BAB III

LANDASAN TEORI

A. Kompresi Data

Kompresi data merupakan proses mengkonversi input data *stream* (aliran sumber) menjadi aliran data yang lain (output, *bitstream*, atau aliran terkompresi) dengan ukuran yang lebih kecil. Sebuah *stream* dapat berupa file atau *buffer* di memori (David , 2006).

Ada berbagai kriteria untuk mengukur kinerja algoritma kompresi. Umumnya efisiensi ruanglah yang merupakan perhatian utama, sedangkan efisiensi waktu merupakan faktor lainnya (Kodituwakku & Amarasinghe, 2010). Berikut adalah beberapa langkah-langkah yang umum digunakan untuk mengevaluasi kinerja dari sebuah metode kompresi:

1. Rasio kompresi

Rasio kompresi merupakan perbandingan antara ukuran data setelah dikompresi dengan ukuran data asli. Jika nilai rasio kompresi kurang dari satu berarti data tersebut terkompresi. Begitupun sebaliknya jika nilai rasio kompresi lebih dari satu berarti data tersebut tidak dikompresi melainkan ditambahkan atau diekspansi. Sedangkan jika nilai rasio kompresi sama dengan satu berarti data tersebut tidak terkompresi.

$$Rasio\ kompresi = \frac{ukuran\ data\ output}{ukuran\ data\ input}$$

2. Faktor kompresi

Menurut David (2006), faktor kompresi merupakan kebalikan dari rasio kompresi. Dalam hal ini, apabila nilai lebih besar dari satu berarti data terkompresi sedangkan nilai-nilai kurang dari satu menunjukan data terekspansi (David , 2006).

$$Faktor\ kompresi = \frac{ukuran\ data\ input}{ukuran\ data\ output}$$

3. Persentase penghematan

Persentase penghematan merupakan perhitungan persentase penyusutan data input.

$$Persentase\ penghematan\ = \frac{ukuran\ data\ input - \ ukuran\ data\ output}{ukuran\ data\ input}\ \%$$

B. Burrows-Wheeler Compression Algorithm (BWCA)

Skema umum BWCA terdiri dari empat tahap sebagaimana dapat dilihat pada gambar berikut (Abel, 2010):

Gambar 3.1 Skema Umum BWCA

Untuk proses kompresi, tahap-tahap tersebut diproses secara berurutan dari kiri ke kanan. Sebaliknya untuk proses dekompresi tahap-tahap tersebut diproses dari kanan ke kiri.

1. Transformasi Burrows-Wheeler (BWT)

Metode kompresi umumnya beroperasi dalam mode *streaming* yang beroperasi dengan membaca kode input berupa satu atau beberapa *byte* dan di proses bertahap sampai akhir file ditemukan. BWT bekerja dalam mode blok, di mana input *stream* dibaca blok demi blok dan setiap blok dikodekan secara terpisah sebagai satu string (David , 2006). BWT mengambil sebuah blok data dan menyusun ulang data tersebut menggunakan algoritma pengurutan. Hasil dari blok output memiliki elemen data yang sama seperti ketika blok di baca, hanya saja urutannya yang berbeda. Transformasi ini dapat dikembalikan (*reversible*), sehingga urutan asli dari elemen data dapat dipulihkan tanpa kehilangan informasi. BWT dilakukan pada seluruh blok data sekaligus sehingga metode ini juga disebut sebagai *block sorting* (Chang, et al., 2009).

BWT tidak memampatkan data namun melakukan permutasi terhadap data input sehingga elemen-elemen data input tersebut berubah posisi. BWT bekerja dengan cara membaca sebuah blok data dengan panjang n sebagai sebuah string (S = S1..Sn), kemudian melakukan rotasi (*cyclic shift*) sebanyak n-1 kali, sehingga menghasilkan matriks M berukuran n x n, dan mengurutkan matriks tersebut secara leksikographi. Hasil transformasi ini berupa semua elemen pada kolom terakhir matriks, dan indeks dari posisi string aslinya (Mantaci, et al., 2007). Tabel 3.1 berikut memberikan gambaran proses BWT dari inputan string “RAKSASA”:

Tabel 3.1 Proses BWT pada String “RAKSASA”

Indeks	Input di rotasi							Rotasi diurutkan							Hasil
	F						L	F						L	
0	R	A	K	S	A	S	A	A	K	S	A	S	A	R	R
1	A	K	S	A	S	A	R	A	R	A	K	S	A	S	S
2	K	S	A	S	A	R	A	A	S	A	R	A	K	S	S
3	S	A	S	A	R	A	K	K	S	A	S	A	R	A	A
4	**A**	**S**	**A**	**R**	**A**	**K**	**S**	**R**	**A**	**K**	**S**	**A**	**S**	**A**	**A**
5	S	A	R	A	K	S	A	S	A	R	A	K	S	A	A
6	A	R	A	K	S	A	S	S	A	S	A	R	A	K	K

Seperti yang terlihat pada tabel 3.1 di atas, dengan melakukan BWT, string “RAKSASA” akan di ubah menjadi “RSSAAAK”. String ini kemudian disimpan bersama dengan indeks dari posisi string aslinya yaitu empat yang diperlukan pada saat proses pengembalian.

Terdapat dua metode untuk proses pengembalian BWT yaitu *reverse transform by sorting* dan *reverse transform by permutation*. Metode yang pertama dilakukan dengan cara menambahkan dan mengurutkan teks keluaran pada matriks M dengan ukuran N x N, namun cara ini kurang efisien karena membutuhkan waktu yang lama (Hamid & Ayyash, 2011). Metode yang kedua merupakan cara yang efisien. Algoritma invers BWT berdasarkan metode kedua ini dapat dilihat pada *pseudocode* berikut (Adjeroh, et al., 2008):

BWT-Decode(L, a)

a. for c ← 1 to |Σ| do

b. K[c] ← 0

c. end for

d. for i ← 1 to n do

e. C[i] ← K[L[i]];

f. K[L[i]] ← K[L[i]] + 1;

g. end for

h. sum ← 1

i. for c ← 1 to |Σ| do

j. M[c] ← sum;

k. sum ← sum + K[c];

l. end for

m. i ← a

n. for j ← n downto 1 do

o. Q[j] ← L[i];

p. i ← C[i] + M[L[i]];

q. end for

2. *Global Sructure Transform* (GST)

GST adalah sebuah proses pengubahan inputan data yang memiliki konteks lokal menjadi output data yang memiliki konteks global. Pada skema BWCA, GST ditempatkan pada tahap kedua karena umumnya output dari BWT mengandung distribusi probabilitas lokal, yakni banyak fragmen simbol dengan konteks yang sama muncul dengan distribusi probabilitas yang berbeda. Pengubahan konteks lokal hasil output BWT menjadi konteks global ini biasanya menggunakan *recency ranking scheme* seperti MTF, M1FF dan sebagainya (Abel, 2010).

MTF dan M1FF tidak mengurangi ukuran data melainkan mengubah simbol inputan menjadi kode integer berupa posisi indeks dari simbol tersebut di dalam *list*

simbol. MTF melakukan *update list* simbol dengan memindahkan karakter yang telah dikodekan keposisi pertama pada *list* simbol, sedangkan M1FF memindahkan simbol yang telah dikodekan ke posisi awal apabila posisi sebelumnya berada pada posisi kedua. Untuk simbol dengan posisi yang lebih tinggi M1FF memindahkannya ke posisi kedua pada *list* simbol.

Sebagai contoh diasumsikan bahwa *list* MTF dan M1FF diinisialisasi dengan empat simbol secara berurutan yaitu [A, K, S, R]. Apabila output BWT dari string "RAKSASA" yaitu "RSSAAAK" dikodekan dengan MTF maka akan menghasilkan output "3302003" sedangkan jika dikodekan dengan M1FF maka output yang akan dihasilkan yaitu "3311003". Perbedaan proses *encoding* tersebut disajikan pada tabel 3.2 dan tabel 3.3 berikut.

Tabel 3.2 Proses *Encoding* MTF

Simbol yang akan dikodekan	Daftar simbol MTF				Kode MTF
	0	1	2	3	
RSSAAAK	A	K	S	(R)	3
SSAAAK	R	A	K	(S)	33
SAAAK	(S)	R	A	K	330
AAAK	S	R	(A)	K	3302
AAK	(A)	S	R	K	33020
AK	(A)	S	R	K	330200
K	A	S	R	(K)	3302003

Tabel 3.3 Proses *Encoding* M1FF

Simbol yang akan dikodekan	Daftar simbol M1FF				Kode M1FF
	0	1	2	3	
RSSAAAK	A	K	K	(R)	3
SSAAAK	A	R	K	(S)	33
SAAAK	A	(S)	R	K	331
AAAK	S	(A)	R	K	3311
AAK	(A)	S	R	K	33110
AK	(A)	S	R	K	331100
K	A	S	R	(K)	3311003

Baik MTF maupun M1FF pada saat melakukan pengkodean sama-sama merubah inputan dari data asli dengan indeks dari posisi simbol tersebut. Sedangkan pada proses *decoding* kedua transformasi ini merubah inputan berupa kode indeks dari simbol dengan simbol pada indeks tersebut. Tabel 3.4 dan tabel 3.5 berikut menampilkan contoh proses *decoding* dari MTF dan M1FF:

Tabel 3.4 Proses *Decoding* MTF

Kode MTF	Daftar simbol MTF				Kode asli
	0	1	2	3	
3302003	A	K	S	**R**	R
302003	R	A	K	**S**	RS
02003	**S**	R	A	K	RSS
2003	S	R	**A**	K	RSSA
003	**A**	S	R	K	RSSAA
03	**A**	S	R	K	RSSAAA
3	A	S	R	**K**	RSSAAAK

Tabel 3.5 Proses *Decoding* M1FF

Kode M1FF	Daftar simbol M1FF				Kode asli
	0	1	2	3	
3311003	A	K	S	**R**	R
311003	A	R	K	**S**	RS
11003	A	**S**	R	K	RSS
1003	S	**A**	R	K	RSSA
003	**A**	S	R	K	RSSAA
03	**A**	S	R	K	RSSAAA
3	A	S	R	**K**	RSSAAAK

3. *Run-Length Encoding* (RLE)

RLE merupakan teknik kompresi klasik yang sering digunakan atau dikombinasikan dengan teknik kompresi data lainnya dalam mengompresi data. Ide dari teknik ini yaitu mengkodekan perulangan string dengan dua buah simbol yaitu panjang dari string dan simbol dari string tersebut ataupun sebaliknya. Sebagai

contoh string “abbaaaaabaabbbaa” dengan panjang 16 *byte* apabila dikodekan dengan RLE akan menjadi “1a2b5a1b2a3b2a” atau “a1b2a5b1a2b3a2” dengan panjang 14 *byte* (Shanmugasundaram & Lourdusamy, 2011).

Dalam publikasinya, Burrows & Wheeler (1994) mengusulkan pendekatan RLE yang berbeda yang disebut RLE0. RLE0 bekerja dengan menjumlahkan simbol selain 0 dengan 1 dan kemudian mengkodekan perulangan dari nol (Burrows & Wheeler, 1994).

4. *Entropy Encoding* (EC)

Tahap akhir dalam BWCA yaitu EC yang mengompresi *stream* hasil output RLE menjadi lebih kecil dengan menggunakan metode statistik. Dua algoritma yang umumnya digunakan dalam tahapan ini yaitu pengkodean Huffman, dan pengkodean aritmatika. Diantara kedua metode ini, pengkodean aritmatika merupakan metode yang paling optimal, namun pengkodean aritmatika lebih lambat dibandingkan dengan pengkodean Huffman, karena pengkodean aritmatika menggunakan operasi pembagian dan perkalian (Sasilal & Govindan, 2013).

Proses *encoding* dengan menggunakan pengkodean aritmatika dapat dilihat pada *pseudecode* berikut:

a. Set low = 0.0 dan set high = 1.0
b. Baca simbol input
c. CR = high – low
d. High = low + (CR * high)
e. Low = low + (CR * low)
f. Ulangi langkah b sampai e hingga semua simbol terbaca
g. Cetak Low

Proses *decoding* dengan menggunakan pengkodean aritmatika dapat dilihat pada *pseudecode* berikut:

a. Baca simbol kode (S)
b. Baca *range* dari simbol yang bernilai S
c. Cetak simbol
d. CR = high – low
e. S = (S – low)/CR
f. Ulangi langkah b sampai e hingga semua code terbaca

C. J-Bit *Encoding* (JBE)

JBE merupakan algoritma kompresi data yang memanipulasi setiap bit data dalam file untuk meminimalkan ukuran tanpa kehilangan data apapun setelah proses *decoding* sehingga algoritma ini diklasifikasikan sebagai algoritma kompresi *lossless* (Sadiq, et al., 2013; Grewal & Kaur, 2014; Sharma, et al., 2014).

Langkah-langkah proses kompresi dan dekompresi dari JBE dapat dilihat pada *pseudocode* berikut (Suarjaya, 2012):

1. Proses Kompresi

a. Baca masukan per *byte*
b. Jika *byte* yang dibaca adalah nol maka tulis bit ‘0’ ke dalam *temporary byte* data. Jika *byte* yang dibaca bukan nol maka tulis *byte* tersebut pada data I dan tulis bit ‘1’ ke dalam *temporary byte* data.
c. Ulangi langkah a dan b hingga *temporary byte* data terisi dengan 8 bit.
d. Jika *temporary byte* data telah terisi dengan 8 bit maka tulis nilai *byte* dari *temporary byte* data tersebut ke dalam data II.

e. Hapus *temporary byte* data.

f. Ulangi langkah a sampai e hingga akhir file dicapai.

g. Tulis kombinasi data I dan II

1) Tulis panjang input asli.

2) Tulis data I.

3) Tulis data II.

h. Jika diikuti oleh algoritma kompresi lain, data I dan data II dapat dikompresi secara terpisah sebelum digabungkan (opsional).

2. Proses Dekompresi:

a. Baca panjang input asli.

b. Jika dikompres secara terpisah, dekompresi data I dan data II (opsional).

c. Baca Data II per bit.

d. Jika bit yang dibaca adalah '1' maka baca data I dan tulis ke output, sebaliknya jika bit yang dibaca adalah '0' maka tulis *byte* nol ke output.

e. Ulangi langkah c dan d hingga panjang input data asli dicapai.