

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE  
FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE  
KATEDRA GEOMATIKY

název předmětu

**GEOINFORMATIKA**

číslo  
úlohy

název úlohy

1

**JPEG komprese a dekomprese rastru**

školní rok

studijní skup.

číslo zadání

Zpracovali:

datum

klasifikace

2025/26

-

Philip-Július Otto  
Mykhailo Radchenko

12.10  
2025

## TECHNICKÁ ZPRÁVA

### ZADÁNÍ:

Úkolem bylo implementovat algoritmus pro JPEG kompresi a rastru zahrnující tyto fáze:

- transformaci do  $YCbCr$  modelu,
- diskrétní kosinovou transformaci,
- kvantizaci koeficientů,

Kompresní algoritmus bylo nutné otestovat na různých typech rastru, kterými jsou: rastr v odstínech šedi, barevný rastr, vhodného rozlišení a velikosti s různými hodnotami faktoru komprese  $q = 10, 50, 70$ .

Pro každou variantu je zadáno vypočítat střední kvadratickou odchylku  $m$  jednotlivých  $RGB$  složek.

$$m = \sqrt{\frac{\sum_{i=0}^{m*n} (z - z')^2}{m*n}}.$$

Úkolem je též zhodnotit vhodnost jednotlivých rastrů pro JPEG kompresi.

### POPIS A ROZBOR PROBLÉMU:

#### JPG komprese

Nejprve je potřeba převést  $RGB$  (*red, green, blue*) do barevného prostoru  $YCbCr$  (*luma, blue and red difference chroma component*) pomocí definovaných konstant.

$$\begin{pmatrix} Y \\ C_B \\ C_R \end{pmatrix} = \begin{pmatrix} 0.2990 & 0.5870 & 0.1140 \\ -0.1687 & -0.3313 & 0.5000 \\ 0.5000 & -0.4187 & -0.0813 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

Dále bylo pro použití diskrétní kosinové transformace nutné vstupní rastr převzorkovat na submatice  $8 \times 8$ , které do ní budou postupně vstupovat. Každý tento blok je podroben zmiňované DCT transformaci, což vytváří množinu koeficientů, které reprezentují frekvenční charakteristiky bloku. DCT je definována rovnicí níže.

$$F(u, v) = \frac{1}{4} C(u) * C(v) \left[ \sum_{x=0}^7 \sum_{y=0}^7 f(x, y) * \cos \frac{(2x+1)u\pi}{16} * \frac{(2y+1)v\pi}{16} \right]$$

Kde

$$C(u) = \begin{cases} \frac{\sqrt{2}}{2}, & u = 0, \\ 1, & u \neq 0. \end{cases}$$
$$C(v) = \begin{cases} \frac{\sqrt{2}}{2}, & v = 0, \\ 1, & v \neq 0. \end{cases}$$

Výsledné koeficienty jsou zaokrouhleny, což je s kvantizací jeden ze ztrátových faktorů JPEG komprese. Dále je provedena samotná kvantizace pomocí definovaných kvantizačních matic. Kvantizace způsobuje, že vyšší frekvenční složky jsou kvantizovány s nižší přesností než nižší frekvenční složky. Použité kvantizační matice vypadají následovně:

$$F_Q(u, v) = \frac{F(u, v)}{Q(u, v)},$$

Kde  $Q(u,v)$  je rozdílné pro složky  $Y$  a  $C$ :

$$Q(u,v)_{50}^Y = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 87 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 26 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}$$

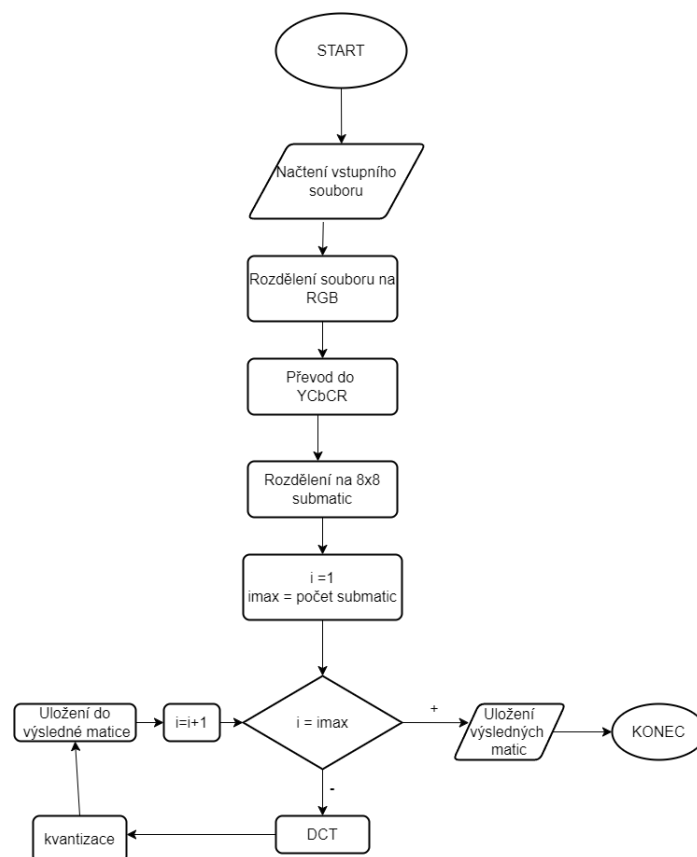
$$Q(u,v)_{50}^C = \begin{pmatrix} 17 & 18 & 24 & 47 & 66 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 69 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{pmatrix}$$

$$Q(u,v) = \frac{50 * Q(u,v)_{50}}{q}$$

Výsledkem jsou kompresované submatice pro jednotlivé kanály barevného prostoru  $YCbCr$ .

### IMPLEMENTACE:

JPEG komprese byla provedena v softwaru MATLAB dle následujícího vývojového diagramu:



Načtení vstupního souboru bylo provedeno pomocí funkce *imread* a zobrazen pomocí *imshow*:

```
input_image = imread('image2.bmp');  
imshow(input_image,[0 80]);
```

Vstupní rastrové soubory 8bitové barevné hloubky mají zvolené rozměry 128x128 pixelů pro urychlený proces výpočtu.

Takto načtený obrázek byl rozdělen na RGB složky:

```
R = double(input_image(:,:,1));  
G = double(input_image(:,:,2));  
B = double(input_image(:,:,3));
```

RGB bylo převedeno na YCbCr:

```
Y=0.2990*R+0.5870*G+0.1140*B;  
Cb=-0.1687*R-0.3313*G+0.5000*B+128;  
Cr=0.5000*R-0.4187*G-0.0813*B+128;
```

Poté byla vytvořena smyčka rozdělující matice na 8x8 pixelů, v níž byla provedena DCT, kvantizace a tvorba kompresovaných matic:

Samotná funkce provádějící DCT pojmenována *dct* přijímá na vstupu 8x8 matici a její výstup je transformovaná matice obsahující DCT koeficienty. Dvojitá vnořená smyčka (u, v) prochází všechny možné frekvenční koeficienty v 8x8 bloku. Pro každou hodnotu *u* a *v* jsou definovány váhy *Cu* a *Cv*. Pokud *u* a *v* jsou rovny 0, váhy jsou nastaveny jako polovina odmocniny ze dvou, jinak jsou nastaveny na 1. Další dvojitá vnořená smyčka pro *x* a *y* prochází všechny pixely v bloku. Pro každý pixel se vypočítá součet podle vzorce pro DCT. Výsledek se přičítá k proměnné *fu**v*. Vypočítaný součet je uložen jako hodnota odpovídajícího DCT koeficientu.

### **BONUSOVÉ ÚLOHY:**

#### **1. Resamplování rastru**

Tato funkce provádí tzv. 2×2 průměrování – tedy zmenší barevné rozlišení tím, že průměruje bloky po 2 pixelech na výšku i na šířku.

1) Vstup:

- Funkce dostane jednu barevnou složku (např. Cb nebo Cr) jako matici:
- Vytvoří novou (stejně velkou) matici, kam uloží průměrované hodnoty.

2) Smyčky po blocích 2×2

- Postupně prochází celou matici po **čtvercích 2×2** (tedy 4 sousední pixely).

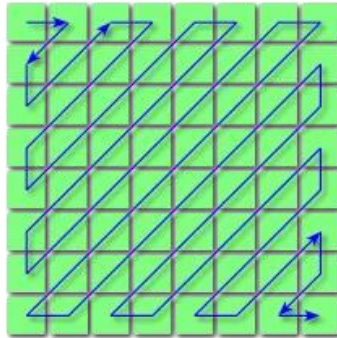
Každý symbol má svůj unikátní binární řetězec (např. 101, 00, 1101...).

3) Výpočet průměru

- Spočítá se průměr všech čtyř hodnot (`mean(block(:))`).
- Tento průměr se zapíše zpět do všech čtyř pozic toho bloku.
- Barevné informace (Cb a Cr) jsou **zjednodušené** – jemné detaily v barvách se ztratí, ale **luminance (Y)** zůstává ostrá.

## 2. Konverze pixelů do ZIG-ZAG sekvencí

Kompresované barevné složky  $Y C_B C_R$  je třeba převést z matic do jedné řady pomocí Zig-zag sekvence. Tím byla převedena 2D data na 1D. Funkce konverze je vyznačena na následujícím obrázku.



Postup funkce:

1) Inicializace proměnných:

- Získá rozměry vstupní matice  $[rows, cols] = size(M)$  pomocí `size` a přiřadí je do proměnných `rows` a `cols`.

2) Průchod přes diagonály:

- Proměnná `s` označuje **součet indexů řádků a sloupců**.

Každá hodnota `s` tedy odpovídá **jedné diagonále** matice.

Např.:

- `s = 0` → pozice (1,1)
- `s = 1` → diagonála obsahující (1,2) a (2,1)
- `s = 2` → diagonála obsahující (1,3), (2,2), (3,1), atd.

3) Směr pohybu se střídá:

- U **sudých** diagonál (`s = 0, 2, 4...`) se čte z **pravého horního rohu diagonály dolů doleva**.
- U **lichých** diagonál (`s = 1, 3, 5...`) se čte **opačně – zespodu nahoru doprava**.

Tím vzniká charakteristický „cik-cak“ (zig-zag) pohyb.

4) Ukládání hodnot

- Každý nalezený prvek se uloží do výstupního vektoru

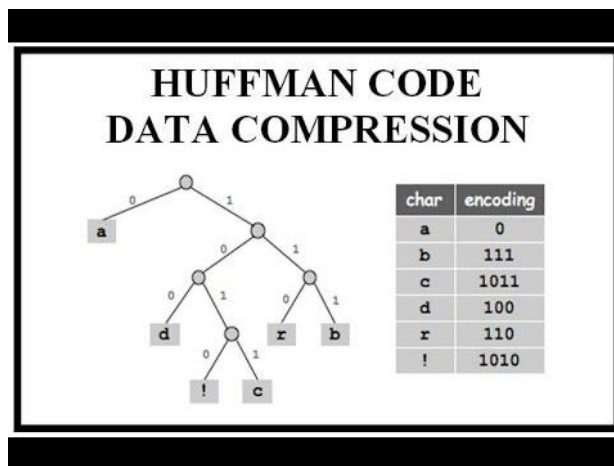
Po dokončení všech diagonál máš vektor `Z` se všemi 64 hodnotami v pořadí používaném v JPEG.

5) Inverse Zig-Zag

Funkce `inverse_zigzag(Z)` dělá přesný opak:

- Opět prochází diagonály ve stejném pořadí.
- Ale místo čtení do vektoru zapisuje hodnoty z `Z` zpět do  $8 \times 8$  matice.

### 3. Huffmanovo kódování



1) Vytvoření tabulky četností:

- Funkce *huffman\_codebook(data)* najde **unikátní symboly** (např. hodnoty kvantovaných DCT koeficientů).
- *symbols = unique(data)* Spočítá, **kolikrát se každý symbol v datech vyskytuje**.
- *counts(i) = sum(data == symbols(i))* Tím získá seznam všech symbolů a jejich četností.

```
newnode.count = tree(a).count + tree(b).count;  
newnode.left = a;  
newnode.right = b;
```

- Tento proces se **opakovaně** provádí, dokud nezůstane jeden kořenový uzel = celý strom.

2) Stavba Huffmanova stromu:

- Každý symbol se nejdřív uloží jako samostatný **uzel** se svou četností.
- Poté kód postupně **spojuje dva nejméně časté uzly** do nového „rodiče“, jehož četnost je součtem obou.

3) Generování kódů:

Pomocí vnořené funkce *traverse()* kód rekurzivně **prochází strom**:

- Každý **levý** krok přidá do kódu '1',
- Každý **pravý** krok přidá '0'.

Každý symbol má svůj unikátní binární řetězec (např. 101, 00, 1101...).

4) Kódování dat (*huffman\_encode*)

- Pro každý symbol z dat najde jeho odpovídající kód v codebooku
- Tyto kódy spojí do jednoho dlouhého **binárního řetězce** encoded, který představuje komprimovanou verzi bloku.

5) Dekódování dat (huffman\_decode)

- Při dekompresi se tento binární řetězec čte **bit po bitu**.
- Skript zkouší, zda aktuální úsek odpovídá nějakému kódu v codebooku
- Jakmile najde shodu, přidá odpovídající symbol do výstupního vektoru decoded.

#### 4. JPG dekomprese

JPG dekomprese postupuje po submaticích stejně jako JPG komprese.

```
Y_submatrix = Y_transformed(i:i+7, j:j+7);
Cb_submatrix = Cb_transformed(i:i+7, j:j+7);
Cr_submatrix = Cr_transformed(i:i+7, j:j+7);
```

Barevné kanály jsou uvnitř smyčky pro celý rastr jsou nejprve dekvantizovány:

```
Y_dequantized = round(Y_submatrix .* Y_quantization_matrix_Y);
Cb_dequantized = round(Cb_submatrix .* CbCr_quantization_matrix_C);
Cr_dequantized = round(Cr_submatrix .* CbCr_quantization_matrix_C);
```

Na dekvantizovaných kanálech je provedena inverzní diskretní kosinová transformace dle vztahů:

$$F(x, y) = \frac{1}{4} \left[ \sum_{u=0}^7 \sum_{v=0}^7 C(u) * C(v) f(u, v) * \cos \frac{(2x+1)u\pi}{16} * \frac{(2y+1)v\pi}{16} \right]$$

kde

$$C(u) = \begin{cases} \frac{\sqrt{2}}{2}, & u = 0, \\ 1, & u \neq 0. \end{cases}$$

$$C(v) = \begin{cases} \frac{\sqrt{2}}{2}, & v = 0, \\ 1, & v \neq 0. \end{cases}$$

Ta byla implementována funkcí *idct*, která funguje obdobně jako dříve popsaná funkce *dct*. Nejprve vnější smyčky pro *x* a *y* iterují přes každý pixel v 8x8 bloku. Vnitřní smyčky pro *u* a *v* iterují přes všechny možné frekvenční složky v 8x8 bloku. Koeficienty *Cu* a *Cv* se vypočítají na základě *u* a *v* obdobně jako u DCT. Výsledek se přičítá k proměnné *sumResult* a konečný výsledek je uložen v matici *resultingInverseTransform*.

Následně byl rastr převeden z barevného prostoru YCbCr do RGB dle:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.0091 & -0.0032 & 1.3955 \\ 1.0091 & -0.3472 & -0.7206 \\ 1.0091 & 1.7689 & -0.0066 \end{pmatrix} \begin{pmatrix} Y \\ C_B \\ C_R \end{pmatrix} - \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}.$$

Výsledný dekomprimovaný byl zobrazen.

**% Output image**

```
output_image = uint8(zeros(size(input_image)));
output_image(:, :, 1) = uint8(R_output);
output_image(:, :, 2) = uint8(G_output);
output_image(:, :, 3) = uint8(B_output);
```

```
% Display the output image
imshow(output_image);
```

### VÝPOČET STŘEDNÍCH KVADRATICKÝCH CHYB:

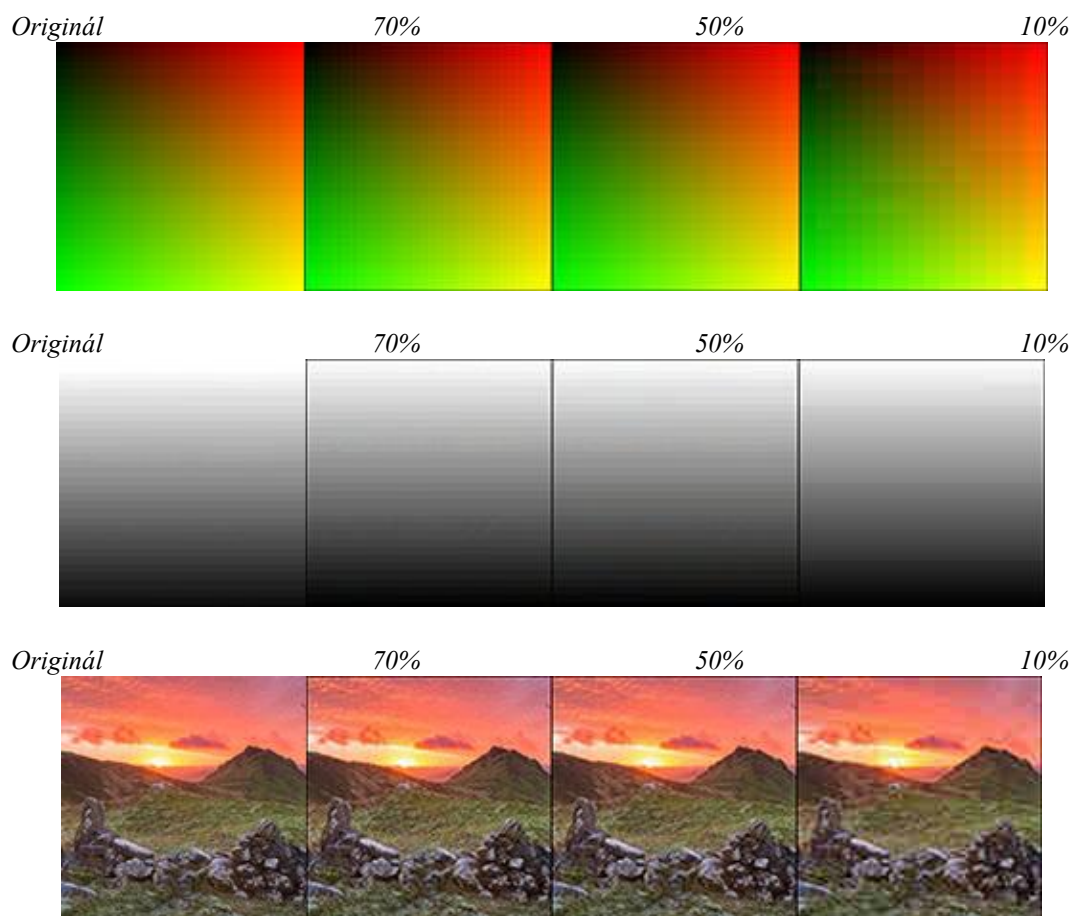
Z barevných složek RGB před a po kompresi jsou vypočteny čtverce jejich rozdílů.

```
delta_R = R_output - R;
delta_G = G_output - G;
delta_B = B_output - B;
delta_R_squared = delta_R .* delta_R;
delta_G_squared = delta_G .* delta_G;
delta_B_squared = delta_B .* delta_B;
```

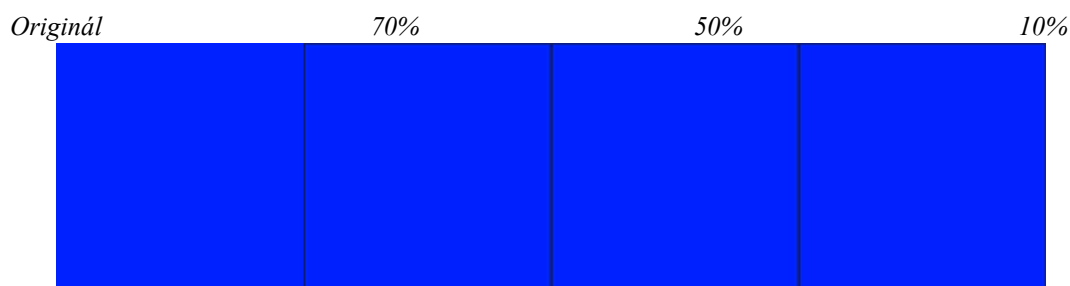
Z nich potom střední kvadratické chyby:

```
mean_R_error = sqrt(sum(sum(delta_R_squared)) / (m * n))
mean_G_error = sqrt(sum(sum(delta_G_squared)) / (m * n))
mean_B_error = sqrt(sum(sum(delta_B_squared)) / (m * n))
```

### VÝSTUPY:







#### STŘEDNÍ KVADRATICKÉ ROZDÍLY RGB SLOŽEK U JEDNOTLIVÝCH RASTRŮ:

Barevný přechod			
q	m R	m G	m B
10	6.6573	3.0777	4.9043
50	2.6738	1.4935	1.8217
70	2.1593	1.6433	1.6253

Fotografie			
q	m R	m G	m B
10	12.5580	11.4898	14.4605
50	6.9776	6.4008	7.4819
70	6.1592	5.4100	6.8481

Barva			
q	m R	m G	m B
10	2.1102	2.4256	1.2803
50	0.1310	0.0918	0.2800
70	0.3675	0.0855	0.5657

Škála ve stupních šedí			
q	m R	m G	m B
10	2.9888	2.9384	3.0505
50	1.7635	1.6171	1.8818
70	1.5601	1.4739	1.6373

#### ZÁVĚR:

Díky výše uvedených tabulek vidíme, že JPG komprese je nejvíce vhodná pro rastry o jedné barvě. Nejméně vhodná je pro rastry vektorové kresby, jelikož vektorový obrázek má ostré hrany s extrémními barevnými přechody, které se díky kompresi rozmazou. U barevných fotografií není vhodné volit faktor komprese nižší jak 70%-80%, jelikož potom obraz znatelně degraduje a ztrácí rozmanitost barev. Černobílé fotografie jsou na JPEG kompresi mnohem méně náchylné, jelikož jsou si původní pixely barevně blíže.

Pro ideální funkci je nezbytné správně zvolit faktor komprese, tak aby byl soubor po kompresi co nejmenší a zároveň bylo zachováno co nejvíce obrazových informací.