**Informacije in kodi**

Poročilo projektne naloge

Študent: Tilen Tinta

Mentorja: izr. prof. dr. Simon Dobrišek, as. dr. Klemen Grm

Predmet: Informacije in kodi

Datum: 12.5.2024

Vsebina

[1 Uvod 1](#_Toc167036046)

[1 Kodirnik 1](#_Toc167036047)

[1.1 QOI\_OP\_RGB 2](#_Toc167036048)

[1.2 QOI\_OP\_RGBA 2](#_Toc167036049)

[1.3 QOI\_OP\_INDEX 2](#_Toc167036050)

[1.4 QOI\_OP\_RUN 3](#_Toc167036051)

[1.5 QOI\_OP\_DIFF 3](#_Toc167036052)

[1.6 QOI\_OP\_LUMA 3](#_Toc167036053)

[2 Dekodirnik 4](#_Toc167036054)

[3 Vrednotenje 4](#_Toc167036055)

[4 Zaključek 4](#_Toc167036056)

[5 Reference 4](#_Toc167036057)

**Ključne besede: QOI, kodiranje, dekodiranje, vrednotenje**

# Uvod

Pri izbirnem projektu pri predmetu Informacije in kodi smo se tokrat ukvarjali z implementacijo kodirnika in dekodirnika slik QOI. Za preverjanje kakovosti kompresije smo izvedi tudi vrednotenje retultatov.

To je razmeroma nov algoritem razvit leta 2022 kot dokaz, da tudi brez izjemno zahtevne matematike lahko dosežemo dobre rezultate kompresije slik. Iz tega izhaja tudi samo ime (the Quite Ok Image format). Razvil ga je Dominic Szablewski.

# Kodirnik

Prva naloga je bilo napisati kodirnik slik za omenjen format. Kot že omenjeno je celoten algoritem precej kratek s strani pravil. Ta ima svojo logiko zapisano na samo eni strani lista A4 formata in ga sestavlja 6 možnih kodirnih funkcij, ki so opisane v nadaljevanju.

Format zapisa bi lahko razdelili v tri dele. To so glava dokumenta, kodiran zapis same slike in konec dokumenta.

Glavo sestavljajo podatki o sliki ter zapis, ki takoj kaže da gre za QOI format.

**qoi\_header {**

**char magic[4];**

**uint32\_t width;**

**uint32\_t height;**

**uint8\_t channels;**

**uint8\_t colorspace;**

**};**

V »originalu« je algoritem napisan v jeziku C, zato je tudi zgornji zapis primerne oblike.

Spremenlivka *magic[4]* vsebuje zapis »qoif«, ki predstavlja sam zapis in je dolžine 4 bajte. Enako dolgi sta tudi naslednji dve spremenljivki, ki predstavljata širino in višino same slike. Zadnji dve spremenljivki sta dolgi le po en bajt. Prva predstavlja število barvnih kanalov, ki so zastopani v sliki. To so trije v primeru RGB zapisa oz. 4 če gre za RGBA zapis. *Colorspace* predstavlja zapis alfa kanala. Zadnji dve spremenlivki, torej *colorspace* in *cannels* sta zgolj informativni in ne spremenita delovanje algoritma.

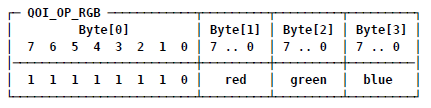
Kodiranje slike se izvaja na sliki od leve proti desni od zgoraj navzdol. Vsak piksel se prebere ter glede na opisane pogoje v nadaljevanju ustrezno kodira.

Konec datoteke predstavlja 8 bajt-ni zapis, katerega sestavlja 7 bajt-ov vrednosti 0 in zadnji vrednosti 1.

Zanimivost je tudi ta, da pri implementaciji ne potrebujemo upoštevati vseh možnih zapisov hkrati. Za kodiranje lahko vzamemo tudi samo določene zapise, kar predvsem olajša pisanje kodirnika.

## QOI\_OP\_RGB

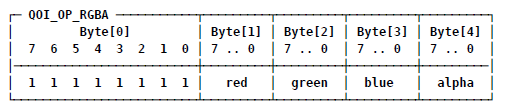
Zapis v tej obliki je najbolj osnoven zapis barvnega elementa oz. piksla. Ta je sestavljen iz 4 bajtov. Prvi bajt je koda, ki predstavlja to obliko zapisa, naslednji trije bajti pa so vsak namenjeni zapisu posamezne barve. Sprejmejo torej vrednosti med 0 in 255 za rdečo, zeleno in modro barvo.



Slika 1‑1: QOI\_OP\_RGB zapis

## QOI\_OP\_RGBA

Na podoben način kot se zapiše osnoven rgb piskel lahko zapišemo tudi rgba piksel, v primeru, da je to oblika s katero je zapisana slika. Zapis je enak, razlikujeta se je prvi bajt, ki predstavlja kodo tega zapisa dodaten bajt na koncu v katerega se vpiše vrednost alfe. Tudi ta sprejme vrednosti med 0 in 255.



Slika 1‑2: QOI\_OP\_RGBA zapis

## QOI\_OP\_INDEX

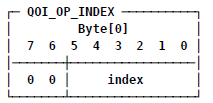
Do sedaj smo pri zapisu enostavno samo pretvarjali vrednosti pikslov v drugi zapis. S tem korakom se postopek spremeni. Ob premikanju skozi sliko sproti gradimo tabelo pikslov velikosti 64 mest, v katero shranjujemo vrednosti zadnjega obdelanega piksla. V primeru, da je trenuten pisel drugačen od prejšnjega na njem izračunamo tako imenovani hash. To je vrednost, ki jo dobimo po sledeči formuli.

Vrednosti piksla množimo s praštevili, ter delimo z velikostjo tabele. Ostanek, ki ga pri tem dobimo je naša lokacija v tabeli kamor shranimo trenutni piksel. Namen uporabe praštevil pri izračunu je, da bi s tem kar se da zmanjšali ponavljanje indeksov in tako prekrivanje ter izgubljanje shranjenih vrednosti v tabeli.

V primeru, da je vrednost trenutnega piksla enaka tej, ki se nahaja v tabeli na lokaciji

na katero kaže trenutno izračunan indeks, se uporabi shranjevanje s tem formatom. Format sestavlja le en bajt. Prva dva bita sta indikatorja da gre za ta zapis, naslednjih šest pa je namenjeno zapisu vrednosti indeksa.

Ta omogoča zapis med 0 in 63, torej indekse cele tabele. To kasneje dekodirniku pove, da je ta piksel enak temu na specifičnem indeksu v tabeli. Enako tabelo gradi tudi dekodirnik.



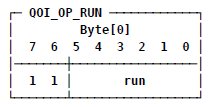
Slika 1‑3: QOI\_OP\_INDEX zapis

Pri zapisovanju s to kodno obliko moramo paziti, da se zaporedoma ne pojavita več kot dva takšna zapisa, ki bi kazala na isti indeks. V tem primeru je za zapis potrebno uporabiti drugo obliko (QOI\_OP\_RUN)

## QOI\_OP\_RUN

Kot je že bilo nekaj omenjeno je ta zapis namenjen shranjevanju večjih enakih zaporednih pikslov. Pri tem si pomagamo z vrednostjo predhodno obdelanega piksla. Če sta trenuten in prejšnji enaka začnemo s štetjem. To izvajamo toliko časa dokler piksel ni več enak prejšnjemu oziroma je število zaporednih doseglo vrednost 62. S to vrednostjo smo omejeni zaradi binarnega zapisa tega koda. Vrednost 63 ima binarni zapis 0b11111110, kar se ujema z zapisom QOI\_OP\_RGB, vrednost 64 pa se zapiše binarno kot ob11111111, kar se ujema z indikatorjem za QOI\_OP\_RGBA zapis. S tem bi naredili v kodu napako in dekodiranje ne bi bilo več možno.

Dodatni pogoj, ki ga je potrebno upoštevati v programu in ni posebej opisan v navodilih je konec slike. V primeru, da dosežemo konec slike in se izvaja RUN zapis oz. štetje je potrebno ob koncu vrednosti shranit, saj v nasprotnem primeru izgubimo podatke.



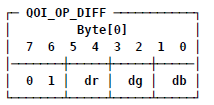
Slika 1‑4: QOI\_OP\_RUN zapis

## QOI\_OP\_DIFF

Osnoven pogoj za izvajanje tega zapisa je, da sta v primeru RGBA formata slike alfa vrednosti trenutnega in predhodnega piksla enaki. Če gre za sliko z zapisom RGB ta pogoj ni pomemben.

Zapis sestavlja en bajt. Prva dva bita sta ponovno indikator, da gre za ta zapis. Ostalih šest bajtov se deli v tri v pare. Vsak od parov je namenjen zapisu razlike posamezne barve piksla. Da se zapis izvede je potreben še dodaten pogoj. Trenutni piksel se od predhodnega za vse tri barve posebej ne sme razlikovati za manj kot 2 ali več kot 1 vrednost. Ker negativnih vrednosti razlik ne moremo zapisat na ta način se razliki prišteje dva, kar nam na koncu za posamezno barvo poda vrednost med 0 in 3.

V primeru, da je razlika ustrezna vendar sta vrednosti na primer 1 in 255 je potrebno pri računanju upoštevati tako imenovan »overflow«.



Slika 1‑5: QOI\_OP\_DIFF zapis

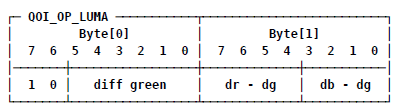
## QOI\_OP\_LUMA

Zapis je podoben prejšnjemu, le da upošteva drugačne pogoje in zahteva daljši izračun zapisanih vrednosti. Upošteva tudi občutljivost človeškega očesa saj daje prednost zeleni barvi.

Za izvedbo zapisa moramo upoštevati enako alfa vrednost trenutnega in predhodnega piksla in ustrezno razliko za posamezno barvo. Rdeča in modra barva morata biti od prejšnje vrednosti manjša od 7 in večja od -8, zelena pa manjša od 31 ter večja od -32.

Če je pogoj izpolnjen je najprej potrebno izračunati razlike med barvami. Vrednosti zelene barve se prišteje 32, s čimer dobimo pozitivno vrednost, ki jo lahko shranimo. To zapišemo v zadnjih 6 bitov prvega bajta. Ravno tako kot pri ostalih zapisih sta tudi pri tem prva dva bita namenjena oznaki zapisa. Za zapis razlike modre in rdeče barve trenutnega in predhodnega piksla potrebujemo izvesti še naslednji izračun.

Izračunani vrednosti je mogoče shraniti v en bajt, torej 4 bite za posamezno vrednost. Tako celoten zapis kodiranja zavzame dva bajta.



Slika 1‑6: QOI\_OP\_LUMA zapis

# Dekodirnik

# Vrednotenje

# Zaključek

V nalogi smo spoznali tako kompresijo kot dekompresijo datotek z algoritmom LWZ. Hkrati je naloga od nas zahtevala ustrezno računanje uspešnosti kompresije katera se glede na velikost datoteke spreminja. To prikazuje tudi graf s slike 2-1, ki prikazuje padanje nivoja kompresije saj se v besedilu začne več vsebine ponavljati in s tem se to lahko optimalnejše zapiše. Hkrati sem algoritem preizkusil tudi na različnih datotekah tako »surovih« kot shranjenih v bolj optimalnih oblikah, kar mi je podalo zanimive rezultate.

# Reference

* Prosojnice vaj: Informacije in kodi, as. Klemen Grm
* Spletni forumi za uporabo python funkcij
* ChatGPT za razlago nekaterih pojmov, algoritmov ter funkcij programskega jezika in snovi
* Implementacija QOI kodirnika: <https://www.youtube.com/watch?v=GgsRQuGSrc0&ab_channel=LowByteProductions>
* Implementacija QOI dekodirnika: <https://www.youtube.com/watch?v=5bWopQj-oQs&ab_channel=LowByteProductions>
* Repozitorij QOI algoritma: <https://github.com/phoboslab/qoi>