Quite OK Image Format (QOI) za kompresiju slike

Štimac, Ivana

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Diplomski sveučilišni studij računarstva

Diplomski rad

Quite OK Image Format (QOI) za kompresiju slike

Rijeka, srpanj 2023.

Ivana Štimac
0069085265

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Diplomski sveučilišni studij računarstva

Diplomski rad

Quite OK Image Format (QOI) za kompresiju slike

Mentor: izv.prof.dr.sc. Jonatan Lerga

Rijeka, srpanj 2023.

Ivana Štimac
0069085265

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 20. ožujka 2023.

Zavod: **Zavod za računarstvo**
Predmet: **Kodiranje i kriptografija**
Grana: **2.09.03 obradba informacija**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Ivana Štimac (0069085265)**
Studij: Sveučilišni diplomski studij računarstva
Modul: Programsko inženjerstvo

Zadatak: **Quite OK Image Format (QOI) za kompresiju slike / Quite OK Image Format (QOI) for Image Compression**

Opis zadatka:

Potrebno je implementirati Quite OK Image Format (QOI) za kompresiju slike i testirati isti na slikama različitih rezolucija. Algoritam je potrebno usporediti s drugim, često korištenim algoritmima za kompresiju slike u vidu omjera kompresije i brzine izvršavanja algoritama. Sam QOI algoritma je potrebno implementirati u programskom jeziku Python.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Ivana Štimac
Zadatak uručen pristupniku: 20. ožujka 2023.

Mentor:

Izv. prof. dr. sc. Jonatan Lerga

Predsjednik povjerenstva za diplomski ispit:

Prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradila ovaj rad.

Rijeka, srpanj 2023.

_____
Ivana Štimac

Zahvala

Zahvaljujem svojem mentoru izv.prof.dr.sc. Jonatanu Lergi na prenesenom znanju, poticanju samostalnog rada te korisnim savjetima tijekom implementacije i izrade diplomskog rada.

Veliko hvala svim kolegama i Kolegici koji su uvijek bili spremni pomoći tijekom izrade ovoga rada, ali i tijekom svih godina studiranja. Vaša podrška i pomoć su neprocjenjive i iznimno cijenjene.

Također, zahvaljujem svojoj obitelji i prijateljima na podršci, razumijevanju te strpljenju tijekom studiranja.

Sadržaj

Popis slika

Popis tablica

Poglavlje 1

Uvod

S neprekidnim napretkom tehnologije dolazi i do stalnog porasta količine generiranih podataka. Sve više dolazi do izražaja potreba za učinkovitom pohranom tih podataka, a posebno digitalne slike koja zauzima znatnu količinu računalne memorije. Stoga je cilj pronaći način kako što učinkovitije pohraniti podatke, istovremeno zadržavajući visoku kvalitetu podataka uz minimalnu moguću upotrebu računalne memorije. Upravo iz ovog razloga, sve veći fokus stavlja se na kompresiju podataka. Primjenom algoritama za kompresiju, osim što se smanjuje količina zauzetog memorijskog prostora, značajno se poboljšavaju i performanse prijenosa podataka. Također, smanjuju se troškovi i povećava iskoristivost dostupnih resursa.

Cilj ovoga rada je implementirati Quite OK Image Format, odnosno QOI algoritam u programskom jeziku Python. Navedeni algoritam je algoritam za kompresiju slike bez gubitaka, a način kodiranja, odnosno dekodiranja slike ovim algoritmom bit će detaljnije objašnjen u Poglavlju 2. Kod implementacije algoritma korištene su gotove knjižnice u programskom jeziku Python, kao što su *Image*, *copy* i *Numpy*, dok su u analizi performansi algoritama korištene knjižnice *time* i *sys*. Implementacija programskog koda bit će objašnjena u Poglavlju 3.

Drugi dio ovoga rada je testiranje efikasnosti QOI algoritma po pitanju brzine izvršavanja i omjera kompresije. Rezultati se uspoređuju s drugim često korištenim algoritmima za kompresiju slike, odnosno s Bzip2, Brotli i LZ4 algoritmima. Navedeni algoritmi su također algoritmi za kompresiju bez gubitaka. Bzip2 i Brotli algoritmi su složeni algoritmi koji se temelje na kombinaciji više osnovnih algoritama, dok LZ4 pripada LZ77 skupini algoritama, a detaljnije će biti objašnjeni u Poglavlju 2.

Ovisnost vremena izvršavanja QOI algoritma o veličini slike, usporedba perfor-

mansi algoritma na različitim formatima zapisa slike te usporedba rezultata mjerenja vremena izvršavanja i omjera kompresije svih navedenih algoritama bit će prikazani u Poglavlju 4.

Poglavlje 2

Metodologija

2.1 Digitalna slika

Digitalna slika je vizualizacija podataka čiji se elementi zovu pikseli. Pikseli su najmanje jedinice slike, imaju konačnu i diskretnu vrijednost te su raspoređeni u 2D matrici. Njihov broj u svakom retku označava širinu slike, njihov broj u svakom stupcu označava visinu slike, a rezolucija slike definira se kao visina x širina. Rezolucija označava sposobnost prikaza detalja slike, a što je ona veća, znači da slika može prikazati više detalja, odnosno da sadrži više informacija [1].

Postoje tri vrste zapisa slike: bitonalna slika (eng. *Bitonal*), crno - bijela slika (eng. *Grayscale*) te slika u boji. Bitonalna slika je 1-bitna slika gdje svaki piksel može poprimiti vrijednost 0 ili 1, najčešće 0 označava crnu, a 1 bijelu boju. Crno - bijele slike sastoje se od piksela koji su definirani s jednom vrijednošću, koja najčešće ima dva do osam bita. Slike korištene kod testiranja algoritama u ovom radu su slike u boji. One se sastoje od piksela koji imaju *r, g, b*, odnosno *Red, Green* i *Blue* komponente, koje označavaju intenzitet svake od pripadajućih boja. Piksel može imati i *a*, odnosno *Alpha* komponentu koja određuje njegovu prozirnost [2].

Svaka komponenta je najčešće 8-bitni broj koji prema tome može poprimiti vrijednosti od 0 do 255. U tom slučaju, svaki piksel kod RGB (*Red Green Blue*) slike ima veličinu od 24 bita (osam bita za svaku od tri komponente), odnosno tri bajta. Iz toga slijedi da je veličina slike u bajtima definirana prema izrazu (2.1), gdje je širina slike jednaka broju piksela po retku, a visina slike jednaka je broju piksela po stupcu [1].

$$\text{veličina slike} = \text{širina slike x visina slike x 3} \tag{2.1}$$

2.1.1 Formati zapisa digitalnih slika

Formati zapisa određuju način pohrane, dijeljenja te procesiranja digitalnih slika. Neki od najčešće korištenih formata, te njihove glavne značajke, prikazani su u tablici 2.1 [3].

Tablica 2.1 Formati zapisa digitalnih slika

Format	Puni naziv formata	Glavne značajke
JPG	*Joint Photographic Group*	mala datoteka, kompresija s gubitcima
TIF	*Tagged Image Format*	velika datoteka, visoka kvaliteta, bez kompresije ili kompresija bez gubitaka
PNG	*Portable Network Graphics*	Široko korišten, mala datoteka
GIF	*Graphics Interchange Format*	najčešće korišten za animacije, mala datoteka
RAW		skup formata slika koji su neprocesirani, bez kompresije (raw, cr2, nef, orf, sr2)
WebP	*Web Picture*	mala datoteka, visoka kvaliteta, kompresija s ili bez gubitaka

2.2 Kompresija slike

Kompresija slike je proces smanjenja veličine slike, odnosno broja bajtova koje ona zauzima. Time se smanjuje količina podataka koju je potrebno pohraniti, prenijeti ili preuzeti za tu sliku te se time povećava učinkovitost navedenih postupaka [4]. Ovim postupkom moguće je smanjenje broja piksela u dekompresiranoj slici,

u slučaju kompresije s gubitcima, dok kod kompresije bez gubitaka ostaje isti broj piksela. Navedene vrste kompresija bit će objašnjene u nastavku rada.

Važna karakteristika digitalnih slika je da su susjedni pikseli često u korelaciji što znači da sadrže redundantne informacije. Cilj kompresije je pronaći manje koreliranu reprezentaciju slike, čime se uklanjaju redundantne i/ili nepotrebne informacije. Smanjenje redundantnih informacija temelji se na uklanjanju ponavljajućih informacija, dok se smanjenje nepotrebnih informacija temelji na uklanjanju onih informacija iz slike koje ljudsko oko ne može ni zamijetiti, primjerice jako male razlike između nijansi boja [5].

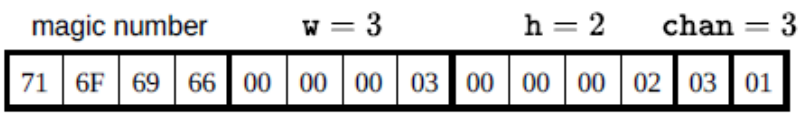
Postoje dvije vrste kompresije slike - kompresija bez gubitaka i kompresija s gubitcima. Kompresija bez gubitaka (eng. *lossless compression*) je oblik kompresije u kojoj se ne gube podaci ni kvaliteta slike. Prema tome, moguće je iz komprimirane slike dobiti sliku koja je u potpunosti identična originalnoj verziji. Međutim, ovaj oblik kompresije ne postiže visok omjer kompresije u usporedbi s kompresijom s gubitcima. Formati zapisa slike koji koriste navedenu kompresiju su primjerice PNG i GIF, a neki od algoritama s kojima se postiže kompresija bez gubitaka su Lempel-Ziv-Welch algoritam, Huffmanov kod te aritmetičko kodiranje. Algoritmi koji će se koristiti u ovom radu s Bzip2 i Brotli algoritmi koji u pozadini koriste više osnovnih algoritama, a bit će detaljnije objašnjeni u Poglavlju 2.4.

S druge strane, kompresija s gubitcima (eng. *lossy compression*) smanjuje veličinu datoteke na način da trajno uklanja manje važne podatke slike. Time je moguće značajno smanjiti veličinu slike, ali istovremeno ovaj oblik kompresije može dovesti do gubitka kvalitete slike. Primjer formata koji koristi ovu kompresiju je JPG, gdje je moguće smanjiti sliku i do 80% njene početne veličine, tako da se smanjuje broj piksela, njihova svjetlina, te gustoća boja [4]. Svjetlina piksela se modificira promjenom prostora boja iz RGB u CMYK (*Cyan, Magenta, Yellow, Key/Black*) prostor boja, a nakon toga primjenjuju se određene transformacije nad tako dobivenim komponentama piksela [6].

2.3 The Quite OK Image format

The Quite OK Image format, odnosno QOI format, je algoritam za kompresiju slike koji pripada skupini kompresije bez gubitaka. Namijenjen je kompresiji RGB ili RGBA slika te je jednostavan za implementaciju [7]. Navedeni algoritam kodira i dekodira slike u jednom prolasku kroz svaki piksel.

Slika koja je pohranjena u QOI formatu sadrži tri dijela - zaglavlje veličine 14 bajta, dio s kompresiranim podacima te oznaku kraja slike veličine osam bajta. U zaglavlju se nalaze sljedeći podaci: "čarobna" (eng. *magic*) riječ *qoif*, koja označava kako se radi o QOI formatu, širina slike u pikselima, visina slike u pikselima, broj kanala slike (RGB ili RGBA) te prostor boja slike. Prostor boja slike može poprimiti vrijednost 0 za sRGB, odnosno standardni RGB prostor boja, ili 1 za prostor boja u kojemu su sve komponente boje linearno reprezentirane (eng. *all channels linear color space*), međutim prostor boja je informativni podatak koji ne utječe na način kodiranja slike. Prva tri podatka zaglavlja zauzimaju po četiri bajta, dok broj kanala slike te prostor boja zauzimaju po jedan bajt zaglavlja. Primjer zaglavlja QOI datoteke prikazan je na slici 2.1.



Slika 2.1 Zaglavlje QOI datoteke, preuzeto iz [8]

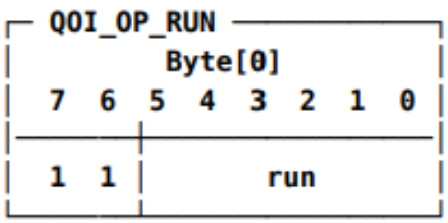
U zaglavlju su prikazane vrijednosti bajtova u slučaju jednostavne slike širine tri (oznaka w) i visine dva (oznaka h) piksela, gdje se svaki piksel sastoji od tri komponente (oznaka *chan*). "Čarobna" riječ (oznaka *magic number*) uvijek sadrži iste vrijednosti, kao što je već prethodno objašnjeno. Nakon određivanja i spremanja navedenih podataka zaglavlja, kreće se s kodiranjem slike.

Cijela slika kodira se redak po redak, s lijeva na desno, od vrha prema dnu, a svaki piksel se kodira na jedan od četiri moguća načina. Rezultirajuće vrijednosti su zapakirane u blokove (eng. *chunks*) koji započinju oznakama veličine dva ili osam bita. Oznake definiraju jednu od metoda kodiranja, a nakon njih slijedi određeni broj

podatkovnih bitova. Mogući načini kodiranja piksela bit će objašnjeni u sljedećim poglavljima.

2.3.1 1. slučaj - Serija istih piksela

Prva metoda kodiranja piksela primjenjuje se kada je trenutni piksel koji se kodira jednak prethodnom pikselu. Tada se povećava vrijednost varijable koja sadrži informaciju o duljini serije (eng. *run*) za jedan, a koja u početku kodiranja iznosi 0. U slučaju kada je trenutni piksel različit od prethodnog, vrijednost varijable koja pohranjuje duljinu serije sprema se u kodirane podatke, njena vrijednost postavlja se u 0, a trenutni piksel se kodira na neki od preostala tri načina, objašnjena u sljedećim poglavljima. Na slici 2.2 prikazan je blok bitova koji se zapisuje u kodirane podatke u 1. slučaju.



Slika 2.2 Blok bitova koji se zapisuje u 1. slučaju, preuzeto iz [9]

Oznaka u slučaju serije istih piksela jednaka je 11, a vrijednost duljine serije je broj veličine šest bita, stoga može poprimiti vrijednost između 1 i 62. Kada bi vrijednost varijable *run* bila 63 tada bi blok bitova imao vrijednost 111110, a kada bi ta vrijednost iznosila 64, blok bitova bi poprimio vrijednost 111111. Navedene vrijednosti blokova bitova zauzete su oznakom blokova bitova trećeg (Poglavlje 2.3.3), odnosno četvrtog (Poglavlje 2.3.4) slučaja te su stoga ilegalne.

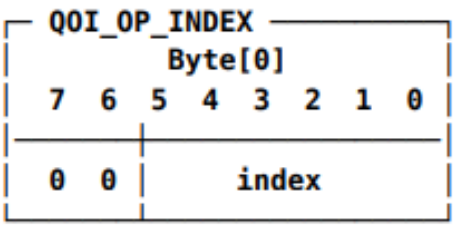
2.3.2 2. slučaj - Indeksiranje pomoću prethodno viđenih piksela

Tijekom kodiranja slike, u polju se pamti 64 prethodno viđenih piksela. Kada se pronađe piksel u navedenom polju isti trenutnom pikselu, indeks polja sprema se u kodirane podatke. Piksel se zapisuje u navedeno polje i u njemu pronalazi pomoću *hash* funkcije, čime se omogućuje da se za svaki piksel pristupa polju samo jednom. *Hash* vrijednost računa se pomoću sljedećeg izraza:

$$index = (r \times 3 + g \times 5 + b \times 7 + a \times 11) \ \% \ 64 \tag{2.2}$$

gdje r označava vrijednost crvene komponente piksela, g označava vrijednost zelene komponente piksela, b označava vrijednost plave komponente piksela, a a označava vrijednost *alpha* komponente piksela, odnosno njegovu prozirnost. Kako bi se omogućilo prikladno pohranjivanje piksela unutar polja veličine 64, odnosno kako bi se rezultat *hash* funkcije nalazio unutar raspona od 0 do 63, koristi se izračun ostataka pri dijeljenju navedenog izraza s brojem 64. Svaki piksel se nakon kodiranja, neovisno kojim od slučaja se on kodira, pohranjuje u navedeno polje.

Na slici 2.3 prikazan je blok bitova koji se zapisuju u kodirane podatke u ovom slučaju.



Slika 2.3 Blok bitova koji se zapisuju u 2. slučaju, preuzeto iz [9]

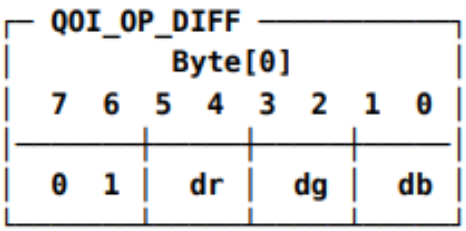
Oznaka bloka je 00, a kao što je i vidljivo iz slike, navedena vrijednost sadrži podatak od šest bita, stoga može poprimiti vrijednost iz raspona od 0 do 63. Dva

uzastopna bloka ne bi smjela imati dva ista bloka ovoga tipa, već bi se taj slučaj trebao kodirati pomoću serije istih piksela, kao što je navedeno u prethodnom slučaju (Poglavlje 2.3.1).

2.3.3 3. slučaj - Razlika u odnosu na prethodni piksel

U slučaju kada piksel koji se trenutno kodira nije previše udaljen od prethodnog piksela po pitanju boja, izračunava se razlika između svake komponente tih piksela. Ovaj slučaj ima dva podslučaja, kada su iznosi razlika po komponentama vrijednosti veličine do dva bita te kada su razlike po komponentama vrijednosti veličine do šest bita.

Ako je udaljenost po svakoj komponenti vrijednost između -2 i 1 (vrijednosti veličine dva bita), razlike između tih komponenti direktno se zapisuju u kodirane podatke. Ovaj slučaj podrazumijeva da su *alpha* komponente piksela iste, dok je blok bitova za ovaj slučaj prikazan na slici 2.4.

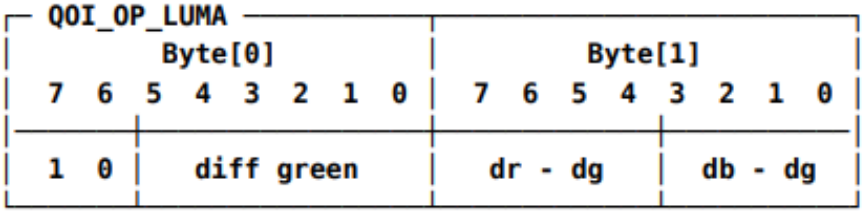


Slika 2.4 Blok bitova koji se zapisuju u slučaju kada je razlika boja između -2 i 1, preuzeto iz [9]

Oznaka takvog bloka bita je 01. Oznaka *dr* označava razliku između crvenih komponenti, *dg* označava razliku između zelenih komponenti, a *db* označava razliku plavih komponenti prethodnog i trenutnog piksela.

Ako je udaljenost po svakoj komponenti između -32 i 31, odnosno ako je veličina vrijednosti razlike komponenti do šest bita, izračunavaju se razlike za svaku komponentu, a u kodirane podatke zapisuje se razlika po zelenoj komponenti, a razlikama po crvenoj i plavoj komponenti oduzima se razlika po zelenoj komponenti. U ovom

slučaju se također podrazumijeva da su *alpha* komponente piksela jednake, a blok bitova koji se zapisuje u kodirane podatke u ovom slučaju prikazan je na slici 2.5.



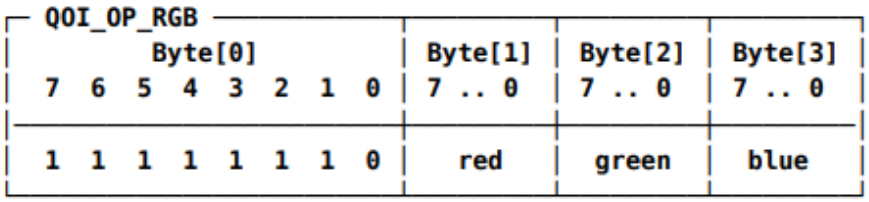
Slika 2.5 Blok bitova koji se zapisuju u slučaju kada je razlika boja između -32 i 31, preuzeto iz [9]

Oznaka za ovaj blok bitova je 10. Oznake *dr*, *dg* i *db* predstavljaju iste vrijednosti kao i u prethodnom slučaju.

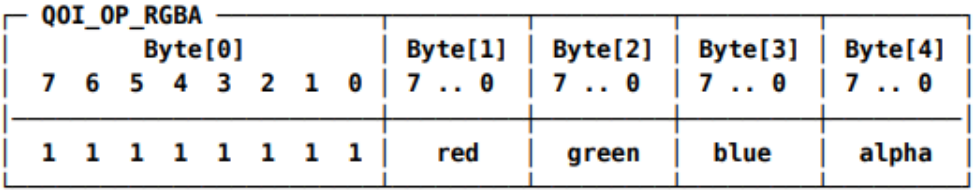
2.3.4 4. slučaj - Spremanje svih RGB ili RGBA vrijednosti piksela

U slučaju kada ne vrijedi niti jedan uvjet iz prethodno navedenih slučajeva, sve komponente piksela koji se trenutno kodira zapisuju se u kodirane podatke. U ovom slučaju također postoje dva podslučaja, ovisno o tome je li *alpha* vrijednost trenutnog piksela jednaka ili različita od *alpha* vrijednosti prethodnog piksela.

Na slici 2.6 prikazan je blok bitova koji se sprema u slučaju kada je *alpha* vrijednost trenutnog piksela jednaka *alpha* vrijednosti prethodnog piksela, dok je oznaka ovakvog bloka bita 11111110, a na slici 2.7 prikazan je blok bitova koji se sprema u slučaju kada se trenutni piksel razlikuje u svim komponentama prethodnog piksela, odnosno kada je i *alpha* vrijednost piksela različita. Oznaka bloka bitova koji se spremaju u kodirane podatke je, u ovom slučaju, 11111111.



Slika 2.6 Blok bitova koji se zapisuju u 4. slučaju kada je *alpha* vrijednost piksela jednaka *alpha* vrijednosti prethodnog piksela, preuzeto iz [9]



Slika 2.7 Blok bitova koji se zapisuju u 4. slučaju kada je *alpha* vrijednost piksela različita od *alpha* vrijednosti prethodnog piksela, preuzeto iz [9]

Posljednji dio slike kodirane u QOI formatu je oznaka za kraj datoteke, koja sadrži sedam 0x00 bajtova te jedan 0x01 bajt.

2.3.5 Dekodiranje slike

Dekodiranje, odnosno dekompresija slike provodi se na način da se prvo iz zaglavlja pročitaju podaci o visini i širini slike te koliko komponenti ima svaki piksel (RGB ili RGBA pikseli). Nakon toga, pomoću oznake bloka bitova određuje se koji način kodiranja je bio primijenjen na pojedinom pikselu.

Ako se radi o posljednjem slučaju kodiranja, odnosno ako su pohranjene sve RGB ili RGBA vrijednosti piksela, za svaku pojedinu komponentu uzima se sljedeći bajt iz kodiranih podataka. U protivnom, provjeravaju se prva dva bita trenutnog bajta, kako bi se odredilo o kojem od preostala tri slučaja kodiranja piksela se radi. Logičkom operacijom I s maskom 11000000, izdvajaju se samo prva dva bita bajta. U slučaju kada je piksel već zapamćen u polju (2. slučaj), potrebno mu je samo pristu-

piti na indeksu koji je zapisan u bloku bitova, a ako se radi o slučaju kada su zapisane razlike komponenti piksela, potrebno ih je iz zapisanih vrijednosti izračunati. Ako se radi o nizu uzastopno istih piksela, varijabla koja sadrži duljinu serije veća je od 0. U tom slučaju, smanjuje se navedena varijabla te se u dekodirane podatke pohranjuje isti piksel kao u prethodnom koraku.

Kada se odrede sve komponente piksela, on se pohranjuje u dekodirane podatke. Nakon toga, prelazi se na sljedeći bajt, sve dok ne preostanu posljednjih osam bajta podataka koji označavaju kraj QOI datoteke.

2.4 Postojeći algoritmi za kompresiju slike

Postoje mnogi algoritmi za kompresiju slike bez gubitaka, a većina najčešće danas korištenih implementira više osnovnih algoritama. Neki od najpoznatijih takvih algoritama su Deflate (temelji se na LZ77 i Huffmanovom kodu), gzip, koji se temelji na Deflate algoritmu, Brotli algoritam (temelji se na LZ77 algoritmu, Huffmanov kodu te modeliranju konteksta drugog reda (eng. *2nd-order context modelling*)), LZW, LZSS, LZMA, LZ4 algoritmi (varijacije LZ77 i LZ78 algoritama), Bzip2 (temelji se na Burrows–Wheeler algoritmu, RLE metodi, MTF tranformaciji i Huffmanovom kodu), te mnogi drugi algoritmi.

Kod usporedbe QOI algoritma s drugim, često korištenim algoritmima za kompresiju slike odabrani su Bzip2, Brotli i LZ4 algoritam zbog svoje popularnosti. Bzip i Brotli algoritmi u pozadini koriste različite osnovne metode, dok se LZ4 temelji na jednom osnovnom algoritmu.

2.4.1 Bzip2 algoritam

Bzip2 (BZ2) je besplatan program otvorenog koda za kompresiju datoteka bez gubitaka koji se temelji na Burrows–Wheeler algoritmu. Komprimira pojedinačne datoteke te zbog toga nije namijenjen za njihovo arhiviranje. Većinu datoteka komprimira učinkovitije od algoritama poput LZW-a i Deflate-a, gdje komprimira podatke u blokove veličine od 100kB do 900kB, ali je vremenski sporiji. Performanse

algoritma su asimetrične, pri čemu dekompresija podataka traje kraće od kompresije podataka, što je i vidljivo u rezultatima mjerenja trajanja kodiranja, odnosno dekodiranja slike, prikazanim u Poglavlju 4 [10].

Bzip2 algoritam koristi nekoliko slojeva kompresije koji uključuju *run-length encoding* (RLE), Burrows-Wheeler transformaciju (BWT), *move-to-front* (MTF) transformaciju i Huffmanov kod. Dekodiranje podataka, odnosno dekompresija provodi se primjenom istih algoritama kao i kod kompresije, ali suprotnim redoslijedom.

Kod kompresije podataka, nad inicijalnim podacima primjenjuje se RLE gdje se ponavljajući dijelovi datoteke zamjenjuju jednim podatkom, u kojem je sadržan element koji se ponavlja te broj njegovih ponavljanja. Nakon toga, primjenjuje se Burrows-Wheeler transformacija za pretvorbu često korištenih sekvenci podataka u jedan *string*. Nad tim podacima primjenjuje se MFT transformacija, ponovno RLE i onda Huffmanov kod. MFT transformacija i Huffmanov kod također se temelje na učestalosti pojavljivanja elemenata. Nakon toga, slijedi odabir između više Huffmanovih tablica, *Unary base-1* kodiranje te delta kodiranje duljine bitove Huffmanovog koda (eng. *Delta encoding of Huffman-code bit lengths*). Naposljetku, podaci se pohranjuju u polju koji sadrži korištene simbole.

2.4.2 Brotli algoritam

Brotli algoritam je algoritam otvorenog koda za kompresiju podataka bez gubitaka. Temelji se na kombinaciji LZ77 algoritma, Huffmanovog koda te modeliranja konteksta drugog reda. Pruža bolji omjer kompresije od gzip algoritma, ali je vremenski sporiji [11]. Kao i kod Bzip2 algoritma, performanse Brotli algoritma su također asimetrične, što se može vidjeti u prikazanim rezultatima mjerenja kompresije i dekompresije slike, u Poglavlju 4.

LZ77 algoritam postiže kompresiju podataka zamjenom ponavljajućih sekvenci s referencama na jednu kopiju tih podataka koja se nalazi u prethodnom dijelu nekomprimiranih podataka. Navedeni algoritam koristi klizeći prozor (eng. *sliding window*) za prethodno viđene elemente tijekom kompresije, a u Brotli algoritmu navedeni prozor je ograničen na 16MiB [12].

2.4.3 LZ4 algoritam

LZ4 algoritam je algoritam za kompresiju bez gubitaka koji pripada LZ77 skupini algoritama. Glavna karakteristika ovoga algoritma je velika brzina kompresije i dekompresije čime se smanjuje omjer kompresije, odnosno postiže lošije rezultate od, primjerice, LZO algoritma iz iste skupine [13].

LZ4 algoritam koristi rječnik u kojem se pohranjuju reference na prethodno viđene sekvence podataka koje se nalaze u prethodnom dijelu nekomprimiranih podataka, kako je već objašnjeno u Poglavlju 2.4.2. U slučaju kada je sekvenca već prethodno viđena, u kodirane podatke pohranjuje se informacija o duljini i poziciji sekvence u rječniku. U protivnom, neponavljajući podaci se kopiraju u komprimiranom obliku bez njihove promjene.

Poglavlje 3

Implementacija programskog rješenja

3.1 Implementacija QOI algoritma

Za implementaciju QOI algoritma korišten je programski jezik Python. Python je skriptni programski jezik visoke razine, široko korišten te otvorenog koda. Pruža mnoge gotove knjižnice koje će biti korištene u implementaciji navedenoga algoritma [14].

Oznake blokova bitova definirane su konstantama kao što je prikazano u isječku programskog koda na slici 3.1. Konstanta QOI_OP_INDEX koristi se u slučaju kada se pohranjuje indeks polja gdje se nalazi prethodno pohranjen piksel, QOI_OP_DIFF i QOI_OP_LUMA koriste se u definiranju bloka piksela kada se pohranjuje razlika komponenti piksela, QOI_OP_RUN se koristi u definiranju bloka bitova kod pohrane duljine serije istih piksela, a QOI_OP_RGB i QOI_OP_RGBA kada se direktno pohranjuju r, g i b ili r, g, b i a komponente piksela. Konstanta QOI_MASK_2 koristi se kod određivanja metode kojom je piksel kodiran, odnosno kod dekodiranja slike kada se logičkom operacijom I određuju prva dva bita bajta.

```
QOI_OP_INDEX = 0x00
QOI_OP_DIFF = 0x40
QOI_OP_LUMA = 0x80
QOI_OP_RUN = 0xc0
QOI_OP_RGB = 0xfe
QOI_OP_RGBA = 0xff
QOI_MASK_2 = 0xc0
```

Slika 3.1 Oznake blokova bitova

QOI magic konstanta definirana je pomoću funkcije *ord()* koja prima jedan znak i vraća njegovu Unicode vrijednost. U ovom slučaju, koristi se za pretvorbu slova 'q', 'o', 'i' i 'f' u brojeve, što omogućuje definiranje magične konstante posmicanjem svakog slova za određeni broj mjesta. Definiranje *QOI magic* konstante prikazano je u isječku programskog koda na slici 3.2.

```
QOI_MAGIC = ord('q') << 24 | ord('o') << 16 | ord('i') << 8 | ord('f')
```

Slika 3.2 *QOI magic* konstanta

Svaki piksel slike definiran je pomoću razreda *Pixel*. Kod instanciranja objekta navedenog razreda, komponente piksela, odnosno varijable, inicijaliziraju su na vrijednosti 0, za r, g i b komponente te na vrijednost 255 za a komponentu. Također, potrebno je implementirati funkciju *equals()* kako bi se kod određivanja jednakosti piksela uspoređivale jednakosti pojedinih komponenti, a ne njihove memorijske lokacije. Isječak programskog koda za razred *Pixel* prikazan je na slici 3.3.

```
class Pixel:
  def __init__(self, r=0, g=0, b=0, a=255):
    self.r = r
    self.g = g
    self.b = b
    self.a = a

  def __eq__(self, other):
      if not isinstance(other, Pixel):
          return NotImplemented

      return self.r == other.r and self.g == other.g and \
          self.b == other.b and self.a == other.a
```

Slika 3.3 Razred *Pixel*

Za upisivanje 32-bitnih vrijednosti u zaglavlje QOI datoteke koristi se funkcija *write_32_bits()* prikazana u isječku programskog koda na slici 3.4.

```
def write_32_bits(value, bytes_array):
    bytes_array.append((0xff000000 & value) >> 24)
    bytes_array.append((0x00ff0000 & value) >> 16)
    bytes_array.append((0x0000ff00 & value) >> 8)
    bytes_array.append((0x000000ff & value))
```

Slika 3.4 Funkcija *write_32_bits()*

Prilikom upisivanja vrijednosti na kraj polja, koristi se funkcija *append()* koja se poziva nad poljem. Kao argument, *append()* prima vrijednost koju je potrebno pohraniti u polje, a ta vrijednost se pohranjuje u postojeću listu. Važno je napomenuti kako se ne kreira nova lista već se dodavanje elementa izvodi nad postojećom listom.

Funkcija *write_32_bits()* koristi se kod kodiranja slike, za pohranu *QOI magic* konstante, visine slike te širine slike. S obzirom na to da su broj kanala slike te prostor boja 8-bitni podaci, moguće ih je dodati u kodirane podatke funkcijom *append()* bez korištenja prethodno opisane funkcije.

Kod dekodiranja slike, za dohvat podataka iz zaglavlja, koristi se funkcija *read_32_bits()*, prikazana u isječku programskog koda na slici 3.5. Navedena funkcija se koristi za dohvat *QOI magic* konstante, visine slike te širine slike.

```
def read_32_bits(bytes_array):
    b1, b2, b3, b4 = bytes_array
    return b1 << 24 | b2 << 16 | b3 << 8 | b4
```

Slika 3.5 Funkcija *read_32_bits()*

Za kodiranje, odnosno komprimiranje slike koristi se funkcija *encode()*, a za dekodiranje, odnosno dekomprimiranje slike, koristi se funkcija *decode()*. Funkcija *encode()* napisana je prema uzoru na pseudokod prikazan sljedećim isječkom koda koji prikazuje kako se kodira svaki pojedini piksel slike:

```
ako je trenutni piksel jednak prethodnom:
    povećaj varijablu run za 1
    ako je run == 62 ili ako nema više piksela u slici:
        primjeni 1. slučaj
inače:
    ako je run > 0:
        primjeni 1. slučaj
    izračunaj index_pos pomoću hash funkcije
    ako je piksel u polju prethodno viđenih piksela na indexu index_pos
    jednak trenutnom pikselu:
        primjeni 2. slučaj
    inače:
        ažuriraj polje prethodno viđenih piksela trenutnim pikselom
        ako su alpha vrijednosti trenutnog i prethodnog piksela jednake:
            izračunaj razlike crvenih komponenti (vr), zelenih komponenti
            (vg) i plavih komponenti (vb) piksela, vg_r kao vr - vg i
            vg_b kao vb - vg
            ako je vr > -3 i vr < 2 i vg > -3 i vg < 2 i vb > -3 i vb < 2:
                primjeni 3. slučaj, 1. podslučaj
            inače ako je vg_r > -9 i vg_r < 8 i vg > -33 i vg < 32 i
            vg_b > -9 i vg_b < 8:
                primjeni 3. slučaj, 2. podslučaj
            inače:
                primjeni 4. slučaj, 1. podslučaj
        inače:
            primjeni 4. slučaj, 2. podslučaj
```

Funkcija *encode()* prima varijablu *pixels* koja je *Image* objekt. Navedeni razred detaljnije će biti objašnjen u Poglavlju 3.2. Ova funkcija vraća polje bajtova koje definira QOI datoteku. Funkcija *decode()* prima polje koje vraća funkcija *encode()*, a vraća jednodimenzionalno polje *pixels* koje sadrži redom komponente r, g, b i a svakoga piksela, *tuple* kolekciju za širinu i visinu slike te varijablu *mode* koja određuje o kojem tipu slike se radi. Varijabla *mode* može poprimiti vrijednosti 'RGB' ili 'RGBA'.

U funkciji *encode()*, kod kodiranja slike, varijabla *run*, odnosno duljina serije istih piksela, inicijalizirana je na 0, a polje indeksa prethodnih piksela je definirano kao polje od 64 objekta razreda *Pixel*. Prethodni i trenutni piksel također su inicijalizirani kao objekti razreda *Pixel*, što znači da imaju početne vrijednosti komponenti 0 za r, g i b te 255 za komponentu a. Polje koje funkcija *encode()* vraća kao polje komprimirane slike, odnosno kao QOI datoteku, je polje bajtova te je iz tog razloga ono definirano pomoću funkcije *bytearray()*. S obzirom da se kodirani podaci dodaju u navedeno polje funkcijom *append()*, nije potrebno unaprijed definirati veličinu ovoga polja. Za pohranu objekta piksela u polje prethodnih piksela korištena je funkcija *copy()* iz knjižnice *copy*.

Korištenjem operatora jednakosti ne kopira se postojeći objekt u novi, već se stvara veza između tih objekata. Za novi objekt, stvara se pokazivač koji pokazuje na istu memorijsku lokaciju kao i pokazivač originalnog objekta. Kako bi se izbjeglo navedeno, odnosno kako se promjenom originalnog objekta ne bi mijenjao novi objekt, koriste se funkcije *copy()* i *deepcopy()*. Funkcija *copy()* izvodi *plitko* kopiranje (eng. *shallow copy*) nad objektom, što znači da stvara novu instancu objekta, ali ako se radi o objektu koji sadrži druge objekte, u novi objekt pohranit će reference na te unutarnje objekte. S druge strane, *deepcopy()* funkcija izvodi *duboko* kopiranje (eng. *deep copy*) gdje stvara novi složeni objekt te u njega rekurzivno kopira sve objekte koji se nalaze u originalnom objektu [15]. S obzirom da objekt razreda *Pixel* ne sadrži druge objekte, u ovom radu koristi se funkcija *copy()*. Primjer korištenja funkcije *copy()* u implementaciji QOI algoritma nalazi se u isječku programskog koda na slici 3.6.

```
index[index_pos] = cp.copy(px)
```

Slika 3.6 Funkcija *copy()*

U navedenom primjeru, funkcija *copy()* koristi se u pohrani trenutnog piksela u polje prethodno viđenih piksela, na indeksu *index_pos*. *index_pos* je izračunat pomoću *hash* funkcije objašnjene u Poglavlju 2.3.2.

U funkciji *decode()*, odnosno kod dekodiranja slike iz QOI datoteke, može se koristiti funkcija *pop()*. Primjer korištenja navedene funkcije prikazan je u isječku

programskog koda na slici 3.7.

```
b1 = bytes_array.pop(0)
```

Slika 3.7 Funkcija *pop()*

Funkcija *pop()* uklanja element iz polja na zadanom indeksu. Također, uklonjeni element vraća, stoga ga je moguće pohraniti u varijablu. U navedenom primjeru vrijednost koju funkcija *pop()* vraća pohranjuje se u varijablu *b1*, a zadani indeks je 0, odnosno funkcija uklanja i vraća element s početka polja. Funkcija *pop(k)* briše element s indeksa k te pomiče svih k elemenata koji se nalaze u listi nakon zadanog indeksa za jedno mjesto ulijevo. Prema tome vremenska složenost ove funkcije je *O(k)*, odnosno *O(n)* ako se funkcija poziva nad prvim elementom liste.

S obzirom da korištenje *pop()* funkcije uvelike usporava izvršavanje dekodiranja slike (rezultati će biti prikazani u Poglavlju 4), lista u kojoj su pohranjeni podaci kodirane slike, zamijenjena je strukturom *deque* iz knjižnice *collections* [16]. *Deque* (skraćeno od *"double-ended queue"*) je generalizacija stoga i reda te podržava operacije dodavanja i uklanjanja elemenata s obje strane *deque*-a, s približno *O(1)* učinkovitošću. Za dodavanje elementa s desne strane koristi se funkcija *append(element)*, za dodavanje s lijeve strane koristi se funkcija *appendleft(element)*, za uklanjanje elementa s kraja strukture koristi se funkcija *pop()*, a za uklanjanje elementa s početka strukture koristi se *popleft()*. Na slici 3.8 prikazan je isječak programskog koda gdje su navedene funkcije vezane uz strukturu *deque* korištene u ovom radu.

```
bytes_array = deque(bytes_array)
px.r = bytes_array.popleft()
```

Slika 3.8 Struktura *deque*

Funkcijom *deque()* pretvara se lista, poslana funkciji kao parametar, u strukturu *deque*. Funkcija *popleft()* uklanja prvi element iz strukture te ga vraća, u vremenu *O(1)*.

3.2 Upravljanje slikama

Za upravljanje slikama koristi se knjižnica *Pillow*, verzija knjižnice *PIL* (Python Imaging Library). Navedena knjižnica pruža podršku za razne formate slika, učinkovitu internu reprezentaciju slika te razne mogućnosti obrade slika [17]. U programskom koda ovoga radi koristi se *Image* modul iz *Pillow* knjižnice. Modul sadrži istoimeni razred *Image* za prikaz *PIL* slika. Također, pruža razne funkcije za manipulaciju slikama, primjerice funkcije za učitavanje slika iz datoteka te stvaranje novih slika [18].

Za otvaranje slike koristi se funkcija *open()*. Navedena funkcija prima kao argument putanju do slike, a vraća objekt razreda *Image* učitane slike. Na slici 3.9 prikazan je isječak programskog koda gdje se koristi navedena funkcija.

```
img = Image.open('./../images/bird.png')
```

Slika 3.9 Funkcija *open()*

S obzirom da funkcija *decode()* vraća jednodimenzionalno polje komponenti svih piksela, potrebno ih je presložiti u pravilan oblik kako bi se dekodirana slika mogla spremiti. Navedeno se postiže funkcijom *create_image_from_pixels()* koja je prikazana u isječku programskog koda na slici 3.10.

Prikazana funkcija prima varijablu *decoded* u kojoj su pohranjene komponente svih piksela slike, *size* koja sadrži širinu i visinu slike te *mode* koja definira radi li se o RGB ili RGBA slici. Funkcijom *new()* stvara se novi objekt razreda *Image*, koja kao argumente prima varijable *mode* i *size*. Nakon toga provjerava se radi li se o RGB ili RGBA slici, odnosno sadrži li svaki piksel slike tri ili četiri komponente. S obzirom na vrstu slike, jednim od dva definirana načina prolazi se *for* petljom kroz polje komponenti piksela, gdje se definira svaki piksel koji se pohranjuje u objekt *Image*, funkcijom *putpixel()*. Funkcija *putpixel()* poziva se nad objektom slike, a prosljeđuju joj se koordinate trenutnog piksela te vrijednosti komponenti piksela. Nakon što se cijela slika definira, *create_image_from_pixels()* funkcija vraća objekt *Image*.

```
def create_image_from_pixels(decoded, size, mode):
    img_qoi = Image.new(mode, size)

    if mode == 'RGBA':
        for i in range(len(decoded) // 4):
            x = i % size[0]
            y = i // size[0]
            r, g, b, a = decoded[i*4:(i+1)*4]
            img_qoi.putpixel((x, y), (r, g, b, a))
    else:
        for i in range(len(decoded) // 3):
            x = i % size[0]
            y = i // size[0]
            r, g, b = decoded[i*3:(i+1)*3]
            img_qoi.putpixel((x, y), (r, g, b))

    return img_qoi
```

Slika 3.10 Funkcija *create_ image_ from_ pixels()*

Funkcija *save()* koristi se kod pohranjivanja slike na računalo. Kao argument prima putanju na računalnu gdje se slika želi spremiti. Navedena funkcija prikazana je u isječku programskog koda na slici 3.11.

```
img_qoi.save('bird_result.png')
```

Slika 3.11 Funkcija *save()*

3.2.1 Pohrana QOI datoteke

Za pohranu QOI datoteke korišten je modul *os.path* kojemu je, u programskom kodu, dodijeljen alias path [19]. Isječak programskog koda za pohranu QOI datoteke prikazan je na slici 3.12.

```
with open(path.join(path.dirname(__file__), 'bird.qoi'), 'wb') as qoi:
    qoi.write(encoded)
```

Slika 3.12 Pohrana QOI datoteke

Prikazani programski kod otvara datoteku imena *bird.qoi* u načinu pisanja binarnih podataka ('wb'), koristeći putanju koja se formira spajanjem direktorija u kojem se nalazi skripta ovoga programskog koda s definiranim imenom datoteke. Otvaranje datoteke vrši se korištenjem funkcije *open()*, definiranje putanje datoteke izvršava se korištenjem funkcije *join()* iz modula *os.path*, a funkcijom *dirname()*, također iz modula *os.path*, pristupa se putanji na računalu gdje se nalazi skripta. Nakon toga, koristi se objekt *qoi*, koji predstavlja otvorenu datoteku, kako bi se u nju zapisali podaci zapisani u varijabli *encoded*. Podaci se zapisuju pomoću funkcije *write()*, koja se poziva nad objektom *qoi*, a nakon završetka ove operacije, datoteka se automatski zatvara.

3.3 Mjerenje i izračun trajanja algoritma

Za mjerenje trajanja algoritma, odnosno za mjerenje trajanja funkcija kodiranja i dekodiranja slike, u implementaciji QOI algoritma, koristi se knjižnica *time*. Ona pruža različite funkcije vezane uz vrijeme [20]. U ovom radu koristi se funkcija *time()*, iz istoimene knjižnice, koja vraća vrijeme proteklo od 1. siječnja 1970. godine (*UNIX epoch*). Navedeno vrijeme služi kao referentna točka od koje brojač mjeri vrijeme. Na slici 3.13 prikazan je isječak programskog koda gdje se koristi navedena funkcija.

```
begin = time.time()
encoded = encode(pixels)
end = time.time()
```

Slika 3.13 Funkcija *time()*

U prikazanom primjeru određuje se vrijeme neposredno prije početka (varijabla *begin*) i neposredno nakon završetka (varijabla *end*) kodiranja slike QOI algoritmom.

Kako bi se odredilo vrijeme trajanja funkcije, ove dvije vrijednosti se oduzimaju. Na isti način određuje se vrijeme trajanja dekodiranja slike.

3.4 Određivanje omjera kompresije

Za određivanje omjera kompresije slike koristi se funkcija *getsizeof()* iz knjižnice *sys*. Knjižnica *sys* pruža varijable i funkcije koje omogućuju interakciju s interpreterom i sustavom [21]. Funkcija *getsizeof()* kao argument prima objekt bilo kojeg tipa podataka, a vraća njegovu veličinu u bajtima. Na slici 3.14 prikazan je isječak programskog koda u kojem se koristi navedena funkcija.

```
img = Image.open('./../images/bird.png')
img_bytearray = bytearray(np.asarray(img))
encoded = encode(img)

compression_rate = (1 - sys.getsizeof(encoded) / sys.getsizeof(img_bytearray)) * 100
```

Slika 3.14 Funkcija *getsizeof()*

Pomoću funkcije *getsizeof()*, određuje se veličina originalne i kodirane slike te se, pomoću tih vrijednosti, računa omjer kompresije te slike. S obzirom da objekt *img* i *encoded* nisu istog tipa podatka, potrebno je *img* pretvoriti u polje pomoću funkcije *asarray()* iz knjižnice *Numpy*, a nakon toga u objekt *bytearray()*, kako bi se pravilno odredio omjer kompresije.

3.5 Pohrana rezultata

Kako bi se sve izračunate vrijednosti spremale u tekstualnu datoteku, koristi se objekt *stdout* iz knjižnice *sys*. Navedeni objekt označava gdje će funkcija *print()* ispisivati vrijednosti, a ako objekt *stdout* nije promijenjen, vrijednosti će se ispisivati u konzoli. Na slici 3.15 prikazan je isječak programskog koda u kojem je korišten navedeni objekt.

```
log_file = open(str(work_dir + 'qoi_results.txt'), 'w')
sys.stdout = log_file
```

Slika 3.15 Objekt *stdout*

Funkcija *str()* stvara *string* iz primljenih argumenata, a u ovom primjeru koristi se kako bi se definirala putanja gdje se otvara željena datoteka. U varijabli *work_dir* pohranjena je putanja do direktorija te se na nju nadodaje ime datoteke, odnosno *qoi_results.txt*. Pomoću funkcije *open()* otvara se datoteka na definiranoj putanji u koju je moguće pisati ('w'). Zatim se ta datoteka (dodijeljena varijabli *log_file*) postavlja kao novi standardni izlaz, što znači da će sve izlazne poruke koje se inače prikazuju na konzoli biti preusmjerene i zapisane u definiranu datoteku.

Na slici 3.16 prikazan je primjer jedne takve izlazne datoteke.

```
QOI encode
Vrijeme trajanje kodiranja slike: 5.642547845840454

Velicina originalne slike (asarray): 994953
Velicina kompresirane slike: 551568
Postotak kompresije: 44

QOI decode
Vrijeme trajanja dekodiranja slike: 6.628206491470337

Velicina rezultantne slike: 994953
```

Slika 3.16 Primjer izlazne datoteke

U nju se zapisuje vrijeme trajanje kodiranja i dekodiranja slike, veličina originalne i kompresirane slike te postotak kompresije. Ovakva izlazna datoteka bit će korištena kod daljnje analize rezultata.

3.6 Dokaz ispravnosti algoritma

Kao što je već navedeno u Poglavlju 2, QOI algoritam je algoritam za kompresiju slike bez gubitaka. Prema tome, ako se na slici primjeni QOI algoritam za kodiranje, a nakon toga za dekodiranje slike, rezultantna slika mora biti isti kao i originalna, odnosno komponente svih piksela rezultantne slike moraju biti jednake komponentama svih piksela originalne slike. Na slici 3.17 prikazan je isječak programskog koda u kojem se provjerava jesu li originalna i rezultantna slika iste.

```
img = Image.open('./../images/bird.png')

encoded = encode(img)
decoded = decode(encoded)
pixels, size, mode = decoded

img_qoi = create_image_from_pixels(pixels, size, mode)

print(np.array_equal(img, img_qoi))
```

Slika 3.17 Dokaz ispravnosti algoritma

Nakon kodiranja i dekodiranja slike, funkcijom *create_image_from_pixels()*, iz piksela se kreira objekt slike, kao što je prethodno objašnjeno. Jednakost originalne i rezultantne slike provjerava se funkcijom *array_equal()*, iz knjižnice *Numpy*, koja kao argumente prima dva polja, odnosno u ovom primjeru dvije slike. Pokretanjem skripte nad bilo kojom slikom, ispisuje se vrijednost *True* čime se dokazuje kako je QOI algoritam, algoritam za kompresiju slike bez gubitaka.

3.7 Knjižnica *imagecodecs*

Imagecodecs je Python knjižnica koja pruža funkcije za transformaciju, kompresiju i dekompresiju slike. Implementira navedene funkcije za razne algoritme, primjerice za Zlib, GZIP, ZStandard (ZSTD), Blosc, Brotli, Snappy, LZMA, BZ2, LZ4 i

za mnoge druge [22]. U ovome radu koristit će se algoritmi, odnosno implementacije funkcija Brotli, BZ2 i LZ4 algoritama.

3.7.1 Implementacija BZ2 algoritma

Knjižica *imagecodecs* pruža razne funkcije u implementaciji BZ2 algoritma, a u ovom radu koristit će se *bz2_ encode()* za kompresiju slike te *bz2_ decode()* za dekompresiju slike. Na slici 3.18 prikazan je isječak programskog koda u kojem se kodira slika algoritmom BZ2.

```
encoded = bz2_encode(np.asarray(img))
```

Slika 3.18 Funkcija *bz2_ encode()*

Slika je učitana funkcijom *open()* iz knjižnice *Image*, koja je objašnjena u Poglavlju 3.2. Nakon toga, pretvorena je u polje piksela funkcijom *asarray()* iz knjižnice *Numpy* kako bi bila u ispravnom obliku za slanje funkciji *bz2_ encode()*. Navedena funkcija vraća objekt *bytes*, komprimirane slike, kojeg je moguće pohraniti u memoriju.

Na slici 3.19 prikazan je isječak programskog koda koji prikazuje korištenje funkcije *bz2_ decode()*, odnosno koji dekodira komprimiranu sliku.

```
decoded = bz2_decode(encoded)
```

Slika 3.19 Funkcija *bz2_ decode()*

Navedena funkcija prima *encoded* objekt vraćen funkcijom *bz2_ encode()*, a vraća objekt *bytes* dekodirane slike.

Vrijeme izvršavanja objašnjenih funkcija izmjereno je pomoću knjižnice *time*, omjer kompresije izračunat je pomoću funkcije *getsizeof()*, a sve je zapisano u datoteku pomoću knjižnice *sys*, kao što je objašnjeno kroz Poglavlje 3.

3.7.2 Implementacija Brotli algoritma

Knjižica *imagecodecs* pruža iste funkcije u implementaciji Brotli algoritma, kao i u implementaciji BZ2 algoritma, a u ovom radu koristit će se *brotli_ encode()* za kompresiju slike te *brotli_ decode()* za dekompresiju slike. Na slici 3.20 prikazan je isječak programskog koda u kojem se kodira slika algoritmom Brotli.

```
encoded = brotli_encode(np.asarray(img))
```

Slika 3.20 Funkcija *brotli_ encode()*

Slika je učitana funkcijom *open()* iz knjižnice *Image*, a nakon toga pretvorena je u polje piksela funkcijom *asarray()* iz knjižnice *Numpy*, na isti način kao i za BZ2 algoritam, te se u tom obliku šalje funkciji *brotli_ encode()*. Navedena funkcija također vraća objekt *bytes* komprimirane slike.

Na slici 3.21 prikazan je isječak programskog koda koji prikazuje korištenje funkcije *brotli_ decode()* za dekodiranje komprimirane slike.

```
decoded = brotli_decode(encoded)
```

Slika 3.21 Funkcija *brotli_ decode()*

Navedena funkcija prima *encoded* objekt vraćen funkcijom *brotli_ encode()*, a vraća objekt *bytes* dekodirane slike.

Mjerenje vremena izvršavanja navedenih funkcija, računanje omjera kompresije slike te zapis u datoteku provedeno je na isti način kao i za algoritam BZ2.

3.7.3 Implementacija LZ4 algoritma

Posljednji algoritam implementiran pomoću *imagecodecs* knjižnice je LZ4 algoritam. Navedena knjižnica pruža funkcije *lz4_ encode()* za kompresiju slike te *lz4_ decode()* za dekompresiju slike. Na slici 3.22 prikazan je isječak programskog koda u kojem se koristi funkcija *lz4_ encode()* za kodiranje slike pomoću LZ4 algoritma.

```
encoded = lz4_encode(np.asarray(img))
```

Slika 3.22 Funkcija *lz4_ encode()*

Slika je učitana na isti način kao i kod prethodno objašnjenih algoritama, funkcijom *open()* iz knjižnice *Image*, te je pretvorena u polje piksela funkcijom *asarray()* iz knjižnice *Numpy*. Funkcija *lz4_ encode()* prima tako definirano polje kao argument, a vraća objekt *bytes* komprimirane slike.

Sliku je moguće dekodirati funkcijom *lz4_ decode()* prikazanom u isječku programskog koda na slici 3.23.

```
decoded = lz4_decode(encoded)
```

Slika 3.23 Funkcija *lz4_ decode()*

Prikazana funkcija prima *encoded* objekt vraćen funkcijom *lz4_ encode()*, a vraća objekt *bytes* dekodirane slike, kao i kod prethodno objašnjenih implementacija algoritama.

Mjerenje vremena izvršavanja funkcija kodiranja i dekodiranja slike, računanje omjera kompresije slike te zapis u datoteku provedeno je na isti način kao i za BZ2 i Brotli algoritme.

Poglavlje 4

Rezultati

Mjerenja na čijim rezultatima je provedena analiza u ovom poglavlju, provedena su na računalu s AMD Ryzen 7 4700U 2.00GHz procesorom i s 8GB RAM memorije. Svako mjerenje utrošenog vremena je provedeno pet puta, nakon čega je izračunata aritmetička sredina i standardna devijacija rezultata.

4.1 Ovisnost vremena o veličini slike

4.1.1 Slika korištena u analizi algoritma

U analizi ovisnosti vremena izvršavanja programskog koda QOI algoritma o veličini originalne, odnosno komprimirane slike koristi se slika 4.1.

Slika 4.1 Testna slika, preuzeta s [23]

Iz prikazane slike izrezani su isječci manjih veličina te su oni korišteni kod analize rezultata. Ovakva slika korištena je u ovom dijelu analize algoritma s obzirom da se sličan uzorak proteže kroz cijelu sliku, odnosno nema velikih oscilacija u vrijednostima piksela. Prema tome, položaj isječka u originalnoj slici ne utječe na rezultate mjerenja.

4.1.2 Kodiranje slike

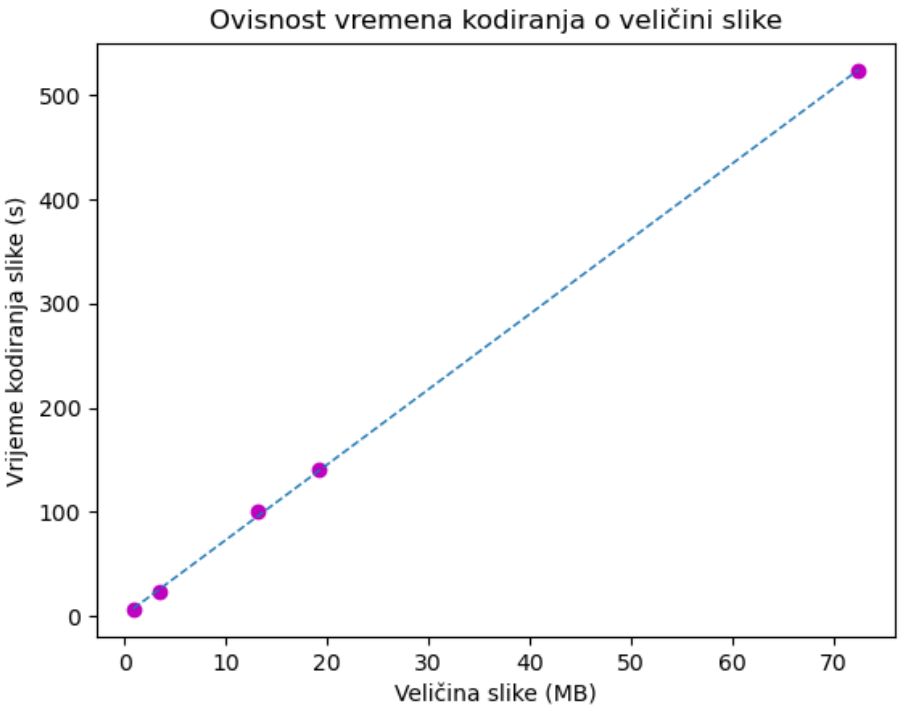
Za analizu ovisnosti vremena kodiranja slike QOI algoritmom o veličini slike korišteno je pet isječaka iz originalne slike. U tablici 4.1 prikazani su rezultati mjerenja. U prvom stupcu prikazana je veličina originalne slike, odnosno isječaka iz originalne slike, u bajtima, u drugom stupcu prikazano je prosječno vrijeme trajanja kodiranja slike u sekundama, nakon pet iteracija algoritma, a u trećem stupcu prikazana je standardna devijacija vremena u sekundama.

Tablica 4.1 Ovisnost vremena o veličini slike za operaciju kodiranja

Veličina slike (B)	Vrijeme kodiranja (s)	St. dev. vremena (s)
828357	5.7789	0.0439
3450789	23.6196	0.5633
13204869	100.2277	2.3611
19209717	140.1149	1.3673
72480825	524.1282	3.5376

Iz prikazanih rezultata vidljivo je kako povećanjem veličine slike približno četiri puta (s vrijednosti 828357B na vrijednost 3450789B), vrijeme kodiranja slike se također povećava približno četiri puta (s vrijednosti 5.7789s, uz standardnu devijaciju od 0.0439s, na vrijednost 23.6196s, uz standardnu devijaciju od 0.5633s). Prema tome, može se zaključiti kako vrijeme izvršavanja kodiranja slike linearno ovisi o veličini slike.

Isti rezultati su vidljivi i iz grafičkog prikaza ovisnosti vremena kodiranja slike o veličini slike prikazanog na slici 4.2, gdje je na osi apcisa prikazana veličina slike u megabajtima, a na osi ordinata je prikazano vrijeme potrebno za izvršavanje kodiranja slike u sekundama.



Slika 4.2 Grafički prikaz ovisnosti vremena kodiranja o veličini slike

4.1.3 Dekodiranje slike korištenjem funkcije *pop()*

Za analizu ovisnosti vremena dekodiranja komprimirane slike QOI algoritmom, korištenjem funkcije *pop()* o veličini komprimirane slike korišteno je četiri isječaka iz originalne slike. S obzirom da vrijeme dekodiranja direktno ovisi o veličini komprimirane slike, u ovom dijelu analize uspoređuju se te vrijednosti, umjesto vrijednosti veličina originalnih slike. U tablici 4.2 prikazani su rezultati mjerenja. U prvom stupcu prikazana je veličina komprimirane slike u bajtima, u drugom stupcu prikazano je prosječno vrijeme trajanja dekodiranja slike u sekundama, nakon pet iteracija algoritma, a u trećem stupcu prikazana je standardna devijacija vremena u sekundama.

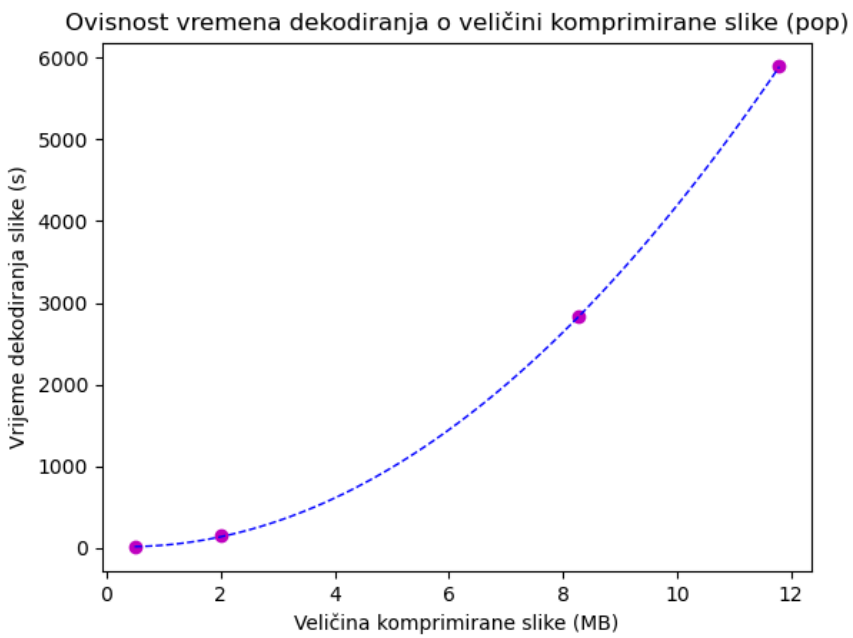
Iz prikazanih rezultata vidljivo je kako je složenost izvršavanja dekodiranja slike veća od kodiranja slike. Kao što je već objašnjeno u Poglavlju 3, funkcija *pop()* ima

Tablica 4.2 Ovisnost vremena o veličini komprimirane slike za operaciju dekodiranja s funkcijom *pop()*

Veličina komprimirane slike (B)	Vrijeme dekodiranja (s)	St. dev. vremena (s)
490284	5.4759	0.0919
2014978	151.7369	4.1033
8281302	2829.1469	7.8753
11791145	5893.9112	9.5726

složenost $O(n)$ iz čega slijedi da je složenost dekodiranja slike približno $O(n^2)$.

Isti rezultati su vidljivi i iz grafičkog prikaza ovisnosti vremena dekodiranja slike o veličini komprimirane slike prikazanog na slici 4.3, gdje je na osi apcisa prikazana veličina slike u megabajtima, a na osi ordinata vrijeme potrebno za izvršavanje dekodiranja slike u sekundama.



Slika 4.3 Grafički prikaz ovisnosti vremena dekodiranja s funkcijom *pop()* o veličini slike

4.1.4 Dekodiranje slike korištenjem strukture *deque*

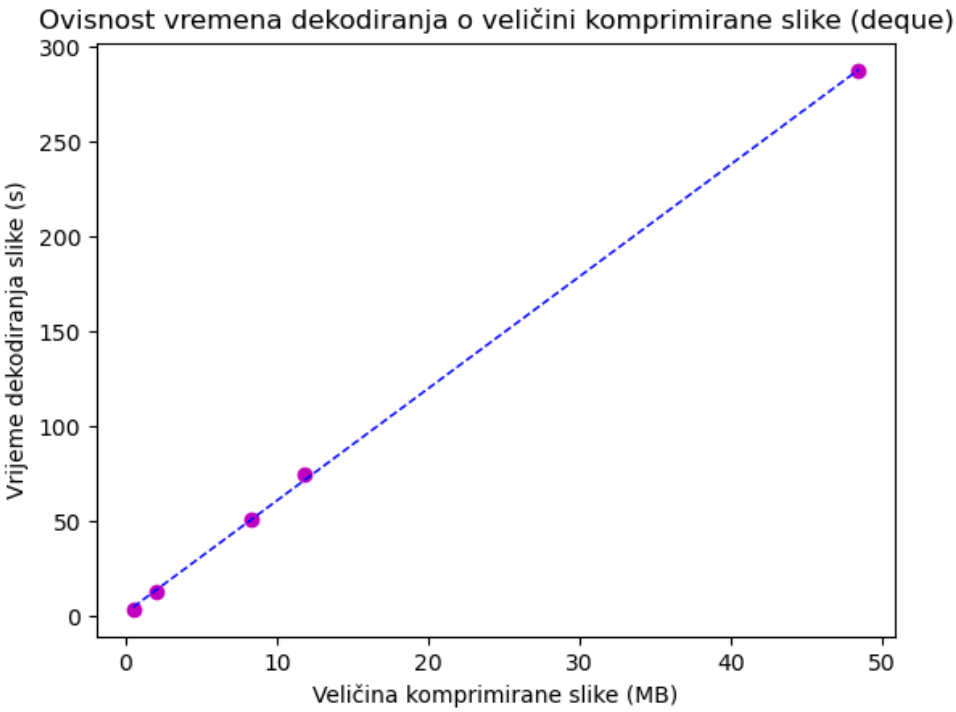
Za analizu ovisnosti vremena dekodiranja komprimirane slike QOI algoritmom, korištenjem strukture *deque* o veličini komprimirane slike korišteno je pet isječaka iz originalne slike. Kao i u prethodnoj analizi, korištene su vrijednosti veličina komprimiranih isječaka slike. U tablici 4.3 prikazani su rezultati mjerenja. U prvom stupcu prikazana je veličina komprimirane slike u bajtima, u drugom stupcu prikazano je prosječno vrijeme trajanja dekodiranja slike u sekundama, nakon pet iteracija algoritma, a u trećem stupcu prikazana je standardna devijacija vremena u sekundama.

Tablica 4.3 Ovisnost vremena o veličini komprimirane slike za operaciju dekodiranja sa strukturom *deque*

Veličina komprimirane slike (B)	Vrijeme dekodiranja (s)	St. dev. vremena (s)
490284	3.1154	0.0730
2014978	12.6484	0.3047
8281302	50.8194	0.6638
11791145	74.4707	0.9387
48460279	287.1971	0.1748

Iz prikazanih rezultata vidljivo je kako povećanjem veličine komprimirane slike četiri puta (s vrijednosti 490284B na vrijednost 2014978B), vrijeme dekodiranja slike također se povećava približno četiri puta (s vrijednosti 3.1154s, uz standardnu devijaciju od 0.0730s, na vrijednost 12.6484s, uz standardnu devijaciju od 0.3047s). Prema tome, vrijeme izvršavanja dekodiranja slike sa strukturom *deque* linearno ovisi o veličini komprimirane slike.

Isto je vidljivo i iz grafičkog prikaza ovisnosti vremena o veličini komprimirane slike prikazanog na slici 4.4, gdje je na osi apcisa prikazana veličina komprimirane slike u megabajtima, a na osi ordinata je prikazano vrijeme u sekundama potrebno za izvršavanje dekodiranja komprimirane slike.



Slika 4.4 Grafički prikaz ovisnosti vremena dekodiranja sa strukturom *deque* o veličini komprimirane slike

S obzirom na prikazane rezultate, u daljnjim mjerenjima koristit će se implementacija QOI algoritma gdje se dekodiranje slike provodi strukturom *deque*.

4.2 Usporedba QOI algoritma na različitim formatima zapisa slike

Kod implementacije usporedbe rezultata na različitim formatima zapisa slike korištena je testna slika prikazana na slici 4.5, preuzeta s [24]. Analiza je provedena na PNG, JPG i TIF formatima zapisa slike.

Slika 4.5 Testna slika, preuzeto s [24]

Veličina originalne slike, veličina komprimirane slike te omjer kompresije za svaki od formata prikazani su u tablici 4.4.

Tablica 4.4 Veličina originalne i komprimirane slike te omjer kompresije

Format	Veličina originalne slike (B)	Veličina komprimirane slike (B)	Omjer kompresije (%)
JPG	12000057	2868978	76.09
PNG	16000057	2868978	82.07
TIF	16000057	2868978	82.07

Slike u PNG i TIF formatima imaju jednaku veličinu, odnosno kada se slike pretvore u *bytearray()* objekti imaju isti broj bajta. Slika u JPG formatu nema *alpha* komponente, odnosno svaki piksel ima tri, a ne četiri komponente, pa prema tome ima i manju veličinu u odnosu na druge formate zapisa slike. Iz rezultata je vidljivo kako PNG i TIF slike imaju isti omjer kompresije, koji za testnu sliku iznosi 82.07%. Navedeno proizlazi iz činjenice da QOI algoritam tijekom kodiranja slike uzima u obzir samo komponente piksela, što znači da omjer kompresije ne ovisi o

formatu slike.

Izmjerena vremena u sekundama kodiranja slike sa standardnim devijacijama u sekundama prikazana su u tablici 4.5, za svaki analizirani format zapisa. Izmjerena vremena u sekundama dekodiranja slike sa standardnim devijacijama u sekundama prikazana su u tablici 4.6, za svaki analizirani format zapisa.

Tablica 4.5 Vrijeme kodiranja slika različitih formata

Format	Vrijeme kodiranja (s)	St. dev. vremena kodiranja (s)
JPG	57.1935	5.4895
PNG	59.3225	2.8519
TIF	64.1046	3.9129

Tablica 4.6 Vrijeme dekodiranja slika različitih formata

Format	Vrijeme dekodiranja (s)	St. dev. vremena dekodiranja (s)
JPG	45.4884	1.6388
PNG	52.7665	4.8061
TIF	48.2612	2.9805

Iz prikazanih rezultata vidljivo je kako se vrijeme kodiranja, odnosno dekodiranja slike ne razlikuje značajno o vrsti zapisa slike. Prema tome, vrijeme izvršavanja QOI algoritma ne ovisi o vrsti zapisa slike. Navedeno također proizlazi iz činjenice da QOI algoritam tijekom kodiranja i dekodiranja slike uzima u obzir samo komponente piksela, a ne vrstu formata zapisa slike.

4.3 Usporedba QOI algoritma s drugim algoritmima za kompresiju slike

4.3.1 Testne slike

Na slikama 4.6 i 4.7 prikazane su testne slike korištene u ovom dijelu analize algoritma.

(a) dog.tiff (b) pug.jpg i flower.jpg

Slika 4.6 Testne slike

Na slici 4.6a nalazi se slika iz osobne privatne kolekcije koja je u TIF formatu zapisa, a na slici 4.6b prikazane su slike preuzete sa [23], a koje se nalaze u JPG formatu zapisa.

(a) bird.png

(b) alfa.jpg

Slika 4.7 Testne slike

Na slici 4.7a nalazi se slika u PNG, a na slici 4.7b nalazi se slika u JPG formatu zapisa slike. Prikazane slike su iz osobne privatne kolekcije.

U tablici 4.7 prikazani su osnovni podaci o testnim slikama. U prvom stupcu nalazi se naslov slike, u drugom rezolucija, a u trećem stupcu veličina slike u bajtima.

Tablica 4.7 Podaci o testnim slikama

Slika	Rezolucija	Veličina (B)
dog.tiff	3517 x 5276	74222825
pug.jpg	4460 x 2973	39778797
flower.jpg	4898 x 3456	50782521
bird.png	3840 x 2160	33177657
alfa.jpg	4000 x 1800	21600057

Veličina svake slike, izražene u bajtima, dobivena je pomoću funkcije *getsizeof()* objašnjene u poglavlju 3.4, a iznosi približno umnošku broja piksela i broja komponenti pojedinog piksela. Izraz za izračun veličine slike detaljnije je objašnjen u Poglavlju 2.1.

4.4 Usporedba QOI algoritma s BZ2 algoritmom

U tablici 4.8 prikazani su omjeri kompresije, za svaku sliku prikazanu u prethodnom poglavlju, nakon kodiranja QOI i BZ2 algoritmima. Omjeri kompresije prikazani su u postotcima, a u svakoj od iteracija algoritama rezultati su jednaki.

Tablica 4.8 Omjer kompresije slika za QOI i BZ2 algoritme

Slika	Omjer kompresije za QOI algoritam (%)	Omjer kompresije za BZ2 algoritam (%)
dog.tiff	54.14	57.41
pug.jpg	66.65	75.22
flower.jpg	32.98	48.40
bird.png	80.28	86.89
alfa.jpg	45.41	59.21

Iz rezultata je vidljivo kako je BZ2 algoritam bolji po pitanju omjera kompresije slike od QOI algoritma za oko 3 do 15%, ovisno o slici. Najveću razliku može se

vidjeti kod slike *flower.jpg* gdje je BZ2 algoritam komprimirao sliku za 15.42% više od QOI algoritma.

U tablici 4.9 prikazani su rezultati mjerenja vremena izvršavanja kodiranja slika. U prvom stupcu nalazi se naziv slike, u drugom i trećem stupcu vrijeme kodiranja i standardna devijacija vremena, u sekundama, za QOI algoritam, a u četvrtom i petom stupcu vrijeme kodiranja slike i standardna devijacija za BZ2 algoritam.

Tablica 4.9 Vrijeme kodiranja slika QOI i BZ2 algoritmima

Slika	Vrijeme kodiranja za QOI algoritam (s)	St.dev. (s)	Vrijeme kodiranja za BZ2 algoritam (s)	St. dev. (s)
dog.tiff	146.4381	1.7299	6.5607	1.1958
pug.jpg	71.2044	0.5061	10.0339	1.2722
flower.jpg	120.5402	1.9860	5.90979	0.8175
bird.png	43.8049	0.1930	6.9538	0.8338
alfa.jpg	45.6521	0.4147	2.4939	0.3732

Iz prikazanih rezultata vidljivo je kako je BZ2 algoritam bolji od QOI algoritma po pitanju vremena izvršavanja kodiranja slika čak i do 22 puta, točnije za sliku *dog.tiff* BZ2 algoritam je brži ≈ 22.32 puta.

U tablici 4.10 prikazana je usporedba mjerenja vremena izvršavanja dekodiranja slika. U prvom stupcu nalazi se naziv slike, u drugom i trećem stupcu vrijeme dekodiranja i standardna devijacija vremena, za QOI algoritam, a u četvrtom i petom stupcu vrijeme dekodiranja slike i standardna devijacija za BZ2 algoritam. Navedena mjerenja izražena su u sekundama.

Tablica 4.10 Vrijeme dekodiranja slika QOI i BZ2 algoritmima

Slika	Vrijeme dekodiranja za QOI algoritam (s)	St.dev. (s)	Vrijeme dekodiranja za BZ2 algoritam (s)	St. dev. (s)
dog.tiff	83.8130	0.9174	3.0058	0.1729
pug.jpg	51.5446	0.1336	1.2045	0.0496
flower.jpg	69.8353	1.7724	2.2042	0.2345
bird.png	30.0133	0.0919	0.7877	0.1441
alfa.jpg	32.2059	0.3806	1.2812	0.1625

Iz rezultata prikazanih u tablici 4.10 može se zaključiti kako je BZ2 algoritam brži i kod izvršavanja dekompresije slike. BZ2 algoritam je brži od QOI algoritma kod dekodiranja slika od ≈ 25.14 puta (za sliku *alfa.jpg*) pa do ≈ 42.79 puta (za sliku *pug.jpg*).

4.5 Usporedba QOI algoritma s Brotli algoritmom

U tablici 4.11 prikazani su omjeri kompresije, za svaku sliku prikazanu u Poglavlju 4.3.1, nakon kodiranja QOI i Brotli algoritmima. Kao i kod prikaza rezultata za BZ2 algoritam, omjeri kompresije prikazani su u postotcima, a u svakoj iteraciji algoritama rezultati su jednaki.

Tablica 4.11 Omjer kompresije slika QOI i Brotli algoritmima

Slika	Omjer kompresije za QOI algoritam (%)	Omjer kompresije za Brotli algoritam (%)
dog.tiff	54.14	54.57
pug.jpg	66.65	74.27
flower.jpg	32.98	43.36
bird.png	80.28	86.10
alfa.jpg	45.41	55.86

Iz rezultata je vidljivo kako Brotli algoritam daje približno jednak omjer kom-

presije za neke slike (u ovom slučaju za *dog.tiff* sliku je bolji samo za 0.43%) kao i QOI algoritam, a za neke slike daje bolje rezultate za oko 8 do 10%, ovisno o slici. Najveću razliku daje za sliku *alfa.jpg* gdje je bolji za 10.45% od QOI algoritma.

U tablici 4.12 prikazani su rezultati mjerenja vremena izvršavanja kodiranja slika za QOI i Brotli algoritme. U prvom stupcu nalazi se naziv slike, u drugom i trećem stupcu vrijeme kodiranja i standardna devijacija vremena, u sekundama, za QOI algoritam, a u četvrtom i petom stupcu vrijeme kodiranja slike i standardna devijacija za Brotli algoritam, također izraženi u sekundama.

Tablica 4.12 Vrijeme kodiranja slika QOI i Brotli algoritmima

Slika	Vrijeme kodiranja za QOI algoritam (s)	St.dev. (s)	Vrijeme kodiranja za Brotli algoritam (s)	St. dev. (s)
dog.tiff	146.4381	1.7299	194.2375	16.6799
pug.jpg	71.2044	0.5061	85.5605	11.5841
flower.jpg	120.5402	1.9860	307.7507	15.9937
bird.png	43.8049	0.1930	77.2783	9.4303
alfa.jpg	45.6521	0.4147	72.7139	5.5535

Iz rezultata je vidljivo kako je QOI algoritam brži u kodiranju, odnosno kompresiji slike od Brotli algoritma, za svaku sliku korištenu u analizi. Za sliku *flower.jpg* postigao je najbolji rezultat u odnosu na Brotli algoritam, odnosno brži je ≈ 2.55 puta u kompresiji slike.

U tablici 4.13 prikazana su izmjerena vremena izvršavanja dekodiranja, za svaku sliku prikazanu u poglavlju 4.3.1, za QOI i Brotli algoritme. Kao i kod prikaza rezultata za BZ2 algoritam, vremena dekodiranja i standardne devijacije prikazane su za oba algoritma, a navedeni podaci izraženi su u sekundama.

Tablica 4.13 Vrijeme dekodiranja slika QOI i Brotli algoritmima

Slika	Vrijeme dekodiranja za QOI algoritam (s)	St.dev. (s)	Vrijeme dekodiranja za Brotli algoritam (s)	St. dev. (s)
dog.tiff	83.8130	0.9174	0.4651	0.0385
pug.jpg	51.5446	0.1336	0.3519	0.4239
flower.jpg	69.8353	1.7724	0.4077	0.0253
bird.png	30.0133	0.0919	\	\
alfa.jpg	32.2059	0.3806	0.1209	0.0114

Iz prikazanih rezultata u tablici 4.13 može se zaključiti kako je Brotli algoritam značajnije brži od QOI algoritma u dekodiranju slika, primjerice za sliku *alfa.jpg* brži je čak ≈ 266.38 puta.

Međutim, Brotli algoritam nije mogao dekodirati sliku *bird.png*. Provedeno je testiranje na drugim slikama istoga formata, koje je uspješno dekodirao, stoga se ne može zaključiti kako uspješnost Brotli algoritma ovisi o formatu zapisa slike. Prema tome, QOI algoritam je pouzdaniji algoritam po pitanju uspješnog kodiranja i dekodiranja slika.

4.6 Usporedba QOI algoritma s LZ4 algoritmom

U tablici 4.14 prikazani su omjeri kompresije svih slika korištenih u provedbi mjerenja, nakon kompresije QOI i LZ4 algoritmima. Kao i kod prikaza rezultata prethodnih algoritama, omjeri kompresije prikazani su u postotcima, a u svakoj iteraciji algoritama rezultati su jednaki.

Tablica 4.14 Omjer kompresije slika QOI i LZ4 algoritmima

Slika	Omjer kompresije za QOI algoritam (%)	Omjer kompresije za LZ4 algoritam (%)
dog.tiff	54.14	8.53
pug.jpg	66.65	47.81
flower.jpg	32.98	8.90
bird.png	80.28	65.42
alfa.jpg	45.41	17.65

Iz rezultata je vidljivo kako je QOI algoritam bolji po omjeru kompresije od LZ4 algoritma, za svaku od testnih slika. Primjerice, za *bird.png* bolji je za 14.86%, dok je, za sliku *dog.tiff*, QOI algoritam postigao omjer kompresije od 54.14%, a LZ4 algoritam samo 8.53%, što daje razliku od 45.61%.

U tablici 4.15 prikazani su rezultati mjerenja vremena izvršavanja komprimiranja slika za QOI i LZ4 algoritme. Kao i kod rezultata prethodnih algoritama, u prvom stupcu nalazi se naziv slike, u drugom i trećem stupcu vrijeme kodiranja i standardna devijacija vremena za QOI algoritam, a u četvrtom i petom stupcu vrijeme kodiranja slike i standardna devijacija za LZ4 algoritam. Izmjerena vremena i standardne devijacije izražene su u sekundama.

Tablica 4.15 Vrijeme kodiranja slika QOI i LZ4 algoritmima

Slika	Vrijeme kodiranja za QOI algoritam (s)	St.dev. (s)	Vrijeme kodiranja za LZ4 algoritam (s)	St. dev. (s)
dog.tiff	146.4381	1.7299	0.2425	0.0275
pug.jpg	71.2044	0.5061	0.3404	0.0078
flower.jpg	120.5402	1.9860	0.3832	0.0093
bird.png	43.8049	0.1930	0.1706	0.0086
alfa.jpg	45.6521	0.4147	0.1354	0.0034

Iz prikazanih rezultata vidljivo je kako je LZ4 algoritam puno brži u kodiranju slika od QOI algoritma. Primjerice, za sliku *dog.tiff* brži je čak ≈ 603.87 puta.

U tablici 4.16 prikazana su izmjerena vremena izvršavanja dekodiranja slike QOI i LZ4 algoritmima, za svaku od testnih slika prikazanih u poglavlju 4.3.1. U prvom stupcu nalazi se naziv slike, u drugom i trećem stupcu vrijeme dekodiranja i standardna devijacija vremena, u sekundama, za QOI algoritam, a u četvrtom i petom stupcu vrijeme dekodiranja slike i standardna devijacija za LZ4 algoritam, također u sekundama.

Tablica 4.16 Vrijeme dekodiranja slika QOI i LZ4 algoritmima

Slika	Vrijeme dekodiranja za QOI algoritam (s)	St.dev. (s)	Vrijeme dekodiranja za LZ4 algoritam (s)	St. dev. (s)
dog.tiff	83.8130	0.9174	0.0491	0.0044
pug.jpg	51.5446	0.1336	0.0259	0.0024
flower.jpg	69.8353	1.7724	\	\
bird.png	30.0133	0.0919	\	\
alfa.jpg	32.2059	0.3806	0.0123	0.0007

Iz prikazanih rezultata u tablici 4.16 vidljivo je kako je LZ4 algoritam značajnije brži od QOI algoritma. Primjerice, sliku *alfa.jpg* dekodirao je ≈ 2618.37 puta brže.

Međutim, LZ4 algoritam nije mogao dekodirati slike *bird.png* i *flower.jpg*, odnosno nije uspio dekodirati dvije od pet testnih slika. Prema tome, iz rezultata se može zaključiti, isto kao i za Brotli algoritam, kako je QOI algoritam pouzdaniji po pitanju kodiranja i dekodiranja slika od LZ4 algoritma.

Poglavlje 5

Zaključak

U ovom radu opisan je Quite OK Image Format, odnosno QOI algoritam, algoritam za kompresiju slike bez gubitaka te je implementiran u programskom jeziku Python. Kompresija slike bez gubitaka označava oblik kompresije u kojoj se ne gube podaci ni kvaliteta slike. QOI algoritam namijenjen je kompresiji RGB ili RGBA slika te se svaki piksel kodira na jedan od četiri moguća načina. Piksel se kodira na prvi način ako je jednak prethodnom, na drugi način ako je jednak jednom od prethodno viđenih piksela, na treći način ako je razlika u vrijednostima komponenti trenutnog i prethodnog piksela unutar unaprijed određenog intervala te, ako ne vrijedi ni jedan od prethodnih slučajeva, upisuju se cijele vrijednosti komponenti piksela.

U drugom dijelu rada prikazana je ovisnost vremena izvršavanja QOI algoritma o veličini slike te je dokazano kako performanse algoritma ne ovise o formatu zapisa slike. Također je prikazana usporedba QOI algoritma s Bzip2, Brotli te LZ4 algoritmima po pitanju vremena izvršavanja algoritama (kodiranja i dekodiranja slike) te omjera kompresije slika. Navedeni algoritmi su također algoritmi za kompresiju bez gubitaka, Bzip2 i Brotli algoritmi su algoritmi složeni od više jednostavnih algoritama za kompresiju, dok LZ4 algoritam pripada LZ77 skupini algoritama.

Analiza je provedena na pet testnih slika. Rezultati su pokazali kako je Bzip2 algoritam bolji od QOI algoritma i po pitanju omjera kompresije i po pitanju izvršavanja kodiranja i dekodiranja slika. Brotli algoritam je brži u slučaju dekodiranja slika, ali je sporiji od QOI algoritma u slučaju kodiranja slika. Također, Brotli algoritam nije uspješno dekodirao jednu od testnih slika, pa se može zaključiti kako je i manje stabilan od QOI algoritma. LZ4 algoritam daje lošije rezultate po pitanju omjera kompresije od QOI algoritma, ali je značajnije brži i po pitanju kodiranja i po pitanju dekodiranja slika. Međutim, nije uspješno dekodirao dvije od pet test-

nih slika. Prema tome, kao i za Brotli algoritam, može se zaključiti kako je manje stabilan od QOI algoritma.

Daljnji rad i detaljnija analiza mogla bi dovesti do zaključka zašto je QOI algoritam značajnije sporiji od Bzip2 i LZ4 algoritama u kodiranju i dekodiranju slika te od Brotli algoritma u slučaju dekodiranja slika. Također, mogla bi se optimizirati implementacija algoritma u programskom jeziku Python, na način da se analiziraju korištene funkcije te, po potrebi, zamijene s efikasnijim funkcijama. S obzirom da se u slučaju indeksiranja pomoću prethodno viđenih piksela koristi polje fiksne veličine, veličina toga polja mogla bi se mijenjati te analizirati ovisnost omjera kompresije i vremena izvršavanja algoritma o toj vrijednosti te doći do vrijednosti koja bi dala najbolji omjer kompresije za zadovoljavajuće vrijeme izvršavanja algoritma.

Bibliografija

[1] "Visual Resources", s Interneta, https://visualresources.princeton.edu/making-images/digital-image-basics/, 11. svibnja 2023.

[2] "Vrste slika", s Interneta, http://preservationtutorial.library.cornell.edu/tutorial/intro/intro-04.html, 11. svibnja 2023.

[3] "Image file formats", s Interneta, https://guides.lib.umich.edu/c.php?g=282942&p=1885348, 11. svibnja 2023.

[4] "Image compression", s Interneta, https://www.techtarget.com/whatis/definition/image-compression, 11. svibnja 2023.

[5] S. Dhawan, "A Review of Image Compression and Comparison of its Algorithms", Deptt. of ECE, UIET, Kurukshetra University, Kurukshetra, Haryana, India, 2011.

[6] "JPG kompresija", s Interneta, https://www.baeldung.com/cs/jpeg-compression, 29. lipnja 2023.

[7] "Lossless Image Compression in O(n) Time", s Interneta, https://phoboslab.org/log/2021/11/qoi-fast-lossless-image-compression, 13. svibnja 2023.

[8] M. Bucev, V. Kunčak, "Formally Verified Quite OK Image Format", School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland, 2022.

[9] "QOI specifikacija", s Interneta, https://qoiformat.org/qoi-specification.pdf, 13. svibnja 2023.

[10] "Bzip2 algoritam", s Interneta, https://en.wikipedia.org/wiki/Bzip2, 17. lipnja 2023.

[11] "Brotli algoritam", s Interneta, https://en.wikipedia.org/wiki/Brotli, 17. lipnja 2023.

[12] "LZ77 algoritam", s Interneta, https://en.wikipedia.org/wiki/LZ77_and_LZ78, 17. lipnja 2023.

[13] "LZ4 algoritam", s Interneta, `https://en.wikipedia.org/wiki/LZ4_(compression_algorithm)`, 30. lipnja 2023.

[14] "Programski jezik Python", s Interneta, `https://www.python.org/about/`, 18. svibnja 2023.

[15] "Funkcije *copy()* i *deepcopy()*", s Interneta, `https://docs.python.org/3/library/copy.html`, 20. svibnja 2023.

[16] "Struktura *deque*", s Interneta, `https://docs.python.org/3/library/collections.html#collections.deque`, 14. lipnja 2023.

[17] "Knjižnica *Pillow*", s Interneta, `https://pypi.org/project/Pillow/`, 21. svibnja 2023.

[18] "Modul *Image*", s Interneta, `https://pillow.readthedocs.io/en/stable/reference/Image.html`, 21. svibnja 2023.

[19] "Modul *os.path*", s Interneta, `https://docs.python.org/3/library/os.path.html`, 21. svibnja 2023.

[20] "Knjižnica *time*", s Interneta, `https://docs.python.org/3/library/time.html`, 22. svibnja 2023.

[21] "Knjižnica *sys*", s Interneta, `https://docs.python.org/3/library/sys.html`, 22. svibnja 2023.

[22] "Knjižnica *imagecodecs*", s Interneta, `https://pypi.org/project/imagecodecs/`, 17. lipnja 2023.

[23] "StockSnap.io", s Interneta, `https://stocksnap.io/`, 14. lipnja 2023.

[24] "cleanpng.com", s Interneta, `https://www.cleanpng.com/`, 14. lipnja 2023.

Sažetak

Tema ovoga rada je implementacija Quite OK Image Format (QOI) algoritma u programskom jeziku Python te njegova usporedba s drugim, često korištenih algoritmima za kompresiju slike po pitanju omjera kompresije i vremena izvršavanja algoritama. QOI algoritam je algoritam za kompresiju slike bez gubitaka te kodira svaki od piksela na jedan od četiri moguća načina. Prikazana je usporedba QOI algoritma s Bzip2 i Brotli algoritmima, koji su također algoritmi za kompresiju bez gubitaka. Rezultati su pokazali kako implementacija QOI algoritma u programskom jeziku Python daje manji omjer kompresije slike te je vrijeme kodiranja i dekodiranja slike dulje.

Ključne riječi — **QOI algoritam, kompresija slike bez gubitaka, Bzip2 algoritam, Brotli algoritam**

Abstract

The topic of this Master's thesis paper is the implementation of the Quite OK Image Format (QOI) algorithm using the Python programming language and its comparison with other commonly used image compression algorithms by measuring compression ratio and algorithm execution time. The QOI algorithm is a lossless image compression algorithm that encodes each pixel in one of four possible ways. Thesis shows the comparison between the QOI algorithm and the Bzip2 and Brotli algorithms, which are also lossless compression algorithms. The results showed that the implementation of the QOI algorithm in the Python programming language yields a lower image compression ratio and longer encoding and decoding time.

Keywords — **QOI algorithm, lossless compression, Bzip2 algorithm, Brotli algorithm**

Dodatak A
Github repozitorij

A.1 Poveznica na repozitorij

- `https://github.com/Ivanastimac/Diplomski-rad`