Kompresija slike nakon pretvaranja u jednodimenzionalan niz

Grgur, Filip

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka, Faculty of Engineering / Sveučilište u Rijeci, Tehnički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:190:647422

Rights / Prava: Attribution 4.0 International/Imenovanje 4.0 međunarodna

Download date / Datum preuzimanja: **2024-07-04**

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Engineering

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij računarstva

Diplomski rad

KOMPRESIJA SLIKE NAKON PRETVARANJA U JEDNODIMENZIONALAN NIZ

Rijeka, rujan 2022.

Filip Grgur

0069082445

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij računarstva

Diplomski rad

KOMPRESIJA SLIKE NAKON PRETVARANJA U JEDNODIMENZIONALAN NIZ

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

Rijeka, rujan 2022.

Filip Grgur

0069082445

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 21. ožujka 2022.

Zavod: **Zavod za računarstvo**
Predmet: **Teorija informacija i kodiranje**
Grana: **2.09.03 obradba informacija**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Filip Grgur (0069082445)**
Studij: Diplomski sveučilišni studij računarstva
Modul: Računalni sustavi

Zadatak: **Kompresija slike nakon pretvaranja u jednodimenzionalan niz / Image Compression After Transformation to One-dimensional Array**

Opis zadatka:

Potrebno je pretvoriti sliku u jednodimenzionalan niz korištenjem više algoritama (uključujući i Hilbertovu krivulju). Zatim je potrebno komprimirati niz nekim od postupaka za kompresiju teksta te usporediti rezultate s klasičnim algoritmima za kompresiju slike.

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

Grgur Filip

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Izv. prof. dr. sc. Jonatan Lerga

Predsjednik povjerenstva za diplomski ispit:

Prof. dr. sc. Kristijan Lenac

IZJAVA

Izjavljujem da sam završni rad „Kompresija slike nakon pretvaranja u jednodimenzionalan niz“ izradio samostalno, koristeći literaturu i znanje stečeno na Tehničkom fakultetu Rijeka, uz pomoć mentora izv. prof. dr. sc. Jonatan Lerge.

_____

Filip Grgur

ZAHVALA

Zahvaljujem svim kolegama, profesorima i mentoru na ukazanoj potpori i prenesenom znanju, a naročito svojim roditeljima koji su me bodrili cijelim putem studiranja.

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

1. UVOD

Sve globalne tvrtke koriste i zapisuju podatke na svoje ili iznajmljene poslužitelje (*engl*. server). Činjenica je da je to samo jedan od ograničenih resursa kojima tvrtka upravlja. Svi digitalni podaci su zapisani kao nule i jedinice neovisno radi li se o slici, tekstu ili nekom drugom formatu. Kako bi se taj prostor na poslužiteljima iskoristio maksimalno, podaci bi se trebali kompresirati. Danas računala, mobiteli, kamere te ostali digitalni uređaji generiraju podatke koji zauzimaju puno računalne memorije. Npr. obična audio snimka zauzima nekoliko megabajta (MB) dok video snimka može zauzimati i po nekoliko gigabajta (GB). Što je datoteka veća to će se teže prenijeti na Internet ili s jednog računala na drugo. Veličinu datoteka bi trebalo smanjiti uz pomoć kompresije. Što je kompresija bolja, to će datoteka biti manja te će se lakše prenijeti na Internet ili između računala. Jedini nedostatak kompresije je što povećava upotrebu računalnih resursa, stoga dolazi do balansiranja između kvalitete kompresije i iskorištenih računalnih resursa (vrijeme pod kojim je sustav opterećen, razina opterećenja).

Kompresija može biti s gubicima (*engl.* lossy compression) i bez gubitaka (*engl.* lossless compression). Kompresija podataka s gubicima se odrađuje tako se dio podataka makne, čime se smanjuje ukupna količina podataka, kod slika je to dubina slike (*engl.* colour depth), odnosno raspon boja na slici. Kompresija bez gubitka podataka je vrlo važna za tekstualne datoteke, elektroničku poštu i poslovne dokumente. Kompresija bez gubitaka ne nudi toliko smanjenje veličine datoteka kao kompresija s gubitcima[1].

Zadatak ovog rada je proces kompresije slika bez gubitaka, PNG slika (*engl*. Portable Network Graphic), pretvaranjem u niz različitim algoritmima i korištenjem raznih kompresijskih alata. Iz slike se očita pojedine piksele u redoslijedu zadanog algoritma. Iako je fokus ovog rada na algoritmu Hilbertove krivulje pokriven je i algoritam Z krivulje kao i algoritam iteracije po redovima, stupcima i dijagonali. Ovaj projekt primarno služi kao temelj ljudima koji žele nastaviti istraživati ovo područje kompresije podataka te onima koji žele implementirati druge algoritme stvaranja nizova iz slika .

2. SLIKE I PIKSELI

Digitalna slika se može definirati kao prostor veličine M puta N u kojemu svaka točka (piksel) ima svoju koordinatu u dvodimenzionalnom Kartezijevom koordinatnom sustavu (*engl.* Cartesian coordinate system). M i N predstavljaju ukupan broj stupaca i redaka slike. Početni piksel slike se nalazi u gornjem lijevom kutu s indeksom P(0, 0), donji lijevi piksel je označen s P(0, N - 1), dok je gornji desni piksel označen s P(M - 1, 0), te donji desni piksel ima koordinate P(M - 1, N - 1). Kod testnih slika ovog projekta bile su izabrane slike dimenzija gdje su M i N jednaki, tj. slike dimenzija N i N. S time da N treba biti potencija broja 2 zbog Hilbertove krivulje, Z krivulje i iteracije po dijagonali. Razlog tome je što ta tri algoritma stvaraju kvadrat na svakoj iteraciji potencije broja 2, dok algoritmi po stupcima i redcima ne ovise o dimenzijama slike. Slika sadrži jedan ili više kanala u boji koji definiraju intenzitet ili boju pojedinom pikselu. U najjednostavnijem slučaju svaki piksel sadrži samo jednu numeričku vrijednost (0 ili 1) koja predstavlja jačinu signala (boje) u tom mjestu na slici. Pretvorba tog signala u stvarnu sliku se ostvaruje preko mape boja (*engl.* colour map). Mapa boja dodjeljuje određenu nijansu boje svakoj razini na slici kako bi se vizualno prikazali podaci. Jedna od mapa boja je siva, te dodjeljuje 256 nijanse sive (uključujući crnu i bijelu) ovisno o razini signala. Osim slika u sivoj boji koja ima samo jednu vrijednost na mjestu piksela (0 do 255) postoje i slike u bojama koje obuhvaćaju cijeli spektar boja. Taj spektar boja je predstavljen kao trostruki vektor (R,G,B) za svaki piksel. Boja piksela je predstavljena kao linearna kombinacija osnovnih boja crvene, zelene i plave (*engl.* Red, Green, Blue (RGB)) te se može reći da se slika sastoji od tri dvodimenzionalne ravnine. Postoji također prikaz boja preko nijanse, zasićenosti i intenziteta (HSV) (*engl.* Hue, saturation, value)[2].

2.1 Rezolucija

Rezolucija slike se može definirati kao broj detalja koje slike sadrži, ili jednostavnije, koliko blizu mogu biti dvije linije na slici, a da su vidljivo razdvojene. Rezolucija se inače poistovjećuje s brojem piksela na digitalnoj slici. Rezolucija se može zapisati kao dva broja, prvi broj koji označava broj stupaca piksela, a drugi broj redaka piksela (npr. 1920 x 1080). Drugi način zapisivanja rezolucije se radi tako se zapiše ukupan broj piksela na slici, poznat kao broj megapiksela (*engl.* megapixels). On se računa množenjem broja stupaca piksela i broja redova piksela, čiji se produkt podijeli s

milijunom kako bi se od broja piksela dobili megapikseli. Još jedna od poznatijih mjera kod digitalnih slika je PPI (*engl.* Pixels per inch *i/ili* Pixels per square inch), odnosno broj piksela po kvadratnom inču. Slika čija širina iznosi 1920 piksela i visina 1080 piksela, ukupno ima 2.1 megapiksela (1920 x 1080 = 2073600 piksela ili 2.1 megapiksel). Ta slika bi bila jako loše rezolucije ako bi se razvukla preko velikog ekrana (npr. 48 PPI), dok bi na manjem ekranu izgledala bolje jer bi imala više PPI (npr. 256 PPI).

Veličina slike je definirana njenom rezolucijom. Rezolucija slike se može podijeliti na 3 dijela:

- Prostorna rezolucija (*engl*. Spatial resolution) se definira kao broj piksela u retku pomnožen s brojem piksela u stupcu. Često se označava kao C x R (stupci puta redci) (npr. 640 x 480, 1024 x 768, 1440 x 720).

- Vremenska rezolucija (*engl.* Temporal resolution) za sustave koji snimaju video zapise, se mjeri kao broj slika koji je snimljen u određenome vremenskom periodu. Označen je kao broj slika u sekundi (FPS) (*engl.* Frames per second).

- Rezolucija bita (*engl.* Bit resolution) je broj mogućih boja koje piksel može poprimiti. Binarna slika može poprimiti samo dvije boje (crnu i bijelu), slika u sivim tonovima (*engl.* grey-scale image) može poprimiti 256 različitih nijansi sivih uključujući crnu i bijelu boju, dok slika u boji ovisi o rasponu boja koje podržava. Rezolucija bita se najčešće zapisuje kao broj bitova koji je potreban da se spremi pojedini piksel, za binarnu sliku to iznosi 2 bita, za sliku u sivim tonovima to iznosi 8 bita, dok za slike u boji je to najčešće 24 bita[2].

2.2 Formati slika

Kako postoji potreba za efikasnim prikazom slika, spremanjem i prebacivanjem istih preko internetskih mreža, razvio se standard za formate digitalnih slika. Format se sastoji od zaglavlja koje govori na koji način je spremljena fotografija te se sastoji od brojčanih vrijednosti. Neki od najviše korištenih formata slike se nalaze u tablici 2.1.

Tablica 2.1. Najkorišteniji formati slika

Format	Puni naziv	Svojstva
GIF	Graphics interchange format	Ograničen na 256 boja (8-bit), kompresija bez gubitaka
JPEG	Joint Photographic Experts Group	Najkorišteniji format, kompresija s gubitcima
BMP	Bit map picture	Običan format slike, kompresija bez gubitaka, nastao od Windows-a
PNG	Portable network graphics	Kompresija bez gubitaka, dizajniran za zamjenu GIF-a
TIF / TIFF	Tagged image (file) format	Nekompresirani oblik slike, pun detalja

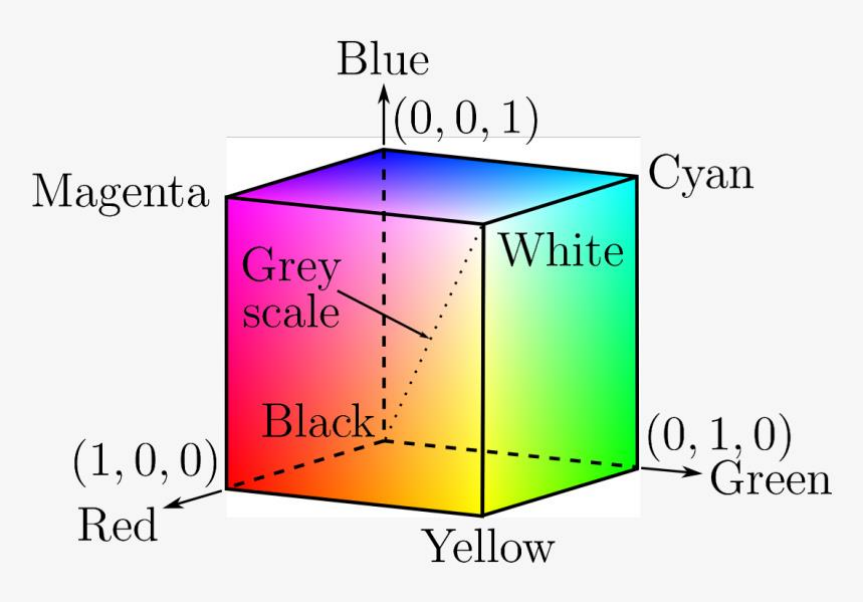
Ovisno o potrebi koriste se drugačiji formati slika. GIF slike su ograničene na samo 256 boja ili 256 nijansi sive boje, što je definirano mapom boja u zaglavlju datoteke. JPEG slike podržavaju čak 24 bitne boje (RGB) i koriste se dosta kod digitalnih fotoaparata. PNG je nastao kao zamjena za GIF, tj. unaprjeđenje za kompresiju bez gubitaka[3].

2.3 Boje i pikseli

Riječ piksel skraćenica je od elementa slike (*engl.* picture element). Pozicija piksela na slici dimenzija M x N označuje se kao P(x, y) odnosno broj stupca (x: [0, M-1]) i retka (y: [0, N-1]). Piksel je najmanji element u digitalnoj slici koji sadrži brojčanu vrijednost koja predstavlja boju ili

intenzitet. Stoga, RGB slika je trodimenzionalni niz kojemu su svakom pikselu pridodijeljene brojčane vrijednosti, svaka vrijednost odgovara crvenoj, zelenoj i plavoj boji, u tom redoslijedu. Trodimenzionalan niz RGB slike može se definirati kao tri dvodimenzionalna niza, od čega se niz sastoji od stupaca i redaka, jedan za svaku boju.

Ako se postavi pretpostavka da se sve boje mogu prikazati s RGB tehnologijom, onda se prostor RGB boja može opisati prostornom kockom boja s dimenzijama crvene, zelene i plave boje. Svaka dimenzija ima vrijednosti od 0 do 1. Vrijednost je skalirana na 0 - 255 jer se zapiše u jedan bajt (*engl.* byte), što čini 3 bajta po pikselu. Crna boja se nalazi u „početku" kocke (0,0,0) dok bijela boja nastaje miješanjem sve tri boje u točki (1,1,1). Sve ostale boje se nalaze unutar tih granica, što je prikazano na slici 2.1. Boje koje se nalaze unutar tih granica kocke su bazirane u elektromagnetskom spektru koji je vidljiv čovjeku (valne duljine od 380 nm do 740 nm)[4].



Slika 2.1. Prikaz boja u 3D prostoru[5]

2.4 Testne slike

Za ovaj projekt odabrane su testne slike formata PNG, format slika bez gubitaka, kako bi slike ostale u istoj rezoluciji nakon kompresije. Slika 2.2 (*ventilator.png*) i slika 2.3 (*sunflower.png*) su uslikane s kamerom pametnog telefona te automatski spremljene u PNG format i odrezane na potrebnu dimenziju. Slika 2.4 (*peppers.png*) i slika 2.5 (*fruits.png*) su standardne slike za obradu te su preuzete sa Github repozitorija[6]. Slike *ventilator.png* i *sunflower.png* su prostorne rezolucije 1024 x 1024 piksela, dok su *peppers.png* i *fruits.png* prostorne rezolucije 512 x 512 piksela.

Kao što je prije navedeno slike moraju biti prostorne rezolucije N x N čime je N broj potencije broja 2, zbog Hilbertove krivulje, Z krivulje i iteracije po dijagonali. Rezolucija bita za sve četiri slike iznosi 24, 8 bitova za predstavljanje crvene boje, 8 bitova za predstavljanje zelene boje i 8 bitova za predstavljanje plave boje. S 8 bitova moguće je zapisati 256 nijansi svake od ove tri boje, što daje 16,777,216 (256 * 256 * 256) mogućih boja koje piksel može poprimiti. Također postoje PNG slike koje koriste RGBA piksele. RGBA (Red, Green, Blue, Alpha), gdje *Red*, *Green*, i *Blue* označavaju boje dok *Alpha* označuje transparentnost piksela (*engl.* transparency). Ova vrsta piksela je zapisana pomoću 32 bita dok zadnjih 8 bitova označuje transparentnost. U ovome projektu se slike s RGBA pikselima se neće koristiti jer kod pretvorbe slike u RGB oblik se gubi dio informacije zbog toga što je izlazna slika uvijek RGB oblika. PNG slike čiji su pikseli zapisani na 8 bitova (*engl.* greyscale) se neće koristiti jer je sliku potrebno pretvoriti u RGB oblik čime se povećava „originalna“ veličina slike.

Slika 2.2. ventilator.png

Slika 2.3. sunflower.png

Slika 2.4. peppers.png

Slika 2.5. fruits.png

3. ALGORITMI I TEHNOLOGIJE

Za razvoj programa korišten je „lagan“ uređivač koda (*engl.* lightweight) Visual Studio Code[7] (VS Code). VS Code je okruženje za uređivanje koda koje podržava operacije poput: otklanjanja pogrešaka (*engl.* debugging), pokretanja zadataka (*engl.* task running) i kontroliranja verzija (*engl.* version control). VS Code također omogućuje jednostavnu implementaciju dodatnih proširenja (*engl.* extension) koja olakšavaju korištenje aplikacije. Postoje dodatci za većinu programskih jezika koji omogućuju lakše snalaženje u programskome kodu. Koristili su se dodatci „Code Runner“ za pokretanje programa i „Python“ za lakše snalaženje u napisanome kodu.

3.1 Python

Cijeli programski kod napisan je u programskom jeziku Python[8]. Korišten je python3, novija verzija programskog jezika Python koja je objavljenja u prosincu 2008. godine. Korištena je njegova zadnja trenutačna inačica: Python 3.10.6. Odabrani jezik je ima veliki broj gotovih knjižnica te je lagan za implementaciju. Knjižnice koje su potrebne za izradu ovog projekta: os, gzip, py7zr, bz2, time, os i pillow.

Knjižnica os[9] omogućava manipuliranje putanjama (*engl.* path), čitanje, pisanje i otvaranje datoteka, također može stvarati privremene datoteke i direktorije te može očitati veličine datoteka i direktorija. Primarno je korištena funkcionalnost za očitanje veličine datoteka i direktorija kako bi se mogla usporediti veličina dobivenih kompresiranih nizova.

Knjižnica gzip[9] omogućava jednostavno sučelje za kompresiranje (*engl.* compress) i dekompresiranje (*engl.* decompress) datoteka upravo onako kako bi program gzip to napravio. Stvara se GzipFile klasa koja omogućuje funkcije poput: open(), compress() i decompress(). GzipFile čita i zapisuje datoteke formata .gzip te automatski kompresira ili dekompresira podatke tako da izgledaju kao obični objekt. Glavne funkcionalnosti ove knjižnice su iskorištene u projektu radi kompresiranja nizova te pohranjivanja istih.

Kako ne bi projekt bio ograničen na samo jednu vrstu kompresiranja teksta implementirane su knjižnice bz2[9] i py7zr[10]. Knjižnica py7zr je ujedno i program za podršku kompresiranja, dekompresiranja, enkripcije i dešifriranja 7zip arhiva. Podržava algoritme i filtre koji podržavaju LZMA(*engl*. Lempel–Ziv–Markov chain) modul i liblzma, također podržava BZip2, Deflate, Zstandard i Brotli. Knjižnica bz2 pruža kompresiranje i dekompresiranje podataka te koristi Burrows-Wheeler algoritam.

Knjižnica time[9] daje mogućnosti praćenja vremena i manipulacijom razlika vremenske zone. Ponuđeno je moguće praćenja vremena u obliku: 'Sun Jun 20 23:21:05 1993' ili vrijeme pretvoreno u sekunde. Mjeri se koliko je sekundi prošlo od „January 1, 1970, 00:00:00 (UTC)“ do trenutka pozivanja funkcije *time.time()*. Funkcija se poziva prije pretvaranja slike u niz i nakon što je taj proces gotov, te dvije vrijednosti se oduzmu i dobije se vrijeme koje je potrebno određenom algoritmu da pretvori sliku u niz.

PIL (Python Imaging Library) nudi mogućnosti obrade slika putem programskog koda. Osnovna biblioteka je dizajnirana za brzi pristup podacima pohranjenim u nekoliko osnovnih formata piksela (npr. L, RGB, RGBA, CMYK, LAB, HSV). PIL je idealan za arhiviranje i manipuliranje slikama. Služi za izradu naslovnica, pretvaranje formata datoteka, ispis slika itd. Za otklanjanje pogrešaka postoji metoda *show()* koja sprema sliku na disk i poziva program za prikaz slike. Knjižnica sadrži osnovnu funkcionalnost obrade slike, uključujući operacije s pojedinim pikselima, filtriranje sa skupom ugrađenih konvolucijskih jezgri (*engl.* built-in convolution kernels) i pretvorbe prostora boja (*engl*. colour space conversions). Ujedno podržava i promjenu veličine slike, rotaciju i proizvoljne affine transformacije (*engl*. arbitrary affine transforms). Sadrži metodu histograma, ona omogućuje izvlačenje statistike iz slike koja se koristi za automatsko poboljšanje kontrasta i za globalnu statističku analizu[11].

3.2 Opis rada programa

Program se sastoji od više dijelova, dijela za učitavanje i pretvorbu slika u potreban PNG format, dijela za pretvaranja slike u niz pomoću različitih algoritama te dijela za kompresija niza i vraćanje slike iz kompresije.

3.2.1. Učitavanje i kompresija podataka

Prije učitavanja slike potrebno je postaviti radni direktorij gdje se nalazi glavni program *main.py*. Onda se učita željena slika koja će se kompresirati iz direktorija *pics*. Pošto su sve slike dimenzija N x N učitava se samo jedna od dimenzija , u ovom slučaju je to širina (*engl.* width). Ako PNG slika koja je učitana nije RGB oblika, ona će biti pretvorena (*engl.* convert) u RGB oblik pomoću gotove funkcije *.convert()* iz knjižnice *pillow*. Nakon što je slika otvorena, učita se mapa piksela slike u varijablu *px*.

```
putanja="C:\\Users\\Tron\\Desktop\\hilbert"
im = Image.open(r""+putanja+"\\pics\\ventilator.png")
N = im.width
rgb_im = im.convert('RGB')
px = rgb_im.load()
```

Nakon što je učitana mapa piksela pokrene se prvi algoritam stvaranja niza, te kada on završi se pokrene drugi. Za svaki od pet algoritama stvaranja niza se napravi prazna varijabla *string_imeAlgoritma*. Budući da je slika N x N dimenzija, broj iteracija potreban da se prođe kroz sve piksele iznosi N^2. Na početku svakog algoritma prvo se pozove funkcija *time()* iz knjižnice *time* koja dohvati trenutačno vrijeme, te se na kraju svakog algoritma pozove ista ta funkcija kako bi se mogla izračunati vremenska razlika. Ta vremenska razlika je vrijeme potrebno algoritmu da zapiše sve piksele u niz određenim redoslijedom.

```
start = time.time()
string_hilbert = ""
i=0
while i < N*N:
    curr = hildex2xy(i,N)
    string_hilbert += rgb_to_hex(px[curr[0], curr[1]])
    i+=1

end = time.time()
current = end - start
```

Vrijednost željenog piksela se očita preko mape piksela. Kod zapisivanja boje piksela, prvo se pošalju RGB vrijednosti piksela (npr. (100, 200, 50)) u funkciju *rgb_to_hex()*. U toj funkciji se vrijednost svake boje piksela pretvori u dva heksadecimalna znaka, čime se dobije nazad šest heksadecimalnih znakova koji se zapišu na kraj trenutnog niza.

```
def rgb_to_hex(rgb):
     return '%02x%02x%02x' % rgb
```

Ovdje je kao primjerak bio dan isječak koda za algoritam Hilbertove krivulje, no svi algoritmi stvaranja nizova će biti opisani u kasnijim poglavljima.

Svaki niz se spremi u svoju datoteku pod nazivom *vrstaAlgoritma_pic.txt*. Ta datoteka predstavlja sliku u nekompresiranom nizu, a ona služi za lakše računanje omjera kompresije te pomaže kod kompresije podataka pomoću 7zip-a.

```
hilbert_txt = open(putanja+"\\hilbert_pic.txt","w")
hilbert_txt.write(string_hilbert)
hilbert_txt.close()
```

Sljedeći korak je kompresiranje nizova i usporedba njihovih performansi. Ovdje je za primjer navedena samo Hilbertova krivulja, no postupak je isti za ostala četiri algoritma. Prije svake kompresije, neovisno o alatu, pokreće se funkcija *time()*. Kod imenovanja kompresiranih nizova, svako ime datoteke je napisano na način:

nazivAlataZaKompresiju_nazivKorištenogAlgoritma.nastavakAlataZaKompresiju

Stoga ovako izgleda primjer za algoritam Hilbertove krivulje pomoću gzip-a: *gzip_hilbert.txt.gz*, pomoću 7zip-a: *7z_hilbert.7z*, te pomoću bz2: *bz2_hilbert.bz2*. Kada se kompresira niz koristeći alat gzip nastavak datoteke je *.txt.gz*. dok kod 7zip-a je to *.7z* te kod bz2 alata on je *.bz2*. Ovi alati za kompresiju podataka su implementirani kako je preporučeno na njihovim službenim stranicama knjižnica[9][10]. Kada je kompresija uspješno napravljena, ponovno se pozove funkcija *time()* te se izračuna vrijeme potrebno za određenu kombinaciju alata kompresije i algoritma stvaranja niza. Onda se zove funkcija *printaj()* koja za argumente prima naziv kompresirane datoteke, vrijeme koje je potrebno za kompresiju, i originalni nekompresirani niz. Funkcija *printaj()* ispiše u terminal veličinu kompresirane datoteke, vrijeme potrebno za kompresiju, omjer veličine kompresirane datoteke i originalnog nekompresiranog niza, te razliku u veličini između originalne slike i kompresirane datoteke. Za pristupanju veličini datoteka koristila se funkcija *os.path.getsize()*. Pri printanju, kako bi rezultati bili čitljiviji, koristila se funkcija *round()* kao i dijeljenje veličina s 1024 kako bi se povećala mjerna jedinica iz B u KB.

```
######### HILBERT GZIP ##########
start = time.time()
with gzip.open(putanja+'\\gzip_hilbert.txt.gz', 'wb') as f:
    f.write(string_hilbert.encode())
end =time.time()
current = end-start
printaj("gzip_hilbert.txt.gz",current,string_hilbert)
```

```
########## HILBERT 7Z ##########
start = time.time()
archive = py7zr.SevenZipFile(putanja+'\\7z_hilbert.7z', 'w')
archive.writeall(putanja+'\\hilbert_pic.txt', "hilbert_pic.txt")
archive.close()
end =time.time()
current = end-start
printaj("7z_hilbert.7z",current,string_hilbert)

########## HILBERT BZ2 ##########
start = time.time()
c = bz2.compress(string_hilbert.encode())
with bz2.open(putanja+"\\bz2_hilbert.bz2", "wb",compresslevel=9) as f:
    f.write(c)
end =time.time()
current = end-start
printaj("bz2_hilbert.bz2",current,string_hilbert)
print("_____________________________\n")
```

3.2.2. Rekonstruiranje slike iz kompresirane datoteke

Kod vraćanja slike iz kompresirane datoteke knjižnica *pillow* nudi funkciju *Image.frombytes()* koja stvara sliku s parametrima koje primi. Funkcija prima 3 osnova parametra: vrstu piksela koju će koristiti, dimenzije slike koju će rekonstruirati i podatke (bajtove koji će se zapisivati u sliku). Pošto je cijela knjižnica *pillow* napisana po obrnutom Kartezijevom koordinatnom sustavu, odnosno prima vrijednosti kao (x, y) no obrađuje ih kao (y, x)[12], tako i funkcija *Image.frombytes()* podatke koje dobiva tretira kao da su zapisani po stupcima. Stoga ta funkcija se jedino može iskoristiti za rekonstruiranje slike kod algoritma za iteraciju po stupcima. Podaci koji se šalju kao argument moraju biti prethodno pretvoreni u bajtove pomoću funkcije *bytes.fromhex().*

```
start = time.time()
with gzip.open(putanja+'\\gzip_columns.txt.gz', 'rb') as f:
    file_content = f.read()

file_content=file_content.decode()
data = bytes.fromhex(file_content)
img = Image.frombytes("RGB", (N,N), data)
img.save(putanja+"\\remake_columns.png",optimize=True)
end = time.time()
current = end - start
```

Kod ostalih algoritama za rekonstruiranje slike pristup je bio drugačiji. Jedan od problema koji nastaju kada se slika prebaci iz PNG oblika u tekstualnu datoteku (*.txt*)je da se gube sve dodatne informacije o toj slici kao i njezina rezolucija. Ali pošto su slike N x N dimenzija odnosno, sadržavaju N^2 piksela, a piksel je zapisan preko 6 heksadecimalnih vrijednosti (3 bajta), moguće je izračunati dimenziju N tako da se ukupna duljina niza podijeli sa 6 te se izvuče korijen iz tog rezultata. Nakon što je rezolucija poznata, pomoću funkcije *Image.new()* se napravi nova jednobojna slika. Funkcija kao parametre prima model piksela koji će se koristiti (RGB), veličinu dimenzija slike (N x N), te boju kojom će popuniti sliku (boja slike može biti bilo koja, u ovom slučaju je izabrana plava). Nakon što je napravljena nova „prazna“ slika u jednoj boji, učita se mapa piksela preko koje će se manipulirati vrijednosti piksela. Postupak stvaranja slike iz niza je sličan kao i postupak stvaranje niza iz slike. U ovom slučaju umjesto niza na kojeg se nadodaju vrijednosti piksela se pristupa mapi piksela preko koje se mijenja vrijednost trenutnog piksela. Varijabla *read* služi kao iterator kroz niz te se svakom iteracijom poveća za 6 jer je to veličina jednog piksela u nizu. Niz od 6 heksadecimalnih vrijednosti se pretvara u RGB zapis pomoću funkcije *hex_to_rgb()*. U ovome primjeru je naveden postupak za rekonstrukciju preko Hilbertove krivulje, no postupak je sličan i za ostale algoritme. Nakon što se izračuna originalna lokacija (x, y) za trenutnu iteraciju (i), očita se vrijednost piksela iz niza uz pomoć varijable *read*, pretvori u RGB, te se nadjača (*engl.* override) trenutna (početna) vrijednost piksela. Nakon što su svi pikseli nadjačani, slika se sprema uz pomoć funkcije *save()* koja nudi opciju *optimaze=True* koja dodatno smanji krajnju veličinu slike.

```
start = time.time()
with gzip.open(putanja+'\\gzip_hilbert.txt.gz', 'rb') as f:
    file_content = f.read()
file_content = file_content.decode()
N = int((len(file_content)/6)**0.5)

input = Image.new(mode="RGB",size=(N, N),
                        color="blue")
input.save("input", format="png")
pixel_map = input.load()
i=0

read = 0
while i < ((N*N)):
    curr = hildex2xy(i,N)
    pixel_map[curr[0],curr[1]] = hex_to_rgb(file_content[read:read+6])
    i+=1
    read+=6

input.save(putanja+"\\remake_hilbert.png",optimize=True)
end = time.time()
current = end - start
```

```
def hex_to_rgb(hex):
    return tuple(int(hex[i:i+2], 16) for i in (0, 2, 4))
```

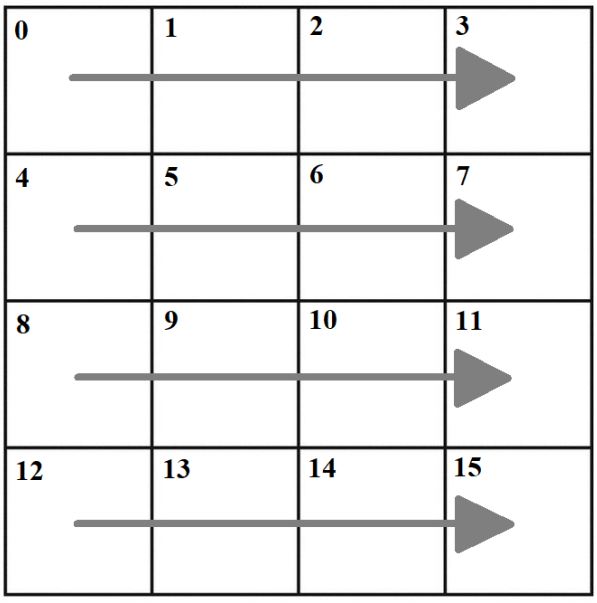
3.3 Jednostavni algoritmi stvaranja niza

Algoritmi stvaranja niza su podijeljeni na dvije kategorije: jednostavni algoritmi stvaranja niza i složeni algoritmi stvaranja niza. Jednostavni algoritmi stvaranja niza se smatraju oni koji ne ovise o dimenzijama slike. Implementirana su dva jednostavna algoritma stvaranja niza: algoritam iteriranja po redcima i algoritam iteriranja po stupcima.

3.3.1. Algoritam iteriranja po redcima

Ovaj algoritam je dobio naziv po tome što iterira po redcima preko duple *for* petlje. Prikazano na slici 3.1, kada završi iteracija prvog reda u zadnjem stupcu prelazi se na sljedeći red i prvi stupac.

```
string_rows = ""
for i in range(N):
    for j in range(N):
        string_rows += rgb_to_hex(px[i, j])
```



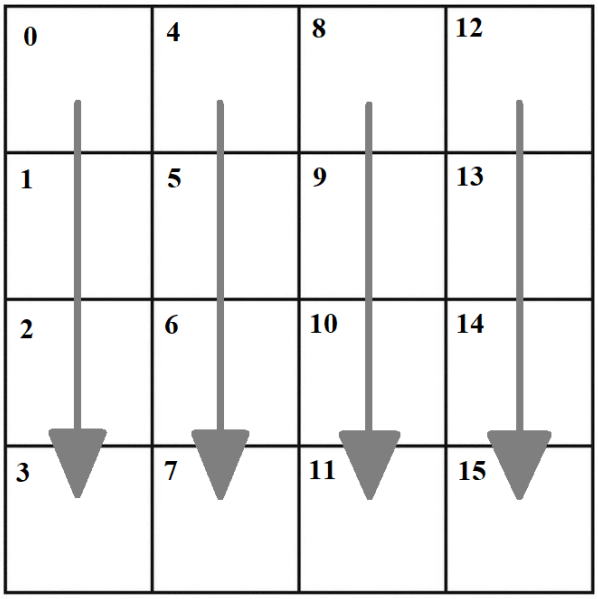
Slika 3.1. Algoritam iteracije po redovima

3.3.2. Algoritam iteriranja po stupcima

Ovaj algoritam je dobio naziv po tome što iterira po stupcima preko duple *for* petlje. Prikazano na slici 3.2, kada završi iteracija prvog stupca u zadnjem redu prelazi se na sljedeći stupac i prvi red.

```
string_columns = ""
for i in range(N):
    for j in range(N):
        string_columns += rgb_to_hex(px[j, i])
```

Ovaj algoritam je dosta sličan algoritmu iteriranja po redcima, jedina razlika između njih je zamjena x i y koordinata. Iako se ovo moglo napraviti pomoću jedne duple for petlje, radi preciznijeg mjerenja vremena su implementirani odvojeno.



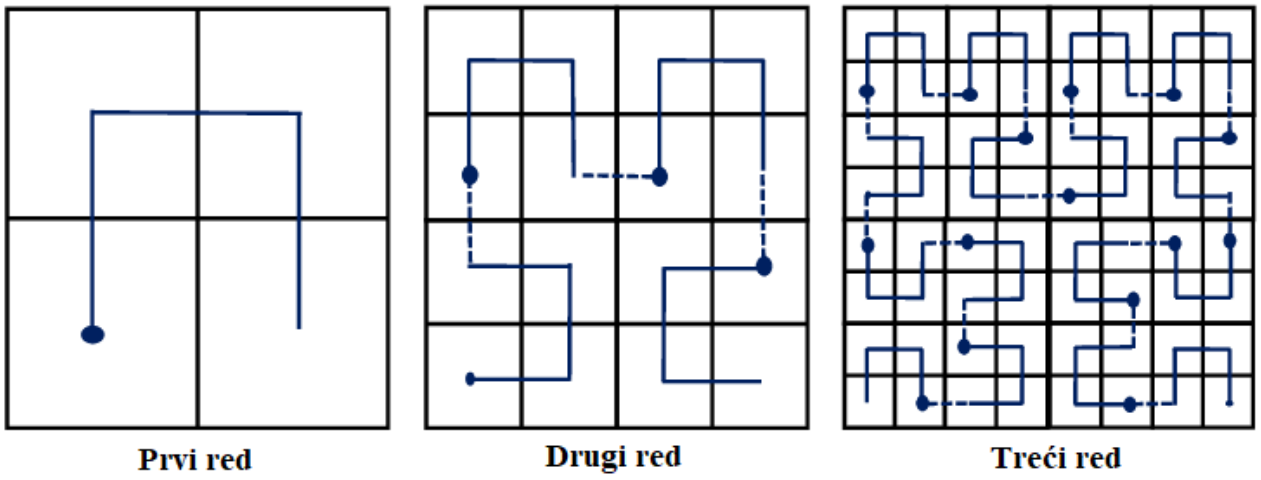
Slika 3.2. Algoritam iteracije po stupcima

3.4 Složeni algoritmi stvaranja niza

Složeni algoritmi stvaranja niza se smatraju oni koji ovise o dimenzijama slike na koju se primjenjuju. Implementirana su tri složena algoritma stvaranja niza: Hilbertova krivulja, Z krivulja te algoritam iteracije po dijagonali.

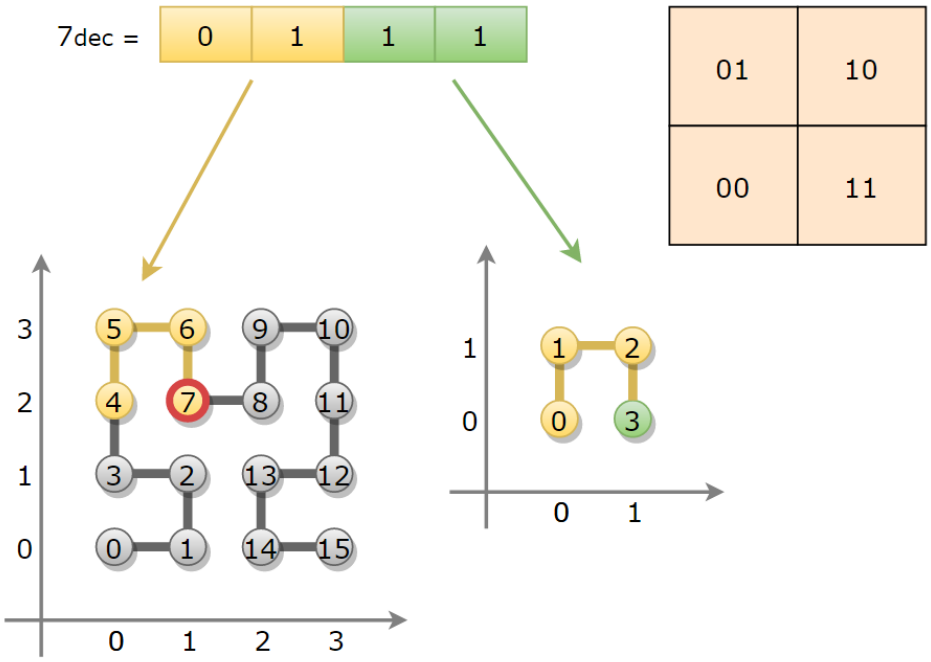
3.4.1 Hilbertova krivulja

Hilbertova krivulja (*engl.* Hilbert space-filling curve) je kontinuirana fraktalna krivulja koja popunjavanja prostor. Prvi ju je opisao njemački matematičar David Hilbert 1891. godine[13], kao varijantu Peano-vih krivulja (*engl.* Peano curves) koje također popunjavaju prostor. Peano-vu krivulju je otkrio Giuseppe Peano 1890. godine[14]. Za svakih 2^N točaka u prostoru Hilbertova krivulja popuni jedinični kvadrat. Duljina krivulje raste eksponencijalno s N te je osmišljena tako da će susjedne točke biti međusobno blizu. Hilbertova krivulja je opisana po Kartezijevom koordinatnom sustavu iako je u programiranju, zbog jednostavnosti, os y zrcaljena u odnosu na os x. Radi jednostavnijeg razumijevanja Hilbertove krivulje logika će biti objašnjena po klasičnom Kartezijevom sustavu. Početak krivulje se nalazi u donjem lijevom kutu dok završetak je u donjem desnom kutu. Na slici 3.3. prikazana su prva 3 reda Hilbertove krivulje.



Slika 3.3. Prva 3 reda Hilbertove krivulje[15]

Algoritam ne koristi rekurziju već koristi običnu iteraciju, iterirajući od bitova najmanje vrijednosti do bitova najviše vrijednosti (*engl.* Bottom-up approach). Kod binarne reprezentacije točke, zadnja 2 bita (2 bita najmanje vrijednosti) predstavljaju točku iteracije Hilbertove krivulje prvog reda (N = 2). Za početnu poziciju se očitaju koordinate točke iteracije Hilbertove krivulje u Kartezijevom koordinatnom sustavu. Sljedeća dva bita u „*Bottom-up* pristupu“ predstavljaju kvadrant u kojem je pozicionirana Hilbertova krivulja prvog reda unutar krivulje višeg reda (N = 4). Na slici 3.4 je prikazano kako se za N = 4, pronađu koordinate za 7. točku Hilbertove krivulje (i = 7).

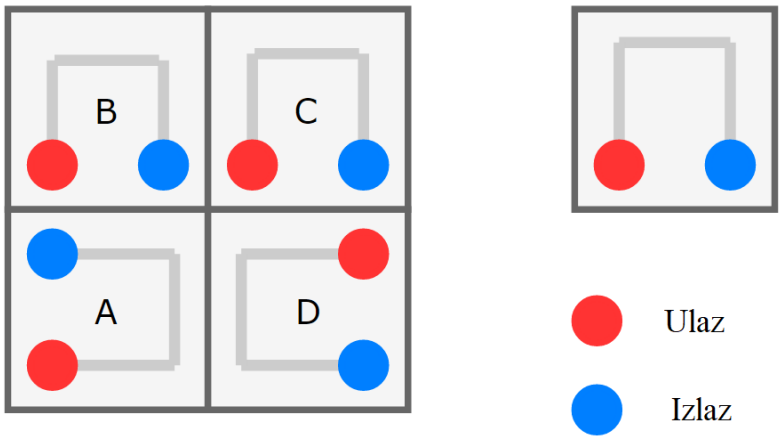


Slika 3.4. Hilbertova krivulja za N = 4, i = 7[16]

Broj 7 u binarnome iznosi $0111_{(2)}$. Zadnja dva bita $0111_{(2)}$ su $11_{(2)}$. Ta dva bita odgovaraju zadnjoj iteraciji u Hilbertovoj krivulji prvog reda za N = 2 (označeno zeleno na slici 3.4. desno). Koordinate te točke gledane u Kartezijevom koordinatnom sustavu jesu (x = 1, y = 0), te koordinate označuju početnu poziciju. Kada je očitana početna pozicija, svaka sljedeća dva bita služe za pozicioniranje unutar krivulje viših redaova. Sljedeća dva bita $0111_{(2)}$ su $01_{(2)}$. Ta kombinacija bitova odgovara prvom kvadratnu (brojač kreće od nule) Hilbertove krivulje višeg reda, tj. gornjem lijevom kvadrantu (označeno žuto na slici 3.4. lijevo). S obzirom na to da je N = 4, znači da se radi o Hilbertovoj krivulji drugog reda. Gledajući sliku 3.4. lijevo, može se uočiti kako je žutim označen

kvadrant Hilbertove krivulje prvog reda, pomaknut za N = 2 po y osi. Kako bi se dobile njegove koordinate, potrebno je trenutnoj poziciji (x = 1, y = 0) zbrojiti pomak od (0, 2), 0 jer nema pomaka po x osi, a 2 jer je pomak po y osi. Time se dobiju koordinate točke (x = 1, y = 2) za 7. iteraciju unutar Hilbertove krivulje N = 4.

Ovaj navedeni primjer je bio isključivo za N = 4, odnosno krivulju drugog reda. Ako se žele pronaći koordinate točke u krivulji višeg reda, N = 2K, gdje su poznate koordinate točke za N = K krivulju, koristeći „*bottom-up* pristup“, uvijek postoje 4 moguća slučaja (nakon što je pronađena početna pozicija točke). To je prikazano na slici 3.5 te opisano kodom *while* petlje u nastavku ovog poglavlja.



Slika 3.5. Pravila za određivanje pozicije kvadranta[16]

Početna pozicija se određuje po Kartezijevom koordinatnom sustavu. Zadnja 2 bita, tj. 2 bita najmanje vrijednosti će odrediti početnu poziciju točke unutar Hilbertove krivulje prvog reda. Određuje se tako da vrijednost zadnja 2 bita odgovara točki iteracije u prvoj Hilbertovoj krivulji, te se pozicija te točke očita iz Kartezijevog koordinatnog sustava. Npr. za $10_{(2)}$ = (x = 1, y = 1) jer se druga iteracija, krećući brojanje od nule, nalazi u Kartezijevom koordinatnom sustavu na poziciji (1, 1).

```
def last2bits(x):
    return x & 3

def hildex2xy(index, N):
    # N = 2
    positions = [[0,0], #    0,0   1,0
                 [0,1],  #     |     |
                 [1,1],  #    0,1 - 1,1
                 [1,0]]  #
    tmp = positions[last2bits(index)]
    index = (index >> 2)
    x = tmp[0]
    y = tmp[1]
    n = 4
    while n <= N:
        n2 = n/2

        if last2bits(index) == 0: # A
            old_x = x
            x = y
            y = old_x
        elif last2bits(index) == 1:     # B
            x = x
            y = y + n2
        elif last2bits(index) == 2:     # C
            x = x + n2
            y = y + n2
        else: # last2bits(index) == 3: # D
            old_y = y
            y = (n2-1) - x
            x = (n2-1) - old_y
            x = x + n2
        index = index >> 2
        n = n*2
    return [x,y]
```

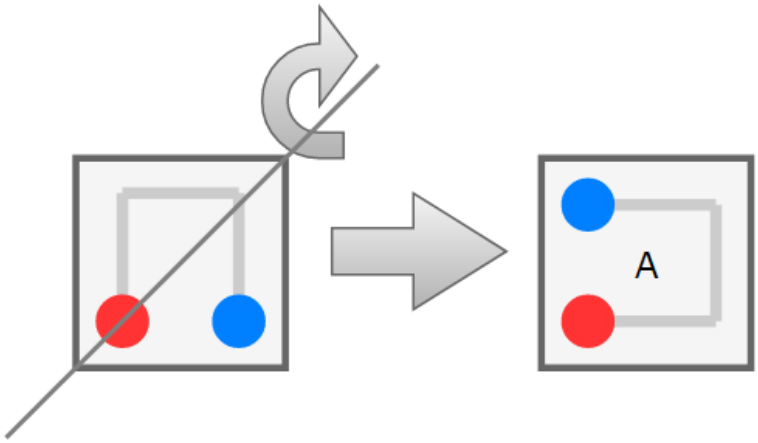
Nakon što se pronađe početna pozicija točke, određivanje pozicije točke u višim redovima se određuje iteracijom po brojevima potencije broja 2 sve dok taj broj ne bude jednak dimenziji slike N. Varijabla *n* određuje Hilbertovu krivulju $\log_2 n$ reda, dok varijabla *n2* određuje Hilbertovu krivulju jednog reda manje. Tj. Hilbertova krivulja N = 2K sastoji se od četiri N = K krivulje, tj. od četiri kvadranta (A, B, C i D sa slike 3.5).

B i C kvadranti su jednostavni za izračunati jer N = 2K krivulja sadrži kopije N = K krivulje (slika 3.5. desno) koje su samo pomaknute po x i y osi. Koordinate u B kvadrantu za N = 2K krivulju se dobiju kao koordinate za N = K krivulju uvećane za (0, K) tj. (x + 0, y + K), dok koordinate u C kvadrantu za N = 2K krivulju se dobiju kao koordinate za N = K krivulju uvećane za (K, K) tj. (x + K , y + K).

```
        elif last2bits(index) == 1:     # B
            x = x
            y = y + n2
        elif last2bits(index) == 2:     # C
            x = x + n2
            y = y + n2
```

Za A i D kvadrante postupak je nešto drugačiji, potrebno je transformirati N = K krivulju. Kod A kvadranta, krivulja N = K poklapa se s prvom točkom krivulje N = 2K, te kraj N = K krivulje bi trebao završiti tamo gdje počinje B kvadrant. Kako je to prikazano na slici 3.6 gdje je početak označen crvenom bojom, a izlaz plavom bojom. Ovo je moguće napraviti okrećući koordinate oko dijagonale, odnosno N = K (x, y) je u N = 2K (y, x).

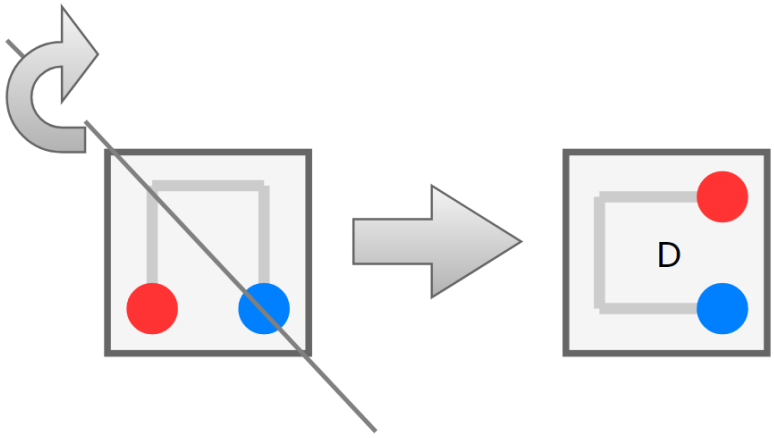
```
        if last2bits(index) == 0: # A
            old_x = x
            x = y
            y = old_x
```



Slika 3.6. Transformacija A kvadranta[16]

D kvadrant je sličan postupak, jedino se krivulja N = K okreće po sporednoj dijagonali. Čime se koordinate krivulje dobiju u dva koraka. Okretanjem kvadranta po sporednoj dijagonali te uvećanjem x koordinate za K. Radi lakšeg razumijevanja okretanje po sporednoj dijagonali je skicirano na slici 3.7.

```
        else: # last2bits(index) == 3:

            old_y = y
            y = (n2-1) - x
            x = (n2-1) - old_y
            x = x + n2
```

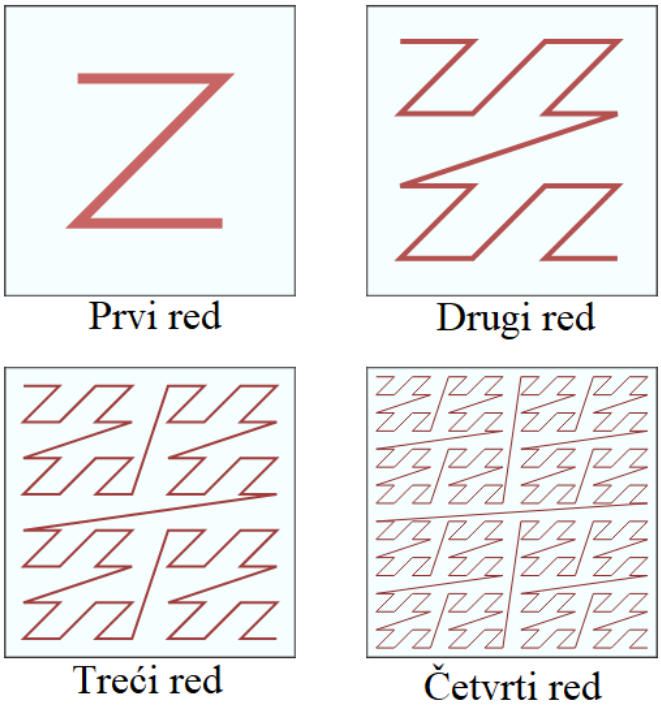


Slika 3.7. Transformacija D kvadranta[16]

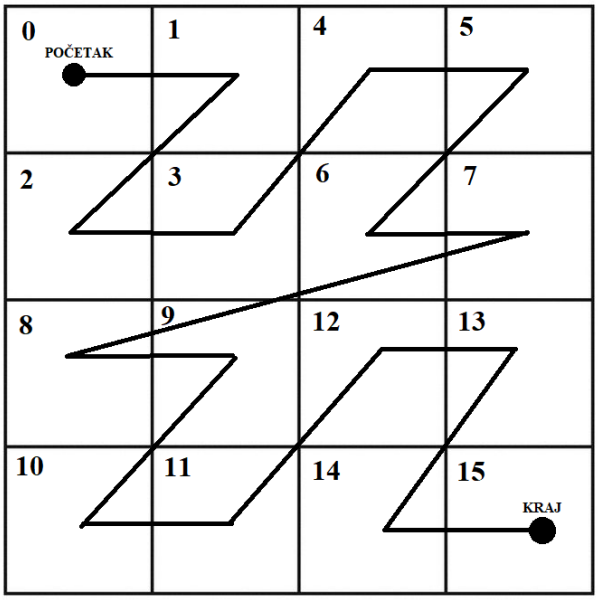
3.4.2. Z krivulja

Z krivulja (*engl.* Z-order curve) je također jedna od krivulja koje popunjavaju prostor (*engl.* space-filling-curve). Z krivulja je opisana po Kartezijevom koordinatnom sustavu. Početak krivulje se nalazi u gornjem lijevom kutu, a završetak u donjem desnom kutu. Na slici 3.8. prikazana su prva 4 reda Z krivulje. Četvrti red se sastoji od 4 treća reda Z krivulje, treći red se sastoji od 4 druga reda Z krivulje te drugi red se sastoji od 4 prva reda Z krivulje. Iako Z krivulja ima sličnosti s Hilbertovom krivuljom ona je dosta jednostavnija od Hilbertove krivulje. Prvi razlog tome je što kod prelaženja iz N = K Z krivulje u N = 2K Z krivulju nije potrebno okretanje kvadranata oko dijagonale već samo njihov pomak po Kartezijevom koordinatnom sustavu. Z krivulju je moguće iskoristiti samo na

slikama koje su N x N rezolucije. Za primjer njene iteracije kroz kvadratno polje (sliku) dana je slika 3.9 na kojoj je polje veličine 4 x 4. Algoritam je dobio naziv po tome što ispunjava prostor u obliku slova z, te svaki sljedeći red Z krivulje će popuniti prostor u obliku slova z.



Slika 3.8. Prva 4 reda Z krivulje[17]



Slika 3.9. Algoritam Z krivulje

Programski kod je sličan kodu za Hilbertovu krivulju s par izmjena pomaka. Početna pozicija se određuje po Kartezijevom koordinatnom sustavu. Kod binarne reprezentacije točke, zadnja 2 bita (2 bita najmanje vrijednosti) predstavljaju točku iteracije Z krivulje prvog reda. Za početnu poziciju se očitaju koordinate točke iteracije Z krivulje u Kartezijevom koordinatnom sustavu. Npr. ako su zadnja 2 bita $01_{(2)}$ to je jednako vrijednosti 1. Ako se pristupi polju *positions*[1] može se vidjeti da polje vrati vrijednost [1,0] što je jednako poziciji u drugoj iteraciji Z krivulje.

```
def ztoxy(index, N):

    positions = [[0,0], #     0,0 ---- 1,0
                 [1,0],  #           /
                 [0,1],  #         /
                 [1,1]]  #     0,1 ---- 1,1
    tmp = positions[last2bits(index)]
    index = (index >> 2)
    x = tmp[0]
    y = tmp[1]
```

Kod nalaženja koordinata Z krivulja višeg reda postoje 4 moguća slučaja: $00_{(2)}$, $01_{(2)}$, $10_{(2)}$ i $11_{(2)}$. Kod prvog slučaja, vrijednost kada su zadnja dva bita $00_{(2)}$, točka ostaje na mjestu. Kod slučaja kada su zadnja dva bita $01_{(2)}$ pomak se odvija po x osi za *n2*, tj. za dimenziju N prošlog reda krivulje.

```
    while n <= N:
        n2 = n/2
        if last2bits(index) == 1:
            x = x + n2
            y = y
```

U trećem slučaju kada su dva zadnja bita $10_{(2)}$, onda se pomak odvija po y osi za *n2*.

```
        elif last2bits(index) == 2:
            x = x
            y = y + n2
```

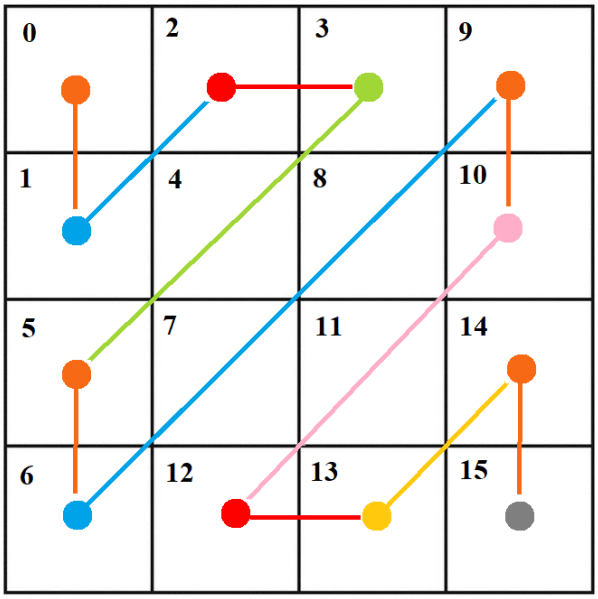
U posljednjem slučaju, kada su zadnja 2 bita $11_{(2)}$, pomak se odvija i po y i po x osi za *n2*. Kada se ispuni jedan od ova četiri slučaja, bitovi broja iteracije se pomaknu (*engl.* shift) za dva u desno kako bi se pronašao kvadrant iduće reda Z krivulje. Uz to se pomnoži varijabla *n* s 2 kako bi se skalirali pomaci za novi red Z krivulje.

```
        elif last2bits(index) == 3:
            y = y + n2
            x = x + n2
        index = index >> 2
        n = n*2
    return [x,y]
```

3.4.3. Algoritam iteriranja po dijagonali

Ovaj algoritam u ovom projektu je napravljen isključivo za slike dimenzija N x N, iako ga je moguće napraviti za slike dimenzija N x M . Pomoću varijabli *i* i *j* se manipulira iteracija kroz polje piksela. Radi lakšeg inkrementa i dekrementa varijabli koristi se *while* petlja koja iterira s varijablom *iterator*, dok se varijable *i* i *j* povećavaju i smanjuju ovisno o uvjetima. Ovaj algoritam nije pisan po Kartezijevom koordinatnom sustavu već onom „programskom“, odnosno kod točke[i][j] varijabla *i* pristupa broju retka, a varijabla *j* pristupa broju stupca. Programski kod se sastoji od 6 glavnih uvjeta. Prvi od uvjeta je kada je *j* na prvom ili zadnjem stupcu, krivulja će ići dolje samo ako broj retka djeljiv s 2 bez ostatka, označeno na slici 3.10. narančastom bojom. Drugi uvjet krivulje je ako je *j* u prvome stupcu te ako broj retka je djeljiv s 2 s ostatkom, onda krivulja ide dijagonalno desno gore, što je označeno na slici 3.10. plavom bojom. Treći uvjet je ako je *i* na zadnjem retku te broj stupca je djeljiv s 2 bez ostatka, onda krivulja također ide dijagonalno desno gore, označeno na slici 3.10. žutom bojom. Četvrti uvjet je zadovoljen ako je *i* na prvom ili zadnjem retku te ako je broj trenutačnog stupca djeljiv s 2 s ostatkom onda krivulja ide desno, što je označeno crvenom bojom na slici 3.10. Zadnja dva uvjeta su za pomak dijagonalno u smjeru dolje lijevo, jedan od uvjeta će biti ispunjen ako je *i* na prvome retku i broj stupca je djeljiv s 2 bez ostatka, to je označeno zelenom bojom na slici 3.10. Drugi uvjet kretanja dijagonalno u smjeru dolje lijevo je zadovoljen kada je *j* na zadnjem stupcu i kada je broj retka djeljiv s 2 s ostatkom, označeno na slici 3.10. rozom bojom.

```
iterator = 0
i=0
j=0
string_diagonally = ""
start = time.time()
while (iterator < N*N ):
    if (j == 0 or j == N-1) and (i%2 == 0):     # dolje
        string_diagonally += rgb_to_hex(px[i, j])
        i += 1
        iterator += 1

    elif((j == 0) and (i%2 == 1)):     # desno gore
        while i > 0:
            string_diagonally += rgb_to_hex(px[i, j])
            i-=1
            j+=1
            iterator += 1

    elif (i==N-1 and j%2 == 0):         # desno gore
        while j < N-1:
            string_diagonally += rgb_to_hex(px[i, j])
            i-=1
            j+=1
            iterator += 1

    elif (i == 0 or i == N-1) and (j%2 == 1):     # desno
        string_diagonally += rgb_to_hex(px[i, j])
        j += 1
        iterator += 1

    elif (i == 0) and (j%2 == 0):     # dolje lijevo
        while j > 0:
            string_diagonally += rgb_to_hex(px[i, j])
            i+=1
            j-=1
            iterator += 1

    elif (j == N-1) and (i%2 == 1):     # dolje lijevo
        while (i < N-1):
            string_diagonally += rgb_to_hex(px[i, j])
            j-=1
            i+=1
            iterator += 1
end = time.time()
current = end - start
```



Slika 3.10. Algoritam iteracije po dijagonali

3.5 Algoritmi kompresije

U ovome poglavlju će biti opisani algoritmi kompresije koji se koriste u Zlib-u, 7zip-u i Bz2. U teoriji informacija cilj je prenijeti što više informacija kroz što manje bitova tako da ni jedno kodiranje nije dvosmisleno. Npr. kada bi se kodirala slova A, B, C i D u što manje bitova, svako slovo bi se moglo kodirati kao $1_{(2)}$, no to bi onda bilo dvosmisleno jer poruka $1111_{(2)}$ bi mogla znači „ABCD“ ili „BBAA“. Kodiranje može biti fiksne duljine (*engl.* fixed-length encoding) ili promjenjive duljine (*engl*. variable-length encoding). Kod fiksne duljine svaki simbol ima jednak broj bitova dok kod kodiranja promjenjive duljine neki simboli mogu imati različitu duljinu bitova.

3.5.1. Zlib

Zlib alat koristi algoritam Deflate. Deflate je kombinacija Huffmanovog koda (*engl*. Huffman code) i LZ77 kompresije. Huffmanov kod je efikasan način kompresije podataka bez gubitka informacije. Kodiranje se odvija po frekvenciji pojavljivanja određenih znakova u poruci. Ne postoji Huffmanov kod koji će raditi za sve poruke , tj. Huffmanov kod za slanje poruke A će vjerojatno biti drugačiji od onog za slanje poruke B. Huffmanov kod koristi kodiranje bitova promjenjive duljine te predstavlja simbole u ovisnosti koliko često se ponavljaju. Simboli koji se češće pojavljuju trebali bi

se sastajati od manje bitova dok simboli koji se pojavljuju rjeđe se mogu sastojati od više bitova. Na ovaj način kodiranje određene poruke će potencijalno biti kraće u odnosu na korištenje kodiranja fiksne duljine bitova.

Huffmanov algoritam se sastoji od toga da se uzme lista simbola i njihova šansa pojavljivanja. Odaberu se dva simbola koji imaju najmanju šansu pojavljivanja, te se napravi stablo od ta dva simbola. Jedna grana ima vrijednost $1_{(2)}$ druga $0_{(2)}$, raspored grana nije bitan sve dok se u ostatku stabla drži pravilo (ako je prva grana lijevo $1_{(2)}$ onda su sve ostale grane lijevo $1_{(2)}$. Zbroje se vjerojatnosti oba simbola da se dobije vjerojatnost tog podstabla. Onda se ta dva simbola maknu s popisa (liste) simbola i doda se podstablo na popis. Ponovno se prolazi kroz listu i uzimaju se 2 simbola ili podstabla s najnižim vjerojatnostima i oni tvore novo podstablo. Simboli ili podstabla od kojeg je nastalo novo stablo se brišu s popisa i dodaje se novo podstablo na popis. Proces se ponavlja dok svi elementi nisu povezani. Ovaj pristup je dobar kodiranju ako se radi o kodiranju pojedinačnih znakova , ali za kodiranje više znakova postoje bolje metode kodiranja[18].

LZ77 kompresija radi tako da pronađe isječke podataka koji se ponavljaju. Koristi se klizni prostor (*engl.* sliding window) koji sadrži podatak o tome koji se znak koliko puta pojavljivao u zadnjih x znakova. Klizni prostor od 32K znači da kompresor i dekompresor imaju podatak o zadnjih 32768 znakova. Ako se neki niz znakova koji će biti kompresiran već pojavio ponovo unutar kliznog prozora, onda se oba niza zamijene s dva broja: udaljenost od kliznog prozora i veličina (duljina znakovnog niza)[19].

Deflate kompresor ima dosta fleksibilnosti na koje sve načine da kompresira podatke. Prvi od tri modaliteta je da ne kompresira podatke uopće , to se koristi kod podataka koji su već kompresirani. Ukupna veličina datoteke u ovom modalitetu će se malo povećati ali bi se povećala više da su se probali kompresirati već kompresirani podaci. Drugi modalitet koristi prvo kompresiju s LZ77 pa onda Huffmanov kod. Stabala koja se kompresiraju u ovome modalitetu su definirana specifikacijom Deflate-a pa ne zauzimaju ništa dodatnog prostora. Te treći modalitet koji također prvo radi kompresiju uz pomoć LZ77 te zatim Huffmanovo kodiranje sa stablima koje kompresor

stvara i pohranjuje zajedno sa podacima. Podaci su podijeljeni u blokove, te svaki blok može koristiti bilo koji od modula, no izmjena modula nije moguća dok se blok ne zatvori[19].

3.5.2. 7zip

Iako 7zip (7z) podržava razne algoritme kompresije poput LZMA, LZMA2, PPMD, BCJ, BCJ2, Bzip2 i Deflate u ovome projektu je korišten zadani (*engl.* default) kompresor LZMA. Lempel-Ziv-Markov lančani algoritam (*engl.* Lempel-Ziv-Markov chain algorithm (LZMA)) je algoritam koji se koristi kako bi se napravila kompresija podataka bez gubitaka. Algoritam koristi kompresiju uz pomoć rječnika, slično kao algoritam LZ77 (koji je opisan u prethodnom poglavlju) te nudi veliki omjer kompresije (inače veći omjer kompresije od Bzip2[20]).

3.5.3. Bzip2

Bizp2 (bz2) je besplatan program otvorenog koda koji kompresira pojedine datoteke koristeći Burrow-Wheeler algoritam. Bzip2 koristi nekoliko različitih slojeva kompresijskih tehnologija nadovezanih jednu na drugu. Ti slojevi dolaze u određenom poretku prilikom kompresije i u obrnutom poretku za vrijeme dekompresije. Prvi korak je *Run-length encoding (RLE),* to je kompresija bez gubitka podataka gdje se nizovi podataka pohranjuju kao jedna vrijednost. Drugi korak je Burrows-Wheeler transformacija koja preuređuje niz znakova tako da se isti znakovi zapišu jedan za drugim. Ovo je dobro za kompresiju budući da se lako kompresiraju nizovi ponovljenih znakova. Ovaj postupak se može „okrenuti“ bez ikakvih dodanih podataka osim prvog zapisanog znaka. Sljedeći korak je transformacija pomicanja prema naprijed (*engl*. move-to-front (MTF) transformation). To je kodiranje podataka u namjeri da poboljšaju performanse entropijskih kodiranja kompresije. Nakon MTF transformacije ponovno dolazi na red RLE, koji se u sljedećem koraku kodira pomoću Huffmanovog koda. Sljedeći korak je odabir Huffmanovih tablica koje su popraćene kodiranjem pomoću *Unary base-1*. Delta kodiranje duljine bitova (*engl.* Delta encoding) Huffmanovog koda je osmi korak. Te kao zadnji korak u postupku je raščlanjenje niza bitova simbola koji se koriste. Postupak za vraćanje iz kompresije je obrnut[21].

4. REZULTATI

Za svaku odabranu sliku testiranja, potrebno je izvući pojedine informacije iz izlaza i obraditi ih. Obrada rezultata se može podijeliti u 3 kategorije. Prva kategorija su rezultati pretvaranja slike u niz, gdje se gleda veličina niza i vrijeme potrebno za zapis cijelog niza. Druga kategorija jesu rezultati kompresije niza, odnosno tu se gleda veličina kompresiranog niza i vrijeme potrebno za njegovu kompresiju. Također se računa omjer veličine niza i kompresije te omjer razlike u veličini kompresiranog niza i originalne veličine slike. Omjer veličine niza i kompresije se računa kao *1 – (veličina kompresije / veličina originalnog niza)*, dok se omjer razlike u veličini kompresiranog niza i originalne veličine slike računa kao *((veličina kompresije – veličina originalne slike) / veličina originalne slike) * 100.* U slučaju da je veličina kompresije manja od veličine originalne slike onda će omjer doći negativan što označava da je kompresija bila uspješna. Veličina kompresije umanjena za veličinu originalne slike je izražena u stupcu *+ / - KB u veličini kompresije.* U trećoj kategoriji se gledaju rezultati rekonstruiranja slike, veličina rekonstruirane slike i vrijeme potrebno da se rekonstruira slika.

4.1 Rezultati pretvaranja slike u niz

U tablici 4.1. možemo vidjeti nazive svih slika, njihove veličine, dimenzije i veličine slika u nizu . Veličina slike u nizu uvijek ovisi o dimenzijama slike. Pod stupcem *Veličina slike* navedena je originalna veličina slike.

Tablica 4.1. Osnovne informacije o slikama

Naziv slike	Veličina slike	Dimenzije slike	Veličina slike u nizu
ventilator.png	1838.09 KB	1024 x 1024	6144 KB
sunflower.png	1716.75 KB	1024 x 1024	6144 KB
peppers.png	526.12 KB	512 x 512	1536 KB
fruits.png	461.04 KB	512 x 512	1536 KB

U tablici 4.2 su zapisana vremena koja su potrebna pojedinom algoritmu da stvori niz određenim redoslijedom. Algoritam iteriranja po stupcima je najbrži za slike dimenzija 1024 x 1024, dok algoritam iteriranja po redcima je brži kod slika dimenzija 512 x 512, iako vremenska razlika između ta dva algoritma najmanja u odnosnu na vremenske razlike ostalih algoritama. Najduže izvršavanje za sve četiri slike zauzima Hilbertova krivulja, koja se čak 3 do 5 puta dulje izvršava od algoritama iteracije po stupcima, redcima i dijagonali.

Tablica 4.2. Vrijeme za stvaranje niza određenim algoritmom

Algoritam	**ventilator.png**	**sunflower.png**	**peppers.png**	**fruits.png**
Hilbert	5.5998 s	5.5735 s	1.1393 s	1.1407 s
Stupci	**1.6889 s**	**1.6246 s**	0.22 s	0.223 s
Redci	1.7003 s	1.6706 s	**0.218 s**	**0.2195 s**
Dijagonala	1.8454 s	1.8231 s	0.259 s	0.2674 s
Z	5.3647 s	5.3493 s	1.086 s	1.0796 s

4.2 Rezultati kompresije niza

Kompresije nizova napravljene su za svaku sliku zasebno, te su tako raspisani i rezultati. Rezultati za svaku sliku se sastoje od 15 setova kombinacija između alata za kompresiju i algoritama zapisivanja nizova (3 alata kompresije x 5 algoritama zapisivanja nizova). Naziv datoteke je napravljen kao *naziv kompresijskog alata* + *_naziv korištenog algoritma za zapisivanje niza* + *.nastavak koji koristi kompresijski niz.* U svakoj tablici vremena i omjera kompresije zapisano je vrijeme potrebno da se niz znakova kompresira, te koji omjer veličine između originalnog niza i kompresiranog niza. Kod tablica veličine i razlike u kompresiji zapisana je veličina kompresiranog niza, koliko je kompresirani niz veći (+) ili manji (-) u odnosu na originalnu sliku, te je ta razlika prikazana u postotku originalne slike. U tablicama su podebljani najbolji rezultati.

U tablici 4.3 je zapisano kompresijsko vrijeme i omjer slike *ventilator.png*. Može se uočiti kako algoritam Hilbertove krivulje u kombinaciji sa 7zip-om daje najveći omjer kompresije, dok algoritam iteracije po stupcima u kombinaciji s gzip-om je vremenski najbrži. U tablici 4.4 koja sadrži veličine kompresiranih nizova slike *ventilator.png* se vidi da jedine dvije kombinacije koje daju manju veličinu kompresiranog niza od originalne slike koriste alat 7zip. Najviše ušteđene računalne memorije nudi algoritam Hilbertove krivulje a odmah iza njega je algoritam Z krivulje. Kombinacija algoritma iteracije po redovima i kompresijskog alata bzip2 je najbolji „kompromis“ brzine kompresije i omjera kompresije, 150ms je sporiji od najbrže kombinacije a 10% ima bolji omjer kompresije. A ako se usporedi s kombinacijom koja ima najbolji omjer kompresije, onda ima 2.5% lošiji omjer kompresije ali je zato 5 puta brži.

Tablica 4.3. Vrijeme i omjer kompresije ventilator.png

Datoteka	Vrijeme	Omjer veličine niza i kompresije
gzip_hilbert.txt.gz	0.5715 s	0.6195
7z_hilbert.7z	2.3537 s	**0.7166**
bz2_hilbert.bz2	0.5864 s	0.6894
gzip_columns.txt.gz	**0.4155 s**	0.5897
7z_columns.7z	2.6061 s	0.6986
bz2_columns.bz2	0.6075 s	0.6857
gzip_rows.txt.gz	0.4285 s	0.5914
7z_rows.7z	2.3826 s	0.7005
bz2_rows.bz2	0.5860 s	0.6902
gzip_diagonally.txt.gz	0.4413 s	0.583
7z_diagonally.7z	2.5920 s	0.6851
bz2_diagonally.bz2	0.5974 s	0.6753
gzip_z.txt.gz	0.5565 s	0.6112
7z_z.7z	2.3701 s	0.7082
bz2_z.bz2	0.5920 s	0.6756

Tablica 4.4. Veličina i razlika u kompresiji ventilator.png

Datoteka	Veličina	+ / - KB u veličini kompresije	+ / - % u veličini kompresije
gzip_hilbert.txt.gz	2337.89 KB	499.80 KB	27.19 %
7z_hilbert.7z	**1741.18 KB**	**-96.90 KB**	**-5.23 %**
bz2_hilbert.bz2	1908.50 KB	70.41 KB	3.83 %
gzip_columns.txt.gz	2521.04 KB	682.96 KB	37.16 %
7z_columns.7z	1851.99 KB	13.91 KB	0.76 %
bz2_columns.bz2	1931.23 KB	93.14 KB	5.07 %
gzip_rows.txt.gz	2510.52 KB	672.44 KB	36.58 %
7z_rows.7z	1839.96 KB	1.87 KB	0.10 %
bz2_rows.bz2	1903.46 KB	65.37 KB	3.56 %
gzip_diagonally.txt.gz	2561.99 KB	723.90 KB	39.38 %
7z_diagonally.7z	1934.83 KB	96.74 KB	5.26 %
bz2_diagonally.bz2	1995.17 KB	157.08 KB	8.55 %
gzip_z.txt.gz	2388.70 KB	550.61 KB	29.96 %
7z_z.7z	1792.59 KB	-45.5 KB	-2.48 %
bz2_z.bz2	1992.85 KB	154.77 KB	8.42 %

Za sliku *sunflower.png* najbolje vrijeme kompresije nudi algoritam iteracije po redcima u kombinaciji s gzip-om, a za najveći omjer kompresije je najbolja kombinacija algoritama Hilbertove krivulje s alatom 7zip kako je navedeno u tablici 4.5. Po tablici 4.6. može se uočiti da za ovu sliku kompresija nije imala nikakvog pozitivnog utjecaja na veličinu. Kombinacija s najmanjom veličinom kompresije je veća od originalne slike za 0.20% (3KB). Algoritam iteriranja po dijagonali u kombinaciji s gzip-om je najlošiji za ovu sliku što se tiče kompresije podataka, jer vrati kompresiranu datoteku koja 49.28% veća od originalne slike.

Tablica 4.5. Vrijeme i omjer kompresije sunflower.png

Datoteka	**Vrijeme**	**Omjer veličine niza i kompresije**
gzip_hilbert.txt.gz	0.5445 s	0.6253
7z_hilbert.7z	2.0409 s	**0.72**
bz2_hilbert.bz2	0.5965 s	0.6688
gzip_columns.txt.gz	0.3506 s	0.5895
7z_columns.7z	2.1845 s	0.694
bz2_columns.bz2	0.6057 s	0.6676
gzip_rows.txt.gz	**0.3309 s**	0.59
7z_rows.7z	2.2441 s	0.7005
bz2_rows.bz2	0.6115 s	0.6664
gzip_diagonally.txt.gz	0.399 s	0.5829
7z_diagonally.7z	2.2231 s	0.6804
bz2_diagonally.bz2	0.621 s	0.6577
gzip_z.txt.gz	0.532 s	0.6167
7z_z.7z	2.0631 s	0.7113
bz2_z.bz2	0.6005 s	0.6569

Tablica 4.6. Veličina i razlika u kompresiji sunflower.png

Datoteka	Veličina	+ / - KB u veličini kompresije	+ / - % u veličini kompresije
gzip_hilbert.txt.gz	2301.86 KB	585.10 KB	34.08 %
7z_hilbert.7z	**1720.12 KB**	**3.37 KB**	**0.20 %**
bz2_hilbert.bz2	2034.63 KB	317.88 KB	18.52 %
gzip_columns.txt.gz	2522.06 KB	805.31 KB	46.91 %
7z_columns.7z	1880.00 KB	163.25 KB	9.51 %
bz2_columns.bz2	2041.96 KB	325.21 KB	18.94 %
gzip_rows.txt.gz	2518.94 KB	802.19 KB	46.73 %
7z_rows.7z	1840.04 KB	123.29 KB	7.18 %
bz2_rows.bz2	2049.83 KB	333.08 KB	19.40 %
gzip_diagonally.txt.gz	2562.77 KB	846.02 KB	49.28 %
7z_diagonally.7z	1963.59 KB	246.83 KB	14.38 %
bz2_diagonally.bz2	2102.87 KB	386.12 KB	22.49 %
gzip_z.txt.gz	2355.12 KB	638.36 KB	37.18 %
7z_z.7z	1773.95 KB	57.20 KB	3.33 %
bz2_z.bz2	2108.22 KB	391.46 KB	22.80 %

Rezultati testne slike *pepper.png* u tablici 4.7 također imaju najbolji omjer kompresije kad se koristi alat 7zip u kombinaciji s algoritmom Hilbertove krivulje, dok najbrže izvršavanje ima kombinacija alata gzip i algoritma iteracije po stupcima. U tablici 4.8 se može vidjeti kako ni jedna kombinacija kompresije nije manja od originalne slike, najmanja kompresija je 10% veća od originalne slike.

Tablica 4.7. Vrijeme i omjer kompresije peppers.png

Datoteka	**Vrijeme**	**Omjer veličine niza i kompresije**
gzip_hilbert.txt.gz	0.1253 s	0.5181
7z_hilbert.7z	0.4197 s	**0.6216**
bz2_hilbert.bz2	0.1865 s	0.5596
gzip_columns.txt.gz	**0.107 s**	0.4925
7z_columns.7z	0.4487 s	0.5957
bz2_columns.bz2	0.1773 s	0.5574
gzip_rows.txt.gz	0.1205 s	0.4993
7z_rows.7z	0.4747 s	0.6029
bz2_rows.bz2	0.1745 s	0.5604
gzip_diagonally.txt.gz	0.1141 s	0.4937
7z_diagonally.7z	0.4629 s	0.5934
bz2_diagonally.bz2	0.1947 s	0.5569
gzip_z.txt.gz	0.1226 s	0.5133
7z_z.7z	0.4663 s	0.6148
bz2_z.bz2	0.1835 s	0.5558

Tablica 4.8. Veličina i razlika u kompresiji peppers.png

Datoteka	Veličina	+ / - KB u veličini kompresije	+ / - % u veličini kompresije
gzip_hilbert.txt.gz	740.26 KB	214.14 KB	40.70 %
7z_hilbert.7z	**581.25 KB**	**55.13 KB**	**10.48 %**
bz2_hilbert.bz2	676.44 KB	150.31 KB	28.57 %
gzip_columns.txt.gz	779.60 KB	253.47 KB	48.18 %
7z_columns.7z	620.93 KB	94.81 KB	18.20 %
bz2_columns.bz2	679.88 KB	153.76 KB	29.23 %
gzip_rows.txt.gz	769.04 KB	242.92 KB	46.17 %
7z_rows.7z	610.02 KB	83.90 KB	15.95 %
bz2_rows.bz2	675.25 KB	149.12 KB	28.34 %
gzip_diagonally.txt.gz	777.72 KB	251.60 KB	47.82 %
7z_diagonally.7z	624.54 KB	98.42 KB	18.71 %
bz2_diagonally.bz2	680.54 KB	154.42 KB	29.35 %
gzip_z.txt.gz	747.56 KB	221.43 KB	42.09 %
7z_z.7z	591.73 KB	65.60 KB	12.47 %
bz2_z.bz2	682.36 KB	156.23 KB	29.70 %

Rezultati za sliku *fruits.png* u tablici 4.9. može se reći da „prate trend“ prijašnjih rezultata kada je riječ o najboljoj kombinaciji za omjer kompresije (kombinacija algoritma Hilbertove krivulje i 7zip-a). Dok najbrže vrijeme kompresije ima kombinacija gzip-a i algoritma iteracije po redovima. U tablici 4.10 ponovno jedino dvije kombinacije kompresije imaju manju veličinu od originalne slike. Kombinacija algoritma Hilbertove krivulje s alatom 7zip daje najmanju veličinu kompresijske datoteke koja je 8.14% manja od originalne slike, odmah nakon nje je kombinacija 7zip-a i algoritma Z krivulje koja daje kompresijsku datoteku manju za 6.31% od originalne slike.

Tablica 4.9. Vrijeme i omjer kompresije fruits.png

Datoteka	**Vrijeme**	**Omjer veličine niza i kompresije**
gzip_hilbert.txt.gz	0.144 s	0.6308
7z_hilbert.7z	0.4359 s	**0.7243**
bz2_hilbert.bz2	0.1622 s	0.6848
gzip_columns.txt.gz	0.1217 s	0.6043
7z_columns.7z	0.4488 s	0.699
bz2_columns.bz2	0.1494 s	0.694
gzip_rows.txt.gz	**0.1119 s**	0.5967
7z_rows.7z	0.4937 s	0.6953
bz2_rows.bz2	0.151 s	0.6853
gzip_diagonally.txt.gz	0.1146 s	0.6009
7z_diagonally.7z	0.4674 s	0.6883
bz2_diagonally.bz2	0.2341 s	0.684
gzip_z.txt.gz	0.1399 s	0.6253
7z_z.7z	0.4279 s	0.7188
bz2_z.bz2	0.1561 s	0.687

Tablica 4.10. Veličina i razlika u kompresiji fruits.png

Datoteka	Veličina	+ / - KB u veličini kompresije	+ / - % u veličini kompresije
gzip_hilbert.txt.gz	567.09 KB	106.06 KB	23.00 %
7z_hilbert.7z	**423.50 KB**	**-37.54 KB**	**-8.14 %**
bz2_hilbert.bz2	484.21 KB	23.18 KB	5.03 %
gzip_columns.txt.gz	607.74 KB	146.70 KB	31.82 %
7z_columns.7z	462.33 KB	1.29 KB	0.28 %
bz2_columns.bz2	470.08 KB	9.05 KB	1.96 %
gzip_rows.txt.gz	619.46 KB	158.43 KB	34.36 %
7z_rows.7z	468.03 KB	7.00 KB	1.52 %
bz2_rows.bz2	483.42 KB	22.38 KB	4.86 %
gzip_diagonally.txt.gz	613.06 KB	152.03 KB	32.98 %
7z_diagonally.7z	478.83 KB	17.79 KB	3.86 %
bz2_diagonally.bz2	485.40 KB	24.37 KB	5.28 %
gzip_z.txt.gz	575.48 KB	114.44 KB	24.82 %
7z_z.7z	431.95 KB	-29.09 KB	-6.31 %
bz2_z.bz2	480.81 KB	19.77 KB	4.29 %

Bzip2 pokazuje najbolji „kompromis“ vremena kompresije i omjera kompresije. Općenito ima duže vrijeme kompresiranja od gzip-a, ali ima zato i veći omjer kompresije. A ako ga se usporedi sa 7zip-om onda ima malo lošiji omjer kompresije ali je zato 3 do 5 puta brži (3 do 5 puta manje vremena je potrebno da se niz kompresira).

4.3 Rezultati rekonstruiranja slike

U tablici 4.11. se mogu vidjeti rekonstruirane i originalne veličine testnih slika. Sve rekonstruirane slike su manje od originalnih slika zbog parametra *compress* u funkciji *.save()* koja dodatno kompresira sliku. Osim dodatne kompresije, manjoj veličini također pridonose dodatni podaci koje sadržava PNG oblik, a koji se izgube prilikom pretvaranja slike u niz.

Tablica 4.11. Veličine rekonstruiranih slika

Naziv slike	Veličina rekonstruirane slike	Veličina originalne slike	Omjer razlike
ventilator.png	1823.20 KB	1838.09 KB	-0.81%
sunflower.png	1623.43 KB	1716.75 KB	-5.44%
peppers.png	493.83 KB	526.12 KB	-6.13%
fruits.png	436.03 KB	461.04 KB	-5.42%

U tablici 4.12. prikazano je vrijeme potrebno da se slika rekonstruira pojedinim algoritmom. Algoritam iteracije po stupcima ima najkraće vrijeme izvršavanja jer se iskoristila gotova funkcija *.frombytes()*, koja prima jedino niz piksela zapisan u redoslijedu po stupcima. Veličina rekonstruirane slike ne ovisi o načinu po kojem se puni mapa piksela već o funkciji koja sprema tu mapu piksela, stoga svih 5 algoritama imaju jednaku veličinu rekonstruirane slike.

Tablica 4.12. Vrijeme potrebno za rekonstruiranje slike iz kompresiranog niza

Algoritam	ventilator.png	sunflower.png	peppers.png	fruits.png
Hilbert	6.0257 s	6.1808 s	1.3138 s	1.3555 s
stupci	**0.6006 s**	**0.8485 s**	**0.055 s**	**0.1045 s**
redci	1.9531 s	2.1866 s	0.3985 s	0.4505 s
dijagonala	2.148 s	2.4071 s	1.2942 s	0.4839 s
Z	5.8904 s	6.0398 s	0.4299 s	1.3831 s

5. ZAKLJUČAK

Ideja ovog rada bilo je saznati hoće li pretvaranje slike u niz te njeno kompresiranje imati utjecaj na njenu veličinu pohranjivanja. Testiranje se izvršilo na PNG slikama N x N dimenzija, 24 bitnih piksela. Cilj je bilo pretvoriti sliku u znakovni niz pomoću različitih algoritama te s nekoliko kompresijskih alata usporediti rezultate. Ovaj postupak pretvaranja slike u kompresirani niz te rekonstruiranje slike iz niza, zadržava rezoluciju PNG slike, no gube se dodatni podaci koje PNG oblik sprema. Kada se provuče slika kroz algoritam ne postaje ništa više od šifriranog niza znakova koji sadržava informaciju, te jedini ključ za dešifriranje tog niza je taj isti algoritam kako bi se informacija povratila. Najbrži algoritmi stvaranja niza pokazali su se iteracija po stupcima i iteracija po redcima jer nemaju nikakvih dodatnih izračunavanja za sljedeće točke u nizu već jedino pristup željenom elementu. Najbolji rezultat kompresije su davali algoritmi Hilbertove krivulje i Z krivulje, iako je Hilbertova krivulja imala bolje rezultate. Jedino ta dva algoritma su uspješno kompresirala slike u nizove koji su manji od veličine originalne slike. No, to nije bio slučaj za sve slike, za neke slike ni jedan algoritam nije uspio kompresirati niz ispod veličine originalne slike. Hilbertova krivulja i Z krivulja su krivulje koje popunjavaju prostor tako da točke (pikseli) koji su bliže međusobno na Kartezijevom koordinatnom sustavu će biti bliže i na krivulji (nizu). Na slikama su inače pikseli iste boje također blizu jedni drugome. Stoga, koristeći pristup Hilbertove krivulje ili Z krivulje, niz će sadržavati više piksela iste boje za redom nego algoritmi iteracije po redovima, stupcima i dijagonali. Alati kompresije rade tako da više istih znakova ili nizove znakova kompresiraju u sebi prepoznatljive kratice, a pikseli nisu ništa više od heksadecimalnih vrijednosti, stoga će Hilbertova krivulja biti više kompresirana od drugih algoritama. Jedini alat koji je uspio kompresirati nizove ispod originalne veličine slike je 7zip koji koristi LZMA algoritam. Kompresija slike je jako bitna kada imamo ograničene resurse te je svaka kompresija „kompromis“ između vremena potrebnog za kompresiju i kvalitete kompresije. Bzip2 kompresijski alat daje najbolji omjer brzine i kvalitete kompresije, gzip je najbrži alat za kompresiju, dok 7zip je najsporiji ali radi najkvalitetniju kompresiju. Ovaj projekt daje uvid u algoritme prostornih krivulja i postupka pretvaranja slike u niz. Nakon brojnih rezultata može se zaključiti da je kombinacija algoritma Hilbertove krivulje i 7zip alata najučinkovitija kompresija za slike.

6. LITERATURA

[1] „Data representation“, s Interneta https://www.bbc.co.uk/bitesize/guides/zfspfcw/revision/10 1.9.2022.

[2] Solomon C., Breckon T.: „Fundamentals of Digital Image Processing“, 2011.

[3] „Formati slike“, s Interneta https://guides.lib.umich.edu/c.php?g=282942&p=1885348 5.rujna 2022

[4] Pascale D.: „A review of RGB Color … from xyY to R'G'B'“, 2003.

[5] „Boje u 3D prostoru“, s Interneta https://www.pngitem.com/middle/ThRhwmb_rgb-color-model-cube-hd-png-download/ 8. rujna 2022.

[6] „Repozitorij slika za obradu“, s Interneta https://github.com/mohammadimtiazz/standard-test-images-for-Image-Processing/tree/master/standard_test_images 9.rujna 2022.

[7] „Visual studio code“, s Interneta, https://code.visualstudio.com 11.kolovoza 2022.

[8] „Python“, s Interneta, https://www.python.org 11.kolovoza 2022.

[9] „Knjižnice Python-a“, s Interneta https://docs.python.org/3/library/ 11.kolovoza 2022.

[10] „Python 7-zip“, s Interneta https://pypi.org/project/py7zr/ 20.kolovoza 2022.

[11] „Python Pillow“, s Interneta https://pillow.readthedocs.io/en/stable/index.html 20.kolovoza 2022.

[12] „Pillow [y][x]“, s Interneta https://github.com/python-pillow/Pillow/blob/main/src/PIL/PyAccess.py#L136 6. rujna 2022.

[13] Hilbert D.: "Über die stetige Abbildung einer Linie auf ein Flächenstück.",1891.

[14] Peano G.: „Sur une courbe, qui remplit toute une aire plane.“, 1890.

[15] „Prva 3 reda Hilbertove krivulje“, s Interneta https://www.researchgate.net/figure/The-figure-illustrates-Hilbert-curves-for-different-orders-In-this-work-third-order_fig2_330566355 9.rujna 2022.

[16] „Hilbertova krivulja“, s Interneta http://blog.marcinchwedczuk.pl/iterative-algorithm-for-drawing-hilbert-curve 9.rujna 2022.

[17] „Prva 4 reda Z krivulje“, s Interneta https://en.wikipedia.org/wiki/Z-order_curve 12.rujna.2022.

[18] „Huffmanov kod“, s Interneta https://brilliant.org/wiki/huffman-encoding 11.rujna 2022.

[19] „Deflate“, s Interneta https://zlib.net/feldspar.html 11.rujna 2022.

[20] „Usporedba kompresijskih alata“, s Interneta https://tukaani.org/lzma/benchmarks.html 11.rujna 2022.

[21] „ bzip2“, s Interneta https://en.wikipedia.org/wiki/Bzip2 11.rujna 2022.

7. SAŽETAK

Ovaj rad opisuje postupke pretvaranja slike u nizove različitim algoritmima. Isti su zatim komprimirani korištenjem nekih od alata za kompresiju te su uspoređeni dobiveni rezultati. Objašnjeni su algoritmi Hilbertove krivulje, Z krivulje, pretvaranja slike po redovima, stupcima i dijagonali. Za kompresiju su implementirani postupci Zlib, 7zip i Bzip2. Analizirani su i uspoređeni rezultati istraživanja te je opisana najbolja kombinacija algoritma pretvaranja slike u niz i alata kompresije.

Ključne riječi:

Hilbertova krivulja, Z krivulja, kompresija, Obrada slike, Zlib, 7zip, Bzip2, RGB, PNG, Piksel

SUMMARY

This thesis describes the process of converting an image into strings using different algorithms. These are then compressed using some compression tools, and the achieved results are compared. Hilbert space-filling curve, Z-order curve, row-wise, column-wise, and diagonal-wise converting image into a string is explained. For compression Zlib, 7zip, and Bzip2 are implemented. The research results were analyzed and compared, and described the best combination of the image-to-string converting algorithm and compression tool.

Keywords:

Hilbert space-filling curve, Z-order curve, compression, image processing, Zlib, 7zip, Bzip2, RGB, PNG, Pixel

8. POPIS OZNAKA I KRATICA

PNG	Portable Network Graphic
RGB	Red, Green, Blue
HSV	Hue, Saturation, Value
VS Code	Visual Studio Code
MB	Megabyte
GB	Gigabyte
KB	Kilobyte
B	Byte
Bin	binary
7z	7zip
bz2	Bzip2
RLE	Run-length encoding
MTF	Move-to-front
s	seconds
ms	milliseconds