Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

Team project ©2017
Dony Pratidana S. Hum | Bima Agus Setyawan S. IIP

BAB II

LANDASAN TEORI

2.1 File

File merupakan koleksi data yang terdapat dalam media seperti *harddisk*, disket, CD (*Compact Disc*), atau USB *flash drive*. Berdasarkan bentuknya, *file* dapat berupa gambar, teks, atau suara, dimana nama dari masing-masing *file* haruslah unik. Secara otomatis, komputer akan menandai setiap jenis *file* tertentu yang disebut ekstensi (tiga huruf di belakang nama *file*) (Maryono, 2007). Setiap *file* tersebut tentunya memiliki ukuran yang berbeda-beda dari yang terkecil dalam satuan byte (1 *byte* = 8 *bits*) hingga ukuran yang terus berkembang hingga terabyte (1 *terabyte* = 240 *bytes*) maupun *petabyte* (1 *petabyte* = 250 *bytes*), dan saat ini *byte* digunakan sebagai dasar dari pengukuran *memory* walaupun masih ada yang menggunakan *nibble* (1 *nibble* = 4 *bits*) (Sammes, 2007).

2.2 Kompresi

Kompresi data adalah sebuah cara yang dilalui untuk memadatkan data, dimana data tersebut nantinya akan lebih kecil dari data sebelum dipadatkan sehingga dapat mengefisienkan tempat penyimpanan data. David Salomon (2007:2) mengatakan bahwa data kompresi adalah proses mengubah sebuah *input* data *stream* (*stream* sumber atau data mentah asli) menjadi data *stream* lainnya (*bitstream* hasil, atau *stream* yang telah terkompresi) yang berukuran lebih kecil dari sebelumnya.

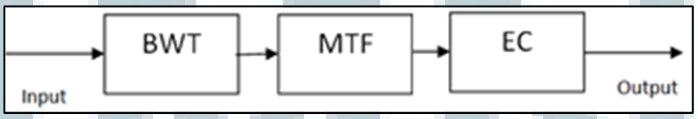
Saat ini ada dua alasan utama mengapa kompresi data banyak digunakan (Salomon, 2007):

1. Saat ini orang-orang suka mengumpulkan data, namun tidak peduli seberapa besar media penyimpanan, suatu saat akan terjadi *overflow* sehingga butuh adanya kompresi data.
2. Orang-orang benci menunggu lama untuk memindahkan data sehingga dengan demikian kompresi data diperlukan untuk meminimalisasi waktu distribusi data.

2.3 Burrows Wheeler

Algoritma Burrows-Wheeler dipublikasikan pada tahun 1994 oleh Michael Burrows dan David Wheeler dalam sebuah laporan penelitian yang berjudul "A Block-Sorting Lossless Data Compression Algorithm". Makalah ini membahas algoritma menyeluruh mengenai kompresi dan dekompresi, dan inti yang terkandung dalam makalah ini adalah pembahasan mengenai metode *Burrows-Wheeler Transform* (BWT) (Nelson, 1996).

Algoritma Burrows-Wheeler mempunyai struktur utama sebagai berikut (Schiller, 2012:1):



Gambar 2.1 Struktur Algoritma Burrows-Wheeler

Tahap pertama *Burrows-Wheeler Transformation* (BWT) berguna untuk mengurutkan data agar data yang identik dapat berada dekat satu sama lain. Sementara pada tahap kedua *Move to Front* (MTF) data belum terkompresi sama

sekali, namun dengan adanya tahapan ini maka data yang tadinya tidak beraturan dapat tersusun dengan rapi sehingga ketika dikompresi akan menghasilkan data kompresi yang lebih efisien.

Tahap terakhir adalah *Entropy Coding*, pada tahap ini kompresi yang sebenarnya akan dimulai, pada umumnya algoritma yang akan digunakan untuk kompresi data adalah *Huffman Coding* atau *Arithmetic Coding*. Pada sistem yang dibuat ini nantinya akan menggunakan *Huffman Coding.*

2.3.1 Burrows Wheeler Transform

Menurut M.Burrows dan D.J. Wheeler (1994), cara kerja BWT ini tidak memproses data masukan secara antrian, tapi memproses satu blok data teks sebagai satuan. Menciptakan suatu bentuk baru yang cenderung mengelompokkan karakter yang sama secara berurutan (memiliki peluang yang lebih tinggi). Transformasi ini memiliki sifat *reversible* yaitu dapat dikembalikan ke bentuk asal.

Tujuan dari BWT ini adalah mengurutkan *input* karakter yang identik berurutan. Tahapan proses BWT (Schiller, 2012: 2):

1. Urutkan *input* n (n adalah panjang dari *input*) dikali dengan n sendiri, setelah itu rotasi setiap baris sebanyak satu karakter ke sebelah kanan dibandingkan dengan baris sebelumnya
2. Urutkan semua baris-baris secara *lexicographical* atau diurutkan seperti urutan yang ada pada kamus bahasa. *Output* dari tahap ini nantinya L-kolom (L-kolom digunakan untuk kolom terakhir dan F-Kolom untuk kolom yang

pertama dari *matrix* yang telah diurutkan) dan indeks yang mengandung kata asal disimpan.

Sebagai contoh akan dijelaskan misalkan *input* kata “BANANA”. Pertama diketahui *string* S = “BANANA”, jumlah panjang n = 6, setelah itu dibuat *matrix* yang berukuran n x n (6x6). Setelah itu lakukan rotasi setiap baris sebanyak satu karakter ke sebelah kiri.

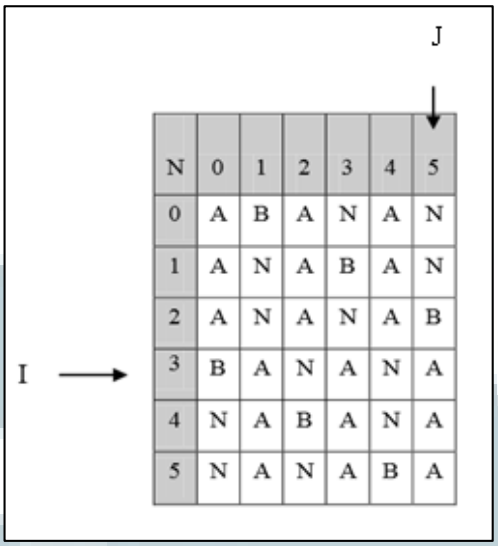
N	0	1	2	3	4	5
0	B	A	N	A	N	A
1	A	N	A	N	A	B
2	N	A	N	A	B	A
3	A	N	A	B	A	N
4	N	A	B	A	N	A
5	A	B	A	N	A	N

Pengurutan →

N	0	1	2	3	4	5
0	A	B	A	N	A	N
1	A	N	A	B	A	N
2	A	N	A	N	A	B
3	B	A	N	A	N	A
4	N	A	B	A	N	A
5	N	A	N	A	B	A

Gambar 2.2 Pengurutan secara *lexicographical.*

Dari matriks hasil di atas, maka *string* L diambil dari kolom terakhir matriks M yang terbentuk, yaitu karakter-karakter L[0]…L[N-1] sama dengan M[0, N-1]… M[N-1, N-1] . Setelah itu ambil indeks yang berisikan kata *original* “BANANA” setelah tabel diurutkan secara *lexicographical*. Sehingga nantinya akan diperoleh hasil yaitu L= “NNBAAA” dengan indeks I=”3” dari kata asli “BANANA”



Gambar 2.3 Hasil pada kolom terakhir matriks serta indeks kata asli.

2.3.2 Move to Front

Move to front (MTF) adalah sebuah algoritma transformasi data yang didesain untuk meningkatkan performansi dari teknik kompresi *entropy encoding*. Ide utama dari algoritma ini adalah mengganti setiap simbol dengan sebuah indeks angka dari kumpulan simbol yang sering atau baru saja digunakan. Simbol yang bermunculan secara berurutan dalam jumlah besar dan identik akan menghasilkan banyak angka nol yang mana akan menguntungkan teknik kompresi *entropy encoding* seperti huffman *coding*. Berikut adalah tahapan-tahapan pada MTF (Schiller, 2012:3):

1. Simpan indeks yang mengandung huruf pertama *input*
2. Geser huruf yang telah disimpan sebelumnya ke indeks 0 dan sisanya digeser ke sebelah kanan
3. Pada tahap ke 3 ulangi tahap 1 dan 2 untuk huruf-huruf lainnya dengan urutan sesuai dengan *input*.

Berikut contoh berdasarkan *input* BWT sebelumnya “NNBAAA”

Tahap 1:

Input : NNBAAA

Y= [A, B, N] simpan indeks 2

Tahap 2:

Y = [A, B, N]

Y = [N, A, B]

Tahap 3:

Y = [N, A, B], simpan indeks 0 Y = [N, A, B]

Y = [N, A, B], simpan indeks 2 Y = [B, N, A]

Y = [B, N, A], simpan indeks 2 Y = [A, B, N]

Y = [A, B, N], simpan indeks 0 Y = [A, B, N]

Y = [A, B, N], simpan indeks 0 Y = [A, B, N]

Berdasarkan inputan sebelumnya yaitu "NNBAAA" maka akan menghasilkan *output* "202200".

2.3.3 Huffman Coding

Algoritma Huffman ditemukan oleh David Huffman pada tahun 1952. Algoritma ini menggunakan pengkodean yang mirip dengan kode Morse. Berdasarkan tipe kode yang digunakan, algoritma Huffman termasuk metode *statistic*. Sedangkan berdasarkan teknik pengkodeannya menggunakan metode *symbolwise*. Algoritma Huffman merupakan salah satu algoritma yang digunakan untuk mengompres teks. Algoritma Huffman secara lengkap (Munir Rinaldi, 2003)

1. Contoh *input* "30331". Pilih dua simbol dengan peluang (*probability*) paling kecil (pada contoh di atas simbol 1 dan 0). Kedua simbol tadi dikombinasikan

sebagai simpul *parent node* simbol 1 dan 0. Sehingga menjadi simbol 10 dengan peluang 1/5 + 1/5 = 2/5, yaitu jumlah peluang dua *child nodes*.

2. Selanjutnya, pilih dua simbol berikutnya, termasuk simbol baru, yang mempunyai peluang terkecil.
3. Ulangi langkah 1 dan 2 sampai seluruh simbol habis. Sebagai contoh, dalam kode ASCII *string* tujuh huruf "30331" membutuhkan representasi 5 × 8 bit = 40 bit (7 *byte*), dengan rincian sebagai berikut:

00110011	00110000	00110011	00110011	00110001
3	0	3	3	1

Untuk mengurangi jumlah bit yang dibutuhkan, panjang kode untuk tiap karakter dapat dipersingkat, terutama untuk karakter yang frekuensi kemunculannya besar. Pada *string* di atas, frekuensi kemunculan 3 = 3, 0 = 1, dan 1 = 1 sehingga dengan menggunakan algoritma di atas diperoleh kode Huffman seperti pada tabel di bawah.

Tabel 2.1 Tabel kode Huffman.

Karakter	Frekuensi	Peluang	Huffman
3	3	3/5	0
1	1	1/5	10
0	1	1/5	11

Dengan menggunakan kode Huffman ini, *string* "30331" direpresentasikan menjadi rangkaian bit: 0 11 0 0 10. Jadi, jumlah *bit* yang dibutuhkan hanya tujuh *bit* dari yang seharusnya dibutuhkan empat puluh *bit*.

2.3.4 Decompression Huffman Coding

Untuk menguraikan kembali data yang sudah dikodekan sebelumnya dengan algoritma Huffman, dapat digunakan cara sebagai berikut:

1. Baca bit pertama dari *string* biner masukan
2. Lakukan transversal pada pohon Huffman mulai dari akar sesuai dengan *bit* yang dibaca. Jika *bit* yang dibaca adalah 0 maka baca anak kiri, tetapi jika *bit* yang dibaca adalah 1 maka baca anak kanan.
3. Jika anak dari pohon bukan daun (simpul tanpa anak), maka baca *bit* berikutnya dari *string* biner masukan.
4. Hal ini diulang (*traversal*) hingga ditemukan daun.
5. Pada daun tersebut simbol ditemukan dan proses penguraian kode selesai
6. Proses penguraian kode ini dilakukan

Pada *input* di atas yang merupakan "0 11 0 0 10" dikembalikan lagi menjadi "30331" dengan bantuan tabel yang telah dibuat sebelumnya.

2.2.5 Move to Front Backtransformation

Pada tahap ini, hasil dari dekompresi tahap sebelumnya akan ditranformasi kembali ke bentuk asal. Beberapa cara yang dilakukan pada tahap ini adalah (Schiller, 2012: 6):

1. Gunakan patokan indeks huruf sebelumnya yaitu Y=[A, B, N] dan digunakan *input* sebelumnya yaitu "202200"
2. Simpan indeks yang mengandung huruf pertama *input*

3. Geser huruf yang telah disimpan sebelumnya ke indeks 0 dan sisanya digeser ke sebelah kanan

4. Pada tahap ke 4 ulangi tahap 2 dan 3 untuk huruf-huruf lainnya dengan urutan sesuai dengan *input*.

   Nantinya dari *input* “202200” akan menghasilkan huruf “NNBAAA”.

2.2.6 Burrows-Wheeler Backtransformation

Pada tahap ini data akan dikembalikan ke bentuk semula. Beberapa cara yang dilakukan pada tahap ini adalah (Schiller, 2012:7):

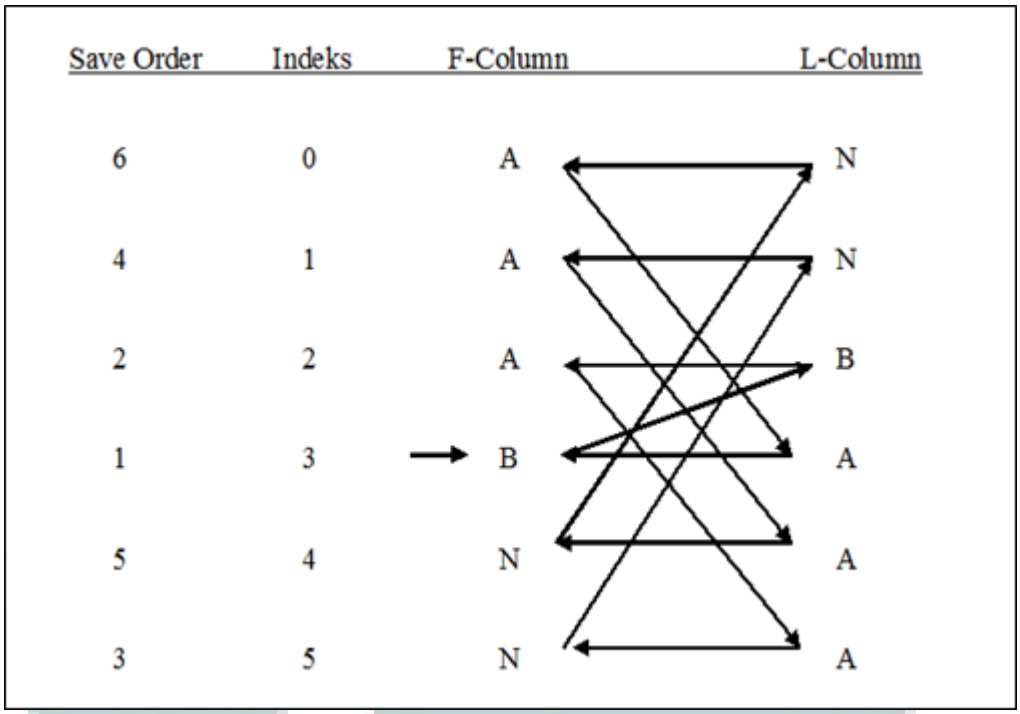
1. Urutkan kolom L (*input* terakhir) sejajar dengan kolom F (pertama)

2. Gunakan indeks yang telah disimpan sebelumnya (menyimpan kata yang benar).

3. Carilah indeks I dimana I adalah nilai indeks dari huruf pada kolom L yang identik dengan urutan pada kolom F.

4. Simpan huruf yang tertuju yang terletak pada kolom F pada nilai indeks i

5. Ulangi tahap 3 dan 4 sampai *value* I sama dengan indeks *input*.



Gambar 2.4 Proses Burrows-Wheeler Backtransformation

Pada gambar 2.4 dapat dilihat kolom L (*input* terakhir) sejajar dengan kolom F (pertama), lalu dengan indeks yang telah disimpan sebelumnya didapatkan huruf B pada kolom F (pertama). Setelah itu cari karakter yang identik dari huruf pada kolom L yang identik dengan urutan pada kolom F. Setelah mendapatkan huruf pada kolom L, selanjutnya didapatkan huruf berikutnya yang tertuju pada kolom F berdasarkan indeks yang sama. Lalu indeks urutan pencarian huruf disimpan pada kolom *Save Order*.

2.4 Dropbox

Dropbox adalah sebuah solusi sederhana untuk kepentingan akses dan *sharing* dokumen melalui layanan *cloud* yang dikembangkan sejak tahun 2007 oleh dua orang mahasiswa MIT saat itu yaitu Drew Houston dan Arash Ferdowsi. Layanan ini mempermudah pengguna untuk menyimpan *file* secara *online* antar

komputer (Prayudi, 2011:109). Dalam situsnya, disebutkan bahwa Dropbox adalah servis gratis yang memungkinkan untuk membawa foto, *docs*, dan video dimana saja dan dapat dibagikan dengan mudah (Dropbox, 2014). Dropbox merupakan suatu aplikasi gratis yang terdapat pada sistem operasi *window*s, mac, dan linux. Dropbox dapat secara langsung mem*backup* dan melakukan sinkronisasi *file* yang berada di internet ke semua komputer yang ter-*install* dropbox. Ketika pertama kali membuat akun dan menjalankan aplikasi, dropbox akan membuat suatu *folder* dalam *hard drive*. Dropbox secara *background* dan konstan akan memonitor *folder* tersebut, sehingga setiap *file* yang disimpan dalam *folder* tersebut akan disinkronisasikan dengan komputer-komputer lain yang meng*install* dropbox. Setiap *file* yang mengalami perubahan ataupun bertambah akan melakukan *backup* serta sinkronisasi. Dropbox juga menyediakan *file revision history*, hal ini untuk menjaga *file* yang tidak sengaja terhapus atau berubah dapat dikembalikan ke versi sebelumnya (Pash, 2011).