Aplikasi Kode Huffman dalam Kompresi Gambar Berformat JPEG

Ibnu Alam
NIM : 13506024
Jurusan Teknik Informatika ITB, Bandung , email: ibngtr@students.itb.ac.id

Abstrak –*. Makalah ini membahas pengaplikasian kode Huffman pada pengkompresian gambar berformat JPEG. Dengan membahas cara kerja kode Huffman dan hubungannya dengan entropi data, peran kode Huffman dapat diperlihatkan dalam memperkecil jumlah data yang perlu disimpan untuk merepresentasikan kembali sebuah gambar. Berhubung operasi encoding dalam aplikasi sebenarnya gambar JPEG cukup kompleks dan tidak efisien jika dilakukan tanpa bantuan komputer, pembahasan dalam makalah ini dibatasi dengan penyederhanaan dan penerangan konsep saja.*

Kata Kunci*: entropi, JPEG, Huffman, DCT*

1. PENDAHULUAN

Kompresi data adalah sebuah metode untuk menurunkan besar data dengan *embedding* data ke dalam format tertentu yang bertujuan memberikan beban minimal untuk transfer atau penyimpanan informasi data digital.

Secara umum kompresi data dilakukan dengan tiga cara:

1. *Coding*
2. *Data Reduction*
3. *Decorrelation*

2. JPEG

2.1 Definisi

Dalam ilmu komputer, JPEG (baca: /ˈdʒeɪpɛg/) adalah metode kompresi yang umum digunakan untuk gambar-gambar fotografi. JPEG merupakan singkatan dari *Joint Photographic Experts Group*, nama dari komite yang menetapkan standar JPEG. Pada tahun 1994, standar JPEG disahkan sebagai ISO 10918-1.

Standar JPEG memberikan spesifikasi *codec* kompresi data ke dalam *stream* data *byte* dan pendekompresian kembali ke bentuk gambar serta format data penyimpananannya. Metode kompresi data yang digunakan umumnya berupa *lossy compression*, yang membuang detail visual tertentu, dimana hilangnya data tersebut tidak bisa dikembalikan. File JPEG memiliki ekstensi .jpg, .jpeg, .jpe, .jfif, dan .jif.

Format JPEG banyak digunakan untuk penyimpanan dan transfer data fotografi lewat internet. Dalam hal ini format JPEG jauh lebih baik dari GIF yang menggunakan *pallete* maksimum 256 warna. Sebaliknya, algoritma kompresi JPEG tidak cocok untuk menyimpan data gambar garis, teks, dan icon. Pada kasus ini digunakan format PNG atau GIF.

2.2. Codec JPEG

Gambar dalam format JPEG umumnya dikompresi dengan menggunakan JFIF encoding:

1. Representasi warna dalam gambar diubah dari RGB (Red, Green, Blue) ke YCbCr, yaitu satu komponen *brightness*, luma (Y), dan dua komponen warna, chroma (Cb, Cr).
2. Resolusi data chroma diturunkan (*downsampling*), biasanya dengan faktor pembagian 2. Hal ini dikarenakan mata manusia lebih peka terhadap detail *brightness* daripada detail warna.
3. Gambar dibagi ke dalam blok-blok 8x8 piksel. Tiap blok akan melalui proses transformasi Discrete Cosine Transform (DCT). DCT menghasilkan spectrum frekuensi spatial dari data Y, Cb, dan Cr.
4. Amplitudo dari frekuensi komponen-komponen tesebut dikuantisasi. Mata manusia lebih sensitif terhadap variasi kecil warna atau *brightness* dalam lingkup area yang luas daripada variasi *brightness* pada frekuensi tinggi. Oleh karena itu, nilai dari komponen yang berfrekuensi tinggi disimpan dalam akurasi yang lebih rendah daripada komponen yang berfrekuensi rendah. Dalam kasus *encoding* dengan *settings* kualitas yang sangat rendah, komponen frekuensi tinggi akan dibuang seluruhnya.
5. Hasil dari setiap blok 8x8 tersebut akan dikompresi lebih lanjut dengan algoritma *loss-less* yang merupakan variasi dari Huffman *encoding*.

3. KONSEP DALAM KOMPRESI

3.1 Entropi Informasi

Entropi informasi atau entropi Shannon adalah ukuran ketidakpastian yang dihubungkan dengan variabel

acak (*random variable*). Entropi ini mengukur informasi yang terkandung dalam *bit*, ukuran terkecil yang diperlukan untuk menyampaikan informasi.

Entropi ini juga memperlihatkan batas absolut dari kompresi *lossless*. Berhubungan dengan hal ini, performa algoritma kompresi data dapat dilihat sebagai perkiraan kasar entropi suatu data.

3.2. Redundancy

Redundancy dalam teori informasi adalah banyaknya bit yang diperlukan untuk mengirimkan sebuah pesan dikurangi banyaknya bit yang diperlukan untuk mengirimkan informasi di dalam pesan tersebut. Dengan kata lain, ia adalah banyaknya "pemborosan ruang" (pengulangan yang tidak perlu) dalam pengiriman sebuah data tertentu. Kompresi data adalah cara untuk menghilangkan redundancy yang tidak diinginkan, sementara checksum adalah cara untuk menambah redundancy yang diinginkan untuk mengecek adanya error.

$$r = \lim_{n\to\infty} \frac{1}{n} H(M_1, M_2, \dots M_n),$$

Untuk melihat redundancy pada *raw data*, ambil nilai rata-rata sumber informasi dari rata-rata entropi per simbol. Pada kasus umum *n* menuju tak hingga.

Nilai rata-rata absolut adalah algoritma kardinalitas dari ruang yang dipakai dalam pesan (alfabet) / Fungsi Hartley :

$$R = \log |\mathbb{M}|,$$

R menunjukkan nilai rata-rata maksimum informasi dapat disampaikan dengan alphabet tersebut.

Lalu, redundancy absolute adalah selisih dari rata-rata absolute dan rata-rata:

$$D = R - r,$$

$\frac{D}{R}$ adalah nilai redundancy relative yang memberikan rasio pengkompresian data maksimum. Rasio maksimum perbandingan besar data tidak terkompresi dan terkompresi adalah *R:r*

Komplemen dari konsep redundancy relatif adalah efisiensi.

$\frac{r}{R}$, maka $\frac{r}{R} + \frac{D}{R} = 1$.

Data yang memiliki efisiensi 100% tidak dapat dikompresi.

3.3. DCT

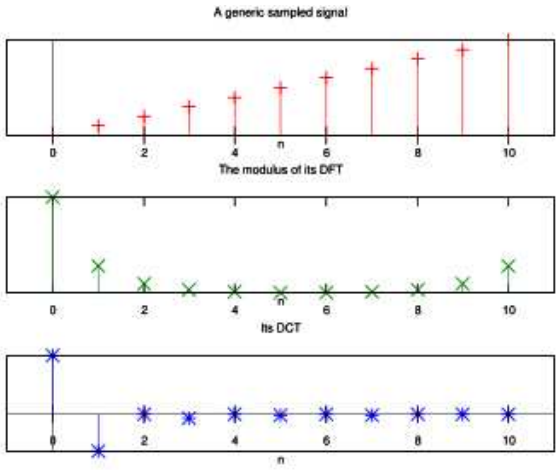
Rumus DCT-II :

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right]$$

$$k = 0, \dots, N-1.$$

DCT digunakan dalam kompresi gambar atau video JPEG, MJPEG, MPEG, DV. Dalam kompresi JPEG,

DCT digunakan untuk menyusutkan data Y, Cb, dan Cr menjadi bentuk yang lebih *compact.* Hal ini disebabkan DCT memiliki sifat menempatkan nilai-nilai datanya pada frekuensi-frekuensi rendah (Gbr 3.3-1) . JPEG menggunakan DCT-II dalam prosesnya.



Gambar 3.3-1

DCT-II yang bersifat dua dimensi NxN blok (JPEG membagi gambar menjadi blok-blok 8 x 8) dikomputasi dan hasilnya dikuantisasi dan dikode entropi. DCT-II diaplikasikan pada tiap baris dan kolom dari tiap blok. Hasilnya adalah koefisien transformasi 8 x 8 dimana elemen (0,0) (kiri-atas) adalah komponen DC (*zero frequency*) dan data lain pada indeks vertikal dan horizontal yang lebih besar merepresentasikan frekuensi spatial horizontal dan vertikal yang lebih tinggi.

Beberapa hal yang ditekankan dalam DCT-II: x_n genap di sekitar n=-1/2 dan ganjil di sekitar n=N-1/2; X_k genap di sekitar k=0 dan ganjil di sekitar k=N.

3.4 *Lossy* V.S. *Loss-less*

Kompresi data *lossy* adalah metode kompresi yang mengambil data lebih sedikit dari sumber aslinya. Penyampaian informasi masih dapat terjadi dengan baik karena sifat indra manusia yang terbatas, tidak dapat menerima semua spektrum dari informasi yang ada pada sumber. Oleh karena itu metode ini sering digunakan dalam penyimpanan file-file gambar, video dan suara. Data yang direpresentasikan dapat disusutkan hingga batas kepekaan manusia pada perbedaan frekuensi dan amplitudo dari data-data tersebut (suara dan cahaya adalah gelombang, dan yang disimpan dalam data digital adalah besar satuannya [*magnitudes*]). File audio dan gambar dapat dikompresi hingga 10:1, video dapat mencapai 300:1, tanpa kehilangan kualitas yang dapat dipersepsikan manusia. Penurunan kualitas yang tertangkap oleh indra manusia disebut *artefak kompresi*.

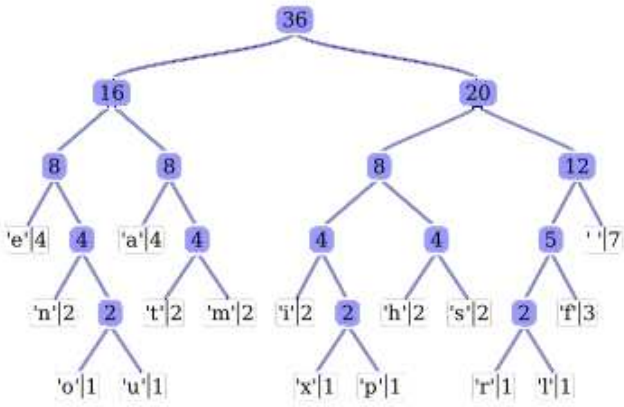
Kompresi *lossy* mengalami penurunan kualitas progresif setelah kompresi dan dekompresi berulang-ulang yang disebut *generation loss.*

Sebaliknya kompresi *loss-less* tidak membuang data. Hal ini berguna untuk menyimpan data yang tidak boleh mengalami *loss* seperti file teks dan data seperti catatan bank, tulisan, makalah, dan sebagainya.

4. ALGORITMA HUFFMAN

Dalam ilmu komputer dan teori informasi, kode Huffman adalah algoritma pengkodean entropi untuk kompresi data *lossless.* Istilah ini merujuk kepada penggunaan tabel kode yang memiliki panjang bervariasi (*variable length code*) dimana tabel kode tersebut diturunkan dengan cara tertentu berdasarkan prakiraan probabilitas kemunculan setiap nilai dalam sumber data. Kode Huffman dikembangkan oleh David A. Huffman pada saat mengambil gelar Ph.D. di MIT.

Kode Huffman menggunakan metode spesifik untuk merepresentasikan setiap symbol, menghasilkan prefix-free code (*string* dari bit representasi sebuah simbol tidak pernah menjadi prefix [awalan] dari sebuah simbol lain). Yang merepresentasikan karakter yang lebih sering muncul dengan *bit string* yang lebih pendek daripada karakter yang jarang muncul dalam suatu sumber data. Kompresi Huffman adalah metode paling efisien dari metode lain yang sejenis karena pemetaan lain simbol dari sumber data menjadi string unik menghasilkan file *output* yang lebih kecil ketika frekuensi symbol sesuai dengan frekuensi yang digunakan untuk menghasilkan kodenya. Kemudian ditemukan metode untuk melakukan pemetaan ini dalam waktu linear dengan mengurutkan probabilitas *input*.

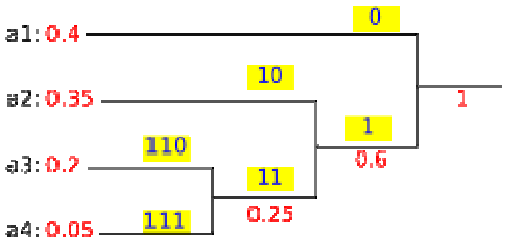


Gambar 4.a

Untuk suatu set simbol dengan distribusi probabilitas tersebar dan jumlah elemen sebesar kelipatan pangkat dua, kode Huffman setara dengan *binary block encoding* sederhana. Penggunaan kode Huffman begitu luas sampai-sampai semua kode *prefix free* disinonimkan dengan Huffman walaupun algoritmanya tidak sesuai dengan Huffman.

Walaupun kode Huffman optimal untuk pengkodean simbol demi simbol dengan probabilitas distribusi input yang diketahui, kadang-kadang keefesienannya dilebih-lebihkan. Sebagai contoh kompresi hasil kode aritmatika dan LZW sering memiliki efisiensi yang lebih baik dari Huffman pada kasus probabilitas input tidak diketahui.

4.1 Teknik Dasar



Gambar 4.1-1

Misalkan sebuah sumber menghasilkan empat simbol berbeda { a_1,a_2,a_3,a_4} dengan probabilitas {0.4;0.35;0.2;0.05}. Dengan membuat pohon biner dari kiri ke kanan (Gbr 4.1-1). Ambil dua buah symbol yang memiliki probabilitas paling rendah, dijumlahkan menjadi probabilitas dua simbol tadi, ulangi sampai hanya ada satu simbol di kanan. Baca pohon terbalik dari kanan ke kiri sambil memberikan kode biner berbeda pada tiap cabang.

Hasil akhir kode Huffman:

Symbol	**Code**
a1	0
a2	10

a3 110

a4 111

Cara standar untuk merepresentasikan sinyal dari 4 simbol adalah dengan menggunakan 2 bit/simbol, tetapi entropi dari sumber datanya 1.73 bit/symbol. Jika kode Huffman ini digunakan, entropi akan diturunkan ke 1.83 bit/symbol; masih jauh dari batas teoritis karena probabilitas dari simbol tidak sama dengan pangkat negatif dari dua.

Kode Huffman dikerjakan dengan membuat simpul pohon biner yang dapat disimpan dalam *array* biasa yang berukuran sesuai dengan banyaknya simbol, *n*. Sebuah simpul bisa berupa daun atau pohon. Pohon diinisialisasi dengan membuat semua simpul menjadi daun yang berisi simbol itu sendiri, frekuensi kemunculan, dan mungkin *pointer* ke *parent*nya untuk kemudahan kembali ke *address* jika membaca mundur. Simpul pohon memiliki dua simpul anak (Left(P), Right(P)) Mengikuti metode umum, Left diberi bit `0` dan Right diberi bit `1`. Pohon Huffman yang telah selesai memiliki *n* daun dan $n - 1$ simpul pohon.

Metode yang memakan waktu linear (asumsi setiap daun sudah diurutkan berdasarkan *weight* awal, dimana tanpa pengurutan ini, waktu yang diperlukan adalah $O(n \log n)$; *O adalah* notasi *big O*) untuk menyusun pohon Huffman adalah dengan menggunakan dua *queue* (antrian), yang pertama berisi *weight* awal dan yang kedua *weight* gabungan. Hal ini untuk memastikan *weight* terendah akan berada di posisi depan dari queue tersebut.

Membuat pohon :

Inisiasi daun sebanyak simbol.

Masukkan semua daun ke dalam *queue* dalam urutan probabilitas meningkat sehingga obyek yang memiliki kemunculan terendah ada pada *head queue*.

WHILE lebih dari satu simpul dalam *queue*

Keluarkan 2 simpul dengan *weight* terendah dari *queue.*

Buat sebuah subpohon dengan 2 simpul tadi (urutan *parent – child* tidak dipentingkan)

Masukkan simpul tersebut di belakang *queue* kedua

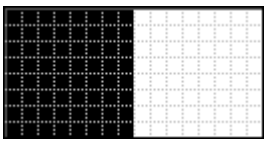
ENDWHILE

Simpul yang tersisa adalah akar. Pohon telah selesai. Secara umum, meminimalkan panjang codeword menguntungkan. Contoh, suatu *buffer* komunikasi harus memiliki daya simpan lebih besar untuk menangani simbol yang di-*encode* ke Huffman jika pohonnya sangat tidak seimbang.

5. APLIKASI DALAM JPEG

5.1. Sampel Gambar

Sebagai cara menjelaskan pengaplikasian kode Huffman ke dalam encoding ke JPEG, digunakan contoh gambar berikut:



Gambar 16x8 piksel, hitam dan putih. Perhatikan gambar ini adalah kelipatan dari blok JPEG yang dibagi dalam 8 x 8 piksel (Minimum Coded Unit [MCU]). Gambar ini tidak memiliki metadata dan tidak memakai optimalisasi, sehingga dalam konversinya ke format JPEG tidak menambah kompleksitas algoritmanya.

File JPEG mengandung maksimal 4 tabel Huffman dengan kode dengan panjang bervariasi dari 1 hingga 16 bit dan nilai kodenya 8 bit. Tabel Huffman yang dipakai dapat berasal dari standar JPEG atau program *image editor*nya sendiri yang mendefinisikan dengan DCT.

Gambar 16x8 piksel di atas memiliki *hex dump* sebagai berikut:

```
0270h: CA DA EA FA FF DA 00 0C 03 01 00 02 11 03 11 00
0280h: 3F 00 FC FF 00 E2 AF EF F3 15 7F FF D9
```

Start of Scan (SOS marker 0xFFDA berwarna kuning), bit tambahan (hijau) dan data isinya (biru tua), dan terminasi **End of Image** (EOI marker 0xFFD9 (biru muda). Panjang file ini 9 *byte.*

5.2. Perbandingan Besar File

Gambar asli (bitmap) (16 x 8 piksel) berukuran 128 piksel (2 MCU). Dengan 8 bit per channel (RGB), besar file adalah:

128 piksel x 8 *bit*/channel x 3 *channel* x 1 *byte*/8 *bit* = 384 *byte*.

File Format	Total Size	Overhead Size	Image Content Size
BMP (Uncompressed)	440 Bytes	56 Bytes	384 Bytes
JPEG	653 Bytes	644 Bytes	9 Bytes
JPEG (Optimized)	304 Bytes	297 Bytes	7 Bytes
GIF	60 Bytes	38 Bytes	22 Bytes

5.3. Dekode Scan Data

Scan data:

FC FF 00 E2 AF EF F3 15 7F

Keluarkan *padding bytes* (0xFF00 menjadi 0xFF):

FC FF E2 AF EF F3 15 7F

Kandungan gambar adalah 3 *channel* (Y, Cb, Cr) dengan masing-masing satu nilai DC dan 63 nilai AC.

Section	1	2	3	4	5	6
Component	Y		Cb		Cr	
AC / DC	DC	AC	DC	AC	DC	AC

Gambar 5.3-1 adalah hasil DCT dari 1 MCU, prosesnya tidak dibahas

Komponen DC memberikan nilai rata-rata (warna) dari setiap piksel dalam MCU (8x8). Kode pada isian DC (#0) mengindikasikan ukuran yang telah diencode dengan Huffman yang menunjukkan banyak bit yang diperlukan untuk merepresentasikan nilai rata-rata tiap MCU (misal. -511...+511).

Scan data di atas diubah ke biner:

1111 1100 1111 1111 1110 0010 1010 1111 1110 1111 1111 0011 0001 0101 0111 1111

20	-10	2	-2	-1	0	0	0
11	-9	-4	3	0	0	0	0
1	-8	2	0	0	0	0	0
-1	1	0	0	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

Gambar 5.3-1

Setelah DCT:

Konversi ke Domain Spatial (Tidak dibahas)

Konversi DCT ke RGB (Tidak dibahas)

Ekpansi DHT (Define Huffman Table) ke *binary bit strings*

Table 2 - Huffman - Luminance (Y) - AC

Length	Bits	Code
2 bits	00	01
	01	02
3 bits	100	03
4 bits	1010	11
	1011	04
	1100	00 (End of Block)
5 bits	1101 0	05
	1101 1	21
	1110 0	12
6 bits	1110 10	31
	1110 11	41
...	...	...
	...	...
12 bits	1111 1111 0011	F0 (ZRL)
	...	...
...	...	...
	...	...
16 bits	1111 1111 1111 1110	FA

Class = 1 (AC Table)
Destination ID = 0
Codes of length 01 bits (000 total):
Codes of length 02 bits (002 total): 01 02
Codes of length 03 bits (001 total): 03
Codes of length 04 bits (003 total): 11 04 00
Codes of length 05 bits (003 total): 05 21 12
Codes of length 06 bits (002 total): 31 41
Codes of length 07 bits (004 total): 51 06 13 61
Codes of length 08 bits (002 total): 22 71
Codes of length 09 bits (006 total):
81 14 32 91 A1 07
Codes of length 10 bits (007 total):
15 B1 42 23 C1 52 D1
Codes of length 11 bits (003 total): E1 33 16
Codes of length 12 bits (004 total): 62 F0 24 72
Codes of length 13 bits (002 total): 82 F1
Codes of length 14 bits (006 total):
25 43 34 53 92 A2
Codes of length 15 bits (002 total): B2 63
Codes of length 16 bits (115 total):
73 C2 35 44 27 93 A3 B3 36 17 54 64 74 C3 D2 E2
08 26 83 09 0A 18 19 84 94 45 46 A4 B4 56 D3 55
28 1A F2 E3 F3 C4 D4 E4 F4 65 75 85 95 A5 B5 C5
D5 E5 F5 66 76 86 96 A6 B6 C6 D6 E6 F6 37 47 57
67 77 87 97 A7 B7 C7 D7 E7 F7 38 48 58 68 78 88
98 A8 B8 C8 D8 E8 F8 29 39 49 59 69 79 89 99 A9
B9 C9 D9 E9 F9 2A 3A 4A 5A 6A 7A 8A 9A AA BA
CA DA EA FA

Jumlah kode: 162

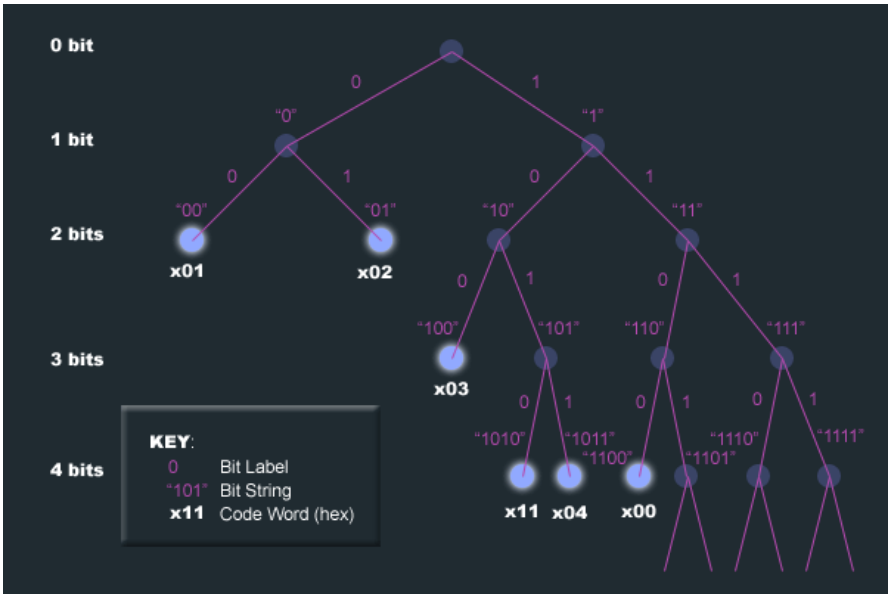
Membangun Pohon Huffman:

- **B aris 0,** beri 0 pada anak kiri dan 1 pada anak kanan.
- **Baris 1**, melihat DHT, tidak ada kode dengan panjang 1 bit. beri 0 pada anak kiri dan 1 pada anak kanan.
- **Baris 2**, ada 2 buah kode dengan panjang 2 bit. Tandai dengan x01 dan x02 pada 2 daun pertama dari kiri. 2 daun lagi diberi cabang.
- Baris **3**, ada 4 simpul tetapi hanya satu kode yang panjangnya 3. Beri x03 pada daun paling kiri dan beri cabang pada sisanya.
- Baris **4**, ada 6 simpul tetapi hanya ada 3 kode. Beri nilai pada 3 daun pertama (x11, x04, x00) dan beri cabang pada 3 lainnya

dan seterusnya. Ilustrasi pohon pada gambar 5.

Membaca biner dari pohon Huffman dengan mengambil jalur dari akar ke daun. `0` untuk anak kiri dan `1` untuk anak kanan:

Codes of length 02 bits:
00 = 01
01 = 02
Codes of length 03 bits:
100 = 03
Codes of length 04 bits:
1010 = 11
1011 = 04
1100 = 00 (EOB)
Codes of length 05 bits:
11010 = 05
11011 = 21
11100 = 12
Codes of length 06 bits:
111010 = 31
111011 = 41
Codes of length 07 bits:
1111000 = 51
...
Codes of length 15 bits:
111111111000100 = B2
111111111000101 = 63
Codes of length 16 bits:
1111111110001100 = 73
1111111110001101 = C2
...
1111111111111100 = DA
1111111111111101 = EA
1111111111111110 = FA



Waktu akses : 1 Januari 2008, 1:30

6. KESIMPULAN

Kesimpulan yang dapat diambil dari pembahasan ini:

1. Kode Huffman adalah algoritma yang penting untuk mengkompresi data karena kemampuannya untuk memperpendek panjang bit dalam representasi informasi dan menghilangkan redundancy.
2. Peran kode Huffman dalam kompresi gambar berformat JPEG adalah untuk penyusutan besar file lebih lanjut setelah *downsampling*, DCT, dan kuantisasi.
3. Dalam kompresi JPEG, Huffman adalah satu-satunya metode yang *loss-less* dalam rentetan proses *encoding*.

DAFTAR REFERENSI

[1] Munir, Rinaldi. (2006). *Diktat Kuliah IF2153 Matematika Diskrit*. Edisi 4. Program Studi Teknik Informatika, Institut Teknologi Bandung.

[2] *http://computer.howstuffworks.com/file-compression1.htm*
Waktu akses : 1 Januari 2008, 00:30

[3] *http://www.dspdesignline.com/howto/showArticle.jhtml?articleID=199904067*
Waktu akses : 1 Januari 2008, 00:30

[4] *http://en.wikipedia.org/wiki/Discrete_cosine_transform.htm*
Waktu akses : 1 Januari 2008, 1:30

[5] *http://en.wikipedia.org/wiki/Huffman_coding*
Waktu akses : 1 Januari 2008, 00:10

[6] *http://en.wikipedia.org/wiki/Information_entropy.htm*
Waktu akses : 29 Desember 2007, 23:00

[7] *http://en.wikipedia.org/wiki/JPEG.htm*
Waktu akses : 29 Desember 2007, 23:00

[8] *http://en.wikipedia.org/wiki/Lossy_compression.htm*
Waktu akses : 29 Desember 2007, 23:00

[9] *http://en.wikipedia.org/wiki/Quantization_(signal_processing).htm*
Waktu akses : 29 Desember 2007, 23:00

[10] *http://en.wikipedia.org/wiki/Redundancy_(information_theory).htm*
Waktu akses : 29 Desember 2007, 23:00

[11] *http://www.impulseadventure.com/*
Waktu akses : 29 Desember 2007, 23:30

[12] *http://www.jmg-galleries.com/articles/jpeg_compression.html*
Waktu akses : 1 Januari 2008, 1:30

[13] *http://navatrump.de/Technology/Datacompression/compression.ppt*