Univerza v Ljubljani
Fakulteta za matematiko in fiziko

JPEG algoritem

Milica Vukićević

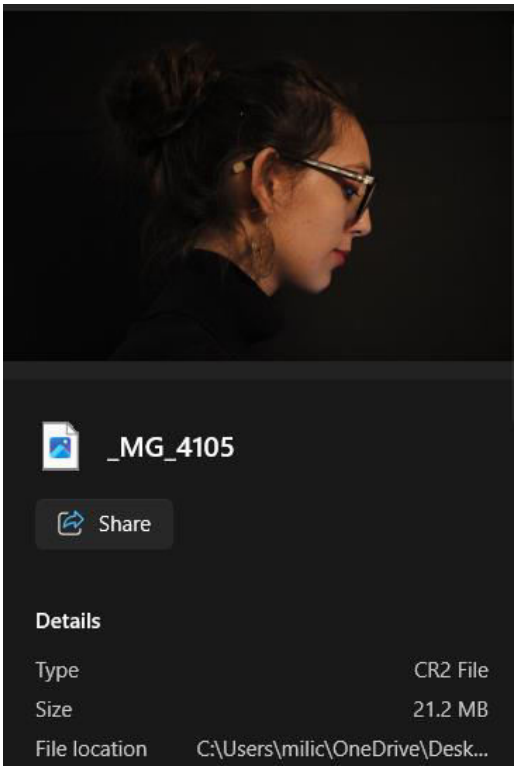
Ljubljana, 2024

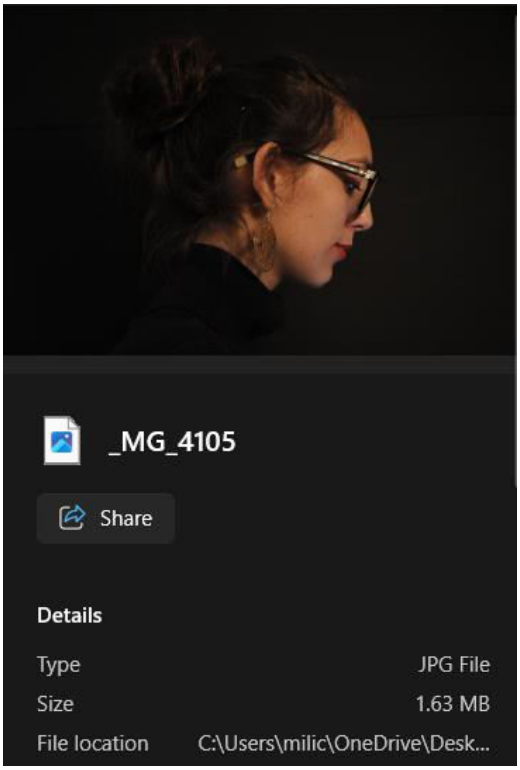
Povzetek

V naslednjem tekstu bomo raziskali proces kodiranja slik s pomočjo JPEG algoritma. Pojasnili bomo, kako JPEG zmanjša velikost slik s kompresijo, pri čemer bomo obravnavali ključne korake, kot so pretvorba slik iz RAW v RGB in nato v YCrCb format. Osredotočili se bomo na postopek diskretne kosinusne transformacije (DCT), kvantizacijo in linearno transformacijo podatkov. Prav tako bomo razložili, kako se DC in AC komponente kodirajo, vključno z uporabo kodiranja po dolžini (Run-Length Encoding) in Huffmanovega kodiranja za dodatno stiskanje podatkov. Na koncu bomo obravnavali shranjevanje kodiranih podatkov in njihovo organizacijo v JPEG formatu.

Kje se uporablja JPEG

Ko kamera zajame sliko, senzorji ustvarijo RAW format, ki beleži izjemno natančne podrobnosti in vključuje ogromno podatkov, ki jih naše oko ne zazna. Ker pa so ti podatki pogosto preveliki za shranjevanje, se uporablja JPEG algoritem, ki ohranja le tiste informacije, ki jih človeško oko najbolj opazi, ostale pa odstrani. S tem se velikost slike znatno zmanjša, ne da bi pri tem opazili občutno izgubo kakovosti. Proces pretvorbe iz RAW v JPEG poteka zelo hitro, saj se na večini naprav, kot so pametni telefoni, slike že v osnovi shranjujejo v JPEG formatu, ki je danes eden najbolj razširjenih standardov za shranjevanje slik. V spodnjem primeru lahko opazimo, da je velikost RAW formata 21 MB, medtem ko je JPEG format bistveno manjši, le 1,6 MB.



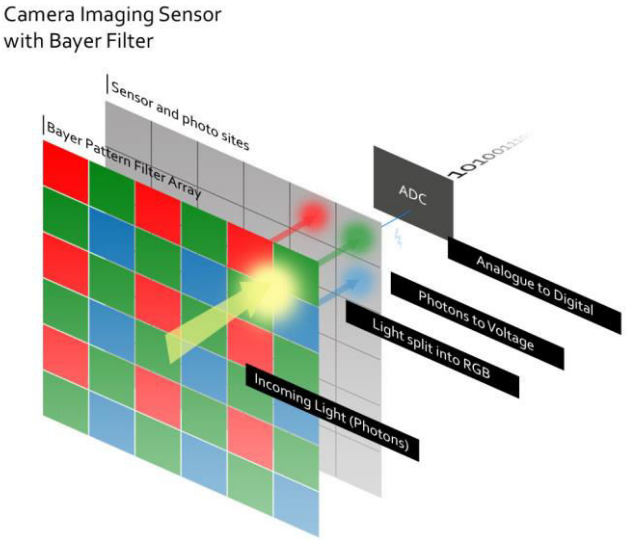


Sliki 1 in 2: predstavitev velikosti RAW in JPEG formata

Kako nastane slika

Slike, ki jih posname fotoaparat, predstavljajo enega izmed možnih vhodnih podatkov za JPEG algoritem, pri čemer je RAW format surovi zapis, ki vsebuje vse zajete informacije. Medtem ko so na voljo tudi drugi formati, kot so JPEG, PNG, TIFF in Bitmap, se bomo osredotočili na RAW format.

Fotoaparat je opremljen s senzorji, ki zaznavajo barvo in intenziteto svetlobe, te podatke pa nato pretvarjajo v električne signale. Vsak piksel na senzorju je sestavljen iz štirih detektorjev barvne svetlobe: enega za modro, enega za rdečo in dveh za zeleno svetlobo. Ta razporeditev ni naključna; izbrana je zato, ker je človeško oko bolj občutljivo na zeleno barvo, saj ima več receptorjev za zaznavanje zelene svetlobe kot za rdečo in modro.



Slika 3: Senzorji kamere in Bajerovi filterji

Zbrane informacije iz senzorja se razporedijo v obliki mozaika, kar tvori RAW format. Nato sledi proces demosaiciranja, kjer se surove informacije iz RAW formata pretvorijo v RGB format. Že v tem koraku lahko pride do izgube nekaterih podatkov. Obstaja veliko različnih algoritmov za demosaiciranje, ki vplivajo na končno kakovost slike.

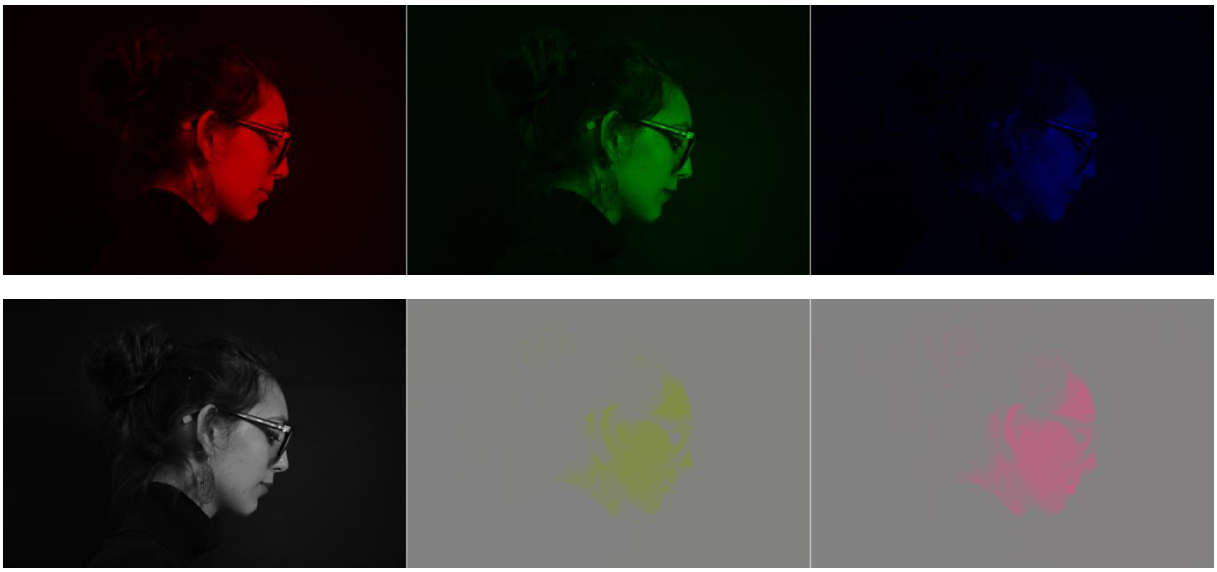
Po demosaiciranju dobimo sliko v RGB formatu. Več o demosaiciranju si lahko preberete na naslednji povezavi (**Canon Europe.** “Image Sensors Explained.”).

Slika 5: Primer slike v RAW formatu

Pretvorba iz RGB v YCrCb format

Naslednji korak v JPEG algoritmu je pretvorba podatkov iz RGB formata v YCrCb format. V tem formatu komponenta Y predstavlja svetilnost, medtem ko Cr in Cb določata kromatičnost rdeče in modre barve.

Na spodnjih primerih lahko vidimo, kako izgledajo RGB in YCrCb komponente slike. Opazimo lahko, da svetilnost (Y) vpliva na zaznavanje slike veliko bolj kot razlike v kromatičnosti.



Sliki 6 in 7: Slika raskosana na RGB in YCrCb komponente

Pretvorba iz RGB v YCrCb se izvaja s pomočjo naslednjih formul:

- Y = 16 + 65.738 * R/256 + 129.057 * G/256 + 25.064 * B/256

- Cb = 128 - 37.945 * R/256 - 74.494 * G/256 + 112.439 * B/256

- Cr = 128 + 112.439 * R/256 - 94.154 * G/256 - 18.285 * B/256

Pri tem postopku ne izgubimo nobenih podatkov.

Ker so naše oči veliko bolj občutljive na svetilnost (Y) kot na kromatične komponente (Cr in Cb), lahko zmanjšamo količino podatkov pri Cr in Cb. To dosežemo tako, da povprečimo vrednosti Cr in Cb štirih sosednjih pikslov in shranimo le to povprečno vrednost. Tako sicer izgubimo nekaj podatkov, vendar je ta izguba skoraj neopazna, saj človeško oko ni tako občutljivo na kromatične informacije.

Poglejmo kako postopek izgleda na primeru. Recimo da imamo naslednjo matriko:

12	45	78	100	155	202	225	240
23	56	82	115	142	189	213	233
34	78	91	130	160	195	215	248
45	78	104	143	175	202	220	246
56	89	113	152	180	210	230	240
67	90	125	160	192	215	243	245
78	103	136	172	205	220	228	246
89	115	145	185	210	234	240	248

Vzamemo vrednosti podmatrike velikosti 2x2, in ih povprečimo. V zgornjem primeru bi to bila matrika:

$$\begin{bmatrix} 12 & 45 \\ 23 & 56 \end{bmatrix}$$

Povprečimo vrednosti v matriki:

$$\frac{12 + 45 + 23 + 56}{4} = 34$$

Če je potrebno, jih zaobkrožimo na najbližje celo število. Na koncu postopka dobimo 4x4 matriko povprečnih vrednosti.

34	94	172	228
59	117	183	232
76	138	199	240
96	160	217	241

Po koncu algoritma, lahko rekonstruiramo 8x8 matriko, ki bo imela povprečno vrednost namesto vrednosti v podmatriki.

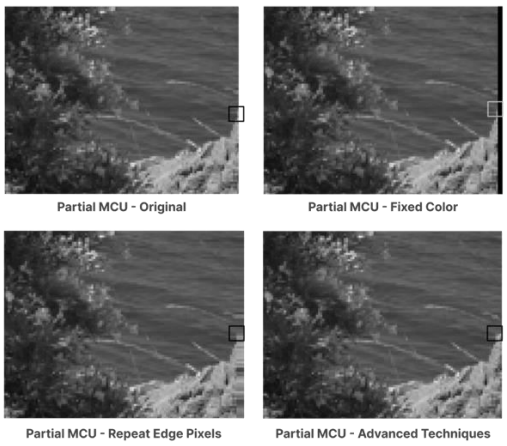
34	34	94	94	172	172	228	228
34	34	94	94	172	172	228	228
59	59	117	117	183	183	232	232
59	59	117	117	183	183	232	232
76	76	138	138	199	199	240	240
76	76	138	138	199	199	240	240
96	96	160	160	217	217	241	241
96	96	160	160	217	217	241	241

Po tem koraku se podatki za Y, Cr in Cb obravnavajo na podoben način, zato se bomo v nadaljevanju osredotočili predvsem na Y - svetilnost.

Kosanje slike

Naslednji korak je razdelitev slike na kvadratke velikosti 8x8 pikslov. Če slika ni v dimenzijah, kjer sta dolžina in širina deljivi z 8, lahko uporabimo eno od naslednjih treh tehnik za prilagoditev dimenzij:

1. **Dodajanje Robov**: Dodamo dodatne vrstice in stolpce nevtralne barve ob robovih slike, da dosežemo želene dimenzije.
2. **Rezanje**: Odrežemo robove slike, da dolžina in širina postaneta deljiva z 8.
3. **Podvojimo Robove**: Povečamo sliko tako, da podvojimo vrstice in stolpce na robovih, dokler ne dosežemo dimenzijo ki je deljiva z 8



Slika 8: Primeri kosanja slike

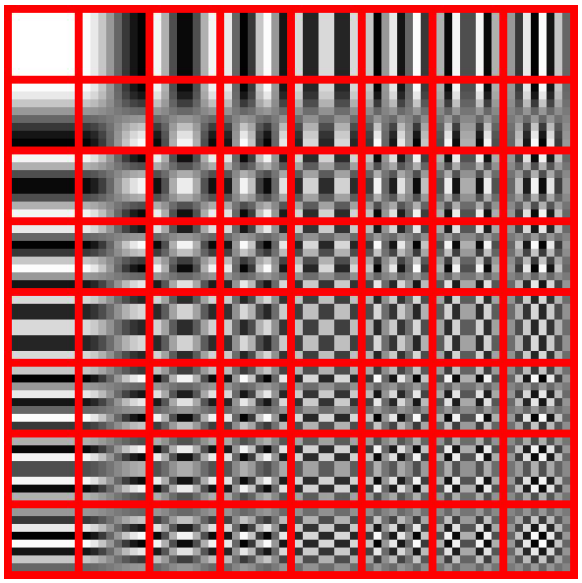
Diskretna Kosinusna Transformacija

Nato preidemo na postopek diskretne kosinusne transformacije (DCT).

Vhodni podatek za DCT je matrika velikosti 8x8, kjer vsako število predstavlja svetlobno vrednost (Y) posameznega piksla v tem bloku slike.

52	55	61	66	70	61	64	73
63	59	55	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	68	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94

Po izvajanju DCT na tej matriki dobimo novo matriko, kjer vsak element (i, j) predstavlja število pojavitev frekvenčne komponente (i, j) izmed 64-ih možnih.



Slika 9: Frekvenčne komponente pri DCT-ju

DCT lahko razumemo kot linearno kombinacijo baznih vektorjev, kjer so ti bazni vektorji predstavitve osnovnih slikovnih vzorcev (frekvenčnih komponent), števila v rezultantni matriki pa predstavljajo skalarne koeficiente. Z združevanjem teh skalarjev z ustreznimi baznimi slikami dobimo naš originalni 8x8 blok slike.

Ta postopek se izvaja z uporabo naslednje matematične formule.

$$DCT(i,j) = \frac{1}{\sqrt{2N}} C(i)\,C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} pixel(x,y) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

$$C(x) = \frac{1}{\sqrt{2}} \text{ if } x \text{ is } 0, \text{ else } 1 \text{ if } x > 0$$

Formula 1: Formula za DCT

V formuli N predstavlja dimenzijo vhodne matrike, ki je v našem primeru 8. Pixel (x,y) predstavlja vrednost iz vhodne matrike na poziciji (x,y). DCT(i,j) predstavlja vrednost (i,j) izhodne matrike.

Pri DCT-ju ne izgubimo nobenih podatkov.

Po koncu DCT-ja imamo podatke v obliki matrike frekvenc baznih komponent.

1219	-123	-31	-24	-10	4	-4	-1
-214	13	-20	22	-18	-10	2	3
-2	11	-17	-9	-3	-1	3	-4
8	5	10	11	-4	6	-3	-3
-5	-2	-7	3	-6	-6	1	-1
-4	1	3	-3	1	-1	1	0
-3	2	-1	-1	-2	0	-1	1
-1	1	-1	-2	-1	0	0	0

Formula za DCT ima tudi inverz.

Kvantizacija

Naslednji korak v procesiranju slike je kvantizacija. Ko imamo 8x8 matriko, pridobljeno z diskretno kosinusno transformacijo (DCT), jo kvantiziramo tako, da vsak njen element delimo z ustreznim elementom kvantizacijske matrike. To izvedemo na naslednji način: za vsak element na poziciji (i, j) v DCT-matriki delimo njegovo vrednost z vrednostjo na isti poziciji (i, j) v kvantizacijski matriki.

Kvantizacijske matrike so vnaprej določene in njihova izbira vpliva na kakovost končne slike; razlikujejo se tudi glede na vrsto slike (Y za luminanco in Cb/Cr za barvnost).

Svetilnost

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Barvnost

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Slika 10: Primera kvantizacijskih matrik

Naše oči so bolj občutljive na frekvence, ki so zapisane v zgornjem levem kotu matrike. Zato so v kvantizacijski matriki vrednosti v tem območju manjše, kar pomaga ohraniti več detajlov. Po drugi strani smo manj občutljivi na frekvence v spodnjem desnem kotu matrike. Zato so v tem delu kvantizacijske matrike vrednosti večje, kar omogoča večje izgube podatkov brez vidnega vpliva na kakovost slike. Ta metoda omogoča učinkovitejše stiskanje podatkov in zmanjšanje velikosti datoteke, medtem ko še vedno ohranja sprejemljivo kakovost slike.

Oglejmo si, kako kvantizacija izgleda na naslednjem primeru:

Matrika, ki jo dobimo po koncu DCT-ja:

1219	-123	-31	-24	-10	4	-4	-1
-214	13	-20	22	-18	-10	2	3
-2	11	-17	-9	-3	-1	3	-4
8	5	10	11	-4	6	-3	-3
-5	-2	-7	3	-6	-6	1	-1
-4	1	3	-3	1	-1	1	0
-3	2	-1	-1	-2	0	-1	1
-1	1	-1	-2	-1	0	0	0

Kvantizacijska matrika za svetilnost:

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

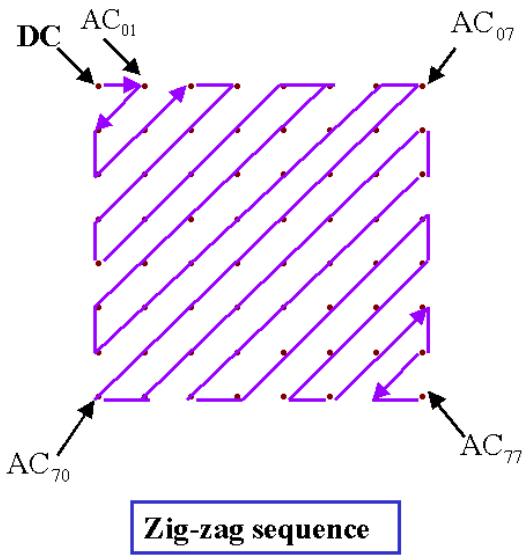
Pri procesu kvantizacije vsako število iz matrike, ki jo dobimo po izvedeni DCT, delimo s številom na isti poziciji v kvantizacijski matriki in nato zaokrožimo na najbližje celo število. Na primer, 1219 delimo s 16, nato -123 delimo z 11, in tako naprej. Ko zaključimo postopek kvantizacije, dobimo novo matriko, v kateri je veliko ničel.

Matrika po končani kvantizaciji:

76	-11	-3	-2	0	0	0	0
-18	0	-1	1	0	0	0	0
0	1	-1	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Linearna transformacija

Naslednji korak je linearna transformacija, kjer številke iz matrike zapišemo v eno samo vrstico. Namesto da jih beremo po vrsticah matrike, jih bomo zapisali v zig-zag vzorcu. Ta pristop nam omogoča, da združimo čim več ničel skupaj, kar olajša učinkovito stiskanje podatkov in omogoča krajši zapis.



Slika 11: Primer linearne transformacije

Poglejmo, kako to izgleda v praksi:

76	-11	-3	-2	0	0	0	0
-18	0	-1	1	0	0	0	0
0	1	-1	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Če uporabimo linearno transformacijo na zgornji matriki, dobimo naslednji niz vrednosti: {76, -11, -18, 0, 1, -3, -2, -1, 1, 1, 0, 0, -1, 1, 0, …, 0}. Opazimo, da je prva številka bistveno večja

od ostalih. Ta komponenta se imenuje DC komponenta, medtem ko so ostale vrednosti AC komponente.

DC komponente so vedno večje od AC komponent, poleg tega pa se DC komponente zaporednih 8x8 blokov slike med seboj le malo razlikujejo. To je posledica dejstva, da se osvetlitev in intenziteta barv med sosednjimi bloki 8x8 pikslov običajno ne spreminjata bistveno. Zaradi tega DC komponente obravnavamo posebej. Namesto shranjevanja vsake vrednosti DC komponente, zabeležimo začetno vrednost in nato shranjujemo le razlike med zaporednimi DC komponentami.

Oglejmo si, kako to deluje v praksi:

Recimo da imamo matriko DC komponent 8x8 kosov pikslov v sliki.

76	75	78	72	68	65	65	…
78	79	77	73	72	…		
79	80	81	83	…			
⋮							

Sedaj shranimo samo prvo vrednost v vrstici matrike, nato pa shranimo le razlike zaporednih komponent. Torej je $d_{i+1} = DC_{i+1} - DC_i$, kjer je d_i element na i-tem mestu v novi matriki.

76	-1	3	-6	-4	-3	0	…
78	1	-2	-4	-1	…		
79	1	11	2	…			
⋮							

Kodiranje po dolžini (Run-Length Encoding)

Zdaj, ko vemo, kako zakodirati DC komponente, se postavlja vprašanje, kako zakodiramo AC komponente. Najbolje je to prikazati na primeru:

- {76, - 11, - 18, 0,1, -3, -2, -1, 1, 1, 0, 0, -1, 1, 0,...., 0}

Opazimo, da se na koncu zapisa pojavi 50 zaporednih ničel. Namesto da bi shranili vseh 50 ničel, jih lahko zapišemo kot (50, 0). Tako namesto 50 številk shranimo le dve. Na ta način bomo obdelali vse AC komponente. Torej, imamo:

- {76, - 11, - 18, 0,1, -3, -2, -1, 1, 1, 0, 0, -1, 1, 0,...., 0}

$\Longrightarrow$ (1, -11), (1, - 18), (1, 0), (1, -3), (1, -2), (1, -1), (2, 1), (2, 0), (1, -1), (1, 1), (50, 0)

- {100, 5, -3, 5, 0, 0, 1, 8, -3, 2, -1, -1, -1, 0, 0, 0, 1, 0,..., 0}

$\Longrightarrow$ (1, 5), (1, -3), (1, 5), (0, 2), (1, 1), (1, 8), (1,-3), (1, 2), (3, -1), (3, 0), (1, 1), (47,0)

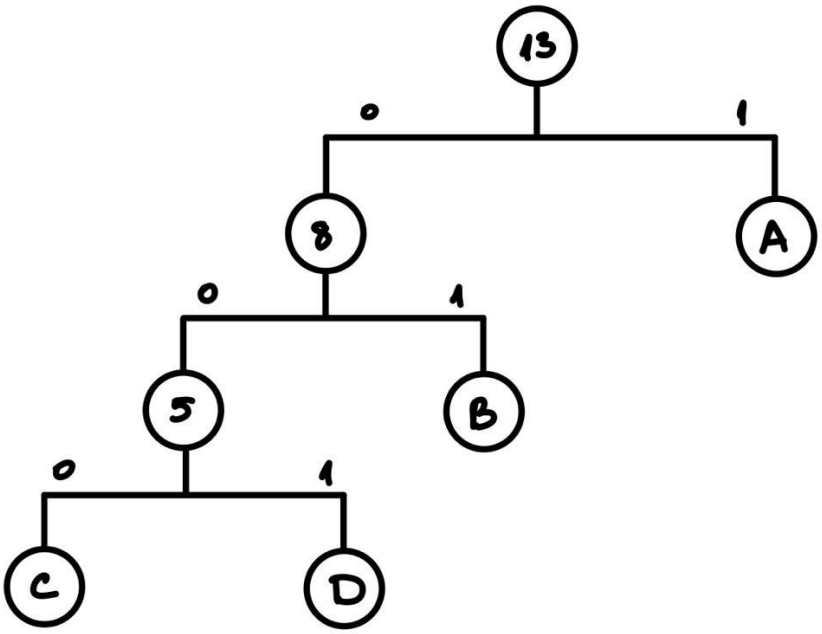
S pomočjo kodiranja po dolžini ustvarimo krajši zapis podatkov, kar je eden izmed razlogov, zakaj JPEG format potrebuje tako malo prostora za shranjevanje slik.

Huffmanovo kodiranje

Ko so AC in DC komponente zakodirane, jih dodatno stisnemo s Huffmanovim kodiranjem. Huffmanovo kodiranje deluje tako, da najprej uredimo vse podatke glede na njihovo frekvenco pojavljanja in nato zgradimo Huffmanovo drevo. To drevo nam omogoči krajši zapis podatkov.

Poglejmo primer: recimo, da želimo zakodirati niz “ABCDAACDCBBAA”. Če bi ta niz zapisali z uporabo ASCII binarnega zapisa, bi potrebovali 8 bitov za vsako črko, kar bi skupaj znašalo 104 bitov (8x13). Namesto tega zgradimo Huffmanovo drevo, ki omogoča bolj učinkovit zapis. Obstaja več načinov za izgradnjo Huffmanovega drevesa, več o tem si lahko preberete na tej povezavi (**Wikipedia.** “Huffman Coding.”).

Če s podanimi podatki zgradimo Huffmanovo drevo, dobimo novo drevo, kjer 1 pomeni premik v desno poddrevo, 0 pa premik v levo. S tem pristopom ustvarimo krajši zapis za naš niz.



Slika 12: Primer Huffmanovega drevesa

Namesto 104 bitov zdaj potrebujemo le 26 bitov (5x1 + 3x2 + 3x3 + 2x3) za zapis istega niza.

Črka	Št. ponavljanj	ASCII binarni zapis	Zapis po Huffmanovem kodiranju
A	5	01000001	1
B	3	01000010	01
C	3	01000011	000
D	2	01000100	001

V JPEG algoritmu lahko zgradimo Huffmanovo drevo na podlagi podatkov iz slike, kar še dodatno zmanjša velikost zapisa. Obstajajo tudi standardizirane različice Huffmanovih dreves, ki so ustvarjene na podlagi analize velikega števila slik. Če zgradimo svoje Huffmanovo drevo, ga moramo shraniti skupaj s podatki. S pomočjo Huffmanovega kodiranja JPEG algoritem znatno zmanjša velikost datoteke slike, kar prispeva k učinkovitemu shranjevanju.

Shranjevanje podatkov

Ko je kodiranje zaključeno, sledi še shranjevanje podatkov. V JPEG algoritmu obstaja določen vrstni red kodiranja podatkov, ki zagotavlja optimalno stiskanje in kakovost slike. Več podrobnosti o tem postopku lahko preberete na naslednji povezavi (Charette 2017).

Viri:

- **Adams, Richard.** “How Image Sensors Work.” *Silent Peak Photo*, 19. junij 2020. https://www.silentpeakphoto.com/how-image-sensors-work/ (dostop: 24. november 2023).
- **Charette, Stéphane.** “Anatomy of a JPEG.” *CCoderun.ca*, 1. februar 2017. https://www.ccoderun.ca/programming/2017-01-31_jpeg/ (dostop: 25. avgust 2024).
- **CMLab.** “Encoding Process.” *CMLab - National Taiwan University*, brez datuma. https://www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/jpeg/jpeg/encoder.htm (dostop: 25. avgust 2024).
- **Friendly College.** “Discrete Cosine Transform and Haar Transform with Examples.” *YouTube*, 2. april 2021. https://www.youtube.com/watch?v=lQnUex31f5Q (dostop: 25. avgust 2024).
- **Computerphile.** “JPEG DCT, Discrete Cosine Transform (JPEG Pt2).” *YouTube*, 22. maj 2015. https://www.youtube.com/watch?v=Q2aEzeMDHMA (dostop: 25. avgust 2024).
- **ProGrade Digital.** “Understanding Camera Sensors: A Comprehensive Guide.” *ProGrade Digital*, 3. julij 2023. https://progradedigital.com/understanding-camera-sensors-a-comprehensive-guide/ (dostop: 25. avgust 2024).
- **ENGEGY.** “The JPEG Compression Algorithm.” *YouTube*, 22. maj 2016. https://www.youtube.com/watch?v=aFbGqXFT0Nw (dostop: 25. avgust 2024).
- **GeeksforGeeks.** “JPEG Full Form.” *GeeksforGeeks*, 30. maj 2022. https://www.geeksforgeeks.org/jpeg-full-form/ (dostop: 25. avgust 2024).
- **Griffin, Jonathan.** “JPEG DCT Padding.” *DSP Stack Exchange*, 23. januar 2023. https://dsp.stackexchange.com/questions/35339/jpeg-dct-padding (dostop: 25. avgust 2024).
- **Reducible.** “Huffman Codes: An Information Theory Perspective.” *YouTube*, 30. julij 2021. https://www.youtube.com/watch?v=B3y0RsVCyrw (dostop: 25. avgust 2024).
- **Pizzey Technology.** “Huffman Coding Step-by-Step Example.” *YouTube*, 13. januar 2019. https://www.youtube.com/watch?v=iEm1NRyEe5c (dostop: 25. avgust 2024).
- **Trauth, Martin H.** “Multispectral Cameras, Bayer Mosaics, and Image Processing with Matlab.” *Matlab and Python Recipes for Earth Sciences*, 10. julij 2018. http://mres.uni-potsdam.de/index.php/2018/07/10/multispectral-cameras-bayer-mosaics-and-image-processing-with-matlab/ (dostop: 25. avgust 2024).
- **Wikipedia.** “YCbCr.” brez datuma. https://en.wikipedia.org/wiki/YCbCr (dostop: 25. avgust 2024).
- **Wikipedia.** “Bayer Filter.” brez datuma. https://en.wikipedia.org/wiki/Bayer_filter (dostop: 25. avgust 2024).
- **Wikipedia.** “Demosaicing.” 18. junij 2024. https://en.wikipedia.org/wiki/Demosaicing (dostop: 25. avgust 2024).
- **Wikipedia.** “JPEG.” brez datuma. https://en.wikipedia.org/wiki/JPEG (dostop: 25. avgust 2024).
- **Wikipedia.** “Huffman Coding.” brez datuma. https://en.wikipedia.org/wiki/Huffman_coding (dostop: 25. avgust 2024).

- **Canon Europe.** “Image Sensors Explained.” *Canon Europe*. https://www.canon-europe.com/pro/infobank/image-sensors-explained/#:~:text=Just%20as%20the%20retina%20in,to%20form%20a%20digital%20image (dostop: 25. Avgust 2024).