Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

Robert Šircelj

PRIMERJAVA ALGORITMOV STISKANJA PO METODAH SHANNON-FANO, HUFFMAN IN HUFFMAN S PRILAGAJANJEM

Diplomsko delo

Maribor, avgust 2016

PRIMERJAVA ALGORITMOV STISKANJA PO METODAH SHANNON-FANO, HUFFMAN IN HUFFMAN S PRILAGAJANJEM

Diplomsko delo

Študent:	Robert Šircelj
Študijski program:	VS ŠP, Računalništvo in informatika
Smer:	Programska oprema
Mentor:	red. prof. dr. Borut Žalik

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

Številka: 93508992
Datum in kraj: 22. 04. 2016, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 44/2015)
izdajam

SKLEP O DIPLOMSKEM DELU

1. **Robertu Širclju,** študentu visokošolskega strokovnega študijskega programa RAČUNALNIŠTVO IN INFORMATIKA, smer Programska oprema, se dovoljuje izdelati diplomsko delo.

2. **MENTOR: red. prof. dr. Borut Žalik**

3. **Naslov diplomskega dela:**
PRIMERJAVA ALGORITMOV STISKANJA PO METODAH SHANNON-FANO, HUFFMAN IN HUFFMAN S PRILAGAJANJEM

4. **Naslov diplomskega dela v angleškem jeziku:**
COMPARISON OF SHANNON-FANO, HUFFMAN, AND ADAPTIVE HUFFMAN COMPRESSION ALGORITHM METHODS

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela". Skladno s 7. členom *Pravilnika o postopku priprave in zagovora diplomskega dela na dodiplomskem študiju* je bilo odobreno podaljšanje roka za oddajo diplomskega dela do 31. 08. 2016. Diplomsko delo študent-ka odda v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

Dekan:
red. prof. dr. Borut Žalik

UNIVERZA V MARIBORU FAKULTETA ZA ELEKTROTEHNIKO, RAČUNALNIŠTVO IN INFORMATIKO

Obvestiti:
- kandidata,
- mentorja,
- odložiti v arhiv.

ZAHVALA

Zahvaljujem se mentorju prof. dr. Borutu Žaliku za pomoč pri pripravi diplomske naloge.

Zahvaljujem se tudi staršem, ki so mi študij omogočili ter me nenehno spodbujali, da ga tudi dokončam.

PRIMERJAVA ALGORITMOV STISKANJA PO METODAH SHANNON-FANO, HUFFMAN IN HUFFMAN S PRILAGAJANJEM

Ključne besede: algoritmi stiskanja podatkov, Shannon-Fanojev algoritem, Huffmanov algoritem, Huffmanov algoritem s prilagajanjem, primerjava algoritmov.

UDK: **004.627.021(043.2)**

Povzetek

V diplomski nalogi predstavimo in implementiramo algoritme za stiskanje podatkov brez izgub po metodah Shannon-Fano, Huffman in Huffman s prilagajanjem. Učinkovitost algoritmov testiramo na različnih vrstah datotek ter jih med seboj primerjamo glede na razmerje stiskanja ter čas kodiranja in dekodiranja. Ugotovili smo, da v razmerju stiskanja med metodami ni bistvenih razlik.

COMPARISON OF SHANNON-FANO, HUFFMAN, AND ADAPTIVE HUFFMAN COMPRESSION ALGORITHM METHODS

Key words: data compression algorithms, Shannon-Fano algorithm, Huffman algorithm, adaptive Huffman algorithm, algorithm comparison.

UDK: **004.627.021(043.2)**

Abstract

In this thesis, Shannon-Fano, Huffman, and adaptive Huffman lossless compression algorithms are presented and implemented. The efficiency of the algorithms was tested using various file types. The compression ratio, encoding and decoding time were compared. We found that there are no significant differences in compression ratio results between these methods.

KAZALO

KAZALO SLIK

KAZALO TABEL

1 UVOD

Stiskanje podatkov je proces kodiranja podatkov, pri katerem je rezultat manjše število bitov kot jih je imela izvorna informacija. Pomembno je predvsem pri arhiviranju datotek in pri prenosu podatkov. Za arhiviranje datotek tako potrebujemo manj prostora na trdem disku, pri prenosu pa manjšo pasovno širino povezave, kar nam omogoča hitrejše prenose. Postopek kodiranja oz. stiskanja podatkov opravlja program, ki ga imenujemo kodirnik (angl. coder), postopek razširjanja oz. dekodiranja pa dekodirnik (angl. decoder) [1]. Razširjanje podatkov je postopek, pri katerem stisnjene podatke povrnemo nazaj v njihovo izvorno obliko.

Algoritme za stiskanje podatkov delimo na takšne, ki podatke stiskajo brez izgub (angl. lossless) in takšne, ki podatke stiskajo z izgubami (angl. lossy). Slednji se uporabljajo predvsem pri stiskanju slik, videa in zvoka, kjer si ponavadi lahko privoščimo, da izgubimo delček izvorne informacije (zaradi tega lahko dosežejo boljšo stopnjo stiskanja), pri tem pa človek ponavadi ne opazi bistvene razlike med stisnjeno datoteko in originalom [1].

Podatke lahko uspešno stiskamo samo takrat, kadar je v izvorni datoteki oz. sporočilu prisotna določena stopnja odvečne informacije, ki jo imenujemo redundanca [1]. Vsaka metoda stiskanja podatkov poizkuša redundanco v sporočilu bolj ali manj uspešno odpraviti.

Kako ugotovimo, kako velika je redundanca v sporočilu? S tem problemom se je proti koncu 1940-ih ukvarjal oče informacijske teorije Claude Shannon in ugotovil, da obstaja povezava med informacijo in logaritemsko funkcijo ter pokazal, da je količina informacije (v bitih) simbola s frekvenco pojavitve *P* enaka $-\log_2 P$ [1]. Na osnovi tega spoznanja je nato definiral še enačbo entropije, ki je količina za merjenje informacije v sporočilu, hkrati pa predstavlja tudi teoretični minimum, do katerega lahko podatke stisnemo.

Entropijo sporočila izračunamo po enačbi [1]:

$$H(X) = -\sum_{i=1}^{n} P_i \log_2 P_i \,, \tag{1.1}$$

kjer je:

X – zaloga vrednosti simbolov v sporočilu, ki vsebuje n simbolov x_i,

P_i – verjetnost pojavitve simbola x_i

Glede na enačbo 1.1 entropija sporočila »ABAA« znaša 0,81 bita na simbol, ta pa predstavlja minimalno teoretično dolžino kode, ki bi jo potrebovali za predstavitev posameznega simbola v sporočilu. Tekstovne datoteke v praksi pogosto uporabljajo daljše dolžine kod za predstavitev simbolov, to pa nam omogoča, da podatke stisnemo. Razširjen nabor ASCII (American Standard Code for Information Interchange) [2], ki ga pogosto uporabljamo za predstavitev simbolov v tekstovnih datotekah, vsak simbol v sporočilu predstavi s kodo enake dolžine 8 bitov, kar je zelo potratno. Povprečna dolžina kode za predstavitev simbolov v sporočilu »ABAA« pri uporabi kodiranja ASCII tako znaša 8 bitov na simbol. Redundanco definiramo kot razliko med povprečno dolžino kode simbola in entropije sporočila [1]. Redundanca sporočila »ABAA« tako znaša 7,19 bitov na simbol (8 – 0,81).

Mnogi algoritmi stiskanja redundanco podatkov zmanjšujejo tako, da simbole kodirajo z uporabo kod spremenljivih dolžin (angl. VLC - variable length code). Simbolom, ki se v sporočilu pojavljajo pogosteje, dodelijo krajše kode (manjše število bitov), redkejšim pa daljše (večje število bitov). Da se nekateri simboli pojavljajo pogosteje kot drugi, lahko vidimo tudi iz primera pojavljanja črk v slovenskem leposlovju (tabela 1.1), to pa predstavlja velik potencial za stiskanje podatkov.

Tabela 1.1: Frekvenca črk v slovenskem leposlovju [3]

Mesto	Črka	Relativna frekvenca v slo. leposlovju
1	e	10,71%
2	a	10,47%
3	o	9,08%
4	i	9,04%
5	n	6,33%
6	l	5,27%
7	s	5,05%
8	r	5,01%
9	j	4,67%
10	t	4,33%
11	v	3,76%
12	k	3,70%
13	d	3,39%
14	p	3,37%
15	m	3,30%
16	z	2,10%
17	b	1,94%
18	u	1,88%
19	g	1,64%
20	č	1,48%
21	h	1,05%
22	š	1,00%
23	c	0,66%
24	ž	0,65%
25	f	0,11%

Pomembno je, da kode spremenljivih dolžin lahko enolično dekodiramo, kar pomeni, da ko kodi simbola enkrat dodelimo določeno zaporedje bitov, se nato nobena izmed kod preostalih simbolov ne sme začeti z istim zaporedjem [1]. Ti algoritmi spadajo v kategorijo algoritmov, ki podatke stiskajo na osnovi statističnega modela. Omenimo še, da obstajajo tudi algoritmi, ki podatke stiskajo z uporabo slovarja, vendar jih v tem diplomskem delu ne bomo predstavili.

Učinkovitost stiskalnih metod merimo s kazalnikom razmerja stiskanja (1.2) [1].

$$Razmerje\ stiskanja = \frac{velikost\ izhodnega\ toka\ podatkov}{velikost\ vhodnega\ toka\ podatkov} \tag{1.2}$$

Razmerje stiskanja manjše od 1 kaže, da je bilo stiskanje uspešno, vrednosti večje od 1 pa nam povedo, da je bilo stiskanje neuspešno (stisnjena datoteka je večja od izvorne datoteke). Povedano z drugimi besedami, razmerje stiskanja 0,4 pomeni, da stisnjena datoteka zavzema 40% prostora izvorne datoteke.

Cilj diplomskega dela je predstavitev, implementacija in primerjava algoritmov za stiskanje podatkov. Ker na področju stiskanja podatkov obstaja veliko različnih algoritmov, se bomo omejili na predstavitev in implementacijo algoritmov Shannon-Fano, Huffman in Huffman s prilagajanjem, ki spadajo v skupino statističnih algoritmov stiskanja ter podatke stiskajo z uporabo kod spremenljivih dolžin. Te algoritme bomo podrobneje predstavili v poglavju 2. Znanje iz tega poglavja bomo uporabili v praktičnem delu diplomske naloge, kjer bomo algoritme implementirali v programskem jeziku C# [4], nato pa jih bomo testirali na različnih vrstah datotek. V poglavju 3 bomo rezultate predstavili in jih med seboj primerjali ter nato podali naše ugotovitve.

2 STISKANJE PODATKOV Z UPORABO KOD SPREMENLJIVIH DOLŽIN

2.1 Metoda Shannon-Fano

Metoda Shannon-Fano je bila prva uporabna kodirna metoda, ki za podan niz simbolov poišče kode spremenljivih dolžin. Ime je dobila po avtorjih Claudu Shannonu in Robertu Fanu [1]. Osnovna ideja metode je, da simbolom, ki se v vhodnem nizu pojavljajo pogosteje, dodeli krajše kodirne kode, redkejšim pa daljše. Metoda to doseže tako, da najprej analizira vhodni niz podatkov in glede na število pojavljanj posameznega simbola zgradi tabelo verjetnosti, nato pa zgradi Shannon-Fanojevo drevo, ki ga uporablja za kodiranje in dekodiranje simbolov.

Algoritem za gradnjo Shannon-Fanon drevesa [5]:

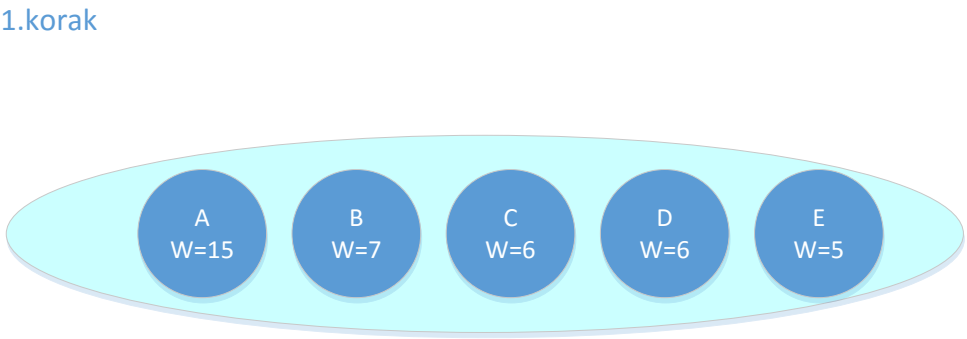
1. Za vsak simbol v vhodnem nizu izračunamo verjetnost pojavljanja le-tega in zgradimo tabelo verjetnosti za vse simbole.
2. Tabelo verjetnosti uredimo tako, da so simboli z največjo verjetnostjo na levi strani tabele, tisti z najmanjšo pa na desni strani tabele.
3. Tabelo verjetnosti razdelimo na levi in desni del tako, da je vsota verjetnosti simbolov v levem delu čim bližje vsoti verjetnosti v desnem delu.
4. Gradnjo drevesa začnemo v korenu (od zgoraj navzdol). Najprej ustvarimo očeta. Simboli, ki so v levem delu tabele verjetnosti, postanejo sinovi na levi strani očeta, simboli z desnega dela pa na desni strani očeta.
5. Rekurzivno ponavljamo koraka 3 in 4, dokler imamo v levem ali desnem delu več kot 1 simbol oz. smo prišli do lista drevesa.

Za lažje razumevanje gradnje Shannon-Fanojevega drevesa le-to razložimo na primeru, ki ga povzemamo po [6]. Za primer bomo uporabili vhodni niz, v katerem se pojavlja pet simbolov. Frekvenco in verjetnost pojavljanj posameznega simbola v tem nizu predstavlja tabela 1.

Tabela 2.1: Verjetnostna tabela vhodnega niza

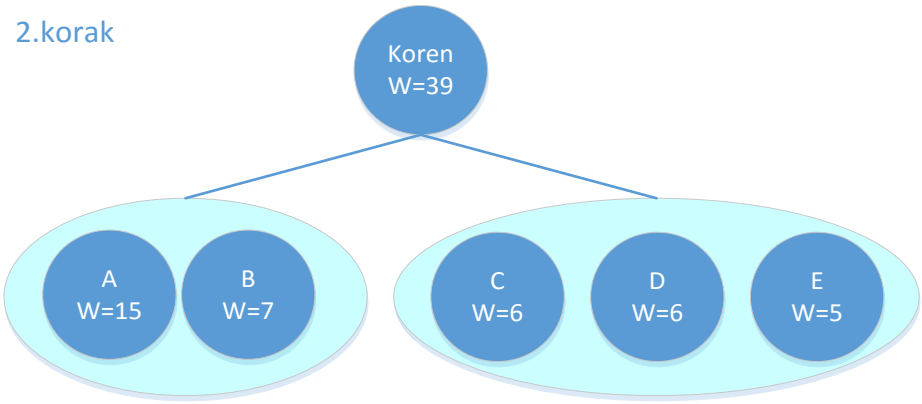
Simbol	A	B	C	D	E
Št. pojavljanj	15	7	6	6	5
Verjetnost	0,38	0,18	0,15	0,15	0,13

Začetni seznam vozlišč vidimo na sliki 1. Število pojavljanj posameznega simbola je na slikah prikazano s črko W.



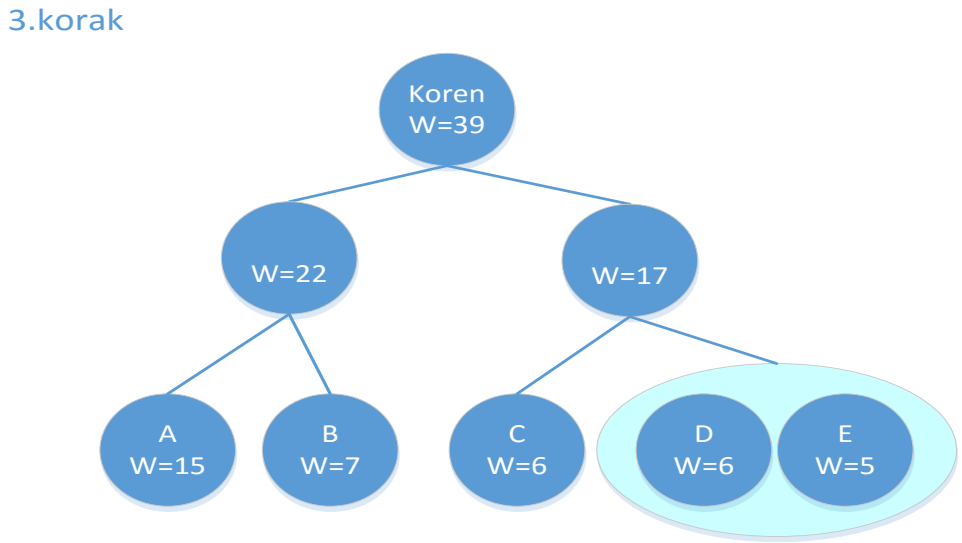
Slika 2.1: Gradnja Shannon-Fano drevesa (1.korak)

Seznam vozlišč razdelimo na levi in desni del. V levem delu seznama se tako nahajata simbola A in B, ki imata skupno frekvenco pojavljanj 22, v desnem pa simbole C, D in E, ki imajo skupno frekvenco pojavljanj 17. Nato ustvarimo korensko vozlišče in mu priredimo vsoto frekvenc pojavljanja iz levega in desnega seznama. Levi del seznama postane levi sin korenskega vozlišča, desni del pa desni sin. Rezultat vidimo na sliki 2.2.



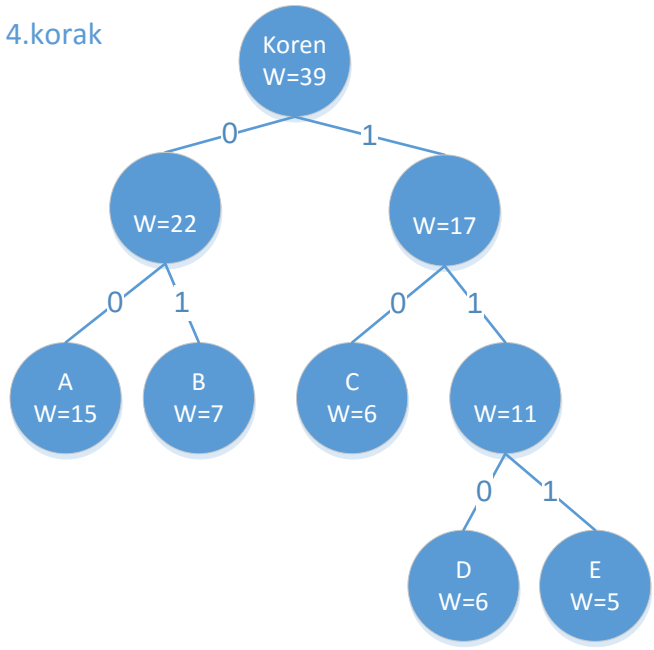
Slika 2.2: Gradnja Shannon-Fano drevesa (2.korak)

Levo poddrevo ponovno razdelimo v dva seznama. V levem delu seznama se nahaja vozlišče A s frekvenco pojavljanj 15, v desnem pa vozlišče B s frekvenco pojavljanj 7. Zanju ustvarimo očeta s frekvenco pojavljanj 22, ki postane sin korenskega vozlišča. Vozlišče A postane njegov levi sin, vozlišče B pa desni sin. Seznama v levem delu drevesa sta vsebovala vsak po samo en element, tako da je gradnja levega dela drevesa zaključena. Desno poddrevo prav tako razdelimo v dva seznama. Levi del seznama tako vsebuje vozlišče C s frekvenco pojavljanj 6, desni pa vozlišči D in E, ki imata skupno frekvenco pojavljanj 11. Tudi zanju ustvarimo očeta s frekvenco pojavljanj 17, ki postane sin korenskega vozlišča. Stanje po tem koraku vidimo na sliki 2.3.



Slika 2.3: Gradnja Shannon-Fano drevesa (3.korak)

Za vozlišči D in E moramo opraviti še zadnje deljenje in ustvariti njunega očeta s skupno frekvenco pojavljanj 11. Vozlišče D postane levi sin, vozlišče E pa desni sin. Končano Shannon-Fanojevo drevo prikazuje slika 2.4.



Slika 2.4: Dokončano Shannon-Fano drevo

Sedaj, ko je drevo enkrat izgrajeno, ga lahko uporabimo za kodiranje ali dekodiranje simbolov iz vhodnega toka podatkov. Kodiranje poteka tako, da iz nekodiranega vhodnega toka podatkov preberemo naslednji simbol, nato pa v drevesu poiščemo list s tem simbolom. Od lista se nato po drevesu pomikamo proti korenu drevesa. Če se premaknemo proti očetu iz smeri levega sina, kodi tega simbola dodamo bit 0, če pa naredimo pomik iz smeri desnega sina pa bit 1. Kodiranje simbola je končano takrat, ko prispemo do korena drevesa. Da dobimo dokončno kodo, ki jo nato lahko zapišemo v kodiran tok podatkov, moramo vrstni red bitov najprej obrniti, saj dekodiranje poteka v obratni smeri (od korena proti listom). To kodo nato zapišemo v kodiran tok podatkov, nato pa nadaljujemo s kodiranjem naslednjega simbola.

Za primer vzamemo kodiranje simbola D ter ga kodiramo z uporabo drevesa na sliki 2.4. Začnemo v listu D in se pomikamo proti korenu, pri tem dobimo kodo 011, nato pa še vrstni red bitov obrnemo in dobimo kodo 110.

Dekodiranje poteka tako, da iz vhodnega toka kodiranih podatkov beremo bit za bitom in se iz korena drevesa spuščamo proti listom. Če preberemo bit z vrednostjo 1 se pomaknemo proti desnemu sinu, sicer se pomaknemo proti levemu sinu. Ko prispemo v list, je dekodiranje simbola končano, le-tega pa zapišemo v izhodni nekodiran podatkovni tok ter nadaljujemo z dekodiranjem naslednjega simbola.

Za primer vzamemo dekodiranje vhodnega niza 110 ter ga dekodiramo z uporabo drevesa na sliki 2.4. Od korena drevesa se pomaknemo dvakrat proti desnemu sinu in enkrat proti levemu sinu ter pristanemo v listu simbola D.

Seznam vseh Shannon-Fano kod za podan primer prikazuje tabela 2.2.

Tabela 2.2: Seznam Shannon-Fano kodirnih kod

Simbol	A	B	C	D	E
Koda	00	01	10	110	111

Iz tabele 2.2 je razvidno, da simboli A, B in C, ki se najpogosteje pojavljajo v vhodnem toku podatkov, dobijo krajšo kodo, ki je dolga 2 bita, medtem ko simbola D in E, ki imata manjšo frekvenco pojavljanja, dobita daljšo kodo.

V vhodnem toku podatkov imamo vsega skupaj 39 simbolov, nestisnjen tok podatkov bi tako zavzel 39 x 8 bitov = 312 bitov, v kolikor bi pa niz stisnili oz. kodirali po metodi Shanon-Fano pa bi le-ta zavzel (15 + 7 + 6) x 2 bita + (6 + 5) x 3 bite = 56 + 33 = 89 bitov.

2.2 Metoda Huffman

Huffmanova metoda je načeloma zelo podobna Shannon-Fanojevi metodi, le da Huffmanova metoda vedno tvori optimalne kode spremenljivih dolžin (angl. variable length code) in je zaradi tega večinoma izpodrinila uporabo Shannon-Fanojevo metodo [5]. Je zelo priljubljena in enostavna metoda in se pogosto uporablja kot eden izmed korakov v bolj zapletenih algoritmih stiskanja, kot je npr. DEFLATE [7]. Tako kot Shannon-Fanojeva metoda tudi Huffmanova metoda tvori najboljše kode, kadar so frekvence pojavljanj simbolov enake negativnim potencam števila 2 [1]. Ime je dobila po Davidu A. Huffmanu, ki je algoritem razvil in objavil leta 1952 še v času svojega študija na M.I.T. [7]. Spada v skupino statističnih algoritmov, ki vhodni tok podatkov stisnejo brez izgub.

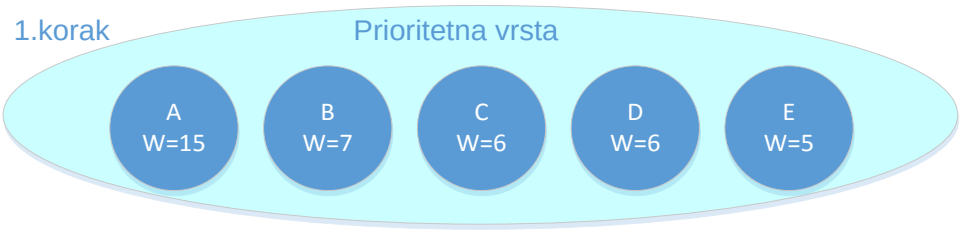
Tako kot pri Shannon-Fanojevi metodi tudi tukaj stiskanje poteka kot dvokoračni postopek. V prvem koraku metoda preleti celotni tok vhodnih podatkov ter zgradi tabelo verjetnosti pojavljanj posameznih simbolov, nato pa zgradi Huffmanovo drevo, ki ga v drugem koraku uporablja za kodiranje oz. dekodiranje simbolov v vhodnem toku podatkov. Bistvena razlika v primerjavi s Shannon-Fanojevo metodo se pojavi pri gradnji drevesa. Huffmanova metoda

drevo gradi od spodaj navzgor oz. od listov drevesa proti korenu, medtem ko Shannon-Fanojeva metoda drevo gradi od zgoraj navzdol oz. od korena proti listom.

Drevo začnemo graditi tako, da za vse simbole, ki se pojavijo v vhodnem toku podatkov, ustvarimo tabelo verjetnosti, nato pa iz tabele zgradimo vozlišča drevesa, ki jih nato vstavimo v prioritetno vrsto (angl. priority queue). Prioritetna vrsta je podatkovna struktura, podobna vrsti ali skladu, vendar pa imajo elementi v njej tudi prioriteto [8]. Pri vstavljanju in jemanju vozlišč iz vrste kot prioriteto uporabimo število pojavljanj vozlišča. Z njo dosežemo, da pri jemanju vozlišča iz vrste vedno dobimo vozlišče z najmanjšim številom pojavljanj. Iz vrste nato vzamemo dve vozlišči in zanju ustvarimo očeta oz. notranje vozlišče drevesa. Očetu za število pojavljanj priredimo vsoto obeh vozlišč, ki smo jih vzeli iz vrste. Vozlišči, ki smo ju ravnokar vzeli iz vrste pa postaneta njegova sinova, nato pa očeta vstavimo nazaj v vrsto. Število vozlišč v vrsti se po tem koraku zmanjša za 1. Postopek ponavljamo tako dolgo, dokler nam v vrsti ne ostane samo eno vozliče in je drevo končano. To zadnje vozlišče postane koren Huffmanovega drevesa.

Za lažje razumevanje gradnjo Huffmanovega drevesa prikažemo še na primeru [6]. Za razlago primera uporabimo isto verjetnostno tabelo, ki smo jo uporabili že pri gradnji Shannon-Fanon drevesa in sicer tabelo 2.1.

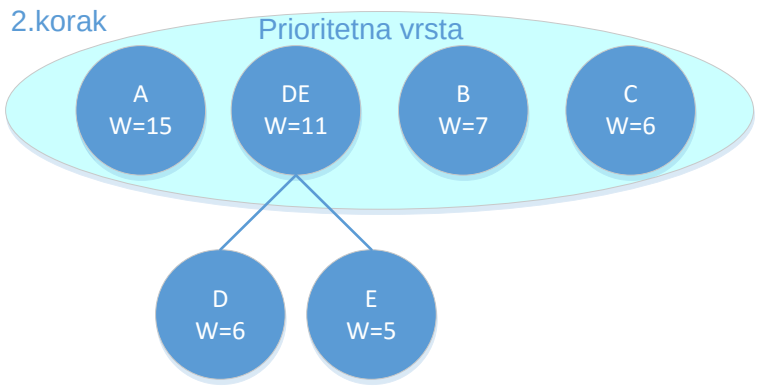
V prvem koraku za vsak simbol ustvarimo vozlišče V_S, nato pa vozlišča vstavimo v prioritetno vrsto. Vsako vozlišče vsebuje informacijo o številu pojavljanj simbola, ki je na slikah v nadaljevanju predstavljeno s črko W. Prioritetna vrsta tako vsebuje vozlišča V_A, V_B, V_C, V_D in V_E, ki so padajoče urejena glede na frekvenco pojavljanj posameznega simbola. Stanje po prvem koraku prikazuje slika 2.5.



Slika 2.5 : Gradnja Huffman drevesa (1.korak)

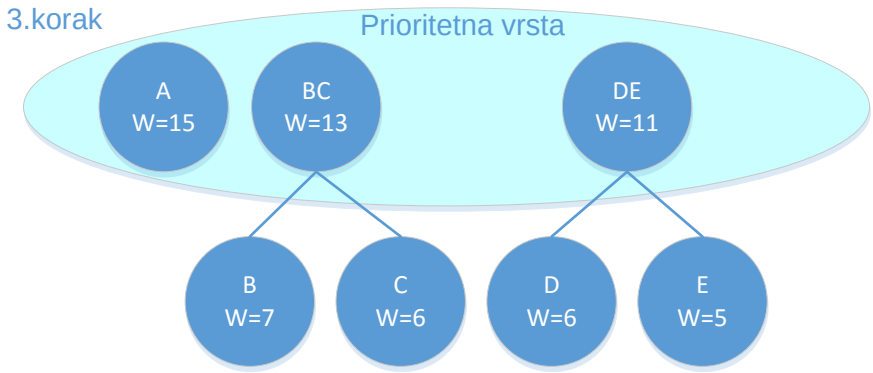
V naslednjem koraku iz vrste vzamemo dve vozlišči, ki imata najmanjši frekvenci pojavljanj. Iz vrste tako vzamemo vozlišči V_E in V_D, nato pa zanju ustvarimo očeta V_{DE}. Vozlišču V_{DE}

priredimo frekvenco pojavljanj na 11, ki predstavlja vsoto frekvenc pojavljanj vozlišč V_E in V_D. Vozlišče V_{DE} nato vstavimo nazaj v vrsto, pri tem pa opazimo da se je število vozlišč v vrsti zmanjšalo za 1. Stanje po tem koraku prikazuje slika 2.6.



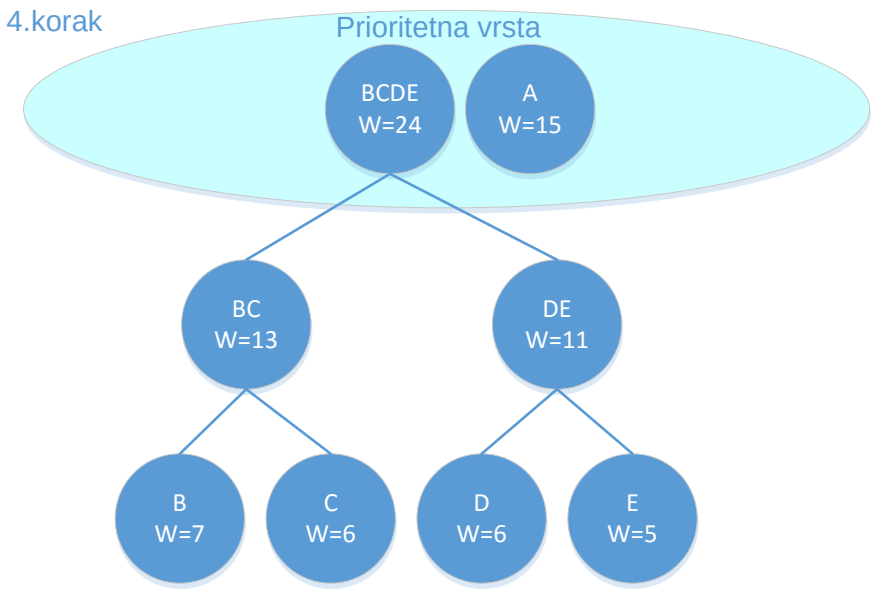
Slika 2.6: Gradnja Huffman drevesa (2.korak)

Postopek nadaljujemo in ponovno iz vrste vzamemo dve vozlišči z najmanjšim številom pojavljanj. Iz vrste tokrat vzamemo vozlišči V_B in V_C ter zanju ustvarimo očeta V_{BC}. Število pojavljanj vozlišča V_{BC} nastavimo na 13, nato pa ga vstavimo nazaj v vrsto. Novo stanje delno izgrajenega drevesa prikazuje slika 2.7.



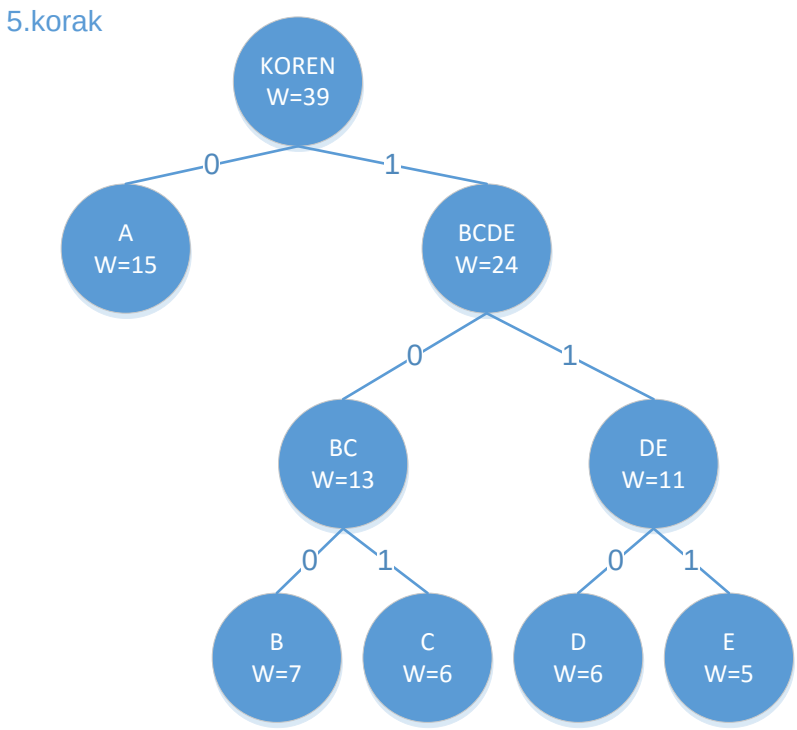
Slika 2.7: Gradnja Huffman drevesa (3.korak)

V naslednjem koraku iz vrste vzamemo vozlišči V_{DE} in V_{BC} ter zanju ustvarimo očeta V_{BCDE} ter mu priredimo število pojavljanj 24. Očeta V_{BCDE} vstavimo nazaj v vrsto. Kot je razvidno iz slike 2.8, po tej iteraciji v vrsti ostaneta samo še dve vozlišči V_A in V_{BCDE}.



Slika 2.8: Gradnja Huffman drevesa (4.korak)

Iz vrste nato vzamemo zadnji dve vozlišči V_A in V_{BCDE}. Zanju ustvarimo očetovsko vozlišče V_{ABCDE}, ki mu ponovno priredimo vsoto števila pojavljanj vozlišč V_A in V_{BCDE}. Vozlišče V_{ABCDE} nato vstavimo nazaj v vrsto in opazimo, da imamo v vrsti samo še eno vozlišče, ki postane koren Huffmanovega drevesa, kar pomeni da je gradnja drevesa končana. Končano Huffmanovo drevo prikazuje slika 2.9.



Slika 2.9: Dokončano Huffmanovo drevo

Dokončano Huffmanovo drevo nato uporabimo za kodiranje ali dekodiranje simbolov v vhodnem toku podatkov. Kodiranje in dekodiranje poteka na povsem enak način kot pri Shannon-Fanojevi metodi. Iz vhodnega toka podatkov preberemo nekodiran simbol, nato pa se od njegovega lista pomikamo proti korenu drevesa.

Dekodiranje poteka v obratni smeri, pričnemo v korenu drevesa, iz vhodnega toka kodiranih podatkov beremo bit za bitom, nato pa se pomikamo proti listom. V primeru da iz vhodnega toka preberemo vrednost 1, se pomaknemo proti desnemu sinu, sicer se pomaknemo proti levemu.

Na osnovi Huffmanovega drevesa iz slike 2.9 tvorimo kode za kodiranje simbolov, ki so prikazane v tabeli 2.3.

Tabela 2.3: Huffmanove kode simbolov

Simbol	A	B	C	D	E
Koda	0	100	101	110	111

V vhodnem toku podatkov imamo vsega skupaj 39 simbolov, nestisnjen tok podatkov bi tako zavzel 39 x 8 bitov = 312 bitov, v kolikor bi pa niz stisnili oz. kodirali po Huffmanovi metodi pa bi le-ta zavzel 15 x 1bit + (7 + 6 + 6 + 5) x 3 bite = 15 + 72 = 87 bitov.

2.3 Metoda Huffman s prilagajanjem

Klasična Huffmanova metoda ni vedno primerna za stiskanje vseh vrst vhodnih podatkov. Eden izmed problemov, na katerega lahko naletimo, je ta, da pri uporabi Huffmanove metode dobimo izhodni tok podatkov, ki zavzame več prostora kot vhodni tok podatkov, kar pomeni, da je bilo stiskanje neuspešno. To se pogosto zgodi predvsem takrat, kadar stiskamo zelo kratke vhodne tokove podatkov. Krivec za to je pogosto tabela verjetnosti oz. Huffmanov drevo, ki ga mora kodirnik dodatno zapisati v kodiran tok podatkov, saj ga dekodirnik potrebuje, da lahko uspešno dekodira sporočilo.

Klasične Huffmanove metode prav tako ne moremo uporabiti, kadar nimamo dostopa do celotnega vhodnega toka podatkov, kar pa je v praksi zelo pogost pojav (npr. prenos videa ali zvoka v živo), saj Huffmanova metoda deluje kot dvokoračni postopek, ki mora v prvem koraku najprej analizirati vhodni tok podatkov in izgraditi tabelo verjetnosti pojavljanja simbolov. Zgoraj omenjenim problemom se lahko izognemo z uporabo tako imenovane Huffmanove metode s prilagajanjem (angl. adaptive Huffman coding), ki so jo razvili Faller, Gallager in Knuth [1]. To metodo uporablja tudi stiskalni program compact, ki je del operacijskega sistema UNIX [1].

Osnovna ideja algoritma je, da kodirnik in dekodirnik proces kodiranja oz. dekodiranja pričneta s praznim Huffmanovim drevesom, nato pa po vsakem kodiranju oz. dekodiranju posameznega simbola Huffmanovo drevo sproti gradita oz. prilagodita do tedaj znanim frekvencam pojavljanj simbolov, kar povzroči, da kodirne kode simbolov niso statične, ampak se med procesom kodiranja oz. dekodiranja spreminjajo. Povedano z drugimi besedami, dekodirnik pri gradnji drevesa povsem oponaša korake kodirnika, posledično pa zaradi tega ni potrebno prenesti tabele verjetnosti pojavljanj simbolov med kodirnikom in dekodirnikom. Prednost takšnega pristopa je, da lahko podatke kodiramo v realnem času [9].

Postopek kodiranja poteka tako, da pričnemo s praznim drevesom. Ko novi simbol prvič preberemo iz vhodnega toka podatkov, ga nekodiranega (8-bitna ASCII koda simbola)

zapišemo v kodiran tok podatkov, saj kodirne kode še nimamo, nato pa ga s frekvenco pojavljanj 1 dodamo v Huffmanovo drevo. Naslednjič, ko iz vhodnega toka podatkov preberemo isti simbol, nam je Huffmanova kodirna koda že na voljo, zato le-to zapišemo v kodiran tok podatkov, nato pa v drevesu frekvenco pojavljanj tega simbola in vseh njegovih predhodnikov povečamo za 1. S povečanjem frekvence pojavljanj simbola se nam lahko zgodi to, da takšno drevo ni več Huffmanovo drevo. V tem primeru moramo drevo ažurirati, kar pomeni da moramo določena vozlišča v drevesu med seboj zamenjati.

Binarno drevo je Huffmanovo drevo, kadar izpolnjuje lastnost dvojčkov (angl. sibling property) [1]. Drevo izpolnjuje lastnost dvojčkov, kadar imajo vsa vozlišča v drevesu, razen korena drevesa, svojega dvojčka (to so vozlišča, ki si delijo skupnega očeta) in v katerem lahko vse frekvence pojavljanj vozlišč naraščajoče uredimo v seznam, za vse pare dvojčkov pa velja, da sta dvojčka iz istega para soseda v urejenem seznamu. V primeru da z povečanjem frekvence pojavljanj simbola katero izmed vozlišč v drevesu poruši lastnost dvojčkov (frekvenca pojavljanj trenutnega vozliča ima večjo vrednost od frekvence pojavljanj naslednjega vozlišča v urejenem seznamu), moramo to vozlišče (skupaj z njegovimi sinovi) zamenjati z zadnjim vozliščem v urejenem seznamu, ki ima manjšo frekvenco pojavljanj od tega vozlišča.

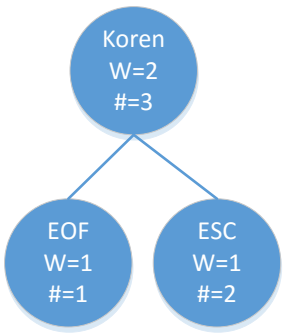
Dekodiranje prav tako pričnemo s praznim drevesom. V primeru da pri dekodiranju iz kodiranega toka podatkov preberemo nekodiran simbol (ponavadi je to 8-bitna koda ASCII simbola), tega vstavimo v Huffmanovo drevo s frekvenco pojavljanj 1, kadar pa preberemo kodiran (Huffmanova koda spremenljive dolžine) simbol, le-temu v drevesu povečamo frekvenco pojavljanj za 1.

Pri dekodiranju tako naletimo na problem, da ne vemo, kdaj je potrebno simbole brati nekodirane in kdaj bit za bitom ter nato s pomočjo trenutnega Huffmanovega drevesa simbole dekodirati. Rešitev se skriva v uporabi ubežne kode (angl. escape code) oz. posebnega simbola ESC [1]. Simbol ESC dobi svojo Huffmanovo kodirno kodo, ki jo med procesom kodiranja zapišemo v kodiran tok podatkov in na tak način dekodirniku sporočimo, da sledi nekodiran simbol. V procesu dekodiranja tako vedno beremo samo bit za bitom in dekodiramo simbole. V primeru, da kodo dekodiramo v simbol ESC, izstopimo iz trenutnega konteksta in preberemo nekodiran simbol, nato pa nadaljujemo z običajnim postopkom branja bita za bitom.

Drugi problem, ki se lahko pojavi pri dekodiranju, je, da včasih ne vemo, kdaj smo z dekodiranjem celotnega vhodnega toka podatkov zaključili. Predstavljajmo si, da je Huffmanova koda zadnjega simbola v datoteki dolga 3 bite, operacijski sistemi pa imajo večinoma omejitev, da je minimalna enota, ki jo lahko zapišemo v datoteko, dolga 1 zlog oz. 8 bitov. V tem primeru bi dekodirnik poizkušal dekodirati tudi teh zadnjih 5 bitov, ki dejansko ne predstavljajo ničesar, delovanje programa za razširjanje podatkov pa bi postalo nezanesljivo. Tudi ta problem rešimo tako, da med postopkom kodiranja uporabimo posebni simbol EOF (angl. end of file), s katerim dekodirniku povemo, da smo zaključili s kodiranjem vhodnega toka podatkov.

Kodirnik in dekodirnik Huffmanovo drevo tako inicializirata s simboloma EOF in ESC, ki imata frekvenco pojavljanja vedno 1 in sta zaradi tega vedno na dnu Huffmanovega drevesa [1].

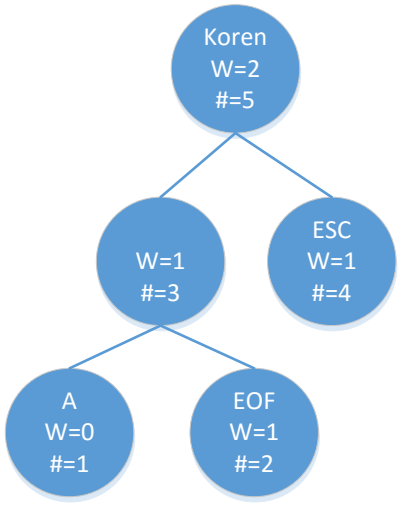
Za lažje razumevanje, gradnjo in ažuriranje Huffmanovega drevesa predstavimo na primeru vhodnega niza simbolov »ABAA«. Pričnemo z začetnim stanjem Huffmanovega drevesa, v katerem že imamo vozlišči V_{ESC} in V_{EOF} s frekvenco pojavljanja 1. Vsa vozliča v drevesu so naraščajoče urejena (označeno z #) v seznam po frekvenci pojavljanj (označeno z W) od leve proti desni in od listov proti korenu drevesa. Opazimo tudi to, da drevo izpolnjuje lastnost dvojčkov, saj so vsi dvojčki v drevesu sosedje v urejenem seznamu. Začetno drevo prikazuje slika 2.10.



Slika 2.10: Začetno Huffmanovo drevo

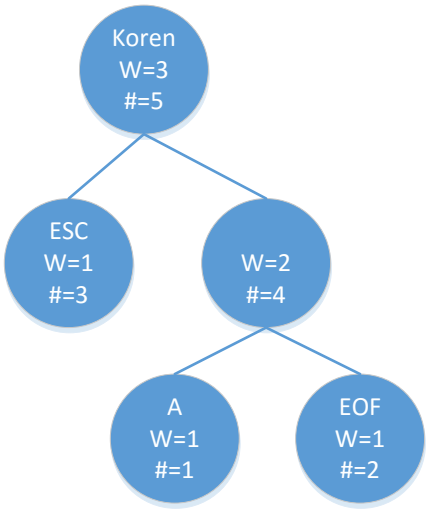
Prvi simbol, ki ga preberemo, je simbol A. Ker simbola A še nimamo v Huffmanovem drevesu, ustvarimo vozlišče V_A s frekvenco pojavljanj 0 in ga vstavimo na začetek urejenega seznama. Ustvarimo novo notranje vozlišče $V_{\#3}$, ki postane oče vozlišč V_A in

V_{EOF}. Vozlišču $V_{\#3}$ priredimo vsoto frekvenc pojavljanj njegovih sinov, vozlišč V_A in V_{EOF}. Stanje po tem koraku vidimo na sliki 2.11.



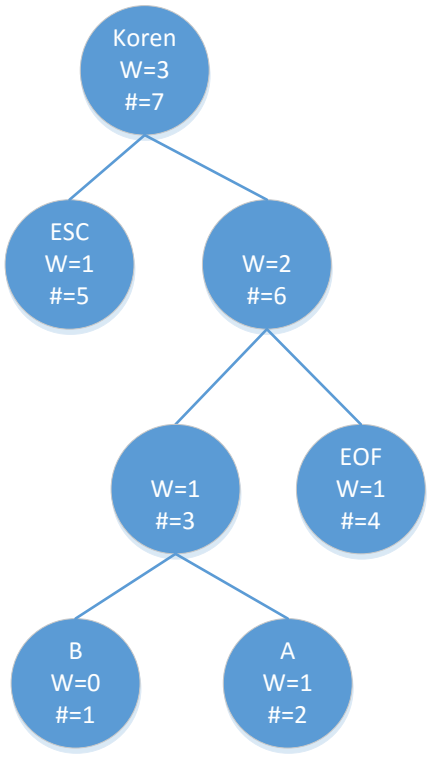
Slika 2.11: Huffmanovo drevo pred povečanjem frekvence pojavljanj simbola A

V naslednjem koraku povečamo frekvenco pojavljanj V_A za 1. Opazimo, da pri tem nismo prekršili pravila dvojčkov, saj frekvenca pojavljanj V_A ne presega frekvence pojavljanj vozlišča V_{EOF}, ki je njegov naslednik, zato v naslednjem koraku povečamo frekvenco pojavljanj za 1 njegovemu očetu $V_{\#3}$. Opazimo, da smo pri tem koraku prekršili pravilo dvojčkov, saj je nova frekvenca pojavljanj $V_{\#3}$ (W=2) večja od frekvence pojavljanj njegovega naslednika vozlišča V_{ESC} (W=1). Zadnje vozlišče v seznamu, ki ima manjšo težo od vozlišča $V_{\#3}$, je vozlišče V_{ESC}, zato ti dve vozlišči med seboj zamenjamo, nato pa očetu, vozlišču V_{KOREN} frekvenco pojavljanja povečamo iz 2 na 3 (Slika 2.12).

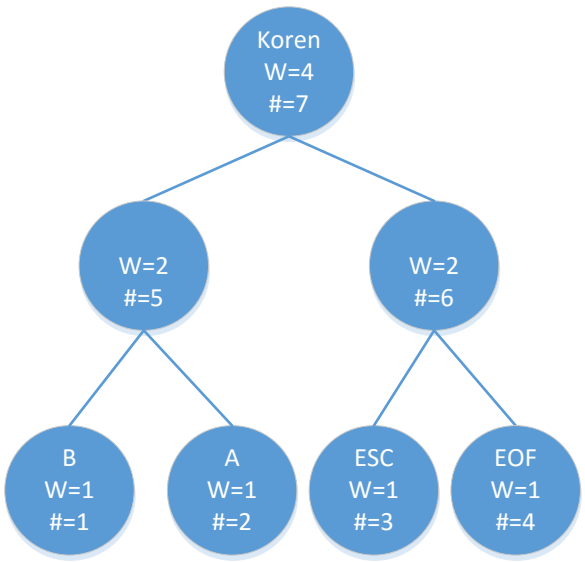


Slika 2.12: Huffmanovo drevo po prebranem nizu »A«

Naslednji simbol, ki ga preberemo, je simbol B. Simbola B prav tako še nimamo v drevesu, zato po istem postopku kot za simbol A, ustvarimo vozlišče V_B s frekvenco pojavljanj 0 in ga vstavimo na začetek seznama. Ustvarimo tudi novo očetovsko vozlišče $V_{\#3}$, njegova sinova pa postaneta vozlišči V_B in V_A. Vozlišču V_B ter vsem njegovim predhodnikom nato povečamo frekvenco pojavljanj za 1, pri tem pa pazimo, da ažuriramo lego vozlišč, kadar prekršimo pravilo dvojčkov. Pravilo dvojčkov ponovno prekršimo, ko vozlišču $V_{\#3}$ frekvenco povečamo na 2, zato moramo nato v drevesu med seboj zamenjati vozlišči $V_{\#3}$ in V_{ESC}. Stanje pred zamenjavo vidimo na sliki 2.13, stanje po zamenjavi in povečanju vseh frekvenc pojavljanja pa na sliki 2.14.

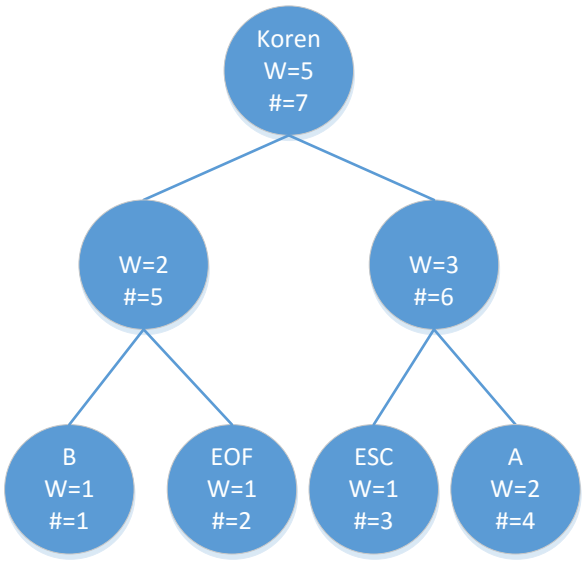


Slika 2.13: Drevo pred povečanjem frekvence pojavljanj simbola B



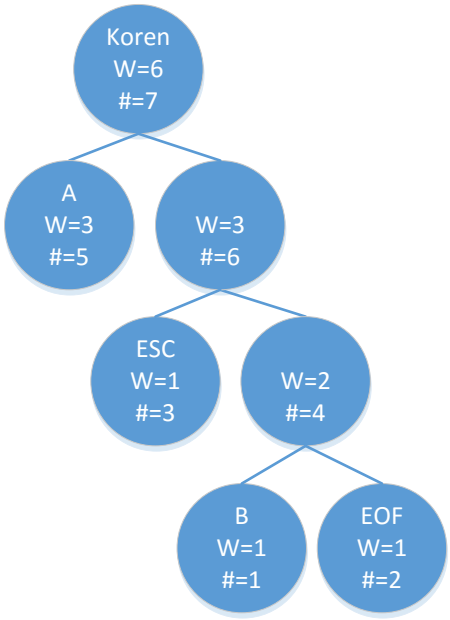
Slika 2.14: Drevo po prebranem nizu »AB«

V naslednjem koraku ponovno preberemo simbol A. Tokrat se simbol A že nahaja v drevesu, zato moramo vozlišču V_A samo povečati frekvenco pojavljanj za 1. Pri tem ponovno pride do prekršitve pravila dvojčkov, zato vozlišče V_A zamenjamo z vozliščem V_{EOF}. Stanje drevesa po prebranem nizu »ABA« prikazuje slika 2.15.



Slika 2.15: Huffmanovo drevo po prebranem nizu »ABA«

Iz vhodnega toka podatkov še enkrat preberemo simbol A, ter mu ponovno povečamo frekvenco pojavljanj za 1. Ponovno prekršimo pravilo dvojčkov, ko vozlišču V_A frekvenco pojavljanj povečamo iz 2 na 3, zato zamenjamo vozlišči V_A in $V_{\#5}$. Končano drevo vidimo na sliki 2.16.



Slika 2.16: Huffmanovo drevo po prebranem nizu »ABAA«

Tabela 2.4 prikazuje prilagajanje Huffmanovih kod po vsakem kodirnem koraku.

Tabela 2.4: prilagajanje Huffmanovih kod

Simbol	A	B	ESC	EOF
Koda po nizu »«	-	-	1	0
Koda po nizu »A«	10	-	0	11
Koda po nizu »AB«	01	00	10	11
Koda po nizu »ABA«	11	00	10	01
Koda po nizu »ABAA«	0	110	10	111

Vhodni niz »ABAA« bi tako zakodirali kot zaporedje bitov »1 01000001 0 01000010 01 11 111«, ki skupaj zavzamejo 25 bitov, kar je 7 bitov manj od nekodirane predstavitve istega niza, ki bi zavzel 4 x 8 bitov = 32 bitov.

3 REZULTATI

Metode smo implementirali v programskem jeziku C# v programskem okolju Microsoft Visual Studio 2013. Teste smo izvedli na namiznem računalniku z naslednjo specifikacijo:

- procesor Intel Core 2 Quad 2,83 GHz,
- delovni pomnilnik Corsair DDR2 8 GB,
- osnovna plošča ASUS P5Q-E,
- trdi disk Samsung 840 Pro 256 GB,
- grafična kartica NVIDIA GeForce 8800 GTX,
- operacijski sistem Windows 10 Pro.

Vse tri metode smo testirali na datotekah različnih tipov, vsebin in velikosti. Te so:

- slovenska leposlovna besedila [10]:
    - Fran Levstik POPOTOVANJE OD LITIJE DO ČATEŽA.txt (v nadaljevanju LS1.TXT) v velikosti 46 KB,
    - Ivan Cankar HLAPEC JERNEJ IN NJEGOVA PRAVICA.txt (v nadaljevanju LS2.TXT) v velikosti 104 KB,
    - Jakob Sket MIKLOVA ZALA.txt (v nadaljevanju LS3.TXT) v velikosti 206 KB,
    - Gustav Šilih BELI DVOR.txt (v nadaljevanju LS4.TXT) v velikosti 792 KB.

- datoteke z naključnim pojavljanjem simbolov, ki smo jih ustvarili sami. V datotekah AlphaRandom se naključno pojavljajo črke abecede [A-Za-z], medtem ko se v datotekah Random naključno pojavlja vseh 256 znakov standarda ASCII.
    - AlphaRandom50kb.txt (v nadaljevanju ARND1.TXT) v velikosti 50 KB,
    - AlphaRandom100kb.txt (v nadaljevanju ARND2.TXT) v velikosti 100 KB,
    - AlphaRandom200kb.txt (v nadaljevanju ARND3.TXT) v velikosti 200 KB,
    - Random50kb.txt (v nadaljevanju RND1.TXT) v velikosti 50 KB,
    - Random100kb.txt (v nadaljevanju RND2.TXT) v velikosti 100 KB,
    - Random200kb.txt (v nadaljevanju RND3.TXT) v velikosti 200 KB.

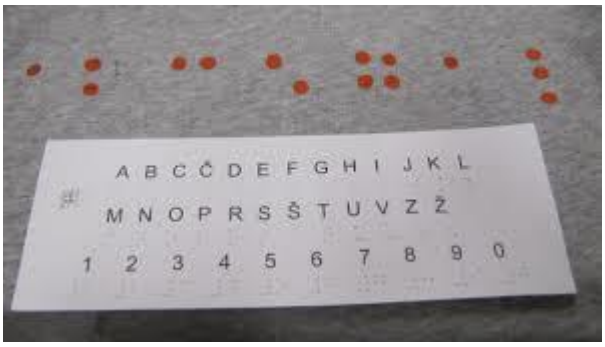
- delno urejene datoteke, ki smo jih ustvari sami. V teh datotekah se periodično ponavljajo črke [A-Z]. Vsaka črka se ponovi 2048 krat, nato pa pričnemo s ponavljanjem naslednje črke. Ko zadnjo črko (Z) ponovimo 2048 krat, ponovno pričnemo s ponavljanjem prve črke (A).
    - PartiallySorted50kb.txt (v nadaljevanju PS1.TXT) v velikosti 50 KB,
    - PartiallySorted100kb.txt (v nadaljevanju PS2.TXT) v velikosti 100 KB

- PartiallySorted200kb.txt (v nadaljevanju PS3.TXT) v velikosti 200 KB.

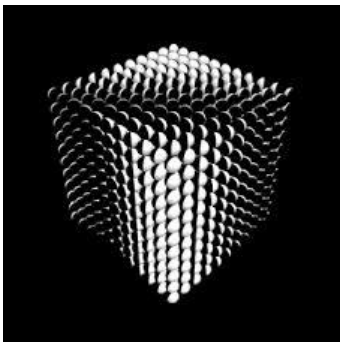
- urejena datoteka english.dic v velikosti 3972 KB. Datoteka vsebuje angleške besede, ki so urejene po abecedi,

- binarne datoteke:
  - netscape.exe (v nadaljevanju BIN1.EXE) v velikosti 2866 KB,
  - uTorrent.exe (v nadaljevanju BIN2.EXE) v velikosti 2314 KB.

- slike (3.1, 3.2, 3.3 in 3.4):
  - abeceda.bmp (v nadaljevanju PIC1.BMP) v velikosti 148 KB,
  - črno bela kocka.bmp (v nadaljevanju PIC2.BMP) v velikosti 149 KB,
  - sončnica.jpg (v nadaljevanju PIC3.JPG) v velikosti 861 KB,
  - višinski pasovi.bmp (v nadaljevanju PIC4.BMP) v velikosti 861 KB.

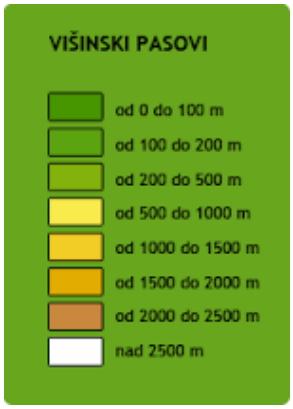


Slika 3.1: PIC1.BMP



Slika 3.2: PIC2.BMP

Slika 3.3: PIC3.JPG



Slika 3.4: PIC4.BMP

- videodatoteke, ki smo jih posneli z mobilnim telefonom:
  - klopotec.3gp (v nadaljevanju VID1.3gp) v velikosti 6511 KB,
  - novol.jelka.3gp (v nadaljevanju VID2.3gp) v velikosti 18229 KB.

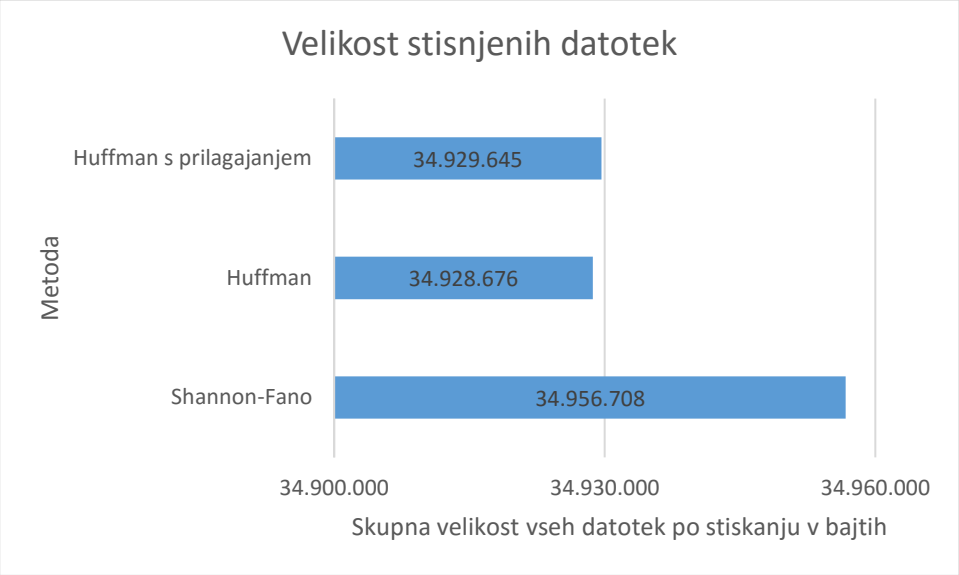
- zelo kratka datoteka ABAAB.txt v velikosti 5 zlogov, katere edina vsebina je niz »ABAAB«.

V tabeli 3.1 smo prikazali velikost testnih datotek ter njihovo entropijo, ki jo izračunamo po enačbi 1.1. Pri testiranju vseh metod smo merili velikost stisnjene datoteke, čas kodiranja, čas dekodiranja, entropijo stisnjene datoteke ter razmerje stiskanja. Razmerje stiskanja smo izračunali po enačbi 1.2. Vse teste smo izvedli trikrat z namenom, da bi dobili čimbolj verodostojne čase kodiranja in dekodiranja, nato pa smo izračunali povprečne vrednosti, ki jih prikazujemo v tabelah v naslednjih podpoglavjih.

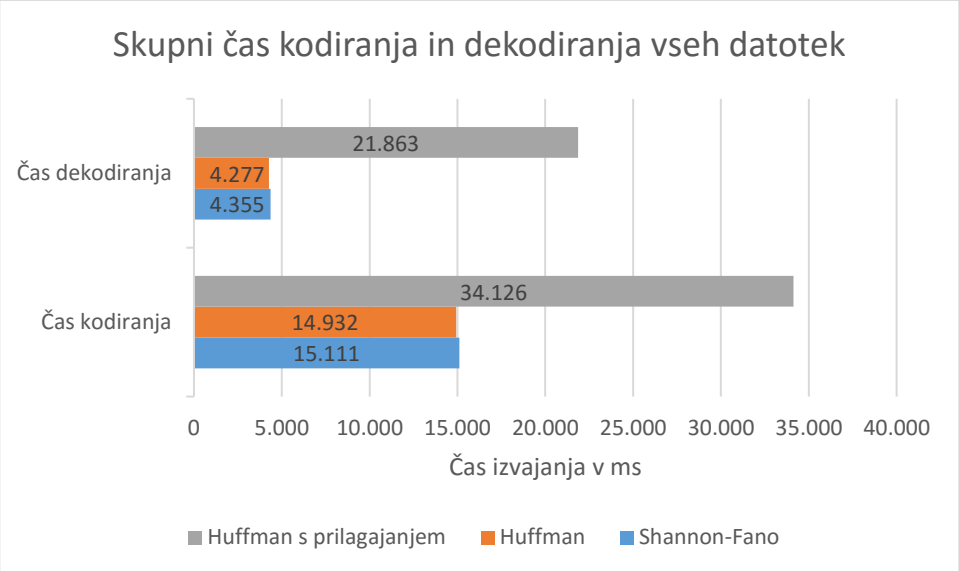
Tabela 3.1: Velikost in entropija testnih datotek

Datoteka	Velikost v zlogih	Entropija
LS1.TXT	47.425	4,60
LS2.TXT	106.119	4,60
LS3.TXT	210.464	4,55
LS4.TXT	811.012	4,52
ARND1.TXT	51.200	5,70
ARND2.TXT	102.400	5,70
ARND3.TXT	204.800	5,70
RND1.TXT	51.200	8,00
RND2.TXT	102.400	8,00
RND3.TXT	204.800	8,00
PS1.TXT	51.200	4,64
PS2.TXT	102.400	4,68
PS3.TXT	204.800	4,69
BIN1.EXE	2.934.336	6,46
BIN2.EXE	2.369.536	7,99
english.dic	4.067.439	4,32
PIC1.BMP	151.254	7,05
PIC2.BPM	152.154	3,08
PIC3.JPG	881.737	7,84
PIC4.BMP	96.726	3,76
VID1.3gp	6.667.402	8,00
VID2.3gp	18.666.272	8,00
ABAAB.TXT	5	0,97

Slika 3.5 prikazuje velikost vseh stisnjenih datotek po metodah. Čas kodiranja in dekodiranja pa je prikazan na sliki 3.6.



Slika 3.5: Velikost stisnjenih datotek



Slika 3.6: Skupni čas kodiranja in dekodiranja testnih datotek

3.1 Metoda Shannon-Fano

V tabeli 3.2 prikazujemo rezultate stiskanja datotek, ki smo jih dobili z metodo Shannon-Fano. Metoda se je izkazala za uspešno, saj je stisnila 16 od 23 ($\approx 70\%$) testnih datotek. Najbolje se je izkazala pri stiskanju slik PIC2.BMP in PIC4.BMP ter leposlovnih besedil, ni ji pa uspelo stisniti nobene izmed videodatotek (VID1.3gp in VID2.3gp) ter datotek z naključno porazdelitvijo podatkov (RND1.TXT, RND2.TXT, RND3.TXT). To je pričakovano, saj imajo datoteke z visoko entropijo (nad 7,8) zelo malo redundance. Na splošno smo opazili povezavo med entropijo izvorne datoteke ter razmerjem stiskanja. Manjša kot je entropija izvorne datoteke, boljše je razmerje stiskanja, ki jo metoda doseže. Edina izjema je bila datoteka ABAAB.TXT, ki ima nizko entropijo, vendar je metoda ni uspela stisniti, saj je kodirnik moral v datoteko zapisati tudi kodirno drevo ali pa tabelo verjetnosti pojavljanja simbolov, kar pa pri tako majhni datoteki zavzame več prostora, kot je velika izvorna datoteka. Metoda je uspešno stisnila tudi datoteke z naključno porazdelitvijo simbolov (ARND1.TXT, ARND2.TXT, ARND3.TXT). To je možno zato, ker se v datoteki ne pojavlja 256 različnih simbolov ASCII, ampak samo 50 simbolov, ki jih lahko predstavimo s kodirnimi kodami, ki so krajše od 8 bitov.

Na čas kodiranja in čas dekodiranja vplivata 2 faktorja:

- velikost vhodne datoteke,
- število različnih simbolov v datoteki (kodiranje datotek RND*.TXT, v katerih se pojavlja 256 različnih simbolov, traja več časa, kot kodiranje datotek ARND*.TXT, kjer se pojavlja samo 50 različnih simbolov).

Tabela 3.2: Rezultati stiskanja po metodi Shannon-Fano

Datoteka	Velikost stisnjene datoteke v zlogih	Čas kodiranja (ms)	Čas dekodiranja (ms)	Entropija stisnjene datoteke	Razmerje stiskanja
LS1.TXT	27.663	25	4	7,88	0,58330
LS2.TXT	61.901	35	8	7,88	0,58332
LS3.TXT	121.078	70	17	7,85	0,57529
LS4.TXT	462.648	249	66	7,90	0,57046
ARND1.TXT	37.095	19	4	7,97	0,72451
ARND2.TXT	74.139	36	9	7,98	0,72401
ARND3.TXT	148.218	74	19	7,97	0,72372
RND1.TXT	51.697	22	6	7,99	1,00971
RND2.TXT	103.025	44	12	8,00	1,00610
RND3.TXT	205.576	88	25	8,00	1,00379
PS1.TXT	30.501	14	3	5,13	0,59572
PS2.TXT	61.222	29	6	5,16	0,59787
PS3.TXT	122.662	57	12	5,17	0,59894
BIN1.EXE	2.397.149	1.125	310	7,91	0,81693
BIN2.EXE	2.369.991	988	290	8,00	1,00019
english.dic	2.214.195	1.164	321	7,75	0,54437
PIC1.BMP	134.436	58	16	7,95	0,88881
PIC2.BMP	61.218	38	7	7,10	0,40234
PIC3.JPG	871.373	385	108	7,96	0,98825
PIC4.BMP	47.646	26	6	6,45	0,49259
VID1.3gp	6.671.633	2.762	815	8,00	1,00063
VID2.3gp	18.681.636	7.803	2.291	8,00	1,00082
ABAAB.TXT	6	0	0	2,58	1,20000

3.2 Metoda Huffman

Rezultati kodiranja datotek s Huffmanovo metodo (tabela 3.3) so zelo podobni rezultatom metode Shannon-Fano. Tudi ta metoda je uspešno stisnila 16 od 23 ($\approx 70\%$) testnih datotek. Razlike v razmerju stiskanja so minimalne, skorajda zanemarljive, je pa Huffmanova metoda 19 datotek bolje stisnila, 4 datoteke pa enako. Tudi pri času kodiranja in dekodiranja nismo opazili bistvenih razlik.

Tabela 3.3: Rezultati stiskanja po metodi Huffman

Datoteka	Velikost stisnjene datoteke v zlogih	Čas kodiranja (ms)	Čas dekodiranja (ms)	Entropija stisnjene datoteke	Razmerje stiskanja
LS1.TXT	27.636	18	4	7,87	0,58273
LS2.TXT	61.698	32	8	7,87	0,58140
LS3.TXT	120.900	64	16	7,85	0,57445
LS4.TXT	461.749	252	65	7,86	0,56935
ARND1.TXT	37.055	18	4	7,96	0,72373
ARND2.TXT	74.076	36	9	7,97	0,72340
ARND3.TXT	148.086	71	19	7,97	0,72308
RND1.TXT	51.575	21	6	8,00	1,00732
RND2.TXT	102.797	42	12	8,00	1,00388
RND3.TXT	205.244	84	24	8,00	1,00217
PS1.TXT	30.501	14	3	5,08	0,59572
PS2.TXT	61.222	28	6	5,17	0,59787
PS3.TXT	122.662	53	12	5,17	0,59894
BIN1.EXE	2.385.857	1.131	300	7,91	0,81308
BIN2.EXE	2.369.295	975	285	8,00	0,99990
english.dic	2.212.608	1.114	304	7,81	0,54398
PIC1.BMP	134.080	57	16	7,96	0,88646
PIC2.BMP	61.133	39	7	7,09	0,40178
PIC3.JPG	868.190	372	106	7,98	0,98464
PIC4.BMP	45.797	24	5	6,20	0,47347
VID1.3gp	6.670.927	2.742	804	8,00	1,00053
VID2.3gp	18.675.582	7.745	2.262	8,00	1,00050
ABAAB.TXT	6	0	0	2,58	1,20000

3.3 Metoda Huffman s prilagajanjem

Tudi Huffmanova metoda s prilagajanjem je večino datotek uspešno stisnila in sicer 18 od 23 datotek (≈ 78%). V primerjavi s klasično Huffmanovo metodo je 12 datotek stisnila slabše, 11 pa bolje. Vendar so razlike ponovno minimalne, saj se velikosti stisnjenih datotek razlikujejo le za nekaj deset zlogov (+/-). Je edina metoda, s katero smo uspešno stisnili tudi zelo kratko datoteko ABAAB.txt, saj kodirniku ni bilo potrebno shraniti Huffmanovega drevesa v stisnjeno datoteko.

Pri tej metodi pa opazimo občutno povečanje časa, ki ga ta metode potrebuje za kodiranje oz. dekodiranje datotek, kar je posledica nenehnega ažuriranja Huffmanovega drevesa.

Tabela 3.4: Rezultati stiskanja po metodi Huffman s prilagajanjem

Datoteka	Velikost stisnjene datoteke v zlogih	Čas kodiranja (ms)	Čas dekodiranja (ms)	Entropija stisnjene datoteke	Razmerje stiskanja
LS1.TXT	27.619	25	11	7,92	0,58237
LS2.TXT	61.692	47	23	7,92	0,58135
LS3.TXT	120.905	100	45	7,91	0,57447
LS4.TXT	461.739	358	173	7,90	0,56934
ARND1.TXT	37.099	30	16	7,96	0,72459
ARND2.TXT	74.147	60	31	7,97	0,72409
ARND3.TXT	148.192	113	58	7,97	0,72359
RND1.TXT	51.514	128	112	8,00	1,00613
RND2.TXT	102.739	235	194	8,00	1,00331
RND3.TXT	205.187	435	348	8,00	1,00189
PS1.TXT	30.491	19	9	5,72	0,59553
PS2.TXT	61.213	39	18	5,84	0,59778
PS3.TXT	122.655	77	36	5,80	0,59890
BIN1.EXE	2.385.962	1.784	913	7,96	0,81312
BIN2.EXE	2.369.318	3.402	2.509	8,00	0,99991
english.dic	2.212.627	1.627	804	7,90	0,54399
PIC1.BMP	134.127	120	76	7,98	0,88677
PIC2.BMP	61.158	92	60	7,16	0,40195
PIC3.JPG	868.275	787	484	7,99	0,98473
PIC4.BMP	45.796	43	23	6,81	0,47346
VID1.3gp	6.671.279	7.505	5.162	8,00	1,00058
VID2.3gp	18.675.907	17.100	10.758	8,00	1,00052
ABAAB.TXT	4	0	0	2,00	0,80000

4 SKLEP

S to diplomsko nalogo smo predstavili metode stiskanja podatkov s kodami spremenljivih dolžin. Osredotočili smo se na metode Shannon-Fano, Huffman in Huffman s prilagajanjem. V praktičnem delu diplomske naloge smo vse tri metode implementirali in primerjali njihovo učinkovitost. Izkazalo se je, da med metodami ni bistvenih razlik oz. so le-te minimalne, če jih primerjamo glede na razmerje stiskanja. Vse metode učinkovito stisnejo datoteke, ki vsebujejo vsaj nekaj odvečne informacije oz. redundance. Večja kot je redundanca v izvorni datoteki, boljše je razmerje stiskanja in obratno.

Klasična Huffmanova metoda se je na testiranih datotekah izkazala kot najboljša, tako po času stiskanja in razširjanja ter po velikosti stisnjenih datotek. Sledi ji Huffmanova metoda s prilagajanjem, ki je nekatere datoteke bolje stisnila kot klasična, vendar je bila skupna velikost vseh stisnjenih testiranih datotek večja. Zaradi narave delovanja Huffmanove metode s prilagajanjem je le-ta za stiskanje in razširjanje podatkov porabila največ časa izmed vseh testiranih metod. Gledano v celoti pa je dosegla enako razmerje stiskanja kot klasična Huffmanov metoda, zato jo lahko uporabimo v primerih, kjer klasične Huffmanove metode ne moremo (za stiskanje podatkov v realnem času, kjer vnaprej nimamo dostopa do statističnega modela). Izmed vseh testiranih metod je metoda Shannon-Fano najslabše stisnila podatke, tako da je bolje uporabiti eno izmed Huffmanovih metod.

5 VIRI IN LITERATURA

[1] Salomon, D., Data Compression: The Complete Reference, 4th edition. London: Springer, 2007
[2] Wikipedia – ASCII. Dostopno na https://en.wikipedia.org/wiki/ASCII [01.08.2016]
[3] Wikipedia – Frekvence črk v slovenskem jeziku. Dostopno na https://sl.wikipedia.org/wiki/Frekvence_%C4%8Drk [01.08.2016]
[4] C# Language Reference. Dostopno na https://msdn.microsoft.com/en-us/library/618ayhy6.aspx [01.08.2016]
[5] Wikipedia – Shannon-Fano coding. Dostopno na http://en.wikipedia.org/wiki/Shannon%E2%80%93Fano_coding [01.08.2016]
[6] Žalik, B., Zapiski predavanj pri predmetu Multimedia in navidezna resničnost, UM FERI, 2004
[7] Wikipedia – Huffman coding. Dostopno na http://en.wikipedia.org/wiki/Huffman_coding [01.08.2016]
[8] Wikipedia – Priority queue. Dostopno na https://en.wikipedia.org/wiki/Priority_queue [01.08.2016]
[9] Wikipedia – Adaptive Huffman coding. Dostopno na http://en.wikipedia.org/wiki/Adaptive_Huffman_coding [01.08.2016]
[10] Slovenska leposlovna besedila. Dostopno na https://github.com/SuhadolcBarbara/Statisticna-Analiza-Slovenskih-Besedil/ [01.08.2016]

6 PRILOGE

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

IZJAVA O AVTORSTVU

Spodaj podpisani/-a ROBERT ŠIRCELJ

z vpisno številko 93508992

sem avtor/-ica diplomskega dela z naslovom:

PRIMERJAVA ALGORITMOV STISKANJA PO METODAH SHANNON-FANO,
HUFFMAN IN HUFFMAN S PRILAGAJANJEM
(naslov diplomskega dela)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

red. prof. dr. BORUT ŽALIK

in somentorstvom (naziv, ime in priimek)

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela.
- soglašam z javno objavo elektronske oblike diplomskega dela v DKUM.

V Mariboru, dne 18.08.2016

Podpis avtorja/-ice:

Univerza v Mariboru

FERI

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

IZJAVA O USTREZNOSTI ZAKLJUČNEGA DELA

Podpisani mentor :

red. prof. dr. BORUT ŽALIK

(ime in priimek mentorja)

in somentor (eden ali več, če obstajata):

_____

(ime in priimek somentorja)

Izjavljam (-va), da je študent

Ime in priimek: ROBERT ŠIRCELJ

Vpisna številka: 93508992

Na programu: FERI-RI VS Programska oprema

izdelal zaključno delo z naslovom:

PRIMERJAVA ALGORITMOV STISKANJA PO METODAH SHANNON-FANO, HUFFMAN IN HUFFMAN S PRILAGAJANJEM

(naslov zaključnega dela v slovenskem in angleškem jeziku)

COMPARISON OF SHANNON-FANO, HUFFMAN, AND ADAPTIVE HUFFMAN COMPRESSION ALGORITHM METHODS

v skladu z odobreno temo zaključnega dela, Navodilih o pripravi zaključnih del in mojimi (najinimi oziroma našimi) navodili.

Preveril (-a, -i) in pregledal (-a, -i) sem (sva, smo) poročilo o plagiatorstvu.

Datum in kraj: 18.8.2016, Maribor

Podpis mentorja: B. Žalik

Datum in kraj:

Podpis somentorja (če obstaja):

Priloga:

- Poročilo o preverjanju podobnosti z drugimi deli.

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE ZAKLJUČNEGA DELA IN OBJAVI OSEBNIH PODATKOV DIPLOMANTOV

Ime in priimek avtorja-ice: ROBERT ŠIRCELJ

Vpisna številka: 93508992

Študijski program: FERI-RI VS PROGRAMSKA OPREMA

Naslov zaključnega dela: PRIMERJAVA ALGORITMOV STISKANJA PO METODAH SHANNON-FANO, HUFFMAN IN HUFFMAN S PRILAGAJANJEM

Mentor: red. prof. dr. BORUT ŽALIK

Somentor: _____

Podpisani-a ROBERT ŠIRCELJ izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru. Zaključno delo sem izdelal-a sam-a ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija zaključnega dela je istovetna z elektronsko verzijo elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru.

Zaključno delo zaradi zagotavljanja konkurenčne prednosti, varstva industrijske lastnine ali tajnosti podatkov naročnika: _____ ne sme biti javno dostopno do _____(datum odloga javne objave ne sme biti daljši kot 3 leta od zagovora dela).

Podpisani izjavljam, da dovoljujem objavo osebnih podatkov, vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum zaključka študija, naslov zaključnega dela), na spletnih straneh in v publikacijah UM.

Datum in kraj: 18.8.2016, Maribor Podpis avtorja-ice: R Šircelj

Podpis mentorja: _____
(samo v primeru, če delo ne sme biti javno dostopno)

Podpis odgovorne osebe naročnika in žig:
(samo v primeru, če delo ne sme biti javno dostopno) _____