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

TINJAUAN PUSTAKA

2.1 Dropbox

Dalam situs dropbox, dituliskan dropbox adalah sebuah servise gratis yang memungkinkan semua foto,dokumen,video terdapat dimana saja. Semua file yang disimpan di dropbox akan otomatis tersimpan disemua komputer, phone, dan website dropbox (dropbox,2011). Dropbox adalah aplikasi gratis untuk windows, mac dan linux. Dropbox secara langsung membackup dan melakukan sinkronisasi file yang berada di internet ke semua komputer yang menginstall dropbox. Ketika pertama kali membuat account pada dropbox dan menjalankan aplikasi, Dropbox akan membuat folder pada hard drive. Aplikasi ini berjalan di background, secara konstan memonitor dropbox folder. Setiap file yang ditaruh di dropbox folder secara otomatis akan disinkronisasi dengan komputer yang telah terinstal dropbox. Setiap file berubah atau bertambah, dropbox akan melakukan back up dan sinkronisasi. Dropbox juga menyediakan *file revision history*, hal ini untuk menjaga file yang tidak sengaja terhapus atau berubah dapat dikembalikan lagi (Pash,2011).

2.2 File

Dituliskan dalam *Computer Science*, *File* adalah sebuah koleksi data yang tersimpan. *File* yang terlalu besar untuk disimpan di *main storage* akan disimpan

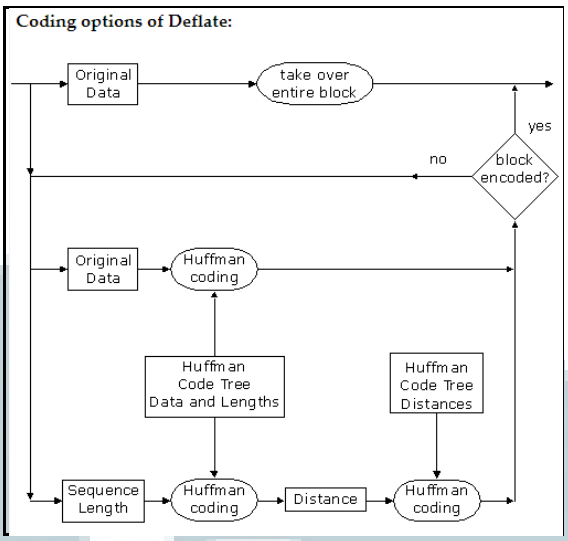
Implementasi Algoritma ..., Agi Reza Jasuma S, FTI UMN, 2013

pada tempat penyimpanan cadangan seperti *magnetic disks*. Ketika data pada sebuah *file* dibutuhkan, *file* dibaca dan dimasukkan ke dalam *main storage* dalam jumlah yang sesuai (French,1996).

Sedangkan dalam buku new perspectives on computer concepts, *file* adalah koleksi data yang terdapat dalam media penyimpanan, seperti *hard disk*, cd, dvd, atau USB *flash drive*. Sebuah *file* bisa berisi dokumen, foto, musik, video, surat elektronik dan program komputer (Parsons,2012).

2.3 Deflate

Salomon menuliskan Deflate adalah metode kompresi yang populer dan dipakai oleh Zip dan GZip bahkan diadopsi oleh banyak aplikasi. Metode Deflate di desain oleh Philip Katz sebagai bagian dari zip *file* format. ZIP format dan metode Deflate merupakan public domain, sehingga implementasinya dapat muncul di berbagai platform (Salmon, 2007). Metode Deflate ini menggabungkan dua algoritma. Algoritma pertama LZ77, algoritma ini digunakan untuk meng*encode* urutan yang berulang. Algoritma kedua Huffman code, algortitma ini digunakan untuk entropy coding (BinaryEssence, 2000). Dibawah ini merupakan flow diagram dari algoritma deflate.



Gambar 2.1 Deflate Flow Diagram (BinaryEssence, 2000)

Secara teori Deflate hanya menyediakan tiga cara untuk meng*encode* data. Cara pertama adalah *Uncompressed original data*. Data dari semua *block* diambil dan tidak diubah hanya dengan persyaratan minimum untuk informasi *header*. Cara kedua adalah *Huffman encoded original data.* Simbol *original* byte langsung di*encode* berdasarkan Huffman tree. Cara ketiga adalah *Reference to a previous sequence (LZ77).* Cara ini menggunakan algoritma LZ77 untuk menghasilkan pasangan "panjang *sequence*" dan "jarak". Kedua parameter ini kemudian di*encode* dengan menggunakan algoritma Huffman code dan menghasilkan *code* panjang dan jarak (BinaryEssence, 2000).

Dalam buku Handbook of Data Compression, tiga cara yang sebelumnya dijelaskan diberi nama *No Compression*, *Compression with Fixed Code Tables*, dan *Compression with Individual Code Tables*. Pada Deflate, setiap block dimulai dengan tiga bit *header.* Bit pertama digunakan sebagai penanda block terakhir dan dua bit

yang lain digunakan untuk menentukan mode kompresi. Format pada mode pertama yaitu tiga bit header 000 atau 100 lalu diikuti dengan empat bytes yang berisi LEN. LEN disini adalah jumlah bytes dalam block, sehingga hanya dibatasi sampai 65.535 bytes. Terakhir yaitu data. Format pada mode dua dimulai dengan tiga bit header 001 atau 101. Dilanjutkan dengan data yang sudah dikomrpes. Terakhir *code* 256 yang menandakan akhir dari block.

Pada *compression with fixed code table*, dua tabel yang sudah ditentukan, dimasukkan ke dalam Deflate *encoder* dan *decoder*. Dengan cara ini dapat mempercepat kompresi dan dekompresi, dan tabel kompresi tidak perlu diikutkan dalam data. Terdapat kemungkinan performa dari kompresi tidak baik dikarenakan data secara statistik tidak sama dengan tabel. *Literal* dan *match length* pada tabel disebut sebagai "edoc". Setiap edoc akan diubah menjadi prefix yang nantinya dijadikan hasil kompresi. Tabel edoc untuk literal dimulai dari nol hingga 255, edoc 256 dijadikan sebagai end of block, dan 257 hingga 285 untuk *length*. Dikarenakan 29 edoc tidak akan cukup untuk menyimpan *length* dari tiga hingga 258, maka ditambahkan extrabit. Untuk menyimpan jarak digunakan 30 *codes*. Code ini hanya terdiri dari lima bit saja dengan extrabit. Ukuran maksimal *search buffer* pada deflate adalah 32,768. Berikut merupakan tabel *literal* atau *length* dan *distance* yang digunakan pada kompresi ini.

Code	Extra bits	Lengths	Code	Extra bits	Lengths	Code	Extra bits	Lengths
257	0	3	267	1	15,16	277	4	67–82
258	0	4	268	1	17,18	278	4	83–98
259	0	5	269	2	19–22	279	4	99–114
260	0	6	270	2	23–26	280	4	115–130
261	0	7	271	2	27–30	281	5	131–162
262	0	8	272	2	31–34	282	5	163–194
263	0	9	273	3	35–42	283	5	195–226
264	0	10	274	3	43–50	284	5	227–257
265	1	11,12	275	3	51–58	285	0	258
266	1	13,14	276	3	59–66			

Table 6.38: Literal/Length Edocs for Mode 2.

edoc	Bits	Prefix codes
0–143	8	00110000–10111111
144–255	9	110010000–111111111
256–279	7	0000000–0010111
280–287	8	11000000–11000111

Gambar 2.2 Tabel Literal dan Length (Salomon, 2010)

Code	Extra bits	Distance	Code	Extra bits	Distance	Code	Extra bits	Distance
0	0	1	10	4	33–48	20	9	1025–1536
1	0	2	11	4	49–64	21	9	1537–2048
2	0	3	12	5	65–96	22	10	2049–3072
3	0	4	13	5	97–128	23	10	3073–4096
4	1	5,6	14	6	129–192	24	11	4097–6144
5	1	7,8	15	6	193–256	25	11	6145–8192
6	2	9–12	16	7	257–384	26	12	8193–12288
7	2	13–16	17	7	385–512	27	12	12289–16384
8	3	17–24	18	8	513–768	28	13	16385–24576
9	3	25–32	19	8	769–1024	29	13	24577–32768

Gambar 2.3 Tabel Prefix pada Distance (Salomon, 2010)

Pada *Compression with Individual Code Table*, encoder akan mengambil statistik pada data dan menggunakannya untuk menghasilkan tabel yang lebih baik. Kompresi ini hampir sama seperti pada *fixed table*, hanya saja tabel yang dihasilkan pada kompresi ini harus diikutkan pada data (Salomon,2010).

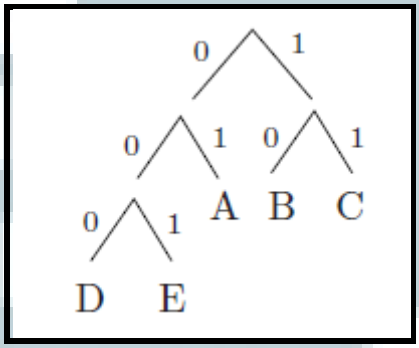
2.4 LZ77

LZ77 secara teori menggunakan *input stream* yang sudah diproses menjadi kamus. *Encoder* mengatur *window* pada *input stream* dan menggeser *input* dalam *window* tersebut dari kanan ke kiri sambil melakukan *encode*. Metode ini bekerja berdasarkan *sliding window*. *Window* pada algoritma ini dibagi menjadi dua bagian. Bagian pertama yang terdapat di sebelah kiri *window* disebut *search buffer*. Bagian ini merupakan kamus, dan terdiri dari input yang telah melalui tahap encode sebelumnya. Bagian ke dua yang terdapat di sebelah kanan *window* disebut *look-ahead buffer*. Bagian ini terdiri dari text yang akan di*encode* (Salomon, 2010). Dalam bukunya salomon menuliskan, *encoder* melakukan *scan* pada *search buffer* secara terbalik (kanan ke kiri) mencari simbol yang sama dengan simbol pertama pada *look-ahead buffer*. Ketika menemukan kesamaan pada simbol pertama, *encoder* akan terus mencari kesamaan yang lebih panjang dari kesamaan yang sebelumnya. Setelah menemukan kesamaan yang paling panjang maka encoder akan menghasilkan output yaitu posisi posisi awal simbol yang sama pada kamus, panjang simbol yang sama dan simbol selanjutnya setelah simbol yang sama pada *look-ahead buffer* (Salomon, 2010).

2.5 Huffman Code

Dituliskan bahwa algoritma Huffman dimulai dari me*list* semua simbol sercara *descending* berdasarkan probabilitasnya. Kemudian berdasarkan *list* tersebut, dibuatlah *tree* dengan simbol pada setiap *leaf*nya, mulai dari bawah hingga ke atas.

Tree dibuat dengan menggabungkan dua simbol yang memiliki probabilitas terkecil. Setelah kedua simbol ini digabungkan, terciptalah *node* yang baru berisi probabilitas dari penjumlahan probabilitas kedua simbol yang digabungkan. Kemudian node ini akan dimasukkan ke dalam *list* dan di *sorting* secara *descending* berdasarkan probabilitasnya. Proses ini akan berulang terus-menerus hingga hanya terdapat satu node saja. Node ini lah yang kemudian akan menjadi root dari *tree*. Setelah *tree* dibentuk, setiap simbol akan diberikan code yang baru. Code dibuat dengan cara memberikan nilai 0 pada bagian kiri dari node dan 1 pada bagian kanan dari node (Salomon, 2010). Gambar 2.4 menunjukkan *tree* yang dihasilkan oleh Huffman Code.



Gambar 2.4 Huffman Tree (Salomon, 2010)

2.6 Huffman Canonical

Dalam buku "Data Compression: The Complete Reference" oleh David Solomon, dituliskan bahwa kata "canonical" berarti *code* tertentu dipilih dari sekian banyak *Huffman code* yang ada karena code ini sangat mudah digunakan dan cepat. Algoritma ini efektif ketika terdapat banyak *alphabet* yang harus di *code*kan dan membutuhkan waktu *decoding* yang sangat cepat. Dikarenakan cara *code* dibuat,

mudah bagi *decoder* untuk menentukan panjang *code* dengan cara membaca input *bit* satu per satu. Berikut adalah *pseudocode* yang digunakan untuk men*decode*:

```
l:= 1; input v;
while v <  first[l]
        append next input bit to v;
        l:= l + 1;
endwhile
```

Sebagai contoh jika *code* adalah 00110. Dalam contoh ini *bit* digunakan sebagai inputan dan sambungkan ke *v*. Ketika *pseudocode* dijalankan, hasil yang didapat 0, 00 = 0, 001 = 1, 0011 = 3, 00110 = 6 ketika *l* ditambah dari 1 to 5. Semua langkah memenuhi persyaratan $v < first[l]$, kecuali langkah terakhir (Salomon, 2007).

2.7 Length-Limited Huffman

Sayood menuliskan dalam bukunya bahwa, algoritman *Huffman code* mencoba mengecilkan rata-rata panjang *codewords.* Bagaimanapun, tidak ada batasan untuk *codewords* dari setiap *code*. Hal ini tidak menjadi masalah ketika *alphabet* yang di*code*kan sedikit, tetapi akan menjadi masalah ketika jumlah *alphabet* yang di*code*kan banyak. Dengan jumlah *alphabet* yang banyak ada kemungkinan didapatkan *Huffman code* yang sangat panjang, sehingga dalam proses *coding* menjadi tidak efisien. Sebagai contoh, apabila terdapat *alphabet* $A = [a_1, a_2, a_3, a_4, a_5, a_6]$ dengan probabilitas $P(a_1) = 0.05$, $P(a_2) = 0.1$, $P(a_3) = 0.15$, $P(a_4) = 0.2$, $P(a_5) = 0.2$ dan $P(a_6) = 0.3$. Proses dimulai dengan *list alphabet* yang telah diurutkan berdasarkan probabilitas setiap *alphabet* dimana $p_1 \leq p_2 \leq \ldots \leq p_m$. Pada tahap ini,

dibuat paket pertama. Paket akan dinotasikan sebagai $a_{jkl}(p)$, dimana *subscript* mengindikasikan angka pada *alphabet* di paket dan kuantitas mengindikasikan total probabilitas. Pada paket pertama, dilakukan penggabungan *alphabet* dua dua, sehingga menghasilkan paket seperti dibawah ini:

Paket$_1$: [a_{12}(0.15), a_{34}(0.35), a_{56}(0.5)]

Langkah selanjutnya dilakukan penggabungan paket pertama dengan list original, sehingga didapatkan hasil:

Merge$_1$: [a_1(0.05), a_2(0.1), a_3(0.15), a_{12}(0.15), a_4(0.2), a_5(0.2), a_6(0.3), a_{34}(0.35), a_{56}(0.5)]

Agar didapatkan hasil *code* terbatas hanya tiga *bit*, dibutuhkan sekali iterasi lagi. Pada tahap kedua, paket dibuat berdasarkan hasil *merge* sebelumnya. Sehingga didapatkan hasil paket seperti dibawah ini:

Paket$_2$: [a_{12}(0.15), a_{312}(0.3), a_{45}(0.4), a_{634}(0.65)]

Langkah selanjutnya dilakukan penggabungan paket kedua dengan *list* original, sehingga didapat hasil:

Merge$_2$: [a_1(0.05), a_2(0.1), a_3(0.15), a_{12}(0.15), a_4(0.2), a_5(0.2), a_6(0.3), a_{312}(0.3), a_{45}(0.4), a_{634}(0.65)]

Kemudian diambil sepuluh *item* (2 x 6 – 2 = 10) dengan probabilitas terkecil dan dihitung kemunculan *alphabet* dari sepuluh *item* tersebut. Aturan pengambilan item ini adalah 2 x *jumlah list awal* – 2. Berdasarkan aturan ini sehingga didapat panjang dari *codeword* seperti {3, 3, 3, 3, 2, 2} (Sayood, 2012).

2.8 .NET Framework

.Net *Framework* adalah *platform* baru yang dibuat oleh Microsoft untuk membuat suatu aplikasi. *Framework* ini dapat digunakan pada sistem operasi Windows. Dalam .Net *Framework* ada beberapa *library* yang berisikan *code* yang dapat langsung digunakan dalam bahasa seperti C#, juga dalam C++ , Visual basic, JScript, bahkan oleh bahasa yang sudah cukup lama yaitu COBOL (watson,2007). Sedangkan dalam buku .Net *Framework Essentials* dituliskan .Net *Framework* merupakan *framework* yang menyediakan *programming interface* baru untuk *windows service* dan API. *framework* ini terintegrasi dengan banyak teknologi microsoft. .Net *platform* terdiri dari empat produk yang terpisah.

Development tools and libraries

Terdiri dari beberapa bahasa pemrograman seperti c#,j# dan VB.NET, dan juga development tools termasuk Visual Studio .Net.

WebServices

menyediakan *web service* komersial yang dikhususkan untuk .NET *service*.

Specialized servers

kumpulan dari .NET *enterprise server*, termasuk SQL *server*, *exchange server*, BizTalk Server dan sebagainya. Ini berfungsi untuk relational *data storage*, *email*, dan B2B *commerce*.

Devices

.Net yang digunakan untuk alat non-PC seperti handphone atau game console (Thai,2003).