

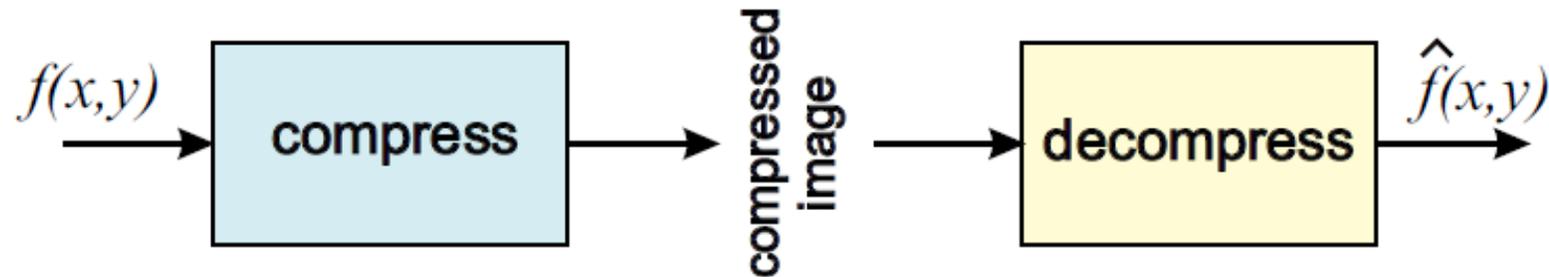
# Pemampatan Citra

IF4073 Interpretasi dan Pengolahan Citra

Oleh: Rinaldi Munir

Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung  
2019

# Pemampatan vs Penirmampatan



- *Image compression* = pemampatan citra, kompresi citra
- *Image decompression* = penirmampatan citra, dekompresi citra
- Citra dimampatkan ketika ia disimpan ke dalam *storage* atau ditransmisikan.
- Citra dinirmampatkan ketika ia ditampilkan ke layar atau disimpan ke dalam dokumen dengan format tidak mampat

# Mengapa citra perlu dimampatkan?

- Representasi citra digital membutuhkan memori yang besar.
- Pemampatan citra adalah metode untuk mereduksi redundansi pada representasi citra sehingga dapat mengurangi kebutuhan memori untuk ruang penyimpanan.
- Citra dimampatkan tanpa mengurangi kualitas citra secara visual
- Tujuan:
  1. Mengurangi kebutuhan ruang penyimpanan sembari tetap mempertahankan kualitas citra secara visual. (Gonzalez, Woods and Eddins, 2017).
  2. Merepresentasikan citra dengan kualitas yang hampir sama dengan citra aslinya namun dalam bentuk yang lebih kompak.

Dapatkah anda melihat perbedaan kualitas hasil pemampatan?



Original image  
(not compressed)



Compressed image



peppers.bmp, 256 x 256  
(193 KB)



peppers.jpg, 256 x 256  
(31 KB), JPEG Quality = 5



peppers2.jpg, 256 x 256  
(24 KB), JPEG Quality = 1

- Misalkan sebuah citra berwarna berukuran 1200x1600

Kebutuhan ruang penyimpanan:

$$\begin{aligned}1200 \times 1600 \times 3 &= 5760000 \text{ byte} \\ &= 5,760 \text{ Kbyte} \\ &= 5.76 \text{ Mbyte}\end{aligned}$$

- Misalkan sebuah film digital dengan resolusi 720x480, 30 frame/sec, selama 2 jam.

Kebutuhan ruang penyimpanan:

$$\begin{aligned}30 \frac{\text{frame}}{\text{sec}} \times (720 \times 480) \frac{\text{pixels}}{\text{frame}} \times 3 \frac{\text{bytes}}{\text{pixel}} &= 31,104,000 \text{ bytes / sec} \\ 31,104,000 \times \frac{\text{bytes}}{\text{sec}} \times (60 \times 60) \frac{\text{sec}}{\text{hour}} \times 2 \text{ hours} &= 2.24 \times 10^{11} \text{ bytes} \\ &= 224 \text{ GByte.}\end{aligned}$$

# Aplikasi pemampatan citra

## 1. Penyimpanan data di dalam media sekunder (*storage*)

Citra mampat membutuhkan memori di dalam *storage* yang lebih sedikit dibandingkan dengan citra yang tidak mampat.

Contoh: file citra berformat JPEG/JPG versus citra berformat BMP

## 2. Pengiriman data (*data transmission*) pada saluran komunikasi data

Citra mampat membutuhkan waktu pengiriman yang lebih singkat dibandingkan dengan citra tidak mampat.

Contoh: pengiriman gambar via email, *videoconference*, via satelit luar angkasa, mengunduh gambar dari internet, dan sebagainya.

# Redundansi

- Redudansi pada citra adalah konten citra yang sebenarnya tidak perlu direpresentasikan.
- Dua macam redundansi:
  1. *Coding redundancy*: biasanya muncul sebagai hasil pengkodean yang seragam pada setiap *pixel*.
  2. *Spatial/temporal redundancy*: misalnya *pixel-pixel* bertetangga memiliki nilai intensitas yang tidak jauh berbeda.
  3. *Psychovisual redundancy*: persepsi visual mengakibatkan *redundancy*

# Coding redundancy

Symbol $r_k$	Probability $p_r(r_k)$	Code 1	Length $l_1(r_k)$
$r_0 = A$	0.19	000	3
$r_1 = B$	0.25	001	3
$r_2 = C$	0.21	010	3
$r_3 = D$	0.16	011	3
$r_4 = E$	0.08	100	3
$r_5 = F$	0.06	101	3
$r_6 = G$	0.03	110	3
$r_7 = H$	0.02	111	3

- Tiap simbol, tanpa memperhatikan frekuensi kemunculannya, dikodekan dengan panjang bit yang tetap (fixed-length encoding), yaitu 3 bit.
- Rata-rata panjang bit kode untuk setiap simbol 3 bit

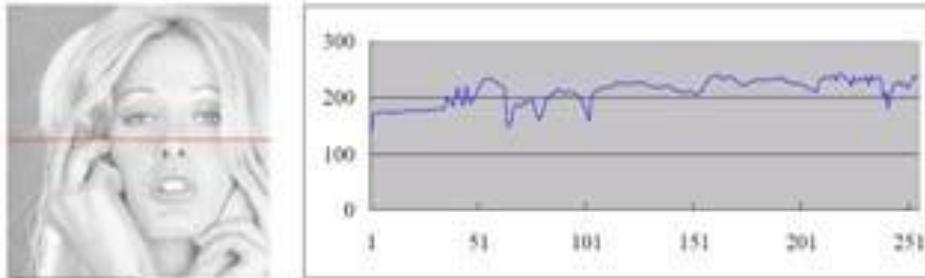
- *Coding redundancy* dapat dikurangi dengan mengkodekan simbol yang sering muncul dengan jumlah bit yang lebih sedikit

Symbol $r_k$	Probability $p_r(r_k)$	Code 2	Length $l_2(r_k)$
$r_0 = A$	0.19	11	2
$r_1 = B$	0.25	01	2
$r_2 = C$	0.21	10	2
$r_3 = D$	0.16	001	3
$r_4 = E$	0.08	0001	4
$r_5 = F$	0.06	00001	5
$r_6 = G$	0.03	000001	6
$r_7 = H$	0.02	000000	6

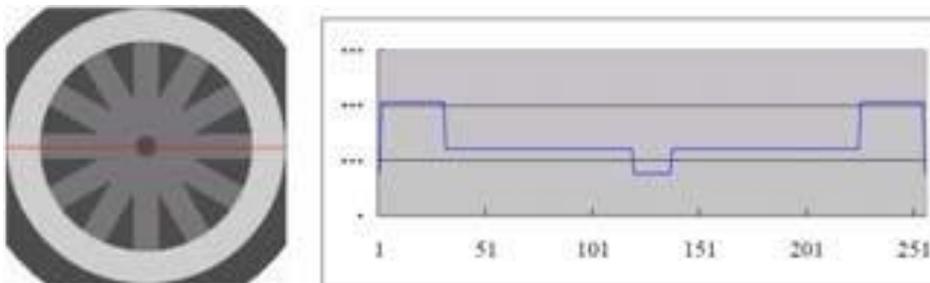
- Rata-rata panjang bit kode untuk setiap simbol =  $\{(19 \times 2) + (25 \times 2) + (21 \times 2) + \dots + (3 \times 6) + (2 \times 6)\} / 100 = 2.7$
- Nisbah pemampatan =  $3 / 2.7 = 1.11$

# Spatial redundancy

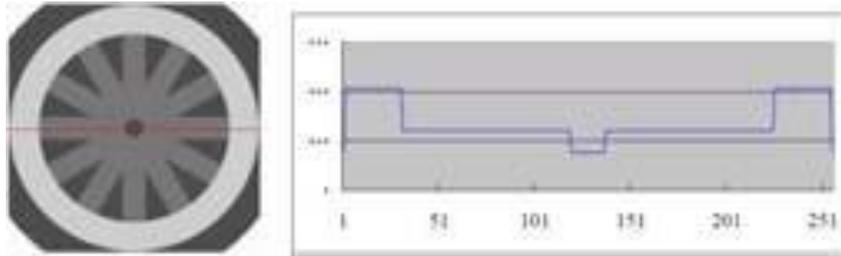
Nilai-nilai pixel-pixel citra Tiffany pada baris 128:



Nilai-nilai pixel-pixel citra roda pada baris 128:



- Pixel-pixel baris 128 pada citra roda (256 pixel):



- Misalkan  $n1$  menyatakan graylevel dan  $n2$  adalah frekuensi kemunculannya

Pada baris 128,  $(n1, n2)$ :  $(77, 1)$ ,  $(206, 30)$ ,  
 $(121, 88)$ ,  $(77, 18)$ ,  $(121, 88)$ ,  $(206, 30)$ ,  $(77, 1)$

Kodekan setiap segmen dengan 16 bit ( $n1$  delapan bit,  $n2$  delapan bit):  
Nisbah pemampatan =  $(256 \times 8) / (7 \times 16) = 18.3$

# Psychovisual Redundancy

- Untuk persepsi visual manusia, informasi tertentu kurang penting. E.g ,: kuantisasi graylevel yang tepat tidak memengaruhi kualitas visualnya.
- Citra Tiffany ini jika dikodekan 5 bit/pixel tidak mempengaruhi persepsi visual manusia



8 bits/pixel



5 bits/pixel

- Metode pemampatan kuantisasi.

# Teori Informasi

- Mendefinisikan jumlah informasi di dalam pesan sebagai jumlah minimum bit yang dibutuhkan untuk mengkodekan pesan.
- Contoh:
  - 1 bit untuk mengkodekan jenis kelamin
  - 3 bit untuk mengkodekan nama hari
  - 4 bit untuk mengkodekan 0 s/d 9

- *Entropy*: ukuran yang menyatakan jumlah informasi di dalam pesan.
- Biasanya dinyatakan dalam satuan bit.
  
- Entropi berguna untuk memperkirakan jumlah bit rata-rata untuk mengkodekan elemen dari pesan.
  
- Contoh: entropi untuk pesan yang menyatakan jenis kelamin = 1 bit, entropi untuk pesan yang menyatakan nama hari = 3 bit

- Secara umum, entropi pesan dihitung dengan rumus yang dikemukakan oleh Claude Shannon, 1948:

$$H(X) = -\sum_{i=1}^n p_i \log(p_i)$$

$p_i$  = peluang kemunculan simbol ke- $i$  di dalam pesan  $X$

- Entropi dinyatakan dalam satuan bit

- Contoh: misalkan pesan  $X = \text{'AABBCBDB'}$

$n = 4$  (yaitu huruf A, B, C, D)

$p(A) = 2/8, p(B) = 4/8$

$p(C) = 1/8, p(D) = 1/8$

$$\begin{aligned}
 H(x) &= -\{2/8 \log_2(2/8) + 4/8 \log_2(4/8) + 1/8 \log_2(1/8) + 1/8 \log_2(1/8)\} \\
 &= -\{1/4 \log_2(1/4) + 1/2 \log_2(1/2) + 1/8 \log_2(1/8) + 1/8 \log_2(1/8)\} \\
 &= -\{(1/4) (-2 \log_2(4)) + (1/2) (-2 \log_2(2)) + (1/8) (-2 \log_2(8)) + (1/8) (-2 \log_2(8))\} \\
 &= -\{(1/4) (-2) + (1/2) (-1) + (1/8) (-3) + (1/8) (-3)\} \\
 &= -\{-1/2 - 1/2 - 3/8 - 3/8\} \\
 &= -(-1.75) \\
 &= 1.75
 \end{aligned}$$

Entropi = 1,75 bit per simbol

Let only two symbols a, b occur in the message.

### Example 1

$$p(\mathbf{a}) = p(\mathbf{b}) = \frac{1}{2}$$

$$H = - \left( \frac{1}{2} \log_2 \frac{1}{2} + \frac{1}{2} \log_2 \frac{1}{2} \right) = \left( \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1 \right) = 1$$

### Example 2

$$p(\mathbf{a}) = 0,99; p(\mathbf{b}) = 0,01$$

$$\begin{aligned} H &= - (0,99 \log_2 0,99 + 0,01 \log_2 0,01) \\ &= - (0,99 \cdot (-0,0145) + 0,01 \cdot (-6,6439)) \\ &= 0,0144 + 0,0664 = 0,0808 \end{aligned}$$

# Tipe metode pemampatan citra

## 1. Lossy compression

- Metode *lossy* menghasilkan citra hasil pemampatan yang *hampir* sama dengan citra semula. Ada informasi yang hilang akibat pemampatan, tetapi dapat ditolerir oleh persepsi visual.
- Bertujuan untuk memperoleh nisbah pemampatan yang tinggi
- Contoh: JPEG compression, fractal image compression

## 2. Lossless compression

- Metode *lossless* selalu menghasilkan citra hasil penirmampatan yang tepat sama dengan citra semula, *pixel per pixel*. Tidak ada informasi yang hilang akibat pemampatan.
- Nisbah pemampatan rendah, namun kualitas citra mampat tetap tinggi
- Dibutuhkan untuk memampatkan citra yang tidak boleh terdegradasi akibat pemampatan, misalnya citra medis, citra x-ray
- Contoh: metode Huffman, *run-length encoding* (RLE), *quantized coding*

Lossless compression

Lossy compression



peppers.bmp, 256 x 256  
(193 KB)



peppers.png, 256 x 256  
(127 KB)



peppers2.jpg, 256 x 256  
(24 KB), JPEG Quality = 1

# Metode Pemampatan Huffman

- *Lossless compression*
- Berdasarkan algoritma *greedy*
- Prinsip kerja algoritma: *pixel* dengan nilai keabuan yang sering muncul dikodekan dengan panjang bit yang lebih sedikit, sebaliknya nilai keabuan yang jarang muncul dikodekan dengan bit yang lebih panjang

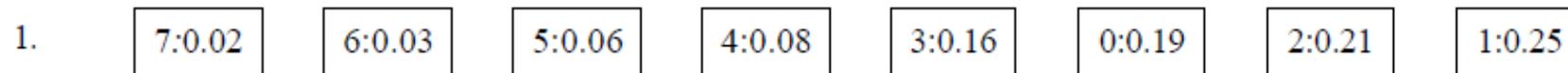
## Algoritma:

1. Setiap nilai keabuan dinyatakan sebagai simpul. Setiap simpul di-*assign* dengan frekuensi kemunculan nilai keabuan tersebut.
2. Urutkan secara menaik (*ascending order*) simpul-simpul berdasarkan frekuensi kemunculannya.
3. Gabung dua buah simpul yang mempunyai frekuensi kemunculan paling kecil pada sebuah akar. Akar mempunyai frekuensi yang merupakan jumlah dari frekuensi dua buah pohon penyusunnya.
4. Ulangi langkah 2 sampai tersisa hanya satu buah pohon biner.
5. Beri label setiap sisi pada pohon biner. Sisi kiri dilabeli dengan 0 dan sisi kanan dilabeli dengan 1.
6. Telusuri pohon biner dari akar ke daun. Barisan label-label sisi dari akar ke daun menyatakan kode Huffman untuk derajat keabuan yang bersesuaian.

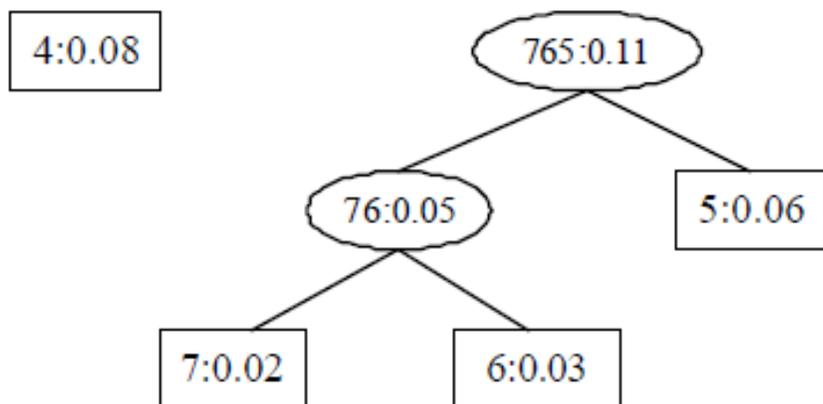
**Contoh:** Misalkan terdapat citra yang berukuran  $64 \times 64$  dengan 8 derajat keabuan ( $k$ ) dan jumlah seluruh *pixel* ( $n$ ) =  $64 \times 64 = 4096$ .

$k$	$n_k$	$p(k) = n_k/n$
0	790	0.19
1	1023	0.25
2	850	0.21
3	656	0.16
4	329	0.08
5	245	0.06
6	122	0.03
7	81	0.02

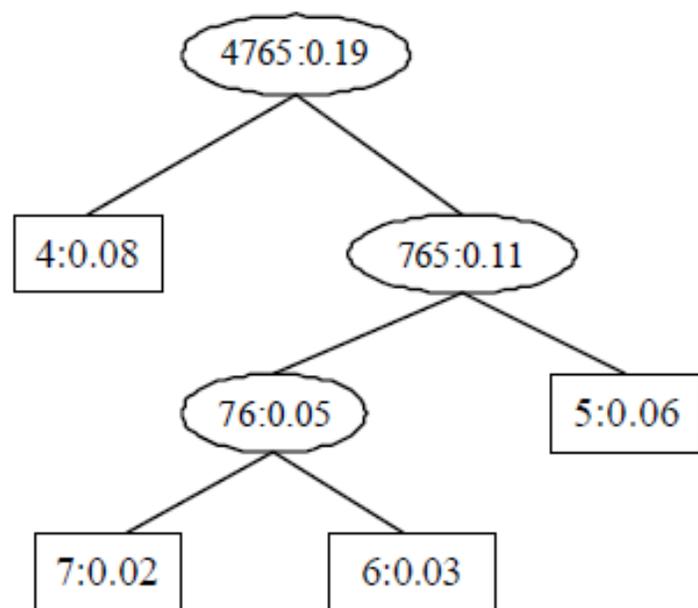
Proses pembentukan pohon Huffman:



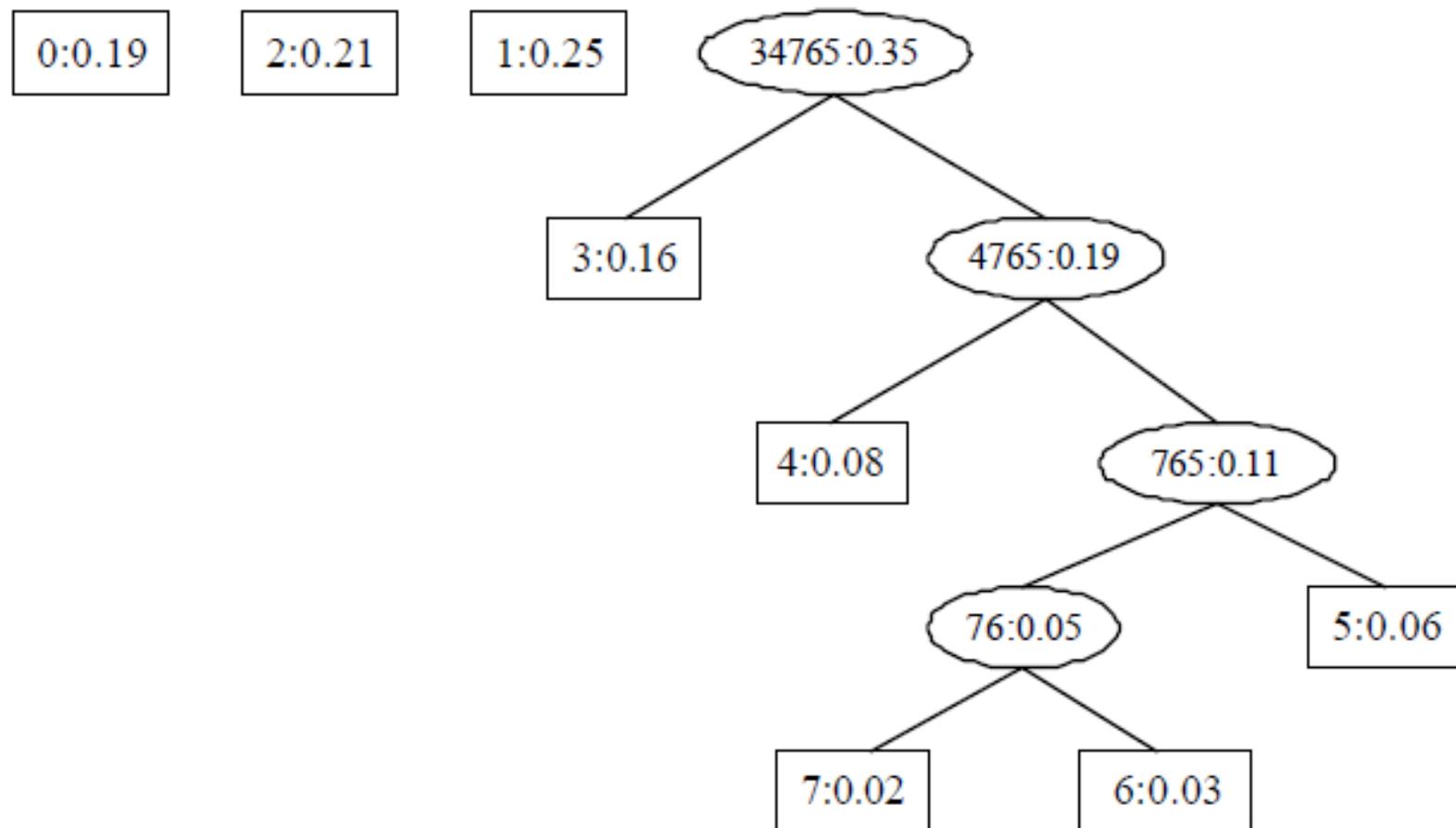
3.



4.

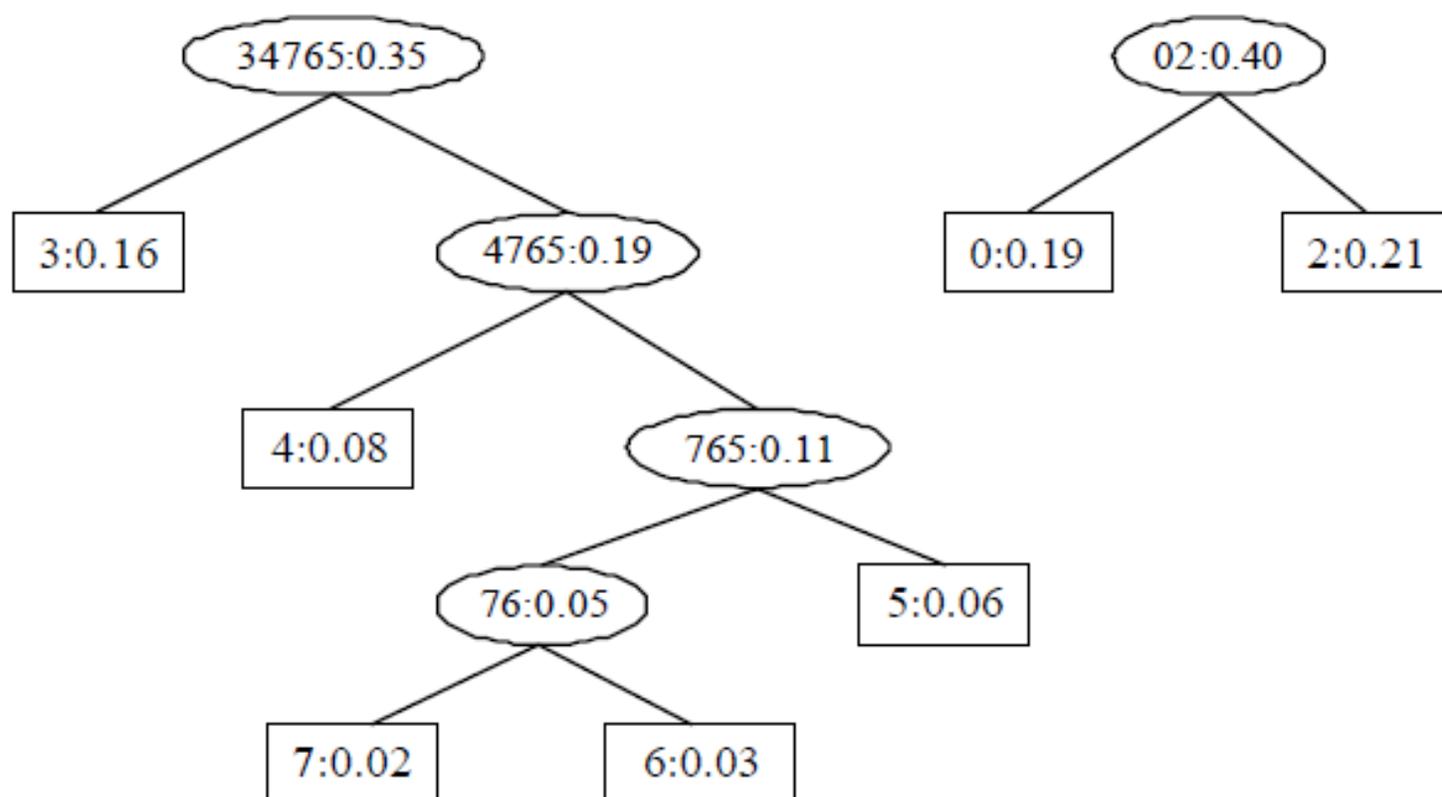


5.

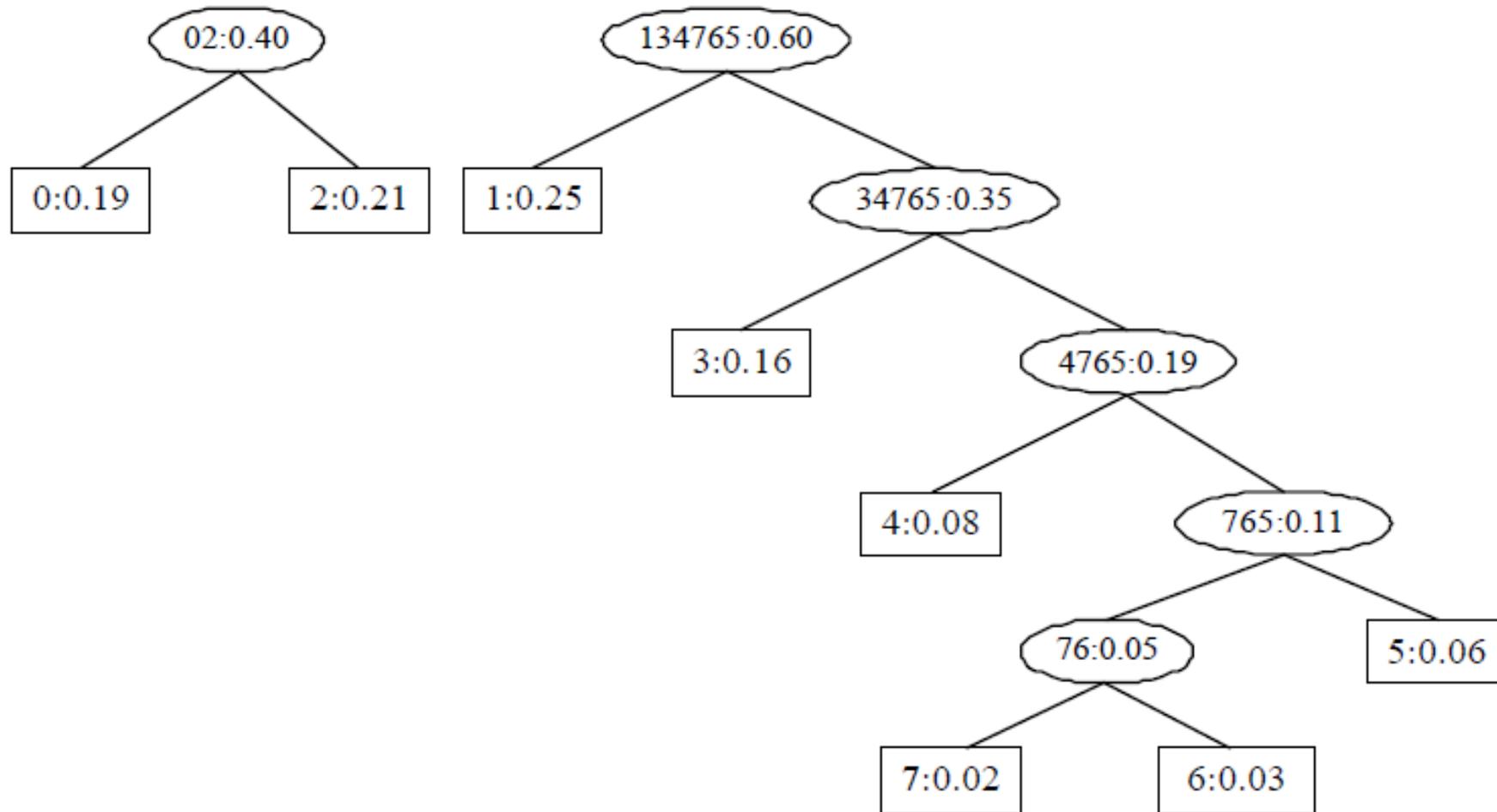


6.

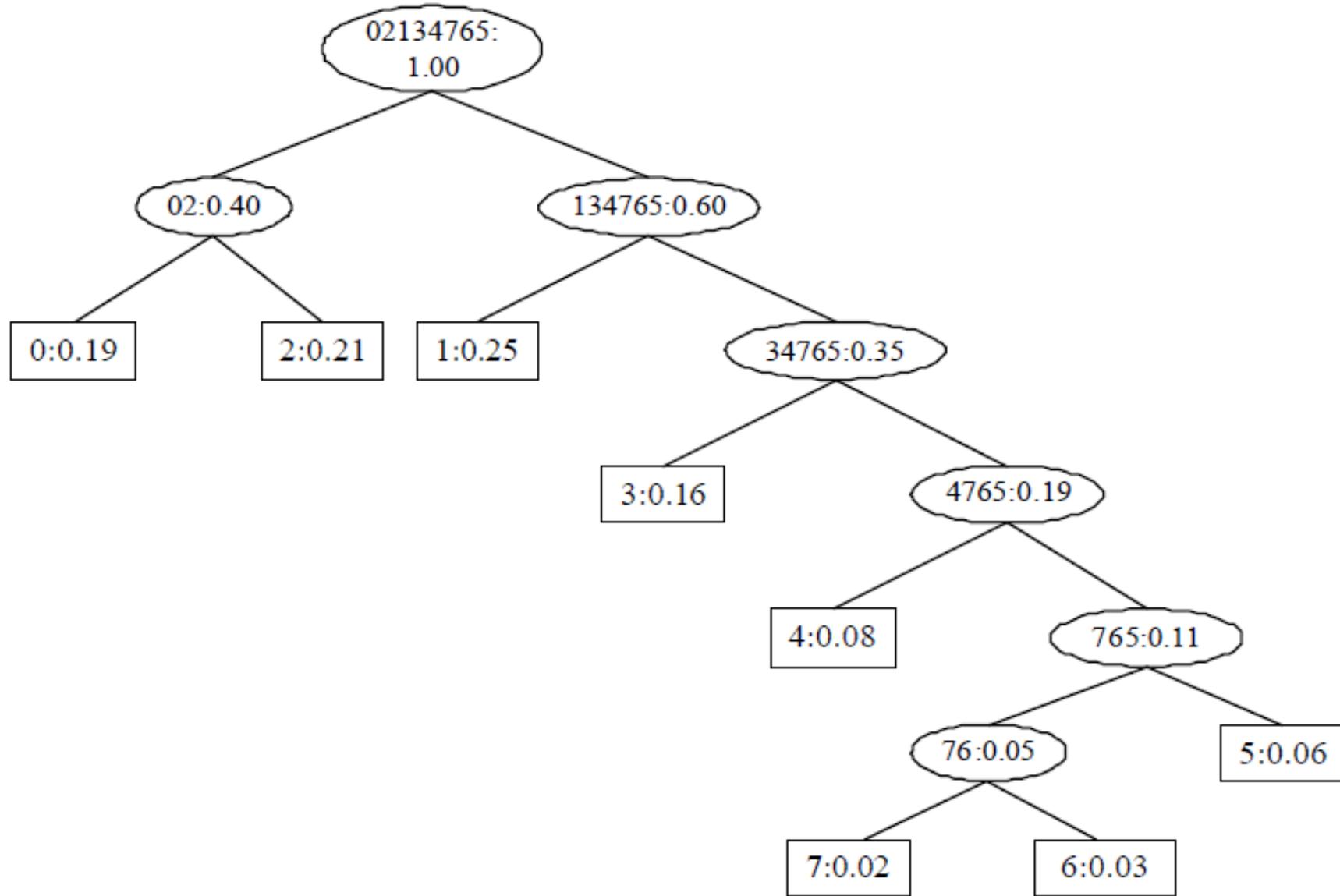
1:0.25



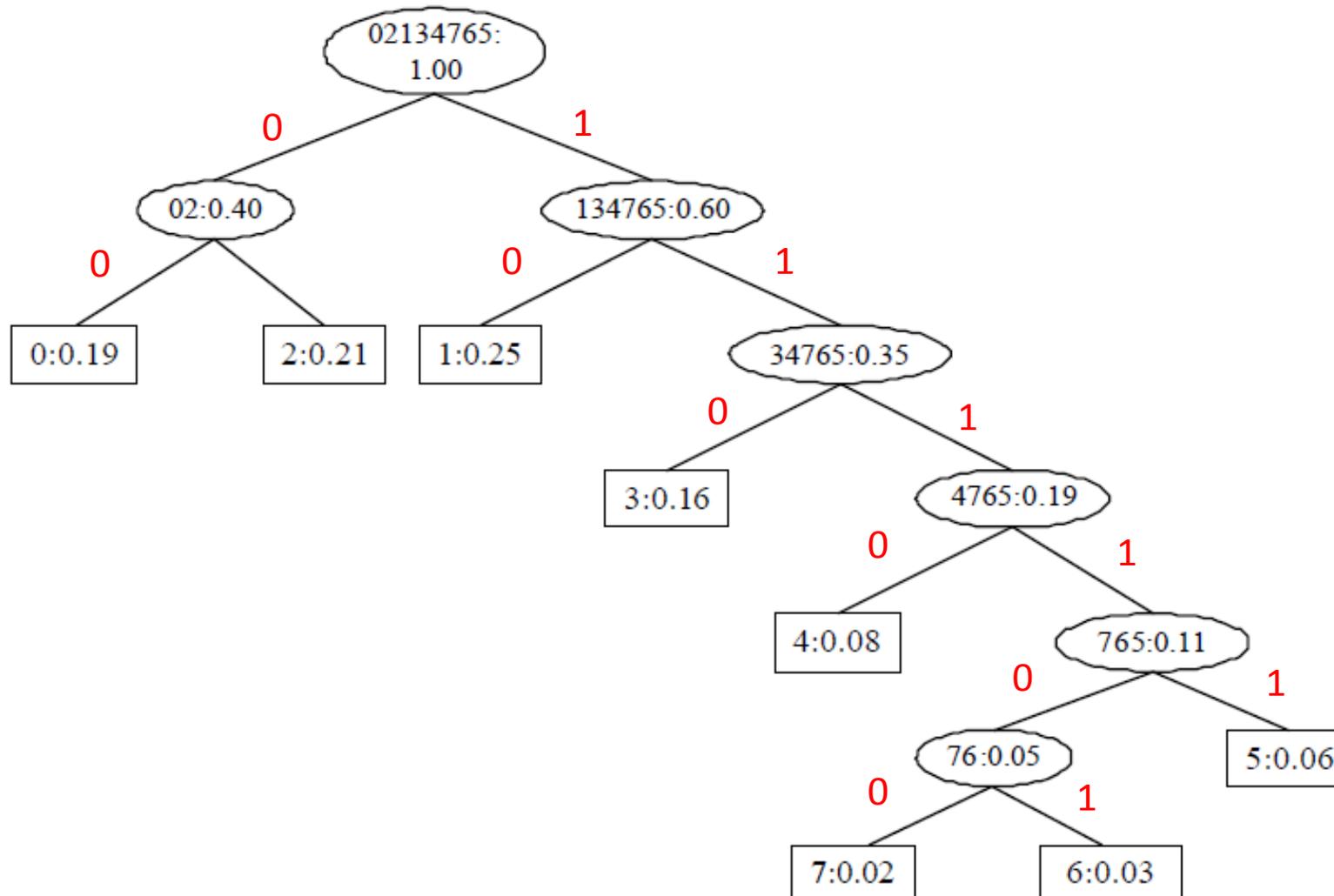
7.



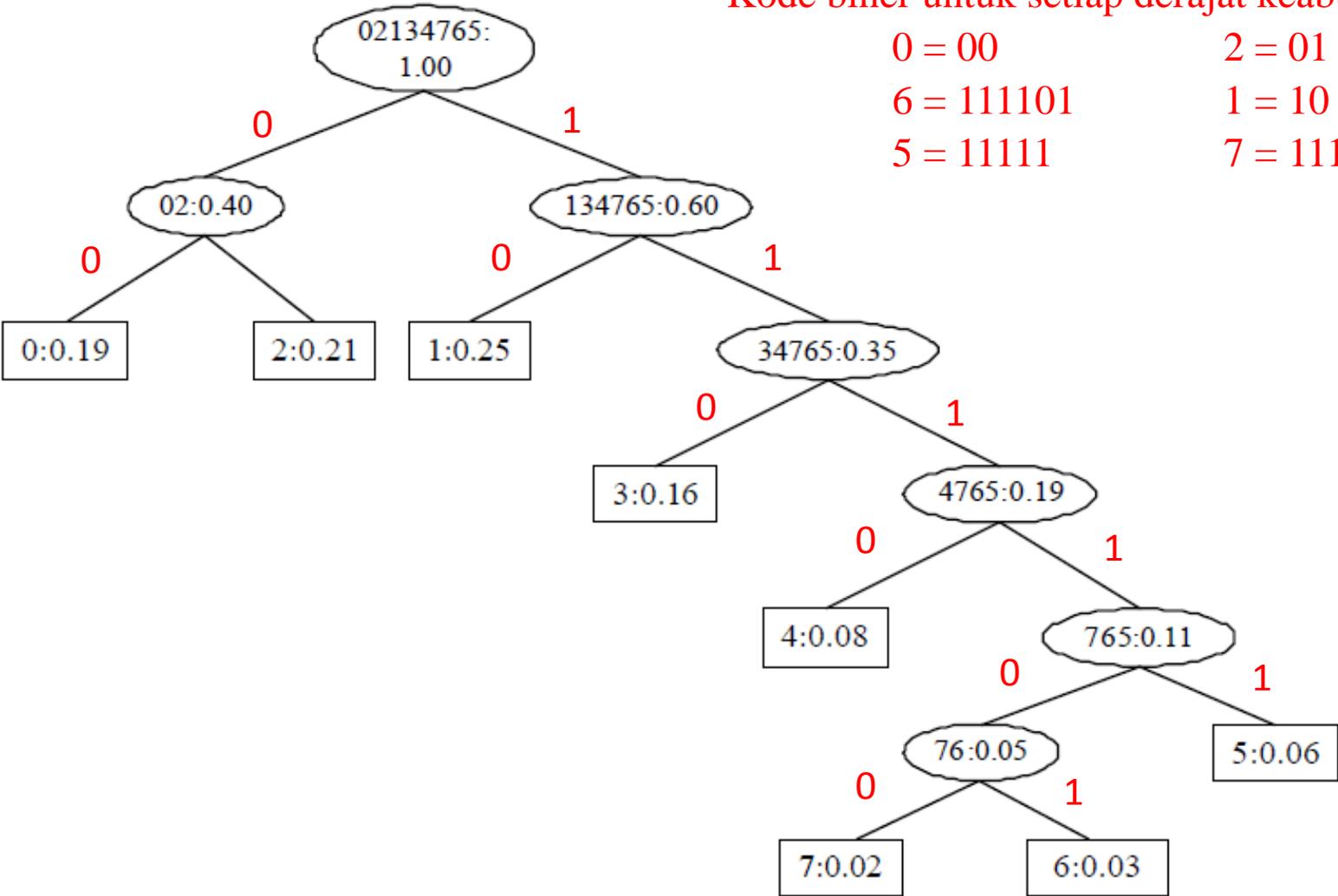
8.



Beri label setiap sisi pada pohon biner. Sisi kiri dilabeli dengan 0 dan sisi kanan dilabeli dengan 1



Telusuri pohon biner dari akar ke daun. Barisan label-label sisi dari akar ke daun menyatakan kode Huffman untuk derajat keabuan yang bersesuaian.



Kode biner untuk setiap derajat keabuan sebagai berikut:

- 0 = 00
- 2 = 01
- 4 = 1110
- 6 = 111101
- 1 = 10
- 3 = 110
- 5 = 11111
- 7 = 111100

- Ukuran citra sebelum pemampatan (1 derajat keabuan = 3 bit) adalah  $4096 \times 3 \text{ bit} = 12288 \text{ bit}$
- Ukuran citra setelah pemampatan:  
 $(790 \times 2 \text{ bit}) + (1023 \times 2 \text{ bit}) + (850 \times 2 \text{ bit}) +$   
 $(656 \times 3 \text{ bit}) + (329 \times 4 \text{ bit}) + (245 \times 5 \text{ bit}) +$   
 $(122 \times 6 \text{ bit}) + (81 \times 6 \text{ bit}) = 11053 \text{ bit}$
- Nisbah (ratio) pemampatan =  $\frac{11053}{12288} \times 100\% = 89.95\%$

Artinya: - citra berhasil dimampatkan menjadi 89.95% dari citra semula  
- 10.05 % dari citra semula telah dimampatkan

# Metode *Run-Length Encoding* (RLE)

- Metode *RLE* cocok digunakan untuk memampatkan citra yang memiliki kelompok-kelompok *pixel* berderajat keabuan sama.
- Pemampatan citra dengan metode *RLE* dilakukan dengan membuat rangkaian pasangan nilai  $(p, q)$  untuk setiap baris *pixel*.
- Nilai pertama ( $p$ ) menyatakan derajat keabuan (*graylevel*)
- Nilai kedua ( $q$ ) menyatakan jumlah *pixel* berurutan yang memiliki derajat keabuan tersebut (dinamakan *run length*).

**Contoh:** Tinjau citra  $10 \times 10$  *pixel* dengan 8 derajat keabuan yang dinyatakan sebagai matriks derajat keabuan sebagai berikut (100 buah nilai):

0	0	0	0	0	2	2	2	2	2
0	0	0	1	1	1	1	2	2	2
1	1	1	1	1	1	1	1	1	1
4	4	4	4	3	3	3	3	2	2
3	3	3	5	5	7	7	7	7	6
2	2	6	0	0	0	0	1	1	0
3	3	4	4	3	2	2	2	1	1
0	0	0	0	0	0	0	0	1	1
1	1	1	1	0	0	0	2	2	2
3	3	3	2	2	2	1	1	1	1

Pasangan nilai untuk setiap baris *run*:

- (0, 5), (2, 5)
- (0, 3), (1, 4), (2, 3)
- (1, 10)
- (4, 4), (3, 4), (2, 2)
- (3, 3), (5, 2), (7, 4), (6, 1)
- (2, 2), (6, 1), (0, 4), (1, 2), (0, 1)
- (3, 2), (4, 2), (3, 1), (2, 2), (1, 2)
- (0, 8), (1, 2)
- (1, 4), (0, 3), (2, 3)
- (3, 3), (2, 3), (1, 4)

Semuanya ada 31 pasangan nilai,  $31 \times 2 = 62$  nilai.

- Ukuran citra sebelum pemampatan (1 derajat keabuan = 3 bit) adalah  $100 \times 3 \text{ bit} = 300 \text{ bit}$ .
- Ukuran citra setelah pemampatan (derajat keabuan = 3 bit, run length = 4 bit):

$$(31 \times 3) + (31 \times 4) \text{ bit} = 217 \text{ bit}$$

- Nisbah pemampatan =  $\frac{217}{300} \times 100\% = 72.33\%$

Artinya: - citra berhasil dimampatkan menjadi 72.33% dari citra semula  
- 27.67 % dari citra semula telah dimampatkan

- Metode *RLE* dapat dikombinasikan dengan metode Huffman untuk mengkodekan nilai-nilai hasil pemampatan *RLE*
- Tujuannya untuk meningkatkan nisbah pemampatan.
- Mula-mula lakukan pemampatan RLE, lalu hasilnya dimampatkan lagi dengan metode Huffman.

# Metode Pemampatan Kuantisasi (*Quantizing Compression*)

- Metode ini mengurangi jumlah derajat keabuan, misalnya dari 256 menjadi 16, yang tentu saja mengurangi jumlah bit yang dibutuhkan untuk merepresentasikan citra.
- Misalkan  $P$  adalah jumlah pixel di dalam citra semula, akan dimampatkan menjadi  $n$  derajat keabuan.
- Algoritma metode kuantisasi:
  1. Buat histogram citra semula (citra yang akan dimampatkan).
  2. Identifikasi  $n$  buah kelompok di dalam histogram sedemikian sehingga setiap kelompok mempunyai kira-kira  $P/n$  buah *pixel*.
  3. Nyatakan setiap kelompok dengan derajat keabuan 0 sampai  $n - 1$ . Setiap *pixel* di dalam kelompok dikodekan kembali dengan nilai derajat keabuan yang baru.

Contoh: Tinjau citra yang berukuran  $5 \times 13$  *pixel*:

2	9	6	4	8	2	6	3	8	5	9	3	7
3	8	5	4	7	6	3	8	2	8	4	7	3
3	8	4	7	4	9	2	3	8	2	7	4	9
3	9	4	7	2	7	6	2	1	6	5	3	0
2	0	4	3	8	9	5	4	7	1	2	8	3

yang akan dimampatkan menjadi citra dengan 4 derajat keabuan (0 s/d 3), jadi setiap derajat keabuan direpresentasikan dengan 2 bit

## Histogram citra semula:

```

0 **
1 **
2 ****
3 ****
4 ****
5 ****
6 ****
7 ****
8 ****
9 ****

```

Ada 65 *pixel*, dikelompokkan menjadi 4 kelompok derajat keabuan. Tiap kelompok ada sebanyak rata-rata  $65/4 = 16.25$  *pixel* per kelompok:

```

-----
13  0 **
    1 **
    2 ****
-----
20  3 ****
    4 ****
-----
17  5 ****
    6 ****
    7 ****
-----
15  8 ****
    9 ****
-----

```

Citra setelah dimampatkan menjadi:

0	3	2	1	3	0	2	1	3	2	3	1	2
1	3	2	1	2	2	1	3	0	3	1	2	1
1	3	1	2	1	3	0	1	3	0	2	1	3
1	3	1	2	0	2	2	0	0	2	2	1	0
0	0	1	1	3	3	2	1	2	0	0	3	0

Ukuran citra sebelum pemampatan (1 derajat keabuan = 4 bit):

$$65 \times 4 \text{ bit} = 260 \text{ bit}$$

Ukuran citra setelah pemampatan (1 derajat keabuan = 2 bit):

$$65 \times 2 \text{ bit} = 130 \text{ bit}$$

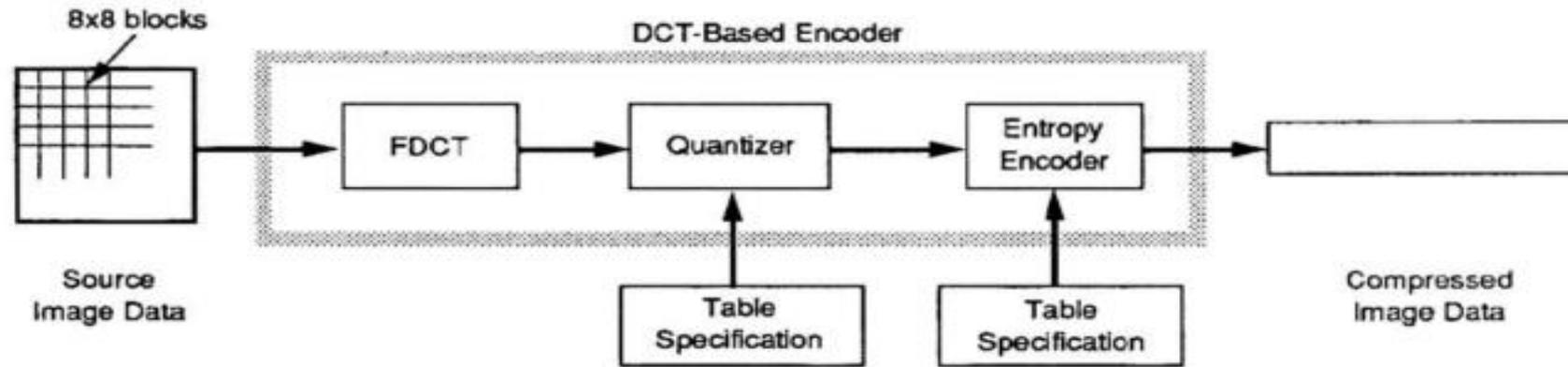
Nisbah pemampatan =  $13/260 \times 100\% = 50\%$

# Metode Pemampatan JPEG

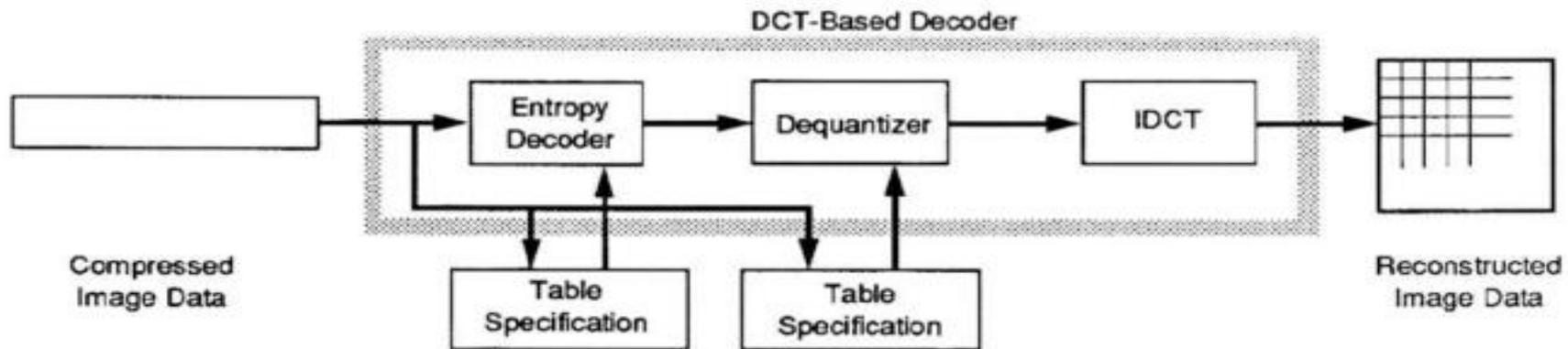
- JPEG = *Joint Photographic Experts Groups*.
- Merupakan standard kompresi citra sejak tahun 1992.
- Termasuk *lossy compression*, yang berarti beberapa kualitas visual ada yang hilang selama proses kompresi. Hasil dekompresi tidak kembali sama dengan citra semula.
- Metode JPEG dapat diterapkan pada citra *grayscale* dan citra berwarna RGB.
- Jika citra berwarna, maka RGB dikonversi terlebih dahulu ke ruang warna *YCbCr*, yang dalam hal ini *Y* adalah komponen *luminance*, dan *Cb* dan *Cr* adalah komponen *chrominance* dari citra.
- Terhadap komponen *chrominance* dilakukan *subsampling* untuk mengurangi ukuran file. *Subsampling* dikerjakan dengan mengambil 4 *pixel* bertetangga dan menghitung rata-ratanya menjadi satu nilai.

# Diagram umum JPEG Compression

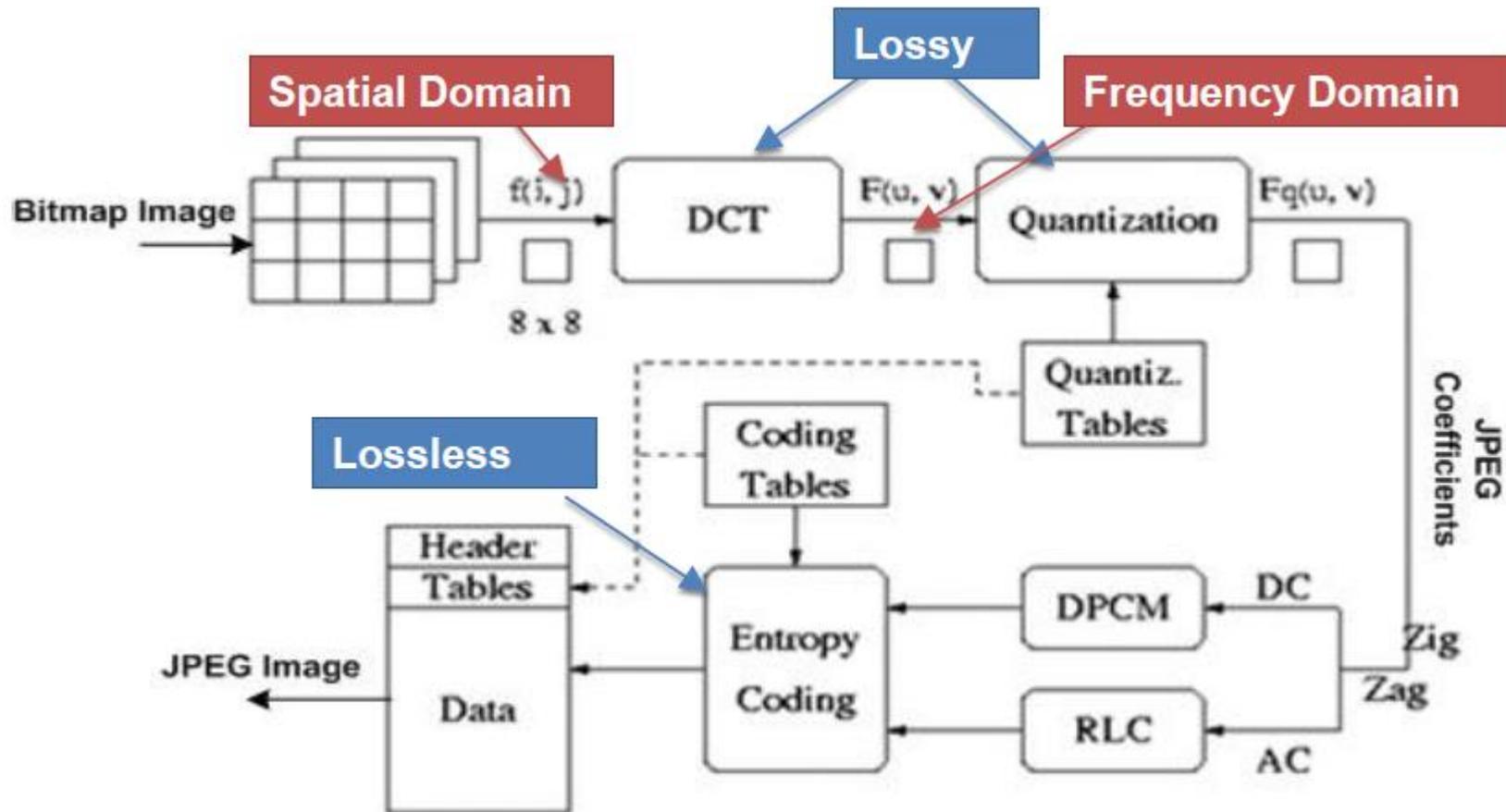
Encoder:



Decoder:

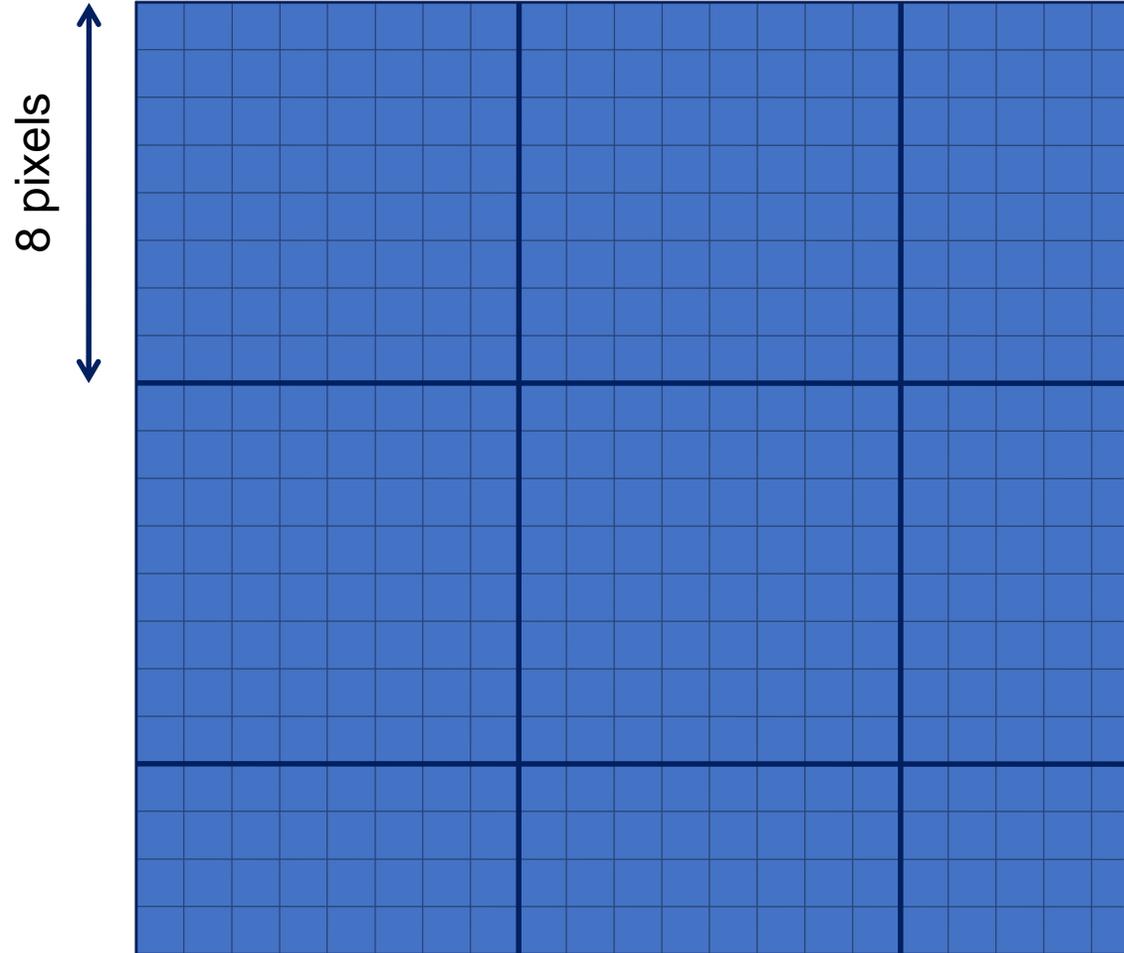


- Diagram rinci JPEG encoder):



Sumber: Mahendra Kumar, *Steganography and Steganalysis of JPEG Images*.

Citra dibagi menjadi blok-blok berukuran  $8 \times 8$  *pixel*:



Setiap blok (dalam ranah spasial) ditransformasi ke ranah frekuensi dengan *Discrete Cosine Transform* (DCT):

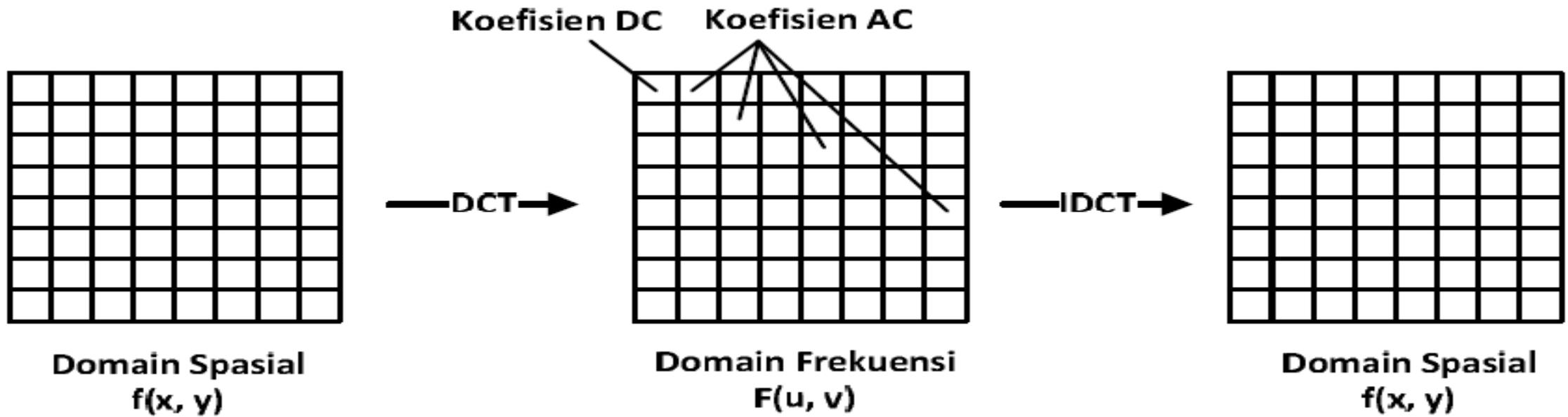
$$F(u, v) = \frac{1}{4} C(u)C(v) \sum_{x=0}^7 \sum_{y=0}^7 f(x, y) \cos \frac{\pi(2x+1)u}{16} \cos \frac{\pi(2y+1)v}{16}$$

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

$$C(u), C(v) = 1, \text{ untuk } u, v \text{ lainnya}$$

Sedangkan transformasi balikkannya adalah Inverse *Discrete Cosine Transform* (IDCT):

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



Elemen pojok kiri atas (1,1) disebut koefisien DC, sedangkan 63 elemen lainnya disebut koefisien AC.

- Hasilnya adalah blok-blok berukuran 8 x 8 dengan nilai *floating point*.

- Contoh:

$$F(u,v) = \begin{bmatrix} 515 & 65 & -12 & 4 & 1 & 2 & -8 & 5 \\ -16 & 3 & 2 & 0 & 0 & -11 & -2 & 3 \\ -12 & 6 & 11 & -1 & 3 & 0 & 1 & -2 \\ -8 & 3 & -4 & 2 & -2 & -3 & -5 & -2 \\ 0 & -2 & 7 & -5 & 4 & 0 & -1 & -4 \\ 0 & -3 & -1 & 0 & 4 & 1 & -1 & 0 \\ 3 & -2 & -3 & 3 & 3 & -1 & -1 & 3 \\ -2 & 5 & -2 & 4 & -2 & 2 & -3 & 0 \end{bmatrix}$$

- Selanjutnya, setiap nilai di dalam blok dikuantisasi menjadi *integer* dengan cara membaginya dengan elemen matriks kuantisasi Q berukuran 8 x 8 dan membulatkannya ke integer terdekat.

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

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

- Contoh hasil proses kuantisasi:

32	6	-1	0	0	0	0	0
-1	0	0	0	0	0	0	0
-1	0	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

$$\hat{F}(u, v)$$

- Kuantisasi ke *integer* membuat komponen frekuensi tinggi dibulatkan menjadi nol, sedangkan komponen frekuensi sisanya menjadi bilangan positif kecil dan bilangan negatif kecil.
- Proses kuantisasi inilah yang dinamakan *lossy*, karena ada informasi yang hilang akibat pembulatan.

- Pada proses penirmampatan (di dalam *decoder*), hampiran nilai  $F(u,v)$  diperoleh kembali dengan mengalikan  $\hat{F}(u,v)$  dengan  $Q(u,v)$ .

$$\tilde{F}(u,v) = \hat{F}(u,v) \times Q(u,v)$$

Contoh:



- An 8 × 8 block from the Y image of 'Lena'

200 202 189 188 189 175 175 175	515 65 -12 4 1 2 -8 5
200 203 198 188 189 182 178 175	-16 3 2 0 0 -11 -2 3
203 200 200 195 200 187 185 175	-12 6 11 -1 3 0 1 -2
200 200 200 200 197 187 187 187	-8 3 -4 2 -2 -3 -5 -2
200 205 200 200 195 188 187 175	0 -2 7 -5 4 0 -1 -4
200 200 200 200 200 190 187 175	0 -3 -1 0 4 1 -1 0
205 200 199 200 191 187 187 175	3 -2 -3 3 3 -1 -1 3
210 200 200 200 188 185 187 186	-2 5 -2 4 -2 2 -3 0

Original Image Block	$f(i, j)$	$F(u, v)$	DCT Coefficients
-------------------------	-----------	-----------	---------------------

- Fig. 9.2: JPEG compression for a smooth image block.

	32	6	-1	0	0	0	0	0	512	66	-10	0	0	0	0	0	
	-1	0	0	0	0	0	0	0	-12	0	0	0	0	0	0	0	
	-1	0	1	0	0	0	0	0	-14	0	16	0	0	0	0	0	
Quantized	-1	0	0	0	0	0	0	0	-14	0	0	0	0	0	0	0	De-Quantized
DCT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Coefficients
Coefficients	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	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	
	$\hat{F}(u, v)$								$\tilde{F}(u, v)$								
	199	196	191	186	182	178	177	176	1	6	-2	2	7	-3	-2	-1	
	201	199	196	192	188	183	180	178	-1	4	2	-4	1	-1	-2	-3	
	203	203	202	200	195	189	183	180	0	-3	-2	-5	5	-2	2	-5	
Reconstructed	202	203	204	203	198	191	183	179	-2	-3	-4	-3	-1	-4	4	8	Error
Image Block	200	201	202	201	196	189	182	177	0	4	-2	-1	-1	-1	5	-2	
	200	200	199	197	192	186	181	177	0	0	1	3	8	4	6	-2	
	204	202	199	195	190	186	183	181	1	-2	0	5	1	1	4	-6	
	207	204	200	194	190	187	185	184	3	-4	0	6	-2	-2	2	2	
	$\tilde{f}(i, j)$								$(i, j) = f(i, j) - \tilde{f}(i, j)$								

Fig. 9.2 (cont'd): JPEG compression for a smooth image block.



- Another  $8 \times 8$  block from the Y image of 'Lena'

70	70	100	70	87	87	150	187	-80	-40	89	-73	44	32	53	-3
85	100	96	79	87	154	87	113	-135	-59	-26	6	14	-3	-13	-28
100	85	116	79	70	87	86	196	47	-76	66	-3	-108	-78	33	59
136	69	87	200	79	71	117	96	-2	10	-18	0	33	11	-21	1
161	70	87	200	103	71	96	113	-1	-9	-22	8	32	65	-36	-1
161	123	147	133	113	113	85	161	5	-20	28	-46	3	24	-30	24
146	147	175	100	103	103	163	187	6	-20	37	-28	12	-35	33	17
156	146	189	70	113	161	163	197	-5	-23	33	-30	17	-5	-4	20
$f(i, j)$								$F(u, v)$							

Fig. 9.3: JPEG compression for a textured image block.

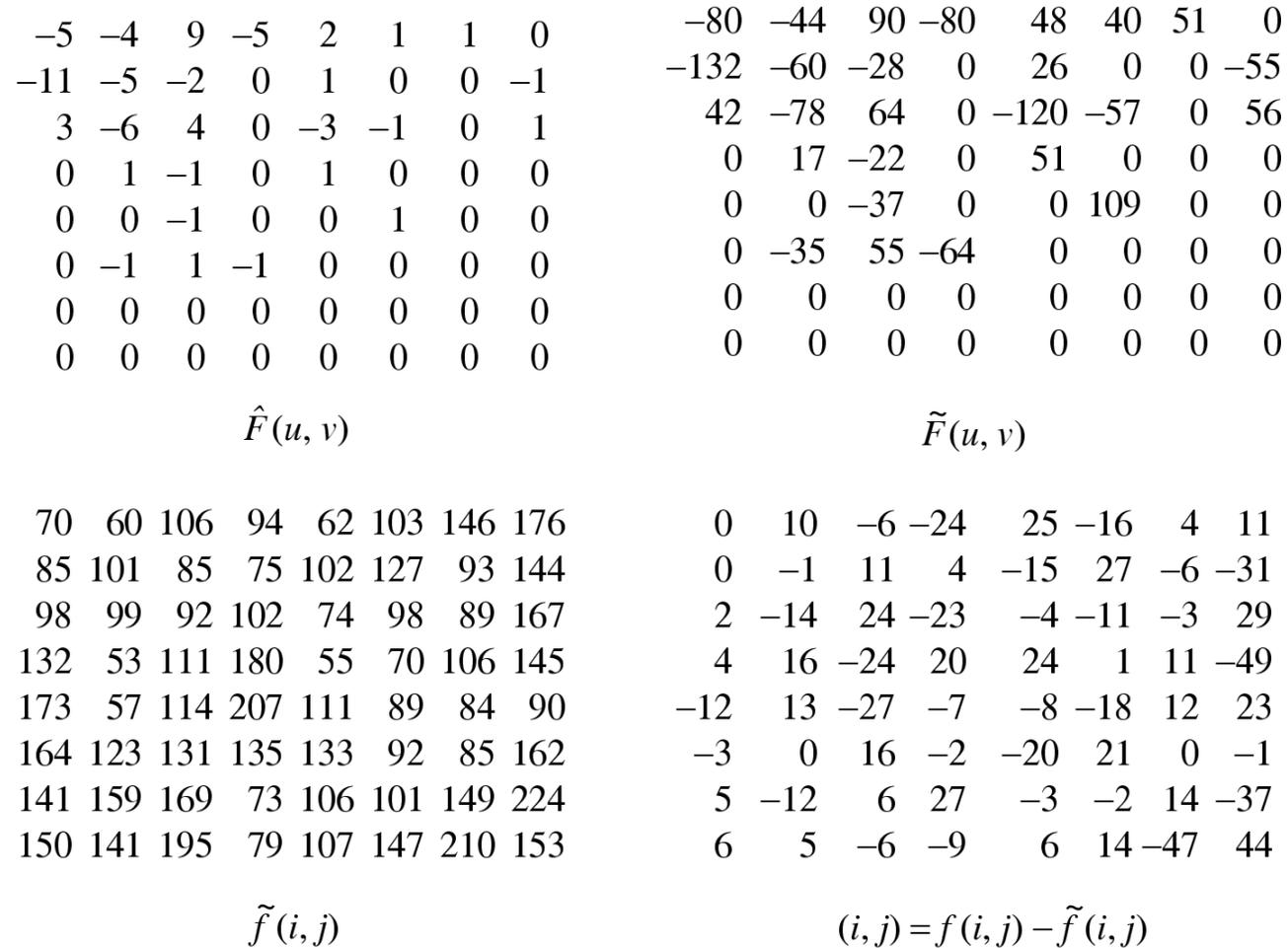
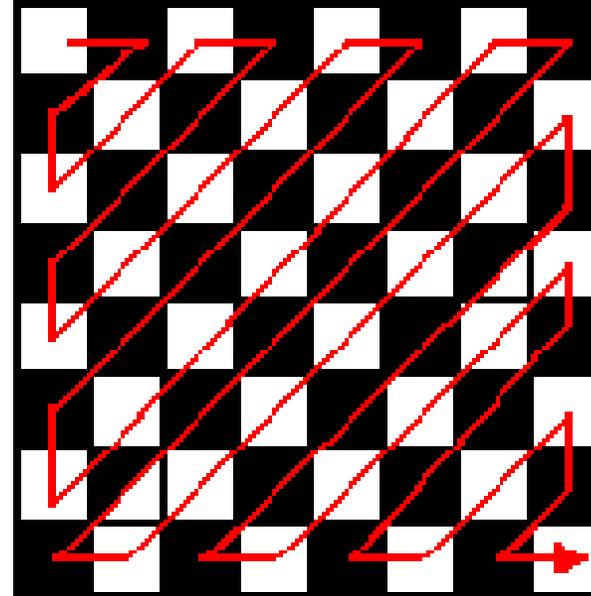


Fig. 9.3 (cont'd): JPEG compression for a textured image block.

- Selanjutnya, koefisien DCT hasil kuantisasi dibaca secara zig-zag untuk membantu tahap *entropy coding*.

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

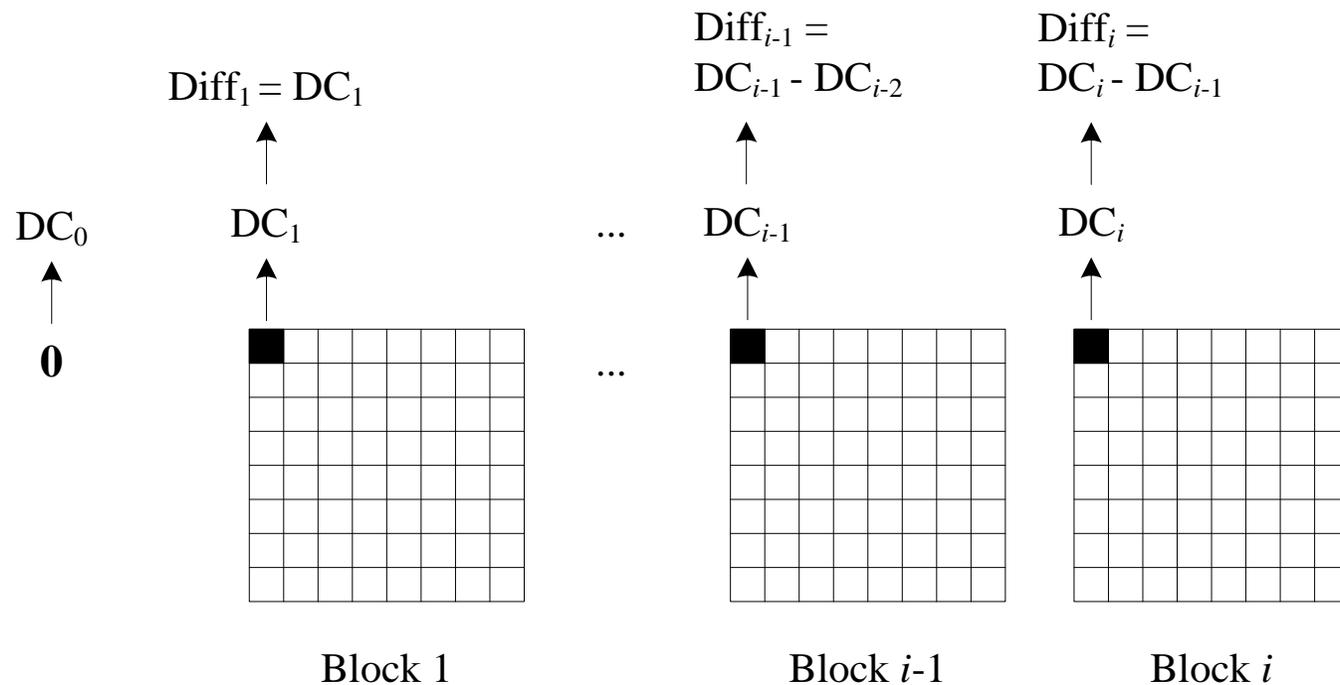


- Tahap *entropy coding* adalah *arithmetic coding* (RLE dan DPCM) dan *Huffman coding*, keduanya adalah metode *lossless compression*.
- Huffman coding* mengkompresi hasil *arithmetic coding*. Pohon Huffman kemudian disimpan sebagai *header* file JPEG.

# DPCM pada Koefisien DC

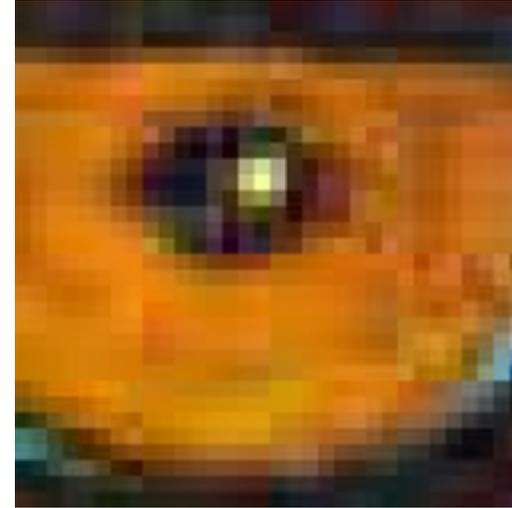
- Koefisien DC pada setiap blok dikodekan dengan DPCM sebagai berikut:

$$Diff_i = DC_{i+1} - DC_i \quad \text{dan} \quad Diff_0 = DC_0.$$



- Jika koefisien DC untuk 5 blok pertama adalah 150, 155, 149, 152, 144, maka DPCM akan menghasilkan 150, 5, -6, 3, -8.
- Selanjutnya, semua *Diff* tersebut dikodekan dengan Huffman Coding bersama-sama dengan koefisien AC





Original Image

Compressed Image

- JPEG image compression menghasilkan artefak-artefak di dalam citra hasil pemampatan, namun masih dapat ditolerir secara visual
- *Less computational complexity*

# Metrik Pengukuran Pemampatan Citra

## 1. Nisbah (*ratio*) pemampatan

Bermacam-macam rumus menghitung nisbah pemampatan

$$\text{Nisbah1} = \frac{\text{ukuran citra sesudah dimampatkan}}{\text{ukuran citra sebelum dimampatkan}} \times 100\%$$

artinya ukuran citra sekarang menjadi Nisbah1 (dalam persen) kali ukuran citra semula.

$$\text{Nisbah2} = 100\% - \frac{\text{ukuran citra sesudah dimampatkan}}{\text{ukuran citra sebelum dimampatkan}} \times 100\%$$

artinya citra sebanyak Nisbah2 (dalam persen) telah dimampatkan.

- Contoh: Citra semula berukuran  $256 \times 256$  pixels, 8-bit per pixel, grayscale.

Ukuran citra adalah 65536 byte (64 kb).

Setelah dimampatkan, ukuran citra menjadi 40280 byte

Nisbah =  $(40280/65536) \times 100\% = 61,5\%$

(artinya ukuran citra menjadi 61,5% dari ukuran semula)

Nisbah =  $100\% - 61,5\% = 38,5\%$

(artinya 38,5% citra sudah dimampatkan)

## 2. Ukuran *fidelity*

- Kualitas sebuah citra bersifat subyektif dan relatif, bergantung pada pengamatan orang yang menilainya
- Kualitas hasil pemampatan dapat diukur secara kuantitatif dengan menggunakan besaran *PSNR* (*peak signal-to-noise ratio*).
- *PSNR* dihitung untuk mengukur perbedaan antara citra semula dengan citra hasil pemampatan

$$PSNR = 20 \times \log_{10} \left( \frac{b}{rms} \right)$$

$b$  adalah nilai sinyal terbesar (pada citra dengan 256 derajat keabuan,  $b = 255$ )

$rms$  (*root mean square*) adalah akar pangkat dua dari selisih antara citra semula dengan citra hasil pemampatan

$$rms = \sqrt{\frac{1}{\text{Lebar} \times \text{Tinggi}} \sum_{i=1}^N \sum_{j=1}^M (f_{ij} - f'_{ij})^2}$$

$f$  dan  $f'$  masing-masing menyatakan nilai pixel citra semula dan nilai pixel citra hasil pemampatan

- *PSNR* memiliki satuan *decibel* (dB).
- *PSNR* berbanding terbalik dengan *rms*. Nilai *rms* yang rendah yang menyiratkan bahwa citra hasil pemampatan tidak jauh berbeda dengan citra semula, sehingga menghasilkan *PSNR* yang tinggi, yang berarti kualitas pemampatannya bagus.
- Nilai *rms* yang tinggi menyatakan galat yang besar akibat pemampatan, sehingga menghasilkan *PSNR* yang rendah.
- Semakin besar nilai *PSNR*, semakin bagus kualitas pemampatannya.
- Dalam prakteknya, nilai  $PSNR \geq 30$  menyatakan kualitas citra yang sudah dianggap baik. Di bawah 30 itu dikatakan citra mengalami degradasi yang semakin besar dengan menurunnya *PSNR*.



peppers.bmp, 256 x 256  
(193 KB)

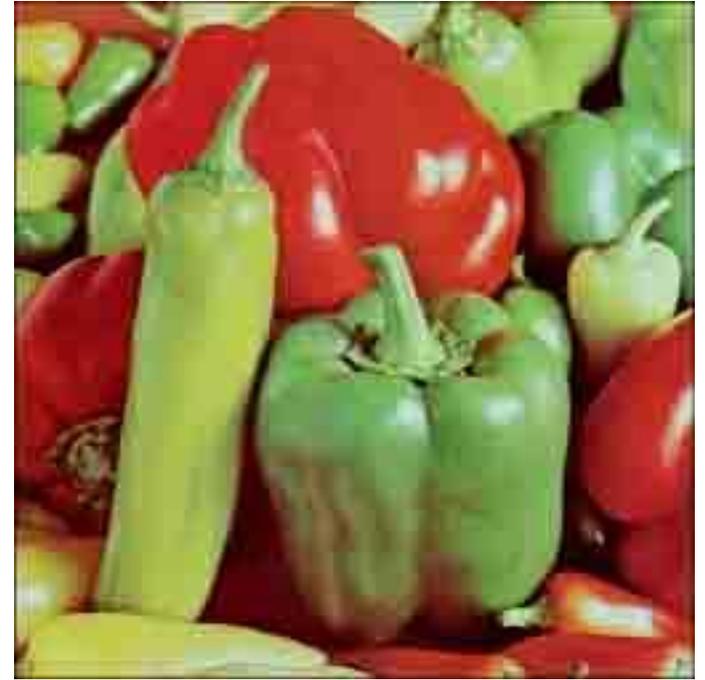


peppers.jpg, 256 x 256  
(52.2 KB), JPEG Quality = 5

```
>> ref = imread('peppers512.bmp');  
>> A = imread('peppers512-med.jpg');  
>> psnr = psnr(A, ref)
```

psnr =

35.08



peppers2.jpg, 256 x 256  
(24 KB), JPEG Quality = 1

```
>> ref = imread('peppers512.bmp');  
>> A = imread('peppers512-low.jpg');  
>> psnr = psnr(A, ref)
```

psnr =

30.5051

# Fractal Image Compression

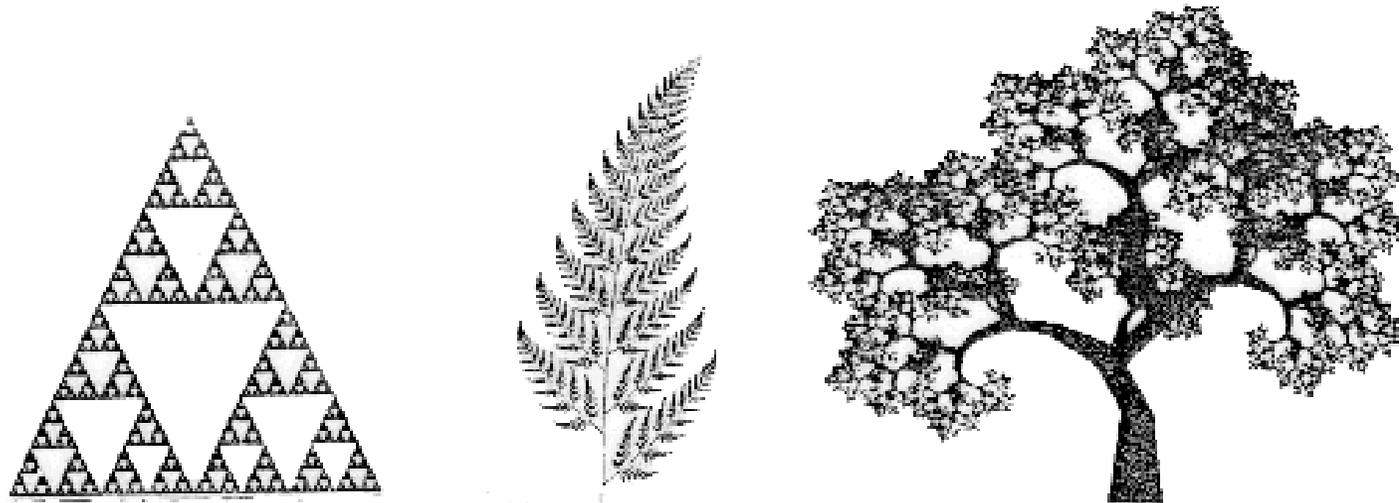
- Algoritma pemampatan citra dengan cara kerja yang unik.
- Prinsip: mencari bagian di dalam citra yang memiliki kemiripan dengan bagian lainya namun ukurannya lebih besar (*self similarity*).
- Cari matriks yang mentransformasikan bagian yang lebih besar tersebut dengan bagian yang lebih kecil.
- Simpan hanya elemen-elemen dari sekumpulan matriks transformasi tersebut (yang disebut matriks transformasi *affine*).
- Pada proses penirmampatan, matriks ransformasi *affine* di-iterasi sejumlah kali terhadap sembarang citra awal.

# Fraktal

## Definisi

**Fraktal:** objek yang memiliki kemiripan dirinya-sendiri (*self-similarity*) namun dalam skala yang berbeda.

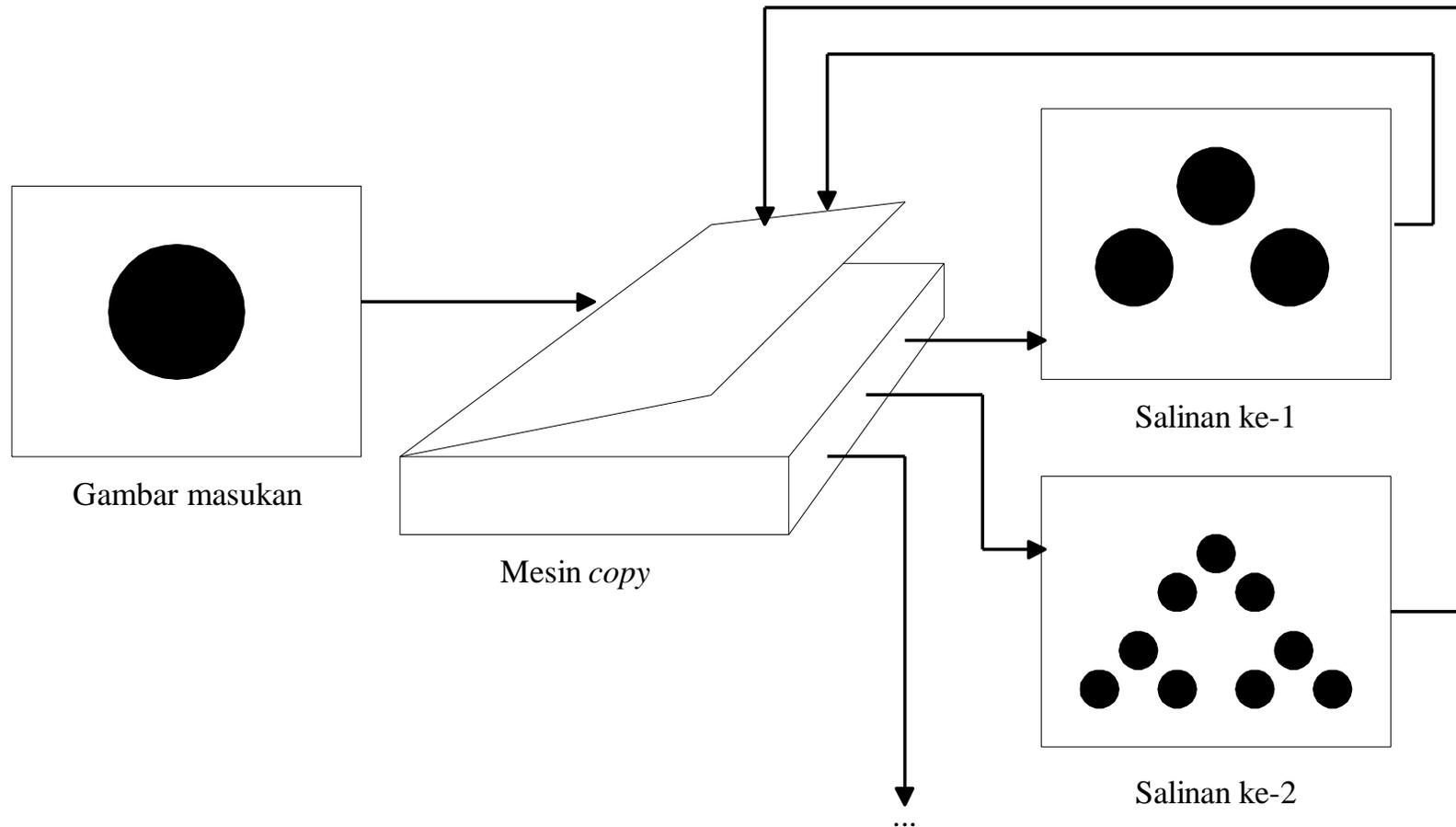
**Fraktal:** objek yang memiliki matra berupa pecahan (*fractional*). Kata terakhir inilah yang menurunkan kata **fraktal**



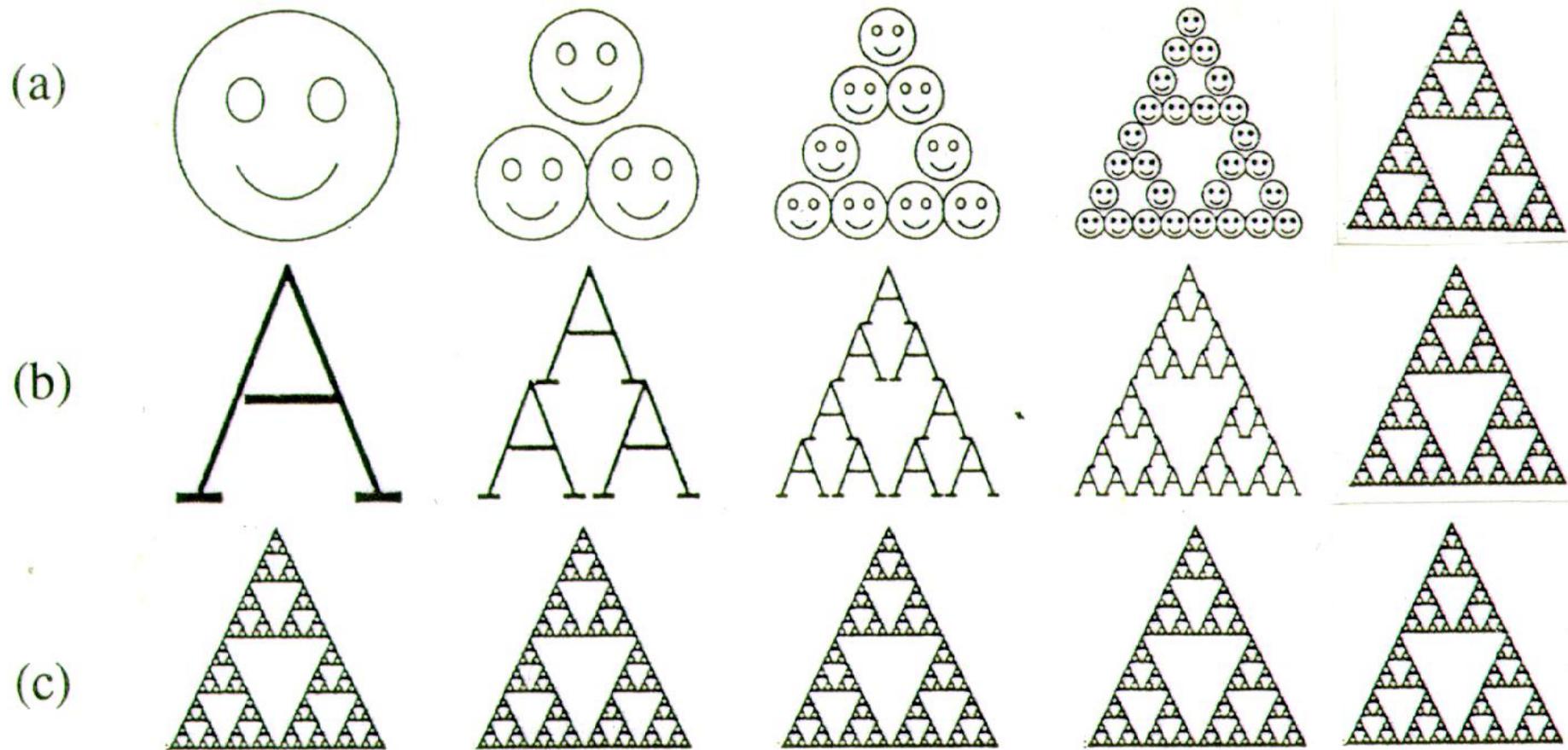
Segitiga Sierpinski, daun pakis Barsnsley, dan pohon fractal

# *Iterated Function System (IFS)*

Penemu: Michael Barnsley (1988)



*Multiple Reduction Copy Machine (MRCM)*



*Apapun gambar awalnya, MRCM selalu menghasilkan segitiga Sierpienski .*

- MRCM dapat dinyatakan dalam bentuk transformasi *affine*:

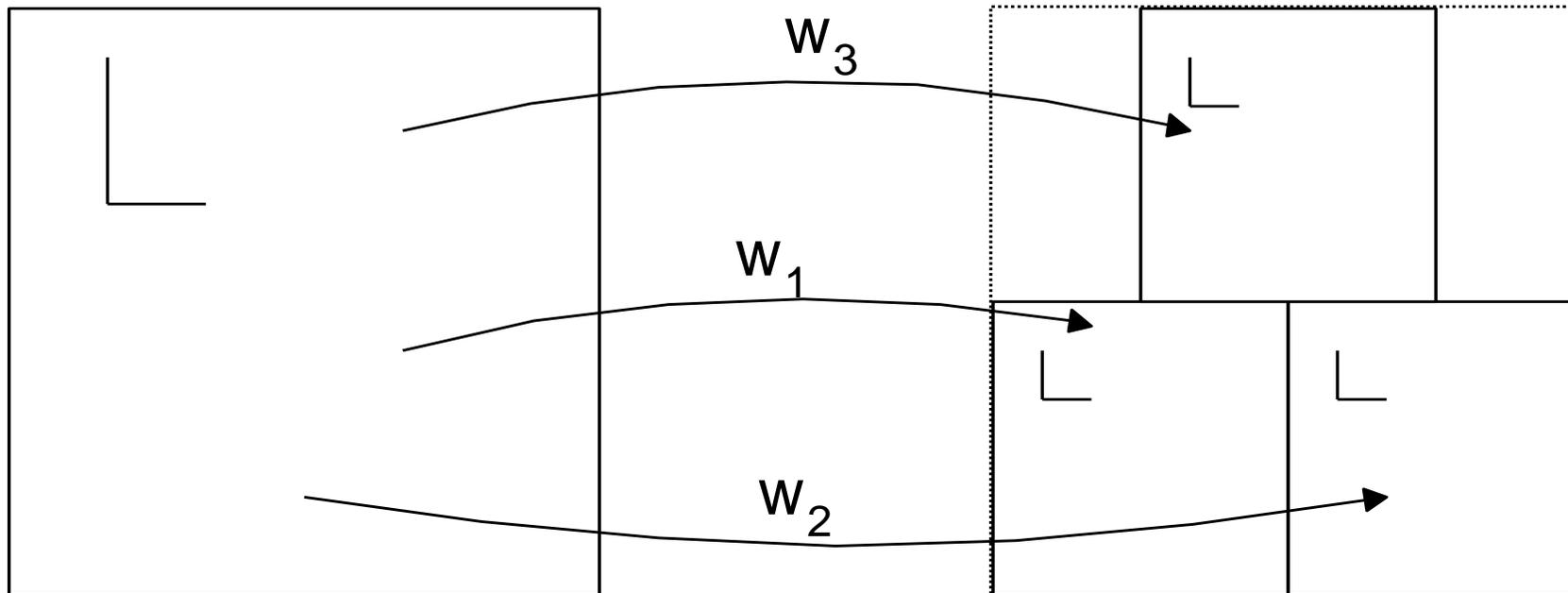
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = w \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} = Ax + t$$

- Untuk sembarang citra awal  $A$ , dihasilkan salinan *affine*,  $w_1(A)$ ,  $w_2(A)$ , ...,  $w_n(A)$ .
- Gabungan dari seluruh salinan tersebut adalah  $W(A)$ , yang merupakan keluaran dari mesin,

$$W(A) = w_1(A) + w_2(A) + \dots + w_n(A)$$

- Transformasi *affine* yang menghasilkan citra segitiga Sierpinski:

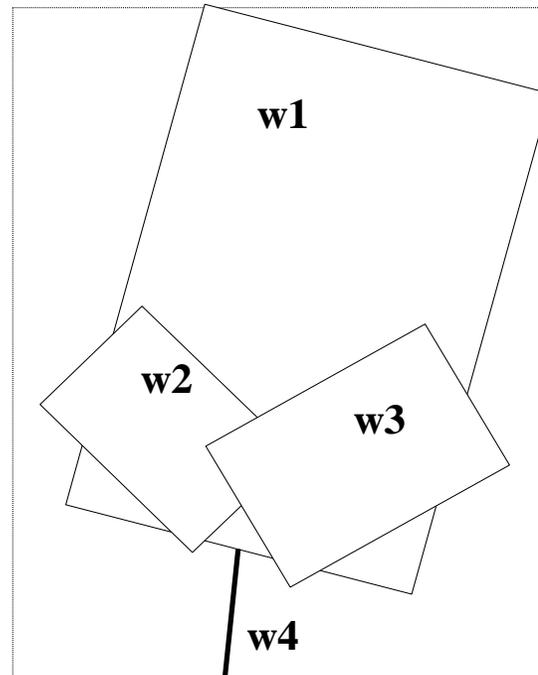
$$w_1 = \begin{bmatrix} 0.5 & 0.0 & 0.0 \\ 0.0 & 0.5 & 0.0 \end{bmatrix} \quad w_2 = \begin{bmatrix} 0.5 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 \end{bmatrix} \quad w_3 = \begin{bmatrix} 0.5 & 0.0 & 0.25 \\ 0.0 & 0.5 & 0.5 \end{bmatrix}$$



- Transformasi *affine* yang menghasilkan citra daun pakis:

$$w_1 = \begin{bmatrix} 0.85 & 0.04 & 0.0 \\ -0.04 & 0.85 & 1.6 \end{bmatrix} \quad w_2 = \begin{bmatrix} 0.20 & -0.26 & 0.0 \\ 0.23 & 0.22 & 1.6 \end{bmatrix}$$

$$w_3 = \begin{bmatrix} -0.15 & 0.28 & 0.0 \\ 0.26 & 0.52 & 0.44 \end{bmatrix} \quad w_4 = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.16 & 0.0 \end{bmatrix}$$



- Menyimpan citra sebagai kumpulan pixel membutuhkan memori yang besar, namun bila yang disimpan adalah transformasi affine-nya, maka memori yang dibutuhkan jauh lebih sedikit.
- Cara ini melahirkan gagasan pengkodean citra dengan nisbah pemampatan yang tinggi.
- Pakis Barnsley misalnya, dibangkitkan dengan empat buah transformasi affine, masing-masingnya terdiri atas enam buah bilangan riil (4 byte), sehingga dibutuhkan  $4 \times 6 \times 4 \text{ byte} = 96 \text{ byte}$  untuk menyimpan keempat transformasi itu.
- Bandingkan bila citra pakis Barnsley disimpan dengan representasi pixel hitam putih (1 pixel = 1 byte) berukuran  $550 \times 480$  membutuhkan memori sebesar 264.000 byte. Maka, nisbah pemampatan citra pakis adalah  $264.000 : 96 = 2750 : 1$ , suatu nisbah yang sangat tinggi.

# *Partitioned Iterated Function System (PIFS)*

Penemu: Arnaud D. Jacquin (1992), mahasiswa bimbingan Michael Barnsley

Dasar pemikiran:

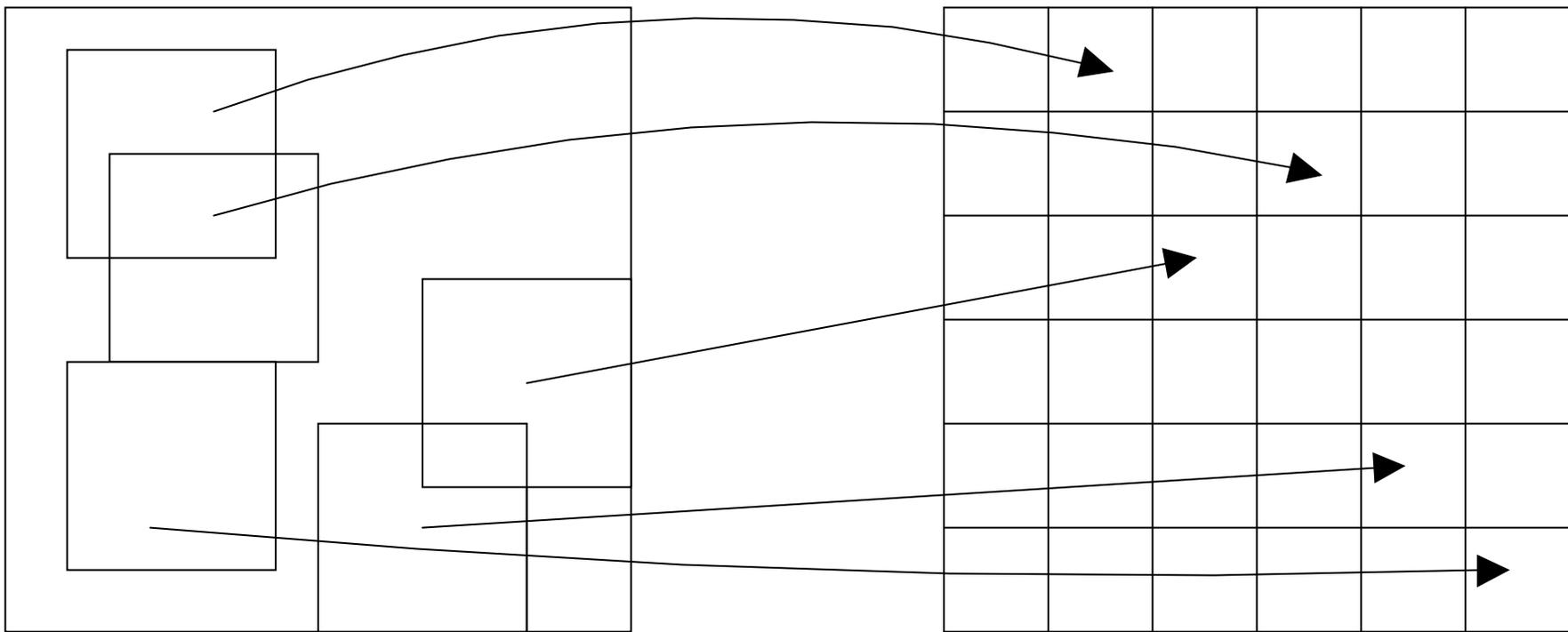
- Citra alami (*natural image*) umumnya hampir tidak pernah *self-similar* secara keseluruhan.
- Karena itu, citra alami pada umumnya tidak mempunyai transformasi *affine* terhadap dirinya sendiri.
- Tetapi, untungnya citra alami seringkali memiliki *self-similarity* lokal, yaitu memiliki bagian citra yang mirip dengan bagian lainnya.
- Setiap transformasi itu dari bagian citra ke bagian citra lain yang mirip dapat direpresentasikan dengan transformasi *IFS* lokal atau *Partitioned Iterated Function System (PIFS)*



*Kemiripan lokal pada citra Lena*

## Algoritma:

1. Bagi citra atas sejumlah blok yang berukuran sama dan tidak saling beririsan, yang disebut blok jelajah (*range*).
2. Untuk setiap blok jelajah, cari bagian citra yang berukuran lebih besar dari blok jelajah –yang disebut blok ranah (*domain*)- dan paling mirip (cocok) dengan blok jelajah tersebut.
3. Turunkan transformasi *affine* (*IFS* lokal)  $w_i$  yang memetakan blok ranah ke blok jelajah.
4. Hasil dari semua pemasangan ini adalah *Partitioned Iterated Function System* (*PIFS*).



Blok ranah

Blok jelajah

*Pemetaan dari blok ranah ke blok jelajah*

- Kemiripan antara dua buah (blok) citra diukur dengan metrik jarak. Metrik jarak yang digunakan misalnya *rms* (*root mean square*):

$$d_{rms} = \frac{1}{n} \sqrt{\sum_{i=1}^n \sum_{j=1}^n (z'_{ij} - z_{ij})^2}$$

$z$  dan  $z'$  adalah nilai *pixel* dari dua buah blok, dan  $n$  = jumlah *pixel* di dalam citra

- Ukuran blok ranah diambil dua kali blok jelajah.
- Contoh: Untuk blok jelajah  $8 \times 8$  *pixel* dan blok ranah berukuran  $16 \times 16$  *pixel*, citra  $256 \times 256$  dibagi menjadi 1024 buah blok jelajah yang tidak saling beririsan dan  $(256 - 16 + 1)^2 = 58.081$  buah blok ranah berbeda (yang beririsan).
- Himpunan blok ranah yang digunakan dalam proses pencarian kemiripan dimasukkan ke dalam *pul ranah* (*domain pool*).
- Pul ranah yang besar menghasilkan kualitas pemampatan yang lebih baik, tetapi membutuhkan waktu pencocokan yang lebih lama.

- Transformasi affine di dalam PIFS memiliki komponen z selain komponen koordinat (x,y):

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix}$$

- $s_i$  menyatakan faktor kontras *pixel* (nilainya antara 0 dan 1)
  - $o_i$  menyatakan ofset kecerahan (*brightness*) *pixel*
  - $z' = s_i z + o_i$
- Dengan asumsi ukuran blok ranah = dua kali ukuran blok jelaja (2:1), maka transformasi *affine* menjadi lebih sederhana, yaitu:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix}$$

- Parameter  $e_i$  dan  $f_i$  mudah dihitung karena keduanya menyatakan pergeseran sudut kiri blok ranah ke sudut kiri blok jelajah yang bersesuaian.

- Sedangkan  $s_i$  dan  $o_i$  dihitung dengan menggunakan rumus regresi berikut:

$$s = \frac{\left[ n \sum_{i=1}^n d_i r_i - \sum_{i=1}^n d_i \sum_{i=1}^n r_i \right]}{\left[ n \sum_{i=1}^n d_i^2 - \left( \sum_{i=1}^n d_i \right)^2 \right]} \quad o = \frac{1}{n} \left[ \sum_{i=1}^n r_i - s \sum_{i=1}^n d_i \right]$$

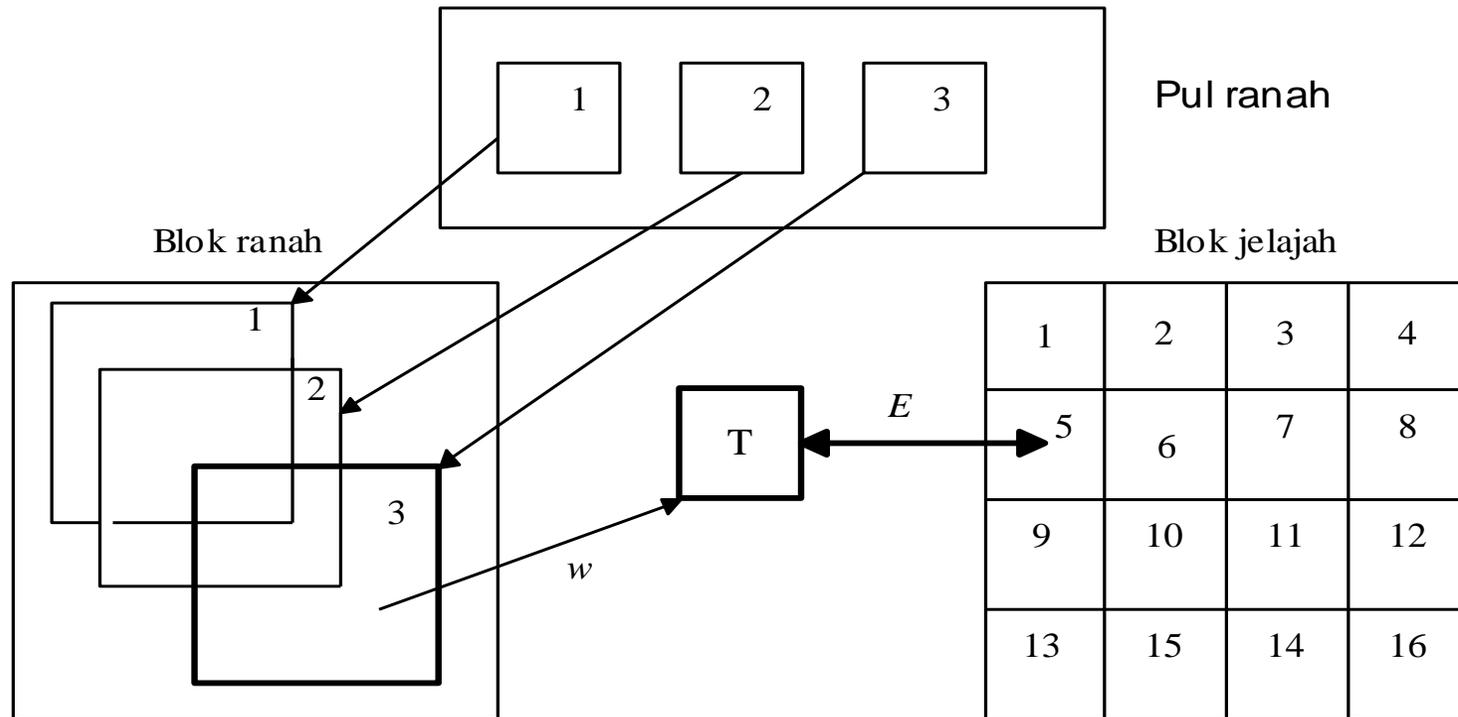
- Dengan nilai  $s$  dan  $o$  yang telah diperoleh, maka kuadrat galat antara blok jelajah dan blok ranah adalah

$$E = \frac{1}{n} \left[ \sum_{i=1}^n r_i^2 + s \left( s \sum_{i=1}^n d_i^2 - 2 \sum_{i=1}^n d_i r_i + 2o \sum_{i=1}^n d_i \right) + o \left( no - 2 \sum_{i=1}^n r_i \right) \right]$$

- Lalu hitung  $rms$  sebagai berikut:

$$d_{rms} = \sqrt{E} / n$$

- Transformasi *affine*  $w_i$  diuji terhadap blok ranah  $D_i$  menghasilkan blok uji  $T_i = w_i(D_i)$ .
- Jarak antara  $T$  dan  $R_i$  dihitung dengan persamaan  $d_{rms}$ .
- Transformasi *affine* yang terbaik ialah  $w$  yang meminimumkan  $d_{rms}$ .

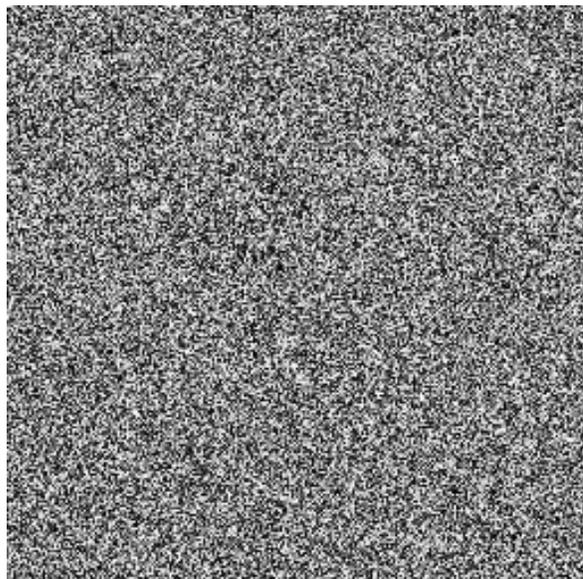


*Blok jelajah 5 dibandingkan dengan blok ranah 3 di dalam pul ranah. Transformasi  $w$  ditentukan, lalu blok ranah 3 ditransformasikan dengan  $w$  menghasilkan  $T$ . Jarak antara  $T$  dengan blok jelajah 5 diukur.*

- Runtunan pencarian dilanjutkan untuk blok jelajah berikutnya sampai seluruh blok jelajah sudah dipasangkan dengan blok ranah.
- Hasil dari proses pemampatan adalah sejumlah *IFS* lokal yang disebut *PIFS*.
- Seluruh parameter *PIFS* di-pak dan disimpan di dalam berkas eksternal. Parameter *PIFS* yang perlu disimpan hanya  $e_i, f_i, s_i, o_i$ .
- Algoritma pencocokan blok yang dijelaskan di atas adalah algoritma *brute force*, karena untuk setiap blok jelajah pencocokan dilakukan dengan seluruh blok ranah di dalam pul untuk memperoleh pencocokan terbaik.

## Rekonstruksi Citra (penirmpatan)

- Rekonstruksi (dekompresi) citra dilakukan dengan melelarkan *PIFS* dari citra awal sembarang.
- Karena setiap *PIFS* lokal kontraktif, baik kontraktif dalam matra intensitas maupun kontraktif dalam matra spasial maka lelarannya akan konvergen ke citra titik-tetap *PIFS*.
- Kontraktif intensitas penting untuk menjamin konvergensi ke citra semula, sedangkan kontraktif spasial berguna untuk membuat rincian pada citra untuk setiap skala.
- Konvergensi ke citra titik-tetap berlangsung cepat. Konvergensi umumnya dapat diperoleh dalam 8 sampai 10 kali lelaran



Citra awal



Citra lelaran ke-1



Citra lelaran ke-2



Citra lelaran ke-6

- Beberapa hasil pemampatan fraktal:



Citra asli ( $256 \times 256$  pixel)  
*Lena.bmp* (66 KB)



Citra hasil pemampatan fraktal  
*Lena.fra* (7 KB)



Citra asli ( $256 \times 256$  pixel)  
*Collie.bmp* (66 KB)



Citra hasil pemampatan fraktal  
*Collie.fra* (9 KB)



Citra asli ( $512 \times 512$  pixel)  
*Kapal.bmp* (258 KB)



Citra hasil pemampatan fraktal  
*Kapal.fra* (9 KB)



Citra asli ( $316 \times 404$  pixel)  
*Potret.bmp* (126 KB)



Citra hasil pemampatan fraktal  
*Potret.fra* (17 KB)

**Tabel 1.** Perbandingan ukuran berkas citra sebelum dan sesudah dimampatkan

<b>No.</b>	<b>Citra BMP (<i>byte</i>)</b>	<b>Ukuran (<i>byte</i>)</b>	<b>Citra FRA (<i>byte</i>)</b>	<b>Ukuran</b>	<b>Nisbah (%)</b>
1	Kapal.bmp	263.222	KAPAL512.FRA	8.956	96,6
2	Lena.bmp	66.614	LENA256.FRA	8.137	87,6
3	Collie.bmp	66.614	COLLI256.FRA	9.150	86,3
4	Potret.bmp	128.782	POTRET.FRA	17.437	86,5

**Tabel 2.** Perbandingan ukuran citra berformat BMP, JPG, GIF, dan fraktal(FRA)

<b>Nama Citra</b>	<b>Format BMP (byte)</b>	<b>Format JPG (byte)</b>	<b>Format GIF (byte)</b>	<b>Format FRA (byte)</b>
<i>Kapal.bmp</i>	263.222	24.367	242.452	8.956
<i>Lena.bmp</i>	66.614	7.126	70.292	8.137
<i>Collie.bmp</i>	66.614	7.021	69.965	9.150
<i>Potret.bmp</i>	128.782	16.377	136.377	17.437