Penggunaan Pohon Huffman pada Algoritma Deflate

Nurul Fithria Lubis (13510012)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
nurulfithria@students.itb.ac.id

***Abstrak*—Proses kompresi adalah proses yang sangat penting dalam dunia komputer dan berbagai algoritma telah dirancang untuk mengatasi kasus ini. Salah satu dari algoritma tersebut adalah algoritma Deflate. Dalam perancangan algoirtma tersebut digunakan pohon Huffman dan dipadukan pula konsep eliminasi string kembar. Algoritma ini terbukti sangat berguna dilihat dari implementasinya pada banyak aspek di bidang komputer.**

***Kata kunci*—kompresi, Deflate, Huffman, pohon**

I. PENDAHULUAN

Dalam kehidupan sehari-hari, setiap orang tanpa sadar melakukan compressing untuk pemaikan ruang yang lebih sangkil dan mangkus. Misalnya, melipat kaca spion saat memarkir mobil di pinggir jalan, melipat baju agar lemari dapat memuat lebih banyak baju, juga menyingkat kata saat menulis pesan singkat agar karakter yang digunakan tidak lebih dari 160 karakter.

Dalam proses mengolahan dan penyimpanan data, peran proses kompresi bahkan relatif lebih signifikan dari itu. Tanpa proses kompresi, mungkin kita membutuhkan ratusan MB untuk menyimpan sebuah foto saja, sedang dengan proses kompresi, ukuran gambar tersebut mungkin dapat menjadi 100 kali lipat lebih kecil.

Seiring dengan munculnya algoritma-algoritma kompresi, muncul suatu masalah, yaitu beberapa proses kompresi tidak memungkinkan detail informasi pada suatu file untuk kembali sepenuhnya setelah proses encoding dan decoding. Untuk menjawab masalah ini, dirancang algoritma kompresi yang memungkinkan output encoding yang sama persis dengan input decoding, algoritma ini dinamakan *lossless compression algorithms*. Salah satu dari algoritma-algoritma ini ialah algoritma Deflate.

Pada dasarnya, algoritma Deflate menggabungkan konsep pohon Huffman dan eliminasi string kembar, kedua konsep tersebut sering digunakan untuk kebutuhan minimisasi bit data.

Algoritma Deflate telah dipatenkan sebagai US patent 5051745 atas nama Phil Katz, perancang algoritma tersebut. Algoritma ini dipakai pada program GZIP dan ZLIB, juga pada gambar dengan format PNG.

II. POHON HUFFMAN

Pohon Huffman melakukan kompresi data dengan cara menghitung frekuensi dari data-data yang lebih kecil. Data-data tersebut kemudian direpresentasikan dengan notasi biner.

Ambil sebagai contoh kode standar ASCII yang digunakan untuk merepresentasikan teks pada enkoder komputer.ASCII merepresentasikan tiap karakter pada teks menjadi sekuen dengan panjang tujuh bit. Hal ini berarti ada 2^7 atau 128 kemungkinan data yang dapat terbaca dari sebuah sekuen.

Kode standar ASCII termasuk kode dengan jenis fixed-length codes karena kode tersebut merepresentasikan tiap karakter dengan jumlah bit yang tetap.

Berbeda dengan, ambil sebagai contoh, kode Morse. Pada kode Morse, tiap karakter direpresentasikan dengan jumlah titik dan garis yang bervariasi sesuai dengan frekuensi dari masing-masing karakter. Huruf E sebagai huruf yang paling sering muncul direpresentasikan dengan sebuah titik saja. Karena karaktersitik ini, kode Morse termasuk tipe variable-length codes.

Secara general, jika data yang kita punya terdiri dari banyak sekuen dengan sebuah sekuen muncul lebih sering dari yang lain dan dengan suatu sekuen yang sangat jarang muncul, proses pengkodean dapat kita lakukan dengan lebih mangkus. Jika kode yang lebih singkat kita berikan untuk merepresentasikan sekuen yang lebih sering muncul, pemakaian bit di akhir akan lebih sedikit.

Misal kita memiliki string berikut:

```
ABFBAAABCHEAADACG
```

Pada string secara keseluruhan, terdapat huruf A sampai H, yaitu 8 huruf. Secara general, untuk merepresentasikan *n* karakter dengan simbol-simbol yang berbeda, kita membutuhkan $^2\log n$ bit per simbol. Berarti, pada kasus ini kita membutuhkan $^2\log 8$ atau 3 bit untuk setiap simbol. Setelah pengkodean, tiap karakter memiliki simbol unik sebagai berikut:

```
A   000
B   001
```

```
C    010
D    011
E    100
F    101
G    110
H    111
```

Berarti, string di atas dapat direpresentasikan dengan simbol menjadi sebagai berikut:

```
00000110100100000000000101011110000000001100
0010110
```

Data hasil pengkodean terdiri dari 53 bit.

Namun, setelah string yang kita miliki diamati lebih jauh, diketahui bahwa A adalah karakter yang paling sering muncul. Jadi, kita berikan kode dengan panjang 1bit untuk merepresentasikan A. Sehingga simbol perkarakter menjadi sebagai berikut:

```
A    0
B    100
C    1010
D    1011
E    1100
F    1101
G    1110
H    1111
```

Menggunakan simbol di atas, hasil konversi string menhasilkan deret notasi biner berikut:

```
01001101100000100101011111100001011010101110
```

Jumlah bit yang dibutuhkan untuk merepresentasikan string yang sama dengan pengkodean kedua adalah 41 bit, sekitar 20% lebih sedikit dibanding kebutuhan bit awal.

Meskipun demikian, terdapat beberapa kesulitan menyangkut penggunaan variable-length codes. Satu diantaranya ialah kesulitan dalam proses pembacaan, di mana akhir dari suatu simbol sulit untuk diketahui oleh pembaca data.

Untuk menyelesaikan masalah ini, dapat dirancang kode di mana akhiran simbol yang dikompresi tidak menjadi awalan simbol lainnya. Pada proses pengkodean di atas, A disimbolkan dengan 0 dan B dengan 100. Berarti, tidak boleh ada simbol lain yang dimulai dengan 0 maupun 100. Hal ini dapat mempersulit proses pengkodean.

A. Membentuk Pohon Huffman

Di sinilah pohon Huffman muncul sebagai alat yang ampuh dan sangkil. Pohon Huffman menguntungkan karena bekerja dengan konsep frekuensi relatif ari simbol-simbol. Kode Huffman dapat direpresentasikan dengan pohon biner yang daun-daunnya adalah simbol hasil pengkodean dan tiap pada simpul terdapat bobot yang merupakan jumlah dari bobot simbol-simbol yang terdapat dibawahnya

Sebagai contoh, mari kita selesaikan proses kompresi untuk representasi bit biner pada string yang kita miliki sebelumnya. Penyelesaian masalah kompresi dengan pohon Huffman dimulai dengan identifikasi masalah. Pada kasus ini, kita memiliki string dengan panjang 17 karakter dengan total 8 karakter unik. Setiap karakter unik ini memiliki bobot atau frekuensi kemunculan yang berbeda-beda, ditinjau dari kemunculan karakter itu sendiri dibanding dengan kemunculan karakter secara keseluruhan.

Jika dilakukan analisa lebih lanjut, kita dapatkan set yang berisi informasi mengenai karakter-karakter unik tersebut beserta bobot dari masing-masing karakter:

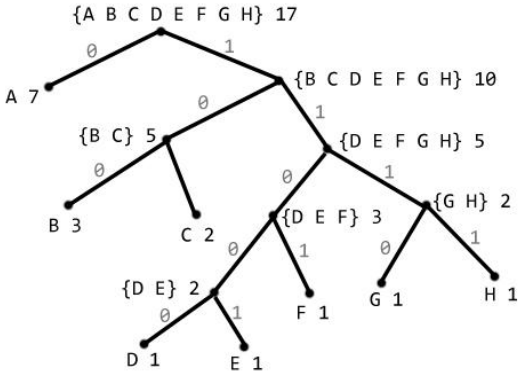
Karakter	Jumlah kemunculan	Frekuensi atau Bobot relatif
A	7	0.41
B	3	0.18
C	2	0.11
D	1	0.06
E	1	0.06
F	1	0.06
G	1	0.06
H	1	0.06

Dari set tersebut, perlu kita tentukan sebuah pohon Huffman sebagai pohon dengan jumalh lintasan minimum sebagai representasi kode prefiks yang akan menyatakan kode dari masing-masing karakter.

Dari set di atas, akan dibentuk suatu pohon Huffman. Pohon Huffman ini dibentuk dengan cara menempatkan set awal pada akar, beserta dengan kemunculkan total seluruh karakter. Kemudian dari akar, yang sejatinya adalah sebuah simpul, dibentuk 2 lintasan, tiap lintasan diberi kode 0 dan 1. Kode inilan yang pada akhirnya akan menjadi kode Huffman untuk tiap karakter.

Diujung lintasan kiri yang dilambangkan dengan 0, diletakkan sebuah atau dua buah karakter dengan kemunculan terbanyak. Pada kasus ini, kita letakkan sebuah karakter saja karena selisih kemunculan cukup signifikan. Pada ujung lintasan lainnya, yaitu lintasan kanan, diletakkan sisa set. Kemudian, dari simpul yang memuat sisa set, dilakukan hal serupa. Begitu seterusnya hingga seluruh karakter menjadi daun pada pohon Huffman.

Pada masalah yang akan kita selesaikan, dibentuk pohon Huffman sebagai berikut:



B. Pembacaan Pohon Huffman

Pembacaan kode Huffman akhir untuk tiap karakter dilakukan dari akar mengikuti lintasan untuk tiap karakter. Misalnya untuk karakter D, pembacaan kode dilakukan mengikut lintasan dari akar menuju daun D. Dari akar, diikuti lintasan kanan, kemudian kanan, kiri, kiri, dan kiri lagi, sehingga didapatkan kode 11000. Hal serupa dilakukan untuk tiap karakter.

Dari hasil pembacaan pohon diatas, didapatkan simbol untuk masing-masing karakter sebagai berikut:

Karakter	**Kode Huffman**
A	0
B	100
C	101
D	11000
E	11001
F	111
G	1110
H	1111

String yang kita miliki kemudian dituliskan dalam kode Huffman, menghasilkan deret bilangan dalam notasi biner sebagai berikut:

```
01001111000001001011111110010011000010111110
```

C. Menerjemahkan Kode Huffman

Hal penting lainnya adalah menerjemahkan kode Huffman menjadi string biasa. Untuk melakukan hal ini, kita lakukan pembacaan terhadap pohon Huffman untuk tiap kode yang akan diterjemahkan. Pembacaan ini dilakukan dari akar mengikuti urutan kode, baik 0 maupun 1, hingga mencapai daun.

Sebagai contoh, akan diterjemahkan kode Huffman di atas. Pembacaan dimulai dari 0, kemudian 1. Karena lintasan 0 berujung pada daun A dan simpul tersebut tidak mempunyai anak, berarti hasil terjemahan dari pembacaan pertama adalah A. Kemudian pembacaan dimulai kembali dari 1, kemudian 0, dan 0. Lagi, dari simpul ini tidak terdapat anak, berarti pembacaan berhenti sampai 0, menghasilkan 100, yang berarti karakter B, begitu seterusnya pembacaan dilakukan hingga akhir deret kode Huffman.

D. Pohon Huffman Minimum

Disamping algoritma Huffman yang cukup jelas, bukan tidak mungkin kita berakhir dengan pohon Huffman yang kurang sangkil. Untuk menentukan pohon Huffman yanf paling minimum, dapat kita lakukan minimisasi mulai dari daun-daun yang merupakan tiap karakter unik. Kemudian dilakukan penggabungan daun-daun membentuk simpul dimulai dari daun dengan bobot terendah.

Langkah pembentukan pohon Huffman minimum dapat dilihat pada tabel berikut:

Langkah	**Pohon**
Inisialisi	{(A 7) (B 3) (C 2) (D 1) (E 1) (F 1) (G 1) (H 1)}
1	{(A 7) (B 3) (C 2) ({D E} 2) (F 1) (G 1) (H 1)}
2	{(A 7) (B 3) (C 2) ({D E} 2) ({F G} 2) (H 1)}
3	{(A 7) (B 3) ({D E H} 3) (C 2) ({F G} 2)}
4	{(A 7) ({C F G} 4) (B 3) ({D E H} 3)}
5	{(A 7) ({B D E H} 6) ({C F G} 4)}
6	{(A 7) ({B D E H C F G} 10)}
7	{(A B D E H C F G) 17}

Dari ketujuh langkah di atas akan didapatkan pohon Huffman minimum. Algoritma yang sama dapat dilakukan untuk seluruh permasalahan kompresi menggunakan pohon Huffman.

III. ELIMINASI STRING KEMBAR

Konsep eliminasi string kembar dikenal dengan terma kompresi LZ77. Konsep minimisasi ini pada dasarnya sederhana. Kompresi LZ77 bekerja dengan cara mencari sekuen data yang berulang dengan menggunakan '*sliding window*' atau jendela luncur. Ukuran jendela luncur ini beragam, jendela dengan ukuran 32K berarti kompresor dan juga dekompresor merekam 32768 (32*1024) karakter terakhir yang terbaca.

Mari kita lihat cara kerja LZ77 pada contoh. Misal kita memiliki data karakter-karakter sebagai berikut:

```
Strukdis strukdis strukdis strukdis strukdis
```

Jendela luncur akan mulai membaca data dari karakter 'S', kemudian 't', 'r', 'u', 'k', 'd', 'i', 's', dan 's'. Namun, jika diperhatikan kecara menyeluruh, hasil bacaan 9 karakter berikutnya identik dengan hasil bacaan yang telah didapat:

```
Strukdis strukdis strukdis strukdis strukdis
```

Sekuen kedua identik dengan sekuen pertama dan ditemukan tepat 9 karakter setelah titik permulaan pembacaan.

Data sejauh ini yang telah dibaca:

```
Strukdis strukdis s
```

Setelah kompresi:

```
Strukdis s[D=9, L = 9]
```

Melihat sisa hasil bacaan yang sebenarnya juga identik dengan sekuen pertama dan kedua, dapat disimpulkan bahwa kompresi LZ77 dapat ditingkatkan lagi. Untuk melakukan hal tersebut, dibandingkan karakter yang

mengikuti awal sekuen. Pada seluruh kasus, karakter tersebut adalah ‘t’ sehingga panjang sekuen dapat diganti menjadi 10. Demikian pula pada karakter seterusnya, juga 9 karakter berikutnya. Analisis lebih lanjut menunjukkan bahwa 34 karakter dimulai dari karakter ke-10 identik dengan 34 karakter yang dimulai dari karakter ke-2. Dengan demikian, kompresi akan menghasilkan data berikut:

```
Strukdis s[D=9, L = 34]
```

IV. ALGORITMA DEFLATE

Algoritma Deflate adalah algoritma lossless data compression yang menggabungkan dua konsep yag telah kita bahas di atas. Algoritma ini banyak digunakan pada kompresi data dengan format seperti GZIP dan PNG.

Algoritma Deflate tergolong algoritma yang fleksibel dalam menentukan bagaimana proses kompresi dilakukan. Pemrogram tentu harus cerdas dalam menentukan algoritma yang digunakan untuk menyelesaikan masalah, namun kompresor pada algoritma Deflate sendiri memilihi pilihan atas bagaimana kompresi dilakukan.

Walau sebenarnya terdapat 4 mode dalam kompresi Deflate, namun 1 tidak digunakan, sehingga pada akhirnya hanya ada tiga mode kompresi pada kompresor Deflate yaitu:

1. Tidak dikompresi sama sekali. Mode ini cocok digunakan untuk data yang sudah melwati proses kompresi. Data yang dikamprosei menggunakan mode ini akan sedikit ‘membesar’, namun tidak sebanyak jika data tersebut sudah dikompresi dan kita coba kompresi lagi dengan mode lainnya. Kompresi dengan mode ini adalah kompresi yang paling mudah dilakukan karena pada dasarnya input dapat langsung diberikan ke output.
2. Kompresi, diawali dengan LZ77 dan diteruskan dengan pengkodean Huffman. Pohon yang digunakan pada mode ini ditentukan oleh spesifikasi Deflate itu sendiri. Pohon Huffman pada mode ini disebut juga pohon Huffman statik. Menggunakan pohon Huffman statik berarti tidak memerlukan ruang ekstra untuk menyimpan pohon tersebut. Namun dengan menggunakan pohon Huffman statik kemungkinan besar kompresi yang dilakukan tidaklah optimal.
3. Kompresi, diawali dengan LZ77 dan dilanjutkan dengan pengkodean Huffman. Perbedaan mode ini dengan mode pada poin 2 adalah, pada mode ini pohon Huffman yang digunakan dirancang dan dibuat oleh kompresor dan disimpan bersama data-data yang dikompresi. Pohon Huffman ini disebut juga pohon Huffman dinamik. Dengan menggunakan mode ini berarti kita perlu menentukan karakter yang digunakan untuk kompresi blok. Karakter ini kemudian disimpan sebelum data dan bersifat tetap dalam artian tidak dapat diubah lagi.
4. *Reserved*.

Pada kompresi Deflate, data input kompresi dipecahkan dalam blok-blok. Tiap blok membuat header 3 bit di mana bit pertama adalah penanda blok terakhir pada keseluruhan data, 1 menandakan blok tersebut adalah blok terakhir, dan 0 menandakan masih ada blok lain yang harus diproses setelah blok ini, dan 2 bit terakhir pada header untuk menetukan mode kompresi yang dipakai pada blok ini dengan 00 menandakan penggunaan mode 1, 01 penggunaan mode 2, 10 untk penggunaan mode 3, dan 11 penggunaan mode 4 namun mode ini seharusnya tidak ditemukan dan digunakan. Sebagian besar blok yang dikompresi Deflate menggunakan mode 3.

Jika ingin dilakukan pergantian mode, misalnya dari mode tanpa kompresi ke mode kompresi dengan pohon rancangan sendiri, atau dari mode kompresi dengan pohon spesifikasi Deflate ke mode tanpa kompresi, blok yang dalam proses kompresi harus diakhiri dan dimulai kompresi pada blok yang baru.

Bagaimana kedua konsep kompresi yang telah dibahas, yakni LZ77 dan Huffman bekerja bersama dalam sebuah algoritma akan kita amati dan analisa dengan lebih detail. Untuk melihat cara kerja ini, akan kita analogikan kode-kode untuk suatu data spesifik dengan nomor telepon, di mana sebuah sekuen nomor tertentu terhubung dengan sebuah pesawat telefon tertentu. Atas dasar analogi ini, awal pembacaan kode akan kita sebut dengan istilah nada hubung. Dalam sebuah nada hubung, terdapat tiga kemungkinan; karakter, pasangan panjang dan jarak, atau akhir dari blok.

Dalam pembacaan nada hubung, tentu kita harus dapat membedakan sebuah kode dengan lainnya. Untuk kepentingan ini, seluruh kemungkinan karakter, atau disebut literal, elemen yang merepresentsikan panjang yang mungkin, dan simbol akhir suatu blok, digabungkan ke satu karakter yang sama. Jarak tidak perlu diikutsertakan dalam hal ini karena jarak hanya mungkin muncul setelah panjang terbaca. Kemudian katakter inilah yang akan menjadi akar atau basis dari pohon Huffman yang akan digunakan dalam proses kompresi.

Saat suatu literal selesai diterjemahkan, kita akan berada di nada hubung baru lainnya dan pembacaan dimulai kembali. Jika terbaca sebuah simbol yang menyatakan akhir dari suatu blok, terdapat dua kemungkinan, yakni kita berada pada awal suatu blok lain, atau berada pada akhir data hasil kompresi.

Kode yang merepresentasikan panjang atau jarak dapat berupa suatu kode basis yang diikuti bit ekstra yang membentuk sebuah integer untuk ditambahkan pada nilai basis.

Sekarang mari kita lihat proses kompresi dengan Algoritma Deflate. Kompresi dimulai dengan elminasi string kembar. Pembacaan dan identifikasi string kembar dilakukan untuk tiap blok yang diproses. Saat ditemukan bit-bit kembar, dituliskan sebuah referensi berisi panjang sekuen dan jaraknya daria wal blok. Kemungkinan panjang blok adalah 2-258 bytes dan kemungkinan jarak atau lebar papan luncur yakni 1-32,768 bytes. Referensi ini dapat dibuat pada blok manapun selama masih dalam jangkauan papan luncur

Tahap kompresi selanjutnya dalah kompresi menggunakan pohon Huffman, yakni mengganti sekuen bit atau data yang sering muncul dengan simbol tertentu yang tentunya lebih pendek dan mengganti sekuen yang jarang muncul dengan simbol yang relatif lebih panjang. Hal ini berarti jumlah kemunculan suatu sekuen berbanding terbalik dengan panjang kode Huffman untuk sekuen tersebut nantinya.Pada kompresi Deflate, dibuat suatu pohon Huffman yang memiliki ruang untuk 288 simbol.

Interval	Keterangan
0-255	Representasi literal bytes/simbol 0-255
256	Representasi blok terakhir, berhenti jika blok terakhir, proses blok selanjutnya jika bukan blok terakhir
257-285	Kombinasi dengan bit ekstra, kemungkinan panjang 3-258 bytes
286-287	Tidak digunakan, ilegal namun masih bagian dari pohon

Kode untuk panjang sekuen selalu diikuti dengan kode yang merepresentasikan jarak. Berdasarkan hasil pembacaan kode jarak, ekstra bit selanjutnya mungkin perlu dibaca untuk menghasilkan jarak akhir.

Untuk mengetahui apakah sebuah bit adalah panjang atau bukan, dilihat hasil pembacaan. Jika hasil bacaan adalah 257 atau lebih, maka karakter tersebut bukanlah literal melainkan panjang sekuen. Karakter 257 berarti panjang sekuen adalah 3, 258 berarti panjang sekuen adalah 5, dan seterusnya.

Pohon jarak memiliki ruang untuk 32 simbol.

Simbol	Jarak	Bit ekstra
0-3	1-4	0
4-5	5-8	1
6-7	9-16	2
8-9	17-23	3
...	...	...
26-27	8,193-16, 384	12
28-29	16,385-32,768	13
30-31	Tidak digunakan, tetapi masih merupaan bagian dari pohon	

Untuk kode jarak 2-29, jumlah bit ekstra dapat dihitung dengan (1)

$$\frac{n}{2}-1 \quad (1)$$

Pada proses kompresi, enkoderlah yang menentukan waktu yang digunakan untuk mencari sekuen yang cocok dan kembar.

Bila kita lihat pada implementasi pada zlib/gzip, pengguna diberikan kebebasan untuk menentukan skala papan luncur yang akan digunakan. Opsi yang mungkin dimulai dari 0, yang berarti kompresi data tidak perlu dilakukan, hingga 9, yakni kemampuan maksimum implementasi pada zlip/gzip untuk kompresi data.

Sejatinya enkoder tidak perlu mengirimkan seluruh kemungkinan simbol pada alfabet. Pada kasus di mana tidak seluruh alfabet dipakai, pada pohon node dari karakter tersebut akan dibiarkan kosong atau bahkan dipakai untuk keperluan lain. Hal ini berarti mungkin bahwa literal alfabet hanya sepanjang 16 karakter dan bukan 256. Hal ini dapat terjadi jika hanya terdapat 16 karakter unik pada data yang akan dikompresi. Hal ini penting untuk diperhatikan agak penggunakan ruang untuk pohon Huffman dapat menjadi seminimal mungkin.

Kompresi data menggunakan algoritma Deflate berarti mengetahui cara mendapat alfabet yang benar untuk minimisasi yang optimal. Hal ini dapat diketahui dengan statistik atau menghitung frekuensi tiap alfabet pada data yang akan dikompresi. Walaupun membutuhan waktu yang lebih lama, namun cara kedua lebih ideal untuk mewujudkan kompresi yang optimal.

V. APLIKASI

Implementais atau aplikasi dari algoritma Deflate yang telah dibahas dapat kita amati pada banyak format data. Salah satu variasi dari Deflate adalah Deflate64. Pada Deflate 64, mekanisme dasar proses kompresi yang digunakan sejatinya sama, namun perbedaan terletak pada lebar jendela luncur yang diperbesar menjadi 64kB dna penambahan 14 bit pada kode jarak juga, 16 bit pada kode panjang sehingga dapat didefinisikan panjang dari 3 hinggal 65538. Hal ini berujung pada kemampuan kompresi yang lebih tinggi pada Deflate 64, bersamaan dengan kebutuhan waktu yang sedikit lebih singkat, dibanding dengan Deflate. Konsep Deflate64 digunakan pada beberapa proyek open sourse, contohnya 7-zip.

Selain itu terdapat pula implementasi pada encoder, salah satunya tentu ialah PKZIP, implementasi pertama dari algoritma Deflate oleh penciptanya sendiri, Phil Katz, sebagai bagian dari PKZip. Implementasi lainnya dapat dilihat pada Halibut, Plan 9 From bell Labs, Deflate stream, dan banyak encoder lainnya.

VI. KESIMPULAN

Pohon Huffman adalah salah satu aplikasi dari teori dasar pohon yang sering digunakan dalam proses kompresi data. Kompresi dengan pohon Huffman relatif sangkil karena proses kompresi memperhatikan frekuensi dari tiap data yang unik pada data secara keseluruhan. Data dengan frekuensi tinggi kemudian diberikan kode dengan panjang lebih pendek dibanding dengan data dengan frekuensi rendah. Hal ini terbukti menghasilkan kode yang lebih pendek dibanding proses pengkodean di mana panjang kode tiap data sama.

Salah satu aplikasi dari pohon Huffman terdapat dapa algoritma kompresi Deflate. Algoritma Deflate menggabungkan konsep pohon Huffman dengan konsep eliminasi string kembar. Penggabungan kedua konsep ini menghasilkan sebuah algoritma kompresi yang sangat baik dan sudah banyak diimplementasikan pada perangkat lunak kompresi juga format-format data untuk

meningkatkan kesangkilan pemakaian memori pada komputer.

Meskipun algoritma Deflate adalah algoritma yang sangat baik, bukan berarti pengembangan lebih lanjut tidak dapat dilakukan. Sebagai bukti, terdapat suatu variasi baru yang berakar dari algoritma Deflate, yakni Deflate64 yang terbukti memiliki kemampuan kompresi lebih tinggi dan membutuhkan waktu kompresi lebih singkat dibanding algoritma Deflate. Hal ini berarti pengembangan dapat terus dilakukan untuk mewujudkan algoritma yang semakin baik untuk proses kompresi yang makin optimal.

VII. DAFTAR PUSTAKA

[1] Binary Essence: Deflate64(Enhanced Deflate). http://www.binaryessence.com/dct/imp/en000225.htm

[2] Bender, Ryan. "Example: Huffman Encoding Trees". http://mitpress.mit.edu/sicp/full-text/sicp/book/node41.html

[3] Feldspar, Antaneus. "An Explanation of the *Deflate* Algortihm". Comp.Compression. 1997.

[4] Solomon, David. "Data Compression: The Complete Reference". London: Springer-Verlag London Ltd, 2007, p. 241.

[5] Thijssen, Joshua. "Deflating the Universe". http://www.adayinthelifeof.nl/2010/06/02/deflating-the-universe/.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 10 Desember 2011

Nurul Fithria Lubis (13510012)