

Univerzitet u Nišu Elektronski fakultet

**Seminarski rad**

Metodi i sistemi za obradu signala

Kompresija podataka primenom LZW protokola

## Studenti: Profesor:

## Teodora Zlatanović Prof. dr Miloš Radmanović Aleksa Milić

Sadržaj

1. [Kompresija podataka 3](#_bookmark0)
   1. [Kompresija bez gubitaka 4](#_bookmark1)
2. [LZ algoritmi 4](#_bookmark2)
   1. [LZ77 5](#_bookmark3)
   2. [LZ78 6](#_bookmark4)
   3. [Moderni kompresori bez gubitaka 7](#_bookmark5)
3. [Kompresori sa gubicima 8](#_bookmark6)
4. [LZW 8](#_bookmark7)
   1. [LZW kodiranje 9](#_bookmark8)
   2. [LZW dekodiranje 10](#_bookmark9)
5. [Primena 11](#_bookmark10)
6. [Varijante 12](#_bookmark11)

7. Izbor programskog jezika za implementaciju [14](#_bookmark12)

8. [Rad sa razlicitim tipovima podataka 15](#_bookmark13)

9. [Izgled i rad LZW aplikacije 16](#_bookmark14)

10. [Ponasanje LZW-a nad tekstualnim podacima 19](#_bookmark15)

11. [Konacni rezultati 20](#_bookmark16)

12. [Moguca unpredjenja 21](#_bookmark17)

13. [Zakljucak 22](#_bookmark18)

14. Komercijalna upotreba [23](#_bookmark19)

15. [Istorija upotrebe 23](#_bookmark20)

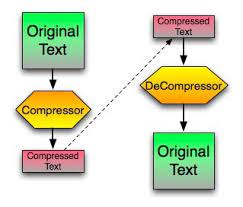
16. [Za vise informacija of LZW-u 25](#_bookmark21)

17. Reference ...........................................................................................................................26

# 1. Kompresija podataka

U obradi signala, kompresija podataka, kodiranje izvora, ili redukcija bitne stope obuhvata kodiranje informacija koristeći manji broj bitova nego u originalnoj reprezentaciji. Kompresija može da bude bilo sa gubicima ili bez gubitaka. Kompresija bez gubitka redukuje bitove putem identifikacije i eliminacije statističke metode. Informacije se ne gube pri tom vidu kopresije. Kompresija sa gubicima redukuje bitove uklanjanjem nepotrebnih ili manje važnih informacija.

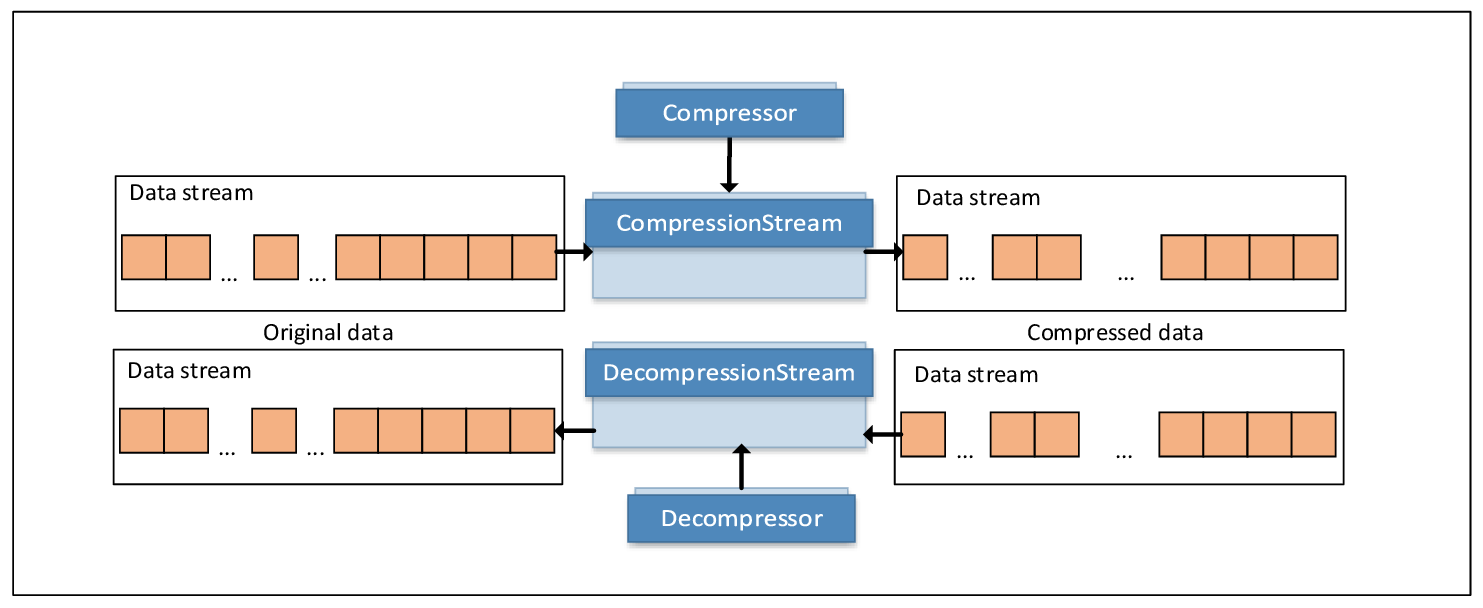
Proces redukovanja veličine datoteke sa podacima se obično naziva kompresijom podataka. U kontekstu prenosa podataka, toj proces se naziva kodiranjem izvora; kodiranje se tipično vrši na izvoru podataka pre njihovog skladištenja ili prenosa. Kodiranje izvora ne treba mešati sa kodiranjem kanala radi detekcije i korekcije grešaka, ili kodiranjem linije sredstva za mapiranje podataka na signal.



Kompresija je korisna jer se njom redukuju resursi neophodni za skladištenje i prenos podataka. Računarski resursi se konzumiraju pri procesu kompresije i isto tako pri suprotnom procesu (dekompresiji). Kompresija podataka je zavisna od kompromisa prostorno–vremenske kompleksnosti. Na primer, kompresiona šema za video može da zahteva skup hardver da bi se video dovolno brzo dekompresovao tako da se može gledati kao da je već bio dekomprimovao. Opcija da se video u potpunosti dekompresuje pre gledanja može da pude nepodesna ili da

zahteva znatan dodatni prostor. Dizajn šema za kompresiju podataka obuhvata kompromise među mnoštvom različitih faktora, uključujući stepen kompresije, količinu unešene distorzije (pri korištenju kompresije sa gubitkom), i računarske resurse neophodne za komprimovanje i dekomprimovanje podataka.

Na slici je simbolički prikazan metod kompresije podataka, gde se vidi razlika između originalnih podataka i podataka nakon kompresije, tj. kompresovanih podataka.



Originalni podaci – kompresovani podaci

# 

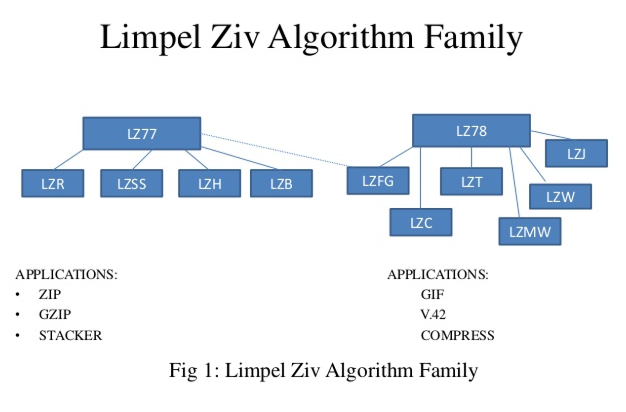
# 1.1. Kompresija bez gubitaka

Algoritmi kompresije bez gubitka obično iskorišćavaju statističku redundanciju za prikazivanje podatka bez gubitaka informacija, tako da je proces reverzibilan. Kompresija bez gubitaka je moguća zato što većina podataka iz realnog sveta pokazuje statističku izlišnost. Na primer, jedna slika može imati područja boje koja se ne menjaju tokom nekoliko piksela; umesto kodiranja „crveni piksel, crveni piksel, ...” podaci mogu da budu kodirani kao „279 crvenih piksela”. Ovo je bazni primer kodiranja dužine trajanja; postoje mnoge šeme za smanjenje veličine datoteke uklanjanjem redundancije.

2. LZ Algoritmi

Svakako jedni od najpopularnijih algoritmima za skladištenje bez gubitaka jesu Lempel–Zivovi (LZ) kompresioni metodi. DEFLATE je varijacija LZ pristupa optimizovana za dekompresiju govora i poboljšanje kompresionog odnosa, ali kompresija može da bude spora. Tokom sredine 1980-ih, nakon rada Terija Velča, Lempel-Ziv-Velčov (LZW) algoritam brzo je postao preferentni metod za većinu kompresionih sistema opšte namene. LZW se koristi u GIF slikama, programima kao što je PKZIP, i hardverskim uređajima kao što su modemi.

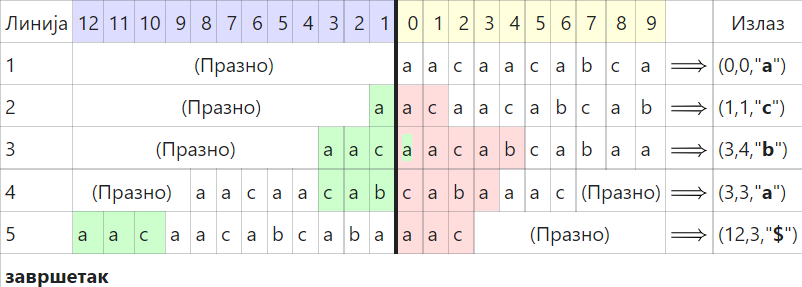
LZ metodi koriste kompresioni model koji je baziran na tabelama, pri čemu se tabelarnim unosima zamenjuju ponavljajući nizovi podataka. Za većinu LZ metoda, ova tabela se dinamički generiše iz ranijih podataka na ulazu. Sama tabela je obično kodirana pomoću Hafmanovih kodova. Gramatički zasnovini kodovi poput ovih mogu da mogu da kompresuju visoko repetitivne podatke sa ekstremnom efikasnošću, na primer, kolekcija biologiških podataka o istoj ili blisko srodnim vrstama, ogromna kolekcija dokumenata sa višestrukim verzijama, internetsko arhiviranje, itd. Osnovni zadatak na gramatici zasnovanih kodova je konstruisanje bezkontekstne gramatike izvedene iz pojedinačnih nizova. Drugi praktični gramatički kompresioni algoritmi su Sekvitur i Re-Pair.



2.1. LZ77

Algoritam LZ77 teži postiči kompresiju zamenom niza podataka koji se ponavljaju referencom na jedan od njih koji će se nalaziti u nekompresovanom delu datoteke. Da bi ceo postupak mogao da se izvede algoriam mora pamtiti neku količinu podataka za vreme izvršavanja postupka. Strukturu u kojoj se ovi podaci održavaju nazivamo klizni-prozor (Sliding window protocol) te kroz nju možemo provući poslednja 2, 4 ili 32 kilobajta podataka. Koder mora zadržati ove podatke sve dok ne postavi referencu na svaki duplikat.

U nastavku ćemo prikazati princip rada ovog algoritma korišćenjem rečnika bafera veličine 12, te veličine 9. Kao što se može videti na priloženoj slici, položaj uz desnu iviču rečnika se prilikom komprimacije mora uzeti u obzir a kao proizvod rada algoritma imamo kolonu izlaz (poslednju kolonu naše tabele).



Baferi rade po principu kliznih-prozora. Podatak koji uđe u bafer moraće proći dužinu rečnika i postaviti reference na sve duplikate. Ovakav način rada neće biti vremenski najefikasniji, ali će rezultat biti efikasan jer u bafer ni jednog trenutka neće dospeti neki duplikat.

Osim toga, kada je LZ77 u pitanju, postoje i izvesna ograničenja:

# Ako je rastojanje između dva ponovljena šablona veće od veličine bafera za pretragu, LZ77 algoritmi ne mogu efikasno da rade,

# Fiksna veličina oba bafera implicira da podudarni niz ne može biti duži od zbir veličina dva bafera, što znači još jedno ograničenje efikasnosti kodiranja

# Povećanje veličine bafera za pretragu i bafera za gledanje unapred će naizgled rešiti probleme

# 2.2. LZ78

# Budući da je ovaj algoritam usledio godinu dana nakon algoritma LZ77, poznati su još i kao LZ1 i LZ2. Algoritmi LZ78 kompresuju sekvencijalne podatke tako što iz ulaza prave rečnik sekvenci tokena, a zatim zamenjuju drugo i sledeće pojavljivanje sekvence u toku podataka referencom na unos iz rečnika. Zapažanje je da je broj ponovljenih sekvenci dobra mera neslučajne prirode sekvence. Algoritmi predstavljaju rečnik kao n-arno stablo gde je n broj tokena koji se koriste za formiranje sekvenci tokena. Svaki unos iz rečnika je oblika

# rečnik[...] = {indeks, token},

# gde je indeks indeks unosa rečnika koji predstavlja prethodno viđenu sekvencu, a token je sledeći token iz unosa koji ovaj unos čini jedinstvenim u rečnik. Treba obratiti pažnju na to da algoritam ništa ne dodaje u tabelu dok se ne pronađe jedinstveni token za pravljenje. Inicijalizuje se poslednji odgovarajući indeks = 0 i sledeći dostupni indeks = 1, a zatim, za svaki token ulaznog toka, rečnik je tražio podudaranje:

# {poslednji indeks podudaranja, token}.

# Ako je pronađeno podudaranje, onda se indeks poslednjeg podudaranja postavlja na indeks odgovarajućeg unosa, ništa se ne izlazi, a poslednji odgovarajući indeks je ostavljen i predstavlja dosadašnji ulaz. Unos se obrađuje dok se ne pronađe podudaranje. Zatim se kreira novi unos u rečniku:

# rečnik[sledeći dostupni indeks] = {poslednji odgovarajući indeks, token},

# a algoritam daje poslednji odgovarajući indeks, praćen token, zatim resetuje poslednji odgovarajući indeks = 0 i povećava sledeći dostupni indeks.

# U nastavku možemo videti tabelu koja pokazuje prvih 14 koraka u kodiranju stringa:

# C:\Users\Teodora\Desktop\Fakultet\Signali i sistemi\Untitledx.png

# **Ograničenja sa LZ78**:

# U ovom domenu treba pomenuti i ograničenja LZ78:

# nema upotrebe kliznog prozora,

# svaki put kada se izda pokazivač, kodirani niz je uključen u rečnik

# kada se dostigne unapred podešeno ograničenje veličine rečnika, rečnik je fiksiran za budućnost (ako je efikasnost kodiranja dobra), ili se resetuje na nulu, tj. mora se ponovo pokrenuti

# umesto trojki koje se koriste u LZ77, u LZ78 se koriste samo parovi. Konkretno, samo položaj.

# 2.3. Moderni kompresori bez gubitaka

# Najsnažniji moderni kompresori bez gubitaka koriste [probabilističke](https://sr.m.wikipedia.org/wiki/Randomized_algorithm) modele, kao što je [predviđanje parcijalnog podudaranja](https://sr.m.wikipedia.org/w/index.php?title=Prediction_by_partial_matching&action=edit&redlink=1). [Barous-Vilerova transformacija](https://sr.m.wikipedia.org/w/index.php?title=Burrows%E2%80%93Wheeler_transform&action=edit&redlink=1) se isto tako može smatrati indirektnom formom statističkog modelovanja. U daljem poboljšanju direktne upotrebe [probabilističkog modelovanja](https://sr.m.wikipedia.org/w/index.php?title=Probabilistic_model&action=edit&redlink=1), statističke procene se mogu povezati sa algoritmom zvanim [aritmetičko kodiranje](https://sr.m.wikipedia.org/w/index.php?title=Arithmetic_coding&action=edit&redlink=1).

# Aritmetičko kodiranje je moderna tehnika kodiranja koja koristi matematičke proračune [mašine konačnog stanja](https://sr.m.wikipedia.org/wiki/Finite-state_machine) za proizvođenje niza kodiranih bitova iz serije simbola inputnih podataka. Ovim pristupom se može ostvariti superiorna kompresija u odnosu na druge tehnike, kao što je dobro poznati Hafmanov algoritam.

# Pri aritmetičkom kodiranju se koriste unutrašnja memorijska stanja da bi se izbegla potreba za mapiranjem jedan-na-jedan individualnih ulaznih simbola na distinktne reprezentacije koje koriste celobrojni broj bitova. Do čišćenja unutrašnje memorije dolazi samo nakon što se iskodira celokupni niz simbola podataka. Aritmetičko kodiranje je veoma podesno za zadatke prilagodljive kompresije podataka gde statistika varira i zavisi od konteksta, jer se ono lako može povezati sa adaptivnim modelom [distribucije verovatnoće](https://sr.m.wikipedia.org/wiki/Probability_distribution) ulaznih podataka.

# Rani primer upotrebe aritmetičkog kodiranja je bio u opcionom svojstvu [JPEG](https://sr.m.wikipedia.org/wiki/JPEG) standarda.

# 3. Kompresija sa gubicima

Budući da je tema ovog rada kompresija bez gubitaka, ovom prilikom ćemo napraviti samo kratak osvrt na kompresiju sa gubicima, sve sa ciljem sticanja kompletnije slike.

U kasnim 1980-im, digitalne slike su postale uobičajene, i standardi za kompresiju slika bez gubitaka su se pojavili. U ranim 1990-im, metode kompresije sa gubicima su ušle u široku upotrebu. U tim šemama, izvestan gubitak informacije je prihvatljiv, jer odbacivanje nebitnih detalja može da uštedi skladišni prostor. Postoji korespondirajući kompromis između očuvanja informacija i smanjenja veličine podataka. Šeme kompresije podataka sa gubitkom su dizajnirane istraživanjem načina na koji ljudi spoznaju date podatke. Na primer, ljudsko oko je senzitivnije za suptilne varijacije u osvetljenju, nego za varijacije boje. Kompresija JPEG slika se delimično ostvaruje zaokruživanjem nebitnih bitova informacija. Brojni popularni kompresioni formati iskorištavaju ove perceptivne razlike, uključujući psihoakustiku za zvuk, i psihovizualne efekte za slike i video.

Kompresija sa gubicima se može koristiti u digitalnim kamerama, za povećanje kapaciteta skladištenja uz minimalnu degradaciju kvaliteta slike. Slično tome, DVD

tehnologija koristi MPEG-2 video kodirajući format sa gubicima za video kompresiju.

U audio kompresiji sa gubicima, metodi psihoakustike se koriste za uklanjanje nečujnih (ili manje čujnih) komponenti audio signala. Kompresija ljudskog govora se često izvodi još više specijalizovanim tehnikama; kodiranje govora, ili kodiranje glasa, ponekad se izdvaja kao zasebna disciplina od audio kompresije. Različiti standardi audio i govorne kompresije su prisutni u formatima audio kodiranja. Kompresija glasa se koristi u internetskoj telefoniji, dok se audio kompresija koristi za CD skladištenje, i audio plejeri dekodiraju te zapise.

# 4. LZW

# Lempel–Ziv–Velch (LZW) je univerzalni algoritam za kompresiju podataka bez gubitaka koji su kreirali Abraham Lempel, Jacob Ziv i Terri Velch. Objavio ga je Velč 1984. kao poboljšanu implementaciju algoritma LZ78 koji su objavili Lempel i Ziv 1978. Algoritam je jednostavan za implementaciju i ima potencijal za veoma visoku propusnost u hardverskim implementacijama. To je algoritam široko korišćenog uslužnog programa za kompresiju Unik datoteka i koristi se u formatu GIF slike.

# Diagram implementacije LZW algoritma:

# 

# 4.1. LZW Kodiranje

# Algoritam kodiranja je sledeći:

# inicijalizacija rečnika tako da sadrži sve nizove dužine jedan,

# pronalaženje najdužeg niza V u rečniku koji odgovara trenutnom unosu,

# emitujtovanje indeksa rečnika za V za izlaz i uklonjanje V sa ulaza,

# dodavanje V kao i sledećeg simbola u unosu u rečnik,

# povratak na korak 2.

# C:\Users\Teodora\Desktop\Fakultet\Signali i sistemi\download.png

# Rečnik je inicijalizovan tako da sadrži nizove od jednog znaka koji odgovaraju svim mogućim ulaznim znakovima i ništa drugo osim kodova za brisanje i zaustavljanje ako se koriste. Algoritam funkcioniše tako što skenira ulazni niz u potrazi za uzastopno dužim podstringovima dok ne pronađe onaj koji nije u rečniku. Kada se takav niz pronađe, indeks za string bez poslednjeg karaktera – odnosno najduži podniz koji se nalazi u rečniku – se preuzima iz rečnika i šalje na izlaz, a novi string, uključujući poslednji karakter, se dodaje u rečnik sa sledećim dostupnim kodom. Poslednji ulazni karakter se zatim koristi kao sledeća početna tačka za traženje podstringova.

# Na ovaj način se sukcesivno duži nizovi registruju u rečniku i dostupni za naknadno kodiranje kao pojedinačne izlazne vrednosti. Algoritam najbolje funkcioniše na podacima sa ponovljenim obrascima, tako da početni delovi poruke vide malo kompresije. Međutim, kako poruka raste, odnos kompresije asimptotski teži maksimumu.

# Primer nase implementacije:

# 

# 4.2. LZW dekodiranje

# Ovde je prikazan prikaz visokog nivoa algoritma za dekodiranje:

# inicijalizacija rečnika tako da sadrži sve nizove dužine jedan,

# čitanje sledećeg kodiranog simbola i račvanje u zavisnosti da li je kodiran u rečniku:

# Jeste:

# emitovanje odgovarajućeg niza V za izlaz,

# spajanje prethodnog stringa koji se emituje za izlaz sa prvim simbolom iz niza V i dodavanje ovoga u rečnik.

# Nije:

# spajanje prethodnog stringa koji se emituje na izlaz sa njegovim prvim simbolom i dodeljivanje naziva V,

# dodavanje V u rečnik i emitovanje na izlaz.

# ponavljanje drugog koraka do kraja ulaznog niza.

# C:\Users\Teodora\Desktop\Fakultet\Signali i sistemi\images.png

# Algoritam za dekodiranje funkcioniše tako što čita vrednost sa kodiranog ulaza i izbacuje odgovarajući niz iz rečnika. Međutim, nije potreban ceo rečnik, već samo početni rečnik koji sadrži nizove od jednog karaktera – i koji je obično tvrdo kodiran u programu, umesto da se šalje sa kodiranim podacima. Umesto toga, ceo rečnik se ponovo gradi tokom procesa dekodiranja na sledeći način: nakon dekodiranja vrednosti i izlaza niza, dekoder ga spaja sa prvim karakterom sledećeg dekodiranog niza i ažurira rečnik sa novom žicom. Dekoder zatim prelazi na sledeći ulaz, koji je već pročitan u prethodnoj iteraciji, i obrađuje ga kao i ranije, i tako sve dok ne iscrpi ulazni tok.

# Primer nase implementacije:

# 5. Primena

# Mnoge aplikacije primenjuju dalje kodiranje na sekvencu izlaznih simbola. Neki pakuju kodirani tok kao znakove za štampanje koristeći neki oblik kodiranja binarnog u tekst. Ovo povećava kodiranu dužinu i smanjuje stopu kompresije.

# Suprotno tome, povećana kompresija se često može postići pomoću adaptivnog entropijskog enkodera. Takav koder procenjuje distribuciju verovatnoće za vrednost sledećeg simbola, na osnovu do sada posmatranih frekvencija vrednosti. Standardno entropijsko kodiranje kao što je Hafmanovo ili aritmetičko kodiranje zatim koristi kraće kodove za vrednosti sa većom verovatnoćom.

# LZW kompresija je postala prva široko korišćena univerzalna metoda kompresije podataka na računarima. Velika tekstualna datoteka na engleskom obično se može komprimovati preko LZW-a na otprilike polovinu svoje originalne veličine.

# LZW je korišćen u kompresovanju programa u javnom domenu, koji je postao manje-više standardni uslužni program u Unik sistemima oko 1986. Od tada je nestao iz mnogih distribucija, kako zbog toga što je prekršio LZW patent, tako i zato što je gzip proizvodio bolje omjere kompresije koristeći LZ77 baziran na DEFLATE algoritmu, ali od 2008. barem FreeBSD uključuje kompresiju i raspakivanje kao deo distribucije. Nekoliko drugih popularnih uslužnih programa za kompresiju takođe je koristilo LZW ili blisko povezane metode.

# LZW je postao veoma široko korišćen kada je postao deo GIF formata slike 1987. Takođe se (opciono) može koristiti u TIFF i PDF datotekama. (Iako je LZW dostupan u softveru Adobe Acrobat, Acrobat podrazumevano koristi DEFLATE za većinu tekstualnih i slikovnih podataka zasnovanih na tabelama boja u PDF datotekama.)

# C:\Users\Teodora\Desktop\Fakultet\Signali i sistemi\download (1).png

# 

# 6. Varijante

# LZW je adaptivna metoda kompresije podataka, ali se sporo prilagođava svom unosu, pošto stringovi u rečniku dobijaju samo po jedan znak duži. LZMW metoda, je varijanta LZW koja prevazilazi ovaj problem. Njegov glavni princip je sledeći: umesto dodavanja I plus jedan znak sledeće fraze u rečnik, dodajte I plus celu sledeću frazu u rečnik.

# LZAP metoda je još jedna varijanta zasnovana na ideji da umesto toga da samo spaja poslednje dve fraze i stavlja rezultat u rečnik, smešta sve prefikse konkatenacijom u rečnik. Tačnije, ako su S i T poslednja dva podudaranja, dodajte St u rečnik za svaki neprazan prefiks t od T, uključujući T sama.

# Tabela ispod sumira principe LZW, LZMW i LZAP i pokazuje kako oni prirodno predlažu drugu varijantu, LZY.

# C:\Users\Teodora\Desktop\Fakultet\Signali i sistemi\Untitledx.png

# LZW dodaje jedan niz rečnika po frazi i povećava nizove za jedan simbol po vremenu. LZMW dodaje jedan niz rečnika po frazi i povećava nizove za nekoliko simbola u ​​isto vreme. LZAP dodaje jedan niz rečnika po ulaznom simbolu i povećava nizove po nekoliko simbola istovremeno. LZY odgovara četvrtoj ćeliji tabele. To je metod koji dodaje jedan niz rečnika po ulaznom simbolu i povećava nizove po jednom simbolu.

# 7. Izbor programskog jezika za implementaciju

# Na performanse algoritma može uticati programski jezik u kojem je implementiran. Neki programski jezici su generalno brži od drugih, zbog razlika u njihovom dizajnu, implementaciji i osnovnim sistemima za izvršavanje.

# Na primer, jezici koji su kompajlirani u izvorni kod, kao što su C ili C++, često mogu da rade brže od jezika koji se tumače, kao što su Python ili Javascript. To je zato što se kompajlirani kod može optimizovati za određeni procesor i operativni sistem na kojem radi, dok se interpretirani kod mora izvoditi preko interpretatora, što može povećati troškove i smanjiti performanse.

# Međutim, performanse algoritma nisu samo određene programskim jezikom. Složenost algoritma i efikasnost implementacije takođe igraju značajnu ulogu u njegovom izvođenju. Zbog toga je važno uzeti u obzir sve ove faktore kada birate programski jezik za određeni zadatak.

# Takođe je vredno napomenuti da neki programski jezici imaju biblioteke ili okvire koji se mogu koristiti za optimizaciju performansi određenih tipova algoritama. Na primer, Pythonima biblioteke kao što su NumPy i SciPy koje su optimizovane za numeričko računanje, a C++ ima biblioteke kao što su OpenCV i Boost koje su optimizovane za obradu slika, tekstova i raznih tipova podataka. Korišćenje ovih specijalizovanih biblioteka često može da obezbedi značajno povećanje performansi u poređenju sa primenom istih algoritama od nule.

# Za potrebe ovog kursa, koriscen je programski jezik Python bez koriscenja biblioteka za optimizaciju rada algoritma.

# 8. Rad sa razlicitim tipovima podataka

# 

# Jedan od izazova korišćenja LZW algoritma na različitim tipovima datoteka je da efikasnost kompresije može da varira u zavisnosti od prirode podataka. Na primer, LZV algoritam može biti efikasniji u komprimovanju tekstualnih datoteka od binarnih datoteka, pošto tekstualne datoteke obično imaju veći udeo ponavljajućih obrazaca. S druge strane, binarne datoteke kao što su izvršni programi ili mašinski kod mogu da sadrže manje obrazaca koji se ponavljaju, što ih čini manje pogodnim za LZW kompresiju.

# Još jedan izazov korišćenja LZW algoritma je to što može biti intenzivan u računarstvu, posebno za velike datoteke. Algoritam funkcioniše tako što gradi rečnik obrazaca koji se ponavljaju dok obrađuje podatke, što može zahtevati značajnu količinu memorije i procesorske snage. Ovo može biti problem za uređaje sa ograničenim resursima, kao što su mobilni uređaji ili ugrađeni sistemi.

# 9. Izgled i rad LZW aplikacije

# Za implementaciju grafickog interfejsa aplikacije koriscena je Python biblioteka TKInter.

# Postoji nekoliko prednosti posedovanja grafičkog korisničkog interfejsa (GUI) za aplikaciju:

# Poboljšana upotrebljivost: GUI olakšava korisnicima interakciju sa aplikacijom i pristup njenim funkcijama, jer pruža vizuelni prikaz funkcija aplikacije i omogućava korisnicima da se kreću kroz aplikaciju koristeći poznate elemente kao što su dugmad, ikone i meniji.

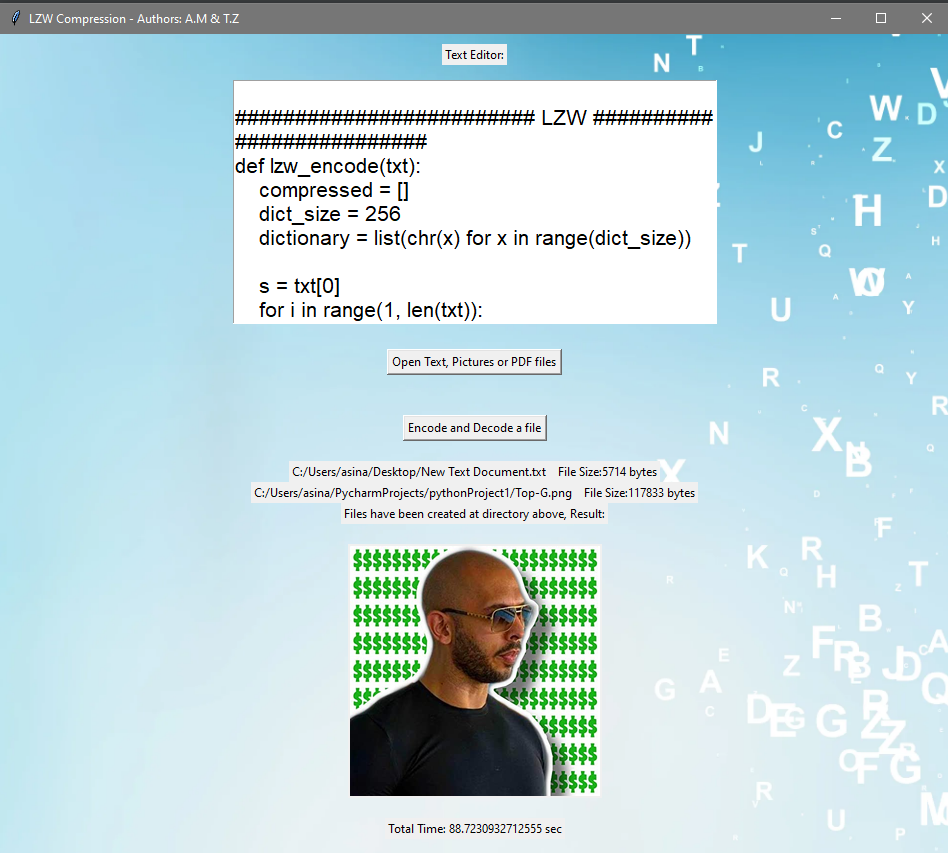
# Poboljšana pristupačnost: GUI može učiniti aplikaciju pristupačnijom korisnicima sa invaliditetom, jer im omogućava interakciju sa aplikacijom koristeći pomoćne tehnologije kao što su čitači ekrana ili softver za pretvaranje govora u tekst.

# Veća privlačnost: Dobro dizajniran GUI može učiniti aplikaciju privlačnijom za korisnike i povećati njenu korisničku bazu.

# Bolja organizacija: GUI može pomoći u organizovanju funkcija i funkcija aplikacije na logičan i intuitivan način, olakšavajući korisnicima da pronađu ono što im treba i efikasno koriste aplikaciju.

# Poboljšana komunikacija: GUI može pomoći u prenošenju informacija korisnicima na jasan i koncizan način, pomažući im da razumeju kako aplikacija funkcioniše i šta može da uradi za njih.

# Poboljšano korisničko iskustvo: Sve u svemu, GUI može poboljšati korisničko iskustvo tako što ga čini intuitivnijim i prijatnijim za korisnike u interakciji sa aplikacijom.



# Na prethodnoj slici vidimo izgled i rad aplikacije. Vidmo kako se aplikacija, odnosno algoritam ponasa pri kompresovanju razlicitih tipova podataka, razlicitih velicina.

# Mozemo primetiti da se mali tekstualni fajlovi od svega nekoliko desetina redova (odnosno oko 6000 bajtova) kompresuju i dekompresuju gotovo u trenutku. Sto kod vecih i drugacijih fajlova to nije slucaj, poslednji visoko kvalitetni png fajl (pogledati prethodnu sliku) velicine ~111 000 bajtova je bilo potrebno cak 88 sekundi. Medjutim, aplikacija je bila zaduzena za kreiranje raznih fajlova u medjukoracima algoritma, ali bez obzira na to, ovaj primer bi trebalo da docara razlicitost rada algoritma sa razlicitim tipovima podataka.

# Primer rada LZW-a sa malim tekstualnim fajlom **Primer.txt**:

# 

# Velicina ovog malog originalnog fajla je 178 bajtova i vidimo kako velicina ovog fajla ne predstavlja nikakav problem za rad algoritma.

# Posledice rada algoritma, odnosno aplikacije su:

# 

# 

# 1. Kreiranje PrimerEncoded fajla (koji je inicijalno bio ekstenzije .lzw, ali je konvertovan u .txt) koji je manje velicine nego originalni, sto se i ocekuje, jer algoritam radi zamenu obrascima koji se ponavljaju u toku podataka kodovima fiksne velicine.

# Sadrzaj PrimerEncoded.lzw fajla (velicine 139 bajta):

# 97 98 256 258 257 257 256 260 259 259 263 266 264 267 269 258 262 268 265 273 270 276 264 272 276 277 281 275 97 279 275 282 269 10 289 10

# 2. Kreiranje PrimerDecoded.txt fajla (vise nije lzw ekstenzije vec .txt) koji je manje vise iste velicine kao i originalni. Ovo je ocekivano ponasanje jer algoritam mora biti sposoban da vrati fajl na originalni sadrzaj sa originalnom velicinom.

# Sadrzaj PrimerEncoded.lzw fajla (velicine 178 bajta):

# abababababaabbabababababbababababababababababababababaabbabababababbababababababababababababababaabbabababababbababababababababababababababaabbabababababbababababababababab

# 10. Ponasanje LZW-a nad tekstualnim podacima

# Uoceni obrazac rada LZW algoritma pri razlicitom sadrzaju tekstualnih fajlova

Obrazac konvertovanja LZW-a nad recima razlicitih duzina

|  |  |  |  |
| --- | --- | --- | --- |
| Velicina reci | Procenat Kompresije | Kompresovana velicina u bitovima | Originalna velicina u bitovima |
| 4 | 53.125 | 17 | 28 |
| 10 | 33.75 | 27 | 70 |
| 38 | 30.11 | 95 | 265 |
| 50 | 29.75 | 119 | 350 |

# Sledeći primer daje jasnije razumevanje LZV algoritma. Pretpostavimo da imamo rečnik koji je predstavljen sa 8 bitova u ASCII-u i dužina kompresijskog koda je 9 bita. Tabela 1 prikazuje LZV algoritam za ulazni niz „aabaababbaabbabaaabb“.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Kompresija** | | | **Dekompresija** | | |
| Input | Updateddictionary | Output | Input | Output | Updateddictionary |
| aa | aa=256 | 97 | 97 | a |  |
| ab | ab=257 | 97 | 97 | a | aa=256 |
| ba | ba=258 | 98 | 98 | b | ab=257 |
| aab | aab=259 | 256 | 256 | aa | ba=258 |
| bab | bab=260 | 258 | 258 | ba | aab=259 |
| bb | bb=261 | 98 | 98 | b | bab=260 |
| baa | baa=262 | 258 | 258 | ba | bb=261 |
| abb | abb=263 | 257 | 257 | ab | baa=262 |
| baba | baba=264 | 260 | 260 | bab | abb=263 |
| aaa | aaa=265 | 256 | 256 | aa | baba=264 |
| abb |  | 263 | 263 | abb | aaa=265 |

# 11. Konacni rezultati

# Koriscenjem raznih primera koji su razlicitih tipova podataka (text, png, jpg, pdf) demonstriramo rad i rezultate LZW algoritma i njegovo ponasanje pri nasoj implementaciji.

**Tabela:** Uporedna tabela kompresije podataka.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ime Primera** | **Originalvelicina** | **LZW**  **Comp.**  **(.txt)** | **Compression ratio** | **Tip podataka** | **LZW**  **Decomp.** | **Vreme**  **[sec]** |
| Primer1 | 178 B | 149 B | 1.19 | text | 178 B | 0.0 |
| Primer2 | 43.1 KB | 9.75 KB | 4.42 | text | 43.1 KB | 4.46 |
| Primer3 | 12.7 KB | 3.22 KB | 3.94 | text | 12.7 KB | 1.09 |
| Primer4 | 81.4 KB | 19.4 KB | 4.19 | text | 81.4 KB | 9.19 |
| Top-G | 117.8 KB | 264 KB | 0.44 | PNG | 117.8 KB | 88.72 |
| Top-G | 117.8 KB | 264 KB | 0.44 | JPG | 117.8 KB | 88.72 |
| F-27 | 19.3 KB | 41.1 KB | 0.46 | PNG | 19.3 KB | 14.49 |
| F-27 | 19.3 KB | 41.1 KB | 0.46 | JPG | 19.3 KB | 14.49 |
| Chatcase1 | 14.7 MB | 23.3 MB | 0.63 | PDF | 14.7 MB | 109.9 |
| **Average** | | | **1.7966** |  |  | **36.7844** |

# Racunar nad kojim su izvodjeni test primeri:

# *Specifikacije: CPU: amd athlon x4 740, RAM: 8GB DDR3*

# Mozemo uvideti da se ova verzija najbolje pokazala pri kompresovanju i dekompresovanju tekstualnih fajlova, bilo u stepenu kompresije, bilo to u utrosku vremena. Dok smo za ostale fajlove dobili ne tako zadovljavajuce rezultate. Uzroci ovih rezultata mogu biti razni, mozda do procesorske moci racunara, a mozda to specificnosti implementacije algoritma ili okruzenja u kojem ovaj algoritam radi i u kojem je implementiran. Svakako, postoji veci broj mogucnosti za unapredjenje rada ovog algoritma.

# 12. Moguca unapredjenja

# Postoji nekoliko načina da se poboljša brzina izvršavanja algoritama:

# Optimizacija samog algoritma: Često se najznačajnija poboljšanja mogu postići optimizacijom dizajna samog algoritma. Ovo može uključivati korišćenje efikasnijih struktura podataka, smanjenje broja izvršenih proračuna ili operacija ili pronalaženje efikasnijih načina za rešavanje problema.

# Rad drugih ljudi koji su implenetirali brzu verziju LZW algoritma:

# 

# Vidmo znacajno vecu efikasnost

# Korišćenje bržeg hardvera: Nadogradnja na brži hardver, kao što je brži procesor ili više memorije, često može značajno poboljšati brzinu algoritama.

# Paralelizacija algoritma: Ako se algoritam može paralelizovati, može se podeliti na manje delove koji se mogu istovremeno pokretati na više procesora ili uređaja. Ovo može znatno poboljšati brzinu algoritma.

# Rad drugih ljudi koji su implenetirali paralelnu verziju LZW algoritma:

# 

# Korišćenje kompajliranih jezika: Tumačenje jezika kao što je Pithon može biti sporije od kompajliranih jezika kao što su C ili C++. Ako je brzina glavna briga, možda ćete želeti da razmotrite implementaciju algoritma u kompajliranom jeziku.

# Korišćenje biblioteka i okvira: Postoji mnogo dostupnih biblioteka i okvira koji mogu da obezbede optimizovane implementacije uobičajenih algoritama i struktura podataka. Korišćenje njih često može poboljšati brzinu našeg koda.

# Na kraju krajeva, najbolji pristup za poboljšanje brzine algoritma zavisiće od specifičnog algoritma i raspoloživih resursa. Možda će biti od pomoći eksperimentisati sa različitim pristupima i meriti performanse svakog da bi se odredilo najefikasnije rešenje.

# U ovom slucaju, zadovljicemo se ovom jednostavnijom impementacijom

# 13. Zakljucak

# Jedna od ključnih prednosti LZW-a je da je relativno jednostavan za implementaciju (za samo dvadesetak linija koda mozemo videti algoritam u akciji) i da može brzo kompresovati podatke. Takođe se široko koristi i podržava, sa implementacijama dostupnim za različite programske jezike.

# Međutim, LZW ima nekoliko potencijalnih nedostataka. Jedan je taj što je potrebna relativno velika količina memorije za skladištenje rečnika, što može biti problem za velike skupove podataka. Drugi je da ne postiže uvek najveći mogući odnos kompresije, posebno za skupove podataka koji ne sadrže mnogo ponovljenih obrazaca.

# Sve u svemu, LZW je široko korišćen i efikasan algoritam kompresije podataka, ali možda nije najbolji izbor za svaku situaciju. Važno je uzeti u obzir specifične potrebe i karakteristike podataka koji se komprimuju kada odlučujete koji algoritam ćete koristiti.

# 14. Komercijalna upotreba

# Konačno, LZW algoritam je patentiran i to Unisys LZW lincencnim dogovorom, što može biti izazov za programere koji žele da ga koriste u svom softveru. Iako je patent istekao u nekim zemljama, još uvek je na snazi u drugim, što znači da će programeri možda morati da plate naknade za licenciranje ili da koriste alternativne tehnike kompresije u određenim jurisdikcijama.

# 15. Istorija upotrebe

Da bi smo bolje razumeli sta *Unisys* *LZW* licensing agreement moze znaciti za nas, hajde da zavirimo u proslost njegovog nastanka.

* U 1977, Abraham Lempel i Jakob Ziv su objavili papir o univerzalnom algoritmu za kompresiju podataka. Ovo se zvalo LZ77 algoritmom kompresije.
* U 1978, Lempel i Ziv su objavili unapredjen, recnikom-baziran semu za kompreziju zvanu LZ78.
* U 1981, dok su radili za *Sperry Corporation*, Lempel i Ziv, i radom uz Cohen i Eastman-a, popunili patenat tvrdeci da LZ78 algoritam za kompresiju njihov. I dobili su zvanicni patenat (broj 4,464,650) u 1984.
* Takodje u 1984, dok su radili za Sperry Corporation, Terry Welch je modifikovao LZ78 algoritam da poveca efikasnost za implementaciju u visoko-performansnim disk controlerima. Rezulat je bio LZW algorithm, koji je on opisao u an *IEEE Computer* novinarskom clanku (pogledaj " Vise informacija o LZW-u") nakon sto je dao otkaz u korporaciji Sperry.
* Godine 1985. *Sperry Corporati*on je dobila patent (broj 4,558,302) za Velchov izum i implementaciju LZW algoritma kompresije podataka. Od tog vremena, ovaj LZW patent je javno dostupan svima da ga vide u Uredu za patente SAD i mnogim javnim bibliotekama, a dostupan je i preko mnogih onlajn usluga. Pored toga, strani patenti sa čak širim obimom od patenta u SAD su na čekanju ili su odobreni u Kanadi, Francuskoj, Italiji, Nemačkoj, Velikoj Britaniji i Japanu.
* 1986. godine, *Sperry Corporation* i *Burroughs* su se spojili u *Unysis*. U to vreme, vlasništvo and patentom Sperry preneto je na Unysis.
* U 1987. *CompuServe* je kreirao GIF format datoteke za korišćenje u skladištenju i onlajn preuzimanju bitmapiranih grafičkih podataka. GIF specifikacija je zahtevala korišćenje LZW algoritma za komprimovanje podataka uskladištenih u svakoj GIF datoteci. Vrlo je moguće da CompuServe nije proverio patentne datoteke kako bi utvrdio da li GIF format krši bilo koji patent, što je trebalo da bude učinjeno s obzirom na njihovu široku promociju GIF formata. Takva provera bi otkrila patent **Velch LZW**, koji je tada bio u vlasništvu Unisis-a. U to vreme, Unisis takođe očigledno nije znao da je LZW metod kompresije koji koristi veoma popularni GIF format datoteke.
* Godine 1988. *Aldus Corporation* je izdala reviziju 5.0 TIFF formata datoteke. Ova revizija je dodala novu funkciju koja daje TIFF-u mogućnost da skladišti RGB bitmapirane podatke koristeći sirovi format ili komprimovani format koji koristi LZW algoritam. (Iako se LZW algoritam koji koristi TIFF smatra „pokvarenim“, i dalje je pokriven Unisis patentom.) Od 1991. godine, u skladu sa Aldusovim sporazumom sa Unisis-om, Aldus je bio obavezan da postavi obaveštenje u vezi sa Unisis patentom , i njegovu primenljivost na TIFF, u *Aldus* dokumentaciji. Revizija 6.0 TIFF specifikacije iz 1992. uključuje ovo obaveštenje o Unisis patentu u vezi sa LZW-om.
* U 1990, Unisys je lincencirao Adobe za koriscenje Unisys LZW patenta u svrhu PostScript-a.
* U 1991, *Unisys* je lincencirao *Aldus* za koriscenje Unisys LZW patena u TIFF format.
* U 1993, *Unisys* je postao svestan da CompuServe GIF fajl format koristi LZW algoritam. Pregovor su poceli sa CompuServe da kreiraju lincencirani sporazum za komercijalnu upotrebu LZW-a.
* 1994. godine, Unisis i CompuServe su došli do razumevanja da će korišćenje LZW algoritma od strane Compuserve-a biti licencirano za primenu GIF formata datoteke u softveru koji se prvenstveno koristi za pristup CompuServe Information Service.
* 1995. godine, *America Online* i *Prodigy* takođe su sklopili licencne ugovore sa *Unisis*-om za korišćenje LZW-a.

Od 1990. stotine kompanija je sklopilo LZW ugovore o licenciranju sa Unisis-om.

# 16. Za vise informacija o LZW-u

# Mnoge knjige o kompresiji podataka sadrže informacije o LZ i LZV algoritmima kompresije. Prva referenca u nastavku je definitivan izvor za veoma opšte objašnjenje samog LZV algoritma i ne fokusira se posebno na podatke bitmap slike:

# Velch, T.A., "Tehnika za kompresiju podataka visokih performansi", IEEE Computer, vol. 17, br. 6, jun 1984.

# TIFF specifikacija (obe revizije 5.0 i 6.0) sadrži objašnjenje varijacije TIFF-a na LZV kompresiji. Pogledajte odeljak „Za dodatne informacije“ u TIFF članku za informacije i pogledajte CD-ROM za samu specifikaciju.

# Sledeći članci i rukopisi su takođe posebno vezani za LZV:

# Bell, Timothi C., "Bolja OPM/L kompresija teksta", IEEE Transactions on Communications, vol. 34, br. 12, decembar 1986, 1176-1182.

# Bernstein, Daniel J., I kodiranje, nacrt 4b, 19. mart 1991, rukopisni deo paketa Iabba I Coding.

# Blackstock, Steve, LZV i GIF Ekplained, rukopis u javnom vlasništvu, 1987.

# Montgomeri, Bob, LZV kompresija koja se koristi za kodiranje/dekodiranje GIF datoteke, rukopis u javnom vlasništvu, 1988.

# Nelson, Mark R., "LZV kompresija podataka", Dr. Dobbs Journal, oktobar 1989, str. 29-36.

# Phillips, Dvaine, "LZV Data Compression", The Computer Applications Journal, Circuit Cellar Ink, vol. 27, jun/jul 1992, str. 36-48.

# Rodriguez, Karen, "Patent formata grafičkih datoteka Unisis traži autorske naknade od programera GIF-a", InfoVorld, vol. 17, 9. januar 1995, str. 3.

# Thomborson, Clark, "V.42bis standard za modeme za kompresiju podataka", IEEE Micro, oktobar 1992, str. 41-53.

# Ziv, J. i A. Lempel, "Univerzalni algoritam za sekvencijalnu kompresiju podataka", IEEE Transactions on Information Theori, vol. 23, br. 3, 1977, str. 337-343.

# Ziv, J. i A. Lempel, "Kompresija pojedinačnih sekvenci putem kodiranja sa promenljivom brzinom", IEEE Transactions on Information Theori, vol. 24, br. 5. septembar 1978.

# Možete dobiti kopiju stvarnog LZW patenta od Ureda za patente SAD. Patent je takođe dostupan na mnogim internet sajtovima, uključujući:

# *ftp://cs.columbia.edu/archives/mirror2/world-info/obi/USPatents/lzw-patent.gz* *ftp://ftp.std.com/obi/USPatents/lzw-patent.Z* *ftp://ftp.uu.net/doc/literary/obi/USPatents/lzw-patent.Z* *ftp://gatekeeper.dec.com/.8/misc/lzw-patent.Z*

# Reference

1. Data Compression, The Complete Reference, 3rd Edition, David Salomon, Springer <https://drive.google.com/file/d/18qt5ZIeRwHOOimXVtKVaxSE61M1h0Di6/view>
2. <https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch>
3. A Technique for High-Performance Data Compression, Welch, Terry (1984)
4. [Lempel-Ziv-Welch (LZW) Compression (narod.ru)](http://netghost.narod.ru/gff/graphics/book/ch09_04.htm)
5. Compression of individual sequences via variable-rate coding, Ziv, Lempel (1978)
6. [LZW compression technique | Download Scientific Diagram (researchgate.net)](https://www.researchgate.net/figure/LZW-compression-technique_fig1_337499220)
7. [EE451FinalProject/EE451\_LZW\_project\_paper.pdf at master · Edwin-Chan/EE451FinalProject (github.com)](https://github.com/Edwin-Chan/EE451FinalProject/blob/master/EE451_LZW_project_paper.pdf)