

# Kompresní algoritmy

# Kompresa

**Cíl komprese:** redukovat objem dat za účelem

přenosu dat

archivace dat

vytvoření distribuce sw

ochrany před viry

**Kvalita komprese:**

rychlost komprese

symetrie/asymetrie kompresního algoritmu

- Symetrické algoritmy – stejný čas potřebný pro kompresi i dekompresi
- Asymetrické algoritmy – čas potřebný pro kompresi a dekompresi se liší

kompresní poměr = poměr objemu komprimovaných dat k objemu dat nekomprimovaných

## Kompresce:

**bezztrátová** - po kódování a dekódování je výsledek 100% shodný,

- nižší kompresní poměr
- používají s výhradně pro kompresi textů a v případech, kdy nelze připustit ztrátu informace

**ztrátová** - po kódování a dekódování dochází ke ztrátě

- obvykle vyšší kompresní poměr než bezztrátové
- lze použít pouze v případech kdy ztráta je akceptovatelná (kompresce obrazů, zvuku)

## Metody komprese:

- **jednoduché** – založené na kódování opakujících se posloupností znaků (RLE)
- **statistické** – založené na četnosti výskytu znaků v komprimovaném souboru (Huffmanovo kódování, Aritmetické kódování)
- **slovníkové** – založené na kódování všech vyskytujících se posloupností (LZW)
- **transformační** – založené na ortogonálních popř. jiných transformacích (JPEG, waveletová komprese, fraktálová komprese)

# Jednoduché metody komprese

## RLE (Run Length Encoding) – kódování délkou běhu

- **Charakteristika:** bezztrátová metoda
- **Použití:** zřídka pro kompresi textů, častěji pro obrazovou informaci
- **Princip:** opakující se symboly se kódují dvojicí  
(počet\_opakování , symbol)

Kódování délky se provádí:

- přímo — u každého znaku je udán počet opakování

Př. Vstup: AAAABBCDDDDABD

Výstup: 4A2B1C4D1A1B1D

Nevýhoda: pokud se znaky neopakují často nedochází ke kompresi, ale naopak k prodloužení kódovaného souboru

- pomocí *escape* sekvencí – kódují se pouze opakující se sekvence delší než 3 znaky, kratší sekvence se zapisují přímo do výstupního souboru

Př. Vstup: AAAABBCDDDDABD

Výstup: #4ABBC#4DABD

Výhoda: neprodlužuje soubor, kde není co komprimovat to zůstane v původní podobě

Pozor !!! z množiny znaků je nutné vyčlenit symbol, který se nevyskytuje v komprimovaném souboru. Dále může nastat problém pokud je opakující se sekvence delší než 255 znaků (pokud kódujeme délku běhu na 8 bitech). Řešení závisí na konkrétní aplikaci

Použití RLE : např. obrazový formát BMP



# Slovníková metoda komprese

## LZW (Lempel-Ziv-Welch) metoda

Princip:

- vyhledávání opakujících se posloupností znaků, ukládání těchto posloupností do slovníku pro další použití a přiřazení jednoznakového kódu těmto posloupnostem.
- jednorůchodová metoda (nevyžaduje předběžnou analýzu souboru)
- Kódované znaky musí mít délku (počet bitů) větší než délka původních znaků (např. pro ASCII znaky (8 bitů) se obvykle používá nová délka znaků 12 bitů popř. větší.)
- Při průchodu komprimovaným souborem se vytváří slovník (počet položek slovníku odpovídá hodnotě  $2^{(\text{počet bitů nového kódu})}$ , kde prvních  $2^{(\text{počet bitů původního kódu})}$  položek jsou znaky původní abecedy a zbývající položky tvoří posloupnosti znaků obsažené v komprimovaném souboru.

# Algoritmus komprese a vytvoření slovníku

```
S := přečti znak ze vstupu;

while (jsou další znaky na vstupu) do
begin
C := přečti znak ze vstupu;
if S+C je v kódovací tabulce then
S := S+C
else begin
zapiš na výstup kód pro S
přidej do kódovací tabulky (S+C)
S := C
end;
end;
zapiš na výstup kód pro S;
```

Výsledný výstupní řetězec:

65 66 67 256 258 257 68 259

*Příklad:* Komprese řetězce ABCABCABCDABC

Postup kódování

S (prefix)	C (suffix)	výstup (kód)
A	B	A(65)
B	C	B(66)
C	A	C(67)
A	B	—
AB	C	AB(256)
C	A	—
CA	B	CA(258)
B	C	—
BC	D	BC(257)
D	A	D(68)
A	B	—
AB	C	—
ABC	—	ABC(259)

kód	posloupnost
0..255	jednotlivé znaky
256	AB
257	BC
258	CA
259	ABC
260	CAB
261	BCD
262	DA



# Algoritmus dekomprese a vytvoření slovníku

```
přečti OLD_CODE;
zapiš OLD_CODE na výstup;
while (na vstupu jsou další kódy) do
begin
  přečti NEW_CODE;
  if NEW_CODE není v kódovací tabulce then
  begin
    S := posloupnost zakódovaná kódem OLD_CODE;
    S := S+C;
  end
  else S := posloupnost zakódovaná kódem NEW_CODE;

  zapiš S na výstup;
  C := první znak S;
  přidej do kódovací tabulky (OLD_CODE+C);
  OLD_CODE := NEW_CODE;
end;
```

Vstupní řetězec:

65 66 67 256 258 257 68 259

OLD_CODE	NEW_CODE	S	C	Výstup
A(65)				A
A(65)	B(66)	B	B	B
B(66)	C(67)	C	C	C
C(67)	AB(256)	AB	A	AB
AB(256)	CA(258)	CA	C	CA
CA(258)	BC(257)	BC	B	BC
BC(257)	D(68)	D	D	D
D(68)	ABC(259)	ABC	A	ABC

Výstupní řetězec:

ABCABCBCDABC

Test existence NEW\_CODE možná vypadá zbytečně, ale existují případy ve kterých to bez tohoto testu nefunguje, např. u řetězce ABABABAB !!!!

Použití : často používaná metoda u textových i grafických souborů (např. PKZIP, ARJ, ZIP, TIFF, GIF)

# Statistické metody komprese

- Huffmanovo kódování
- Aritmetické kódování

# Huffmanovo kódování

- algoritmus navržen v Davidem Huffmanem v roce 1952
- využívá optimálního (nejkratšího) prefixového kódu (kód žádného znaku není prefixem jiného znaku).
- kódové symboly mají proměnnou délku

Princip: Metoda je založená na stanovení četnosti výskytů jednotlivých znaků v kódovaném souboru a kódování znaků s největší četností slovem s nejkratší délkou.

## Algoritmus kódování:

9. Zjištění četnosti jednotlivých znaků v kódovaném souboru
10. Vytvoření binárního stromu (Huffmanova kódu jednotlivých znaků)  
seřadíme posloupnost postupně zleva doprava doprava podle:
  - četnosti
  - podstrom bude vlevo před listem, větší podstrom před menším
  - pořadí v abecedě
11. Uložení stromu
12. Nahrazení symbolů jednotlivými kódy (posloupností bitů)

# Algoritmus vytvoření stromu

jednotlivé znaky označíme za vrcholy grafu (listy stromu) a dáme je do seznamu S

```
while (S.length>1) {  
    v S nalezneme dva vrcholy m, n s nejmenšími počty  
    výskytů  
    p = new Vrchol;  
    p.left = m; p.right = n;  
    p.count = m.count + n.count; // count je # výskytů  
    S.remove(m, n); S.add(p);  
}
```

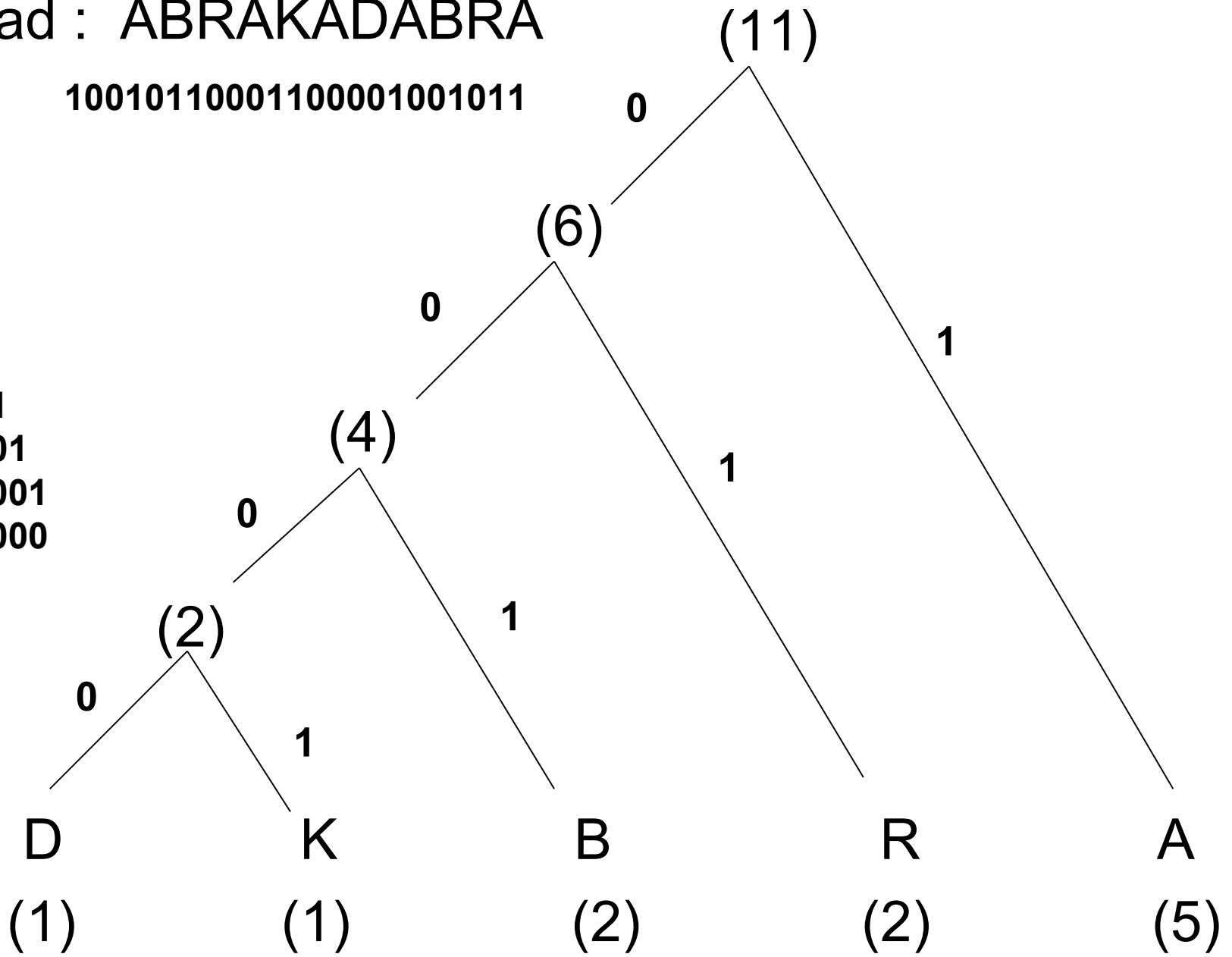
S obsahuje kořen stromu

najdeme kódy jednotlivých znaků (při průchodu z kořene do listu kódujeme 0 při kroku vlevo a 1 vpravo)

# Příklad : ABRAKADABRA

10010110001100001001011

A	1
R	01
B	001
K	0001
D	0000



# Dekomprese

1. Načtení a obnovení stromu, algoritmus je popsán při kompresi X
2. Vlastní dekomprese: Nahrazení kódů původními znaky.

```
v = vrchol stromu
while (!eof(input)) do {
    b = read bit
    if (b==0)
        v = v.left
    else v = v.right
    if (v je list) {
        write v.value // znak reprezentovany tímto
        listem
        v = vrchol stromu
    }
}
```

# Aritmetické kódování

- Statistická metoda
- Kóduje celou zprávu jako jedno kódové slovo ( v původní verzi číslo z intervalu  $[0,1)$ ).

Princip : Aritmetické kódování reprezentuje zprávu jako podinterval intervalu  $[0,1)$ . Na začátku uvažujeme celý tento interval. Jak se zpráva prodlužuje, zpřesňuje se i výsledný interval a jeho horní a dolní mez se k sobě přibližují. Čím je kódovaný znak pravděpodobnější, tím se interval zúží méně a k zápisu delšího (to znamená hrubšího) intervalu stačí méně bitů. Na konec stačí zapsat libovolné číslo z výsledného intervalu.

# Algoritmus komprese

1. Zjištění pravděpodobnosti  $P(i)$  výskytu jednotlivých znaků ve vstupním souboru
2. Stanovení příslušných kumulativních pravděpodobností  $K(0)=0$ ,  $K(i)=K(i-1)+P(i-1)$  a rozdělení intervalu  $<0,1)$  na podintervaly  $I(i)$  odpovídající jednotlivým znakům (seřazeným podle abecedy) tak, aby délky těchto intervalů vyjadřovaly pravděpodobnosti příslušných znaků:  $I(i) = <K(i), K(i+1))$
3. Uložení použitých pravděpodobností
4. Vlastní komprese

začínáme s intervalem  $I=<0,1)$ : označme jeho dolní mez  $D(I)$ , horní  $H(I)$  a délku intervalu  $L(I)=H(I)-D(I)$

```
while (!eof) {  
    read(i)  
    I = <D(I) + K(i) * L(I) , D(I) + K(i+1) * L(I)  
}  
write(D(I))
```





# Příklad kódování

Vstup: **a**       $L = 0$   
                  $H = 0 + 0.5 \cdot 1 = 0.5$

**b**       $L = 0 + 0.5(0.5 - 0) = 0.25$   
                  $H = 0 + 0.5(0.5 - 0) + 0.25(0.5 - 0) = 0.375$

**a**       $L = 0.25$   
                  $H = 0.25 + 0.5(0.375 - 0.25) = 0.3125$

**a**       $L = 0.25$   
                  $H = 0.25 + 0.5(0.3125 - 0.25) = 0.28125$

**b**       $L = 0.25 + 0.5(0.28125 - 0.25) = 0.265625$   
                  $H = 0.25 + 0.5(0.28125 - 0.25) + 0.25$   
                                  $(0.28125 - 0.25) = 0.2734375$

**c**       $L = 0.265625 + 0.5(0.2734375 - 0.265625) + 0.25$   
                  $(0.2734375 - 0.265625)$   
                                  $= 0.271484375$   
                  $H = 0.265625 + 0.5(0.2734375 - 0.265625) + 0.25$   
                                  $(0.2734375 - 0.265625) + 0.125$        $0.25(0.2734375$   
                                  $0.265625) = 0.2724609375$

Atd.

# Dekomprese

1. Rekonstrukce použitých pravděpodobností

3. Vlastní dekomprese

```
read(X) přečteme uložené reálné číslo
while (není obnovena celá zpráva) {
    najdeme i, aby X bylo v [K(i), K(i+1))
    write(i)
    X=(X-K(i))/P(i)
}
```

# Ztrátové komprese

- JPEG
- Waveletová komprese
- Fraktálová komprese

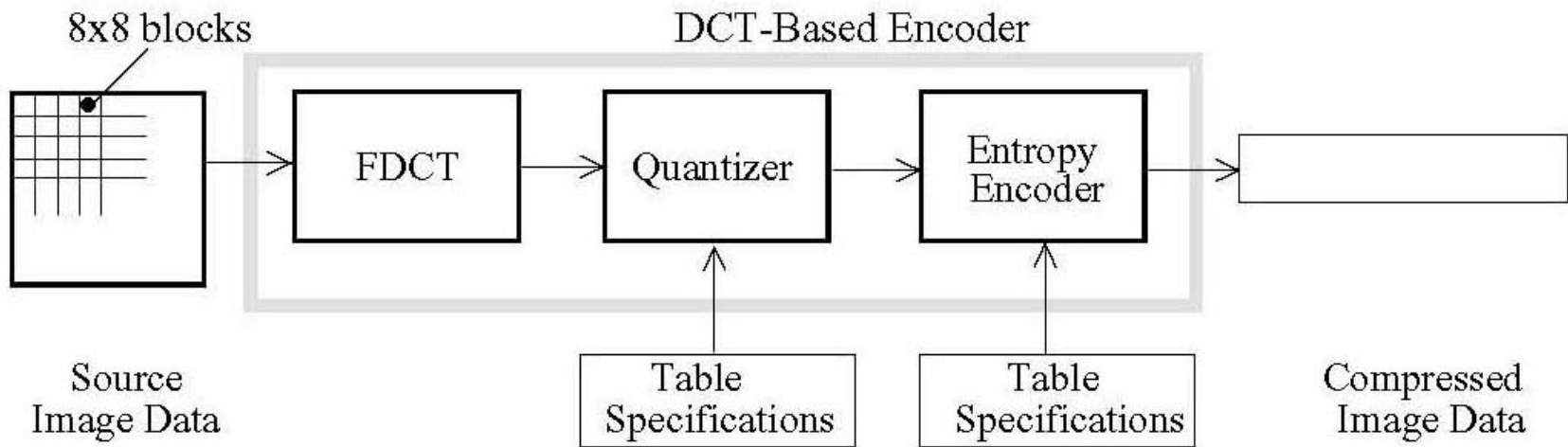
# JPEG (Joint Photographic Experts Group)

- v současné době patří mezi nejvíce používané komprese u obrázků
- je vhodná pro komprimaci fotek, nevhodná pro např. technické výkresy (čarové výkresy) – dochází k viditelnému rozmazání

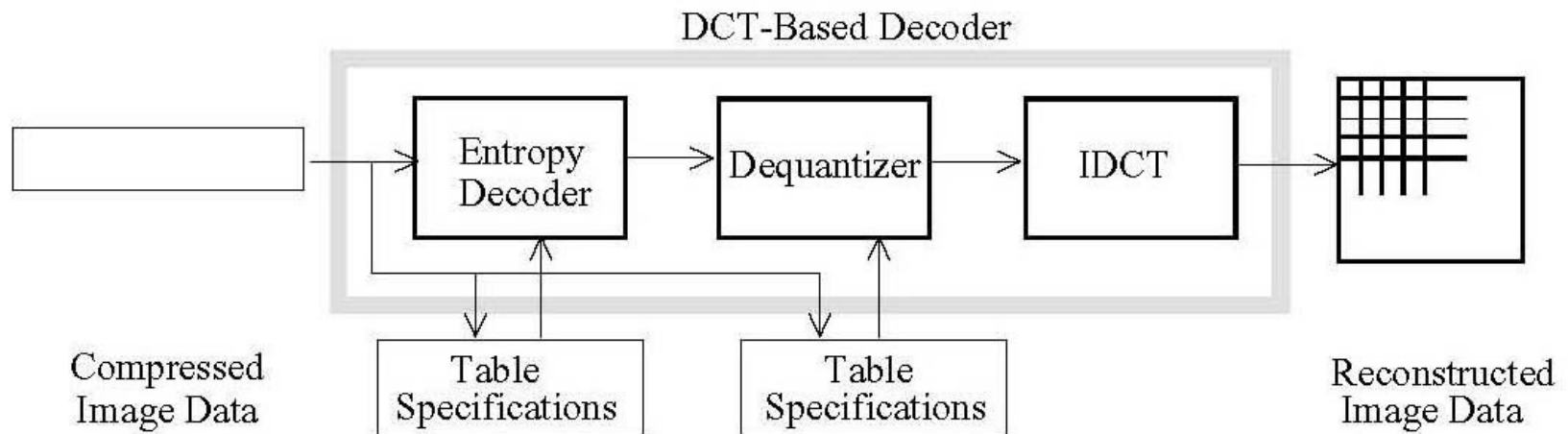
## Princip:

- části obrazu se transformují do frekvenční oblasti (výsledkem je matice „frekvenčních“ koeficientů
- z matice koeficientů se odstraní koeficienty odpovídající vyšším frekvencím ( rychlejší změny jasu – např. hrany v obraze)
- zbývající koeficienty se vhodným způsobem zkomprimují

# Blokové schéma Jpeg komprese (kodér)



# Blokové schéma Jpeg komprese (dekodér)



# DCT – Diskrétní kosinová transformace

- transformuje kódovanou oblast do frekvenční oblasti
- je bezztrátová a existuje k ní inverzní transformace

Postup:

5. Zdrojový obraz se nejprve rozdělí na bloky 8x8 pixelů
6. Hodnoty jasu v každém bloku se nejprve transformují z intervalu  $[0, 2^p-1]$  na interval  $[2^{p-1}, 2^{p-1}-1]$
7. Proveďte se diskrétní kosinová transformace podle vztahu:

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

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

$$C(u), C(v) = \frac{1}{\sqrt{2}}, \text{ pro } u, v = 0$$

$$C(u), C(v) = 1, \quad \text{pro } u, v$$



# Kvantizace / dekvantizace

- V tomto kroku se každý z 64 koeficientů DCT (IDCT) vydělí (vynásobí) odpovídajícím prvkem kvantizační matice a zaokrouhlí na nejbližší celé číslo. V tomto kroku dochází ke ztrátě informace !!!!!

kvantizace

$$F^Q(u, v) = \text{Integer} \left( \frac{F(u, v)}{Q(u, v)} \right)$$

dekvantizace

$$F^{Q'}(u, v) = F^Q(u, v) \cdot Q(u, v)$$

$Q_{50} =$

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

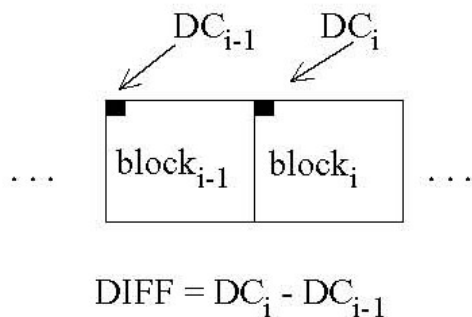
$$Q_q = \frac{Q_{50}(100 - q)}{50} \quad \text{pro } q \in (50, 100)$$

$$Q_q = \frac{50 \cdot Q_{50}}{q} \quad \text{pro } q \in (0, 50)$$

# Kódování DCT koeficientů

- Koeficienty DCT se obvykle kódují pomocí statistických metod (Huffmann, aritmetické kódování)
- Koeficient v pozici (0,0) je označen jako DC koeficient (stejnosečná složka), ostatní se označují jako AC koeficienty
- Vzhledem k tomu že DC koeficienty sousedních bloků jsou obvykle silně korelované (tj. střední hodnota jasů sousedních bloků je podobná) kódují se DC koeficienty odděleně od AC koeficientů
- kódování DC koeficientů – difference hodnot sousedních bloků (DC prvního bloku se kóduje jako přímá hodnota) - výsledná hodnota se kóduje jako dvojice

**(velikost) (amplituda)**

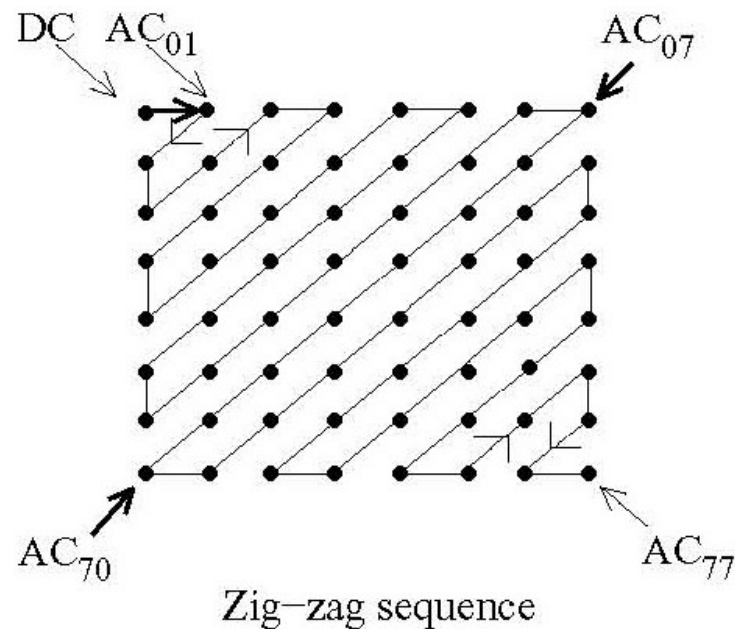


SIZE	AMPLITUDE
1	-1,1
2	-3,-2,2,3
3	-7,-4,4,7
4	-15,-8,8,15
5	-31,-16,16,31
6	-63,-32,32,63
7	-127,-64,64,127
8	-255,-128,128,255
9	-511,-256,256,511
10	-1023,-512,512,1023

- Kódování AC koeficientů – délkou běhu - nejprve se koeficienty uspořádají podle následujícího obrázku pak se kódují jako trojice

**(RL, velikost) (amplituda)**

kde RL je počet nul které jsou před kódovanou hodnotou, velikost a amplituda jsou shodné jako při kódování DC koeficientů



# Příklad kódování bloku obrazu

139	144	149	153	155	155	155	155	235.6	-1.0	-12.1	-5.2	2.1	-1.7	-2.7	1.3	16	11	10	16	24	40	51	61
144	151	153	156	159	156	156	156	-22.6	-17.5	-6.2	-3.2	-2.9	-0.1	0.4	-1.2	12	12	14	19	26	58	60	55
150	155	160	163	158	156	156	156	-10.9	-9.3	-1.6	1.5	0.2	-0.9	-0.6	-0.1	14	13	16	24	40	57	69	56
159	161	162	160	160	159	159	159	-7.1	-1.9	0.2	1.5	0.9	-0.1	0.0	0.3	14	17	22	29	51	87	80	62
159	160	161	162	162	155	155	155	-0.6	-0.8	1.5	1.6	-0.1	-0.7	0.6	1.3	18	22	37	56	68	109	103	77
161	161	161	161	160	157	157	157	1.8	-0.2	1.6	-0.3	-0.8	1.5	1.0	-1.0	24	35	55	64	81	104	113	92
162	162	161	163	162	157	157	157	-1.3	-0.4	-0.3	-1.5	-0.5	1.7	1.1	-0.8	49	64	78	87	103	121	120	101
162	162	161	161	163	158	158	158	-2.6	1.6	-3.8	-1.8	1.9	1.2	-0.6	-0.4	72	92	95	98	112	100	103	99

(a) source image samples

(b) forward DCT coefficients

(c) quantization table

15	0	-1	0	0	0	0	0	240	0	-10	0	0	0	0	0	144	146	149	152	154	156	156	156
-2	-1	0	0	0	0	0	0	-24	-12	0	0	0	0	0	0	148	150	152	154	156	156	156	156
-1	-1	0	0	0	0	0	0	-14	-13	0	0	0	0	0	0	155	156	157	158	158	157	156	155
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	160	161	161	162	161	159	157	155
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	163	163	164	163	162	160	158	156
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	163	164	164	164	162	160	158	157
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	160	161	162	162	162	161	159	158
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	158	159	161	161	162	161	159	158

(d) normalized quantized coefficients

(e) denormalized quantized coefficients

(f) reconstructed image samples

# Příklad kódování bloku obrazu

15	0	-1	0	0	0	0	0
-2	-1	0	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
0	0	0	0	0	0	0	0

**Předpoklad:** předchozí blok měl kvantizovaného DC koeficientu +12

Poslopnost symbolů pro celý obraz je pak možné kódovat Huffmanovým kódem

Poslopnost symbolů, které kódují blok je:

..... **(2)(3), (1,2)(-2), (0,1)(-1), (0,1)(-1), (0,1)(-1), (0,0)** .....

Konec bloku

# Waveletová komprese

## Charakteristika:

- ztrátová komprese
- podobný princip jako u JPEG komprese
- využívá lineární transformaci (waveletovou transformaci)
- obvykle dosahuje vyšších kompresních poměrů

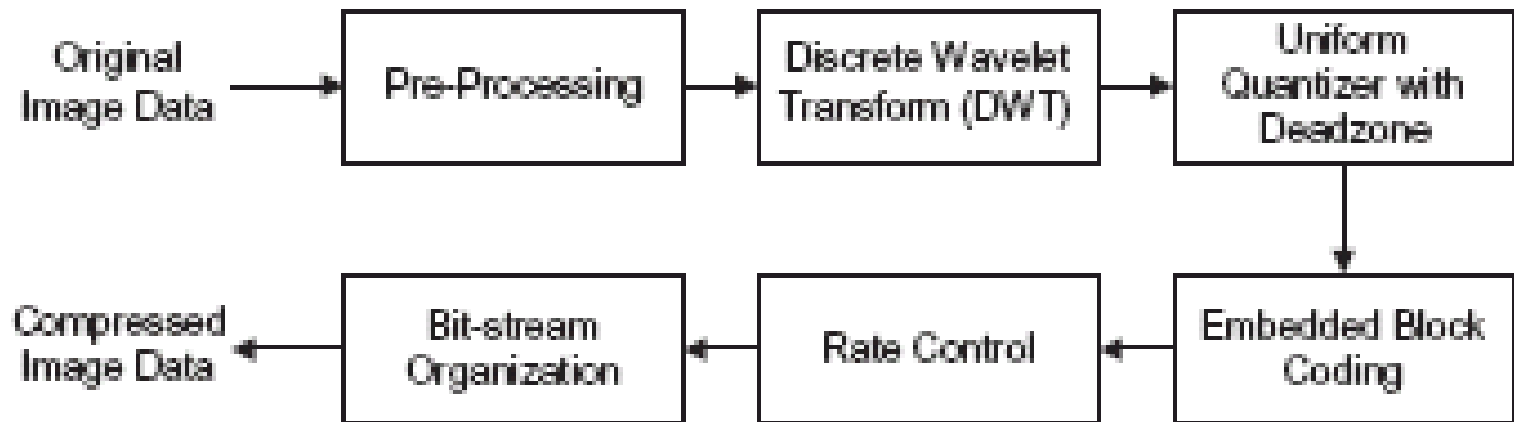
## Použití:

- FBI - komprese otisků prstů
- JPEG 2000

## JPEG 2000

- cílem bylo navrhnout nový obrazový standard, který překonává některé nedostatky které byly u JPEG komprese
- vhodný pro rozdílné typy statických obrazů (binární, šedotónový, barevný) s rozdílnými charakteristikami (scenérie, technické výkresy, družicové snímky)
- vhodný pro rozdílné účely - přenos obrazů, archivace
- kódování JPEG 2000 může být ztrátové nebo bezztrátové

# Blokové schéma kodovacího procesu



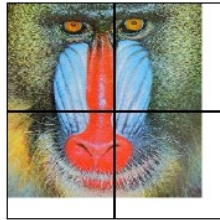


# Předzpracování



tři kroky předzpracování:

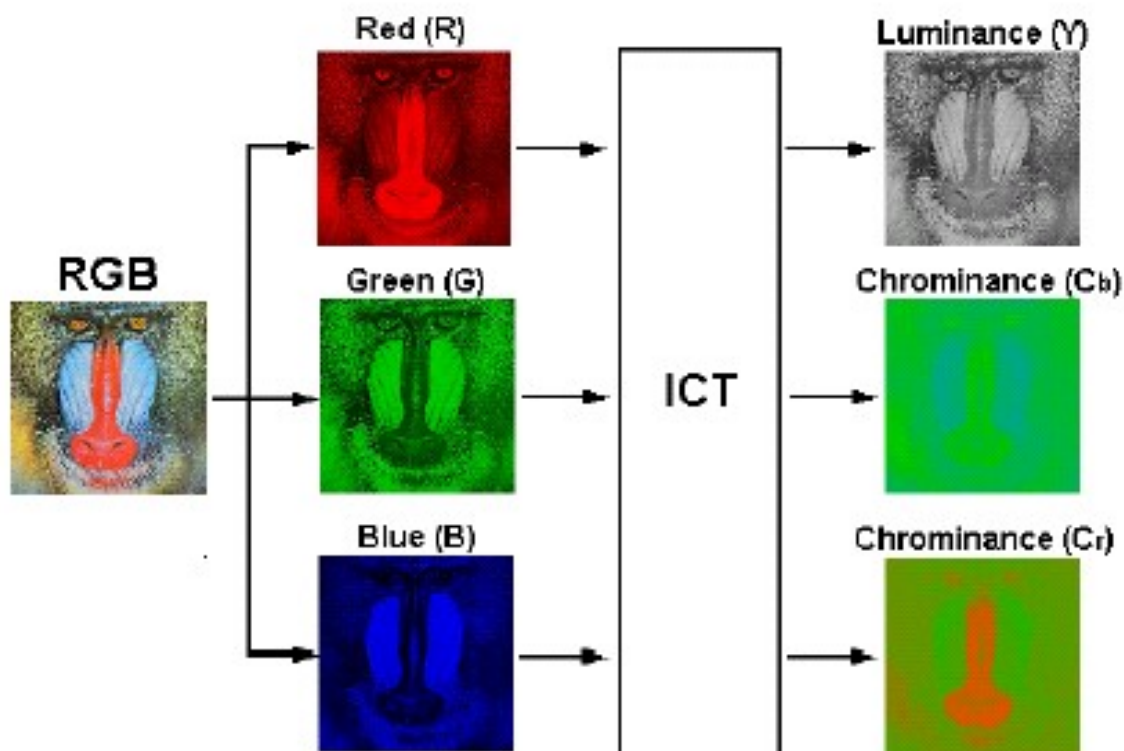
- **rozdělení obrazu na bloky** - bloky se nepřekrývají, jsou stejné velikosti, každý blok je komprimován samostatně s vlastními parametry komprese



- **normalizace úrovní** - jelikož JPEG2000 používá filtr typu horní propust očekává se že rozsah vstupních hodnot je rozložen okolo nuly (je-li rozsah vstupu na B bitech bude vstup po normalizaci v rozsahu  $-2^{B-1} \leq x \leq 2^{B-1}$ )
- **barevná transformace** - většinou jsou komprimovány barevné obrazy v RGB reprezentaci, ta je ale nevhodná pro ztrátovou kompresi (dochází k posuvu barev ) používá se jiný barevný model ( $Y, C_r, C_b$ )

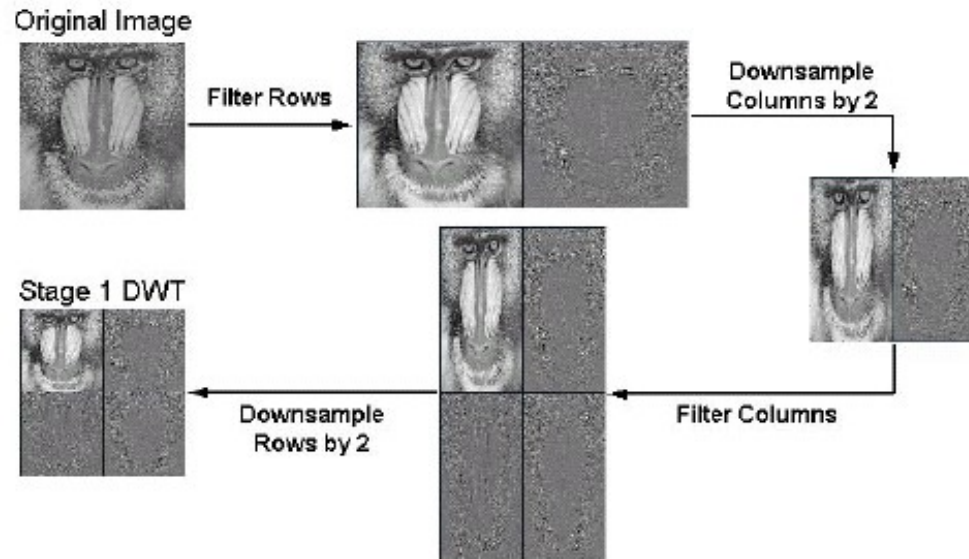
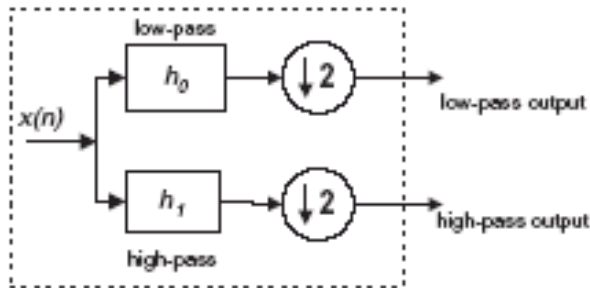
- barevná transformace z RGB do  $Y, C_r, C_b$

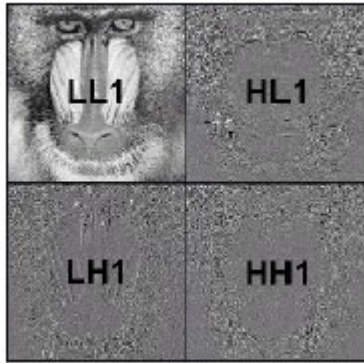
$$\begin{bmatrix} Y \\ C_r \\ C_b \end{bmatrix} = \begin{bmatrix} 0.299 & 0.586 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$



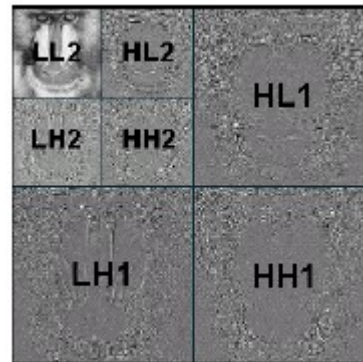
# Diskrétní waveletová transformace

JPEG 2000 je založen na diskrétní waveletové dekompozici DWT

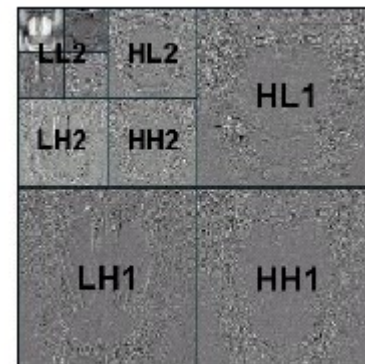




## 1. úroveň



## 2. úroveň



## 3. úroveň

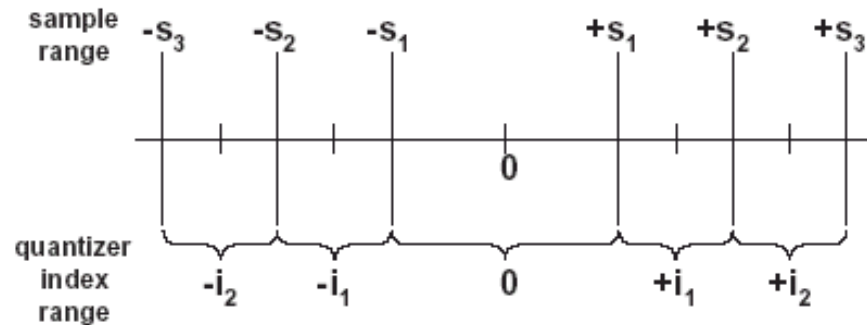
- počet úrovní dekompozice závisí na implementaci

# Kvantizace

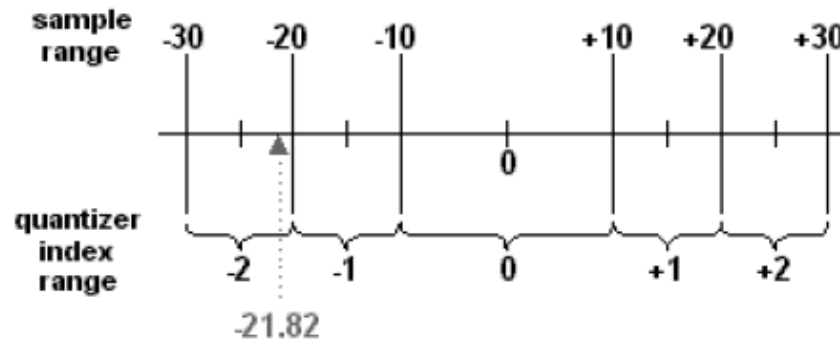
waveletové koeficienty z každé úrovně dekompozice jsou kvantizovány podle vztahu

$$q = \text{sign}(y) \left\lfloor \frac{|y|}{\Delta_b} \right\rfloor$$

výsledkem kvantizace je náhrada hodnoty každého koeficientu kvantizačním indexem



$$\text{Quantizer Index} = - \left\lfloor \frac{21.82}{10} \right\rfloor = -2$$



# System kódování bloků

- každý dekompoziční blok je rozdělen do nepřekrývajících se menších bloků (64x64 nebo 32x32 koeficientů)

