BAB II

LANDASAN TEORI

2.1 Format Berkas Bitmap

Citra disimpan di dalam berkas dengan format tertentu. Format citra yang baku di lingkungan sistem operasi *Microsoft Windows* dan IBM OS/2 adalah berkas bitmap (BMP). Saat ini format BMP memang "kalah" populer dibandingkan format JPG atau GIF. Hal ini karena berkas BMP pada umumnya tidak di kompresi, sehingga ukuran berkasnya relatif lebih besar dari pada berkas JPG maupun GIF. Hal ini juga yang menyebabkan format BMP sudah jarang digunakan.

Meskipun format BMP tidak mangkus dari segi ukuran berkas, namun format BMP mempunyai kelebihan dari segi kualitas gambar. Citra dalam format BMP lebih bagus dari pada citra dalam format lainnya karena citra dalam format BMP umumnya tidak dikompresi sehingga tidak ada informasi yang hilang. Terjemahan bebas bitmap adalah berkas bit. Artinya nilai intensitas *pixel* di dalam citra dipetakan ke sejumlah bit tertentu. Peta bit yang umum adalah 8, artinya setiap *pixel* panjangnya 8 bit. Delapan bit ini merepresentasikan nilai intensitas *pixel*. Dengan demikian ada sebanyak $2^8 = 256$ derajat keabuan, mulai dari 0 sampai 255.

Citra dalam format BMP ada tiga macam: citra biner, citra berwarna dan citra hitam-putih. Citra biner hanya mempunyai dua nilai keabuan, 0 dan 1. Oleh karena itu, 1 bit sudah cukup untuk merepresentasikan nilai *pixel*. Citra berwarna

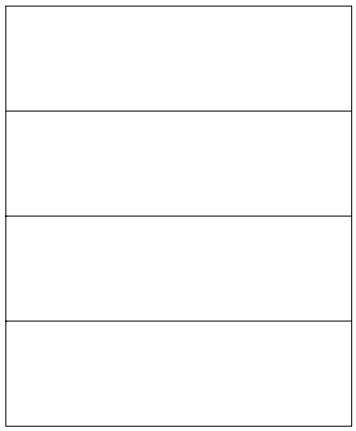
adalah citra yang lebih umum. Warna yang terlihat pada citra bitmap merupakan kombinasi dari tiga warna dasar, yaitu merah, hijau dan biru. Setiap *pixel* disusun oleh tiga komponen warna, yaitu R (*red*), G (*green*) dan B (*blue*). Kombinasi dari ketiga warna tersebut menghasilkan warna yang khas untuk *pixel* yang bersangkutan. Pada citra 256 warna, setiap *pixel* panjangnya 8 bit, tetapi komponen warna RGB-nya disimpan di dalam tabel RGB yang disebut palet. Setiap komponen panjangnya 8 bit, jadi ada 256 nilai keabuan untuk warna merah, 256 untuk warna hijau dan 256 untuk warna nilai keabuan untuk warna biru. Nilai setiap *pixel* tidak menyatakan derajat keabuannya secara langsung, tetapi nilai *pixel* menyatakan indeks tabel RGB yang memuat nilai keabuan merah (R), nilai keabuan hijau (G), dan nilai keabuan biru (B) untuk *pixel* yang bersangkutan. Pada citra hitam-putih, nilai R = G = B untuk menyatakan bahwa citra hitam-putih hanya mempunyai satu kanal warna. Citra hitam-putih umumnya adalah citra 8-bit.

Citra yang lebih kaya warna adalah citra 24-bit. Setiap *pixel* panjangnya 24-bit, karena setiap *pixel* langsung menyatakan komponen warna merah, komponen warna hijau dan komponen warna biru. Masing-masing komponen panjang 8-bit. Citra 24-bit disebut juga citra 16 juta warna, karena ia mampu menghasilkan 2^{24} = 16.777.216 kombinasi warna.

Saat ini beredar tiga versi berkas bitmap, yaitu berkas bitmap versi lama dari *Microsoft Windows* atau IBM OS/2, berkas bitmap versi baru dari *Microsoft Windows* dan yang ketiga adalah berkas bitmap versi baru dari IBM OS/2. Yang membedakan dari ketiga versi diatas adalah panjang *header*-nya. *Header* adalah

data yang terdapat pada bagian awal berkas citra. Data dalam *header* berguna untuk mengetahui bagaimana citra dalam format bitmap dikodekan dan disimpan. Data di dalam *header* misalnya pajang citra, lebar citra, kedalaman *pixel*, dan sebagainya.

Setiap berkas bitmap terdiri atas *header* berkas, *header* bitmap, informasi palet, dan data bitmap (Gambar 2.1).



Gambar 2.1 Struktur berkas citra bitmap

Ukuran *header* berkas sama untuk semua versi, yaitu 14 *byte*. Tabel 2.1 memperlihatkan *filed-filed* penyusunan header berkas.

Tabel 2.1 *Header* berkas bitmap (Panjang = 14 *byte*)

Byte ke-	Panjang (*byte*)	Nama	Keterangan
1-2	2	*Bmp Type*	Tipe berkas bitmap: BA=bitmap *array*, CI = *icon* BM= bitmap, CP= *color pointer* PT= *pointer*
3-6	4	*Bmp Size*	Ukuran berkas bitmap
7-8	2	*XhotSpot*	X *hotspot* untuk kursor
9-10	2	*YhotSpot*	Y *hotspot* untuk kursor
11-14	4	*OffBits*	Ofset ke awal data bitmap (dalam *byte*)

Ukuran *header* bitmap berbeda-beda untuk setiap versi. Untuk berkas bitmap versi lama, *header* bitmap berukuran 12 byte (Tabel 3.2), untuk berkas versi baru dari *Microsoft Windows*, *header* bitmap berukuran 40 (Tabel 3.3), dan untuk berkas bitmap versi baru dari IBM OS/2, *header* bitmap berukuran 64 *byte* (Tabel 3.4)

Tabel 2.2 *Header* bitmap versi lama dari *Microsoft Windows* (12 *byte*)

Byte ke-	Panjang (*byte*)	Nama	Keterangan
1-4	4	*HdrSize*	Ukuran *header* dalam ukuran *byte*
5-6	2	*Width*	Lebar bitmap dalam ukuran *pixel*
7-8	2	*Height*	Tinggi bitmap dalam ukuran *pixel*
9-10	2	*Planes*	Jumlah *plane* (umum-nya selalu satu)
11-12	2	*BitCount*	Jumlah bit per *pixel*

Tabel 2.3 *Header* bitmap versi baru dari IBM OS/2 (64 *byte*)

Byte ke-	Panjang (*byte*)	Nama	Keterangan
1-4	4	*HdrSize*	Ukuran *header* dalam satuan *byte*
5-8	4	*Width*	Lebar bitmap dalam satuan *pixel*
9-12	4	*Height*	Tinggi bitmap dalam satuan *pixel*
13-14	2	*Planes*	Jumlah plane (umum-nya selalu satu)
15-16	2	*BitCount*	Jumlah bit per *pixel*
17-20	4	*Compression*	0=tak dikompresi, 1=dikompresi
21-24	4	*ImgSize*	Ukuran bitmap dalam *byte*
25-28	4	*HorzRes*	Resolusi *horizontal*
29-32	4	*VertRes*	Resolusi *vertical*
33-36	4	*ClrUsed*	Jumlah warna yang digunakan
37-40	4	*ClrImportant*	Jumlah warna yang penting
41-42	2	*Units*	Satuan pengukuran yang dipakai
43-44	2	*Reserved*	*Filed* Cadangan
45-46	2	*Recording*	Algoritma perekaman
47-48	2	*Rendering*	Algoritma *halftoning*
49-52	4	*Size1*	Nilai ukur 1
53-56	4	*Size2*	Nilai ukur 2
57-60	4	*ClrEncoding*	Pengkodean warna
61-64	4	*Identifier*	Kode yang digunakan aplikasi

Tabel 2.4 *Header* bitmap versi baru dari *Microsoft Windows* (40 *byte*)

Byte ke-	Panjang (*byte*)	Nama	Keterangan
1-4	4	*HdrSize*	Ukuran *header* dalam satuan *byte*
5-8	4	*Width*	Lebar bitmap dalam satuan pixel
9-12	4	*Height*	Tinggi bitmap dalam satuan *pixel*
13-14	2	*Planes*	Jumlah *plane* (umum-nya selalu satu)
15-16	2	*BitCount*	Jumlah bit per *pixel*
17-20	4	*Compression*	0=tak dikompresi, 1=dikompresi
21-24	4	*ImgSize*	Ukuran bitmap dalam *byte*
25-28	4	*HorzRes*	Resolusi *horizontal*
29-32	4	*VertRes*	Resolusi *vertical*
33-36	4	*ClrUsd*	Jumlah warna yang digunakan
37-40	4	*ClrImportant*	Jumlah warna yang penting

Informasi palet warna terletak sesudah *header* bitmap, Informasi palet warna dinyatakan dalam suatu tabel RGB. Setiap *entry* pada tabel terdiri atas tiga buah *filed*, yaitu R (*red*), G (*green*), dan B (*blue*).

Tabel 2.5 Panjang informasi palet untuk setiap versi berkas bitmap

Citra *m* bit	Versi lama (*Windows*)	Versi baru (*Windows*)	Versi baru (OS/2)
Citra 4-bit	48 *byte*	64 *byte*	64 *byte*
Citra 8-bit	768 *byte*	1024 *byte*	1024 *byte*
Citra 24-bit	0 *byte*	0 *byte*	0 *byte*

Data bitmap diletakan sesudah informasi palet. Penyimpanan data bitmap di dalam berkas disusun terbalik dari bawah keatas dalam bentuk matriks yang berukuran *Height* x *Width*. Baris ke-0 pada matriks data bitmap menyatakan data *pixel* di citra baris terbawah, sedangkan baris terakhir pada matriks menyatakan data *pixel* di citra baris teratas.

2.2 Kompresi Data

Kompresi data (pemampatan data) merupakan suatu teknik untuk memperkecil jumlah ukuran data (hasil kompresi) dari data aslinya. Kompresi data umumnya diterapkan pada mesin komputer, hal ini dilakukan karena setiap simbol yang dimunculkan pada komputer memiliki nilai bit-bit yang berbeda. Misal pada ASCII setiap simbol yang dimunculkan memiliki panjang 8 bit, misal kode A pada ASCII mempunyai nilai decimal 65, jika dirubah dalam bilangan biner menjadi 010000001. Kompresi data digunakan untuk mengurangkan jumlah bit-bit yang dihasilkan dari setiap simbol yang muncul. Dengan kompresi ini diharapkan dapat mengurangi/memperkecil ukuran data dalam ruang penyimpanan.

2.2.1 *Encoding dan Decoding*

Encoding merupakan teknik untuk mendapatkan kode-kode tertentu. Dari kode-kode tersebut dapat diaplikasikan untuk kompresi data dan keamanan data. Dari data-data yang telah dikodekan tersebut, format-format isi dari data tersebut berbentuk kode-kode yang tidak bisa dibaca oleh user. Agar kode-kode tersebut bisa dibaca oleh user, maka kita perlu mengkodekan ulang data tersebut. Hal ini biasa dikenal dengan nama *decoding*. Proses *decoding,* yaitu proses pengembalian kode-kode yang telah dibuat menjadi simbol-simbol yang dikenal oleh user.

2.2.2 Faktor Penting Dalam Kompresi Data

Dalam kompresi data, terdapat empat faktor penting yang perlu diperhatikan,

yaitu :

1. *Time process*

Yaitu waktu yang dibutuhkan untuk menjalankan proses kompresi.

2. *Completeness*

Yaitu kelengkapan data setelah data tersebut dikompresi.

3. *Ratio Compress*

Yaitu ukuran data setelah dikompresi

4. *Optimaly*

Yaitu perbandingan ukuran data sebelum dikompresi dengan data yang telah dikompresi.

2.2.3 Jenis Kompresi Data

Berdasarkan mode penerimaan data oleh manusia, kompresi data dapat dibagi menjadi dua, yaitu :

a. *Dialoque Mode*

Yaitu proses penerimaan data dimana pengirim dan penerima seakan berdialog (*real time*), dimana kompresi data harus berada dalam batas penglihatan dan pendengaran mausia. Contohnya pada *video conference*.

b. *Retrieval Mode*

Yaitu proses penerimaan data tidak dilakukan secara *real time*. Jenis kompresi ini dapat dilakukan *random access* terhadap data dan dapat bersifat interaktif.

2.2.4 Teknik Kompresi Data

Teknik kompresi data dapat digolongkan menjadi dua kelompok utama yaitu : *lossy* dan *lossless*. Teknik kompresi secara *lossy* dimaksudkan dengan menghilangkan ketelitian data utama guna mendapatkan data sekecil mungkin (kompresi data sebesar mungkin). Teknik kompresi data secara *lossless* yaitu teknik kompresi data dengan mengurangkan jumlah data yang terjadi redudansi (memiliki simbol yang sama) sebelum terjadi kompresi.

a. *Lossy Compression*

Merupakan teknik kompresi yang menghilangkan beberapa informasi data yang dianggap tidak penting. Sehingga hasil data yang telah terkompresi tidak sama dengan data yang sebelum dikompresi. Namun data yang telah terkompresi tersebut sudah cukup untuk digunakan walaupun datanya telah berubah. Keunggulan dari teknik ini adalah file atau data yang telah terkompresi memiliki ukuran yang lebih kecil dari ukuran data aslinya. Biasanya teknik kompresi ini banyak diaplikasikan pada data gambar audio dan video.

b. *Lossless Compression*

Merupakan teknik kompresi yang mempertahankan kebutuhan informasi yang dikandung oleh data, sehingga informasi yang terkandung pada file yang telah terkompresi tetap terjaga meskipun ukurannya telah berubah dari ukuran data aslinya. Keunggulan dari teknik ini adalah data yang telah terkompresi, apabila didekompresi kembali akan menghasilkan data yang sama persis dengan data aslinya. Biasanya teknik kompresi ini banyak diaplikasikan pada data teks.

2.2.5 Klasifikasi Teknik Kompresi Data

Teknik kompresi dapat diklasifikasikan menjadi tiga kelompok, antara lain :

a. *Entropy Encoding*

Merupakan klasifikasi teknik kompresi yang bersifat *loseless compression.* Tekniknya tidak berdasarkan media dengan spesifikasi dan karakteristik tertentu, namun berdasarkan urutan data. Dan *statistical encoding* pada klasifikasi teknik ini tidak memperhatikan semantik data. Contoh kelompok teknik kompresi ini adalah pada *Run-Length* coding, *Huffman* coding, *Arithmetic* coding.

b. *Source Coding*

Merupakan klasifikasi teknik kompresi yang bersifat *lossy compression.* Tekniknya berkaitan dengan data semantik (arti data) dan media. Contoh kelompok teknik kompresi ini adalah pada *prediction (DPCM, DM), transformation (FFT, DCT), layered coding (bit position, subsampling, sub-band coding), vector quantization.*

c. *Hybrid Coding*

Merupakan klasifikasi teknik kompresi yang bersifat gabungan antara *lossy compression* dan *loseless compression.* Contoh kelompok teknik kompresi ini adalah pada JPEG, MPEG, H.261, DVI.

2.2.6 Metode Kompresi Data

Berdasarkan tipe peta kode yang digunakan untuk mengubah pesan awal (isi file input) menjadi sekumpulan *codeword*, metode kompresi terbagi menjadi dua kelompok, yaitu :

a. 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 dan fase kedua untuk mengubah pesan menjadi kumpulan kode yang akan ditransmisikan. Contohnya pada *Huffman static, arithmetic coding.*

b. Metode Dinamik (adaptif)

Menggunakan peta kode yang dapat diubah dari waktu ke waktu. Metode ini disebut adaptif karena peta kode mampu beradaptasi terhadap karakteristik isi file selama proses kompresi berlangsung. Metode ini bersifat *onepass*, karena isi file selama dikompres hanya diperlakukan satu kali pembacaan terhadap isi file. Contohnya pada algoritma LZW dan DMC.

Berdasarkan teknik pengkodean atau pengubahan simbol yang digunakan, metode kompresi dapat dibagi ke dalam tiga kategori, yaitu :

a. Metode *simbolwise*

Menghitung peluang kemunculan dari tiap simbol dalam file input, lalu mengkodekan satu simbol dalam satu waktu, dimana simbol yang lebih sering muncul diberi kode lebih pendek dibandinglan simbol yang lebih jarang muncul. Contohnya pada Huffman coding, arithmetic coding.

b. Metode *dictionary*

Menggantikan karakter/fragmen dalam file input dengan indeks lokasi dari karakter/fragmen tersebut dalam sebuah kamus (*dictionary*). Contohnya pada

algoritma LZW.

c. Metode *predictive*

Menggunakan model *finite-context* atau *finite-state* untuk memprediksi distribusi probabilitas dari simbol-simbol selanjutnya. Contohnya pada algoritma DMC.

2.2.7 Algoritma Huffman

Algoritma Huffman, yang dibuat oleh seorang mahasiswa MIT bernama David Huffman pada tahun 1952, merupakan salah satu metode paling lama dan paling terkenal dalam kompresi teks. Algoritma 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*, algoritma Huffman termasuk kedalam kelas algoritma yang menggunakan metode *static*. Metoda *static* adalah metoda yang selalu menggunakan peta kode yang sama, metoda ini membutuhkan dua fase (*two-pass*). Fase pertama untuk menghitung probabilitas kemunculan tiap simbol dan menentukan peta kodenya, dan fase kedua untuk mengubah pesan menjadi kumpulan kode yang akan di taransmisikan.

Sedangkan berdasarkan teknik pengkodean simbol yang digunakan, algoritma Huffman menggunakan metode *symbolwise*. Metode *symbolwise* adalah metode

yang menghitung peluang kemunculan dari setiap simbol dalam satu waktu, dimana simbol yang lebih sering muncul diberi kode lebih pendek dibandingkan simbol yang jarang muncul.

2.2.7.1 Proses Pembentukan Huffman Tree

Kode Huffman pada dasarnya merupakan kode prefiks (*prefix code*). Kode prefiks adalah himpunan yang berisi sekumpulan kode biner, dimana pada kode prefik ini tidak ada kode biner yang menjadi awal bagi kode biner yang lain. Kode prefix biasanya direpresentasikan sebagai pohon biner yang diberikan nilai atau label. Untuk cabang kiri pada pohon biner diberi label 0, sedangkan pada cabang kanan pada pohon biner diberi label 1. Rangkaian bit yang terbentuk pada setiap lintasan dari akar ke daun merupakan kode prefiks untuk karakter yang berpadanan. Pohon biner ini biasa disebut *Huffman Tree*. *Huffman Tree* ini dibentuk dengan cara menghitung terlebih dahulu frekuensi kemunculan karakter pada string. Kemudian mencari dua buah karakter yang memiliki nilai frekuensi terkecil dan gabungkan kedua karakter tersebut pada sebuah akar. Kemudian cari kembali karakter yang memiliki frekuensi terkecil, dan ulangi langkah tersebut sampai terbentuk sebuah pohon yang daun-daunnya berisi seluruh karakter pada string.

2.2.7.2 Proses Encoding

Encoding adalah cara menyusun *string* biner dari teks yang ada. Proses *encoding* untuk satu karakter dimulai dengan membuat *Huffman Tree* terlebih

dahulu. Setelah itu, kode untuk satu karakter dibuat dengan menyusun nama *string* biner yang dibaca dari akar sampai ke daun *Huffman Tree*. Hasil dari penyusunan nama string biner tersebut menjadi kode biner yang baru untuk setiap karakter. Karakter yang memiliki frekuensi kemunculan yang besar akan memiliki kode biner yang lebih pendek dari pada karakter yang memiliki frekuensi kemunculan yang kecil.

2.2.7.3 Proses Decoding

Decoding merupakan kebalikan dari *encoding*. *Decoding* berarti menyusun kembali data dari *string* biner menjadi sebuah karakter kembali. *Decoding* dapat dilakukan dengan dua cara, yang pertama dengan menggunakan *Huffman Tree* dan yang kedua dengan menggunakan tabel kode Huffman. Melakukan proses decoding dengan menggunakan *Huffman Tree* dilakukan dengan cara menelusuri *Huffman Tree* dan mencatat string biner yang ada pada cabang sampai dengan karakter yang dicari ditemukan. Sedangkan menggunakan tabel kode Huffman adalah dengan cara membuat sabuah table yang berisi karakter yang diencoding beserta string biner yang baru dibentuk dari *Huffman Tree.*

2.2.8 Algoritma LZ77

Algoritma LZ77 (Lempel Ziv 1977) merupakan algoritma kompresi yang bersifat *Loseless.* Algoritma ini dikembangkan oleh Abraham Lempel dan Jacob Ziv pada tahun 1977. Hal ini sesuai dengan nama yang diberikan pada algoritma kompresi tersebut yang merupakan singkatan dari nama penemunya. Algoritma

LZ77 ini dalam melakukan kompresinya menggunakan teknik pengkamusan (*Dictionary Techniques)*. Algoritma LZ77 ini menggunakan dua buah buffer, yaitu *Dictionary Buffer* dan *Lookahead Buffer*. *Dictionary buffer* berfungsi untuk menyimpan symbol yang baru saja dikodekan dari inputan. Sedangkan *Lookahead Buffer* berfungsi untuk menyimpan symbol yang akan dikodekan. Hasil atau output dari proses encoding pada algoritma LZ77 ini terdiri dari tiga unsur atau tiga bit. Bit yang pertama berisi nilai dari posisi yang *match* atau cocok antara *dictionary buffer* dan *lookahead buffer*. Bit yang kedua berisi nilai dari panjang karakter yang *match* atau cocok antara *dictionary buffer* dan *lookahead buffer*. Bit yang ketiga berisi nilai dari karakter yang tidak *match* atau tidak cocok antara *dictionary buffer* dan *lookahead buffer.*

2.2.8.1 Proses Encoding

Proses encoding pada algoritma LZ77 ini dengan cara mencari nilai P,L dan S. Nilai P adalah posisi karakter yang sama pada *lookahead buffer* dengan *dictionary buffer*, nilai L adalah jumlah karakter yang sama pada *lookahead buffer* dengan *dictionary buffer* dan nilai S adalah karakter yang akan disimpan pada *dictionary buffer*. Output dari proses encoding pada algoritma LZ77 ini adalah berupa string gabungan dari nilai-nilai P,L dan S sampai dengan tidak ada lagi karakter pada *lookahead buffer*.

2.2.8.2 Proses Decoding

Proses decoding pada algoritma LZ77 ini dilakukan dengan cara mengambil

3 buah karakter dan kemudian digabungkan menjadi sebuah kelompok karakter. Ssetiap kelompok karakter yang telah dibuat berisikan nilai P, L dan S. Nilai P adalah karakter pertama pada kelompok karakter, nilai L adalah karakter kedua pada kelompok karakter dan nilai S adalah karakter ketiga pada kelompok karakter. String diambil dengan cara mencari posisi karakter pada *dictionary buffer* sesuai dengan nilai P, lalu mengambil banyaknya karakter yang akan diambil pada *dictionary buffer* sesuai dengan nilai S dan gabungkan karakter yang telah diambil dengan nilai S. Hasil dari proses decoding ini adalah string gabungan dari karakter-karakter yang telah diambil dari *dictionary buffer.*

2.2.9 Algoritma Deflate

Algoritma deflate merupakan algoritma persilangan antara algoritma Huffman dan algoritma LZ77. Dalam proses kompresinya, algoritma deflate ini terlebih dahulu melakukan proses pengelompokan karakter dengan menggunakan algoritma LZ77. Kemudian hasil dari pengelompokan karakter tersebut dikompresi lagi dengan menggunakan algoritma Huffman (*Huffman Tree)*. Algoritma deflate ini bersifat *loseless Compression*. Hal ini karena algoritma deflate ini menggabungkan dua algoritma kompresi yang bersifat *loseless.*

2.2.9.1 Proses Encoding

Proses encoding dari algoritma deflate ini dilakukan pada dua tahap. Tahap pertama adalah melakukan proses pembuatan blok-blok atau penyingkatan karakter dengan menggunakan algoritma LZ77. Tahap kedua adalah mengambil

hasil penyingkatan karakter dari algoritma LZ77 dan melakukan proses kompresi dengan menggunakan *Huffman Tree* terhadap karakter tersebut. Hasil dari proses encoding algoritma deflate ini adalah berupa karakter yang telah disingkat yang merupakan hasil dari encoding algoritma LZ77 dan memiliki kode biner yang lebih pendek yang merupakan hasil dari encoding algoritma Huffman.

2.2.9.2 Proses Decoding

Proses decoding pada algoritma deflate merupakan kebalikan dari proses encodingnya. Langkah pertama adalah melakukan proses decoding dengan menggunakan algoritma Huffman. Kemudian langkah kedua adalah mengambil hasil dari proses decoding dengan menggunakan algoritma Huffman dan melakukan proses decoding kembali dengan menggunakan algoritma LZ77.

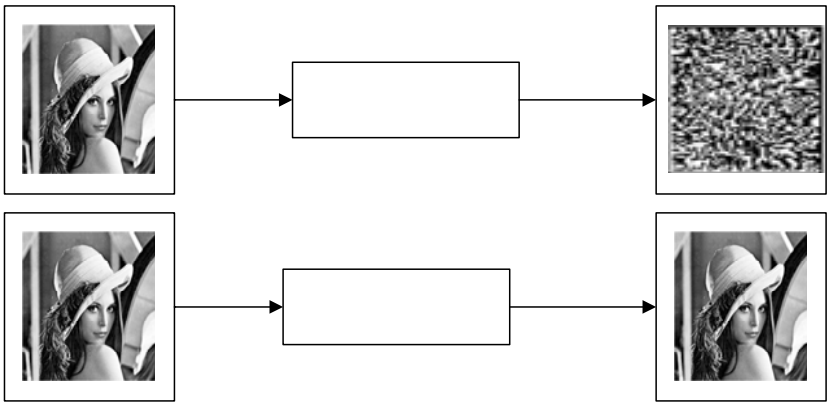
2.3 Steganografi

Steganografi (*steganography*) berasal dari bahasa Yunani yaitu *Steganós* yang berarti menyembunyikan dan *Graptos* yang artinya tulisan sehingga secara keseluruhan artinya adalah tulisan yang disebunyikan. Secara umum steganografi adalah ilmu dan seni menyembunyikan pesan rahasia (*hiding message*) sedemikian sehingga keberadaan (eksistensi) pesan tidak terdeteksi oleh indera manusia.

Steganografi membutuhkan dua properti: wadah penampung (*cover*) dan data rahasia yang akan disembunyikan. Steganografi *digital* menggunakan media *digital* sebagai *cover*, misalnya citra, suara, teks, dan video. Data rahasia yang

disembunyikan juga dapat berupa citra, suara, teks, atau video. Media yang telah disisipi data disebut *stegodata* atau *stegofile*. Proses penyembunyian data ke dalam media disebut penyisipan (*embedding*), sedangkan proses sebaliknya disebut ekstraksi (*extraction*).

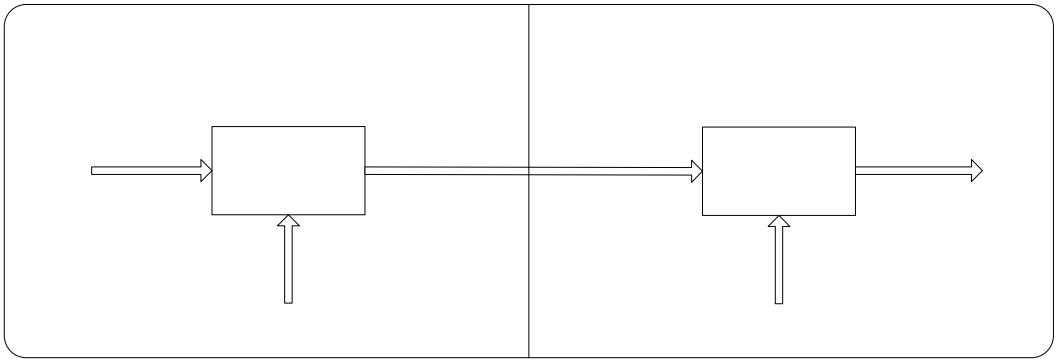
Steganografi berbeda dengan kriptografi, dimana pihak ketiga dapat mendeteksi adanya data (*chipertext*) karena hasil dari kriptografi berupa data yang berbeda dari bentuk aslinya dan biasanya datanya seolah-olah berantakan tetapi dapat dikembalikan ke bentuk semula. Ilustrasi mengenai perbedaan kriptografi dan steganografi dapat dilihat pada Gambar 2.2.



Gambar 2.2 Ilustrasi kriptografi dan steganografi pada citra digital

Steganografi membahas bagaimana sebuah pesan dapat disisipkan ke dalam sebuah berkas media sehingga pihak ketiga tidak menyadarinya. Steganografi memanfaatkan keterbatasan sistem indera manusia seperti mata dan telinga. Dengan adanya keterbatasan inilah metode steganografi ini dapat diterapkan pada berbagai media *digital*. Hasil keluaran dari steganografi ini memiliki bentuk persepsi yang sama dengan bentuk aslinya, tentunya persepsi di sini sebatas oleh kemampuan indera manusia, tetapi tidak oleh komputer atau perangkat pengolah

digital lainnya. Ilustrasi proses steganografi dapat dilihat pada Gambar 2.3.



Gambar 2.3 Diagram Steganografi

2.3.1 Sejarah Steganografi

Steganografi sudah digunakan sejak dahulu kala untuk kepentingan politik, militer, diplomatik, serta untuk kepentingan pribadi sebagai alat. Beberapa contoh penggunaan steganografi:

1. Steganografi pada tubuh budak: Pada jaman Yunani Kuno, Herodotus (sejarawan Yunani) menyampaikan pesan dengan cara mencukur kepala budak dan mentato kepalanya dengan pesan tersebut. Kemudian saat rambutnya tumbuh kembali, budak dikirimkan pada tempat tujuan dan saat rambutnya kembali digunduli maka pesan akan terbaca.
2. Steganografi dalam lapisan lilin: Masih pada jaman Yunani Kuno, pesan rahasia di ukir pada kayu kemudian diberi lapisan lilin untuk menutupi pesan tersebut, dengan begitu pesan dapat disampaikan tanpa menimbulkan kecurigaan.
3. Steganografi pada kertas: Pada Perang Dunia II, pemerintah Amerika Serikat menulis pesan rahasia pada tentaranya yang di tawan oleh Jerman dengan

menggunakan tinta tak terlihat di atas atau di bagian kosong pesan lainnya dan untuk mendeteksinya digunakan air.

4. Steganografi dengan *microdots*: Masih pada Perang Dunia II, agen mata-mata menggunakan *microdots* untuk mengirimkan informasi. Penggunaan teknik ini biasa digunakan pada *microfilm* chip yang harus diperbesar sekitar 200 kali.
5. Steganografi dunia *digital*: Pada suatu pengarahan yang dilakukan oleh FBI pada akhir September 2001, asisten direktur FBI, Ron Dick menyatakan kemungkinan para pembajak pesawat pada serangan 11 September ke gedung *World Trade Center* menggunakan teknologi internet seperti *website* perusahaan palsu atau *website* porno, *e-mail*, *chat rooms, bulletin boar*ds untuk mengkoordinasi serangan.

2.3.2 Faktor Penting Dalam Steganografi

Penyembunyian data rahasia ke dalam media *digital* mengubah kualitas media tersebut. Faktor penting dalam penyembunyian data diantaranya adalah:

1. *Fidelity*. Mutu media penampung tidak jauh berubah. Setelah penambahan data rahasia pengamat tidak mengetahui kalau di dalam media tersebut terdapat data rahasia.
2. *Recovery*. Data yang disembunyikan harus dapat diungkapkan kembali (*recovery*). karena tujuan steganografi adalah menyembunyikan data, maka sewaktu-waktu data rahasia didalam media penampung harus dapat diambil kembali untuk digunakan lebih lanjut.

2.3.3 Media Steganografi

Steganografi menggunakan sebuah *cover*, tujuannya sebagai kamuflase dari pesan yang sebenarnya. Banyak format berkas digital yang dapat dijadikan media untuk menyembunyikan pesan. Pada jaman modern seperti saat ini, steganografi biasanya dilakukan dengan melibatkan berkas-berkas seperti teks, citra, suara dan video.

2.3.4 Teknik Steganografi

Saat ini setidaknya terdapat 6 teknik steganografi, yaitu:

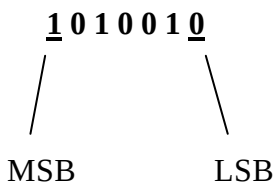
1. Teknik substitusi *(Substitution Techniques),* yaitu dengan melakukan substitusi piksel-piksel tertentu dari berkas *cover* dengan piksel-piksel informasi yang disimpan. Contohnya adalah metode LSB.
2. Teknik Domain Transformasi *(Domain Transform Techniques),* yaitu dengan cara menyimpan informasi rahasia pada transformasi ruang (misalnya domain frekuensi) dari berkas *cover.* Contoh algoritma yang tergolong teknik ini adalah steganografi pada domain DCT *(Discrete Cosine Transform).*
3. Teknik Spread Spectrum *(Spread Spectrum Techniques),* ide dari teknik ini mengadopsi teknik *spread spectrum* pada saluran komunikasi. Dalam teknik ini informasi rahasia yang disimpan disebarkan pada suatu frekuensi tertentu dari berkas *cover.*
4. Teknik Statistik *(Statistical Techniques)* Dengan teknik ini data di *encoding* melalui pengubahan beberapa informasi statistik dari berkas *cover.* Berkas *cover* di bagi dalam blok-blok dimana setiap blok tersebut menyimpan satu

piksel informasi rahasia yang disembunyikan. Jika piksel yang ditemukan pada suatu blok untuk menyimpan data adalah piksel T maka tidak dilakukan perubahan nilai piksel, jika sebaliknya maka dilakukan perubahan nilai piksel. Meskipun secara teoritis mungkin untuk dilakukan, pada kenyataannya teknik ini agak sulit untuk diimplementasikan.

5. Teknik Distorsi *(Distortion Techniques),* informasi yang hendak disembunyikan disimpan berdasarkan distorsi sinyal.
6. Teknik Pembangkitan Wadah *(Cover Generation Techniques),* Teknik ini menyembunyikan informasi rahasia dengan jalan (atau sejalan) dengan pembangkitan *cover*. Misalnya aplikasi pembangkitan teks/paragraf bahasa inggris otomatis.

2.3.5 Least Significant Bit

Metode steganografi yang paling umum pada tipe berkas citra adalah *least significant bit* (LSB). Metode ini menyembunyikan data dengan mengganti bit-bit data yang paling tidak berarti di dalam *cover* dengan bit-bit data rahasia. Pada susunan bit di dalam sebuah *byte* (1 *byte* = 8 bit), ada bit yang paling berarti *(most significant bit* (MSB) dan bit yang paling kurang berarti *least significant bit* (LSB).



Bit yang cocok untuk diganti adalah bit *LSB,* sebab perubahan tersebut hanya mengubah nilai *byte* satu lebih tinggi atau satu lebih rendah dari nilai

sebelumnya. Misalkan pada cover *citra*, *byte* tersebut menyatakan warna merah, maka perubahan satu bit LSB tidak mengubah warna merah tersebut secara berarti, apalagi mata manusia tidak dapat membedakan perubahan yang kecil. Sebagai contoh, misalnya terdapat data *raster original file* citra yang akan digunakan sebagai *cover* sebagai berikut:

00100111 11101001 11001000

00100111 11001000 11101001

11001000 00100111 11101001

Sedangkan representasi biner huruf A adalah 01000001, dengan menyisipkannya ke dalam pixel di atas maka akan dihasilkan

00100111 11101001 11001000

00100111 11001000 11101000

11001000 00100111 11101001

Terlihat hanya tiga bit rendah yang berubah (cetak tebal), untuk penglihatan mata manusia sangatlah mustahil untuk dapat membedakan warna pada *file* citra yang sudah diisi pesan rahasia jika dibandingkan dengan *file* citra asli sebelum disisipi dengan pesan rahasia.

2.4 Pengkodean Data

Data disimpan di dalam komputer pada *main memory* untuk diproses. Sebuah karakter data disimpan dalam *main memory* menempati posisi 1 byte. Komputer generasi pertama, 1 byte terdiri dari 4 bit, komputer generasi kedua, 1 byte terdiri dari 6 bit dan komputer generasi sekarang, 1 byte terdiri dari 8 bit. Suatu karakter

data yang disimpan di *main memory* diwakili oleh kombinasi dari dijit biner (*binary digit* atau bit). Kode biner dapat digunakan untuk mewakili suatu karakter.

Suatu komputer yang berbeda menggunakan kode biner yang berbeda untuk mewakili suatu karakter. Komputer yang 1 byte terdiri dari 4 bit, menggunakan kode binary yang berbentuk kombinasi 4 bit, yaitu BCD (*Binary coded decimal*). Komputer yang menggunakan 6 bit untuk 1 bytenya, menggunakan kode biner yang terdiri dari kombinasi 6 bit, yaitu SBCDIC (*Standard Binary Coded Decimal Interchange Code*). Komputer yang terdiri dari 8 bit, menggunakan kode biner yang terdiri dari kombinasi 8 bit, yaitu EBCDIC (*extended Binary coded decimal interchange code*) atau ASCII (*American standard code of information interchange*).

1. BCD (*Binary Coded Decimal*)

BCD merupakan kode biner yang digunakan hanya untuk mewakili nilai desimal saja, yaitu angka 0 sampai dengan 9. BCD menggunakan kombinasi dari 4 bit, sehingga sebanyak 16 ($2^4 = 16$) kemungkinan kombinasi yang dapat diperoleh dan hanya 10 kombinasi yang dipergunakan. Kode BCD yang orisinil sudah jarang dipergunakan untuk komputer generasi sekarang, karena tidak dapat mewakili huruf atau simbol-simbol karakter khusus.

2. SBCDIC (*Standard Binary Coded Decimal Interchange Code*)

SBCDIC merupakan kode biner perkembangan dari BCD. BCD dianggap tanggung, karena masih ada 6 kombinasi yang tidak dipergunakan, tetapi tidak dapat digunakan untuk mewakili karakter yang lainnya. SBCDIC menggunakan kombinasi 6 bit , sehingga lebih banyak kombinasi yang bisa dihasilkan, sebanyak

64 kombinasi kode, yaitu 10 kode untuk digit angka, 26 kode untuk huruf alphabet dan sisanya karakter-karakter khusus yang dipilih.

Posisi bit di SBCDIC dibagi menjadi 2 *zone*, yaitu 2 bit pertama disebut dengan *alphabet position* dan 4 bit berikutnya disebut dengan *numeric bit position.*

3. EBCDIC (*Extended Binary Coded Decimal Interchange Code*)

EBCDIC terdiri dari kombinasi 8 bit yang memungkinkan untuk mewakili karakter sebanyak 256 kombinasi karakter. Pada EBCDIC, *high-order* bits atau 4 bit pertama disebut dengan zone bits dan *low order* bits atau 4 bit kedua disebut dengan numeric bit.

4. ASCII (*American standard code for information interchange*) 7 bit

Kode ASCII yang standard menggunakan kombinasi 7 bit, dengan kombinasi sebanyak 127 dari 128 kemungkinan kombinasi. Kode ASCII 7 bit ini terdiri dari dua bagian, yaitu *control characters* dan *information characters* yang merupakan karakter-karakter yang mewakili data.

5. ASCII 8 bit

Karakter-karakter graphic yang tidak dapat diwakili oleh ASCII 7 bit, dapat diwakili dengan ASCII 8 bit karena lebih banyak memberikan kombinasi karakter.