Bab 10

Pemampatan Citra

Pada umumnya, representasi citra digital membutuhkan memori yang besar. Sebagai contoh, citra Lena dalam format *bitmap* yang berukuran 512 × 512 *pixel* membutuhkan memori sebesar 32 KB (1 *pixel* = 1 *byte*) untuk representasinya. Semakin besar ukuran citra tentu semakin besar pula memori yang dibutuhkannya. Pada sisi lain, kebanyakan citra mengandung duplikasi data. Duplikasi data pada citra dapat berarti dua hal. Pertama, besar kemungkinan suatu *pixel* dengan *pixel* tetanggganya memiliki initensitas yang sama, sehingga penyimpanan setiap *pixel* memboroskan tempat. Kedua, citra banyak mengandung bagian (*region*) yang sama, sehingga bagian yang sama ini tidak perlu dikodekan berulang kali karena mubazir atau redundan.

Saat ini, kebanyakan aplikasi menginginkan representasi citra dengan kebutuhan memori yang sesedikit mungkin. Pemampatan citra atau kompresi citra (*image compression*) bertujuan meminimalkan kebutuhan memori untuk merepresentasikan citra digital. Prinsip umum yang digunakan pada proses pemampatan citra adalah mengurangi duplikasi data di dalam citra sehingga memori yang dibutuhkan untuk merepresentasikan citra menjadi lebih sedikit daripada representasi citra semula.

10.1 Pemampatan Citra versus Pengkodean Citra

Pemampatan citra kadang-kadang disalahmengertikan dengan pengkodean citra (*image encoding*), yaitu persoalan bagaimana *pixel-pixel* di dalam citra dikodekan dengan representasi tertentu. Pengkodean citra tidak selalu menghasilkan representasi memori yang minimal. Pengkodean citra yang menghasilkan

representasi memori yang lebih sedikit daripada representasi aslinya itulah yang dinamakan pemampatan citra.

Ada dua proses utama dalam persoalan pemampatan citra:

1. Pemampatan citra (*image compression*).
   Pada proses ini, citra dalam representasi tidak mampat dikodekan dengan representasi yang meminimumkan kebutuhan memori. Citra dengan format *bitmap* pada umumnya tidak dalam bentuk mampat. Citra yang sudah dimampatkan disimpan ke dalam arsip dengan format tertentu. Kita mengenal format JPG dan GIF sebagai format citra yang sudah dimampatkan.

2. Penirmampatkan citra (*image decompression*).
   Pada proses ini, citra yang sudah dimampatkan harus dapat dikembalikan lagi (*decoding*) menjadi representasi yang tidak mampat. Proses ini diperlukan jika citra tersebut ditampilkan ke layar atau disimpan ke dalam arsip dengan format tidak mampat. Dengan kata lain, penirmampatan citra mengembalikan citra yang termampatkan menjadi data *bitmap*.

10.2 Aplikasi Pemampatan Citra

Pemampatan citra memberikan sumbangsih manfaat yang besar dalam industri multimedia saat ini. Pemampatan citra bermanfaat untuk aplikasi yang melakukan:

1. Pengiriman data (*data transmission*) pada saluran komunikasi data
   Citra yang telah dimampatkan membutuhkan waktu pengiriman yang lebih singkat dibandingkan dengan citra yang tidak dimampatkan. Contohnya aplikasi pengiriman gambar lewat *fax*, *videoconferencing*, pengiriman data medis, pengiriman gambar dari satelit luar angkasa, pengiriman gambar via telepon genggam. *download* gambar dari internet, dan sebagainya.

2. Penyimpanan data (*data storing*) di dalam media sekunder (*storage*)
   Citra yang telah dimampatkan membutuhkan ruang memori di dalam media *storage* yang lebih sedikit dibandingkan dengan citra yang tidak dimampatkan. Contoh aplikasi nya antara lain aplikasi basisdata gambar, *office automation*, *video storage* (seperti *Video Compact Disc*), dll.

10.3 Kriteria Pemampatan Citra

Saat ini sudah banyak ditemukan metode-metode pemampatan citra. Kriteria yang digunakan dalam mengukur metode pemampatan citra adalah [LOW91]:

1. Waktu pemampatan dan penirmampatan (*decompression*).
   Waktu pemampatan citra dan penirmampatannya sebaiknya cepat. Ada metode pemampatan yang waktu pemampatannya lama, namun waktu penirmampatannya cepat. Ada pula metode yang waktu pemampatannya cepat tetapi waktu penirmampatannya lambat. Tetapi ada pula metode yang waktu pemampatan dan penirmampatannya cepat atau keduanya lambat.

2. Kebutuhan memori.
   Memori yang dibutuhkan untuk merepresentasikan citra seharusnya berkurang secara berarti. Ada metode yang berhasil memampatkan dengan persentase yang besar, ada pula yang kecil. Pada beberapa metode, ukuran memori hasil pemampatan bergantung pada citra itu sendiri. Cira yang mengandung banyak elemen duplikasi (misalnya citra langit cerah tanpa awan, citra lantai keramik) umumnya berhasil dimampatkan dengan memori yang lebih sedikit dibandingkan dengan memampatkan citra yang mengandung banyak objek (misalnya citra pemandangan alam).

3. Kualitas pemampatan (*fidelity*)
   Informasi yang hilang akibat pemampatan seharusnya seminimal mungkin sehingga kualitas hasil pemampatan tetap dipertahankan. Kualitas pemampatan dengan kebutuhan memori biasanya berbanding terbalik. Kualitas pemampatan yang bagus umumnya dicapai pada proses pemampatan yang menghasilkan pengurangan memori yang tidak begitu besar, demikian pula sebaiknya. Dengan kata lain, ada timbal balik (*trade off*) antara kualitas citra dengan ukuran hasil pemampatan.

   Kualitas sebuah citra bersifat subyektif dan relatif, bergantung pada pengamatan orang yang menilainya. Seseorang dapat saja mengatakan kualitas suatu citra bagus, tetapi orang lain mungkin mengatakan kurang bagus, jelek, dan sebagainya.

   Kita dapat membuat ukuran kualitas hasil pemampatan citra menjadi ukuran kuantitatif dengan menggunakan besaran *PSNR* (*peak signal-to-noise ratio*). *PSNR* dihitung untuk mengukur perbedaan antara citra semula dengan citra hasil pemampatan (tentu saja citra hasil pemampatan harus dinirmampatkan terlebih dahulu) dengan citra semula, dengan rumus:

$$PSNR = 20 \times \log_{10}\left(\frac{b}{rms}\right) \qquad (10.1)$$

   dengan b adalah nilai sinyal terbesar (pada citra hitam-putih, b = 255) dan *rms* adalah akar pangkat dua dari selisih antara citra semula dengan citra hasil pemampatan. Nila *rms* dihitung dengan rumus:

$$rms = \sqrt{\frac{1}{\text{Lebar} \times \text{Tinggi}} \sum_{i=1}^{N} \sum_{j=1}^{M} (f_{ij} - f'_{ij})^2} \qquad (10.2)$$

yang dalam hal ini, f dan f' masing-masing menyatakan nilai *pixel* citra semula dan nilai *pixel* citra hasil pemampatan. *PSNR* memiliki satuan decibel (dB). Persamaan (10.2) menyatakan bahwa *PSNR* hanya dapa dihitung setelah proses pernirmapatan citra. Dari persamaan (10.2) terlihat abhwa *PSNR* berbanding terbalik dengan *rms*. Nilai *rms* yang rendah yang menyiratkan bahwa citra hasil pemampatan tidak jauh berbeda dengan citra semula akan menghasilkan *PSNR* yang tinggi, yang berarti kualitas pemampatannya bagus. Semakin besar nilai *PSNR*, semakin bagus kualitas pemampatannya. Seberapa besar nilai *PSNR* yang bagus tidak dapat dinyatakan secara eksplisit, bergantung pada citra yang dimampatkan. Namun kita dapat mengetahui hal ini jika kita melakukan pengujian dengan mencoba berbagai kombinasi parameter pemampatan yang digunakan. Jika nilai *PSNR* semakin membesar, itu berarti parameter pemampatan yang digunakan sudah menuju nilai yang baik. Parameter pemampatan citra bergantung pada metode pemamapatan yang digunakan.

4. Format keluaran
   Format citra hasil pemampatan sebaiknya cocok untuk pengiriman dan penyimpanan data. Pembacaan citra bergantung pada bagaimana citra tersebut direpresentasikan (atau disimpan).

Pemilihan kriteria yang tepat bergantung pada pengguna dan aplikasi. Misalnya, apakah pengguna menginginkan pemampatan yang menghasilkan kualitas yang bagus, namun pengurangan memori yang dibutuhkan tidak terlalu besar, atau sebaliknya. Atau jika waktu pemampatan dapat diabaikan dari pertimbangan (dengan asumsi bahwa pemampatan hanya sekali saja dilakukan, namun pernirmampatan dapat berkali-kali), maka metode yang menghasilkan waktu penirmampatan yang cepat yang perlu dipertimbangkan.

10.4 Jenis Pemampatan Citra

Ada empat pendekatan yang digunakan dalam pemampatan citra [LOW91]:

1. Pendekatan statistik.
   Pemampatan citra didasarkan pada frekuensi kemunculan derajat keabuan *pixel* di dalam seluruh bagian gambar.
   Contoh metode: *Huffman Coding*.

2. Pendekatan ruang
   Pemampatan citra didasarkan pada hubungan spasial antara *pixel-pixel* di dalam suatu kelompok yang memiliki derajat keabuan yang sama di dalam suatu daerah di dalam gambar.
   Contoh metode: *Run-Length Encoding*.

3. Pendekatan kuantisasi
   Pemampatan citra dilakukan dengan mengurangi jumlah derajat keabuan yang tersedia.
   Contoh metode: metode pemampatan kuantisasi.

4. Pendekatan fraktal
   Pemampatan citra didasarkan pada kenyataan bahwa kemiripan bagian-bagian di dalam citra dapat dieksploitasi dengan suatu matriks transformasi.
   Contoh metode: *Fractal Image Compression*.

10.5 Klasifikasi Metode Pemampatan

Metode pemampatan citra dapat diklasifiksikan ke dalam dua kelompok besar:

1. Metode *lossless*
   Metode *lossless* selalu menghasilkan citra hasil penirmampatan yang tepat sama dengan citra semula, *pixel* per *pixel*. Tidak ada informasi yang hilang akibat pemampatan. Sayangnya nisbah (*ratio*) pemampatan citra metode *lossless* sangat rendah.
   Contoh metode *lossless* adalah metode *Huffman*.

   Nisbah pemampatan citra dihitung dengan rumus

$$\text{Nisbah} = 100\% - \left(\frac{\text{ukuran citra hasil pempatatan}}{\text{ukuran citra semula}} \times 100\%\right) \qquad (10.3)$$

   Metode *lossless* cocok untuk memampatkan citra yang mengandung informasi penting yang tidak boleh rusak akibat pemampatan. Misalnya memampatkan gambar hasil diagnosa medis.

2. Metode *lossy*
   Metode *lossy* menghasilkan citra hasil pemampatan yang *hampir* sama dengan citra semula. Ada informasi yang hilang akibat pemampatan, tetapi dapat ditolerir oleh persepsi mata. Mata tidak dapat membedakan perubahan kecil pada gambar. Metode pemampatan *lossy* menghasilkan nisbah pemampatan yang tinggi daripada metode *lossless*. Gambar 10.1 adalah citra sebelum dimampatkan, dan Gambar 10.2 adalah hasil pemampatan citra kapal dengan metode *lossy*.
   Contoh metode *lossy* adalah metode JPEG dan metode fraktal.

Gambar 10.1 *Citra kapal sebelum dimampatkan*

Gambar 10.2 *Citra kapal setelah dimampatkan dengan sebuah metode lossy*

10.6 Metode Pemampatan Huffman

Metode pemampatan Huffman menggunakan prinsip bahwa nilai (atau derajat) keabuan yang sering muncul di dalam citra akan dikodekan dengan jumlah bit yang lebih sedikit sedangkan nilai keabuan yang frekuensi kemunculannya sedikit dikodekan dengan jumlah bit yang lebih panjang.

Algoritma metode Huffman:

1. Urutkan secara menaik (*ascending order*) nilai-nilai keabuan berdasarkan frekuensi kemunculannya (atau berdasarkan peluang kemunculan, p_k, yaitu frekuensi kemunculan (n_k) dibagi dengan jumlah *pixel* di dalam gambar (n)). Setiap nilai keabuan dinyatakan sebagai pohon bersimpul tunggal. Setiap simpul di-*assign* dengan frekuensi kemunculan nilai keabuan tersebut.
2. Gabung dua buah pohon yang mempunyai frekuensi kemunculan paling kecil pada sebuah akar. Akar mempunyai frekuensi yang merupakan jumlah dari frekuensi dua buah pohon penyusunnya.
3. Ulangi langkah 2 sampai tersisa hanya satu buah pohon biner.

Agar pemilihan dua pohon yang akan digabungkan berlangsung cepat, maka semua pohon yang ada selalu terurut menaik berdasarkan frekuensi.

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

Simpul-simpul daun pada pohon biner menyatakan nilai keabuan yang terdapat di dalam citra semula. Untuk mengkodekan setiap pixel di dalam di dalam citra, lakukan langkah kelima berikut:

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

Setiap kode Huffman merupakan kode prefiks, yang artinya tidak ada kode biner suatu nilai keabuan yang merupakan awalan bagi kode biner derajat keabuan yang lain. Dengan cara ini, tidak ada ambiguitas pada proses penirmampatan citra.

Contoh 10.1. Misalkan terdapat citra yang berukuran 64 × 64 dengan 8 derajat keabuan (k) dan jumlah seluruh *pixel* (n) = 64 × 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 yang terbentuk dapat dilihat pada Gambar 10.3. Setiap simpul di dalam pohon berisi pasangan nilai *a:b*, yang dalam hal ini *a* menyatakan nilai keabuan dan *b* menyatakan peluang kemunculan nilai keabuan tersebut di dalam citra. Dari pohon Huffman tersebut kita memperoleh kode untuk setiap derajat keabuan sebagai berikut:

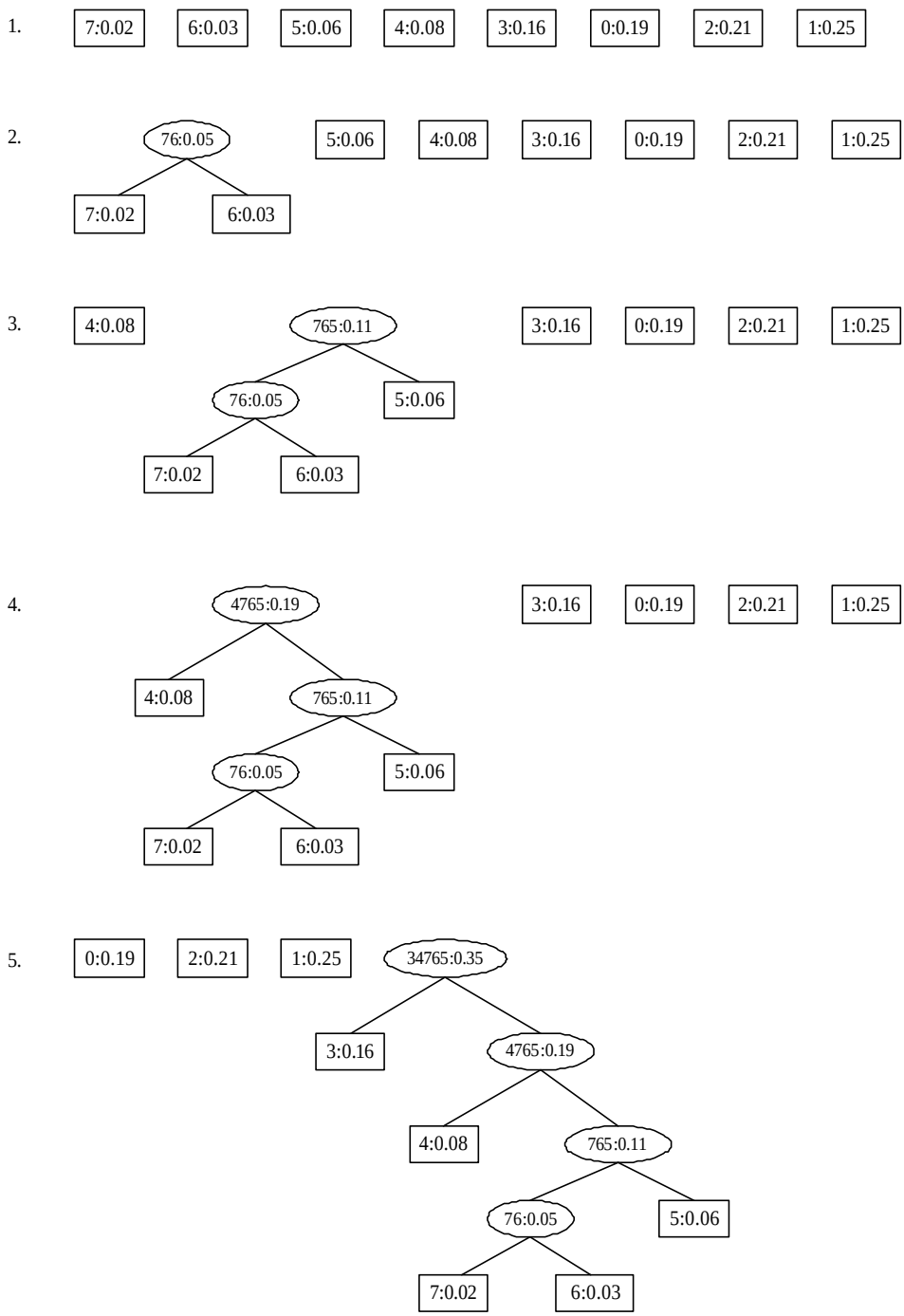
0 = 00	2 = 01	4 = 1110	6 = 111101
1 = 10	3 = 110	5 = 11111	7 = 111100

Ukuran citra sebelum pemampatan (1 derajat keabuan = 3 bit) adalah 4096 × 3 bit = 12288 bit, sedangkan Ukuran citra setelah pemampatan:

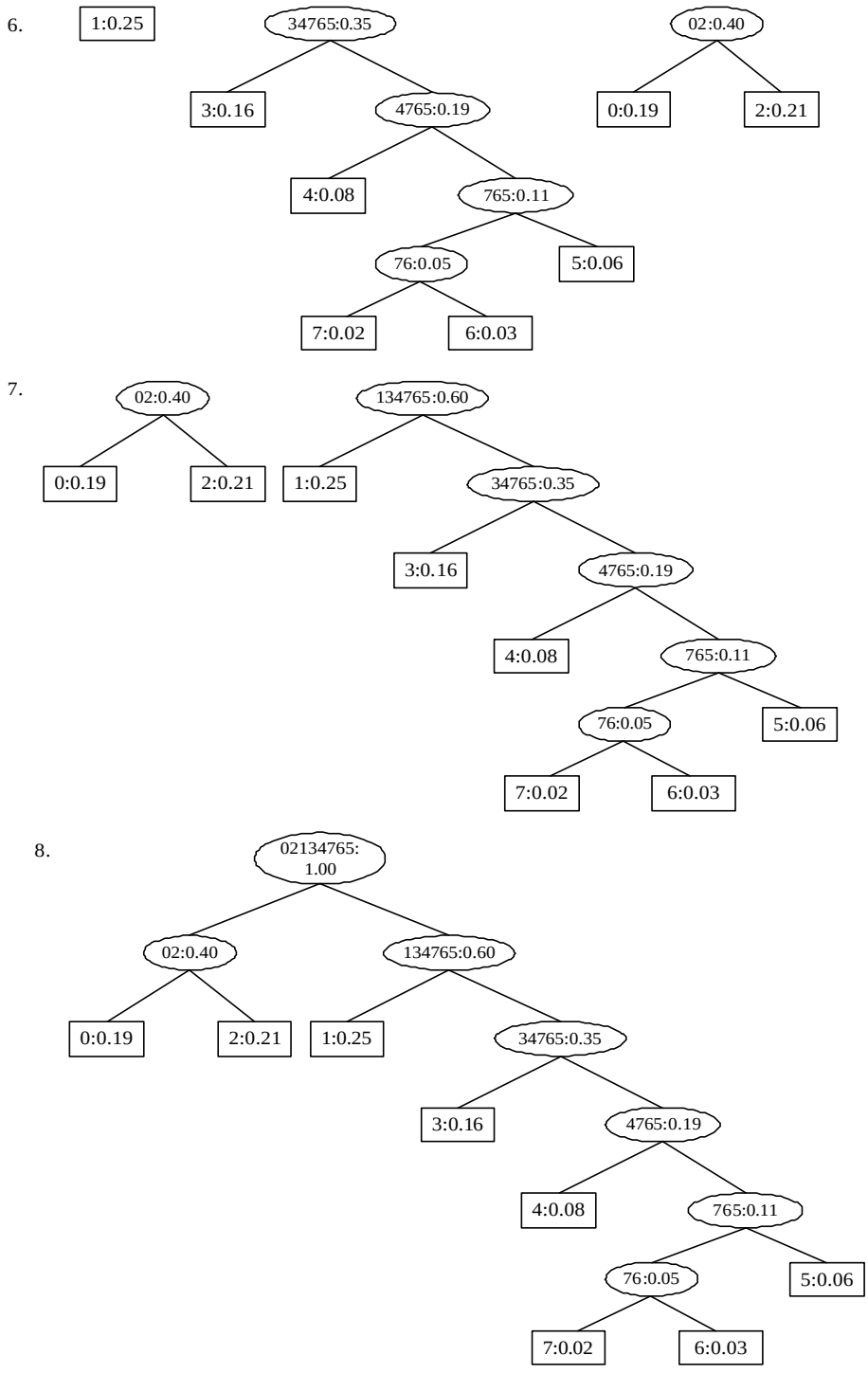
(790 × 2 bit) + (1023 × 2 bit) + (850 × 2 bit) +
(656 × 3 bit) + (329 × 4 bit) + (245 × 5 bit) +
(122 × 6 bit) + (81 × 6 bit) = 11053 bit

Jadi, kebutuhan memori telah dikurangi dari 12288 bit menjadi 11053 bit. Jelas ini tidak banyak menghemat, tetapi jika 256 nilai keabuan yang digunakan (dibanding dengan 8 derajat keabuan deperti pada contoh di atas) , penghematan memori dapat bertambah besar.

Nisbah pemampatan = $(100\% - \frac{11053}{12288} \times 100\%) = 10\%$, yang artinya 10% dari citra semula telah dimampatkan. ■



Gambar 10.3 *Tahapan pembentukan pohon Huffman untuk Contoh 10.1 di atas*



Gambar 10.3 *(lanjutan)*

10.7 Metode Pemampatan *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, sedangkan nilai kedua (*q*) menyatakan jumlah *pixel* berurutan yang memiliki derajat keabuan tersebut (dinamakan *run length*).

Contoh 10.2. [LOW91] Tinjau citra 10×10 *pixel* dengan 8 derajat keabuan yang dinyatakan sebagai matriks derajat keabuan sebagai berikut

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

semuanya ada 100 buah nilai.

Pasangan nilai untuk setiap baris *run* yang dihasilkan dengan metode pemampatan RLE:

(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 atau $31 \times 2 = 62$ nilai.

Ukuran citra sebelum pemampatan (1 derajat keabuan = 3 bit) adalah 100×3 bit = 300 bit, sedangkan ukuran citra setelah pemampatan (derajatk keabuan = 3 bit, *run length* = = 4 bit):

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

Nisbah pemampatan = $(100\% - \frac{217}{300} \times 100\%) = 27.67\%$, yang artinya 27.67% dari citra semula telah dimampatkan. ■

Versi lain dari metode *RLE* adalah dengan menyatakan seluruh baris citra menjadi sebuah baris *run*, lalu menghitung *run-length* untuk setiap derajat keabuan yang berurutan. Sebagai contoh, tinjau sebuah citra sebagai berikut:

1	2	1	1	1	1
1	3	4	4	4	4
1	1	3	3	3	5
1	1	1	1	3	3

Nyatakan sebagai barisan nilai derajat keabuan:

1 2 1 1 1 1 3 4 4 4 4 1 1 3 3 3 5 1 1 1 1 3 3

semuanya ada 24 nilai.

Pasangan nilai dari *run* yang dihasilkan dengan metode pemampatan *RLE*:

(1, 1) (2, 1) (1, 5) (3, 1) (4, 4) (1, 2) (3, 3) (5, 1) (1, 4) (3, 2)

Hasil pengkodean:

1 1 2 1 1 5 3 1 4 4 1 2 3 3 5 1 1 4 3 2

semuanya ada 20 nilai. Jadi, kita sudah menghemat 4 buah nilai.

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

10.8 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. Algoritmanya adalah sebagai berikut:

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 10.3. [LOW91] Tinjau citra yang berukuran 5×13 *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:

```
-------------------------------------------------
        0 **
13      1 **                          0
        2 **********
-------------------------------------------------
20      3 ************
        4 **********                  1
-------------------------------------------------
        5 ****
17      6 ******                      2
        7 *********
-------------------------------------------------
15      8 **********                  3
        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):

65×4 bit = 260 bit

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

65×2 bit = 130 bit

Nisbah pemampatan = $(100\% - \frac{130}{260} \times 100\%) = 50\%$, yang artinya 50% dari citra semula telah dimampatkan. ■

Kelemahan metode pemampatan kuantisasi adalah banyaknya informasi yang hilang, tapi kehilangan informasi ini dapat diminimalkan dengan menjamin bahwa tiap kelompok mempunyai jumlah *pixel* yang hampir sama.

10.9 Metode Pemampatan Fraktal

Metode pemampatan fraktal adalah metode yang relatif baru. Prinsipnya adalah mencari bagian di dalam citra yang memiliki kemiripan dengan bagian lainya namun ukurannya lebih besar (*self similarity*). Kemudian dicari matriks yang mentransformasikan bagian yang lebih besar tersebut dengan bagian yang lebih kecil. Kita cukup hanya menyimpan 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. Hasil iterasi akan konvergen ke citra semula. Metode ini menghasilkan nisbah pemampatan yang tinggi namun waktu pemampatannya relatif lama, sedangkan waktu penirmamoatannya berlangsung cepat. Metode pemampatan fraktal akan dijelaskan secara panjang lebar di dalam Bab tersendiri (Bab 14).