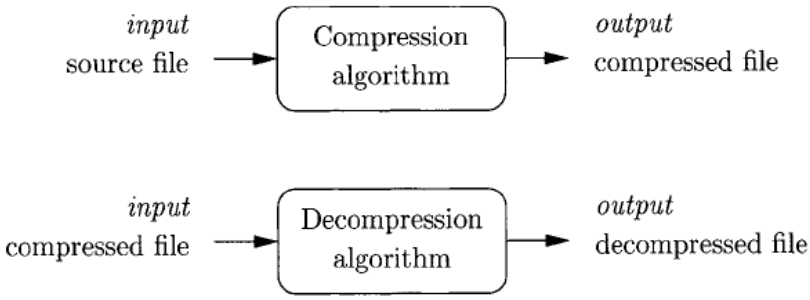
BAB II
DASAR TEORI

2.1. Teknik Kompresi Data

Kompresi data dalam konteks ilmu komputer adalah suatu ilmu (dan seni) merepresentasikan informasi dalam bentuk yang padat[5]. Suatu proses mengubah masukan aliran data (aliran sumber atau data mentah asli) menjadi aliran data lain (keluaran, aliran *bit*, atau aliran terkompresi) yang memiliki ukuran lebih kecil[4].

Kompresi data menjadi kebutuhan umum pada banyak perangkat lunak aplikasi dan juga merupakan wilayah penelitian yang penting dan aktif dalam ilmu komputer. Tanpa teknik kompresi tidak ada *internet* yang akan terus tumbuh, *TV digital*, komunikasi bergerak (*mobile*), atau pertumbuhan teknik komunikasi *video* yang akan menjadi perkembangan praktis[5].



Gambar 2.1 Diagram konteks kompresi dan dekompresi data secara umum[5]

Dari diagram konteks pada Gambar 2.1 dapat dilihat bahwa tahap kompresi adalah *file input* berupa *file* asli lalu dikompresi menggunakan algoritma kompresi tertentu dan menghasilkan *file* terkompresi, sedangkan tahap dekompresi adalah *file* input berupa *file* terkompresi lalu didekompresi menggunakan algoritma yang sama ketika melakukan proses kompresi untuk menghasilkan *file* asli (baik itu yang telah mengalami kehilangan beberapa *bit* data, ataupun tetap utuh).

Berdasarkan kebutuhan rekonstruksinya, skema kompresi data dapat dibagi ke dalam dua kelas besar: skema kompresi *lossless* dan skema kompresi *lossy*[6]. Berikut ini adalah penjelasan untuk dua skema besar itu:

1. *Lossless compression*

Teknik kompresi *lossless*, seperti nama metodenya, metode kompresi ini melibatkan tidak kehilangan informasi. Jika data telah terkompresi secara *lossless*, data asli dapat dipulihkan secara utuh dari data terkompresi. Kompresi *lossless* umumnya digunakan untuk aplikasi yang tidak bisa mentolerir perbedaan antara data asli dan hasil rekonstruksi[6].

Kompresi teks merupakan area yang penting untuk kompresi *lossless*. Hal ini sangat penting bahwa rekonstruksi identik dengan teks asli, perbedaan yang sangat kecil dapat menghasilkan laporan dengan arti yang berbeda, sebagai contoh pada kalimat "*Do not send money*" dan "*Do now send money*" yang tentunya akan berakibat fatal jika proses dekompresi menghasilkan informasi berbeda[6]. Macam-macam algoritma dengan metode kompresi *lossless*[12]:

a. *LZW*,
b. *DEFLATE*,
c. *Lempel–Ziv–Markov chain algorithm* (*LZMA*),
d. *Waveform audio format*,
e. *Free Lossless Audio Codec*,
f. *Dolby TrueHD*,
g. *JPEG 2000*,
h. *PNG*,
i. *OpenCTM*,
j. *Huffyuv*,
k. dan lain-lain.

2. Losy compression

Seperti nama metodenya, metode kompresi ini melibatkan beberapa kehilangan informasi, dan data yang telah dikompresi menggunakan metode *lossy* umumnya tidak dapat dipulihkan atau direkonstruksi persis. Sebagai imbalan untuk menerima distorsi ini dalam rekonstruksi, kita umumnya dapat memperoleh rasio kompresi jauh lebih tinggi daripada yang mungkin dihasilkan jika menggunakan metode kompresi *lossless*[6].

Dalam banyak aplikasi, kurangnya rekonstruksi persis tidak terlalu dimasalahkan. Misalnya, ketika menyimpan atau mengirimkan informasi suara,

nilai yang tepat dari masing-masing sampel dari informasi suara tidak diperlukan. Tergantung pada kualitas yang dibutuhkan dari informasi suara yang direkonstruksi. Jika kualitas dari informasi suara yang direkonstruksi hampir tidak terlihat perbedaanya dengan mendengar di telepon, kehilangan yang signifikan dari informasi dapat ditoleransi[6]. Macam-macam algoritma dengan metode kompresi *lossy*[12]:

a. *JPEG*,
b. *MPEG-1*,
c. *MPEG-2*,
d. *MPEG-4*,
e. *AAC*,
f. *MP3*,
g. *WMA*,
h. dan lain-lain.

Adapun beberapa hal yang perlu diperhatikan dalam setiap teknik kompresi dikarenakan dapat dijadikan acuan untuk menjadi bahan pertimbangan penilaian suatu algoritma teknik kompresi yang paling efisien adalah[5]:

1. Rasio kompresi

Perbandingan antara ukuran data asli sebelum terkompresi dengan ukuran data yang telah terkompresi. Angka rasio kompresi menunjukkan perbandingan ukuran yang dicapai dalam satu proses kompresi, semakin kecil nilai rasio kompresi maka hasil kompresi semakin memuaskan.

$$Rasio\ kompresi = \frac{ukuran\ setelah}{ukuran\ sebelum} \qquad (2.1)$$

2. Faktor kompresi

Perbandingan antara ukuran data setelah terkompresi dengan ukuran data asli sebelum terkompresi. Angka faktor kompresi menunjukkan berapa kali kehandalan hasil kompresi yang dicapai dalam satu proses kompresi, semakin besar nilai faktor kompresi maka hasil kompresi semakin memuaskan.

$$Faktor\ kompresi = \frac{ukuran\ sebelum}{ukuran\ setelah} \qquad (2.2)$$

3. Persentase penghematan

Persentase dari perbandingan antara selisih ukuran data dari sebelum terkompresi dan sesudah terkompresi dengan ukuran data sebelum terkompresi. Angka persentase penghematan menunjukkan seberapa besar penghematan yang dicapai dalam satu proses kompresi, semakin besar persentase penghematan maka hasil kompresi semakin memuaskan.

$$Persentase\ penghematan = \frac{ukuran\ sebelum\ - ukuran\ setelah}{ukuran\ sebelum} \times 100\% \qquad (2.3)$$

2.2. Algoritma LZW

LZW merupakan algoritma *dictionary based compression*, yaitu algoritma kompresi yang berbasiskan kamus, dimana pendekatannya tidak menggunakan model statistik, namun mengidentifikasi adanya pola perulangan karakter[5].

Kelebihan teknik kompresi *LZW* adalah kecepatan waktu kompresi yang sangat singkat dengan tingkat kompresi yang cukup baik yaitu persentase penghematan mencapai sekitar 60.2% ± 28.9[7].

Berikut ini adalah algoritma *pseudocode* proses kompresi pada algoritma *LZW* secara umum[5]:

Algorithm 7.1 LZW encoding

```
1: word ← ''
2: while not EOF do
3:    x ← read_next_character()
4:    if word + x is in the dictionary then
5:       word ← word + x
6:    else
7:       output the dictionary index for word
8:       add word + x to the dictionary
9:       word ← x
10:   end if
11: end while
12: output the dictionary index for word
```

Gambar 2.2 Algoritma *pseudocode* proses kompresi[5]

Prinsip *LZW* adalah *encoder* memasukan simbol satu-persatu dan mengakumulasinya ke dalam *string* I. Setelah tiap simbol dimasukkan dan digabung ke *string* I, kamus mencari untuk *string* I. Selama *string* I ditemukan dalam kamus, proses akan terus berlanjut. Pada saat tertentu, menambahkan

simbol berikutnya menyebabkan pencarian gagal; *String* I ada di kamus, sementara *string* Ix (simbol x yang digabungkan dengan I) tidak ada. Pada saat ini *encoder* (1) mengeluarkan *pointer* kamus yang merujuk *string* I, (2) menyimpan *string* Ix (yang bisa kita sebut frasa) ke dalam entri kamus tambahan selanjutnya, dan (3) menginisialisasi *string* I dengan simbol x[4]. Jelasnya dapat dilihat pada Tabel 2.1 dan Tabel 2.2.

Tabel 2.1 Tabel *dictionary* algoritma *LZW*[5]

Kode	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
Frase	A	B	C	D	E	F	G	H	I	J	K	L	M	N	…

kode	…	17	18	19	20	21	22	23	24	25	26	27	…	256
Frase	…	Q	R	S	T	U	V	W	X	Y	Z	Space	…	9

Tabel 2.2 Langkah proses kompresi algoritma *LZW*[5]

Langkah	Keterangan
1	Langkah Awal: string: ‘ ’
2	Konten yang akan dibaca: ACBBAAC read next_character x: A (dalam dictionary) string+x: A string: A
3	Konten yang akan dibaca: CBBAAC read next_character x: C string+x: AC (tidak ada di dictionary) output: 1 entri baru dictionary: 257=AC string: C
4	Konten yang akan dibaca: BBAAC read next_character x: B string+x: CB (tidak ada di dictionary) output: 3 entri baru dictionary: 258=CB string: B
5	Konten yang akan dibaca: BAAC read next_character x: B string+x: BB (tidak ada di dictionary) output : 2 entri baru dictionary: 259=BB string: B
6	Konten yang akan dibaca: AAC read next_character x: A string+x: BA (tidak ada di dictionary) output: 2 entri baru dictionary: 260=BA string: A
7	Konten yang akan dibaca: AC read next_character x: A string+x: AA (tidak ada di dictionary) output: 1 entri baru dictionary: 261=AA string: A
8	Konten yang akan dibaca: C read next_character x: C string+x: AC (ada di dictionary) string: AC
9	Konten yang akan dibaca: sudah tidak ada (end of file) read next_character x: EOF (i.e. end of file) output: 257

Berikut ini adalah algoritma *pseudocode* proses dekompresi pada algoritma *LZW* secara umum[5]:

Algorithm 7.2 LZW decoding

```
1: read a token x from the compressed file
2: look up dictionary for element at x
3: output element
4: word ← element
5: while not EOF do
6:    read x
7:    look up dictionary for element at x
8:    if there is no entry yet for index x then
9:       element ← word + firstCharOfWord
10:   end if
11:   output element
12:   add word + firstCharOfElement to the dictionary
13:   word ← element
14: end while
```

Gambar 2.3 Algoritma *pseudocode* proses dekompresi *LZW*[5]

Untuk memahami bagaimana *decoder LZW* bekerja, perlu untuk diingat kembali tiga langkah yang dilakukan *encoder* setiap kali menulis sesuatu pada *output stream*. Yaitu (1) mengeluarkan *pointer* yang menunjuk ke kamus *string* I, (2) menyimpan *string* Ix dalam entri yang tersedia berikutnya dari kamus, dan (3) akan menginisialisasi *string* I dengan simbol x. *Decoder* dimulai dengan entri awal dari kamus yang diinisialisasi dengan semua simbol dari alfabet (biasanya 256 simbol). Kemudian membaca *input stream* (yang terdiri dari *pointer* ke dalam kamus) dan menggunakan setiap *pointer* untuk mengambil simbol terkompresi dari kamus dan menuliskannya pada *output stream*. Proses ini juga membangun kamus dengan cara yang sama dengan *encoder* (fakta ini biasanya dinyatakan dengan mengatakan bahwa *encoder* dan *decoder* tersinkronisasi, atau bahwa kedua proses bekerja secara selaras)[4].

Pada Tabel 2.3 adalah contoh langkah proses dekompresi menggunakan algoritma *LZW* berdasarkan algoritma *pseudocode* proses dekompresi *LZW* pada Gambar 2.3. Diketahui *token* (*compressed text*) yang akan didekompresi adalah *token* “1 3 2 2 1 257”[5].

Tabel 2.3 Langkah proses dekompresi algoritma *LZW*

Langkah	Keterangan
1	Langkah Awal: Token yang akan dibaca 1 3 2 2 1 257 baca token x: 1 elemen: A output: A string: A
2	Token yang akan dibaca: 3 2 2 1 257 baca token x: 3 elemen: C output: C tambah entri baru ke dictionary: 257=AC string:C
3	Token yang akan dibaca: 2 2 1 257 baca token x:2 elemen: B output: B tambah entri baru ke dictionary: 258=CB string: B
4	Token yang akan dibaca: 2 1 257 baca token x: 2 elemen: B output: B tambah entri baru ke dictionary: 259=BB string: B
5	Token yang akan dibaca: 1 257 baca token x: 1 elemen: A output: A tambah entri baru ke dictionary: 260=BA string: A
6	Token yang akan dibaca: 257 baca token x: 257 elemen: AC output: AC tambah entri baru ke dictionary: 261=AA string: AC
7	Token yang akan dibaca: (sudah tidak ada) baca token x: EOF (End of File) selesai

2.3. Bahasa Pemrograman PHP

PHP (akronim rekursif untuk "*PHP: Hypertext Preprocessor*") adalah *open source* bahasa *scripting* untuk tujuan umum yang banyak digunakan, sangat cocok untuk pengembangan *web* dan dapat ditanamkan ke dalam *HTML*[9].

PHP adalah bahasa *scripting server-side* yang di desain secara khusus untuk *Web*. Diantara sebuah halaman *HTML*, anda dapat menanamkan kode *PHP* yang akan dieksekusi tiap kali halaman dikunjungi. kode *PHP* anda diinterpretasikan

pada *Web server* dan menghasilkan *HTML* atau keluaran lain yang pengunjung akan lihat[8].

Rasmus Lerdorf pertama pertama kali menemukan *PHP* pada tahun 1994, pengumuman versi kedua *PHP* (*PHP*/*FI*) pada tahun 1996, pengumuman *PHP* 3.0 pada juni 1998, *PHP* 4.0 dirilis pada 22 Mei 2000[10]. Adapun versi *PHP* terbaru saat ini adalah *PHP* 5.0 yang dirilis pada Juli 2004.

Kelebihan dari bahsa pemrograman *PHP* adalah dapat menjalankan perintah *query DDL* dan *DML* seperti *CREATE, INSERT, UPDATE, DELETE, SELECT*. *PHP* juga banyak mendukung sistem manajemen basis data, seperti *MySQL*, *PostgreSQL*, *Interbase*, *ODBC*, *mSQL*, *Oracle*, dan *Sybase*[11].

2.4. Sistem Manajemen Basis Data MySQL

Sistem manajemen basis data (*DBMS*) adalah sekumpulan program yang digunakan untuk mendefiniskan, administrasi, dan memproses basis data dan aplikasi-aplikasi yang terkait dengan basis data. Sebuah *DBMS* adalah perangkat yang anda gunakan untuk membuat struktur dan beroperasi pada data yang berada diantara basis data[1].

MySQL adalah sebuah sistem manajemen basis data relasional yang di desain untuk kegunaan di aristektur *client*/*server*. *MySQL* menjadi sistem basis data *open source* yang paling populer dan paling sukses di dunia. Popularitas ini sebagian besar dikarenakan karena kehandalan, kinerja, dan kemudahan penggunaanya. Lebih dari 8 juta instalasi dari produk *MySQL* di seluruh dunia[3].

Sebuah *RDBS* (Sistem basis data relasional) adalah penyimpanan data dan layanan pengambilan berdasarkan model relasional data, seperti yang diusulkan oleh E.F. Codd pada tahun 1970. Sistem ini adalah mekanisme penyimpanan standar untuk data terstruktur[3].

Beberapa istilah dalam sistem manajemen basis data diantaranya adalah: (*Table, Record, Field, Key,* dan *SQL*). *Table*, merupakan kumpulan data yang diorganisasikan ke dalam *record* dan *field*. *Record,* merupakan kumpulan nilai yang saling terkait. *Field,* merupakan kolom dari sebuah *table*, *field* memiliki tipe dan ukuran yang dapat disesuaikan. *Key,* merupakan suatu *field* yang dijadikan

sebagai kunci operasi tabel (*Primary Key*, *Foreign Key*, dan *Composite Key*). *SQL / Query* merupakan instruksi khusus untuk pengolahan basis data[2].