DIGITALNA OBRADA SLIKE

ČAS 5/6 – KOMPRESIJA SLIKE

Uvod

Kompresija slike predstavlja nauku i umetnost smanjvanja količine podataka potrebnih za reprezentaciju slike.

Kompresija omogućava efikasno skladištenje i razmenu fajlova što je od posebne važnosti kod slika i video sekvenci. Sam proces kompresije i dekompresije je potpuno nevidljiv korisniku.

Bez kompresije bi film FullHD rezolucije (1920x1080) trajanja 2h zauzimao:

$$30\frac{\text{frames}}{\text{sec}} \times (1920 \times 1080)\frac{\text{pixels}}{\text{frame}} \times 3\frac{\text{bytes}}{\text{pixel}} \times 7200\,\text{sec} = 1.22\text{TB}$$

Osnovni pojmovi

Kompresija podataka predstavlja proces smanjivanja količine podataka koji se koriste za predstavljanje određene informacije.

Za reprezentacije koje sadrže nerelevantne ili ponovljene informacije kaže se da sadrže redundantne podatke.

Stepen kompresije se definiše kao odnos količine podataka u dve različite reprezentacije potrebnih za predstavljanje iste informacije:

$$C = \frac{b}{b'}$$

Relativna redudansa podataka se tada definiše kao:

$$R = 1 - \frac{1}{C}$$

Različiti tipovi redudanse

- **Kodna redudansa**

  Broj upotrebljenih bita po pikselu je veći od minimalnog broja bita potrebnog za predstavljanje date slike.

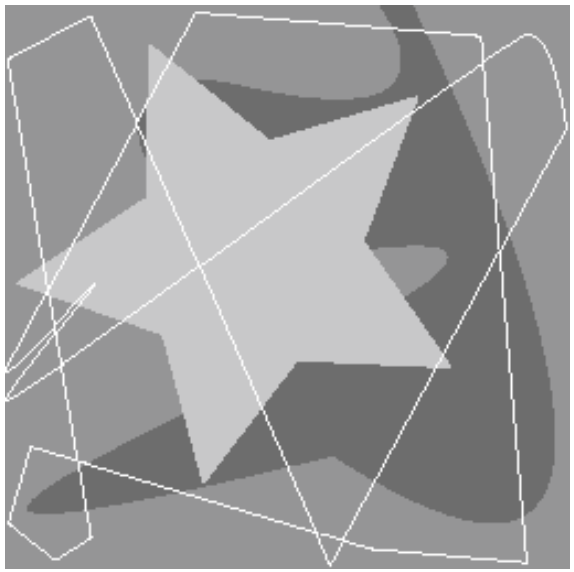
- **Prostorna i vremenska redudansa**

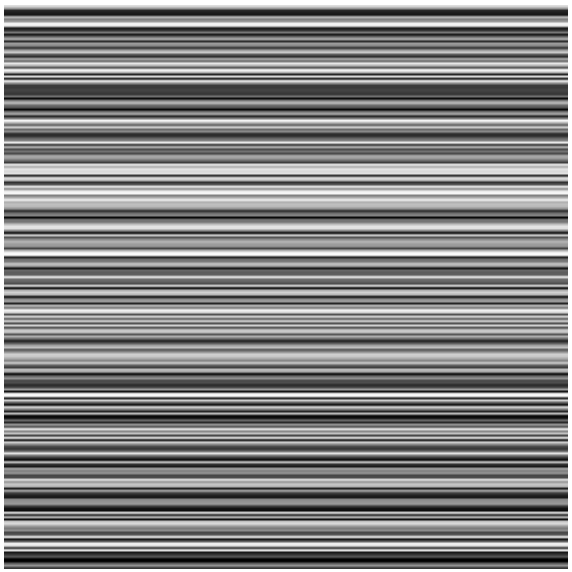
  Kako su pikseli u slici i video signalu prostorno i vremenski korelisani ista informacija se ponavlja predstavljanjem korelisanih piksela.

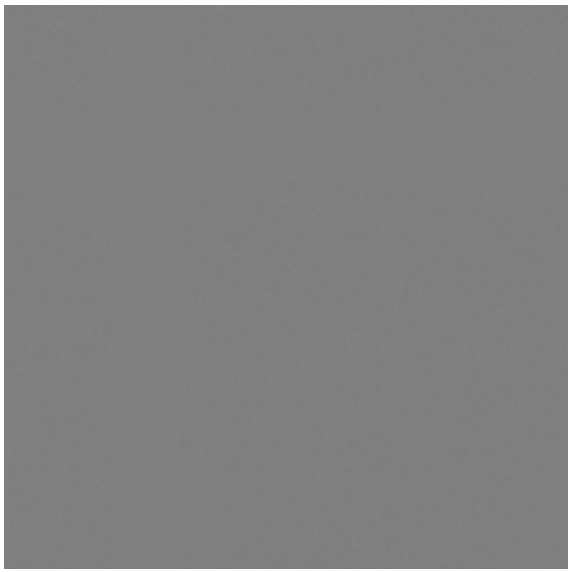
- **Irelevantne informacije**

  Dosta informacije na slikama biva ignorisano od strane ljudskog vizuelnog sistema (ili nekog drugog sistema koji koristi slike). Dakle ove informacije su redudantne u smislu da se ne koriste i da njihov nedostatak neće biti primetan.

Različiti tipovi redudanse







Kodna redudansa

Gustina verovatnoće pojavljivanja određenog od ukupno 2^B nivoa sivog u slici dimenzija MxN definiše se kao:

$$p(i) = \frac{n_i}{MN} \qquad i = 0,1,2,\ldots,2^B - 1$$

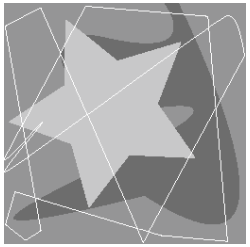
n_i - ukupan broj piksela inteziteta i

Ako je ***L(i)*** broj bita korišćen za predstavljanje intenziteta ***i*** onda je prosečna dužina kodne reči:

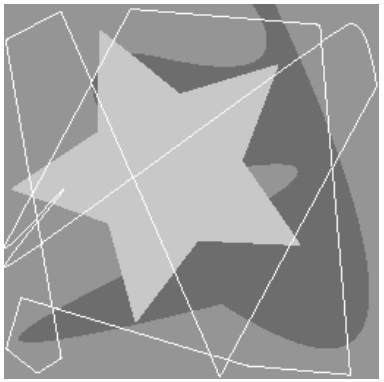
$$L_{avg} = \sum_{i=0}^{2^B-1} L(i)\,p(i)$$

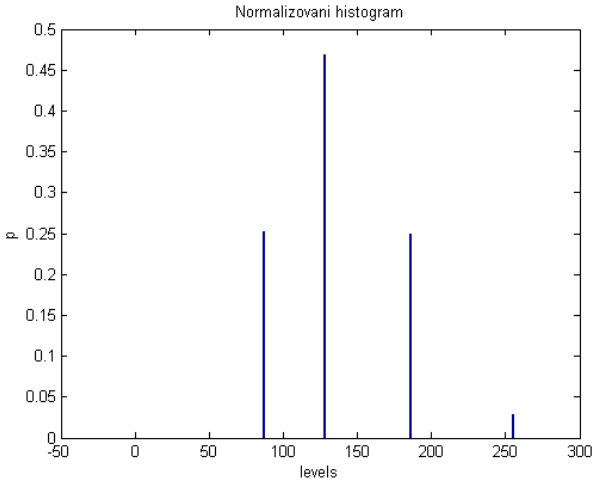
Ukupan broj bita potreban za predstavljanje slike je $\boldsymbol{MNL_{avg}}$

Kodna redudansa - primer

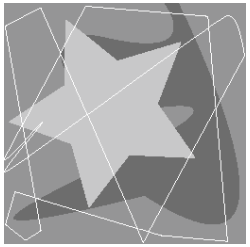


```
I = imread('star.tif');

[M, N] = size(I);

levels = [0:255];

p = hist(I(:), levels)/(M*N);
figure; bar(levels, p);

levels(p>0)

p(p>0)
```





Kodna redudansa - primer



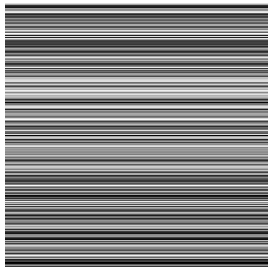
i	p(i)	kod 1	L_1(i)	kod 2	L_2(i)
87	0.25	01010111	8	01	2
128	0.47	10000000	8	1	1
186	0.25	11000100	8	000	3
255	0.03	11111111	8	001	3
ostali	0		8		0

$$L_{avg1} = 8$$

$$L_{avg2} = 0.25 \times 2 + 0.47 \times 1 + 0.25 \times 3 + 0.03 \times 3 = 1.81$$

$$C = \frac{256 \times 256 \times 8}{256 \times 256 \times 1.81} = 4.42$$

Prostorna redudansa - primer



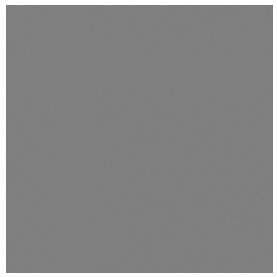
Svi nivoi sivog su podjednako zastupljeni (uniforman histogram) – ne postoji kodna redudansa

Međutim pikseli su potpuno korelisani u horizontalnom pravcu. Slika se može predstaviti u alternativnoj reprezentaciji koja se naziva kodovanje dužine niza. Umesto vrednosti pojedinačnih piksela svaki element se predstavlja kao uređeni par **(vrednost intenziteta piksela, broj susednih piksela koji imaju isti taj intenzitet).** Na ovaj način je za predstavljanje linije u slici sa primera umesto 256 bajtova dovoljno iskoristiti 2 bajta.

Postignuti stepen kompresije u ovom slučaju je:

$$C = \frac{256 \times 256 \times 8}{256 \times 2 \times 8} = 128$$

Irelevantne informacije - primer



Odbacivanjem irelevantnih informacija neminovno se **uvode gubici** u proces kompresije pošto je sam proces odbacivanja informacija **ireverzibilan**.

S druge strane odbacivanjem irelevantnih informacija se postižu **najveći stepeni kompresije**.

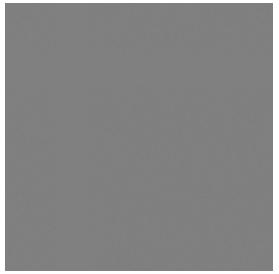
Kako deluje da slika sa primera ima uniformni intenzitet (svi pikseli imaju istu vrednost) to se cela slika može predstaviti jednim bajtom koji predstavlja taj intenzitet.

Postignuti stepen kompresije je:

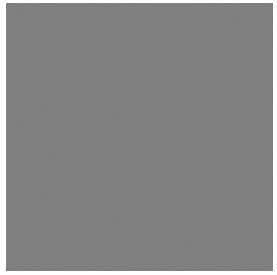
$$C = \frac{256 \times 256 \times 8}{8} = 65536$$

slika gde svi pikseli imaju vrednost 128

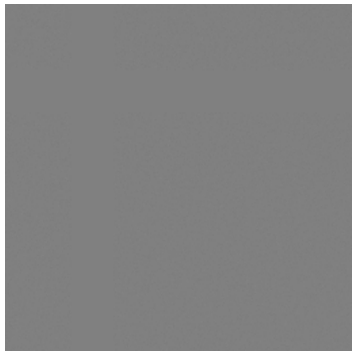
ulazna slika



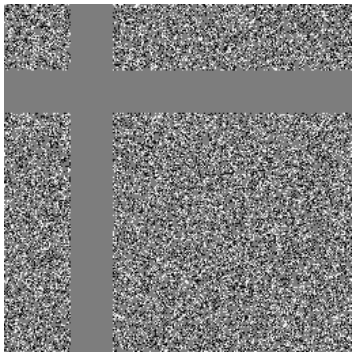
Irelevantne informacije - primer



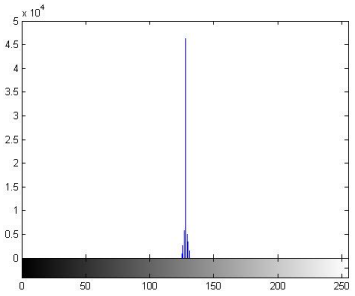
ulazna slika



posle ekvalizacije histograma



histogram ulazne slike



Ekvalizacijom histograma informacija koja je bila neprimetna postaje uočljiva.

U slučaju kompresije iz prethodnog primera, ova informacija se nikako ne bi mogla dobiti iz dekompresovane slike pošto je odbačena u postupku kompresije.

Kod primena gde je svaka informacija od velike važnosti, kao na primer u medicini, kompresije sa gubicima nisu dozvoljene!

Merenje količine informacije

Osnovna pretpostavka je da se generisanje informacije može posmatrati kao slučajni proces. Prema tome slučajni događaj **E** sa verovatnoćom pojavljivanja **P(E)** sadrži **I(E)** jedinica informacije pri čemu je:

$$I(E) = \log \frac{1}{P(E)} = -\log P(E)$$

Osnova logaritma zavisi od jedinice kojom se meri informacija. Ako se informacija meri u bitima onda se koristi logaritam sa osnovom 2.

Ako se posmatra skup nezavisnih slučajnih događaja a_i sa verovatnoćama pojavljivanja $P(a_i)$ onda se prosečna entropija izvora definiše kao:

$$H = -\sum_{j=1}^{J} P(a_j) \log P(a_j)$$

Merenje količine informacije

Ako se pikseli slike posmatraju kao realizacije slučajnih promenljivih koje dolaze iz istog imaginarnog izvora onda se entropija slike može predstaviti kao:

$$H(B) = -\sum_{i=0}^{2^B-1} p(i)\log_2 p(i)$$

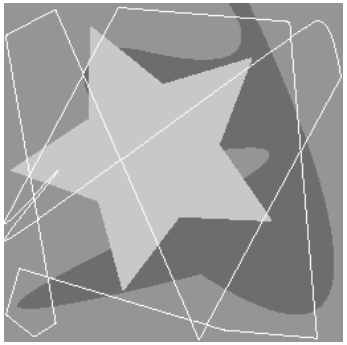
Po prvoj Šenonovoj teoremi donja granica prosečne dužine kodne reči definisana je entropijom slike:

$$L_{avg} \geq H(B)$$

Količina informacije - primer

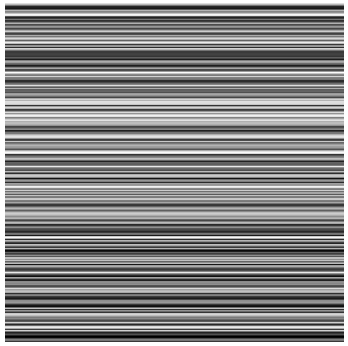
```
I = imread('lena.tif');
entropy(I)

I = imread('star.tif');
entropy(I)

I = imread('stripes.tif');
entropy(I)

I = imread('gray.tif');
entropy(I)
```

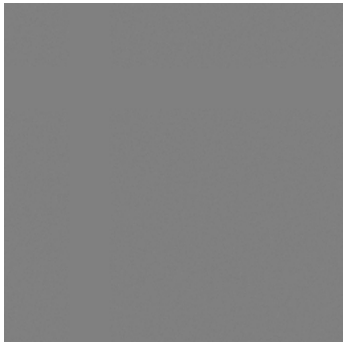
H=7.44



H=1.66

H=8





H=1.57

Hafmanov kodni postupak

Najpopularniji koder za uklanjanje kodne redudanse je Hafmanov koder.

Hafmanov koder obezbeđuje minimalan prosečan broj bita u slučaju kada se svaki piksel koduje zasebno.

Prosečna dužina kodne reči dobijena Hafmanovim kodnim postupkom je dosta bliska entropiji slike:

$$H(B) \leq L_{avg} \leq H(B)+1$$

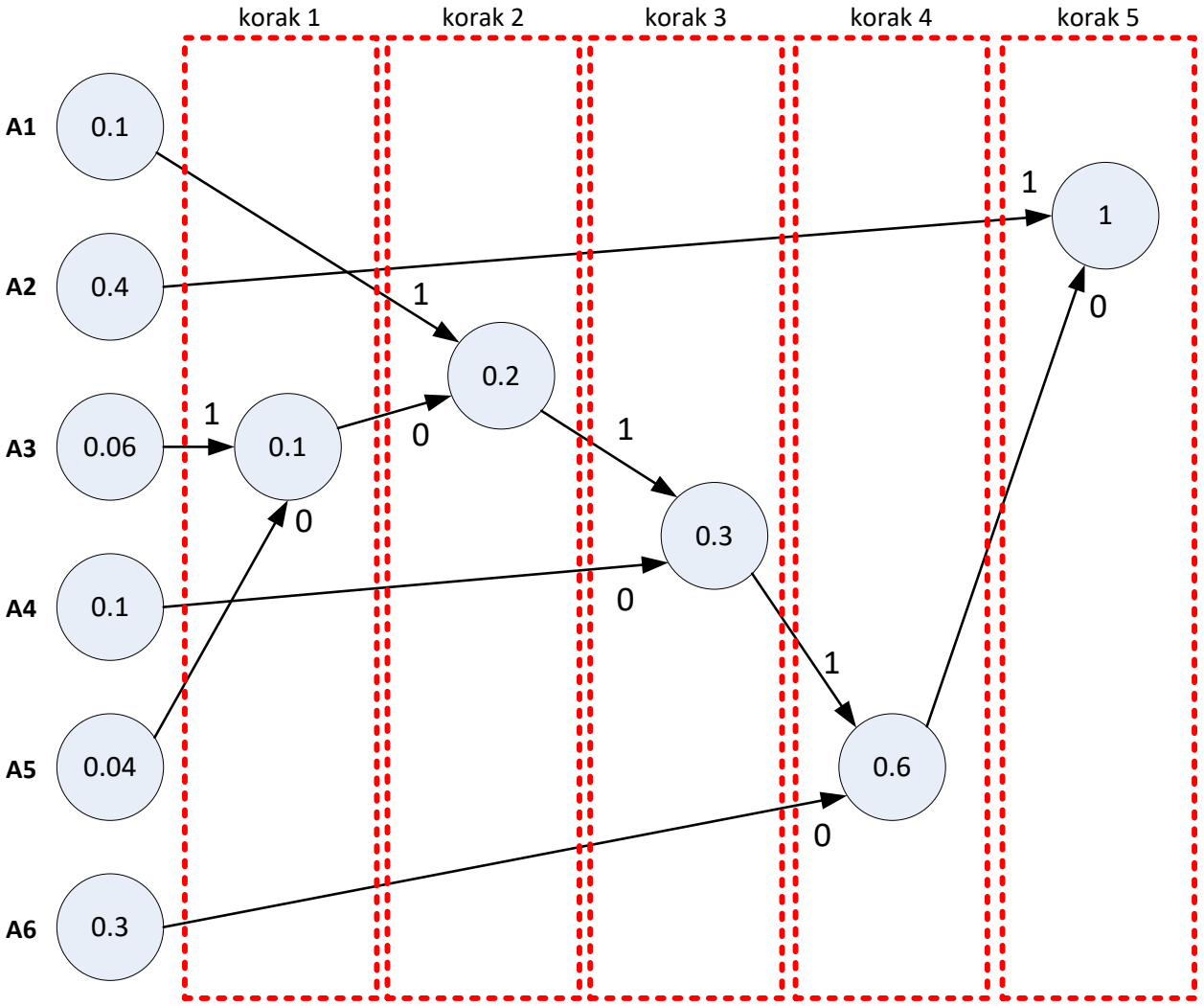
Kod Hafmanovog kodovanja svakom nivou se dodeljuje odgovarajuća kodna reč koje mogu biti različitih dužina i sve kodne reči se upisuju u jedan neprekidni niz bita. Kako bi se obezbedilo jedinstveno dekodovanje potrebno je obezbediti da nijedna kraća kodna reč ne predstavlja prefiks neke duže kodne reči.

Hafmanov kodni postupak

U prvom delu Hafmanovog kodovanja se obavlja niz redukcija simbola tako što se u svakom koraku dva simbola sa najmanjim verovatnoćama spajaju u novi simbol čija je verovatnoća pojavljivanja jednaka zbiru verovatnoća izvorišnih simbola.

Ovaj postupak se nastavlja sve dok ne preostanu samo 2 simbola.

U drugom delu Hafmanovog kodovanja svakom simbolu se dodeljuje odgovarajući kod krećući se unazad kroz stablo simbola.



simboli	verovatnoće	kodovi
A1	0.1	0111
A2	0.4	1
A3	0.06	01101
A4	0.1	010
A5	0.04	01100
A6	0.3	00

$$L_{avg} = 2.2$$

$$H = 2.14$$

Generisanje Hafmanove kodne tabele

```
load F
symbols = unique(F(:));
p = histc(F(:),symbols)./numel(F);

dict = huffman(p);

dict1 = huffmandict(symbols, p);
```

Najpre je potrebno odrediti skup simbola koji se koduje. Skup simbola predstavlja skup jedinstvenih elemenata od kojih će svakom biti dodeljene jedna Hafmanova kodna reč. Skup jedinstvenih elemenata neke matrice može se dobiti korišćenjem funkcije **unique**. Zatim se korišćenjem funkcije histc određuje broj ponavljanja svakog od ovih jedinstvenih elemenata. Drugi element histc funkcije predstavlja niz koji označava granice bin-ova histograma. Normalizacijom ovog histograma dobija se raspodela verovatnoće pojavljivanja jedinstvenih elemenata matrice.

huffmandict - ugrađena Matlab funkcija. Zahteva na ulazu niz koji predstavlja skup simbola kao i niz sa verovatnoćama. Povratni argument je ćelija koja sadrži ćeliju dimenzija 2xN, gde je N broj simbola koji se koduje. Svaki red ove ćelije sadrži vrednost simbola i niz koji predstavlja Hafmanov kod simbola.

Kreiranje kodnog stabla

```
s = cell(length(p), 1);

for i = 1:length(p)
   s{i} = i;
end

while numel(s) > 2
   [p, i] = sort(p);
   p(2) = p(1) + p(2);
   p(1) = [];

   s = s(i);
   s{2} = {s{1}, s{2}};
   s(1) = [];
end
```

Funkcija **reduce** obavlja redukciju simbola spajanjem simbola sa najmanjim verovatnoćama. Na taj način se kreira kodno stablo. Stablo u ovom primeru je predstavljeno pomoću ćelija. Na početku se kreira niz 1xN ćelija gde je vrednost svakog člana niza jednaka verovatnoći nekog od simbola koji se koduje.

Dok god nije potpuno završena redukcija simbola, odnosno dok god je broj simbola veći od 2, niz koji predstavlja verovatnoće trenutnih simbola se sortira u neopadajućem poretku. Povratni argument **i** za svaki element u sortiranom nizu sadrži njegovu poziciju u nesortiranom nizu. Ako se za nesortirani niz p obavi linearno indeksiranje nizom i, p(i) dobija se sortirani niz. Niz **i** se može iskoristiti za sortiranje niza simbola s, opet korišćenjem linearnog indeksiranja s(i).

Redukcija dva simbola sa najmanjim verovatnoćama obavlja se tako što se kreira novi simbol koji predstavlja skup ova dva simbola. Verovatnoća pojavljivanja tog novog simbola jednaka je zbiru verovatnoća pojavljivanja simbola od kojih je nastao. Pošto je od dva elementa sada nastao jedan potrebno je smanjiti nizove simbola i veorvatnoća za 1 element. Ovo se radi odsecanjem elementa sa indeksom 1 tako što mu se dodeljuje vrednost []. Ovim se on fizički briše iz niza.

Kreiranje kodne tabele

```
CODE = cell(length(p), 1);

if length(p) > 1
 p = p / sum(p);
 s = reduce(p);
 CODE = makecode(s, [], CODE);
else
 CODE = {'1'};
end

function CODE = makecode(sc, codeword, CODE)
if isa(sc, 'cell')
 CODE = makecode(sc{1}, [codeword 0], CODE);
 CODE = makecode(sc{2}, [codeword 1], CODE);
else
 CODE{sc} = char('0' + codeword);
end
```

Funkcija **huffman** kreira kodnu tabelu na osnovu niza verovatnoća pojavljivanja simbola. Izlaz ove funkcije predstavlja niz od N elemenata, gde je N broj simbola, pri čemu svaki element predstavlja string koji se sastoji od 0 i 1 i koji predstavlja kodnu reč određenog simbola.

U prvom delu ove funkcije kreira se kodno stablo korišćenjem prethodno opisane funkcije **reduce**.

Nakon toga se prolazi kroz kodno stablo i svakom listu stabla se dodeljuje odgovarajuća kodna reč. Ovo se obavlja rekurzivnim pozivima funkcije **makecode**.

U svakom pozivu funkcije **makecode** se ispituje da li je trenutni čvor list stabla, tako što se ispituje da li je taj element ćelija (sastoji se iz više drugih elemenata) ili običan skalar. U slučaju da je u pitanju ćelija onda se poziva funkcija **makecode** za njegovog levog i desnog suseda, pri čemu kod levog suseda predstavlja trenutni kod na koji se dodaje 0 dok kod desnog suseda predstavlja osnovni kod na koji je dodata 1.

U koliko se utvrdi da je trenutni čvor stabla list vrednost tog elementa predstavlja simbol koji se koduje a trenutna kodna reč predstavlja kod tog simbola. Kako je kodna reč niz 0 i 1, konvertuje se u string dodavanjem na svaki element niza vrednost ascii koda karaktera ‘0’.

Hafmanovo enkodovanje

```
load F
symbols = unique(F(:));
p = histc(F(:),symbols)./numel(F);

dict1 = huffmandict(symbols, p);
hcode1 = huffmanenco(F(:), dict1);

dict = huffman(p);
hcode = mat2huff(F);

disp([dec2bin(hcode.code(1),16)
      dec2bin(hcode.code(2),16)])

disp(char(hcode1' + '0'))
```

Za Hafmanovo enkodovanje niza može se iskoristiti ugrađena funkcija **huffmanenco**. Ova funkcija prihvata niz koje je potrebno enkodovati kao i kodnu tabelu koja sadrži kodnu reč za svaki simbol iz ulaznog niza. Izlaz predstavlja niz čiji elementi imaju vrednost 0 i 1 i koji predstavlja enkodovanu vrednost ulaznog niza.

Funkcija **mat2huff** za ulaznu matricu najpre kreira kodnu tabelu i zatim enkoduje svaki element matrice koristeći kodne reči iz ove tabele. Izlazni argument je struktura koja sadrži sve informacije neophodne za proces dekodovanja. Elementi izlazne strukture su niz **size** koji sadrži dimenzije ulazne matrice, skalar **min** minimalni element ulazane matrice, niz **hist** koji predstavlja histogram ulazne matrice, kao i niz **code** koji se sa sastoji od **uint16** elemenata koji predstavljaju enkodovanu vrednost ulazne matice. Obratite pažnju da se u ovom slučaju ne prosleđuje cela kodna tabela već se ona ponovo generiše prilikom dekodovanja na osnovu histograma polazne slike.

Hafmanovo enkodovanje

```
y.size = uint32(size(x));

x = round(double(x));
xmin = min(x(:));
xmax = max(x(:));
pmin = double(int16(xmin));
pmin = uint16(pmin + 32768);
y.min = pmin;

x = x(:)';
h = histc(x, xmin:xmax);
if max(h) > 65535
    h = 65535 * h / max(h);
end
h = uint16(h);   y.hist = h;
```

U okviru funkcije **mat2huff** najpre se odrede dimenzije ulazne slike. Kako ulazna slika može imati dosta velike dimenzije koristi se 32 bita za svaku dimenziju slike.

Funkcija **mat2huff** radi samo sa celim, označenim 16-bitnim brojevima. Na početku se određuju minimalna i maksimalna vrednost ulaznog niza. Minimalna vrednost će biti potrebna kasnije u procesu dekodovanja.

Nakon toga se računa histogram ulaznog niza za sve celobrojne vrednosti između minimalne i maksimalne. Obratite pažnju da se ovde računa pun histogram bez obzira što se neke od vrednosti uopšte ne pojavljuju u ulaznoj matrici. Čuvanje punog histograma sa jedne strane dovodi do dosta veće kodne tabele, pošto će sada u kodnoj tabeli biti i simboli koji se nikada ne koriste, jer se ne pojavljuju u ulaznom nizu. Međutim ovaj pristup omogućava jedinstveno dekodovanje na osnovu histograma i minimalne vrednosti ulaznog niza. U suprotnom je potrebno čuvati ili celu kodnu tabelu ili vrednosti simbola ulaznog niza.

Hafmanovo enkodovanje

```
map = huffman(double(h));

hx = map(x(:) - xmin + 1);
hx = char(hx)';
hx = hx(:)';
hx(hx == ' ') = [];

ysize = ceil(length(hx) / 16);

hx16 = repmat('0', 1, ysize * 16);
hx16(1:length(hx)) = hx;
hx16 = reshape(hx16, 16, ysize);

y.code = uint16(bin2dec(hx16'))';
```

Na osnovu histograma ulazne matrice kreira se Hafmanova kodna tabela. Potom se za svaki od ulaznih simbola određuje kodna reč i smešta u niz x. Minimalni element matrice odgovara prvom elementu kodne tabele, dok se pozicija koje odgovaraju ostalim simbolima ulazne matrice određuje na osnovu razlike vrednosti tog simbola i minimalnog elementa. Ovo je moguće uraditi pošto je kodna tabela napravljena za sve vrednosti između minimalnog i maksimalnog elementa a na samo za one koje se zapravo pojavljuju u zadatoj matrici.

Niz enkodovanih simbola predstavlja niz stringova i potrebno je najpre pretvoriti ih u jedan jedinstveni string, koji ne sadrži nikakve prazne karaktere.

Nakon toga se određuje broj 16 bitnih reči neophodnih za skladištenje enkodovanog niza. Dodaje se odgovarajući broj 0 na kraj kako bi ukupna dužina bila deljiva sa 16. Enkodovani niz karaktera se deli na N nizova od po 16 elemenata. Pozivom funkcije **bin2dec** se kreira N 16-bitnih brojeva koji predstavljaju kodiranu ulaznu matricu.

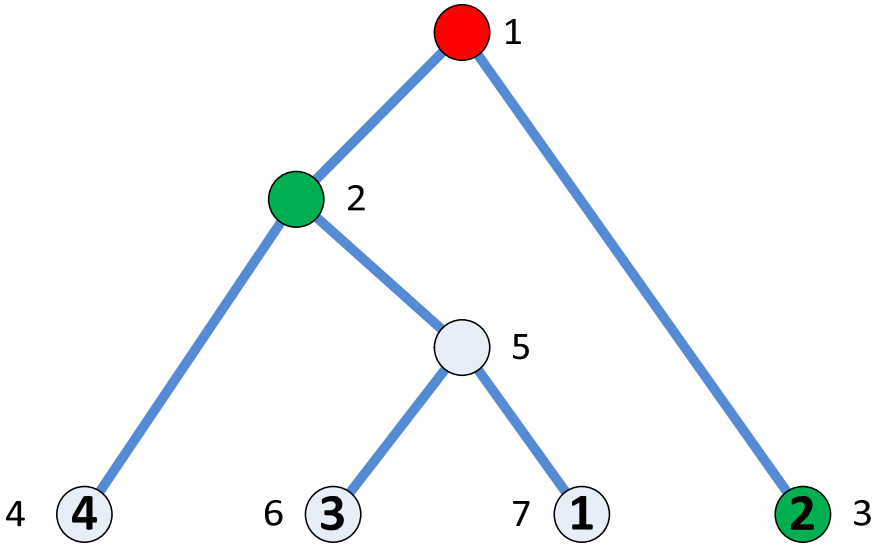
Hafmanovo dekodovanje

```
xmin = double(y.min) - 32768;
map = huffman(double(y.hist));

code = cellstr(char('', '0', '1'));
link = [2; 0; 0];   left = [2 3];
found = 0;   tofind = length(map);

while ~isempty(left) && (found < tofind)
  look = find(strcmp(map, code{left(1)}));
  if look
    link(left(1)) = -look;
    left(1) = [];
    found = found + 1;
  else
    len = length(code);
    link(left(1)) = len + 1;
    link = [link; 0; 0];
    code{end + 1} = strcat(code{left(1)}, '0');
    code{end + 1} = strcat(code{left(1)}, '1');
    left(1) = [];
    left = [left len + 1 len + 2];
  end
end
```

U toku Hafmanovog dekodovanja potrebno je čitati bit po bit iz kodovanog niza i proveravati da li trenutna kodna reč prestavlja validnu kodnu reč i kom simbolu pripada. Kako se prilikom Hafmanovog kodovanja ne dozvoljava da manja kodna reč predstavlja početak veće kodne reči to je proces dekodovanja jednoznačno određen. Ovaj problem ustvari predstavlja problem pretraživanja binarnog stabla koje se generiše u procesu redukcije simbola. Kretanje kroz stablo se određuje pristiglim bitima iz kodovanog niza, pri čemu vrednost 0 označava skretanje u desno a vrednost 1 skretanje u levo. Listovi ovog stabla predstavljaju dekodovane simbole, tako da se treba kretati kroz stablo sve dok se ne dođe do lista stabla. Time je otkrivena odgovarajuća vrednost simbola i ceo postupak se ponavlja za određivanje vrednosti novog simbola.

Kako bi se olakšao proces dekodovanja najpre je kreiran niz link koji omogućava efikasno kretanje kroz binarno stablo. Sam proces dekodovanja je urađen u C-u pomoću funkcije unravel.c pošto je rad sa bitima mnogo efikasniji u C-u nego u Matlabu.

Primer – kreiranje niza link

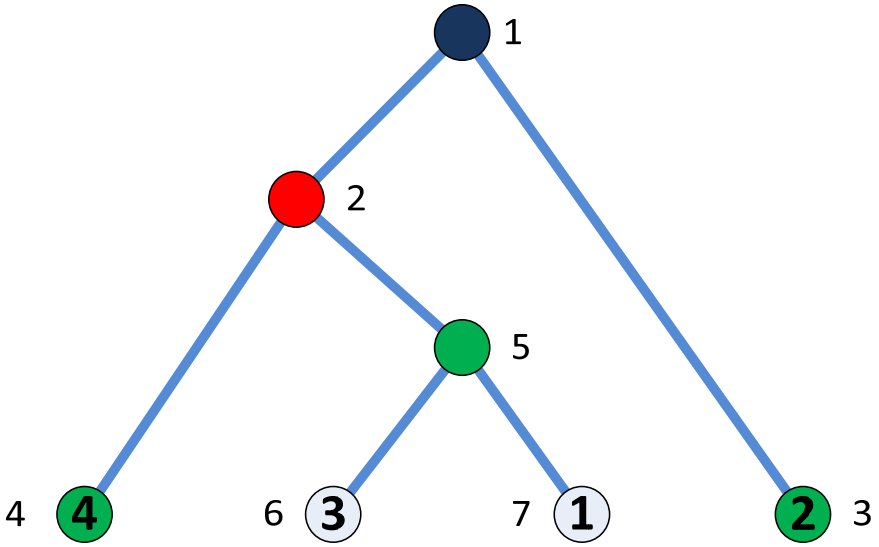


link = [2, 0, 0]

left = [2, 3]

code = {‘’, ‘0’, ‘1’}

Primer – kreiranje niza link

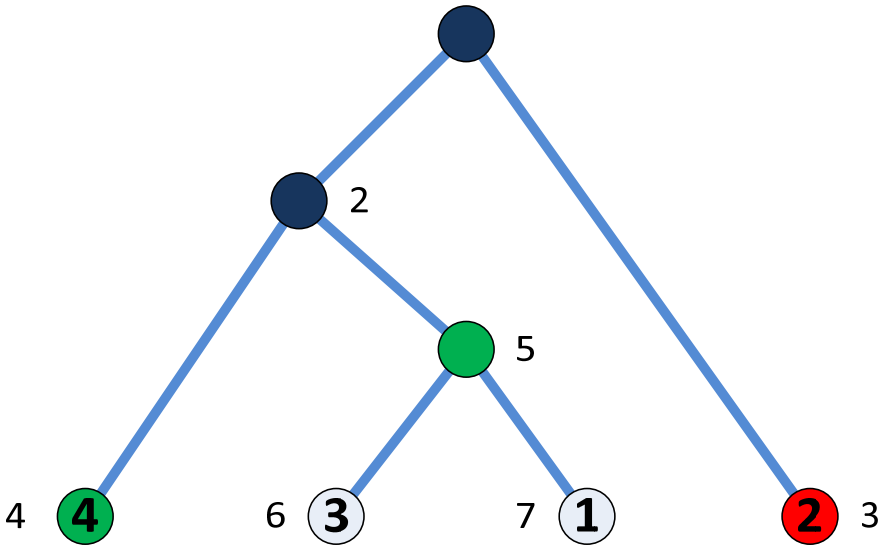


link = [2, 4, 0, 0, 0]

left = [3, 4, 5]

code = {‘’, ‘0’, ‘1’, ’00’, ‘01’}

Primer – kreiranje niza link

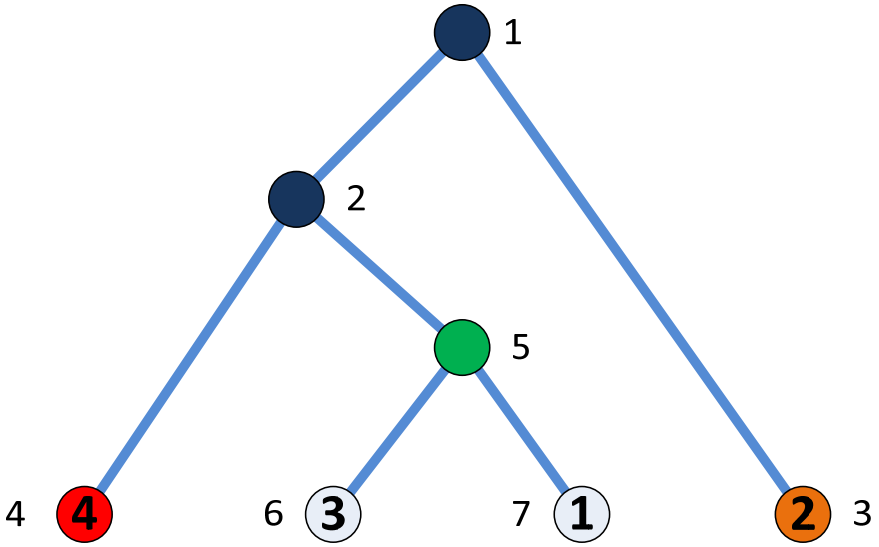


link = [2, 4, -2, 0, 0]

left = [4, 5]

code = {‘’, ‘0’, ‘1’, ’00’, ‘01’}

Primer – kreiranje niza link

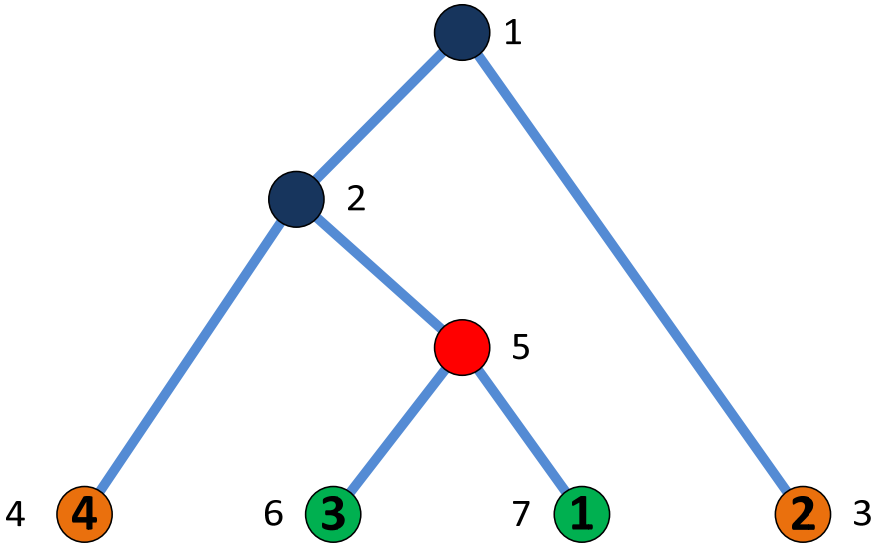


link = [2, 4, -2, -4, 0]

left = [5]

code = {‘’, ‘0’, ‘1’, ’00’, ‘01’}

Primer – kreiranje niza link

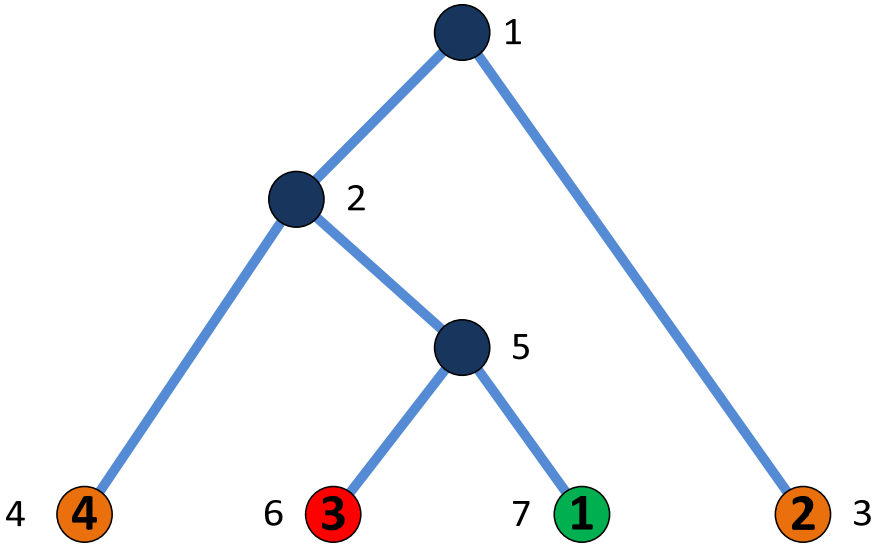


link = [2, 4, -2, -4, 6, 0, 0]

left = [6, 7]

code = {‘’, ‘0’, ‘1’, ’00’, ‘01’, ‘010’, ‘011’}

Primer – kreiranje niza link

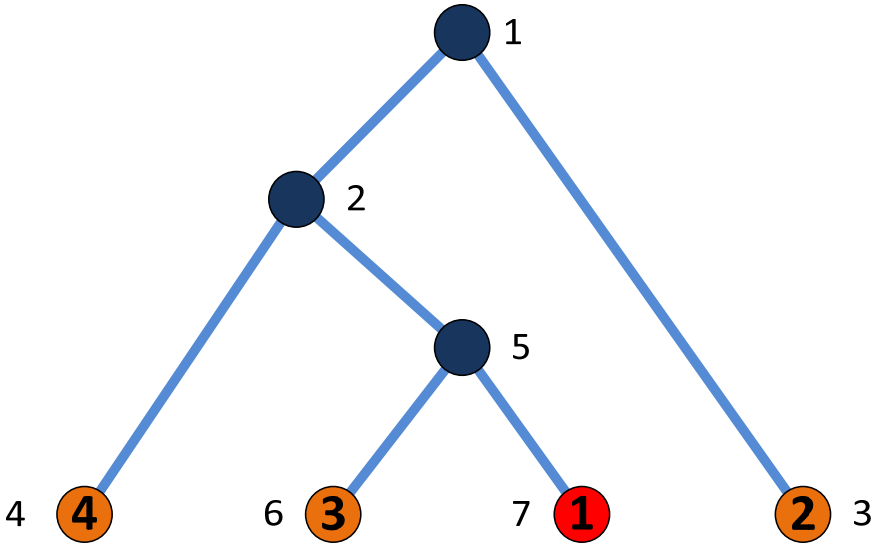


link = [2, 4, -2, -4, 6, -3, 0]

left = [7]

code = {‘’, ‘0’, ‘1’, ’00’, ‘01’, ‘010’, ‘011’}

Primer – kreiranje niza link

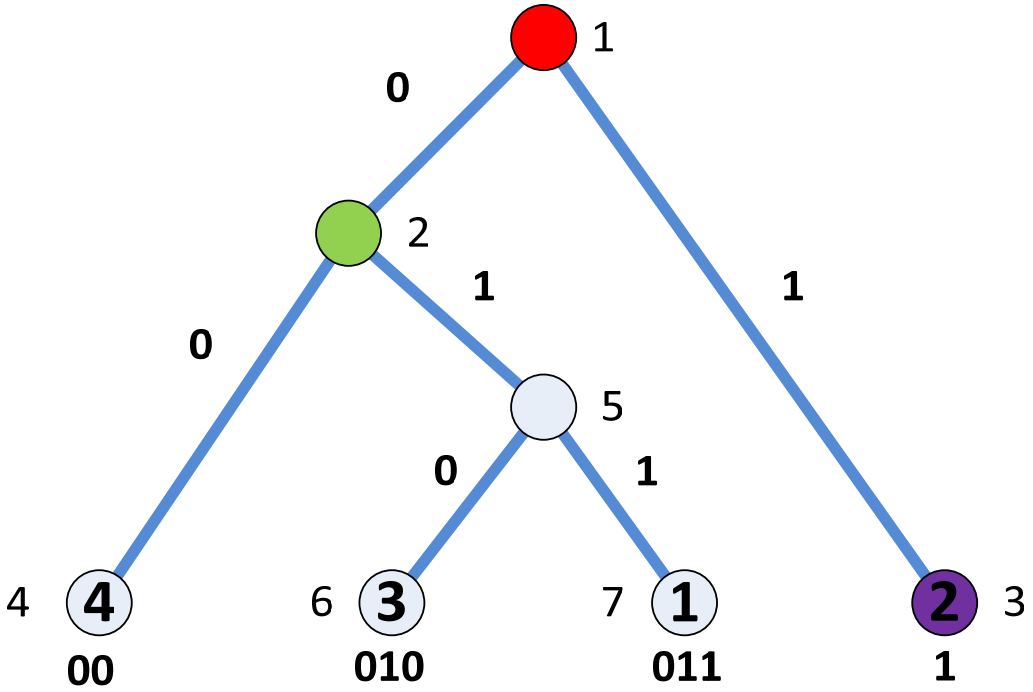


link = [2, 4, -2, -4, 6, -3, -1]

left = []

code = {‘’, ‘0’, ‘1’, ’00’, ‘01’, ‘010’, ‘011’}

Primer – dekodovanje korišćenjem niza link



Deo kodovane sekvence:

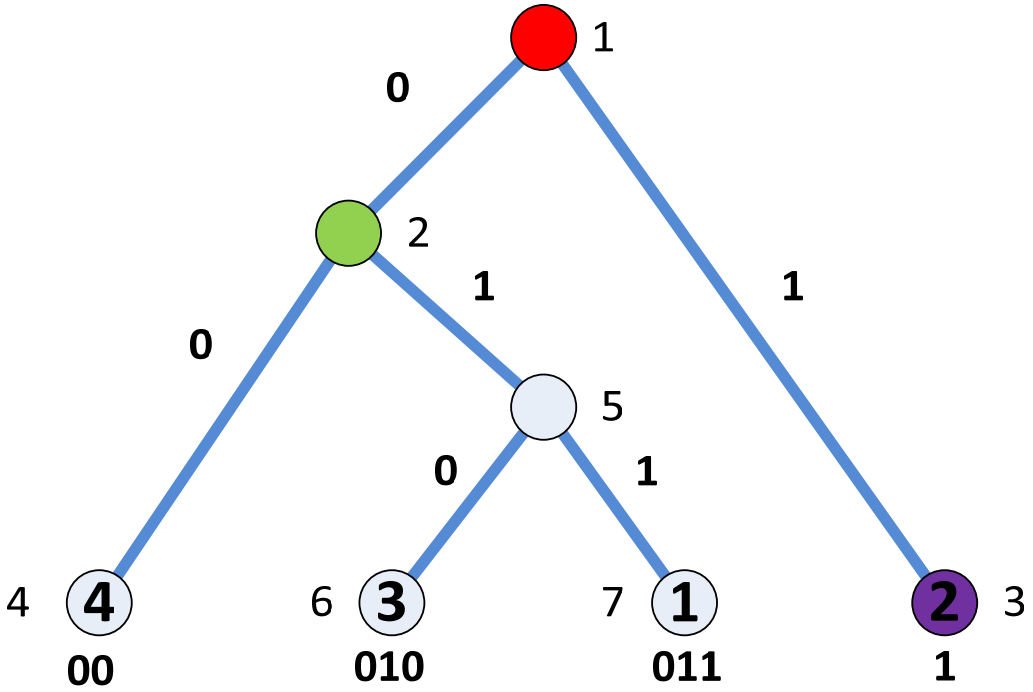
1010100

link = [2, 4, -2, -4, 6, -3, -1]

Trenutno dekodovana sekvenca:

2

Primer – dekodovanje korišćenjem niza link



Deo kodovane sekvence:

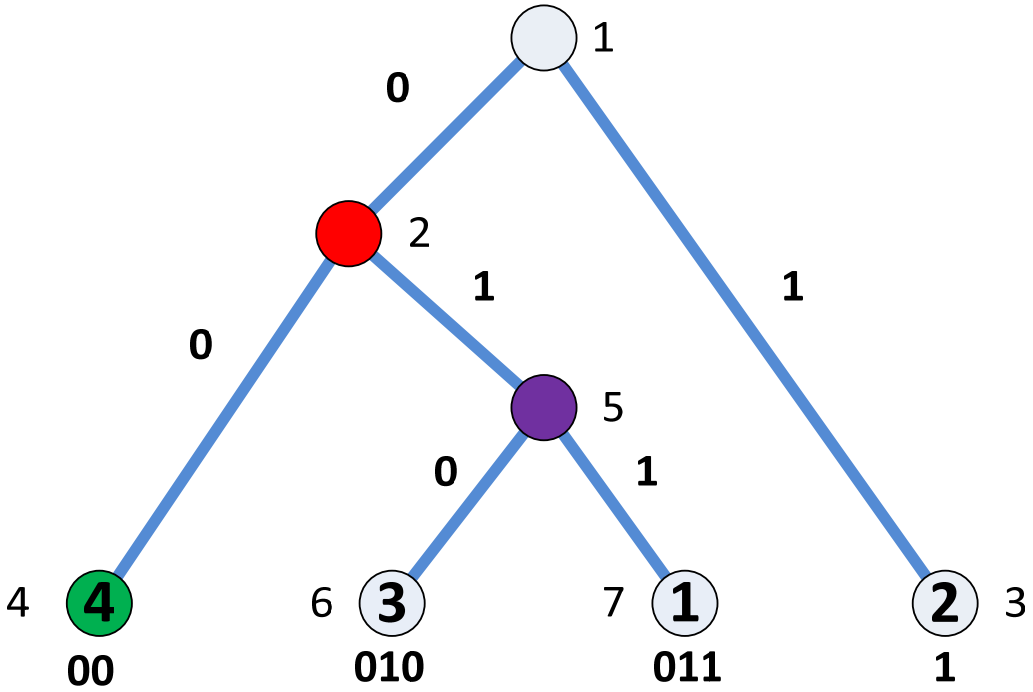
1010100

link = [2, 4, -2, -4, 6, -3, -1]

Trenutno dekodovana sekvenca:

2

Primer – dekodovanje korišćenjem niza link



Deo kodovane sekvence:

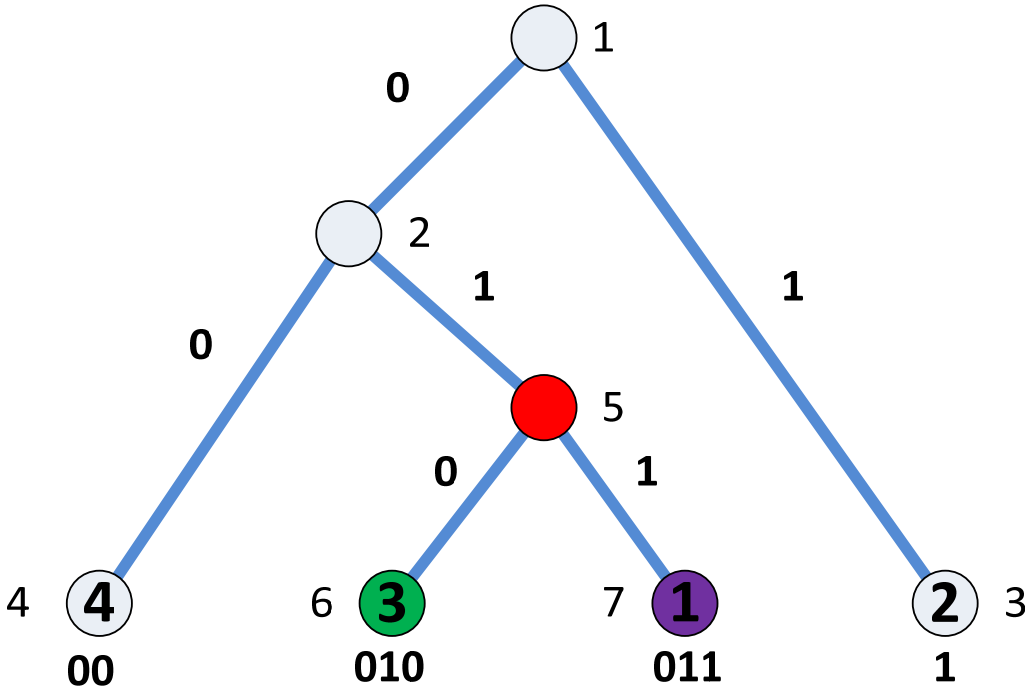
1010100

link = [2, 4, -2, -4, 6, -3, -1]

Trenutno dekodovana sekvenca:

2

Primer – dekodovanje korišćenjem niza link



Deo kodovane sekvence:

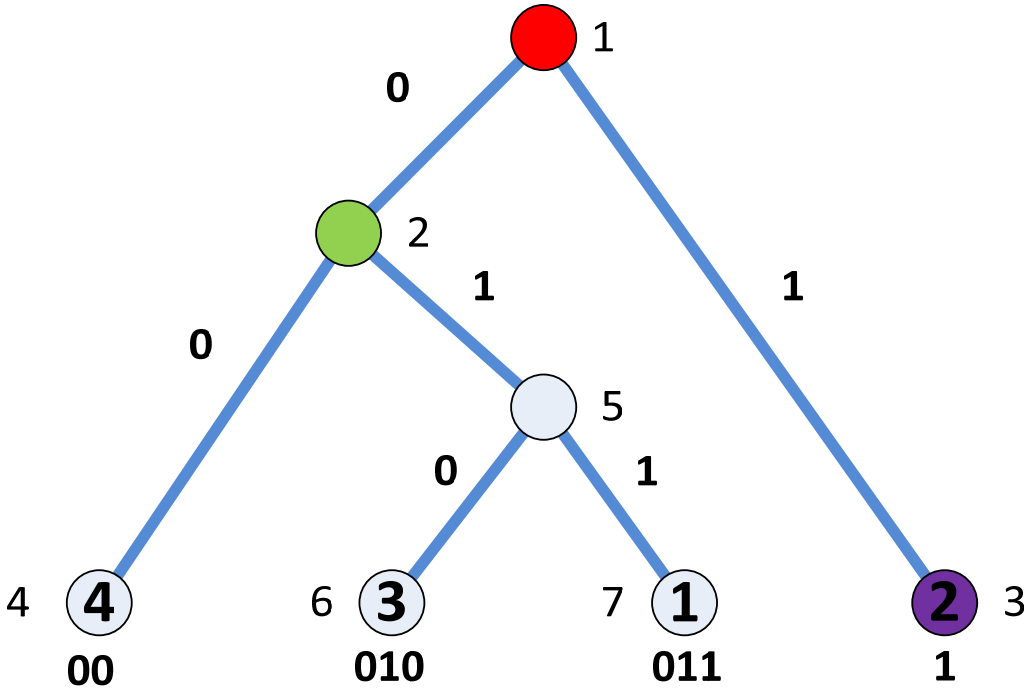
1010100

link = [2, 4, -2, -4, 6, -3, -1]

Trenutno dekodovana sekvenca:

23

Primer – dekodovanje korišćenjem niza link



Deo kodovane sekvence:

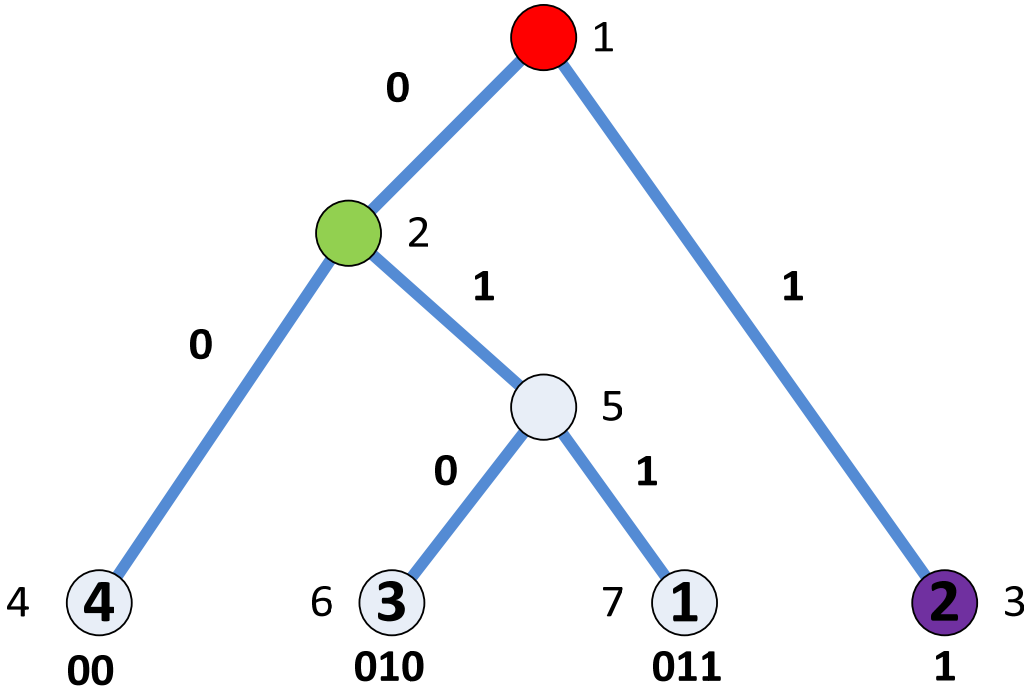
1010100

link = [2, 4, -2, -4, 6, -3, -1]

Trenutno dekodovana sekvenca:

232

Primer – dekodovanje korišćenjem niza link



Deo kodovane sekvence:

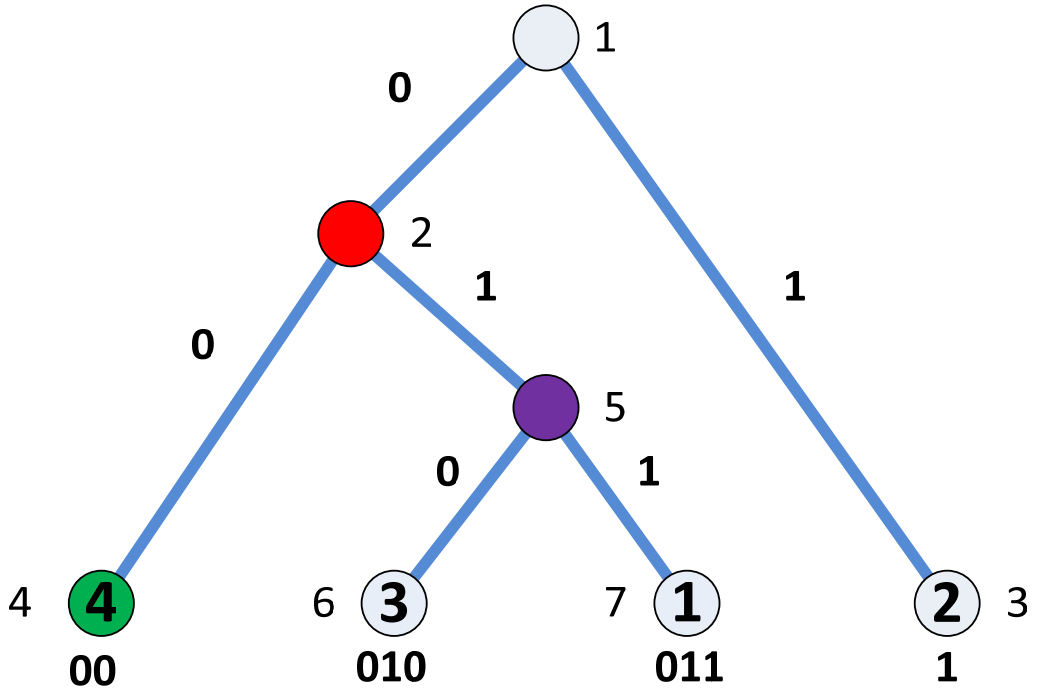
1010100

link = [2, 4, -2, -4, 6, -3, -1]

Trenutno dekodovana sekvenca:

232

Primer – dekodovanje korišćenjem niza link



Deo kodovane sekvence:

1010100

link = [2, 4, -2, -4, 6, -3, -1]

Trenutno dekodovana sekvenca:

2324

Dekodovanje korišćenjem C funkcije

Kako finalni proces dekodovanja na osnovu kodovanog niza i pomoćnog niza link zahteva čitanje ulaznog koda bit po bit, proces dekodovanja se mnogo efikasnije može obaviti u nekom nižem programskom jeziku poput C-a. Matlab dozvoljava poziv prekompajliranih C funkcija korišćenjem specijalnog interfejsa API.

Glavni C fajl iz koga se pozivaju ostale C funkcije, da bi se pozivao iz Matlab-a, mora uključiti fajl **mex.h** i definisati interfejsnu funkciju **mexFunction**. Parametri ove funkcije su broj izlaznih argumenata Matlab funkcije kao i pokazivač na listu argumenata, broj ulaznih argumenata Matlab funkcije kao i pokazivač na listu ulaznih argumenata.

Ulazni argumenti se dohvataju korišćenjem odgovarajućih **mxGet** funkcija. Za dohvatanje pokazivača se koristi **mxGetPr** dok se za dohvatanje skalara koristi funkcija **mxGetScalar**. Funkcijama **mxGetM** i **mxGetN** se mogu dohvatiti dimenzije ulazne matrice.

Izlazne matrice je potrebno alocirati unutar ove interfejsne funkcije pozivom odgovarajućih mex funkcija. Za alokaciju matrice realnih brojeva može se iskoristiti funkcija **mxCreateDoubleMatrix**.

Struktura mex fajla je opisana u dokumentu **Struktura MEX fajla.pdf** dok se više informacija o mex funkcijama može naći u dokumentu **cmex.pdf**.

Kreirajte MEX funkciju adds.c koja prihvata dva skalara i vraća treći skalar koji predstavlja njihov zbir.

Primer adds MEX funkcije

adds.c

```
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    double a, b, c;

    if (nrhs != 2)
        mexErrMsgTxt("Two inputs required.");
    else if (nlhs > 1)
        mexErrMsgTxt("Too many output arguments.");

    if (!mxIsDouble(prhs[0]) || !mxIsDouble(prhs[1]))
        mexErrMsgTxt("Input arguments must be a scalar.");

    a = mxGetScalar(prhs[0]);
    b = mxGetScalar(prhs[1]);

    c = a + b;

    plhs[0] = mxCreateDoubleScalar(c);
}
```

Matlab

```
mex adds.c
```
→ kreira **adds.mexw64**

```
c = adds(2,3)
```
→ poziva **adds** MEX funkciju

Dekodovanje korišćenjem C funkcije

Za dekodovanje u ovom primeru se koristi funkcija **unravel.c**. Ulazni argumenti ove funkcije su niz 16-bitnih brojeva koji predstavljaju kodovanu sekvencu, pomoćni niz link kojim se olakšava pretraga stabla i ukupan broj elemenata koji je potrebno dobiti nakon procesa dekodovanja.

U okviru funkcije **unravel** obavlja se proces dekodovanja opisan na prethodnim slajdovima. Obratiti pažnju da se u slučaju da je trenutni bit koji se dekoduje 0 skače na elementi link niza sa indeksom link[n]-1 a u slučaju da je trenutni bit 1 skače se na element sa indeksom link[n]. Ovo je posledica toga što indeksi u Matlabu kreću od 1 a u C-u od 0.

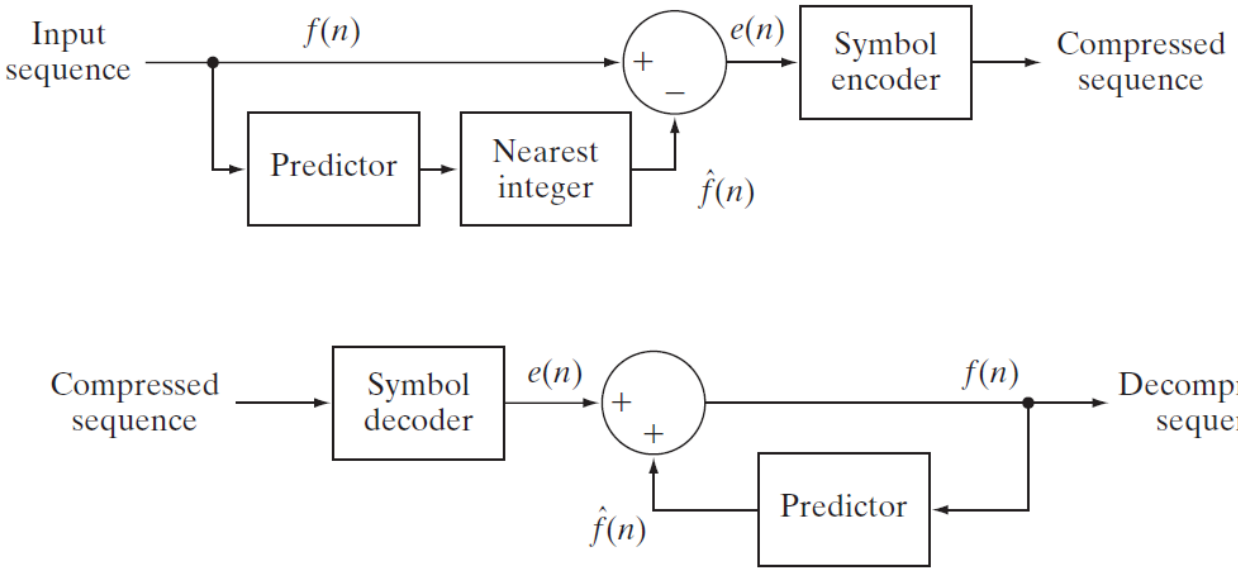
Kompajlirani MEX fajl se dobija pozivom komande **mex unravel.c** u Matlabovom komandnom porzoru. Ako nema grešaka u istom direktorijumu će se pojaviti fajl **unravel.mexw64** ili **unravel.mexw32** u zavisnosti od toga koji operativni sistem i varijanta Matlaba se koristi. Nakon toga se iz Matlaba može pozvati C-ovska funkcija **unravel**. Komentare koji su pisani u C-kodu nije moguće prikazati u matlabu korišćenjem help komande. Zbog toga se obično uz mex funkcije kreira i jedan **M** fajl sa istim imenom koji sadrži odgovarajuće komentare. Ovim se obezbeđuje da poziv **help unravel** ispisuje informacije o koriscenju ove funkcije.

Dekodovanje korišćenjem C funkcije

```
void unravel(unsigned short *hx,
double *link, double *x, double xsz,
int hxsz)
{
  int i = 15, j = 0, k = 0, n = 0;
  while (xsz - k) {
    if (link[n] > 0) {
      if ((hx[j] >> i) & 0x0001)
        n = link[n];
      else
        n = link[n] - 1;
      if (i > 0)
        i = i - 1;
      else
      {
        j = j + 1;
        i = 15;
      }
    }
    else {
      x[k] = -link[n];
      k = k + 1;
      n = 0;
    }
  }
}
```

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    double *link, *x, xsz;
    unsigned short *hx;
    int hxsz;

    if (nrhs != 3)
        mexErrMsgTxt("Three inputs required.");
    else if (nlhs > 1)
        mexErrMsgTxt("Too many output arguments.");

    if (!mxIsDouble(prhs[2]) || mxIsComplex(prhs[2]) || mxGetN(prhs[2]) * mxGetM(prhs[2]) != 1)
        mexErrMsgTxt("Input XSIZE must be a scalar.");

    hx = mxGetPr(prhs[0]);
    link = mxGetPr(prhs[1]);
    xsz = mxGetScalar(prhs[2]);

    hxsz = mxGetM(prhs[0]);
    plhs[0] = mxCreateDoubleMatrix(xsz, 1, mxREAL);
    x = mxGetPr(plhs[0]);
    unravel(hx, link, x, xsz, hxsz);
}
```

Prediktivno kodovanje

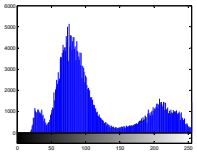


$$f(n) = e(n) + \hat{f}(n)$$

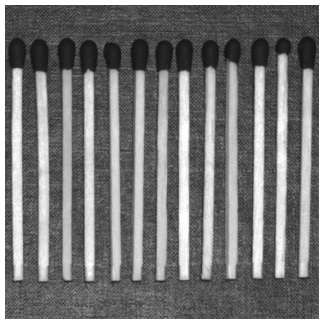
$$\hat{f}(n) = \text{round}\left[\sum_{i=1}^{m} \alpha_i f(n-i)\right]$$

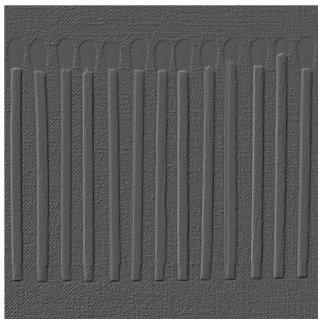
Prediktivno kodovanje

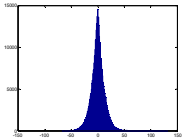
```
f = imread('matches_aligned.jpg');
figure; imshow(f);
entropy(f)

predictor = [1 1; 1 0]/3;
fe1 = predictive_coding(f, predictor);
figure; imshow(fe1, []);
entropy(fe1)

predictor = [1; 0];
fe2 = predictive_coding(f, predictor);
figure; imshow(fe2, []);
entropy(fe2)

predictor = [1 0];
fe3 = predictive_coding(f, predictor);
figure; imshow(fe3, []);
entropy(fe3)
```



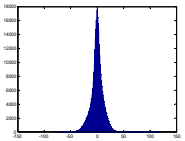
H = 7.3542



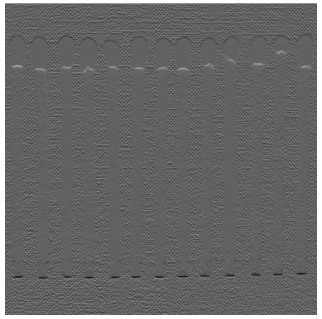


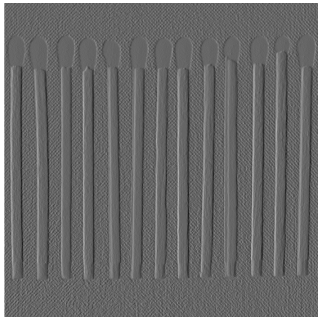


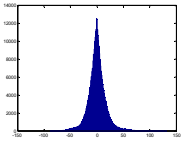
He1 = 5.6766



He2 = 5.4501







He3 = 5.9764

Testirajte prediktivno kodovanje za slike: lena.tif, star.tif, stripes.tif

Golomb kodovanje

Koristi se za skupove podataka ***x*** sa geometrijskom raspodelom

$$f_X(x) = (1-\beta)\beta^x, \frac{1}{2} < \beta < 1; x \geq 0$$

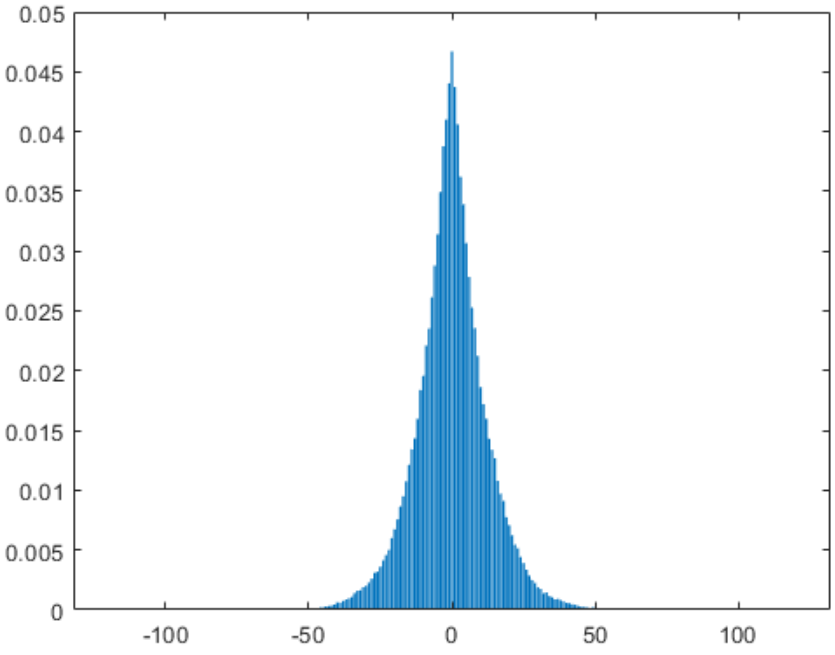
Ako se ***x*** predstavi kao $x = mx_q + x_r \quad x_q = \left\lfloor \frac{x}{m} \right\rfloor \quad x_r = x \bmod m$

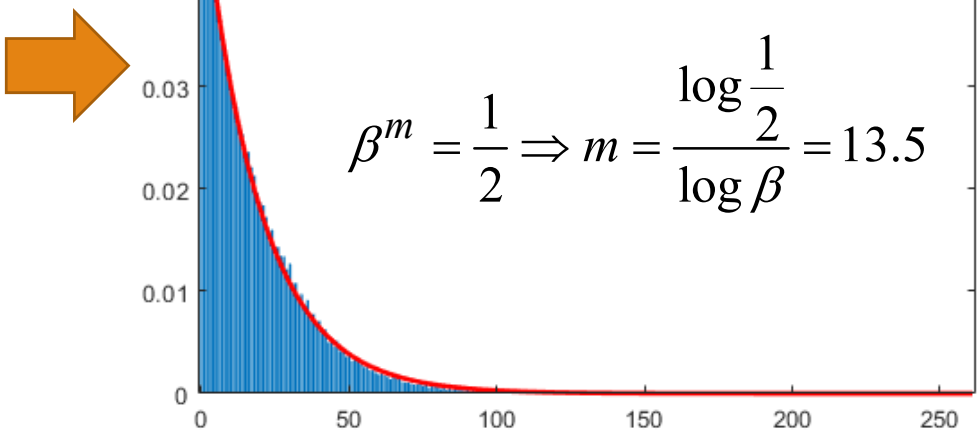
Definišu se sledeće pomoćne promenljive: $k = \lceil \log_2 m \rceil \quad c = 2^k - m$

Prvi deo kodne reči predstavlja x_q kodovano unarnim kodom dok se drugi deo formira na osnovu x_r na sledeći način:

$$x_r' = \begin{cases} x_r \text{ predstavljeno na } k-1 \text{ bita}, & 0 \leq x_r < c \\ x_r + c \text{ predstavljeno na } k \text{ bita}, & x_r \geq c \end{cases}$$

Golomb kodovanje





$$f_X(x) = (1-0.95)\cdot 0.95^x$$

$$\beta^m = \frac{1}{2} \Rightarrow m = \frac{\log\frac{1}{2}}{\log\beta} = 13.5$$

Golomb kodovanje

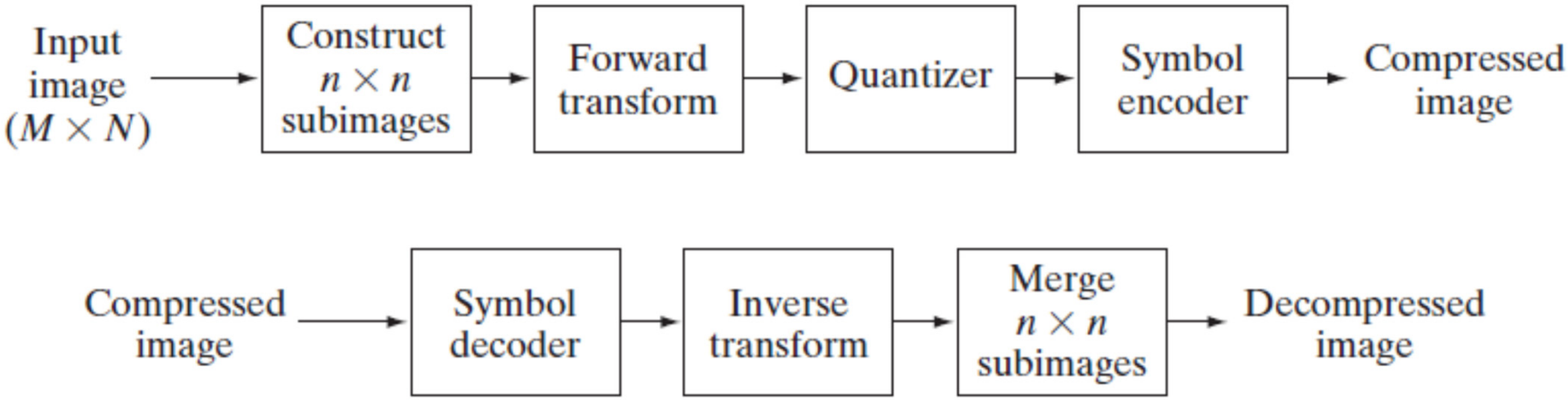
```
fe1_lim = max(abs(fe1(:)));
xe1 = -fe1_lim:1:fe1_lim;

he1 = histc(fe1(:), xe1)/numel(fe1);
figure; bar(xe1, he1);

fe1m(fe1>=0) = 2*fe1(fe1>=0);
fe1m(fe1<0) = -2*fe1(fe1<0)-1;
xe1m = 0:1:2*fe1_lim-1;

he1m = histc(fe1m(:), xe1m)/numel(fe1m);
figure; bar(xe1m, he1m);

beta = 0.95;
ye1m = (1-beta)*beta.^double(xe1m);
figure; bar(xe1m, he1m); hold on;
plot(xe1m, ye1m, 'r', 'LineWidth', 2);
```

```
entropy(fe1)

golomb_average_wordsize(he1m, xe1m, 12)
golomb_average_wordsize(he1m, xe1m, 13)
golomb_average_wordsize(he1m, xe1m, 14)

symbols = unique(fe1(:));
p = histc(fe1(:),symbols)./numel(fe1);
dict1 = huffmandict(symbols, p);
hcode1 = huffmanenco(fe1(:), dict1);
numel(hcode1)/numel(fe1)
```

Entropija = 5.6766

Hafman = 5.7103
Golomb = 5.7236

Transformaciono kodovanje

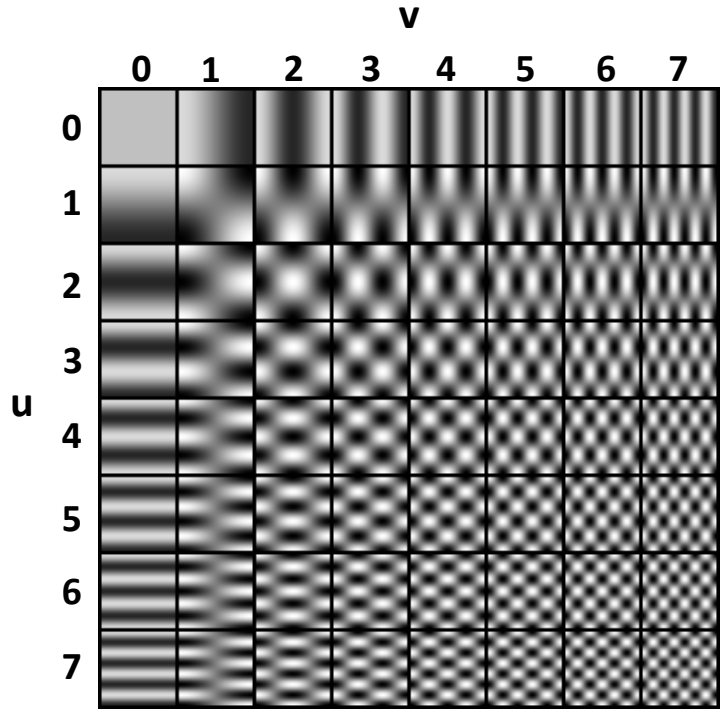


Transformaciono kodovanje

DCT $$T(u,v) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} g(x,y)r(x,y,u,v)$$

IDCT $$g(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} T(u,v)s(x,y,u,v)$$

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & ,u=0 \\ \sqrt{\frac{2}{N}} & ,u>0 \end{cases} \qquad r(x,y,u,v) = s(x,y,u,v) = \alpha(u)\alpha(v)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right]$$



Separabilnost DCT-a

$$r(x,y,u,v) = \alpha(u)\alpha(v)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right] = r_H(u,x)r_V(v,y)$$

$$r_H(u,x) = \alpha(u)\cos\left[\frac{(2x+1)u\pi}{2N}\right]$$

$$T(u,v) = \sum_{y=0}^{N-1}\sum_{x=0}^{N-1} g(x,y)r(x,y,u,v) = \sum_{y=0}^{N-1}\sum_{x=0}^{N-1} g(x,y)r_V(u,x)r_H(v,y) =$$

$$= \sum_{y=0}^{N-1} r_H(v,y)\sum_{x=0}^{N-1} g(x,y)r_V(u,x) = \sum_{y=0}^{N-1} r_H(v,y)T_V(u,y)$$

Matrično predstavljanje DCT-a

$$r_H(u,x) = r_V(u,x) = \alpha(u)\cos\left[\frac{(2x+1)u\pi}{2N}\right]$$

$r(u,x)$

$$\mathbf{R} = \begin{bmatrix} r(0,0) & r(0,1) & \cdots & r(0,N-1) \\ r(1,0) & r(1,1) & \cdots & r(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ r(N-1,0) & r(N-1,1) & \cdots & r(N-1,N-1) \end{bmatrix}$$

$g(x,y)$

$$\mathbf{G} = \begin{bmatrix} g(0,0) & g(0,1) & \cdots & g(0,N-1) \\ g(1,0) & g(1,1) & \cdots & g(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ g(N-1,0) & g(N-1,1) & \cdots & g(N-1,N-1) \end{bmatrix}$$

$T_V(u,y)$

$$\mathbf{T_V} = \begin{bmatrix} T_V(0,0) & T_V(0,1) & \cdots & T_V(0,N-1) \\ T_V(1,0) & T_V(1,1) & \cdots & T_V(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ T_V(N-1,0) & T_V(N-1,1) & \cdots & T_V(N-1,N-1) \end{bmatrix}$$

$$T_V(y,u) = \sum_{x=0}^{N-1} g(x,y)r(x,u)$$

$$\mathbf{T_V} = \mathbf{RG}$$

Matrično predstavljanje DCT-a

$T_V(u,y)$

$$\mathbf{T_V} = \begin{bmatrix} T_V(0,0) & T_V(0,1) & \cdots & T_V(0,N-1) \\ T_V(1,0) & T_V(1,1) & \cdots & T_V(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ T_V(N-1,0) & T_V(N-1,1) & \cdots & T_V(N-1,N-1) \end{bmatrix}$$

$r(v,y)$

$$\mathbf{R} = \begin{bmatrix} r(0,0) & r(0,1) & \cdots & r(0,N-1) \\ r(1,0) & r(1,1) & \cdots & r(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ r(N-1,0) & r(N-1,1) & \cdots & r(N-1,N-1) \end{bmatrix}$$

$T(u,v)$

$$\mathbf{T} = \begin{bmatrix} T(0,0) & T(0,1) & \cdots & T(0,N-1) \\ T(1,0) & T(1,1) & \cdots & T(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ T(N-1,0) & T(N-1,1) & \cdots & T(N-1,N-1) \end{bmatrix}$$

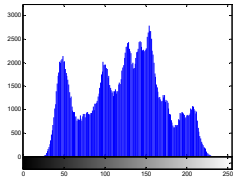
$$T(u,v) = \sum_{y=0}^{N-1} T_V(y,u) r(y,v)$$

$$\mathbf{T} = \mathbf{T_V R}^T = \mathbf{R G R}^T$$

Transformaciono kodovanje

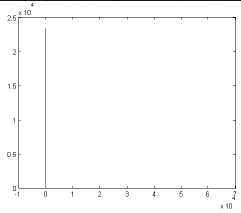
```
f = double(imread('lena.tif'));

R = dctmtx(512);
T = R*f*R';

figure; imshow(log(1+abs(T)), []);
[Th, Nh] = hist(T(:),
round(min(T(:))):round(max(T(:))));
figure; bar(Nh, Th);

g = R'*T*R;
max(abs(f(:) - g(:)))

entropy(f)
T = round(R*f*R');
entropy(T)

g = round(R'*T*R);
max(abs(f(:) - g(:)))

T1 = zeros(512,512);
T1(1:256, 1:256)= T(1:256, 1:256);
entropy(T)
g1 = round(R'*T1*R);
```

f

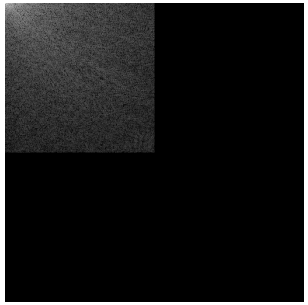


H = 7.4451

T



H = 5.0925



T1

H = 2.4451

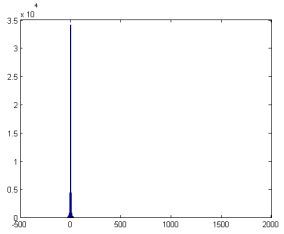
g1

PSNR = 36 dB

Transformaciono kodovanje

```
f = double(imread('lena.tif'));

R = dctmtx(8);
T = blockproc(f, [8 8], @(block)
round(R*block.data*R'));

figure; imshow(log(1+abs(T)), []);
[Th, Nh] = hist(T(:),
round(min(T(:))):round(max(T(:))));
figure; bar(Nh, Th);

entropy(f)
entropy(T)

Zm = zeros(8,8); Zm(1:4, 1:4) = 1;
T1 = blockproc(T, [8 8], @(block)
block.data.*Zm);
entropy(T1)

g1 = blockproc(T1, [8 8], @(block)
round(R'*block.data*R));

figure; imshow(log(1+abs(T1)), []);
figure; imshow(uint8(g1));
```

T



H = 4.7184

T1

H = 2.2926

g1

PSNR = 34.7 dB

Uprošćeni JPEG algoritam - kodovanje

```
load X
load Q
load zig_zag_index

X

R = dctmtx(8);
T = R*X*R'

Tq = round(T./Q)

C = Tq(zig_zag_index)
end_block_ind = find(C, 1, 'last');
C(1:end_block_ind)
```

X =

52	55	61	66	70	61	64	73
63	59	66	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	63	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94

Q =

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

T =

609.3750	-28.6143	-61.6030	24.4652	55.3750	-19.5434	-1.4077	2.8136
6.5734	-20.9729	-61.8988	8.6921	11.0372	-6.7768	-5.7813	6.2007
-45.7521	7.6895	76.5436	-25.1250	-29.9941	10.0455	6.8310	-5.1717
-49.7812	12.4514	34.7742	-15.4356	-8.9943	6.4305	0.2028	2.5192
11.3750	-8.1248	-12.7902	-1.1773	-1.1250	1.1935	-3.7672	0.7835
-9.4695	1.2715	3.3186	-3.0548	-0.6431	0.3676	2.0372	-0.5967
-3.6438	-0.5869	1.8310	-1.0564	1.7353	-3.2897	0.7064	-1.8142
-0.5108	-1.2846	-0.8846	-1.6751	-0.8249	-0.5986	0.0500	-0.4591

Tq =

38	-3	-6	2	2	0	0	0
1	-2	-4	0	0	0	0	0
-3	1	5	-1	-1	0	0	0
-4	1	2	-1	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

C = 38, -3, 1, -3, -2, -6, 2, -4, 1, -4, 1, 1, 5, 0, 2, 0, 0, -1, 2, 0, 0, 0, 0, 0, -1, -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

EOB

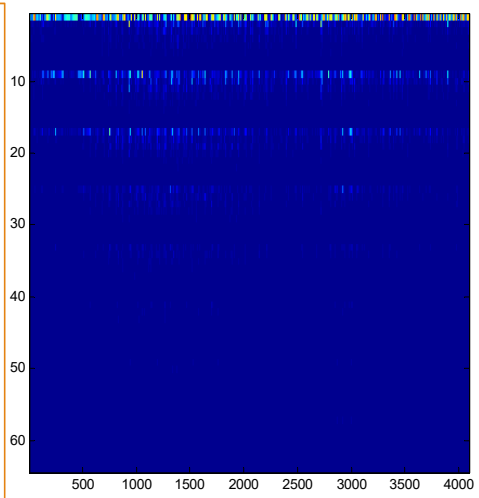
C = 38, -3, 1, -3, -2, -6, 2, -4, 1, -4, 1, 1, 5, 0, 2, 0, 0, -1, 2, 0, 0, 0, 0, 0, -1, -1, EOB

Niz C se dalje entropijski koduje – Hafmanov koder

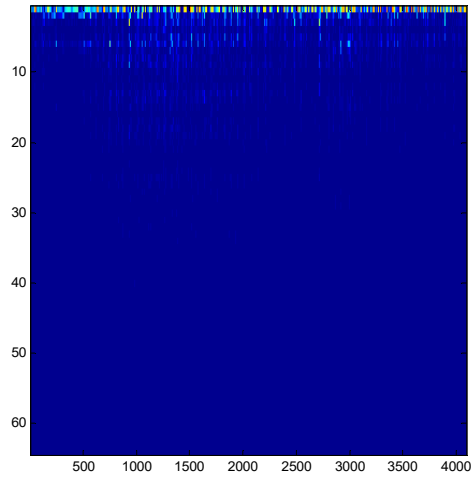
Uprošćeni JPEG algoritam - kodovanje

```
load Q
load zig_zag_index

I = double(imread('lena.tif'));
[height, width] = size(I);
I = I - 128;
R = dctmtx(8);

T = blockproc(I, [8 8], @(block)
R*block.data*R');
Tq = blockproc(T, [8 8], @(block)
round(block.data./Q));

Tq_col = im2col(Tq, [8 8],
'distinct');

block_num = size(Tq_col, 2);
Tq_col_zz = Tq_col(zig_zag_index, :);
```

```
eob = max(Tq_col_zz(:)) + 1;
r = zeros(numel(Tq_col_zz ) +
size(Tq_col_zz, 2), 1);
count = 0;
for j = 1:block_num
   i = find(Tq_col_zz(:, j), 1, 'last');
   if isempty(i)
      i = 0;
   end
   p = count + 1;
   q = p + i;
   r(p:q) = [Tq_col_zz(1:i, j); eob];
count = count + i + 1;
end

r((count + 1):end) = [];

huffman  = mat2huff(r);
```



Tq_col



Tq_col_zz

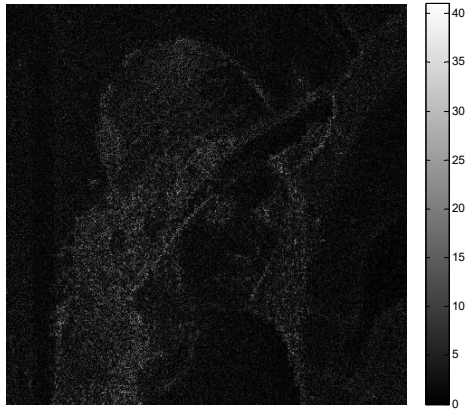
Uprošćeni JPEG algoritam - dekodovanje

```
load Q
load zig_zag_index

inv_zig_zag_index = zig_zag_index;
for k = 1:64
 inv_zig_zag_index(k) = find(zig_zag_index == k);
end

x = huff2mat(huffman);

eob = max(x(:));

Tq_col_zz_dec = zeros(64, block_num);
k = 1;
for j = 1: block_num
  for i = 1:64
    if x(k) == eob
      k = k + 1; break;
    else
      Tq_col_zz_dec(i, j) = x(k);
      k = k + 1;
    end
  end
end
```

```
Tq_col_dec = Tq_col_zz_dec(inv_zig_zag_index, :);
Tq_dec = col2im(Tq_col_dec, [8 8], [height width],
'distinct');
T_dec = blockproc(Tq_dec, [8 8], @(block)block.data.*Q);
R = dctmtx(8);
I_dec = blockproc(T_dec, [8 8], @(block)R'*block.data*R);
I_dec = uint8(I_dec + 128);
```

I

I_dec



abs(I_dec – i)

C = 11.4 **PSNR = 35.8 dB**

Manipulacija video fajlovima

```
shuttleVideo = VideoReader('shuttle360p.avi');

frame1 = rgb2gray(read(shuttleVideo, 1));
frame2 = rgb2gray(read(shuttleVideo, 2));

figure; imshow(frame1); figure; imshow(frame2);

frame_diff = double(frame2) - double(frame1);
figure; imshow(frame_diff, []);

outputVideo =
VideoWriter('shuttle360p_gray.avi', 'Grayscale
AVI');

outputVideo.FrameRate = shuttleVideo.FrameRate;
open(outputVideo);

for i = 1:shuttleVideo.NumberOfFrames
  frame = read(shuttleVideo, i);
  writeVideo(outputVideo, rgb2gray(frame));
end

close(outputVideo)
```

frame 1 | **frame 2** | **frame 2 – frame 1**

H = 2.94

frame 999 | **frame 1000** | **frame 1000 – frame 999**

H = 1.48