Aplikasi Kode *Huffman* pada Kompresi Bit *Frame* Video MPEG-4

Leonardus Brandon Luwianto (13519102)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13519102@std.stei.itb.ac.id

***Abstrak*—Makalah ini menjelaskan aplikasi kode Huffman pada proses kompresi bit sebuah *frame* video MPEG-4. Proses kompresi ini memanfaatkan nilai intensitas warna pada setiap piksel sebagai data dan letak piksel sebagai dimensi matriks. Kemudian data tersebut dikompresi menggunakan metode pohon *Huffman* untuk memperoleh kode-kode *Huffman* dari setiap simbol data *input.* Data simbol sebelum kompresi lalu digantikan dengan kode *Huffman-nya*. Adapun rasio adalah perbandingan antara data terkompresi dan data aslinya. Rasio akan dihitung untuk membandingkan jumlah bit sebelum dan setelah kompresi.**

***Kata Kunci—Huffman*, video, kompresi, MPEG-4**

I. PENDAHULUAN

Berkas video digital asli sebelum dikompresi memiliki ukuran yang cukup besar. Padahal pertukaran media informasi dewasa ini kian cepat dan video termasuk salah satu media digital untuk menyampaikan informasi. Oleh karena itu, kompresi video menjadi hal yang sangat krusial karena bisa menghemat kebutuhan penyimpanan data, mempersingkat waktu distribusi video, dan memperkecil kebutuhan *bandwidth*.

Ada berbagai macam metode kompresi, salah satunya algoritme *Huffman.* Kompresi *Huffman* merupakan kompresi *lossless* yang artinya kompresi pada setiap *frame* video tersebut tidak mengurangi kualitas dari video. Format berkas kompresi video yang cukup populer digunakan yaitu *Motion Picture Expert Group-4* (MPEG-4). Kompresi video dengan MPEG-4 menghasilkan video terkompresi yang ukurannya telah diperkecil dengan mengurangi beberapa bagian detail dari video.

II. DASAR TEORI

2.1 Video Digital

Video digital merupakan jenis berkas komputer yang digunakan untuk menyimpan kumpulan berkas digital seperti video, audio, metadata, informasi, pembagian *chapter*, dan judul sekaligus, yang dapat dimainkan atau digunakan melalui perangkat lunak tertentu pada komputer. Video digital terdiri dari *frame-frame* gambar yang ditampilkan ke layar dengan sangat cepat sehingga mendapatkan kesan gambar yang bergerak. Satuan elemen terkecil dari gambar digital disebut piksel (*pixel*). Ukuran piksel dibuat sangat kecil dan tiap satu piksel hanya memiliki satu komponen yaitu intensitas warna. Intensitas kedalaman warna dalam sebuah gambar atau video ditentukan oleh *bitmap*, yaitu matriks data yang mendeskipsikan karakteristik dari semua piksel yang menyusun sebuah gambar.

2.1.1 *Color Depth*

Intensitas kedalaman warna (*color depth*) memperlihatkan seberapa besar detail warna yang bisa ditampilkan oleh setiap piksel. Intensitas dari warna direpresentasikan oleh bit-bit. Semakin banyak jumlah bit yang dipergunakan makan semakin bervariasi intensitas warna yang bisa digunakan Terdapat tiga macam intensitas warna sesuai jumlah bit pada masing-masing piksel:

- 2-bit (Monokrom)
  Kedalaman warna monokrom merupakan konsep yang paling sederhana untuk merepresentasikan sebuah gambar yang setiap pikselnya mempunyai dua kemungkinan nilai, yaitu 0 dan 1.

- 4-bit (*16 Colors*)
  Kedalaman warna 4-bit merupakan pengembangan lebih lanjut dari monokrom. Jumlah warna yang direpresentasikan dalam 4-bit membuat setiap piksel memiliki 16 variasi intensitas campuran warna hitam dan putih pada pada tiap piksel.

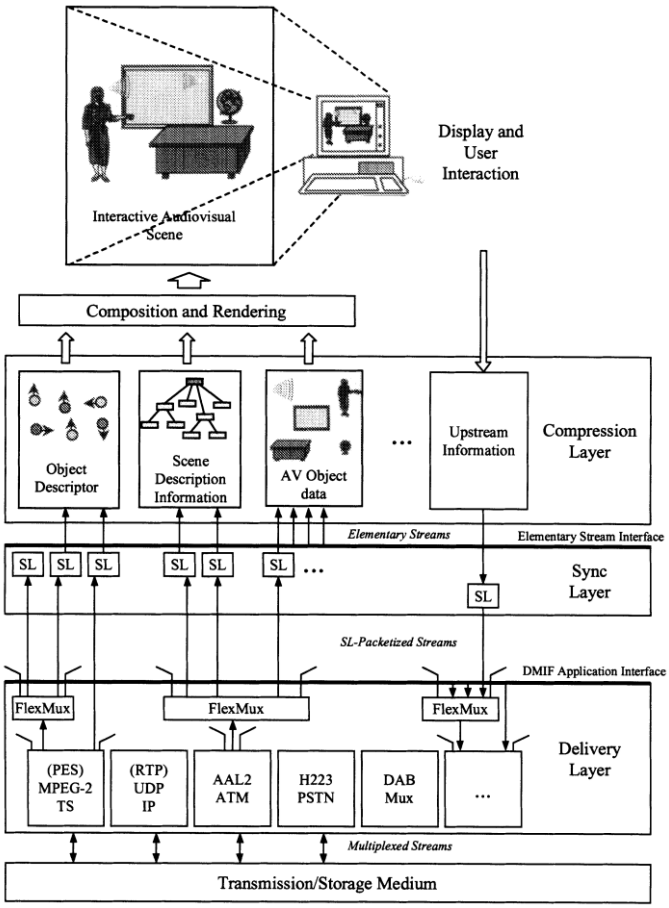
- 8-bit (*256 Colors*)
  Kedalaman warna 8-bit berarti tiap piksel memiliki 256 variasi dari intensitas warna karena setiap intensitas warna dikodekan dalam 8-bit yang dimulai dari 0000 0000 sampai 1111 1111.

2.2 MPEG-4

MPEG-4 adalah suatu metode kompresi data digital audio dan video. Format video MPEG-4 diperkenalkan pada November 1998 dan dan ditetapkan sebagai standar internasional pada Januari 1999. Format video merupakan pengembangan lebih lanjut dari MPEG-1 dan MPEG-2 dan dikembangkan oleh ISO/IEC JTC1/SC 29/WG 11 yang meliputi video bergerak dan *coding* audio.

Arsitektur sistem MPEG-4 tersusun dari tiga elemen utama yaitu *delivery layer, sync layer* (SL), dan *compression layer*,

seperti ditunjukkan pada gambar 2.2.1. *Delivery layer* terdiri dari aliran data (*streams) FlexMux* dan *TransMux*.



Gambar 2.1 Arsitektur sistem terminal MPEG-4.
Sumber: *Image and Video Compression for Multimedia Engineering 2nd Edition* (1999)

2.3 Kompresi Data

Kompresi data adalah proses mengkodekan informasi dalam jumlah bit yang lebih rendah daripada data sebelumnya yang belum terkodekan menggunakan suatu sistem *encoding* untuk merepresentasikan nilai informasi data digital dengan jumlah bit sesedikit mungkin tanpa menghilangkan nilai informasi di dalamnya. Alhasil, *bitrate* yang diperoleh lebih kecil.

Metode kompresi data dapat dikelompokkan berdasarkan berbagai aspek. Berdasarkan tipe peta kode yang digunakan untuk mengubah isi berkas input awal menjadi sekumpulan kode, metode kompresi dapat dibagi menjadi dua kategori:

2.3.1 Metode Statik

Metode statik menggunakan peta kode yang selalu sama. Metode ini membutuhkan dua fase (*two pass*), fase pertama untuk menghitung probabilitas kemunculan tiap simbol karakter dan menentukan peta kodenya. Fase kedua untuk mengubah pesan menjadi kumpulan kode yang akan ditransmisikan. Contoh dari metode ini adalah algoritme *Huffman* statik.

2.3.2 Metode Dinamik (Adaptif)

Metode dinamik enggunakan peta kode yang dapat berubah dari waktu ke waktu. Metode ini disebut adaptif karena peta kode mampu beradaptasi terhadap perubahan karakteristik isi file selama proses kompresi berlangsung. Metode ini bersifat *one pass* karena hanya diperlukan satu kali pembacaan terhadap isi file. Contoh dari metode ini adalah algoritme *Lempel Ziv Welch* (LZW).

Berdasarkan teknik pengkodean/pengubahan simbol yang digunakan, metode kompresi dapat dibagi menjadi tiga kategori:

2.3.3 Metode *Symbolwise*

Metode *symbolwise* menggunakan probabilitas kemunculan suatu karakter (simbol) dari suatu data input kemudian memberikan kode untuk setiap karakter sesuai dengan probabilitas kemunculannya. Simbol yang lebih sering muncul diberi Panjang kode lebih singkat. Contoh penggunaan metode ini yaitu pada algoritme *Huffman*.

2.3.4 Metode *Dictionary*

Metode ini mengambil karakter (simbol) dari suatu data untuk dikodekan kemudian hasil pengkodean tersebut dimasukkan ke dalam suatu kamus (*dictionary*) yang berisikan daftar kode untuk masing-masing karakter. Selanjutnya tiap-tiap karakter dikodekan sesuai dengan kode yang ada pada kamus tersebut sesuai indeks lokasinya.

2.3.5 Metode *Predictive*

Metode satu ini menggunakan model *finite-context* atau *finite-state* untuk memprediksi distribusi probabilitas dari simbol-simbol berikutnya.

Berdasarkan metode algoritme dan *output* yang dihasilkan, metode kompresi dapat dibagi menjadi dua kategori:

2.3.6 Kompresi *Lossy*

Kompresi *lossy* menghasilkan *output* kompresi yang mengorbankan akurasi, dalam hal ini kualitas video, demi meningkatkan tingkat kompresi. Namun, informasi seperti teks, gambar, video, dan audio tidak hilang, hanya terdapat beberapa distorsi kecil akibat proses kompresi *lossy*. Kompresi ini menghasilkan data yang sudah terkompresi dan tidak dapat dikembalikan lagi sama persis dengan data aslinya sehingga kompresi *loosy* bersifat *irreversible* (satu arah).

2.3.7 Kompresi *Lossless*

Kebalikan dari kompresi *lossy*, kompresi *lossless* mempertahankan data asli yang hendak dikompres seingga tidak akan kehilangan data apapun baik akurasi, informasi, maupun kualitasnya. Kompresi *lossless* terjadi akibat algoritme kompresi dan dekompresi balikan sama satu sama lain sehingga tidak ada bagian data yang hilang selama proses. Kompresi jenis ini menghasilkan data yang sudah terkompresi dan dapat dikembalikan ke bentuk awal sesuai dengan data aslinya sehingga bersifat *reversible* (dua arah). Selain *Huffman, dictionary coding, run-length coding,* dan *arithmetic coding* tergolong dalam kompresi *lossless.*

Adapun teknik kompresi data diklasifikan menjadi tiga jenis yakni:

No.	Teknik Kompresi	Sifat-Sifat
1.	*Entropy encoding*	• Bersifat *lossless* • Teknik kompresinya dilakukan berdasarkan urutan data, bukan berdasarkan media dengan spesifikasi dan karakteristik tertentu • *Statistical encoding*, artinya tidak memperhatikan semantik (makna) data • Contoh: *run-length coding, Huffman coding, arithemetic coding*
2.	*Source coding*	• Bersifat *lossy* • Berkaitan dengan semantik (makna) data • Contoh: *Prediction* (DPCM, DM); *Transformation* (FFT, DCT); *Layered Coding* (*bit position, subsampling, sub-band coding), Vector Quantization*
3.	*Hybrid coding*	• Bersifat gabungan antara *lossy* dan *lossless* • Contoh: JPEG, MPEG, H.264, DVI

Rasio Kompresi

Rasio kompresi didefinisikan sebagai perbandingan jumlah bit terkompresi dan jumlah bit asli, seperti yang dinyatakan dalam persamaan berikut:

$$Rasio\ kompresi = \frac{jumlah\ bit\ terkompresi}{jumlah\ bit\ asli} \quad \dots(1)$$

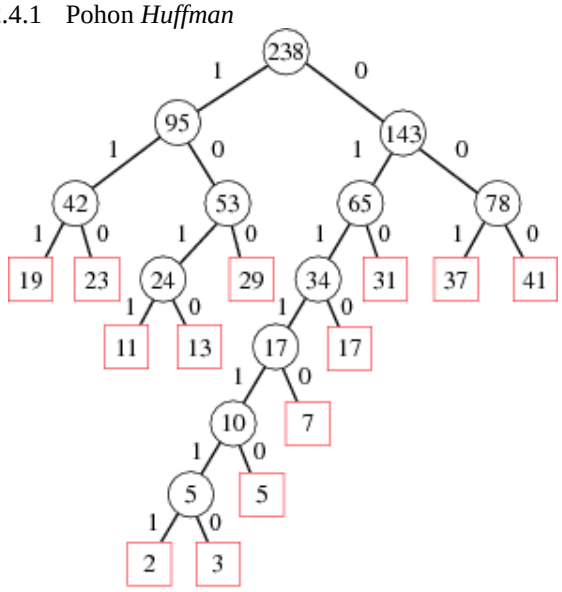
Semakin tinggi rasio kompresinya (mendekati 1) maka kualitas hasil kompresi berkurang lebih sedikit daripada semula dan teknik kompresinya lebih mangkus. Sebaliknya, semakin rendah rasio kompresinya maka kualitas hasil kompresi berkurang lebih banyak daripada semula dan teknik kompresinya lebih tidak mangkus.

2.4 Algoritme *Huffman*

Algoritme *Huffman* diciptakan oleh seorang mahasiswa MIT bernama David Huffman pada tahun 1952, merupakan salah satu metode paling lama dan paling terkenal dalam pemampatan teks. Algoritme *Huffman* menggunakan prinsip pengkodean yang mirip dengan kode *Morse*, yaitu tiap karakter (simbol) dikodekan hanya dengan rangkaian beberapa bit, dimana karakter yang sering muncul dikodekan dengan rangkaian bit yang pendek dan karakter yang jarang muncul dikodekan.dengan rangkaian bit yang lebih panjang. Berdasarkan tipe peta kode yang digunakan untuk mengubah pesan awal (isi data yang diinputkan) menjadi sekumpulan *codeword*, algoritme *Huffman* termasuk kedalam kelas algoritme yang menggunakan metode statik.

Pada awalnya David Huffman hanya men-*encoding* karakter dengan hanya menggunakan pohon biner biasa, namun setelah itu David Huffman menemukan bahwa penggunaan algoritme *greedy* dapat membentuk kode prefiks yang optimal. Penggunaan algoritme greedy pada algoritme *Huffman* adalah pada saat pemilihan dua pohon dengan frekuensi terkecil dalam membuat pohon *Huffman*. Algoritme greedy ini digunakan pada pembentukan pohon *Huffman* agar meminimumkan total *cost* yang dibutuhkan. *Cost* yang digunakan untuk menggabungkan dua buah pohon pada akar setara dengan jumlah frekuensi dua buah pohon yang digabungkan, oleh karena itu total *cost* pembentukan pohon *Huffman* adalah jumlah total seluruh penggabungan. penggabungan dua buah pohon dilakukan setiap langkah dan algoritme *Huffman* selalu memilih dua buah pohon yang mempunyai frekuensi terkecil untuk meminimumkan total *cost.*

2.4.1 Pohon *Huffman*



Gambar 2.2 Pohon *Huffman*
Sumber: https://mathworld.wolfram.com/HuffmanCoding.html

Menurut Forouzan (2013), untuk menggunakan *Huffman coding*, pertama kita perlu membuat pohon *Huffman*. Pohon *Huffman* adalah pohon yang daun pohonnya adalah simbol. Dilakukan dengan cara simbol yang paling sering muncul adalah yang paling dekat dengan akar pohon (dengan jumlah minimum *node* ke akar) dan simbol lebih sedikit muncul adalah yang terjauh dari akar.

Algoritme *encoding*:

1. Pilih dua simbol dengan peluang (*probability*) paling kecil. Kedua simbol tadi dikombinasikan sebagai simpul orang tua yaitu jumlah peluang kedua anaknya.
2. Selanjutnya, pilih dua simbol berikutnya, termasuk simbol baru, yang mempunyai peluang terkecil.
3. Ulangi langkah 1 dan 2 sampai seluruh simbol habis
4. Beri label secara konsisten sisi kiri dengan 0 dan sisi kanan dengan 1.
5. Lintasan dari akar ke daun berisi sisi sisi pohon yang deretan labelnya menyatakan kode Huffman untuk simbol daun tersebut.

Algoritme *decoding*:

1. Baca simbol biner pertama
2. Mulai traversal dari akar mengikuti sisi yang sesuai dengan simbol biner tersebut
3. Baca terus simbol biner dan traversal sisi yang bersesuaian sampai ketemu daun. Kodekan barisan biner yang sudah dibaca dengan simbol daun
4. Baca simbol biner berikutnya dan ulangi traversal dari akar. Ulangi Langkah 4 sampai semua simbol biner habis.

2.4.2 *Coding Table*

Menurut Forouzan (2013) setelah pohon telah dibuat, kita dapat membuat tabel yang menunjukkan bagaimana masing-masing karakter dapat di-*encode* dan *decode*. Kode untuk setiap karakter dapat ditemukan dengan memulai pada akar dan mengikuti cabang yang mengarah pada karakter itu. Kode itu sendiri adalah nilai bit masing-masing di jalan cabang, diambil secara berurutan. Tabel di bawah ini menunjukkan karakter kode untuk contoh sederhana berikut.

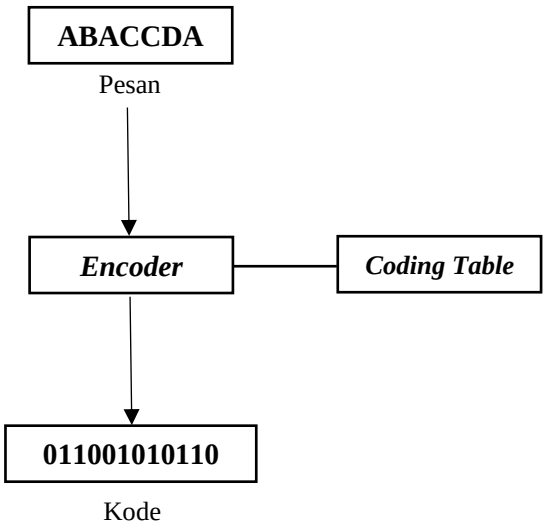
Simbol	Kode
A	00
B	010
C	011
D	10
E	11

Tabel 2.1 *Coding Table*
Sumber: *Data Comunications and Networking*, 5th Edition oleh Forouzan (2013)

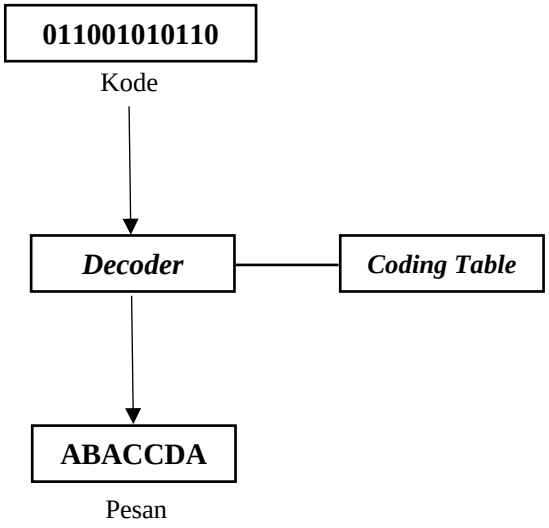
Perhatikan poin-poin tentang kode. pertama, karakter dengan frekuensi yang lebih tinggi menerima kode pendek (A, D dan E) daripada karakter dengan frekuensi yang lebih rendah (B dan C). Bandingkan ini dengan kode yang menetapkan panjang bit sama dengan masing-masing karakter. Kedua, dalam sistem coding, tidak ada kode yang menjadi awalan dari kode lain. Kode 2-bit, 00,10 dan 11, bukan prefiks dari setiap dua kode lain (010 dan 011). Dengan kata lain, kita tidak memiliki kode 3-bit awal dengan 00,10, atau 11. Nilai ini membuat kode *Huffman* menjadi *instantaneous code.*

2.4.3 *Encoding* dan *Decoding*

Dalam *Huffman coding*, tidak ada kode yang menjadi awalan dari kode lain. Ini berarti bahwa kita tidak perlu memasukkan pembatas untuk memisahkan kode untuk satu karakter dari kode yang berikutnya. *Huffman coding* juga memungkinkan seketika saat di-*decoding*, ketika decoder memiliki dua bit 00, segera dapat memecahkan kode itu sebagai karakter A, ia tidak perlu melihat lebih banyak bit. Salah satu kelemahan dari *Huffman coding* adalah bahwa kedua *encoder* dan *decoder* perlu menggunakan tabel pengkodean yang sama. Dengan kata lain, pohon *Huffman* tidak dapat dibuat secara dinamis seperti kamus di LZW *coding*, Namun, jika *encoder* dan *decoder* menggunakan set simbol yang sama sepanjang waktu, pohon dapat dibuat dan dibagi sekali. Jika tidak, tabel perlu dibuat oleh *encoder* dan diberikan kepada penerima.



Gambar 2.3 Skema *encoding* pada *Huffman coding*



Gambar 2.4 Skema *decoding* pada *Huffman coding*

III. ANALISIS

Analisis berikut membahas aplikasi kompresi *Huffman* pada suatu matriks *frame* video. Setiap piksel pada *frame* video menyimpan informasi tentang kedalaman warna dan model warna. Posisi koordinat piksel dan nilai intensitas warna dari setiap piksel yang dikumpulkan akan menghasilkan sebuah matriks yang berisi informasi posisi piksel dan intensitas warna pada piksel tersebut. Pada analisis ini, akan digunakan sebuah matriks gambar *grayscale* berukuran 4 × 4 *pixel*.

50	100	100	50
100	50	100	0
50	100	150	100
200	100	100	50

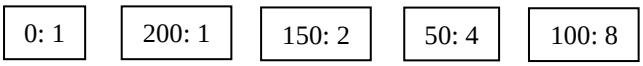
Gambar 3.1 *Matriks gambar grayscale berukuran 4 × 4 pixel*

Data *input* matriks di atas tersebut diubah ke dalam bentuk vektor matriks satu baris. Bentuk matriks pada gambar 3.1 yang telah diubah dalam bentuk vektor satu baris adalah sebagai berikut:

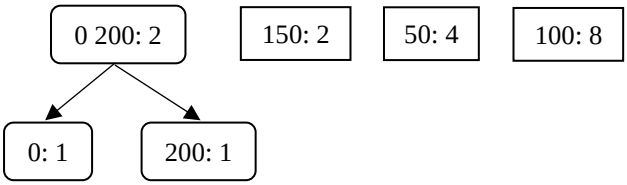
[50 100 100 50 100 50 100 0 50 100 150 100 200 100 100 50]

Setelah itu, disusun pohon *Huffman*-nya dengan algoritme *encoding* (lihat 2.4.1) seperti ditunjukkan pada proses di bawah ini:

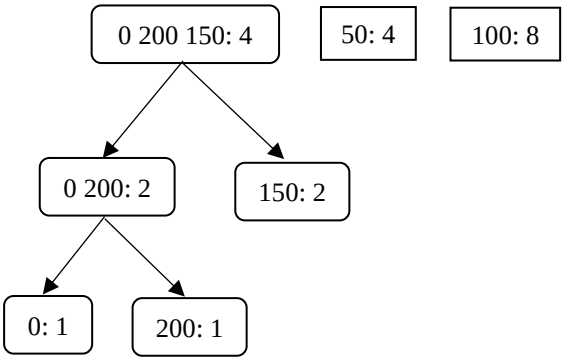
Langkah 1



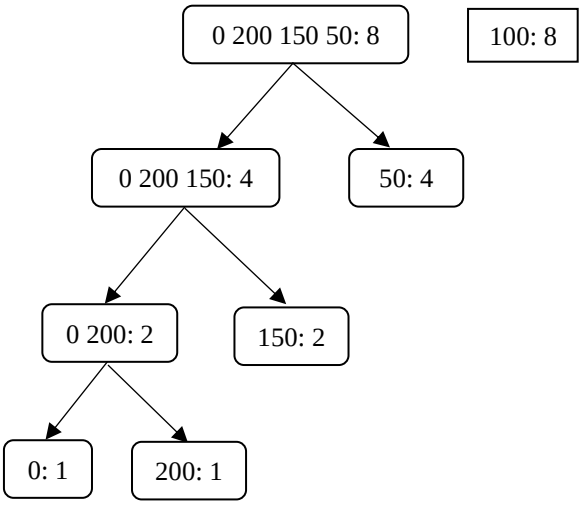
Langkah 2



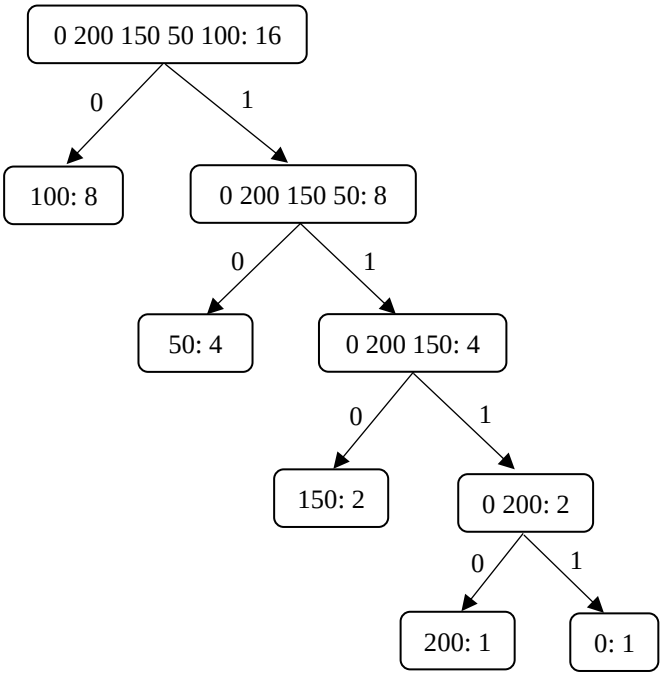
Langkah 3



Langkah 4



Langkah 5



Dari bentuk pohon *Huffman* terakhir, dapat dibuat tabel kekerapan (frekuensi) dan kode *Huffman* hasil *encoding* dari vektor matriks sebagai berikut:

Simbol	Kekerapan	Peluang	Kode *Huffman*
100	8	8/16	0
50	4	4/16	10
150	2	2/16	110
200	1	1/16	1110
0	1	1/16	1111

Tabel 3.1 *Tabel frekuensi dan kode Huffman encoding matriks*

Masing-masing simbol pada vektor dari matriks data *input* kemudian dikonversikan menjadi kode *Huffman*. Jumlah bit pada data hasil konversi *Huffman* dibandingkan dengan jumlah bit pada data *input*. Rasio dapat dihitung menggunakan persamaan (1).

Data *input*:
[50 100 100 50 100 50 100 0 50 100 150 100 200 100 100 50]

Jumlah elemen pada vektor *input* : 16
Kedalaman warna : 8-bit (256 warna)
Jumlah bit pada vektor *input* : 16 × 8 = 128-bit

Data *output*:
[10 0 0 10 0 10 0 1111 10 0 11 0 1110 0 0 10]

Jumlah bit pada vektor *output* : 29-bit
Rasio = jumlah bit terkompresi / jumlah bit asli
= 29 / 128
= 0,2265625

Persentase rasio = 0,2265625 × 100% = 22,65%

IV. KESIMPULAN

Data input yang dijadikan objek kompresi adalah sebuah matriks gambar *grayscale* dengan 4 × 4 *pixel* yang dalam bentuk vektor satu baris adalah [50 100 100 50 100 50 100 0 50 100 150 100 200 100 100 50]. Setiap data simbol dikompresi menggunakan algoritme *encoding* kompresi *Huffman*. Dari penurunan pohon *Huffman*, diperoleh data *output* [10 0 0 10 0 10 0 1111 10 0 11 0 1110 0 0 10].

Jumlah bit pada vektor input menggunakan kedalaman warna 8-bit (256 warna) adalah 128-bit sedangkan jumlah bit pada vektor *output*-nya 29-bit. Dari perolehan bit tersebut, rasio jumlah bit terkompresi dan jumlah bit aslinya adalah sebesar 0.2265625 yang dalam persentase 22,65%.

V. UCAPAN TERIMA KASIH

Puji syukur kepada Tuhan Yang Maha Esa atas berkat dan rahmatNya sehingga makalah ini dapat diselesaikan dengan baik dan tepat waktu dalam rangka memenuhi tugas mata kuliah IF2120 – Matematika Diskret. Penulis mengucapkan terima kasih kepada Ibu Dra. Harlili S., M.Sc. sebagai dosen pembimbing kelas mata kuliah Matematika Diskret dan para pihak yang telah menghasilkan karya-karya yang berguna untuk menyelesaikan makalah ini. Penulis berharap agar makalah ini sekiranya dapat berguna bagi pembaca dan masyarakat.

REFERENSI

[1] Binanto, Iwan (2008). *Multimedia Digital Dasar Teori + Pengembangannya*. ANDI.
[2] Chandra, Ian. 2003. *Utiliti Audio/Video*. PT. Elex Media Komputindo, Jakarta.
[3] Forouzan, B.A. 2013. *Data Comunications and Networking*, 5th Edition. New York, NY: The McGraw-Hill Companies,Inc.
[4] Ilham, R. Alifiansyah dan Arman Diponegoro. 2013. *Kompresi Bit dengan Huffman Code pada Frame Video MPEG-4*. Fakultas Teknik Universitas Indonesia.
[5] Lubis, Nuzul S. 2013. *Analisis Perbandingan Kompresi File Video dengan Motion Picture Expert Group-4 dan Flash Video dengan menggunakan Algoritma Huffman*. Medan. Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara.
[6] Pn, Mokhamad dkk. 2008. *Analisis dan Implementasi Perbandingan Kinerja Algoritma Kompresi Huffman, LZW, dan DMC pada Berbagai Tipe File*. Fakultas Teknik Informatika Universitas Telkom.
[7] Shi, Yun dan Huifang Sun. 1999. *Image and Video Compression for Multimedia Engineering 2nd Edition*. CRC Press.
[8] https://mathworld.wolfram.com/HuffmanCoding.html (diakses 9 Desember 2020)
[9] http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag2.pdf (diakses 9 Desember 2020)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Surakarta, 11 Desember 2020

Tanda Tangan

Leonardus Brandon Luwianto (13519102)