Odabrani algoritmi sažimanja

Kuljiš, Petra

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:547592

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

DIGITALNA KNJIŽNICA

Image not found or type unknown

Sveučilište u Rijeci – Odjel za informatiku

Jednopredmetni preddiplomski studij informatike

Petra Kuljiš

Odabrani algoritmi sažimanja

Završni rad

Mentorica: doc. dr. sc. Marija Brkić Bakarić

Rijeka, rujna 2020

Sadržaj

Sažetak ... 4
Ključne riječi ... 4
1. Uvod ... 5
2. Sažimanje podataka ... 6
3. Sažimanje bez gubitka ... 8
3.1. Shannon-Fanov algoritam ... 9
3.2. Huffmanov algoritam ... 10
3.2.1. Prošireni Huffmanov algoritam ... 13
3.3. LZ77 ... 15
3.4. LZ78 ... 18
3.5. LZW algoritam ... 20
4. Primjena ... 22
4.1. Sažimanje fotografija ... 22
4.2. Sažimanje video zapisa ... 24
4.3. Sažimanje zvučnih zapisa ... 24
Zaključak ... 25
Literatura ... 26

Sažetak

U radu se opisuju procesi za sažimanje podataka, oni sa i bez gubitka. Detaljnije se analiziraju neki od algoritama za sažimanje bez gubitaka, Shannon-Fanov, Huffmanov, LZ77 i LZ78, te su za svaki dani primjeri za lakše razumijevanje. Ti algoritmi veoma su rasprostranjeni te su opisane neke od najčešćih primjena u današnje vrijeme.

Ključne riječi: sažimanje podataka, kompresija, algoritmi za sažimanje podataka bez gubitaka, Shannon-Fanov algoritam, Huffmanov algoritam, LZ77, LZ78

1. Uvod

Od mp3 uređaja za slušanje glazbe, preko mobilnih telefona, DVD-a, do digitalnih televizija, sažimanje (ili kompresija) podataka sastavni je dio gotovo svih informacijskih tehnologija. Dugo je sažimanje koristila samo mala grupa inženjera i znanstvenika, ali sada je sveprisutno. Koristiti ćemo sažimanje želimo li prebaciti program na televiziji, poslati poruku prijatelju na mobilnom telefonu, razgovarati na daljinu, koristimo li se modemom ili odlučimo li se za slanje dokumenta putem faks uređaja. Uključivanje kompresije u sve više sfera naših života ukazuje na određeni stupanj sazrijevanja tehnologije.

Algoritmima za sažimanje podataka koristimo se kako bismo smanjili broj bitova koji su potrebni za prikaz neke slike, videa ili glazbe, tj. informacije zapisujemo u nekom kompaktnom obliku.

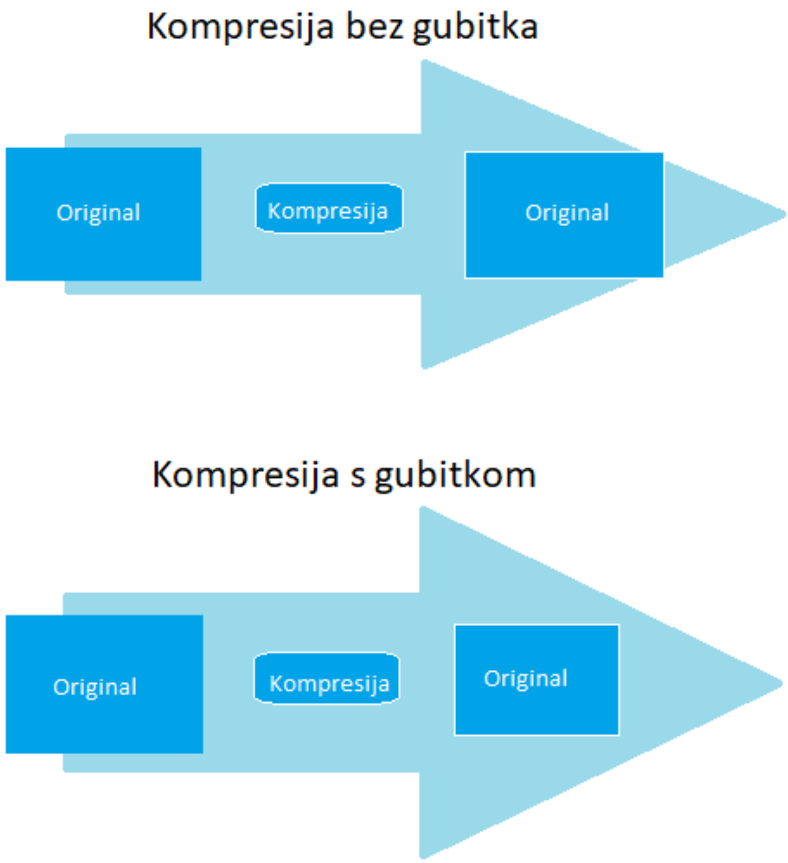
Sažimanje podataka u današnje vrijeme nam je veoma potrebno zbog toga što sve više informacija koje generiramo i koristimo su u digitalnom obliku, tj. u obliku brojeva koji predstavljaju bajtove podataka. A broj bajtova koji su potrebni za zapisivanje multimedijskih podataka može biti ogroman. Na primjer, za digitalni zapis jedne sekunde videa bez sažimanja treba više od 20 megabajta, ili 160 megabita, a za dvije minute nekomprimirane muzike kvalitete za CD potrebno je više od 84 milijuna bitova. Preuzimanje te muzike s neke web stranice bi trajalo podulje.

Iako se skladište za pohranu i brzina prijenosa podataka svakom novom tehnološkom inovacijom povećava, gotovo dvostruko brže raste potreba za masovnom pohranom i prijenosom podataka.

2. Sažimanje podataka

Sažimanje podataka je proces u kojem se smanjuje broj bitova koji su potrebni za zapis objekta u memoriji računala. Ono se sastoji od dva procesa, prvog, gdje se podaci komprimiraju ili kodiraju kako bi smanjili njihovu veličinu, i drugog, u kojemu se podatak dekomprimira ili dekodira kako bi ga vratili u prvobitno stanje.

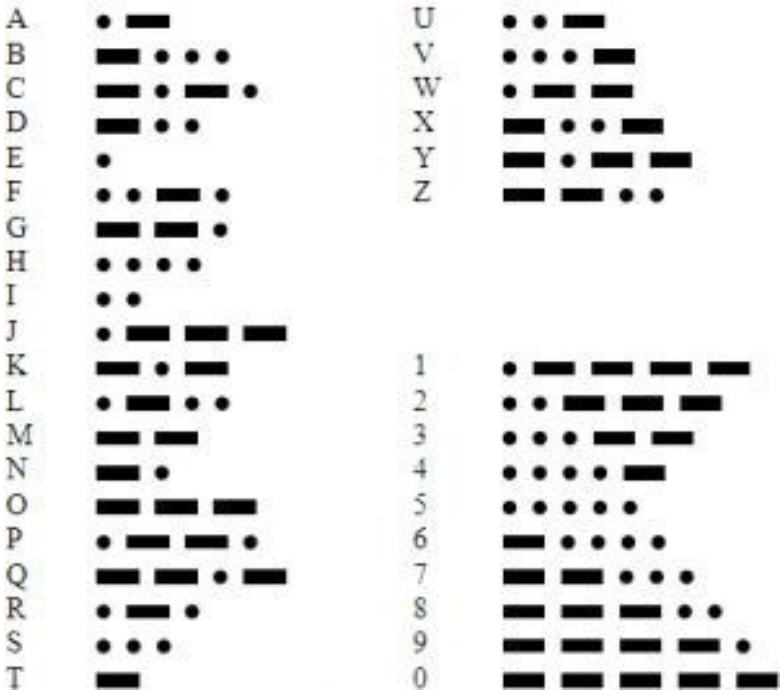
Dalje, metode sažimanja podataka možemo podijeliti u dvije skupine, ili dva osnovna načina sažimanja, kompresija s gubitkom (*lossy*) i kompresija bez gubitaka (*lossless*) (Slika 1). Pri kompresiji podataka sa gubitkom gube se neki podaci ili karakteristike izvornog nekodiranog objekta, koji se kasnije ne mogu oporaviti ili rekonstruirati. Možemo reći da se smanjuje kvaliteta medija kako bismo sadržaj učinili što pogodnijim za sažimanje.



Slika 1 Sažimanje podataka sa i bez gubitka[1]

[1] Slika prilagođena s https://gisgeography.com/lossless-compression-vs-lossy-compression/

Jedan od prvih, i jednostavnijih, primjera sažimanja podataka je Morseov kod, koji su početkom 19. stoljeća izmislili Samuel Morse, fizičar Joseph Henry i strojar Alfred Vail. Iako je Morseov kod trebao biti namijenjen za slanje poruka putem radio valova, uzorci koje koristi možemo identificirati kao metodu za sažimanje podataka. Morseov kod dizajniran je za slanje tekstualnih datoteka telegrafskim žicama. Telegraf je vrlo jednostavan stroj, na jednom kraju ima prekidač, koji kad ga se zatvori proizvodi električnu struju koja putuje kroz žicu na drugi kraj, gdje se nalazi elektromagnet zvani relej koji proizvodi zvuk. Pisma koja su se slala telegrafom kodirala su se točkama i crtama (Slika 2). Kako bi smanjio vrijeme slanja poruke, slovima koja su se češće pojavljivala dodijelio je kraći zapis (npr. 'e' je '·'), a slovima koja se nisu toliko često pojavljivala dodijelio je dulje zapise ('q' se prevodi u '--·-').



Slika 2 Morseova abeceda[2]

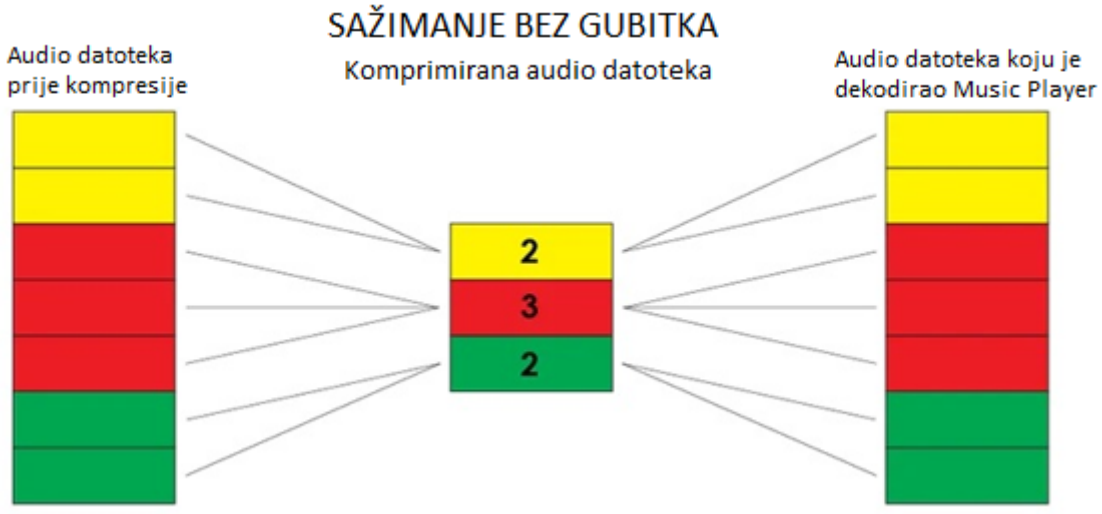
[2] Slika preuzeta s https://www.labin.com/lcplus/morseova-abeceda-6671

3. Sažimanje bez gubitka

U ovom radu, fokusirati ćemo se na sažimanje podataka u kojem ne dolazi do gubitka informacija, te u kojem se svi izvorni podaci nakon dekompresije vraćaju u isto stanje u kojem su bili prije sažimanja (Slika 3). Sažimanje bez gubitka se najviše koristi za baze podataka, datoteke za obradu teksta, ili čak za neke vrste podataka o slikama i video zapisima.

U mnogim situacijama se zahtjeva da rekonstruirani podatak nakon dekompresije bude identičan originalu. Kao na primjer u slučaju sažimanja tekstova, izrazito je bitno da je rekonstrukcija identična originalnom tekstu, jer u nekim slučajevima male razlike mogu dovesti do potpuno drugog značenja. Takav oblik sažimanja također se koristi kod aplikacija koje ne toleriraju razliku između originalnih i rekonstruiranih podataka.

Sažimanje podataka bez gubitaka je proces uklanjanja zalihosti bez gubitaka informacija gdje se izvorna poruka zbija u učinkovitiji prikaz koji je jednakovrijedan izvornom. Samim time, postupci koji spadaju u ovu kategoriju su reverzibilni, tj. svakome pripada određen obratan postupak, koji iz sažete poruke može stvoriti izvornu.



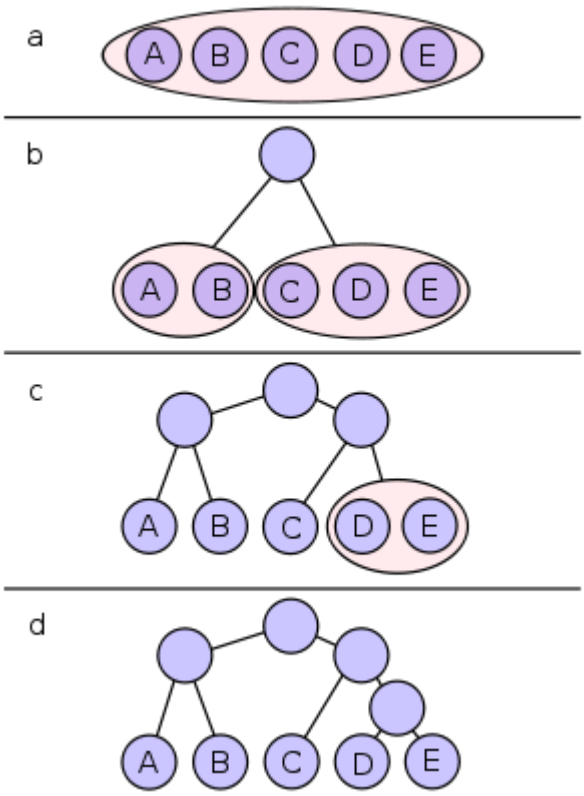
Slika 3 Primjer sažimanja bez gubitka[3]

[3] Slika prilagođena s https://www.retromanufacturing.com/blogs/news/understanding-audio-file-formats-flac-wma-mp3

3.1. Shannon-Fanov algoritam

Claude Shannon izumio je način mjerenja informacijskog sadržaja poruke i nazvao je informacijskom entropijom. 1940-ih zajedno s Robertom Fanom osmišljava Shannon-Fanov algoritam, metodu konstruiranja prefiksnog koda koji se zasniva na setu simbola i njihovom vjerojatnošću da će se pojaviti. To je prvi kod s minimalnom redundancijom, tj. zalihošću.

Postupak rješavanja započinjemo radeći listu vjerojatnosti za neke zadane simbole, a poredamo ih od onih koji se najčešće do onih koji se najmanje pojavljuju. Simbole podijelimo na dva segmenta, tako da im suma vjerojatnosti bude otprilike ista te se prvoj grupi dodijeli kod koji započinje s bitom 0, a drugoj kod s bitom 1. Zatim ta dva podsegmenta ponovno podijelimo na dva segmenta i ponovimo dodjeljivanje brojeva 0 i 1. Kad podsegment sadrži samo dva broja, prvom kodu dodamo bit 0 a drugom 1. Taj proces nastavljamo sve dok ne ostanemo bez segmenata (Slika 4).



Slika 4 Stablo Shannon-Fanovog algoritma[4]

[4] Slika preuzeta s https://en.wikipedia.org/wiki/Shannon%E2%80%93Fano_coding

Tablica 1 Primjer rješavanja Shannon-Fanovog kodiranja

Simbol	Vjerojatnost	1.korak	2.korak	3.korak	Kodna riječ
A	0.35	0	0		00
B	0.20	0	1		01
C	0.175	1	0		10
D	0.150	1	1	0	110
E	0.125	1	1	1	111

```
 1:  begin
 2:     count source units
 3:     sort source units to non-decreasing order
 4:     SF-SplitS
 5:     output(count of symbols, encoded tree, symbols)
 6:     write output
 7:   end
 8:
 9:  procedure SF-Split(S)
10:  begin
11:     if (|S|>1) then
12:      begin
13:        divide S to S1 and S2 with about same count of units
14:        add 1 to codes in S1
15:        add 0 to codes in S2
16:        SF-Split(S1)
17:        SF-Split(S2)
18:      end
19:  end
```

Slika 5 Pseudokod Shanon-Fanovog algoritma[5]

3.2. Huffmanov algoritam

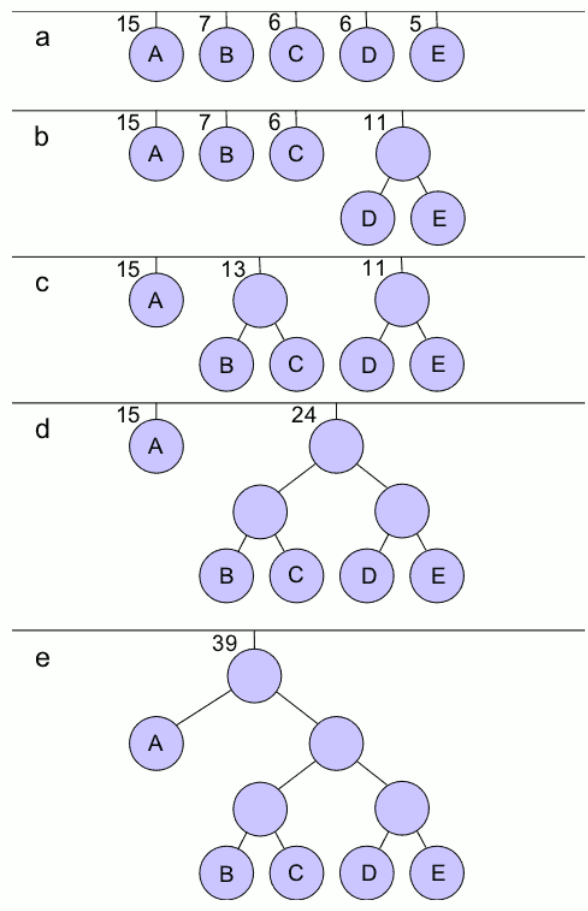
1952. godine D.A. Huffman objavljuje rad u kojem opisuje metodu za sažimanje podataka koja koristi Huffmanovo stablo za komprimiranje skupa podataka, kojim je poboljšao Shannon-Fanov algoritam. Umjesto od tada standardne konstrukcije binarnog stabla od gore prema dole, kao u slučaju Shannon-Fanovog kodiranja, Hauffmanov kod konstruira binarno stablo od dolje

[5] Pseudokod preuzet s http://www.stringology.org/DataCompression/sf/index_en.html

prema gore, koje nazivamo Huffmanovo stablo. Huffmanovo stablo je binarno stablo koje određuje najbolji način dodjeljivanja kodova simbolima u podacima ovisno o frekvencijama pojavljivanja pojedinih simbola. Simbolima koji se češće pojavljuju dodjeljuju se kraći kodovi od onih koji se dodjeljuju simbolima koji se ne pojavljuju toliko često. Upravo zbog toga Huffmanovo kodiranje ima vrlo dobru kompresiju te je izrazito brzo pri komprimiranju i dekomprimiranju.

Za provođenje Huffmanovog kodiranja, kreiramo listu simbola (grane binarnog stabla), koji su poredani od onih s najvećom do onih s najmanjom vjerojatnošću. Odabiremo dva simbola s najmanjom vjerojatnošću, kreiramo binarno stablo čiji je korijen novonastali simbol koji smo dobili sjedinjujući dva izvorna simbola. U listi dva izvorna simbola zamijenimo sa binarnim stablom novonastalog čvora čiji je korijen zbroj vrijednosti Postupak nastavimo tako da od ostalih simbola i novonastalog čvora ponovno biramo dva s najmanjim vjerojatnostima i ponovimo proces, sve dok ne ostane samo jedan „nadsimbol“ (Slika 6).

Za dobivanje koda Huffmanovog stabla, krećemo od korijena kojem dodjeljujemo bit 0. Prelazimo na njegovu djecu, te lijevom dodjeljujemo ponovno bit 0, a desnom 1 i tako sve dok nismo prošli sve čvorove.



Slika 6 Konstrukcija Huffmanovog stabla[6]

Tablica 2 Izračunati Huffmanov kod na primjeru slike

Simbol	Broj pojavljivanja	Huffmanov kod
A	15	1
B	7	100
C	6	101
D	6	110
E	5	111

Huffmanovo kodiranje nije uvijek optimalno među svim metodama kompresije. U slučaju da je potreban bolji omjer kompresije često se zamjenjuje aritmetičkim kodiranjem ili asimetričnim brojevnim sustavom.

[6] Slika preuzeta s https://www.wikiwand.com/en/Huffman_coding

```
 1:  begin
 2:     count frequencies of single characters (source units)
 3:     output(frequencies using Fibonacci Codes of degree 2)
 4:     sort them to non-decreasing sequence
 5:     create a leaf node (character, frequency c, left son = NULL, right son = NULL)
 6:        of the tree for each character and put nodes into queue F
 7:     while (|F|>=2) do
 8:      begin
 9:        pop the first two nodes (u1, u2) with the lowest
10:           frequencies from sorted queue
11:        create a node evaluated with sum of the chosen units,
12:           successors are chosen units (eps, c(u1)+c(u2), u1, u2)
13:        insert new node into queue
14:      end
15:     node evaluate with way from root to leaf node (left son 1, right son 0)
16:     create output from coded intput characters
17:  end
```

Slika 7 Pseudokod Huffmanovog algoritma[7]

3.2.1. Prošireni Huffmanov algoritam

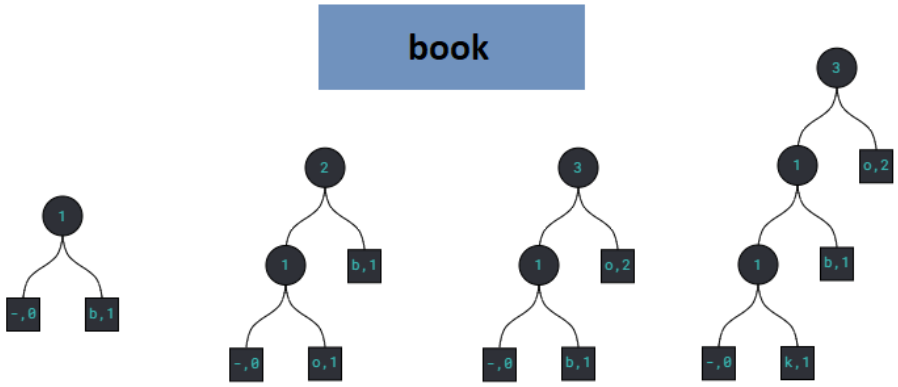
Huffmanovo kodiranje zahtjeva poznavanje vjerojatnosti zadanih sekvenci, no kako ono često nije dostupno, Huffmanovo kodiranje se odvija u dva koraka. Prvi korak je prikupljanje statistike, a drugi kodiranje izvora. Da bi algoritam imao jedan korak, Newton Faller i Robert Gray Gallagher, kasnije i Donald Ervin Knuth i Jeffrey Scott Vitter, razvijaju proširene (adaptivne) algoritme za konstrukciju Huffmanovog koda u jednom koraku.

S proširenim Huffmanovim kodiranjem, naša svrha i cilj su identični kao i kod Huffmanovog algoritma, izgradnja stabla koje će nam dati optimalnu binarnu shemu kodiranja. Glavna razlika je što nećemo unaprijed obraditi podatke koji se unose, već se stablo izgrađuje dok se čita tekst. Za pojašnjenje proširenog Huffmanovog algoritma, koristit ćemo FGK (Faller-Gallager-Knuth) algoritam. Ono ima dva pravila: svi čvorovi, osim korijena, moraju imati brata/sestru te se svi čvorovi redaju po povećanju slijeva nadesno i odozdo prema gore. Druga velika razlika od tradicionalnog Huffmanovog stabla je što FGK stablo ubacuje nulti čvor.

Glavna ideja je da istodobno čitamo ulaz i gradimo stablo. Za razliku od tradicionalnog Huffmanovog stabla, FGK stablo gradimo odozgo prema dolje, tj. započinjemo s korijenom i

[7] Pseudokod preuzet s http://www.stringology.org/DataCompression/sh/index_en.html

gradimo do listova. Nulti čvor koristimo kao brata/sestru za sve nove čvorove koje dodajemo, a stablo modificiramo ovisno o simbolu koji čitamo. Ako je simbol novi dodajemo novi čvor na stablu, a ako se ponavlja ažuriramo njegovu frekvenciju. Stablo koje dobijemo kao krajnji rezultat koristimo kako bismo odredili odgovarajući binarni kod za simbole (Slika 8).



Slika 8 Primjer proširenog Huffmanovog koda[8]

[8] Slika prilagođena s http://ben-tanen.com/adaptive-huffman/

```
 1:  begin
 2:     create node ZERO
 3:     readSymbol(X)
 4:     while (X!=EOF) do
 5:      begin
 6:        if (first_read_of(X)) then
 7:         begin
 8:           output(ZERO)
 9:           output(X)
10:           create new node U with next nodes ZERO and new node X
11:           update_tree(U);
12:         end
13:        else
14:         begin
15:           output(X)
16:           update_tree(X)
17:         end
18:        readSymbol(X)
19:      end
20:  end
21:
22:  procedure update_tree(U)
23:  begin
24:     while (U!=root) do
25:      begin
26:        if (exists node U1 with same value and greater order) then
27:           change U1 and U
28:        increment value of U
29:        U := parent(U)
30:      end
31:     increment value of U, update leaf codes
32:  end
```

Slika 9 Pseudokod Faller-Gallager-Knuth algoritma[9]

3.3. LZ77

Istraživanja na polju kompresije su se temeljila na poboljšanju Huffmanovog algoritma za kodiranje podataka sve dok 1977. godine istraživači Abraham Lempel i Jacob Ziv nisu kreirali

[9] Pseudokod preuzet s http://www.stringology.org/DataCompression/fgk/index_en.html

Lempel-Ziv algoritme za kompresiju bez gubitaka. Važno je razumjeti da simbol nije nužno samo jedan znak u tekstu, već može biti podatak bilo koje količine, ali najčešće je veličine jednog bajta.

Svoju metodu komprimiranja podataka Lempel i Ziv predstavili su u nekoliko varijanti, a dvije najpoznatije su LZ77 i LZ78, a temelje se na ideji kreiranja rječnika od podataka koji su procesirani i zamjenjivanja simbola (ali može biti i neki niz simbola) indeksom iz rječnika. Tim načinom u komprimiranom nizu podataka pohranjujemo samo pokazivače prema frazama koje se nalaze u rječniku.

Prva verzija algoritma je LZ77 (Lempel-Ziv 77, poznat i kao LZ1), gdje je rječnik dio prethodno kodirane sekvence čiji su ulazi predefinirani i konstantni tijekom komprimiranja. U ovoj verziji koder istražuje ulaznu sekvencu koja prolazi kroz klizni prozor koji se sastoji od dva dijela, *buffera* za pretraživanje (trenutni rječnik koji uključuje simbole koji su nedavno bili uneseni i kodirani) i *look-ahead buffer* (međuspremnik koji sadrži ostatak teksta koji se tek treba kodirati). Ulazna sekvenca se kreće kroz klizni prozor s lijeva nadesno kako se *stringovi* prevode. Da bi koder kodirao sekvencu koja se nalazi u *look-ahead bufferu*, koder pomiče pokazivač za pretraživanje natrag kroz buffer za pretraživanje dok ne naiđe na podudaranje s prvim simbolom u *look-ahead bufferu*. Udaljenost pokazivača od *look-ahead buffer* se naziva pomak. Koder zatim istražuje simbole koji se nalaze nakon simbola koji je na mjestu pokazivača da bi utvrdio podudaraju li se s uzastopnim simbolima u *look-ahead bufferu*. Broj uzastopnih simbola u bufferu za pretraživanje koji se podudaraju sa simbolima u *look-ahead bufferu*, počevši s prvim simbolom, se naziva duljina podudarnosti. Koder pretražuje *buffer* za pretraživanje za najdulje podudaranje, i nakon što ga pronađe, koder ga kodira s (o, l, c), gdje je o pomak, l je duljina podudarnosti i c kodna riječ koja pripada *look-ahead bufferu* (Slika 10). Problem LZ77 algoritma je što bi referenca iz rječnika mogla biti i duža od *stringa* koji zamjenjuje.

ACTGACTACACACTAGCTACG$

PROZOR ZA TRAŽENJE							PROZOR ZA KODIRANJE					OSTATAK	IZLAZ	KORAK
						←	A	C	T	G	A	CTA…	(0, 0, A)	1
					←	A	C	T	G	A	C	TAC…	(0, 0, C)	2
				←	A	C	T	G	A	C	T	ACA..	(0, 0, T)	3
			←	A	C	T	G	A	C	T	A	CAC…	(0, 0, G)	4
		←	A	C	T	G	A	C	T	A	C	ACA..	(4, 3, A)	5
C	T	G	A	C	T	A	C	A	C	A	C	TAG…	(3, 1, A)	6
G	A	C	T	A	C	A	C	A	C	T	A	GCT…	(2, 3, T)	7
A	C	A	C	A	C	T	A	G	C	T	A	CG$…	(3, 1, G)	8
A	C	A	C	T	A	G	C	T	A	C	G	$	(4, 3, C)	9
T	A	G	C	T	A	C	G	$					(5, 1, $)	10
G	C	T	A	C	G	$								

Slika 10 Primjer LZ77 kompresije niza ACTGACTACACACTAGCTACG$[10]

Proces komprimiranja podataka koristeći se LZ77 algoritmom je veoma dugotrajan jer se dosta vremena provodi pretražujući pomični prozor za identične fraze. LZ77 ima dulje vrijeme komprimiranja, ali bolji omjer kompresije od Huffmanovog, dok je dekomprimiranje još brže nego kod Huffmanovog.

```
 1:  begin
 2:      fill view from input
 3:      while (view not empty) do
 4:       begin
 5:         find longest prefix p of view starting in coded part
 6:         i := position of p in window
 7:         j := length of p
 8:         X := first char after p in view
 9:         output(i,j,X)
10:         add j+1 chars
11:       end
12:  end
```

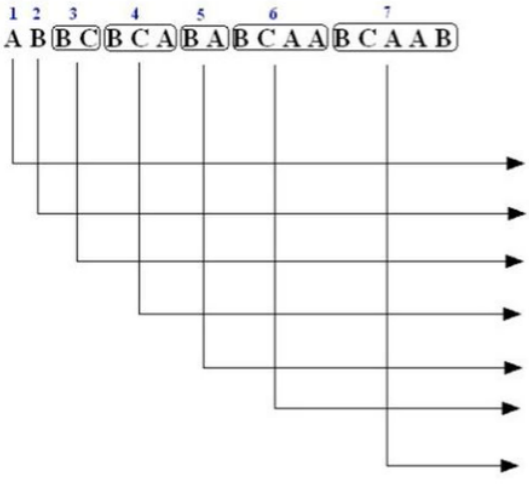
Slika 11 Pseudokod algoritma LZ77[11]

[10] Slika preuzeta s https://bib.irb.hr/datoteka/809633.Zavrni_Zoran_Krito.pdf
[11] Pseudokod preuzet s http://www.stringology.org/DataCompression/lz77/index_en.html

3.4. LZ78

Kako bi se problemi koji su se pojavljivali kod LZ77 izbjegli, Jacob Ziv i Abraham Lampel su razvili drugačiji oblik kompresije podataka bez gubitka informacija. LZ78 je objavljen 1978. godine. Iako je u početku bio popularan, popularnost LZ78 kasnije je umanjena, a najpopularnija modifikacija LZ78 je LZW koju je izradio Terry Welche 1984. godine. LZW algoritam ima predefinirane vrijednosti izraza u svom rječniku prilikom inicijalizacije, stoga se rječnik sastoji od svih mogućih kombinacija izraza za dati opseg. Umjesto čitanja nekoliko simbola i pretrage kroz tekst pojavljuju li se duplikati, LZ algoritmi za nađeni izraz traže je li pronađen ranije. Na početku procesa baza je manja te će većina izraza biti nova, a kasnije će biti više podudaranja u nađenim izrazima.

LZ78 algoritam rječnik izrađuje dinamički, koristeći se samo nekomprimiranim sadržajem. Osnovni princip rada je da kompresija, koja je bazirana na rječniku, koristi listu ili rječnik u kojem pamti izraze nekomprimiranog sadržaja. Kada se izraz iz rječnika pojavi u nekomprimiranom sadržaju za vrijeme kompresije, on se mijenja sa kodom iz rječnika koji ga referencira. Kompresija se povećava sa povećanjem dužine izraza u rječniku i povećanjem frekvencije njihovih pojavljivanja u nekomprimiranom sadržaju. Detaljnije objašnjeno, sheme temeljene na LZ78 djeluju unošenjem izraza u rječnik, te kad se pronađe ponovljena pojava tog određenog izraza dobiva se indeks izraza u rječniku. Svaki korak LZ78 algoritma šalje se (i,a) par izlazu, gdje i označava indeks izraza u rječniku, dok a označava idući simbol nakon detektiranog izraza. Rječnik se prikazuje kao stablo sa numeriranim čvorovima. Ako se kreće od korijena prema određenom čvoru, dobiva se izraz iz ulaznog teksta. U svakom koraku traži se najdulji izraz iz rječnika, koji odgovara neobrađenom djelu ulaznog teksta. Indeks tog izraza zajedno sa simbolom koji slijedi šalju se na izlaz. Prethodni izraz proširen s novim simbolom je zatim zapisan u rječnik, te je taj novi izraz numeriran najmanjim mogućim brojem. Kodiranje počinje sa stablom koje ima samo jedan čvor koji prezentira prazan *string*. Problematika do koje se dolazi je brzo iskorištenje dostupne memorije zbog rasta rječnika bez granica jer se ništa iz njega ne briše, te bi se u praksi rast rječnika morao zaustaviti u nekom stadiju (Slika 12).



	Dictionary	
output	index	string
(0, A)	1	A
(0, B)	2	B
(2, C)	3	BC
(3, A)	4	BCA
(2, A)	5	BA
(4, A)	6	BCAA
(6, B)	7	BCAAB

Slika 12 LZ78 kodiranje[12]

```
1:  begin
2:      initialize a dictionary by empty phrase P
3:      while (not EOF) do
4:        begin
5:          readSymbol(X)
6:          if (F.X> is in the dictionary) then
7:            F = F.X
8:          else
9:           begin
10:            output(pointer(F),X)
11:            encode X to the dictionary
12:            initialize phrase F by empty character
13:           end
14:       end
15:  end
16:
17:  DECODING
18:  begin
19:      initialize a  dictionary by empty phrase
20:      while (not EOF) do
21:        begin
22:          read pair of index and character (i,X) from input
23:          put new phrase string(i).X into dictionary
24:          generate phrase to the output
25:        end
26:  end
```

Slika 13 Pseudokod algoritma LZ78[13]

[12] Slika preuzeta s http://site.iugaza.edu.ps/jroumy/files/Unit31_LZ78.pdf
[13] Pseudokod prilagođen s http://www.stringology.org/DataCompression/lz78/index_en.html

3.5. LZW algoritam

Popularna verzija LZ78, koju je 1984. modificirao Terry Welch, jedna je od najvažnijih tehnika za kompresiju podataka za opće namjene. LZW najčešće komprimira tekst, izvršne kodove, i slične podatkovne datoteke na približno polovicu od njihove originalne veličine.

Kao i svaka metoda dinamičke kompresije ideja je započeti s početnim modelom, čitati podatke dio po dio i ažurirati model i kodirati redom podatke. LZW algoritam je, kao i LZ77 i LZ78, zasnovan na rječniku. Za kodiranje je znači potrebno u izlaznu datoteku zapisati samo jedan kodni broj koji odgovara indeksu tog *stringa* u rječniku.

LZW započinje inicijalizacijom rječnika na sve standardne ASCII kodove, što znači da je prvih 256 unosa rječnika zauzeto prije nego što se upišu bilo koji podaci. Princip LZW-a je da koder unosi simbole jedan po jedan i zapisuje ih u niz I. Nakon što se svaki simbol unese i integrira u I, rječnik se pretražuje za *stringom* I. Dokle god se *string* I nađe u rječniku, proces se nastavlja. No, ako se doda simbol x, i u rječniku ne nalazi *string* koji se sastoji od Ix, ono ga zapisuje u rječnik na prvo slijedeće slobodno mjesto u rječniku, te *string* I postaje x.

Tablica 3 Primjer LZW algoritma za ulazni string /WED/WE/WEE/WEB/WET

Ulaz	Kodirani izlaz	Indeks	Novi string
/W	/	256	/W
E	W	257	WE
D	E	258	ED
/	D	259	D/
WE	256	260	/WE
/	E	261	E/
WEE	260	262	/WEE
/W	261	263	E/W
EB	257	264	WEB
/	B	265	B/
WET	260	266	/WET
EOF	T		

Dekoder započinje s prvih 256 kodova koji su već definirani za prevođenje u nizove s jednim znakom. Kad prijeđe na nepoznate *stringove*, pročita njegov indeks, pretražuje ga u rječniku i izbacuje *substring* koji je povezan s indeksom. Prvi znak ovog *substringa* povezuje se s trenutnim radnim *stringom*. Taj novonastali *string* se dodaje u rječnik , a dekodirani *string* postaje trenutni radni *string* i proces se nastavlja.

```
1:  begin
2:     create_empty_dictionary(D)
3:     for each (a in ABECEDA) do
4:      begin
5:        add_to_dictionary(a,D)
6:      end
7:     S := ""
8:     while (!EOF) do
9:      begin
10:       readSymbol(z)
11:       if (find_in_dictionary(Sz,D)) then
12:          S := Sz
13:       else
14:        begin
15:          output(pointer(S,D))
16:          add_to_dictionary(Sz,D)
17:          S := z
18:        end
19:     end
20: end
```

```
1:  begin
2:     tmp = ""
3:     initialization of the dictionary
4:     while ((code = read code) != EOF) do
5:      begin
6:        if ( is phrase with code (code) in dictionary ) then
7:         begin
8:           phrase := dictionary[code]
9:           output(phrase)
10:          add to dictionary (tmp + first character of phrase)
11:        end
12:       else
13:        begin
14:          if ( code > maximum index of dictionary + 1 ) then
15:             exit
16:          phrase := tmp + first char of tmp
17:          output(phrase)
18:          add to dictionary (phrase)
19:        end
20:       tmp := phrase
21:     end
22: end
```

Slika 14 Pseudokod komprimiranja i dekomprimiranja LZW algoritmom[14]

[14] Pseudokodvi preuzeti s http://www.stringology.org/DataCompression/lzw-e/index_en.html i http://www.stringology.org/DataCompression/lzw-d/index_en.html

4. Primjena

Algoritmi kompresije bez gubitaka obično se koriste u arhivske svrhe ili u druge svrhe visoke vjernosti. Ovi algoritmi omogućuju smanjenje veličine datoteke, istovremeno osiguravajući da se datoteke mogu u potpunosti vratiti u prvobitno stanje ako je potrebno.

Kako društvo postaje sve složenije, stvara se potreba za sve bržom komunikacijom. 1980-ih je zabilježen ogroman rast u korištenju faks uređaja, a to ne bi bilo moguće bez tehnologija kompresije koje omogućuju brzi prijenos. Bez kompresije, slanje jedne stranice dokumenta trajalo bi preko 24 sata.

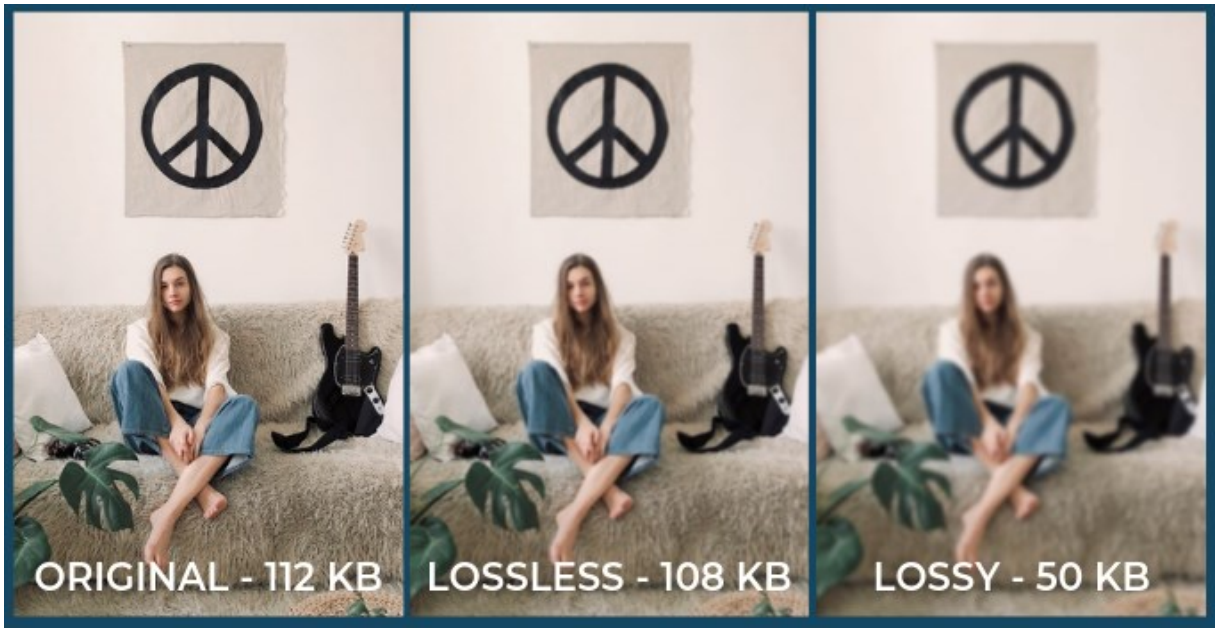
Kompresija je danas itekako dio svakodnevnog života. Velika je vjerojatnost da kad smo na računalu upotrebljavamo i razne proizvode koji se koriste kompresijom. Većina današnjih modema ima mogućnost kompresije koje nam omogućuju prijenos podataka mnogo puta brže nego sto je drugačije moguće. Također, ako smo preuzeli fotografiju s neke web stranice ona je vrlo vjerojatno bila u komprimiranom formatu.

Kombinacija gore navedenih algoritama pojavljuje se kao jedna od metoda sažimanja u ZIP formatu za arhiviranje datoteka, te PNG formatu za zapis slika. Osim kod ZIP-a, sažimanje bez gubitaka primjenjuje se i u drugim formatima za arhiviranje, profesionalnom formatu zapisa glazbe FLAC, u HTTP-u i općenito u situacijama gdje postoji redundancija, a gubitak informacija je nepoželjan.

4.1. Sažimanje fotografija

Cilj kompresije fotografije bez gubitka je prikazati slikovni signal s najmanjim mogućim brojem bitova bez da se izgube informacije, čime se ubrzava prijenos i minimizira zahtjev za pohranom. Kompresija slike uključuje kodiranje pri čemu se podaci smanjuju unutar *framea* slike uklanjanjem informacija koje su ljudskom oku neprimjetne.

Portable Network Graphics(PNG) format je format slike bez gubitka koji se koristi GZIPstyle kompresorom kako bi smanjio količinu podataka. Budući da je to metoda komprimiranja slike bez gubitka, kvaliteta komprimirane slike će biti identična onoj originalnoj. To znači da se može održati visoka kvaliteta slike iako se koristi kompresija.



Slika 15 Usporedba kompresije slike sa i bez gubitka[15]

Graphics Interchange Format(GIF) je još jedan format koji, uz animaciju, podržava transparentnost. On nije metoda kojom se koristimo za kompresiju, već grafička datoteka koja koristi varijantu LZW algoritma da bi se komprimirali podaci. Sastoji se od dva dijela kompresije , prvi korak s gubitkom koji smanjuje paletu boja za cijelu sliku na samo 256 boja, i drugi kompresija s LZW algoritmom. Smanjivanje boja slike na samo 256 ishodi agresivnim smanjenjem kvalitete ali je zato kompresija bolja.

[15] Slika skinuta s https://flat-icons.com/lossy-vs-lossless/

4.2. Sažimanje video zapisa

Svi video podatci zauzimaju mnogo prostora. Sažimanjem se kod video zapisa pokušavaju smanjiti i ukloniti suvišni video podatci kako bi se digitalna video datoteka mogla učinkovito poslati mrežom i pohraniti na računalne diskove. Koristi se prostornim kodiranjem, tj. komprimira se svaki *frame* videa pojedinačno. Korištenjem standardnih tehnika sažimanja osigurava se kompatibilnost i interoperabilnost. Standardi kompresije video zapisa su Motion JPG, MPEG-4 i H.264.

4.3. Sažimanje zvučnih zapisa

Analogni audio signali se pretvaraju u digitalne koristeći se postupkom uzorkovanja, nakon čega se komprimiraju kako bi smanjili veličinu za bolji prijenos i pohranu. Konverzija i kompresija se vrši pomoću kodeka[16].

Najpopularniji kodeci bez gubitaka podataka su FLAC (Free Lossless Audio Codec) i ALAC (Apple Lossless Audio Codec).

[16] Algoritam koji kodira i dekodira audio podatke

Zaključak

Ovaj se rad fokusirao na sažimanje podataka bez gubitka u različitim segmentima te različitim algoritmima kojima se koristimo. Pri sažimanju podataka, analizirali smo dvije tehnike, one koje se temelje na statistici i one koje se temelje na rječniku.

Koristeći se jednostavnijim primjerima tumači se princip rada Shannon-Fanovog, tradicionalnog i proširenog Huffmanovog algoritma, algoritma LZ77 i LZ78, te LZW algoritma. Na kraju svakog objašnjena, priloženi je i pseudokod za svaki od algoritama.

U slučaju statističkih metoda, Huffmanovo kodiranje je puno učinkovitije i optimalnije od Shannon-Fanovog kodiranja, dok kod metoda koje se baziraju na rječniku LZW nadmašuje LZ77 i LZ78.

U nekim je slučajevima potrebno da nakon dekomprimiranja podataka dobijemo isti podatak kao što je bio u originalu. Tipični primjeri su izvršni programi, tekstualni dokumenti te izvorni kodovi, ali se često koristi i kod sažimanja fotografija, video zapis i glazbe.

Literatura

[1] Loudon K., *Mastering Algorithms with C*, O'Reilly Media, Inc., 1999.

[2] McAnlis C., Haecky A., *Understanding Compression*, O'Reilly Media, Inc., 2016.

[3] Salomon D., *Data Compression*, Springer Science+Business Media, LLC, 2007.

[4] Salomon D., Motta G., *Handbook of Data Compression*, British Library Cataloguing, 2010.

[5] Sayood K., *Introduction to Data Compression*, University of Nebraska, 2006.

[6] Visualizing Adaptive Huffman Coding http://ben-tanen.com/adaptive-huffman/ (pokušaj pristupa 21.09.2020.)

[7] *LZW Dana Compression* https://www2.cs.duke.edu/csed/curious/compression/lzw.html (pokušaj pristupa 21.09.2020.)

[8] *Shannon-Fano Algorithm for Data Compression* https://www.geeksforgeeks.org/shannon-fano-algorithm-for-data-compression/ (pokušaj pristupa 15.09.2020.)

[9] *LZW (Lempel-Ziv-Welch) Compression technique* https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/?ref=lbp (pokušaj pristupa 16.09.2020.)

[10] *Lossless Compression vs Lossy Compression* https://gisgeography.com/lossless-compression-vs-lossy-compression/ (pokušaj pristupa 15.09.2020.)

[11] *Static Defined-Word Schemes* https://www.ics.uci.edu/~dan/pubs/DC-Sec3.html (pokušaj pristupa 15.09.2020.)

[12] *Adaptive Huffman Coding* https://www.ics.uci.edu/~dan/pubs/DC-Sec4.html (pokušaj pristupa 16.09.2020.)

[13] *Bricolage: Data Compression* https://perl.plover.com/Huffman/huffman.html (pokušaj pristupa 20.09.2020.)

[14] *Huffman Coding* http://rosettacode.org/wiki/Huffman_coding (pokušaj pristupa 15.09.2020.)

[15] *Lossless Compression: An Overview* https://cs.stanford.edu/people/eroberts/courses/-soco/projects/data-compression/lossless/-index.html (pokušaj pristupa 15.09.2020.)

[16] *Shanon-Fano Coding* http://www.stringology.org/DataCompression/sf/index_en.html (pokušaj pristupa 22.09.2020.)

[17] *Static Huffman Coding* http://www.stringology.org/DataCompression/sh/index_en.html (pokušaj pristupa 22.09.2020.)

[18] *Adaptive Huffman Coding – FGK* http://www.stringology.org/DataCompression/fgk/-index_en.html (pokušaj pristupa 22.09.2020.)

[19] *LZ77* http://www.stringology.org/DataCompression/lz77/index_en.html (pokušaj pristupa 22.09.2020.)

[20] *LZ78* http://www.stringology.org/DataCompression/lz78/index_en.html (pokušaj pristupa 22.09.2020.)

[21] *LZW – compression* http://www.stringology.org/DataCompression/lzw-e/index_en.html (pokušaj pristupa 22.09.2020.)

[22] *LZW - decompression* http://www.stringology.org/DataCompression/lzw-d/index_en.html (pokušaj pristupa 22.09.2020.)

[23] *How data compression works: exploring LZ77* (2019.) https://towardsdatascience.com/-how-data-compression-works-exploring-lz77-3a2c2e06c097 (pokušaj pristupa 20.09.2020.)

Popis slika

Slika 1 Sažimanje podataka sa i bez gubitka 6
Slika 2 Morseova abeceda 7
Slika 3 Primjer sažimanja bez gubitka 8
Slika 4 Stablo Shannon-Fanovog algoritma 9
Slika 5 Pseudokod Shanon-Fanovog algoritma 10
Slika 6 Konstrukcija Huffmanovog stabla 12
Slika 7 Pseudokod Huffmanovog algoritma 13
Slika 8 Primjer proširenog Huffmanovog koda 14
Slika 9 Pseudokod Faller-Gallager-Knuth algoritma 15
Slika 10 Primjer LZ77 kompresije niza ACTGACTACACACTAGCTACG$ 17
Slika 11 Pseudokod algoritma LZ77 17
Slika 12 LZ78 kodiranje 19
Slika 13 Pseudokod algoritma LZ78 19
Slika 14 Pseudokod komprimiranja i dekomprimiranja LZW algoritmom 21
Slika 15 Usporedba kompresije slike sa i bez gubitka 23

Popis tablica

Tablica 1 Primjer rješavanja Shannon-Fanovog kodiranja 10
Tablica 2 Izračunati Huffmanov kod na primjeru slike 12
Tablica 3 Primjer LZW algoritma za ulazni string /WED/WE/WEE/WEB/WET 20