

PENGHEMATAN FILE DATA *TEXT* DENGAN APLIKASI ALGORITMA HUFFMAN STATIS

Farida Arinie S

Dosen Jurusan Teknik Elektro Politeknik Negeri Malang

Abstrak

Huffman Algorithm adalah salah satu algoritma yang digunakan untuk membuat kompresi jenis lossy compression, yaitu kompresi data dimana tidak satu byte pun hilang sehingga data tersebut utuh dan disimpan sesuai dengan aslinya. Algoritma Huffman statik yaitu suatu algoritma yang hanya digunakan untuk kompresi pada file text, sehingga dapat diketahui ukuran sebelum dan sesudah data dikompresi dan tanpa menghilangkan isi informasinya. Hasil kompresi algoritma Huffman menunjukkan data terkompresi mengecil rata-rata sebesar 53.8 %(271400 byte) dari data aslinya.

Kata kunci : Algoritma Huffman, Kompresi , file text

1. PENDAHULUAN

1.1 Latar Belakang

Huffman Algorithm adalah salah satu algoritma kompresi tertua yang disusun oleh David Huffman pada tahun 1952. Algoritma tersebut digunakan untuk membuat kompresi jenis *lossy compression*, yaitu pemampatan data dimana tidak satu byte pun hilang sehingga data tersebut utuh dan disimpan sesuai dengan aslinya.

Prinsip kerja algoritma Huffman adalah mengkodekan setiap karakter ke dalam representasi bit. Representasi bit untuk setiap karakter berbeda satu sama lain berdasarkan frekuensi kemunculan karakter. Semakin sering karakter tersebut muncul, maka semakin pendek panjang representasi bitnya. Sebaliknya bila semakin jarang frekuensi suatu karakter untuk muncul, maka semakin panjang representasi bit untuk karakter tersebut.

Dengan adanya prinsip kerja algoritma huffman tersebut maka akan dilakukan penelitian pada aplikasi kompresi untuk data text sehingga dapat memeperkecil ukuran data *text* yang akan disimpan dalam suatu memori tanpa kehilangan kode yang terkompresi.

1.2 Rumusan Masalah

Berdasarkan latar belakang diatas, rumusan masalah ditekankan pada,

1. Bagaimana mengaplikasikan algoritma Huffman statis yang digunakan untuk kompresi data teks.
2. Bagaimana menganalisa hasil kompresi dari algoritma Huffman statis dari data teks yang bisa disimpan dalam ukuran yang lebih kecil.
3. Bagaimana membuat pemodelan algoritma Huffman statis untuk melakukan proses pembacaan yang berulang terhadap data yang akan kompresi, berdasarkan pengelompokan dan penghitungan karakter-karakter data

1.3 Tujuan

Tujuan yang akan dicapai dalam penelitian ini adalah, dengan mengaplikasikan algoritma Huffman ini suatu data teks dapat dikompresi sehingga akan berukuran lebih kecil dari data yang sebenarnya dan tidak menghilangkan isi informasinya.

2. Tinjauan Pustaka

2.1. Algoritma Huffman

Algoritma Huffman menggunakan prinsip pengkodean yang mirip dengan kode Morse, yaitu tiap karakter (simbol) dikodekan hanya dengan rangkaian beberapa bit, dimana karakter yang sering muncul dikodekan dengan rangkaian bit yang pendek dan karakter yang jarang muncul dikodekan dengan rangkaian bit yang lebih panjang. Pandangan dasar dari source coding adalah menghilangkan redundansi dari sumber. *Source Coding* menghasilkan data kompresi dan mengurangi rate dari transmisi. Secara umum kita dapat meng-kompres data tanpa menghilangkan informasi (*lossless source coding*) atau mengkompres data dengan adanya kehilangan informasi (*loss source coding*) [1].

Teori *Source Encoding* merupakan salah satu dari ketiga teorema dasar dari teori informasi yang diperkenalkan oleh Shannon (1948). Teori Source Encoding mencanangkan sebuah limit dasar dari sebuah ukuran dimana keluaran dari sebuah sumber informasi dapat dikompresi tanpa menyebabkan probabilitas error yang besar. Kita telah mengetahui bahwa *entropy* dari sebuah sumber informasi merupakan sebuah ukuran dari isi informasi pada sebuah sumber. Sehingga, dari pendapat teori *source encoding* bahwa *entropy* dari sebuah sumber sangat penting. Ukuran entropy source coding pada persamaan 1[3].

$$H(X) = - \sum_{i=1}^m p_i \log_2 p_i \dots\dots\dots(1)$$

Efisiensi dari sumber ditentukan oleh $H(X)/H(X)_{\max}$, bila p_i merupakan probabilitas pada simbol ke-I, dan $H(X)$ maksimum ketika sumber mempunyai simbol probabilitas yang sama.

Metode Kompresi Huffman

Kode Huffman pada dasarnya merupakan kode prefiks (*prefix code*). Kode prefiks adalah himpunan yang berisi sekumpulan kode biner, dimana pada kode prefiks ini tidak ada kode biner yang menjadi awal bagi kode biner yang lain. Kode prefiks biasanya direpresentasikan sebagai pohon biner yang diberikan nilai atau label. Diberi label 0, sedangkan pada cabang kanan pada pohon biner diberi label 1. Rangkaian bit yang terbentuk pada setiap lintasan dari akar ke daun merupakan kode prefiks untuk karakter yang berpadanan. Pohon biner ini biasa disebut pohon Huffman. Langkah-langkah pembentukan pohon Huffman adalah sebagai berikut[2]:

- a. Baca semua karakter di dalam teks untuk menghitung frekuensi kemunculan setiap karakter. Setiap karakter penyusun teks dinyatakan sebagai pohon bersimpul tunggal. Setiap simpul di-assign dengan frekuensi kemunculan karakter tersebut.
1. Baca semua karakter di dalam teks untuk menghitung frekuensi kemunculan setiap karakter. Setiap karakter penyusun teks dinyatakan sebagai pohon bersimpul tunggal. Setiap simpul di-assign dengan frekuensi kemunculan karakter tersebut.
- b. Terapkan strategi algoritma greedy sebagai berikut : Gabungkan dua buah pohon yang mempunyai frekuensi terkecil pada sebuah akar. Setelah digabungkan akar tersebut akan mempunyai frekuensi yang merupakan jumlah dari frekuensi dua buah pohon-pohon penyusunnya.
- c. Ulangi langkah 2 sampai hanya tersisa satu buah pohon Huffman. Agar pemilihan dua pohon yang akan digabungkan berlangsung cepat, maka semua

yang ada selalu teratur menaik berdasarkan frekuensi. Sebagai contoh, dalam kode ASCII string 7

huruf “ABACCDAA” membutuhkan representasi $7 \times 8 \text{ bit} = 56 \text{ bit}$ (7 byte), dengan rincian sebagai berikut:

A = 01000001

B = 01000010

A = 01000001

C = 01000011

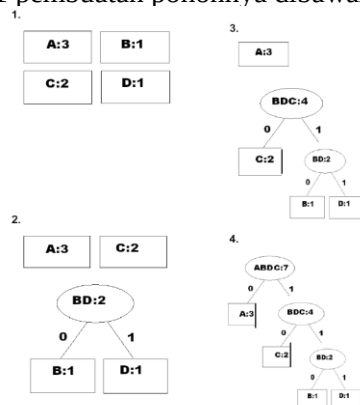
C = 01000011

D = 01000100

A = 01000001

Pada string di atas, frekuensi kemunculan A = 3, B = 1, C = 2, dan D = 1,

Untuk lebih jelas mengenai proses pembentukan pohon huffman, dapat dilihat ilustrasi pada gambar 1 pembuatan pohonnya dibawah ini:



Gambar 1. Pohon H uffman untuk Karakter “ABACCDAA”

Proses Encoding

Encoding adalah cara menyusun string biner dari teks yang ada. Proses encoding untuk satu karakter dimulai dengan membuat pohon Huffman terlebih dahulu. Setelah itu, kode untuk satu karakter dibuat dengan menyusun nama string biner yang dibaca dari akar sampai ke daun pohon Huffman. Langkah-langkah untuk men-encoding suatu string biner adalah sebagai berikut [5]:

1. Tentukan karakter yang akan di-encoding
2. Mulai dari akar, baca setiap bit yang ada pada cabang yang bersesuaian sampai ketemu daun dimana karakter itu berada
3. Ulangi langkah 2 sampai seluruh karakter di-encoding Sebagai contoh kita dapat melihat tabel dibawah ini, yang merupakan hasil encoding untuk pohon Huffman pada gambar 1,

Tabel 1. Kode Huffman untuk Karakter “ABCD”

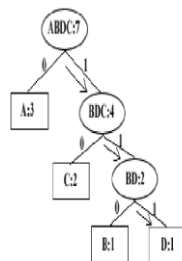
Karakter	String Biner Huffman
A	0
B	110
C	10
D	111

Dengan menggunakan kode Huffman ini, string “ABACCD A” direpresentasikan menjadi rangkaian bit : 0 110 0 10 10 111 0. Jadi, jumlah bit yang dibutuhkan hanya 13 bit.

Proses Decoding

Decoding merupakan kebalikan dari encoding. Decoding berarti menyusun kembali data dari string biner menjadi sebuah karakter kembali. Decoding dapat dilakukan dengan dua cara, yang pertama dengan menggunakan pohon Huffman dan yang kedua dengan menggunakan tabel kode Huffman. Langkah-langkah men – decoding suatu string biner dengan menggunakan pohon Huffman adalah sebagai berikut :

1. Baca sebuah bit dari string biner.
2. Mulai dari akar
3. Untuk setiap bit pada langkah 1, lakukan traversal pada cabang yang bersesuaian.
4. Ulangi langkah 1, 2 dan 3 sampai bertemu daun. Kodekan rangkaian bit yang telah dibaca dengan karakter di daun.
5. Ulangi dari langkah 1 sampai semua bit di dalam string habis. Sebagai contoh kita akan men-decoding string biner yang bernilai ”111” adalah karakter D.



Gambar 2. Proses Decoding dengan Menggunakan Pohon Huffman

Cara yang kedua adalah dengan menggunakan tabel kode Huffman. Sebagai contoh kita akan menggunakan kode Huffman pada Tabel 1 untuk merepresentasikan string “ABACCD A”. Dengan menggunakan Tabel 1 string tersebut akan direpresentasikan menjadi rangkaian bit :

0 110 0 10 10 111 0.

Jadi, jumlah bit yang dibutuhkan hanya 13 bit. Dari Tabel 1 tampak bahwa kode untuk sebuah simbol/karakter tidak boleh menjadi awalan dari kode simbol yang lain guna menghindari keraguan (ambiguitas) dalam proses dekompresi atau decoding. Karena tiap kode Huffman yang dihasilkan unik, maka proses decoding dapat dilakukan dengan mudah. Contoh: saat membaca kode bit pertama dalam rangkaian bit “011001010110”, yaitu bit “0”, dapat langsung disimpulkan bahwa kode bit “0” merupakan pemetaan dari simbol “A”.

Kemudian baca kode bit selanjutnya, yaitu bit “1”. Tidak ada kode Huffman “1”, lalu baca kode bit selanjutnya, sehingga menjadi “11”. Tidak ada juga kode Huffman “11”, lalu baca lagi kode bit berikutnya, sehingga menjadi “110”. Rangkaian kode bit “110” adalah pemetaan dari simbol “B” [4].

3. Hasil Dan Pembahasan

Desain Algoritma Huffman

File hasil pemampatan dengan algoritma *Huffman* harus ditandai pada awal datanya sehingga sewaktu pengembalian ke file asli dapat dikenali apakah file

tersebut benar merupakan hasil pemampatan dengan algoritma ini. Pada program ini format pengenalan file tersebut ditulis pada byte pertama, kedua dan ketiga dengan karakter H, U, dan F. Pergantian format tersebut dengan karakter lain yang diinginkan, sesuai dengan jumlahnya. Karakter keempat, kelima dan keenam berisi informasi ukuran file asli dalam byte, 3 karakter ini dapat berisi maksimal FFFFFFFF H atau 16.777.215 byte. Karakter ketujuh berisi informasi jumlah karakter yang memiliki kode *Huffman* atau dengan kata lain jumlah karakter yang frekuensi kemunculannya pada file asli lebih dari nol, jumlah tersebut dikurangi satu dan hasilnya disimpan pada karakter ke tujuh pada file pemampatan Misalnya suatu file dengan ukuran 3.000 byte dan seluruh karakter ASCII terdapat pada file tersebut, jadi :

Karakter 1-3 : HUF

Karena 3.000 = BB8 H, maka :

Karakter 4-6 : 000B88

Karena seluruh karakter ASCII terdapat pada file tersebut, jadi jumlah karakter yang memiliki kode Huffman adalah 256 buah, $256-1=255 = FF$ H maka : Karakter 7 : FF
Maka 7 byte pertama isi file pemampatan adalah :

	H	U	F	Chr (00H)	Chr (0BH)	Chr (88H)	Chr (FFH)	..
Bit	1	2	3	4	5	6	7	8.
Ke								..
-								

Mulai dari karakter ke delapan berisi daftar kode Huffman berturut-turut karakter (1 byte), kode Huffman (2 byte) dan panjang kode Huffman (1 byte). 4 byte berturutan ini diulang untuk seluruh karakter yang dikodekan. Selanjutnya file diisi hasil pemampatan dengan algoritma Huffman seperti yang telah dijelaskan pada bab sebelumnya. Agar lebih jelas kita ulangi contoh sebelumnya, yaitu file yang berisi karakter “PERKARA”. Jika file ini dimampatkan dengan metode Huffman, maka file hasil pemampatan akan kita dapatkan sebagai berikut :

Karakter 1-3 : HUF = 48 55 46 H

Karena file tersebut berukuran 7 byte, jadi :

Karakter 4-6 : 00 00 07 H.

Daftar kode Huffman yang dicari yaitu :

E = 010

K = 011

P = 00

A = 10

R = 11

Karena jumlah karakter yang memiliki kode *Huffman* ada 5 buah, maka $5-1=4$.

Karakter 7 : 04 H.

Karakter 8 : karakter E = 45 H.

Karena kode *Huffman* dari karakter E adalah 010, sedangkan tempat yang disediakan sebanyak 2 byte maka karakter 9 dan 10 menjadi 00000000 00000010 B.

Karakter 9 : 00000000 B = 00 H.

Karakter 10 : 00000010 B = 02 H.

Panjang kode *Huffman* dari karakter E adalah 3 bit (010), jadi :

Karakter 11: 03H.

Cara tersebut di atas diulang untuk karakter K, P, A dan R sehingga didapat :

Karakter 12: karakter K = 4B H.

Karakter 13: 00000000 B = 00 H.

Karakter 14: 00000011 B = 03 H.

Karakter 15: 03H.

Karakter 16: karakter P = 50 H.

Karakter 17: 00000000 B = 00 H.

Karakter 18: 00000000 B = 00 H.

Karakter 19: 02H.

Karakter 20: karakter A = 41 H.

Karakter 21: 00000000 B = 00 H.

Karakter 22: 00000010 B = 02 H.

Karakter 23: 02H.

Karakter 24: karakter R = 52 H.

Karakter 25: 00000000 B = 00 H.

Karakter 26: 00000011 B = 03 H.

Karakter 27: 02H.

Selanjutnya pemampatan karakter-karakter “PERKARA” adalah :
0001011011101110. Jika kita potong-potong menjadi 8 bit tiap bagian akan menjadi

00010110 = 16 H.

11101110 = EE H.

Jadi :

Karakter 28 : 16 H.

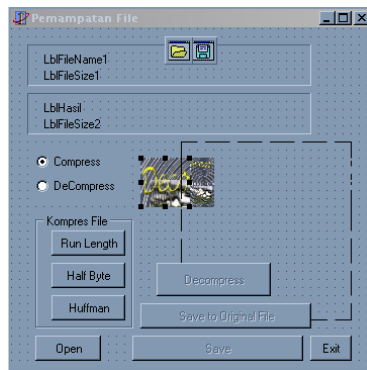
Karakter 29 : EE H.

File hasil pemampatan akan menjadi berukuran 29 byte yang dalam heksadesimal berisi :

48 55 46 00 00 07 04 45 00 02 03 4B 00 03 03 50 00 00 02 41 00 02 02 52 00 03
02 16 EE

Jika kita bandingkan dengan file aslinya yang berukuran 7 byte, hasil ini bukan menjadi lebih kecil, tetapi akan menambah byte sebesar $29-7=22$ byte.

Hal ini wajar dalam file yang berukuran sangat kecil seperti ini, tapi dalam file-file yang berukuran besar, algoritma Huffman ini sangat efektif. Format yang digunakan pada program ini merupakan format sederhana untuk program pemampatan file, anda dapat memodifikasinya dengan format sendiri atau menambahkan informasi-informasi lainnya, misalnya saja dengan menambahkan informasi nama file asli sehingga pada saat pengembalian ke file aslinya, nama filenya sudah disiapkan. Program ini dibuat dengan program Borland Delphi versi 5 yang berorientasi pada objek (Object Orientation), mengingat saat ini hampir semua program sudah menggunakan windows sebagai base-nya. Setelah Hasil dari proses kompresio text adalah sebagai berikut ampak form kompresi.



Gambar 3. Form Delphi 5 . Untuk Kompresi

Tabel 2. Hasil Kompresi Algoritma Huffman

Nama file	Ukuran file asli(byte)	Hasil Kompresi Algoritma Huffman(byte)	Persentasi hasil kompresi
suratkuasa.doc	104448	49277	(47 %)
elevator.cdr	193750	194056	(100 %)
fifa2000.exe	1605632	1305473	(81 %)
bunga.bmp	360054	70890	(19 %)
Event.dat	13195	2427	(18 %)
Readme.txt	10756	6277	(58 %)

Setelah terjadi kompresi file yang diperoleh dari algoritma Huffman menjadi lebih kecil dari file aslinya. Dengan menggunakan algoritma *Huffman*, cukup efektif untuk berbagai macam jenis file, meskipun ada beberapa file yang hasilnya pemampatannya lebih besar dari yang dihasilkan. Tapi yang terpenting disini dapat disimpulkan bahwa tidak ada algoritma yang paling efektif untuk setiap file karena hasil pemampatan setiap algoritma tergantung dari isi file yang akan dikompresi.

4. Kesimpulan

1. Algoritma Huffman statis adalah salah satu algoritma kompresi, yang banyak digunakan dalam kompresi teks.
2. Terdapat 3 tahapan dalam menggunakan algoritma Huffman, yaitu:
 - membentuk pohon Huffman
 - melakukan encoding dengan menggunakan pohon Huffman
 - melakukan decoding
3. Algoritma Huffman mempunyai kompleksitas waktu $O(n \log n)$.
4. Algoritma Huffman adalah salah satu algoritma yang menggunakan prinsip penyusunan pohon biner Huffman.
5. Dari hasil pengujian yang dilakukan, algoritma Huffman statis dapat mengompres teks rata-rata sebesar 53.8 % (271400 byte) jika dibandingkan dengan menggunakan kode ASCII.

6. Tidak ada algoritma yang paling efektif untuk setiap file karena hasil pemampatan setiap algoritma tergantung dari isi file yang akan dimampatkannya .

Daftar Pustaka

Howe, D., "*Free On-line Dictionary of Computing*",
<http://www.foldoc.org/1993>.

Huffman Coding
http://www.en.wikipedia.org/wiki/Huffman_coding

Rinaldi Munir, 2005, *Diktat Kuliah IF2251 Strategi Algoritmik*, Penerbit ITB.

Data Structures and Algorithms: Introduction
http://ciips.ee.uwa.edu.au/~morris/Year2/PLS2_10/introduction.html

Practical Huffman Coding
<http://www.compressconsult.com/huffman/>