KOMPRESIJA SLIKA POMOĆU HUFFMAN CODING ALGORITMA

Svjetličić, Vilim

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni preddiplomski studij

KOMPRESIJA SLIKA POMOĆU HUFFMAN CODING ALGORITMA

Završni rad

Vilim Svjetličić

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 18.09.2021.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Vilim Svjetličić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4275, 26.07.2018.
OIB studenta:	11804830621
Mentor:	Izv. prof. dr. sc. Irena Galić
Sumentor:	Marija Habijan
Sumentor iz tvrtke:	
Naslov završnog rada:	Kompresija slika pomocu Huffman Coding algortima
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	18.09.2021.
Datum potvrde ocjene Odbora:	22.09.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 24.09.2021.

Ime i prezime studenta:	Vilim Svjetličić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4275, 26.07.2018.
Turnitin podudaranje [%]:	8

Ovom izjavom izjavljujem da je rad pod nazivom**: Kompresija slika pomocu Huffman Coding algortima**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Irena Galić

i sumentora Marija Habijan

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.

Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

Kompresija je postupak kojem je cilj smanjiti nepotrebne informacije kako bi se ključne informacije mogle pohraniti ili poslati na najefikasniji način. Napretkom digitalne tehnologije, globalna potreba za pohranom informacija raste kao i količina informacija koje se prenose internetom. Kako bi se smanjili troškovi povećanja kapaciteta i nadogradnje internetske mreže te smanjilo vrijeme pohrane i prijenosa informacija, najelegantnije i najjeftinije rješenje je primjena algoritama koji smanjuju originalnu poruku i ostavljaju samo ključne informacije iz kojih se može rekonstruirati izvorni zapis.

Kompresija se može izvršiti na nizovima slova, brojeva i ostalih znakova, ali najčešću primjenu ima kada smanjuje veličinu binarnog niza u računalu. Slika je binarni niz i jedan od primjera u kojem se gotovo uvijek koristi kompresija. Brojni formati slika koriste se različitim algoritmima kako bi smanjili veličinu zapisa, a slike bez kompresije nemaju važnu primjenu u svakodnevnom životu.

Prvi dio rada opisuje teorijske osnove kompresije slika i objašnjava *Huffman Coding* algoritam, a drugi dio objašnjava implementaciju *Huffman Coding* algoritma za kompresiju slika u programskom jeziku *Python*. Na kraju se dobiveni rezultati uspoređuju i analiziraju na nekoliko proizvoljnih slika.

1.1. Zadatak završnog rada

Istražiti i opisati teorijske osnove kompresije slika i kompresije slika *Huffman Coding* algoritmom. Dati kratak pregled područja. Implementirati *Huffman Coding* algoritam za primjenu u kompresiji slike pomoću programskog jezika *Python*. Testirati implementirani algoritam na proizvoljno odabranim testnim slikama.

2. KOMPRESIJA SLIKE

2.1. Pregled literature

Huffman Coding algoritam je učinkovit algoritam sa širokom primjenom i zato postoje mnogi materijali koji ga objašnjavaju u obrazovne svrhe ili znanstveni radovi koji istražuju i definiraju njegove mogućnosti. Uvod u način funkcioniranja algoritma može se naći u izvoru [1] u kojem se Huffmanov algoritam definira kao algoritam s promjenjivom duljinom zapisa, a razvio ga je David Huffman 1951. godine. Navedeni izvor prikazuje rješavanje algoritma u koracima uz priloženi kod u *Python* programskom jeziku. Huffmanov algoritam je detaljno proučavan u znanstvenim radovima poput izvora [2] u kojem je opisana metoda bržeg kodiranja i dekodiranja vrijednosti iz Huffmanovog stabla, a uvod i opis područja kompresije može se naći u radovima koji se bave ostalim vrstama kompresije poput izvora [3] koji proučava kompresiju i dekompresiju DWT (engl. *Discrete Wavelet Transform*) pristupom. Neki od ostalih algoritama koji se koriste za kompresiju slike su RLE (engl. *Run-length encoding*) [4] koji je učinkovit za kompresiju ikona i zapisuje niz jednakih sekvencijalnih polja u jednu vrijednost, aritmetičko kodiranje (engl. *aritmetic coding*) [5] slično je Huffmanovom po promjenjivom broju bitova i temelji se na vjerojatnosti pojavljivanja i *transform coding* [6] koje se najčešće koristi u kompresiji s gubitkom podataka.

Implementacija *Huffman Coding* algoritma za niz slova prikazuje demonstraciju kodiranja informacija na računalu. Implementacija Huffmanovog algoritma nalazi se u izvoru [7] gdje je implementiran algoritam za niz slova u programskom jeziku *Python*. U ovom radu proširena je implementacija Huffmanovog algoritam iz spomenutog izvora. Implementacija Huffmanovog algoritma za kompresiju slike u programskom jeziku C nalazi se u izvoru [8]. Za rad sa slikama u programskom jeziku *Python* koriste se biblioteke *OpenCV* (engl. *Open Source Computer Vision Library*) [9] i PIL (engl. *Python Imaging Library*) [10]. *Python* aplikacija pokreće kompresiju i dekodiranje kako bi testiranje bilo lakše. Aplikacija je implementirana po uzoru na aplikaciju u izvoru [11] u kojem se dohvaća put do datoteke u računalu. Aplikacija koristi biblioteku *wxPython* [12] za stvaranje sučelja.

2.2. Kompresija slike

Kompresija digitalnih slika je područje koje proučava tehnike smanjenja ukupnog broja bitova koji predstavljaju sliku. Postupkom kompresije eliminiraju se različite redundancije koje postoje na slici [3].

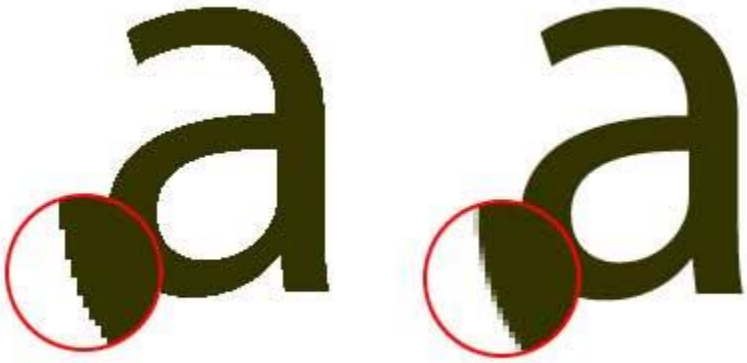
Slika na računalu prikazana je pikselima (engl. *pixels*) koji čine polje određene rezolucije. Ovisno o vrsti slike, pikseli mogu imati različite duljine zapisa. Duljinu zapisa piksela određuje broj kanala koji slika ima. Najčešće su RGB slike koje imaju tri kanala; kanal za crvenu, zelenu i plavu boju. Navedene boje su prikazane u pikselu kao tri cjelobrojne vrijednosti od 0 do 255, a konačna boja piksela je rezultat omjera tih vrijednosti. Jedan RGB piksel ima zapis dug 24 bita. RGBT slika na postojeća tri kanala dodaje kanal za prozirnost piksela i u toj vrsti slike jedan piksel zauzima 32 bita. Slike koje imaju samo sive nijanse (engl. *grayscale*) imaju jedan kanal koji bilježi razinu osvjetljenja. Piksel u slici tog formata zauzima 8 bita, a ako se doda kanal za prozirnost, zauzima 16 bitova. Moguće su i kombinacije različitih kanala te da kanali zauzimaju manje od 8 bitova, ali ključno je da je cijela slika jedan niz cjelobrojnih vrijednosti na kojima se može primijeniti algoritam za kompresiju.

Kompresija slike se dijeli u dvije kategorije, kompresija bez gubitka podataka i kompresija s gubitkom podataka.

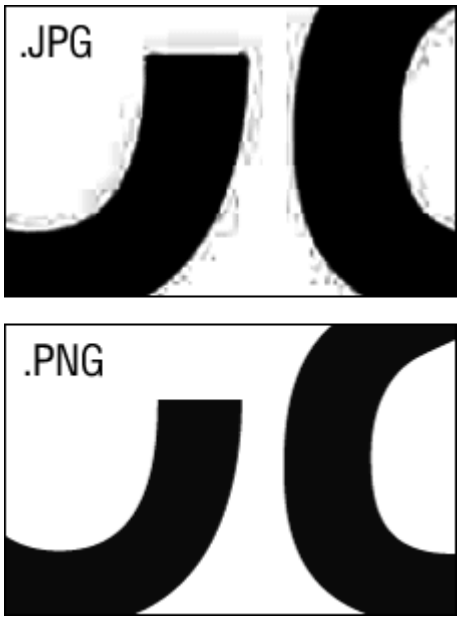
U kompresiji bez gubitka podataka (engl. *lossless*), izvorni se podaci savršeno rekonstruiraju iz komprimiranih podataka. Koriste se svi podaci iz izvorne slike tijekom kompresije, a kad se slika dekomprimira, bit će potpuno identična izvornoj slici. Niti jedan algoritam za kompresiju ne može efikasno raditi na svim vrstama podataka i zato postoji velik broj različitih algoritama koji su dizajnirani za određeni tip ulaznih podataka ili koriste pretpostavke o tome kakvu će redundanciju sadržavati nekomprimirani podaci [13]. Najpoznatija datoteka s ovom vrstom kompresije je PNG (engl. *Portable Network Graphics*). PNG datoteke obično se koriste za pohranu digitalnih slika i slika s prozirnom pozadinom [14].

U kompresiji s gubitkom podataka (engl. *lossy*), dio izvornih podataka se gubi tijekom kompresije i oni su nepovratno izgubljeni u rekonstruiranoj slici. Kompresija se temelji na aproksimaciji i odbacivanju dijela podataka. Najpoznatija datoteka s ovom vrstom kompresije je JPG (engl. *Joint Photographic Experts Group*). Takav pristup omogućuje bolju kompresiju od kompresije bez gubitaka i najčešće se koristi u multimediji [15].

Na slikama 2.1. i 2.2. prikazana je razlika između kompresija s gubitkom i bez gubitka podataka. U kompresiji bez gubitka podataka svaki piksel je jasno definiran što rezultira oštrijom slikom, a u kompresiji s gubitkom podataka slika gubi oštrinu. Zbog toga se preporuča korištenje PNG formata za dokumente kako bi tekst bio jasniji, a nedostatak JPG formata je manje uočljiv na običnim slikama bez strogo definiranih linija, ali zato je veličina slike bitno smanjena.



Slika 2.1. Razlika PNG i JPG formata [16]



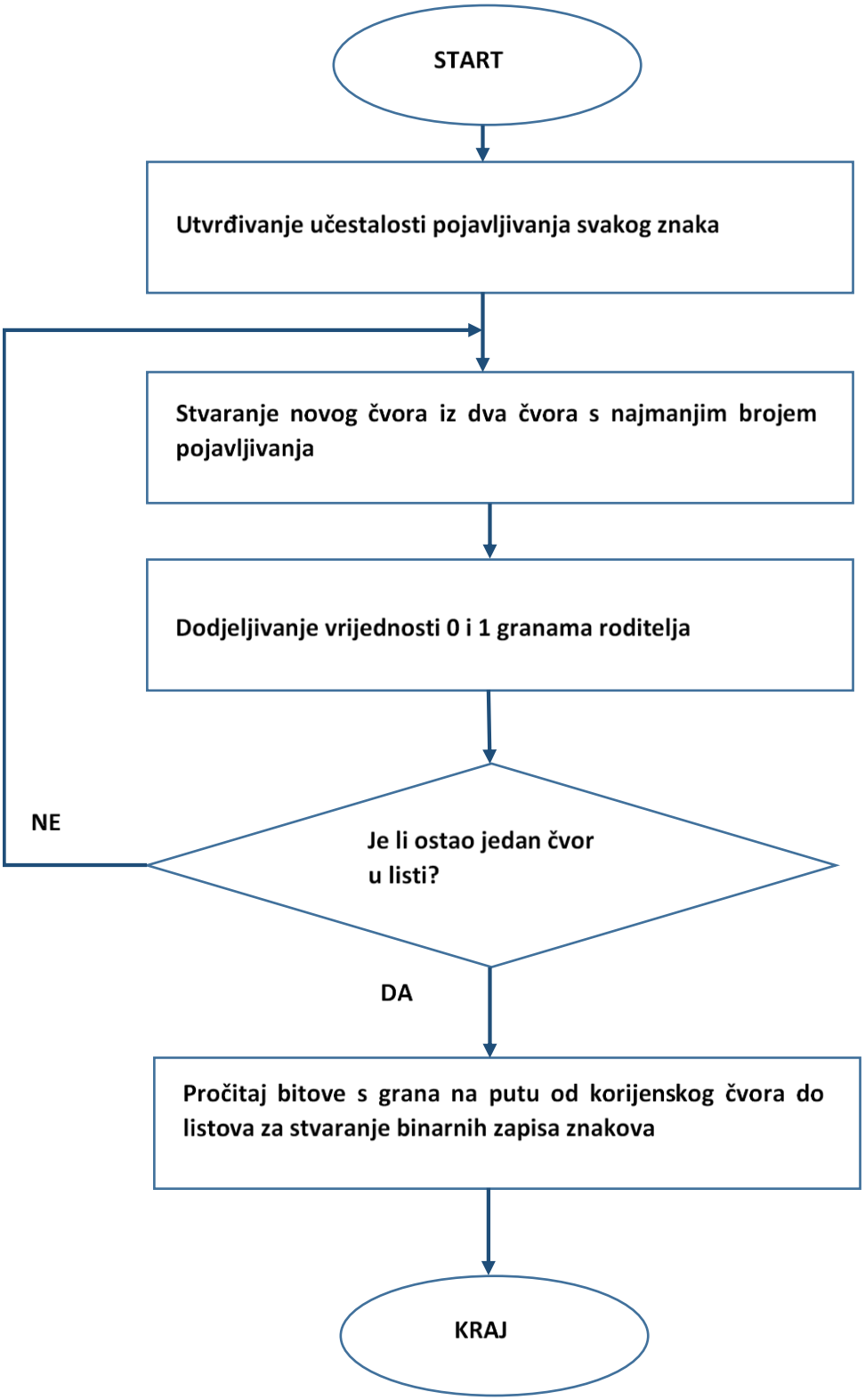
Slika 2.2. Razlika PNG i JPG formata [17]

Slike bez kompresije nemaju široku primjenu kao JPG i PNG datoteke zbog svoje veličine. Poznati format bez kompresije je BMP format. Datoteke s nastavkom BMP predstavljaju *bitmap* datoteke koje se koriste za pohranu *bitmap* digitalnih slika. Te su slike neovisne o grafičkom adapteru i format su slike koji je neovisan o uređaju [18].

2.3. Huffman Coding algoritam

Huffman Coding algoritam jedan je od onih koji se često koriste u kompresiji bez gubitaka. Huffmanovo kodiranje koristi kodiranje promjenjive duljine što znači da se binarni zapis simbola temelji na tome koliko se često taj simbol pojavljuje [19]. Njegova glavna ideja je da se informacije koje se najčešće pojavljuju zapišu najkraće.

Prvi korak algoritma je utvrđivanje učestalosti pojavljivanja svakog znaka u promatranom nizu. Učestalost pojavljivanja je proporcionalna vjerojatnosti pojavljivanja. Svaki znak sa svojim brojem pojavljivanja predstavlja jedan čvor koji se za početne znakove naziva list. Drugi korak algoritma je stvaranje novog čvora iz dva čvora s najmanjim brojem pojavljivanja. Dva čvora se uklanjaju iz liste i postaju lijevo i desno dijete. Novonastali čvor je roditelj koji se dodaje u listu, a učestalost njegovog pojavljivanja je zbroj pojavljivanja njegove djece. U trećem koraku grane koje vode do djece dobivaju vrijednosti. Lijeva grana čvora dobiva vrijednost 0, a desna grana čvora vrijednost 1. Sljedeći korak je provjera je li ostao samo jedan čvor. Ako nije, algoritam se vraća na drugi korak, a ako je ostao jedan čvor, iz nastalog stabla se mogu saznati kodovi za svaki znak. Na slici 2.3. prikazan je blok dijagram opisanog procesa.



Slika 2.3. Blok dijagram Huffmanovog algoritma

2.4. Primjer Huffman Coding algoritma

Primjer Huffman Coding algoritma izveden je za niz prvih šest slova engleske abecede. Prvi korak algoritma je definiranje broja pojavljivanja svakog znaka u nizu. Broj pojavljivanja definiran je u tablici 2.1.

Tablica 2.1. Frekvencija pojavljivanja znakova

A	B	C	D	E	F
12	4	21	14	9	18

Nakon što se slova poredaju po frekvenciji pojavljivanja, cilj je napraviti Huffmanovo stablo, a to je binarno stablo u kojem svaki list stabla odgovara slovu u danoj abecedi. Abeceda u primjeru ima samo šest slova, ali abecedu Huffmanovog stabla mogu činiti i brojevi, znakovi ili nizovi znakova. Stablo se sastoji od čvorova, a svaki od njih može biti povezan na lijevo i desno dijete. Dijete može biti drugi čvor u stablu ili list što znači je posljednji čvor s informacijom koja se kodira.

Sljedeći korak je traženje dva znaka koji se najmanje pojavljuju. To su slova B i E s frekvencijama 4 i 9. Njihovi brojevi pojavljivanja će se zbrojiti i stvorit će se novi čvor.

Novonastali čvor se uvrštava u tablicu s frekvencijama, a dva lista čije su vrijednosti zbrojene se uklanjaju iz tablice. Dva uklonjena čvora postaju lijevo i desno dijete novonastalom čvoru koji im je roditelj.

Postupak se ponavlja dok se ne završi s jednim čvorom koji sadrži zbroj svih vrijednosti pojavljivanja. Ako se želi doći do vrijednosti nekog znaka, tada se ide po stablu počevši od korijenskog čvor. Na putu do vrijednosti prati se preko koje djece se prolazi da bi se nastavio put. Ukoliko put ide preko lijevog djeteta, nizu koji predstavlja traženi znak dodaje se 0, a ukoliko put vodi preko desnog djeteta nizu se dodaje 1.

U tablicu 2.2. uvrštava se novi čvor s proizvoljnim imenom R1 i postupak se nastavlja dok ne ostane samo jedan element koji predstavlja korijenski čvor. Stablo koje se dobije za dane vrijednosti je prikazano na slici 2.5. Iz njega se mogu dobiti binarni nizovi koji predstavljaju znakove zadane abecede. Binarni niz znaka definiran je putom od korijenskog čvora do odgovarajućeg lista.

Tablica 2.2. Stvaranje roditeljskog čvora R1 iz listova B i E

C	F	D	R1	A
21	18	14	13	12

Tablica 2.3. Stvaranje roditeljskog čvora R2 iz lista A i čvora R1

R2	C	F	D
25	21	18	14

Tablica 2.4. Stvaranje roditeljskog čvora R3 iz listova F i D

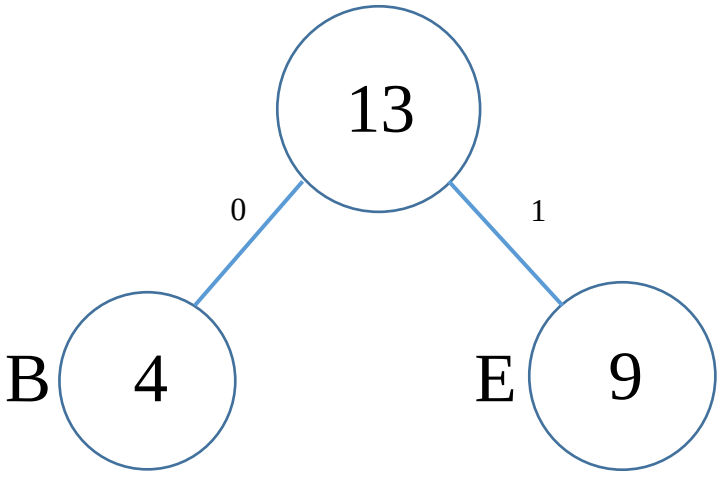
R3	R2	C
32	25	21

Tablica 2.5. Stvaranje roditeljskog čvora R4 iz lista C i čvora R2

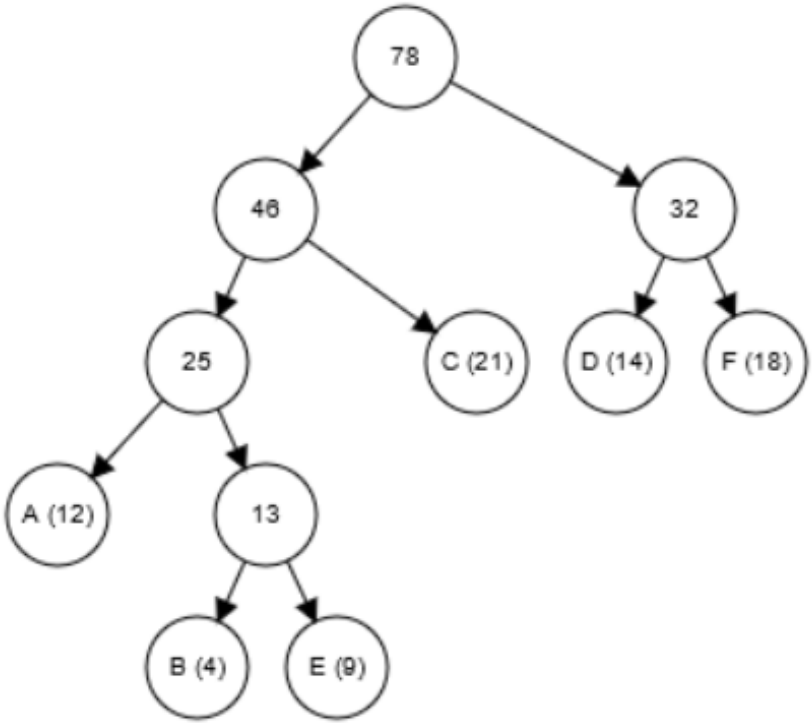
R4	R3
46	32

Tablica 2.6. Stvaranje korijenskog čvora R5 iz čvorova R4 i R3

R5
78



Slika 2.4. Izgled Huffmanovog stabla s jednim čvorom i dva lista



Slika 2.5. Izgled Huffmanovog stablo iz primjera

U tablici 2.7. nalaze se binarni zapisi slova i njihov broj pojavljivanja. Vidljivo je da su slova C, D i F zapisana s dva bita zato što se najčešće pojavljuju, a slova B i E s četiri zato što se pojavljuju najrjeđe. Kako bi se početni niz sa zadanom abecedom prikazao u računalu potrebna su 3 bita za svaki znak i takav niz bi zauzimao 234 bita. Taj isti niz na kojem je primijenjen algoritam zauzima 164 bita što je smanjenje zapisa za 30% bez gubitka podataka.

Tablica 2.7. Binarni zapis slova

A	B	C	D	E	F
12	4	21	14	9	18
000	0010	01	10	0011	11

2.5. Nedostatci Huffman Coding algoritma

Huffmanovo kodiranje predstavljeno binarnom strukturom stabla postupno postaje raspršeno kako raste iz korijena. Ta raspršenost obično uzrokuje ogromno gubljenje memorijskog prostora, osim ako se ne usvoji dobro strukturiran postupak za učinkovitu dodjelu simbola u memoriji. Osim toga, ova raspršenost može rezultirati i dugim postupkom pretraživanja za pronalaženje simbola, što dovodi do povećanja odgode [2].

Budući da su duljine svih binarnih kodova različite, programskoj podršci za dekodiranje postaje teško otkriti jesu li kodirani podaci oštećeni. To može rezultirati pogrešnim dekodiranjem i posljedično pogrešnim izlazom [1].

Nizovi kodirani *Huffman Coding* algoritmom moraju imati pripadajuće Huffmanovo stablo kako bi se mogao rekonstruirati izvorni niz. Huffmanovo stablo sadrži dodatne informacije koje ne prenose poruku, ali moraju se pohraniti. Navedeni nedostatak je razlog zapisa teksta i bitova u istu tekstualnu datoteku u implementaciji aplikacije iz ovog rada.

3. KOMPRESIJA SLIKA POMOĆU HUFFMAN CODING ALGORITMA

Kompresija slike *Huffman Coding* algoritmom implementirana je pomoću programskog jezika *Python*. Algoritam slijedi korake koji su opisani u prethodnom poglavlju i na slici 2.3.

3.1. Kod za kompresiju slike

Za učitavanje slike i dohvaćanje vrijednosti piksela korištena je biblioteka *OpenCV*. Biblioteka ima više od 2500 optimiziranih algoritama, što uključuje sveobuhvatan set klasičnih i najsuvremenijih algoritama za računalni vid i strojno učenje. Ti se algoritmi mogu koristiti za otkrivanje i prepoznavanje lica, prepoznavanje objekata, klasificiranje ljudskih radnji u video zapisima, praćenje kretanja kamere itd. *OpenCV* ima više od 47 tisuća korisnika zajednice i procijenjeni broj preuzimanja koji premašuje 18 milijuna. Biblioteka se koristi u tvrtkama, istraživačkim skupinama i državnim tijelima [9].

Slika 3.1. prikazuje učitavanje *OpenCV* biblioteke u prvom retku i ostatak koda je funkcija koja učitava sliku prema imenu ili prema putu slike na računalu. Slika se učitava sa zastavicom (engl. *flag*) IMREAD_UNCHANGED kako bi učitana slika imala isti broj kanala kao original. Takav način učitavanja omogućuje učitavanje četvrtog kanala za prozirnost ili učitavanje jedinog kanala slike sa sivim nijansama. Učitana slika zato nema gubitka podataka ili dodanih podataka.

Linija ***Kod***

```
1:      import cv2
2:      def loadImage(imageName):
3:           image=cv2.imread(imageName,cv2.IMREAD_UNCHANGED)
4:           return image
```

Slika 3.1. Kod za učitavanje slike pomoću OpenCV biblioteke

Sljedeći korak u algoritmu je traženje frekvencija. Budući da je svaki kanal u pikselu prikazan cijelim brojevima od 0 do 255, u rječniku (engl. *dictionary*), koji ima ulogu liste u programskom jeziku *Python*, spremiti će se svako novo pojavljivanje. Na slici 3.2. prikazana je funkcija koja prima sliku otvorenu s *OpenCV* bibliotekom, zatim u drugoj liniji koda dohvaća dimenzije slike. Za svaku dimenziju postoji jedna petlja kako bi se pristupilo svim vrijednostima. Ako trenutna vrijednost postoji, u 11. liniji koda joj se pribroji svako novo pojavljivanje, a ako vrijednost još ne postoji, u 13. liniji se stvara nova vrijednost kojoj je broj pojavljivanja 1. Kada se petlja izvrši, vrijednosti se sortiraju silazno prema drugom elementu u rječniku koji je broj pojavljivanja i funkcija vraća uređenu listu.

Linija ***Kod***

```
1:      def calculateFrequency(image):
2:        height, width, channels= image.shape
3:        frequency={}
4:        x=0
5:        while x<height:
6:           y=0
7:           while y<width:
8:              z=0
9:              while z<channels:
10:                if str(image[x, y][z]) in frequency:
11:                   frequency[str(image[x, y][z])]+=1
12:                else:
13:                   frequency[str(image[x, y][z])]=1
14:                z+=1
15:             y+=1
16:          x+=1
17:       frequency=sorted(frequency.items(),key=lambda x: x[1], reverse=True)
18:       return frequency
```

Slika 3.2. Određivanje frekvencije pojavljivanja

Sljedeći korak je stvaranje čvorova od dvije vrijednosti s najmanje pojavljivanja. Uz frekvencije pojavljivanja kao ulazne vrijednosti, funkcija na slici 3.3. stvara novi čvor od zadnje dvije vrijednosti u poredanoj listi, uklanja te dvije vrijednosti i u listu stavlja novi čvor nakon čega se svi elementi poredaju po broju pojavljivanja. Dva elementa koji su uklonjeni iz liste spremljeni su kao djeca novog čvora. Čvor je predstavljen klasom koja ima dvije varijable za lijevo i desno dijete i dvije funkcije od kojih je jedna konstruktor, a druga dohvaća varijable djece. Promatrana funkcija na kraju vraća korijenski čvor iz kojeg se može pristupiti svim vrijednostima u stablu.

Linija ***Kod***

```
1:      def makeNodes(frequency):
2:         nodes=frequency
3:         while len(nodes) > 1:
4:            (key1, c1) = nodes[-1]
5:            (key2, c2) = nodes[-2]
6:            nodes = nodes[:-2]
7:            node = myClass.NodeTree(key1, key2)
8:            nodes.append((node, c1 + c2))
9:            nodes = sorted(nodes, key=lambda x: x[1], reverse=True)
10:        return nodes
```

Slika 3.3. Stvaranje stabla i čvorova

Na slici 3.4. prikazan je sljedeći korak kompresije u kojem se dohvaćaju binarni nizovi svake vrijednosti. Ova rekurzivna funkcija prima korijenski čvor i iz njega se poziva na svako sljedeće dijete. Rekurzija se odvija sve dok nije ispunjen uvjet da je tip čvora tekst nakon čega se u rječnik sprema uređeni par s tekstom koji predstavlja kodirani broj od 0 do 255 i binarni niz. Jedan bit se dodaje svakim novim pozivom funkcije, a ovisno o tome poziva li se na lijevom ili desnom djetetu, trenutnom nizu bitova će biti dodano 0 ili 1.

Linija ***Kod***

```
1:      def huffmanCodeTree(node, bitString=''):
2:          if type(node) is str:
3:              return {node: bitString}
4:          (l, r) = node.children()
5:          d = dict()
6:          d.update(huffmanCodeTree(l,bitString + '0'))
7:          d.update(huffmanCodeTree(r,bitString + '1'))
8:          return d
```

Slika 3.4. Stvaranje binarnih vrijednosti

Nakon što su poznate sve vrijednosti za kompresiju slike, potrebno je svaku vrijednost u pikselu zapisati kao binarni niz i spojiti sve zapise u jedan dugi binarni niz kako bi se mogao zapisati u datoteku. Na slici 3.5. je funkcija koja kodira sliku i vraća binarni niz u kojem su zapisane sve vrijednosti piksela, a ulazne varijable su slika učitana bibliotekom *OpenCV* i rječnik s binarnim vrijednostima brojeva koji se nalaze u pikselima. Ključna je 10. linija u kojoj se za svaku vrijednost iz piksela od 0 do 255 traži odgovarajuća binarna vrijednost iz rječnika i ona se dodaje ukupnom binarnom zapisu.

Linija ***Kod***

```
1:      def getImageString(image, huffmanCode):
2:         height, width, channels= image.shape
3:         imageBits=''
4:         x=0
5:         while x<height:
6:            y=0
7:            while y<width:
8:               z=0
9:               while z<channels:
10:                 imageBits+=str(huffmanCode[str(image[x, y][z])])
11:                 z+=1
12:              y+=1
13:           x+=1
14:        return imageBits
```

Slika 3.5. Stvaranje binarnog zapisa slike

3.2. Zapis kompresije u datoteku

U datoteci gdje se zapisuje dobiveni rezultat moraju biti zapisane dodatne informacije kako bi se slika mogla rekonstruirati. Prvu informaciju u datoteci čini tekst koji definira osnovna svojstva slike; visinu, širinu i broj kanala. Nakon navedenih informacija slijedi tekst koji svakom broju u zadanom rasponu daje pripadajući binarni niz. Na slici 3.6. prikazana je funkcija kojoj su ulazne vrijednosti naziv ili put datoteke na računalu, učitana slika, i dva rječnika s frekvencijama pojavljivanja i binarnim zapisima brojeva. U drugoj liniji koda se otvara dokument za pisanje, zatim se zapisuju svojstva slike nakon čega se u petlji zapisuju binarni kodovi za svaki broj silazno po frekvenciji. Zadnja zapisana linija u šestom retku označava mjesto u datoteci gdje počinje binarni zapis slike.

Linija ***Kod***

```
1:      def writeHuffmanFile(name, image, frequency, huffmanCode):
2:          huffmanFile = open(name,"w")
3:          huffmanFile.write('%s\n%s\n%s\n' % (image.shape))
4:          for (number, frequency) in frequency:
5:             huffmanFile.write('%s=>%s\n' % (number, huffmanCode[number]))
6:          huffmanFile.write('imageCode:\n')
7:          huffmanFile.close()
```

Slika 3.6. Zapisivanje informacija za dekompresiju

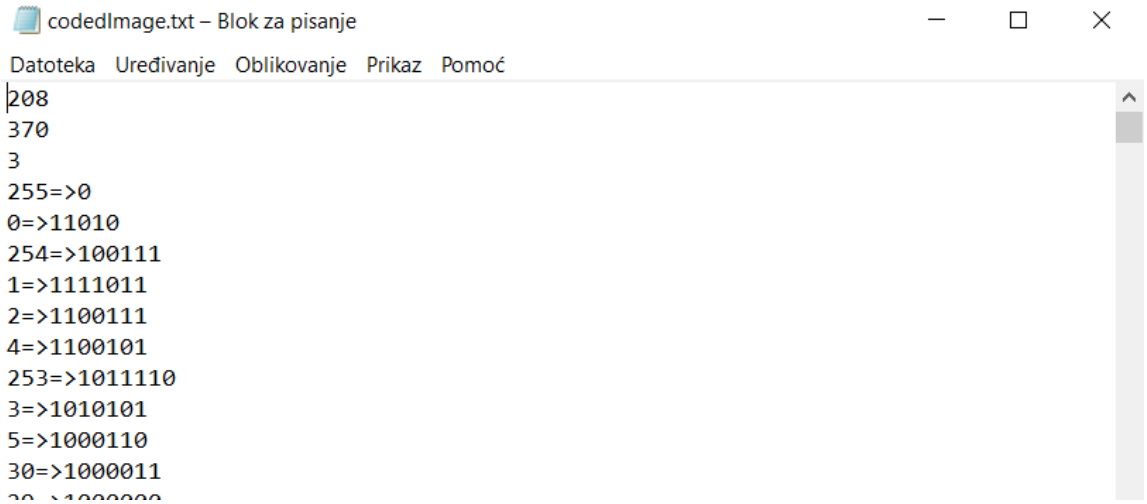
Na slici 3.7. prikazana je funkcija koja zapisuje binarni niz u datoteku. U trećoj liniji koda datoteka se otvara u načinu za pisanje bitova i za nastavak pisanja u datoteku. Za zapis binarnog niza korištena je biblioteka *bitarray*. Ova biblioteka sadrži tip objekta koji učinkovito predstavlja niz logičkih vrijednosti. *Bitarray* je sekvencijalan tip podatka i ponaša se poput uobičajenih lista. Osam bitova predstavljeno je jednim bajtom u susjednom bloku memorije [20]. *Bitarray* podijeli niz bitova u blokove od osam i tako se zapisuju u datoteku. Nakon što je zapis informacija gotov, veza s datotekom se zatvara i završna datoteka ima sve potrebne upute za dekompresiju slike.

Linija ***Kod***

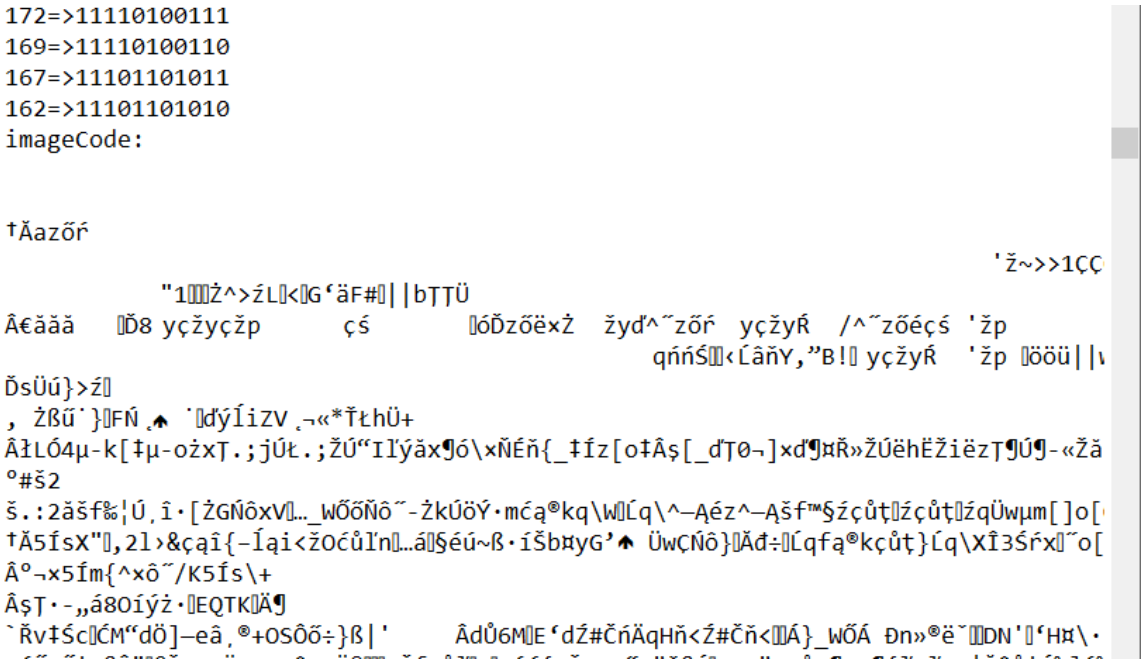
```
1:      def writeImageFile(name, image, huffmanCode):
2:          imageBits=getImageString(image, huffmanCode)
3:          imageFile = open(name,"ba")
4:          bitWriter=bitarray(imageBits)
5:          bitWriter.tofile(imageFile)
6:          imageFile.close()
```

Slika 3.7. Zapis binarnog niza

Slike 3.8. i 3.9. prikazuju izgled zapisanih informacija u tekstualnoj datoteci. Prvi dio zapisa je jasno čitljiv zato što je u datoteku zapisan tekst po dogovorenom standardu. Drugi dio datoteke u koji je zapisan binarni niz nema tekstualnog značenja. Znakovi se pojavljuju nasumično jer kodirani brojevi imaju promjenjivu veličinu binarnog zapisa. Binarni zapisi predstavljaju put na Huffmanovom stablu, a ne znakove po dogovorenom standardu.



Slika 3.8. Početni zapis datoteke



Slika 3.9. Zapis slike u datoteci

3.3. Kod za rekonstrukciju slike

Rekonstrukcija slike implicira uspješnu kompresiju slike. Rekonstrukcija slike sastoji se od dvije glavne funkcije od kojih jedna čita prvi dio datoteke s uputama za dekodiranje dok druga funkcija radi rekonstrukciju slike iz drugog dijela datoteke u kojem je kodirana originalna slika.

Na slici 3.10. prikazan je kod koji dohvaća informacije o slici i kodirane vrijednosti svih brojeva koje se pojavljuju. Datoteka se otvara uz pomoć imena datoteke ili puta datoteke u računalu u načinu za čitanje u binarnom obliku. Nakon očitanih vrijednosti visine, širine i broja kanala u redovima 3, 4 i 5, funkcija dohvaća brojeve i njihove binarne zapise u kompresiji. U tom postupku uklanja oznake i navodne znakove kako bi se tražene vrijednosti mogle upisati u rječnik. Na kraju funkcija vraća otvorenu datoteku za daljnje čitanje, dimenzije slike i rječnik s vrijednostima uz pomoć kojih će se slika rekonstruirati.

Linija ***Kod***

```
1:      def readHuffmanCodes(fileName):
2:        readingFile=open(fileName,"rb")
3:        height=int(readingFile.readline())
4:        width=int(readingFile.readline())
5:        channels=int(readingFile.readline())
6:        huffmanValues={}
7:        huffmanCodes=str(readingFile.readline())
8:        huffmanCodes=huffmanCodes.replace("'","").replace('b','').replace(r'\r\n',r'')
9:        while huffmanCodes != ('imageCode:'):
10:         value=huffmanCodes.split('=>')
11:         huffmanValues[int(value[0])]=value[1]
12:         huffmanCodes=str(readingFile.readline())
13:         huffmanCodes=huffmanCodes.replace("'","").replace('b','').replace(r'\r\n',r'')
14:       return [readingFile,height,width,channels,huffmanValues]
```

Slika 3.10. Čitanje informacija za dekompresiju

Slika 3.11. prikazuje prvi dio funkcije u kojoj se slika rekonstruira. Funkcija prima ime ili put do datoteke u kojoj je slika zapisana i isto za sliku koja će nastati. Za osnovne informacije poziva se funkcija sa slike 3.10. i stvara se nova slika s odgovarajućom rezolucijom i brojem kanala. Za stvaranje nove slike korištena je PIL biblioteka. PIL je besplatna i *open-source* biblioteka za programski jezik *Python* koja daje podršku za otvaranje, manipuliranje i spremanje mnogih različitih formata slika [10]. Prazna slika se sprema na željeno mjesto nakon stvaranja. Sljedeći korak je čitanje binarnog niza u kojem se nalaze vrijednosti bitova. Čitač datoteke je stao na zadnjoj liniji prvog dijela datoteke te se ostatak informacija učitava kao binarni niz u 13. redu koda i nakon toga se veza s datotekom zatvara.

Linija Kod

```
1:      def reconstructImage(imageName, fileName):
2:        [readingFile,height,width,channels,huffmanValues]=readHuffmanCodes(f
3:        if channels==4:
4:          image=Image.new( mode = "RGBA", size = (width, height) )
5:        elif channels==1:
6:          image=Image.new( mode = "L", size = (width, height) )
7:        elif channels==2:
8:          image=Image.new( mode = "LA", size = (width, height) )
9:        else:
10:         image=Image.new( mode = "RGB", size = (width, height) )
11:       image=image.save(imageName)
12:       bitReader=bitarray()
13:       bitReader.fromfile(readingFile)
14:       readingFile.close()
```

Slika 3.11. Rekonstrukcija slike – stvaranje prazne slike

Drugi dio funkcije za dekompresiju je prikazan na slici 3.12. U njemu se vrijednosti piksela u stvorenoj slici postavljaju na vrijednosti izvorne slike. Za postavljanje vrijednosti piksela i spremanje promjena u sliku korištena je biblioteka *OpenCV*. Algoritam učitava bit i dodaje ga ostatku učitanih bitova te traži postoji li takav niz u kodiranim vrijednostima. Ako ne postoji, dodaje sljedeći bit i provjerava ponovno bilježeći pri tome poziciju zadnjeg očitanog bita. Ako postoji takav niz, traži se broj koji je predstvaljen dobivenim nizom i broj se zapisuje u sljedeći prazni piksel na odgovarajuće mjesto. Na kraju procesa se pohranjuju sve promjene u prijašnje stvorenu praznu sliku.

Linija ***Kod***

```
1:        image=cv2.imread(imageName,cv2.IMREAD_UNCHANGED)
2:        stack=''
3:        p=0
4:        x=0
5:        while x<height:
6:          y=0
7:          while y<width:
8:            z=0
9:            while z<channels:
10:             while not isEqual(stack, huffmanValues):
11:               stack=stack+str(bitReader[p])
12:               p+=1
13:             image[x, y][z]=findEqual(stack, huffmanValues)
14:             stack=''
15:             z+=1
16:           y+=1
17:         x+=1
18:       cv2.imwrite(imageName,image)
```

Slika 3.12. Rekonstrukcija slike – popunjavanje prazne slike

3.4. Aplikacija

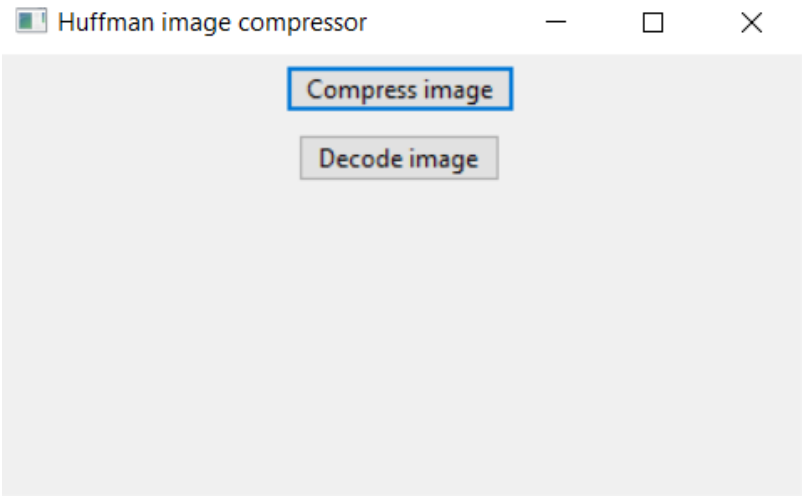
Kako bi se aplikacija za kompresiju slike brže pokretala, opisani kod je implementiran u aplikaciju s dva gumba od kojih jedan stvara kompresiju slike, a drugi stvara rekonstruiranu sliku. Kod aplikacije je modificirani kod iz izvora [11]. Dvije funkcije koje se pozivaju prikazane su na slici 3.13. i u njihovom izvođenju obuhvaćen sav prijašnje opisan kod. Obje funkcije kao ulazne parametre primaju dva puta do datoteke; iz jedne datoteke čitaju informacije, a drugu datoteku stvaraju i u nju zapisuju informacije.

Linija ***Kod***

```
1:      def codeImage(imagePath, filePath):
2:          image=function.loadImage(imagePath)
3:          frequency=function.calculateFrequency(image)
4:          nodes=function.makeNodes(frequency)
5:          huffmanCode = function.huffmanCodeTree(nodes[0][0])
6:          function.writeFile(filePath,image,frequency,huffmanCode)
7:
8:      def decodeFile(filePath, imagePath):
9:          function.reconstructImage(imagePath,filePath)
```

Slika 3.13. Funkcije za kompresiju i rekonstrukciju slike

Pritiskom na jednu od tipki sa slike 3.14. otvara se prozor u kojem se bira datoteka iz koje će se uzimati informacije, a nakon odabiranja datoteke zatvara se trenutni prozor i otvara se novi koji traži ime i mjesto spremanja nove datoteke. U aplikaciji se kompresija slike može primjenjivati samo na slikama BMP formata zato što taj format ne koristi kompresiju.



Slika 3.14. Izgled aplikacije

3.5. Dobiveni rezultati i njihova usporedba

U tablici 3.1. prikazana je usporedba za kompresije na slici 3.15. s dubinom slike od 24 bita. Različite slike će davati različite omjere i rezultate, a u odabranoj slici dominiraju dvije boje; bijela i crvena te je time potencijal za kompresiju veći i ušteda prostora će biti dobro izražena. Rezultati su očekivani budući da JPG format gubi dio informacija u kompresiji, a PNG format primjenjuje

kompresiju kombinacijom više algoritama uključujući i Huffmanov. *Huffman Coding* algoritam je u ovom slučaju smanjio veličinu za približno jednu trećinu veličine izvorne slike.

Slika 3.15. Primjer slike korištene za kompresiju [21]

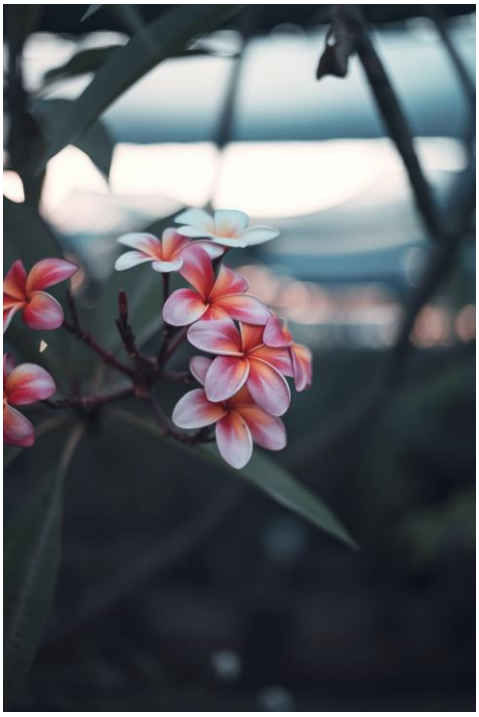
Tablica 3.1. Usporedba kompresija za slike s različitim brojem kanala

	BMP	JPG	PNG	Huffman Coding
Veličina u bajtovima	231.350	19.631	74.964	155.929
Kompresija	0%	91,52%	67.60%	32.60%

3.6. Primjeri kompresije Huffman Coding algoritmom

3.6.1. Kompresija slika s različitim brojem kanala

Usporedbom tri slike ispitana je učinkovitost kompresije algoritma ovisno o broju kanala. Izvorna slika 3.16. zapisana je u BMP formatu tako da izvedene slike koriste jedan, tri i četiri kanala. Dubina bitova u računalu za te slike je 8, 24 i 32 bita. Iz tablice s rezultatima 3.2. vidljivo je da je najveća kompresija ostvarena na najvećoj slici dok su kompresije za slike s jednim i tri kanala slične. Kompresija sve tri slike je manja u usporedbi s kompresijom za sliku 3.15. koja je manje kvalitete i ima manje različitih boja. Budući da se na uobičajenoj slici svaki od 256 brojeva većinom pojavi barem jednom i kodiranje ovisi o omjeru njihova pojavljivanja, iz rezultata se može naslutiti da kompresija nije ovisna o broju kanala koji slika ima, već o veličini slike što je ispitano u sljedećem poglavlju.



Slika 3.16. Izvorna slika [22]

Tablica 3.2. Usporedba kompresija za slike s različitim brojem kanala

Broj kanala	1	3	4
Veličina slike	1.499.078	4.494.698	5.992.698
Veličina nakon kompresije	1.340.651	4.061.270	4.666.378
Kompresija	10,57%	9,64%	22,13%

3.6.2. Kompresija slika s različitim rezolucijama

Kako bi se ispitao utjecaj količine informacija na kompresiju *Huffman Coding* algoritma, visina i širina testne slike 3.17. smanjene su na 80%, 60%, 40% i 20%. Početna širina je 1080, a visina 720 piksela. Rezultati iz tablice 3.3. pokazuju da je kompresija u ovom primjeru učinkovitija za veće slike. Objašnjenje se može naći u tome da slike s više informacija imaju i veći potencijal kompresije, ali ovaj primjer ne predstavlja pravilo zato što kompresija ovisi o bojama i učestalosti pojavljivanja svake vrijednosti te se može dogoditi da manja slika ima veću kompresiju. Sličan primjer vidljiv je u tablici 3.2. gdje slična slika sa sivim tonovima uz manje informacija ima bolju kompresiju od slike s više informacija.

Slika 3.17. Testna slika [23]

Tablica 3.3. Usporedba kompresija za slike s različitim veličinama

Rezolucija	100%	80%	60%	40%	20%
Veličina slike	3.110.454	1.990.710	1.119.798	497.718	124.470
Veličina nakon kompresije	2.637.345	1.689.343	951.854	425.214	109.135
Kompresija	15,21%	15,14%	15,00%	14,57%	12,32%

4. ZAKLJUČAK

Zadatak završnog rada bio je istražiti i opisati teorijske osnove kompresije slika i kompresije slika *Huffman Coding* algoritmom. Bilo je potrebno dati kratak pregled područja te implementirati *Huffman Coding* algoritam za primjenu u kompresiji slike pomoću programskog jezika *Python* i biblioteke *OpenCV*. Implementirani algoritam testirati na proizvoljno odabranim testnim slikama.

Kompresija je neizostavni dio digitalnog svijeta i većina datoteka koristi neki oblik kompresije. Primjena u multimediji je neizbježna zbog velikog broja informacija koji se lako i brzo stvara te različiti nastavci datoteka poput JPG i PNG označavaju formate od kojih svaki koristi jedinstvenu kompresiju. Neki formati gube dio informacija kako bi maksimalno smanjili veličinu zapisa, neki ne gube informacije po cijenu veće datoteke, a rijetki nemaju nikakve kompresije. Format slike koji je korišten za testiranje algoritma je BMP i on nema kompresije.

Huffmanov algoritam se lako primjenjuje na kratkim nizovima kao u primjeru iz rada, ali za kompresiju slika u računalu, nizovi znakova su zamijenjeni dugim nizovima brojeva. Najveći dio napisanog koda nije *Huffman Coding* algoritam, već kod koji dohvaća, zapisuje i rukuje informacijama koje su potrebne da bi se algoritam primijenio. *OpenCV* biblioteka korištena je kako bi se svaka cjelobrojna vrijednosti iz piksela mogla očitati i pohraniti za kasniju kompresiju.

Kompresija je zapisana u tekstualnu datoteku uz ključne informacije za dekompresiju. Za zapisivanje dobivenog binarnog niza u datoteku korištena je biblioteka *bitarray*. Rekonstrukcija slike je dokaz uspješne kompresije, a ona je postignuta uz pomoć PIL i *OpenCV* biblioteka. Zbog bržeg testiranja i pokretanja algoritma, sav kod je obuhvaćen u dvije funkcije, jedna za kompresiju i druga za rekonstrukciju, te je svakoj pridružen jedan gumb u aplikaciji.

U usporedbi *Huffman Coding* algoritma s PNG datotekom koja također ima kompresiju bez gubitaka, PNG datoteka ima uvjerljivo bolju kompresiju i dokaz je kako se Huffmanov algoritam može kombinirati s drugim algoritmima za postizanje boljih rezultata. Iz rezultata kompresije na testnim slikam ne mogu se donositi zaključci o kompresiji *Huffman Coding* algoritma zato što je potrebno proučavanje na većem broju slika. Svaka slika ima drugačije Huffmanovo stablo, a time i binarni zapis vrijednosti.

Neka od ograničenja napisanog algoritma za kompresiju su brzina izvođenja za velike slike, kompresija slika samo nekih BMP formata i rekonstrukcija slike traje mnogo dulje nego kompresija.

LITERATURA

[1] „Huffman Coding Algorithm“. https://www.studytonight.com/data-structures/huffman-coding

[2] R. Hashemian, „Direct Huffman coding and decoding using the table of code-lengths“, 2003, str. 237–241. doi: 10.1109/ITCC.2003.1197533.

[3] Bhonde Nilesh, Shinde Sachin, Nagmode Pradip, i D.B. Rane, „Image Compression Using Discrete Wavelet Transform“. IJCTEE, tra. 2013.

[4] „Run-length encoding“, *Wikipedia*. srp. 07, 2021. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Run-length_encoding&oldid=1032487509

[5] „Arithmetic coding“, *Wikipedia*. srp. 27, 2021. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Arithmetic_coding&oldid=1035769726

[6] „Transform coding“, *Wikipedia*. kol. 09, 2021. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Transform_coding&oldid=1037931082

[7] „Huffman Coding Algorithm - Programiz“. https://www.programiz.com/dsa/huffman-coding

[8] „Image Compression using Huffman Coding“, *GeeksforGeeks*, tra. 30, 2018. https://www.geeksforgeeks.org/image-compression-using-huffman-coding/

[9] „About OpenCV“, *OpenCV*. https://opencv.org/about/

[10] „Python Imaging Library“, *Wikipedia*. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Python_Imaging_Library&oldid=1026128964

[11] „python - wxPython New, Save, and SaveAs Methods“, *Stack Overflow*. https://stackoverflow.com/questions/30483250/wxpython-new-save-and-saveas-methods

[12] T. wxPython Team, „Welcome to wxPython!“, *wxPython*, pros. 24, 2019. https://wxpython.org/index.html (pristupljeno srp. 21, 2021).

[13] „Lossless compression“, *Wikipedia*. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Lossless_compression&oldid=1033439086

[14] „PNG File Extension - What is a .png file and how do I open it?“ https://fileinfo.com/extension/png (pristupljeno ruj. 17, 2021).

[15] „Lossy compression“, *Wikipedia*. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Lossy_compression&oldid=1029400741

[16] „What’s the difference between a PNG and a JPEG? – ReadyTechGo“. https://www.readytechgo.com.au/whats-the-difference-between-a-png-and-a-jpeg/

[17] „CafePress.com : Beginners Image Workshop“. https://www.cafepress.com/cp/info/sell/index.aspx?area=help_images&page=help_images

[18] K. Iqbal, „BMP - Image File Format“. https://docs.fileformat.com/image/bmp/

[19] Sadia Yunas Butt, „HUFFMAN CODING“. 2015.

[20] I. Schnell, *bitarray: efficient arrays of booleans -- C extension*. [Na internetu]. Dostupno na: https://github.com/ilanschnell/bitarray

[21] „swift car“. https://www.ckkadsp.com/products.aspx?cname=swift+car&cid=5

[22] „Small Flowers Pictures“. https://unsplash.com/s/photos/small-flowers

[23] „Tvrdja fortress and Osijek old town“. https://www.outdooractive.com/en/poi/osjecko-baranjska-zupanija/tvrdja-fortress-and-osijek-old-town/51483761/

SAŽETAK

Cilj rada je opisati teorijske osnove kompresije slika i kompresije *Huffman Coding* algoritmom, a nakon toga implementirati *Huffman Coding* algoritam za kompresiju slika. Nakon objašnjene teorijske osnove kompresije i *Huffman Coding* algoritma uz primjer, kod implementiranog algoritma slijedi iste korake kao i prikazani primjer te je objašnjen istim redom. Algoritam je implementiram u *Python* programskom jeziku, a za dohvaćanje vrijednosti iz slike korištena je biblioteka *OpenCV*. Zapis u tekstualnu datoteku se odvija u dva koraka, prvo se zapisuju informacije o dimenzijama i informacije iz Huffmanovog stabla, a nakon toga slijed bitova koji nosi informacije slike i zapisan je pomoću *bitarray* biblioteke. Izvorna slika se može rekonstruirati iz tekstualne datoteke u koju je kompresija zapisana kao dokaz ispravne kompresije.

Ključne riječi: *Huffman Coding* algoritam, kompresija slike, *OpenCV*, *Python*

ABSTRACT

Image compression using Huffman Coding algorithm

The aim of the paper is to describe the theoretical foundations of image compression and compression by Huffman Coding algorithm followed by the implementation of the Huffman Coding algorithm for image compression. After explaining the theoretical foundations of compression and the Huffman Coding algorithm with a simple, the implemented algorithm follows the same steps as the presented example and is explained in the same order. Algorithm is implemented in the Python programming language and the OpenCV library was used to retrieve the values from the image. Writing to the text file takes two steps, first the dimension information and information from the Huffman tree are written, followed by the bit sequence that carries the image information and is written using the bitarray library. The original image can be reconstructed from text file in which the compression is written as proof of correct compression.

Key words: Huffman Coding algorithm, image compression, OpenCV, Python

PRILOG

Kod implementacije dostupan je na optičkom disku i linku:

https://github.com/VilimSvjetlicic/Kompresija-slika-pomocu-Huffman-Coding-algoritma.git