenan, T. Finnerty, W. D. Norcott, H. Sun, A. Witkowski, M. Ziauddin. Materialized views in Oracle. In *Proceedings of Very Large Data Bases'98*, 1998, 659–664. 236

[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. 334

[42] A. Benczúr, A. Kiss, T. Márkus. On a general class of data dependencies in the relational model and its implication problems. *Computers & Mathematics with Applications*, 21(1):1–11, 1991. http://www.sciencedirect.com/science/article/pii/089812219190226T. 405

[43] F. Berger. k alternatives instead of k shortest paths. Master's thesis, Friedrich Schiller Egyetem, Jéna, Matematikai és Informatikai Kar, 2000. Diplomamunka. 488

[44] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, 2002. 177

[45] P. Berkhin, J. D. Becher. Learning simple relations: Theory and applications. In *Proceedings of the Second SIAM International Conference on Data Mining*, 2002. 178

[46] T. Berners-Lee, J. Hendler, O. Lassila. The Semantic Web. *Scientific American*, 284(5):28–37, 2001. 405

[47] C. Bizer, A. Jentzsch, R. Cyganiak. State of the LOD Cloud. *Version 0.3 (September 2011)*, 2011. 405

[48] A. Békéssy, J. Demetrovics. Contribution to the theory of data base relations. *Discrete Mathematics*, 27(1):1–10, 1979. 333

[49] A. Békéssy, J. Demetrovics. *Előadások adatbázis szerkezetekről (Lectures on data base structures)*. ELTE Eötvös Kiadó, 1999. 334

[50] A. Békéssy, J. Demetrovics. *Adatbázis szerkezetek*. Akadémiai Kiadó, 2005. 334

[51] 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. 293

[52] L. A. Bélády. A study of replacement algorithms for a virtual storage computer. *IBM Systems Journal*, 5(2):78–101, 1965. 293

[53] J. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):135–256, 1982. 589, 590

[54] J. Bloomenthal. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers, 1997. 588

[55] F. Bodon. Adatbányászati algoritmusok. http://www.cs.bme.hu/ bodon/magyar/index.htm (elektronikus kézirat), 2005. 145

[56] F. Bodon. A fast APRIORI implementation. In B. Goethals, M. J. Zaki (szerk.), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* (FIMI'03), 2003. 144

[57] F. Bodon. Surprising results of trie-based fim algorithms. In B. Goethals, M. J. Zaki, R. Bayardo (szerk.), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* (FIMI'04, CEUR Workshop Proceedings, **126**), 2004. 144

[58] C. Borgelt. Efficient implementations of apriori and eclat. In B. Bart, M. J. Zaki (szerk.), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* (FIMI'03), 2003. 144

[59] J. Boyar, L. Epstein, A. Levin. Tight results for next fit and worst fit with resource augmentation. *Theoretical Computer Science*, 411(26–28):2572–2580, 2010. 294

[60] N. Bradley. *The XML Companion (harmadik kiadás)*. Addison-Wesley, 2004. Magyarul: *XML kézikönyv*. Szak, 2005. 116

[61] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965. 590

[62] A. Z. Broder. On the Resemblance and containment of documents. In *SEQS: Sequences '91*, 1998. 178

[63] P. Buneman. Semistructured data. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM-Press, 1997, 117–121. 236

[64] P. Buneman, M. Fernandez, D. Suciu. UnQL: a query language and algebra for semistructured data based on structural recursion. *The International Journal on Very Large Data Bases*, 9(1):76–110, 2000. 115

[65] D. Burdick, M. Calimlim, J. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the 17th International Conference on Data Engineering*, 443–452. o. IEEE Computer Society, 2001. 144, 145

[66] 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. 65

[67] Calgary. The Calgary/Canterbury Text Compression. 2004, ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression. corpus. 66

[68] D. Calvanese, D. De Giacomo, M. Lenzerini, M. Vardi. Answering regular path queries using views. In *Proceedings of the Sixteenth International Conference on Data Engineering*, 2000, 190–200. 236

[69] Canterbury. The Canterbury Corpus. http://corpus.canterbury.ac.nz, 2004. 66

[70] E. Catmull, J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350–355, 1978. 589

[71] S. Chaudhury, R. Krishnamurty, S. Potomianos, K. Shim. Optimizing queries with materialized views. In *Proceedings of the Eleventh International Conference on Data Engineering*, 1995, 190–200. 236

[72] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6(5):353–363, 1991. 589

[73] Q. Chen, A. Lim, K. W. Ong. An adaptive structural summary for graph-structured data. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, 134–144. o., 2003. 115

[74] D. Christie. Sorting permutations by block-interchanges. *Information Processing Letters*, 60(4):165–169, 1996. 456

[75] E. F. Codd. A relational model of large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. 293, 333

[76] E. F. Codd. Normalized database structure: A brief tutorial. In *ACM SIGFIDET Workshop on Data Description, Access and Control*, 24–30. o., 1971. 334

[77] E. F. Codd. Relational completeness of database sublanguages. In R. Rustin (szerk.), *Courant Computer Science Symposium 6: Data Base Systems*, 65–98. o. Prentice Hall, 1972. 293, 333, 334

[78] E. F. Codd. Recent investigations in relational data base systems. In *Information Processing 74*, 1017–1021. o. North-Holland, 1974. 334

[79] E. Coffman. *Computer and Job Shop Scheduling*. John Wiley & Sons, 1976. 293

[80] D. Comer, R. Sethi. The complexity of trie index construction. *Journal of the ACM*, 24(3):428–440, 1977. 143

[81] P. M. Consens, O. A. Mendelzon. GraphLog: a visual formalism for real life recursion. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 404–416. o. ACM, 1990. 358, 380

[82] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms*. The MIT Press/McGraw-Hill, 2014 (harmadik kiadás negyedik, javított utánnyomása. Magyarul: *Algoritmusok*. Műszaki Könyvkiadó Műszaki Kiadó, 1999, 2001 és 2003 (első angol kiadás fordítása), valamint HOUNTLER, 2015 (harmadik kiadás fordítása). 64, 294, 452, 589, 626, 627

[83] D. G. Corneil, C. Gotlieb. An efficient algorithm for graph isomorphism. *Journal of the ACM*, 17(1):51–64, 1970. 115

[84] T. M. Cover, J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, 1991. 66

[85] H. S. M. Coxeter. *Projective Geometry*. University of Toronto Press, 1974 (2. kiadás). 588

[86] R. Cristescu, G. Marinescu. *Unele aplicatii ale teoriei distributilor*. Editura academiei republicii socialiste, 1966. Magyarul: *Bevezetés a disztribúcióelméletbe és alkalmazásaiba*, Műszaki könyvkiadó, 1969). 627

[87] J. Csirik, Epstein, Cs. Imreh, A. Levin. On the sum minimization version of the online bin covering problem. *Discrete Applied Mathematics*, 158(13):1381–1393, 2010. 294

[88] I. Csiszár, J. Körner. *Coding Theorems for Discrete Memoryless Systems*. Akadémiai Kiadó, 1981 (Magyarul: *Információelmélet*. Tankönyvkiadó, 1980). 66

[89] M. de Berg. *Efficient Algorithms for Ray Shooting and Hidden Surface Removal*. PhD thesis, Rijksuniversiteit te Utrecht, The Nederlands, 1992. 589

[90] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2000. 589

[91] J. Dean, S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. 358, 380

[92] S. C. Deerwester, S. Dumais, T. K. Landauer, G. W. Furnas, R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990. 178

[93] C. Delobel. Normalization and hierarchical dependencies in the relational data model. *ACM Transactions on Database Systems*, 3(3):201–222, 1978. 333

[94] J. Demetrovics, Gy. O. H. Katona, A. Sali. Minimal representations of branching dependencies. *Discrete Applied Mathematics*, 40:139–153, 1992. 334

[95] J. Demetrovics, Gy. O. H. Katona, A. Sali. Minimal representations of branching dependencies. *Acta Scientiarum Mathematicorum (Szeged)*, 60:213–223, 1995. 334

[96] 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. 334

[97] J. Demmel, B. Diement, D. Malajovich. On the complexity of computing error bounds. *Foundations of Computational Mathematics*, 1:101–125, 2001. 687

[98] P. Denning. Virtual memory. *Computing Surveys*, 2(3):153–189, 1970. 293

[99] Á. Detrekői, Gy. Szabó. *Bevezetés a térinformatikába*. Nemzeti Tankönyvkiadó, 1995. 627

[100] A. Deutsch, M. Fernandez, D. Suciu. Storing semistructured data with stored. In *Proceedings of SIGMOD'99*, 1999, 431–442. 236

[101] A. Deutsch, L. Popa, D. Tannen. Physical data independence, constraints and optimization with universal plans. In *Proceedings of VLDB'99*, 1999, 459–470. 236

[102] A. Deutsch, L. Popa, V. Tannen. Query reformulation with constraints. *ACM SIGMOD Record*, 35(1):65–73, 2006. 405

[103] D. Doernberg. Interview with Donald Knuth. *Computer Literacy*, 1993. 456

[104] Gy. Dósa, J. Sgall. First Fit bin packing: A tight analysis. In N. Portier, T. Wilke (szerk.), *30th Symposium on Theoretical Aspects of Computer Science* STACS'13, 538–549. o. Schloss Dagstuhl, 2013. 294

[105] J. Duncan. *The Elements of Complex Analysis*. John Wiley & Sons, 1968 (Magyarul: Bevezetés a komplex függvénytanba, Műszaki Könyvkiadó, 1974). 627

[106] R. Durbin, S. Eddy, A. Krogh, G. Mitchison. *Biological Sequence Analysis*. University Press, 1998. 452

[107] O. Duschka, M. Genesereth. Query planning in infomaster. In *Proceedings of ACM Symposium on Applied Computing*. ACM-Press, 1997, 109–111. 236

[108] O. Duschka, M. Genesereth. Answering recursive queries using views. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM-Press, 1997, 109–116. 236

[109] N. Dyn, J. Gregory, D. Levin. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9:160–169, 1990. 589

[110] M. Effros, K. Viswesvariah, S. Kulkarni, S. Verdú. Universal lossless source coding with the burrows–wheeler transform. *IEEE Transactions on Information Theory*, 48(5):1061–1081, 2002. 65

[111] L. Epstein, Cs. Imreh, R. vanStee. More on weighted servers or FIFO is better than LRU. *Theoretical Computer Science*, 306:305–317, 2003. 293

[112] L. Epstein, Cs. Imreh, A. Levin. Class constrained bin packing revisited. *Theory of Computing Systems*, 46:246–260, 2010. 294

[113] L. Epstein, Cs. Imreh, A. Levin. Bin covering with cardinality constraints. *Discrete Applied Mathematics*, 161(13–14):1975–1987, 2013. 294

[114] L. Epstein, Y. Kleiman, J., R.. Paging with connections: FIFO strikes again. *Theoretical Computer Science*, 1–3:55–64, 2007. 293

[115] L. Epstein andA. Levin. Bin packing with general cost structures. *Discrete Applied Mathematics*, 132(1–2):355–391, 2012. 294

[116] 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. 456

[117] M. Ester, H. Kriegel, J. Sander, X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, 226–231. o. AAAI Press, 1996. 178

[118] R. Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems*, 2:262–278, 1977. 333, 334

[119] R. Fagin. Armstrong databases. In *Proceedings of IBM Symposium on Mathematical Foundations of Computer Science*, 1982. 333

[120] R. Fagin. Horn clauses and database dependencies. *Journal of ACM*, 29(4):952–985, 1982. 333

[121] R. Fagin, P. G. Koloatis, L. Popa, W-C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Transactions on Database Systems (TODS)*, 30(4):994–1055, 2005. 358, 380

[122] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Morgan Kaufmann Publishers, 1998. 588

[123] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., 2003. 452

[124] R. Fernando. *GPUGems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*. Addison-Wesley, 2004. 590

[125] P. Flach. *Simply Logical: Intelligent Reasoning by Example (Wiley Series in Probability and Mathematical Statistics)*. Wiley & Sons, 1994 (magyarul: *Logikai programozás*, Panem, 2001). 236

[126] D. Florescu, A. Halevy, A. O. Mendelzon. Database techniques for the world-wide web: a survey. *SIGMOD Record*, 27(3):59–74, 1998. 236

[127] D. Florescu, L. Raschid, P. Valduriez. Answering queries using oql view expressions. In *Workshop on Materialized views, in cooperation with ACM SIGMOD*, 1996, 627–638. 236

[128] D. D. Florescu. *Search spaces for object-oriented query optimization*. PhD thesis, University of Paris VI, 1996. 236

[129] F. Floris, B. Goethals, J. Bussche. A tight upper bound on the number of candidate patterns. In *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM '01)*, 155–162. o. IEEE Computer Society, 2001. 143

[130] Z. Fülöp. *Formális nyelvek és szintaktikus elemzésük*. Polygon, 2004 (második kiadás). 452

[131] J. D. Fooley, A., S. K. Feiner, J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1990. 590

[132] P. Fornai, A. Iványi. Bélády's anomaly is unbounded. In Kovács Előd és Winkler Zoltán (szerk.), *5th International Conference on Applied Informatics*, 65–72. o. Molnár és társa, 2002. 293

[133] P. Fornai, A. Iványi. Bélády's anomaly is unbounded. *Acta Universitatis Sapientiae, Informatica*, 2(1):80–89, 2010. 293

[134] M. Friedman, D. S. Weld. Efficient execution of information gathering plans. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1997, 785–791. 236

[135] A. Frommer. *Lösung linearer Gleichungssysteme auf Parallelrechnern*. Vieweg Verlag, 1990. 687

[136] 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. 590

[137] A. Fujimoto, T. Takayuki, I. Kansey. ARTS: accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, 1986. 589

[138] S. Gajdos. *Adatbázisok*. Műegyetemi Kiadó, 2001. 334

[139] T. Gal, T. Stewart, T. Hanne (szerk.). *Multicriteria Decision Making*. Kluwer Academic Publisher, 1999. 489

[140] G. Galambos. *Operációs rendszerek*. Műszaki Könyvkiadó, 2003. 294

[141] H. Garcia-Molina, J. Ullman. *Database System Implementation and J. Widom*. Prentice Hall, 2000. Magyarul: *Adatbázisrendszerek megvalósítása*. Panem, 2001. 334

[142] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 115

[143] W. H. Gates, C. H. Papadimitriou. Bounds for sorting by prefix reversals. *Discrete Mathematics*, 27:47–57, 1979. 456

[144] F. Gécseg, I. Peák. *Algebraic Theory of Automata*. Akadémiai Kiadó, 1972. 293

[145] A. Glassner. *An Introduction to Ray Tracing*. Academic Press, 1989. 589

[146] B. Goethals, M. J. Zaki. Advances in frequent itemset mining implementations: Introduction to fimi03. In B. Goethals, M. J. Zaki (szerk.), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* (FIMI'03, CEUR Workshop Proceedings, **90**), 2003. 144

[147] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989. 489

[148] 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. 454

[149] J. Goldstein, P. A. Larson. Optimizing queries using materialized views: a practical, scalable solution. In *Optimizing queries using materialized views: a practical, scalable solution*, 2001, 331–342. 236

[150] G. H. Golub, C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1996 (3. kiadás). 177

[151] G. Gombos, T. Matuszka, B. Pinczel, G. Rácz, A. Kiss. VOSD: A general-purpose virtual observatory over semantic databases. In Á. Kiss (szerk.), *13th Symposium on Programming Languages and Software Tools* (SPLST 2013, Szeged, 2013. aug. 26–aug. 27). Szegedi Egyetem Informatikai Tanszékcsoport, 90–99, 2013. 358, 381

[152] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982. 452

[153] T. Gottdank. *Szemantikus web. Bevezetés a tudásalapú internet világába*. Computer Books, 2005. 358, 380

[154] G. Gottlob, C. Koch, R. Pichler. The complexity of XPath query evaluation. In *Proceedings of the 22nd Symposium on Principles of Database Systems*, 179–190. o., 2003. 115

[155] H. Gouraud. Computer display of curved surfaces. *IEEE Transactions on Computers*, C-20(6):623–629, 1971. 583

[156] G. Grahne, A. Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proceedings of ICDT'99* Lecture Notes in Computer Science 1540. kötete. Springer-Verlag, 1999, 332–347. 236

[157] G. Grahne, J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In B. Goethals, M. J. Zaki (szerk.), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* (FIMI'03), 2003. 144

[158] J. Grant, J. Minker. Inferences for numerical dependencies. *Theoretical Computer Science*, 41:271–287, 1985. 334

[159] J. Grant, J. Minker. Normalization and axiomatization for numerical dependencies. *Information and Control*, 65:1–17, 1985. 334

[160] A. Grosse, S. Schwarz. Bigblackcell. http://www.minet.uni-jena.de/~BigBlackCell/. 488

[161] S. Guha, R. Rastogi, K. Shim. ROCK: A robust Clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000. 178

[162] D. M. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997. 452

[163] L. Györfi, S. Győri, I. Vajda. *Információ- és kódelmélet (Theory of Information and Coding)*. Typotex, 2002 (2. kiadás). 66

[164] L. Hageman, D. Young. *Applied Iterative Methods*. Academic Press, 1981. 687

[165] Gy. F. Hajós. *Bevezetés a geometriába*. Tankönyvkiadó, 1972. 588

[166] A. Halevy. Logic based techniques in data integration. In J. Minker (szerk.), *Logic-based Artificial Intelligence*. Kluwer Academic Publishers, 2000, 575–595. 236

[167] A. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10:270–294, 2001. 236

[168] T. S. Han, K. Kobayashi. *Mathematics of Information and Coding*. American Mathematical Society, 2002. 66

[169] J. Han, M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufman Publisher, 1972 (magyararul: *Adatbányászat – Koncepciók és technikák*. Panem, 2004). 144, 145

[170] J. Han, M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufman Publisher, 1972 (magyararul: *Adatbányászat – Koncepciók és technikák*. Panem, 2004). 177

[171] D. Hankerson, G. A. Harris, P. D. Johnson. *Introduction to Information Theory and Data Compression*. CRC Press, 2003 (2. kiadás). 66

[172] S. Hannenhalli. Polynomial-time algorithm for computing translocation distance between genomes. *Discrete Applied Mathematics*, 71:137–151, 1996. 456

[173] S. Harris, A. Seaborne. SPARQL 1.1 query language. W3C recommendation, March 2013. 358, 380

[174] J. A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, 1975. 177, 178

[175] B. Cs. Hatvany. *Bitképek feldolgozása Visual Basic programokból*. Computer Books, 2003. 626

[176] T. H. Haveliwala, A., D. Klein, P. Indyk. Evaluating strategies for Similarity search on the web. In *Proceedings of the Eleventh International World Wide Web Conference, 2002.*, 2002. 178

[177] V. Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Czech Technical University, 2001. 590

[178] 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. 456

[179] H. He, J. Yang. Multiresolution indexing of XML for frequent queries. In *Proceedings of the 20th International Conference on Data Engineering*, 683–694. o., 2004. 115

[180] E. A. Heinz. *Algorithmic Enhancements and Experiments at High Search Depths*. Vieweg Verlag, Series on Computational Intelligence, 2000. 489

[181] M. R. Henzinger, T. A. Henzinger, P. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 453–462. o. IEEE Computer Society Press, 1995. 115

[182] I. Herman. *The Use of Projective Geometry in Computer Graphics*. Springer-Verlag, 1991. 588

[183] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18:341–343, 1975. 452

[184] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation* második kiadás. Addison-Wesley, 2001 (németül: *Einführung in Automatentheorie, Formale Sprachen und Komplexitätstheorie*, Pearson Studium, 2002). 293

[185] Á. http://hu.wikipedia.org/wiki/Detrek%C5%91i_%C3%81kos Gy. Szabó. *Térinformatika*. Nemzeti Tankönyvkiadó, 2003 (második kiadás). 627

[186] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. 64

[187] J. Hunt, T. Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, 1977. 453

[188] M. Husain, P. Doshi, L. Khan, J. McGlothlin. Efficient query processing for large rdf graphs using hadoop and mapreduce. Technical report, University of Texas at Dallas, 2009. 358, 380

[189] A. Inokuchi, T. Washio, H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, 13–23. o. Springer-Verlag, 2000. 144

[190] A. Iványi. On dumpling-eating giants. In *Finite and Infinite Sets* (Eger, 1981), olloquia of Mathematical Society János Bolyai, **37**, 379–390. o. North-Holland, 1984. 294

[191] A. Iványi. Performance bounds for simple bin packing algorithms. *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica*, 5:77–82, 1984. 294

[192] A. Iványi. Tight worst-case bounds for bin packing algorithms. In *Theory of Algorithms* (Pécs, 1984, Colloquia of Mathematical Society János Bolyai, **44**), 233–240. o. North-Holland, 1985. 294

[193] A. Iványi, R. Szmeljánszkij. *Elements of Theoretical Programming* (*in Russian*). Moszkvai Állami Egyetem, 1985. 293

[194] 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*, 2149:69–84, 2001. 455

[195] A. Jain, M. Murty, P. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999. 177

[196] A. K. Jain, R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988. 177

[197] M. R. Jerrum. The complexity of finding minimum-length generator sequences. *Theoretical Computer Science*, 36:265–289, 1985. 456

[198] P. Jiménez, F. Thomas, C. Torras. 3D collision detection: A survey. *Computers and Graphics*, 25(2):269–285, 2001. 589

[199] D. Johnson. Near-optimal bin packing algorithms. PhD értekezés, MIT Department of Mathematics, 1973. 294

[200] 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. 294

[201] S. Karlin, M. T. Taylor. *A First Course in Stochastic Processes.* Academic Press, 1975 (magyarul: *Sztochasztikus folyamatok*, Gondolat Kiadó, 1985). 590

[202] L. Kaufman, P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, 1990 (2. kiadás). 177, 178

[203] R. Kaushik, R. Krishnamurthy, J. F. Naughton, R. Ramakrishnan. Exploiting local similarity for indexing paths in graph-structured data. In *Proceedings of the 18th International Conference on Data Engineering*, 129–140. o., 2002. 115

[204] R. Kaushik, R. Shenoy, P. F. Bohannon, E. Gudes. On the integration of structure indexes and inverted lists. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 779–790. o., 2004. 116

[205] R. R. Kaushik, P. Bohannon, J. F. Naughton, H. Korth. Covering indexes for branching path queries. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, 133–144. o., 2002. 115

[206] R. R. Kaushik, P. Bohannon, J. F. Naughton, H. Korth, P. Shenoy. Updates for structure indexes. In *Proceedings of Very Large Data Bases*, 239–250. o., 2002. 115

[207] A. Kóczy, K. Kondorosi. *Operációs rendszerek – mérnöki megközelítésben (magyarul: Operating Systems - An Engineering Approach).* Panem, 1999. 294

[208] 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. 453

[209] B. Kernighan, R. Pike. *The UNIX programming Environment.* Prentice Hall, 1984 (Magyarul: *A UNIX operációs rendszer*, Műszaki Könyvkiadó, 1987 és 1999). 294

[210] A. Kertész. *A térinformatika és alkalmazásai (Geoinformatics and it Applications).* Holnap Kiadó, 1997. 627

[211] M. Khosrow-Pour (szerk.). *Encyclopedia of Information Science and Technology, Vol. 1, Vol. 2, Vol. 3, Vol. 4, Vol. 5.* Idea Group Inc., 2005. 116, 145, 177

[212] J. Kleinberg. An impossibility theorem for clustering. In S. Becker, S. Thrun, K. Obermayer (szerk.), *Proceedings of 15th Conference Neural Information Processing Systems* (Advances in Neural Information Processing Systems, **15**), 2002. 177

[213] G. Klyne, J. Carroll, B. McBridge. Resource Description Framework RDF: concepts and abstract syntax, 2004. W3C recommendation 10 February 2004. 405

[214] B. Knudsen, J. Hein. Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Researchs*, 31(13):3423–3428, 2003. 454

[215] D. E. Knuth, J. H. Morris Jr., V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977. 453

[216] G. Krammer. Notes on the mathematics of the PHIGS output pipeline. *Computer Graphics Forum*, 8(8):219–226, 1989. 588

[217] M. Kuramochi, G. Karypis. Frequent subgraph discovery. In *Proceedings of the 1st IEEE International Conference on Data Mining*, 313–320. o., 2001. 144

[218] 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 (szerk.), *Numbers, Information and Complexity*, 375–383. o. Kluwer Academic Publishers, 2000. 65

[219] C. T. Kwok, D. Weld. Planning to gather information. In *Proceedings of AAAI 13th National Conference on Artificial Intelligence*, 1996, 32–39. 236

[220] T. Lai, S. Sahni. Anomalies in parallel branch and bound algorithms. *Communications of ACM*, 27(6):594–602, 1984. 293

[221] E. T. Lambrecht, S. Kambhampati, S. Gnanaprakasam. Optimizing recursive information gathering plans. In *Proceedings of 16th International Joint Conference on Artificial Intelligence*, 1999, 1204–1211. 236

[222] J. Lamperti. *Stochastic Processes.* Springer-Verlag, 1972. 590

[223] G. Lancia. Integer programming models for computational biology problems. *Journal of Computer Science and Technology*, 19(1):60–77, 2004. 453

[224] G. Landau, U. Vishkin. Eficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986. 453

[225] G. Langdon. An introduction to arithmetic coding. *IBM Journal of Research and Development*, 28:135–149, 1984. 65

[226] R. Laurini, D. Thompson. *Fundamentals of Spatial Information Systems*. Academic Press, 1992. 626

[227] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computer Statistical Data Analysis*, 19(2):191–201, 1995. 178

[228] A. Levy, A. Rajaraman, J. J. Ordille, D. Srivastava. Querying heterogeneous information sources using source descriptions. In *Proceedings of Very Large Data Bases*, 1996, 251–262. 236

[229] A. Y. Levy, A. Mendelzon, Y. Sagiv, D. Srivastava. Answering queries using views. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM-Press, 1995, 95–104. 236

[230] T. Linder, G. Lugosi. *Bevezetés az információelméletbe (Introduction to Information Theory)*. Tankönyvkiadó, 1990. 66

[231] V. Liptchinsky, B. Satzger, R. Zabolotnyi, S. Dustdar. Expressive languages for selecting groups from graph-structured data. In *Proceedings of the 22nd international conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 761–770, 2013. 358, 380

[232] G. Liu, H. Lu, J. X. Yu, W. Wang, X. Xiao. AFOPT: An efficient implementation of pattern growth approach. In B. Goethals, M. J. Zaki (szerk.), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* (FIMI'03), CEUR Workshop Proceedings, 2003. 144

[233] L. Liu, J. Yin, L. Gao. Efficient social network data query processing on MapReduce. In *Proceedings of the 5th ACM HotPlanet Workshop*, 27–32, 2013. 358, 381

[234] A. Álmos, S. Győri, G. Horváth, A. Várkonyiné Kóczy. *Genetikus algoritmusok (Genetic Algorithms)*. Typotex, 2002. 489

[235] P. Longley, D. Maguire, M. Goodschild, D. Rhind. *Geographic Information Systems and Science*. John Wiley & Sons, 2001. 626

[236] L. Lovász. *Combinatorial Problems and Exercises*. Akadémiai Kiadó, 1979 (Magyarul: *Kombinatorikai problémák és feladatok*, Typotex, 1999, magyarul digitálisan Typotex, 2005). 177

[237] L. Lovász, P. Gács. *Algoritmusok*. Műszaki Könyvkiadó és Tankönyvkiadó, 1978. 14

[238] C. Lucchesi, S. . Candidate keys for relations. *Journal of of Computer and System Sciences*, 17(2):270–279, 1978. 334

[239] 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. 455

[240] G. Lunter, I. Miklós, Y. Song, J. Hein. An efficient algorithm for statistical multiple alignment on arbitrary phylogenetic trees. *Journal of Computational Biology*, 10(6):869–889, 2003. 455

[241] R. Lyngso, C. N. S. Pedersen. RNA pseudoknot prediction in energy based models. *Journal of Comp. Biology*, 7(3/4):409–428, 2000. 455

[242] R. Lyngso, M. Zuker, C. Pedersen. Fast evaluation of internal loops in RNA secondary structure prediction. *Bioinformatics*, 15(6):440–445, 1999. 455

[243] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967. 178

[244] D. J. Maguire, M. Goodschild, D. Rhind. *Geographical Information Systems*. Longman, 1991. 626

[245] D. Maier. Minimum covers in the relational database model. *Journal of the ACM*, 27(4):664–674, 1980. 333

[246] D. Maier, A. O. Mendelzon, Y. Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4(4):455–469, 1979. 333

[247] H. Mannila, H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, 146–151. o. AAAI Press, 1996. 144

[248] H. Mannila, H. Toivonen, I. Verkamo. Efficient algorithms for discovering association rules. In U. M. Fayyad, R. Uthurusamy (szerk.), *AAAI Workshop on Knowledge Discovery in Databases(KDD-94)*, 181–192. o. AAAI Press, 1994. 143

[249] H. Mannila, H. Toivonen, I. Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining (KDD'95)*, 210–215. o. AAAI Press, 1995. 144

[250] R. L. Mattson, J. Gecsei, D. R. Slutz, I. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970. 293

[251] T. Matuszka. Augmented reality supported by Semantic Web technologies. In P. Cimiano, O. Corcho, V. Presutti, L. Hollink, S. Rudolph (szerk.), *The Semantic Web: Semantics and Big Data (Lecture Notes in Computer Science)*. Springer, **7882**, 682–686, 2013. 358, 381

[252] T. Matuszka, G. Gombos, A. Kiss. A new approach for indoor navigation using semantic webtechnologies and Augmented Reality. In *Virtual Augmented and Mixed Reality. Designing and Developing Augmented and Virtual Environments*. Springer, 202–210, 2013. 381

[253] E. A. Maxwell. *Methods of Plane Projective Geometry Based on the Use of General Homogenous Coordinates*. Cambridge University Press, 1946. 588

[254] E. A. Maxwell. *General Homogenous Coordinates in Space of Three Dimensions*. Cambridge University Press, 1951. 588

[255] J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29:1105–1119, 1990. 455

[256] B. McLaughlin. *Java and XML*. O'Reilly, 2000. Magyarul: *XML kézikönyv*. Szak, 2005. 116

[257] A. Meskó. *A digitális szeizmikus feldolgozás alapjai (Introduction to Processing of Digital Seismic Data)*. Tankönyvkiadó, 1978. 627

[258] A. Meskó. *Geofizikai adatfeldolgozás (Processing of Geophysical Data)*. Tankönyvkiadó, 1983. 627

[259] I. Meyer, R. Durbin. Comparative ab initio prediction of gene structures using pair HMMs. *Bioinformatics*, 18(10):1309–1318, 2002. 452, 454

[260] I. M. Meyer, R. Durbin. Gene structure conservation aids similarity based gene prediction. *Nucleic Acids Research*, 32(2):776–783, 2004. 454

[261] Sz. Mihnovszkij, N. Shor. Estimation of the page fault number in paged memory (in russian). *Kibernetika* (Kiev), 1(5):18–20, 1965. 293

[262] W. Miller, E. Myers. A file comparison program. *Software – Practice and Experience*, 15(11):1025–1040, 1985. 453

[263] W. Miller, E. W. Myers. Sequence comparison with concave weighting functions. *Bulletin of Mathematical Biology*, 50:97–120, 1988. 452

[264] T. Milo, D. Suciu. Index structures for path expressions. In *Lecture Notes in Computer Science*, ***1540***, 277–295. o. Springer-Verlag, 1999. 115

[265] B. Morgenstern. DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15:211–218, 1999. 453

[266] B. Márkus. *Térinformatika Magyarországon'94*. Technológia Transzfer Centrum, 1994. 627

[267] G. Márton. Sugárkövető algoritmusok átlagos bonyolultságának vizsgálata, 1995. Kandidátusi értekezés. 590

[268] M. Márton, J. Paksi, B. Márkus. *Térinformatikai alapismeretek*. Technológia Transzfer Centrum, 1994. 627

[269] M. Nebel. Identifying good predictions of rna secondary structure, **9**. In R. B. Altman, A. K. Dunker, L. Hunter T. E. Klein (szerk.), *Pacific Symposium on Biocomputing*, 423–434. o. PSB Online, http://psb.stanford.edu/psb-online/, 2004. 455

[270] 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. 452

[271] M. Nelson, J. L. Gailly. *The Data Compression Book*. M&T Books, 1996. 66

[272] 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. 590

[273] Zs. Németh. Near-optimal bin packing algorithms. TDK dolgozat, ELTE, 2011. 294

[274] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987. 589

[275] R. Paige, R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987. 115

[276] R. Pasco. *Source Coding Algorithms for Fast Data Compression.* PhD thesis, Stanford University, 1976. 65

[277] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal. Pruning closed itemset lattices for association rules. In *Proceedings of the BDA French Conference on Advanced Databases*, 1998. 144

[278] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT*, 398–416. o., 1999. 144

[279] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25–46, 1999. 144

[280] A. C. Pataricza. *Formális módszerek az informatikában.* Typotex, 1990. 66

[281] J. S. Pedersen J. Hein. Gene finding with a hidden Markov model of genome structure and evolution. *Bioinformatics*, 19(2):219–227, 2003. 454

[282] J. Pei, J. Han, R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 21–30. o., 2000. 145

[283] I. Peák. *Bevezetés az automaták elméletébe. Az automaták mint információátalakító rendszerek 1. (Introduction to the Theory of Automata. Automata as Systems for the Transformation of the Information).* Tankönyvkiadó, 1977. 293

[284] D. Pelleg, A. Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 277–281. o., 1999. 178

[285] G. Perlman. HCI bibliography, 2004. 488

[286] S. Petrov. Finite axiomatization of languages for representation of system properties. *Information Sciences*, 47:339–372, 1989. 334

[287] P. A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach.* The MIT Press, 2000. 452

[288] N. Pisanti, M. Sagot. Further thoughts on the syntenic distance between genomes. *Algorithmica*, 34(2):157–180, 2002. 456

[289] J. Podani. *Bevezetés a többváltozós biológiai adatfeltárás rejtelmeibe.* Scientia, 1997. 452, 489

[290] N. Polyzotis, M. N. Garofalakis. Statistical synopses for graph-structured XML databases. In *Proceedings of the 2002 ACM SIGMOD international Conference on Management of Data*, 358–369. o., 2002. 116

[291] R. Pottinger. MinCon: A scalable algorithm for answering queries using views. *The VLDB Journal*, 10(2):182–198, 2001. 236

[292] R. Pottinger, A. Halevy. A scalable algorithm for answering queries using views. In *Proceedings of Very Large Data Bases'00*, 2000, 484–495. 236

[293] F. P. Preparata, M. I. ShamosShamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985. 589

[294] J. Pérez, M. Arenas, C. Gutierrez. Semantics of SPARQL. Technical report, TR/DCC-2006-17, Universidad de Chile, 2006. 405

[295] J. Pérez, M. Arenas, C. Gutierrez. Semantics and Complexity of SPARQL. In *The Semantic Web - ISWC 2006.* Springer, 30–43, 2006. 389, 391, 392, 394, 405

[296] J. Pérez, M. Arenas, C. Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3), Article No. 16, 2009. 405

[297] E. Prud'hommeaux, A. Seaborne. SPARQL query language for RDF. W3C recommendation, January 2008. 405

[298] 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. 454

[299] X. Qian. Query folding. In *Proceedings of International Conference on Data Engineering*, 1996, 48–55. 236

[300] B Rácz. Nonordfp: An FP-growth variation without rebuilding the FP-tree. In B. Goethals, M. J. Zaki (szerk.), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* (FIMI'04, CEUR Workshop Proceedings **126**), 2004. 144

[301] P. Resnick, H. R. Varian. Recommender Systems. *Communications of the ACM*, 40(3):56–58, 1997. 488

[302] J. Richards. *Remote Sensing Digital Image Analysis*. Springer-Verlag, Australia, 1986. 627

[303] P. Rigaux, M. Scholl, A. Voisard. *Spatial Databases with Application to GIS*. Morgan Kaufman Publisher, 2001. 626

[304] J. Rissanen, G. Arithmetic coding. *IBM Journal of Research and Development*, 23:149–162, 1979. 65

[305] J. J. Rissanen. Generalized Kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20:198–203, 1976. 65

[306] E. Rivas, S. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology*, 285(5):2053–2068, 1999. 455

[307] L. Rónyai, G. Ivanyos, R. Szabó. *Algoritmusok* (*Algorithms*). Typotex, 1999. 178

[308] L. Rónyai, G. Ivanyos, R. Szabó. *Algoritmusok* (*Algorithms*). Typotex, 1999. 236

[309] D. F. Rogers, J. Adams. *Mathematical Elements for Computer Graphics*. McGraw-Hill Book Co., 1989. 588

[310] S. H. Roosta. *Parallel Processing and Parallel Algorithms*. Springer-Verlag, 1999. 293

[311] P. Rózsa. *Lineáris algebra és alkalmazásai*. Műszaki Könyvkiadó, 1991 (3. kiadás). 177

[312] A. Sali, Sr., A. Sali. Generalized dependencies in relational databases. *Acta Cybernetica*, 13:431–438, 1998. 334

[313] D. Salomon. *Data Compression*. Springer-Verlag, 2004 (3. kiadás). 66

[314] 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. 488

[315] H. Samet. The quadtree and related hierarchical data structures. *ACM Computer Surveys*, 16(2):187–260, 1984. 626

[316] M. San Martin, C. Gutierrez. Representing, querying and transforming social networks with RDF/SPARQL, Lecture Notes in Computer Science. In *The Semantic Web: Research and Applications*. Springer, 293–307, 2009. 358, 380

[317] M. San Martin, C. Gutierrez, P. T. Wood. SNQL: A social networks query and transformation language. In *5th Alberto Mendelzon Workshop on Foundations of Databases*, 2011. 358, 380

[318] D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal of Appliedl Mathematics*, 28:35–42, 1975. 452

[319] L. A. Santaló. *Integral Geometry and Geometric Probability*. Addison-Wesley, 1976. 590

[320] K. Sayood. *Introduction to Data Compression*. Morgan Kaufman Publisher, 2000 (2. kiadás). 66

[321] M. Schmidt, M. Meier, G. Lausen. Foundations of SPARQL query optimization. In *Proceedings of the 13th International Conference on Database Theory*. ACM, 4–33, 2010. 396, 399, 400, 401, 402, 404, 405

[322] 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. 589

[323] C. Semple, M. Steel. *Phylogenetics*. Oxford Lecture Series in Mathematics and Its Applications 24. Oxford Press, 2003. 452

[324] B. Sharp. Implementing subdivision theory. *Game Developer*, 7(2):40–45, 2000. 589

[325] B. Sharp. Subdivision Surface theory. *Game Developer*, 7(1):34–42, 2000. 589

[326] I. Shindyalov, P. Bourne. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Engineering*, 11(9):739–747, 1998. 454

[327] A. Silberschatz, P. Galvin, G. Gagne. *Applied Operating System Concepts*. John Wiley & Sons, 2000. 293

[328] G. Stephen. *String searching algorithms (Lecture Notes Series on Computing, 3)*. World Scientific, Singapore, 1994. 453

[329] R. Stephens. *Visual Basic Graphics Programming*. John Wiley & Sons, 2000. 626

[330] G. Stewart, J. Sun. *Matrix Perturbation Theory*. Academic Press, 1990. 178

[331] L. Stockmeyer, A. R. Meyer. Word problems requiring exponential time. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, 1–9. o. ACM Press, 1973. 115

[332] G. Stoyan (szerk.). *MATLAB 4. és 5. verzió*. Typotex, 1999. 687

[333] I. Sutherland, G. Hodgeman. Reentrant polygon clipping. *Communications of the ACM*, 17(1):32–42, 1974. 589, 590

[334] L. Szécsi. An effective kd-tree implementation. In J. Lander (szerk.), *Graphics Programming Methods*. Charles River Media, 2003. 590

[335] P. Szeredi, G. Lukácsy, T. Benkő. *A szemantikus világháló elmélete és gyakorlata*. Typotex, 2005. 358, 380

[336] L. Szirmay-Kalos (szerk.). *Theory of Three Dimensional Computer Graphics*. Akadémiai Kiadó, 1995. 590

[337] L. Szirmay-Kalos. *Számítógépes grafika*. ComputerBooks, 1999. 590

[338] L. Szirmay-Kalos, Gy. Antal, F. Csonka. *Háromdimenziós grafika, animáció és játékfejlesztés + CD*. Computerbooks, 2003. 589, 590

[339] L. Szirmay-Kalos, G. Márton. Worst-case versus average-case complexity of ray-shooting. *Computing*, 61(2):103–131, 1998. 589, 590

[340] T. Takeaki. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In B. Goethals, M. J. Zaki, R. (szerk.), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* (FIMI'04, CEUR Workshop Proceedings, **126**), 2004. 144

[341] A. S. Tanenbaum, A. Woodhull. *Operating Systems. Design and Implementation*. Prentice Hall, 1997 (Magyarul: *Operációs rendszerek*. Panem, 1999). 293, 294

[342] D. S. Taubman, M. W. Marcelin. *JPEG 2000 – Image Compression, Fundamentals, Standards and Practice*. Society for Industrial and Applied Mathematics, 1983. 65

[343] B. Thalheim. *Dependencies in Relational Databases*. Teubner Verlagsgesellschaft, 1991. 334

[344] L. Thieme. Algorithmic features of Eclat. In B. Goethals, M. J. Zaki, R. (szerk.), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* (FIMI'04, CEUR Workshop Proceedings, **126**), 2004. 144

[345] 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. 452

[346] I. Tinoco, O., Uhlenbeck M. Levine. Estimation of secondary structure in ribonucleic acids. *Nature*, 230:362–367, 1971. 455

[347] O. G. Tsatalos, M. C. Solomon, Y. Ioannidis. The GMAP: a versatile tool for physical data independence. *The VLDB Journal*, 5(2):101–118, 1996. 236

[348] D. M. Tsou, P. C. Fischer. Decomposition of a relation scheme into Boyce–Codd normal form. *SIGACT News*, 14(3):23–29, 1982. 334

[349] A. Tucker. *Handbook of Computer Science*. Chapman & Hall/CRC, 2004. 116

[350] Y. Uemura, A. Hasegawa, Y. Kobayashi, T. Yokomori. Tree adjoining grammars for RNA structure prediction. *Theoretical Computer Science*, 210:277–303, 1999. 455

[351] J. Ullman. Information ntegration using logical view. In *Proceedings of ICDT'97*, Lecture Notes in Computer Science 1186. kötete. Springer-Verlag, 1997, 19–40. 236

[352] J. D. Ullman. *Principles of Database and Knowledge Base Systems. Vol. 1*. Computer Science Press, 1989 (második kiadás). 334

[353] J. D. Ullman, J. Widom. *A First Course in Database Systems*. Prentice Hall, 1997 (Magyarul: *Adatbázisrendszerek. Alapvetés*. Panem, Budapest, 1998). 236, 334

[354] L. Varga. *Rendszerprogramok elmélete és gyakorlata (Theory and Practice of System Programs)*. Akadémiai Kiadó, 1980. 294

[355] V. Vianu. A Web Odyssey: from Codd to XML. In *Proceedings of the 20th Symposium on Principles of Database Systems*, 1–15. o., 2001. 115

[356] T. Várady, R. R. Martin, J. Cox. Reverse engineering of geometric models—an introduction. *Computer-Aided Design*, 29(4):255–269, 1997. 588

[357] G. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34:30–44, 1991. 65

[358] J. Wang, J. Han, J. Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, 2003. 145

[359] L. Wang, T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computatonal Biology*, 1:337–348, 1994. 452

[360] L. Wang, T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computatonal Biology*, 1:337–348, 1994. 452

[361] J. Wang (szerk.). *Encyclopedia of Data Warehousing and Mining, Vol. 1, Vol. 2.* Idea Group Inc., 2005. 177

[362] J. Warren, H. Weimer. *Subdivision Methods for Geometric Design: A Constructive Approach.* Morgan Kaufmann Publishers, 2001. 589

[363] M. S. Waterman, T. F. Smithand, W. A. Beyer. Some biological sequence metrics. *Advances in Mathematics*, 20:367–387, 1976. 452

[364] D. Watkins. Bulge exchanges in algorithms of QR type. *SIAM Journal on Matrix Analysis and Application*, 19(4):1074–1096, 1998. 687

[365] D. Welsh. *Codes and Cryptography.* Oxford University Press, 1988. 64

[366] J. Wilkinson. Convergence of the LR, QR, and related algorithms. *The Computer Journal*, 8(1):77–84, 1965. 687

[367] F. M. J. Willems, Y. M. Shtarkov, T. J. Tjalkens. The context-tree weighting method: basic properties. *IEEE Transactions on Information Theory*, 47:653–664, 1995. 65

[368] 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. 65

[369] I. H. Witten, R. M. Neal, J. G. Cleary. Arithmetic coding for sequential data compression. *Communications of the ACM*, 30:520–540, 1987. 65

[370] S. Wu, E. W. Myers, U. Manber, W. Miller. An $O(\mathrm{NP})$ sequence comparison algorithm. *Information Processing Letters*, 35(6):317–323, 1990. 453

[371] G. Wyvill, C. McPheeters, B. Wyvill. Data structure for soft objects. *The Visual Computer*, 4(2):227–234, 1986. 589

[372] B. Xia Z. Tan. Tighter bounds of the first fit algorithm for the bin-packing problem. *Discrete Applied Mathematics*, 158:1668–1675, 2010. 294

[373] H. Z. Yang, P. A. Larson. Query transformation for PSJ-queries. In *Proceedings of Very Large Data Bases'87*, 1987, 245–254. 236

[374] J. Yang, K., Karlapalem, Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proceedings of Very Large Data Bases'97*, 1997, 136–145. 236

[375] K. Yi, H. He, I. Stanoi, J. Yang. Incremental maintenance of XML structural indexes. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 491–502. o., 2004. 116

[376] D. M. Young. *Iterative Solution of Large Linear Systems.* Academic Press, 1971 (Magyarul: *Nagy lineáris rendszerek iterációs megoldása.* Műszaki Könyvkiadó, 1979). 687

[377] 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. 333

[378] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, M. Urata. Answering complex SQL queries using automatic summary tables. In *Proceedings of SIGMOD'00*, 2000,105–116. 236

[379] M. J. Zaki. Efficiently mining frequent trees in a forest. In *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 71–80. o. ACM Press, 2002. 144

[380] M. J. Zaki, C. Hsiao. Charm: an efficient algorithm for closed association rule mining. Technical report, RPI, 1999. 144

[381] M. J. Zaki, M. Ogihara. Theoretical foundations of association rules. In *Proceedings of 3rd SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'98)*, 1998. 144

[382] M. J. Zaki, S. Parthasarathy, M. Ogihara, W. Li. New algorithms for fast discovery of association rules. In D. Heckerman, H. Mannila, D. Pregibon, R. Uthurusamy, M. Park (szerk.), *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, 283–296. o. AAAI Press, 1997. 144

[383] B. Zalik, G. Clapworthy. A universal trapezoidation algorithms for planar polygons. *Computers and Graphics*, 23(3):353–363, 1999. 589

[384] 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. 333

[385] C. Zaniolo. A new normal form for the design of relational database schemata. *ACM Transactions on Database Systems*, 7:489–499, 1982. 334

[386] J. Ziv, A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337–343, 1977. 65

[387] J. Ziv, A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24:530–536, 1978. 65

[388] Zs. Zsigmondi A. Kiss. Implementation of natural language semantic wildcards using Prolog. In Á. Kiss (szerk.), *13th Symposium on Programming Languages and Software Tools* Szeged, 2013. aug.26–2013. aug. 27. Szegedi Egyetem Informatikai Tanszékcsoport, 185–199, 2013. 358, 381

[389] M. Zuker, D. Sankoff. RNA structures and their prediction. *Bulletin of Mathematical Biology*, 46:591–621, 1986. 455

Ezt az irodalomjegyzéket HBibTEXsegítségével állítottuk össze. A dokumentumokat elsősorban a szerzők neve (első szerző, második szerző stb.), másodsorban a megjelenés éve, harmadsorban pedig a dokumentumok címe alapján rendeztük.

Az aláhúzás azt jelzi, hogy az aláhúzott szövegrészre kattintva a megfelelő honlapra juthatunk. A hivatkozás végén lévő kék számok pedig azt mutatják, mely oldalakon van hivatkozás az adott dokumentumra.

Tárgymutató

Ez a tárgymutató a következő szempontok szerint készült.

Először a matematikai jelöléseket, azután a tárgyszavakat soroljuk fel (a és á, e és é, i és í, o, ó, ö és ő, u, ú, ü és ű között nem teszünk különbséget).

A számokat és görög betűket tartalmazó tárgyszavakat kiejtésük szerint rendeztük: például a „2-3-4 fá"-t „kettőháromnégy fa"-ként, a „d-kupac"-ot „dkupac"-ként, az „ϵ-sűrű gráf"-ot „epszilonsűrű gráf"-ként.

Az algoritmusok (pszeudokódok) nevét a betűmérettel is megkülönböztettük a többi névtől (függvények, modellek, problémák, nyelvek stb.): például BINOMIÁLIS-KUPACBAN-KERES, illetve VAGY-függvény, ÚT, SAT. A kezdőbetűkből álló szavak betűi azonban mindenütt azonos méretűek: például ALAP-FFT, LKR-NYOMTAT, illetve CRCW, FIFO, MFF, MFH.

Ha egy tárgyszó nem a fő szövegre utal, akkor az oldalszámot kiegészítés követi: *gy* gyakorlatot, *fe* feladatot, *áb* ábrát, *lá* lábjegyzetet jelent.

A különböző típusú objektumokat lehetőség szerint tipográfiailag is megkülönböztettük. A matematikai jelöléseket és a programokban használt változók nevét dőlt betűk emelik ki, mint például $\Omega(n \lg n)$ vagy $rang[kulcs]$. Az algoritmusok neveit kiskapitális betűkkel írtuk, mint például GYORSRENDEZ. Az algoritmusok kódjában a programozási alapszavakat félkövéren szedtük, mint például **if, then, for**.

Az algoritmusok nevében kisközjelet használtunk, mint például UTAZÓ-ÜGYNÖK. Az egyes fogalmak meghatározásának és az algoritmusok pszeudokódjának helyére a tárgymutató dőlt oldalszámmal utal.

Elsősorban az algoritmusokat tárgyaló tankönyvek matematikai jelöléseit alkalmaztuk. Az oldalszámok felsorolásánál nem törekedtünk teljességre.