KOMPRESI FILE MENGGUNAKAN ALGORITMA HUFFMAN KANONIK

Asrianda
Dosen Teknik Informatika Universitas Malikussaleh

ABSTRAK

Algoritma Huffman adalah salah satu algoritma kompresi. Algoritma huffman merupakan algoritma yang paling terkenal untuk mengompres teks. Terdapat tiga fase dalam menggunakan algoritma Huffman untuk mengompres sebuah teks, pertama adalah fase pembentukan pohon Huffman, kedua fase encoding dan ketiga fase decoding. Prinsip yang digunakan oleh algoritma Huffman adalah karakter yang sering muncul di -encoding dengan rangkaian bit yang pendek dan karakter yang jarang muncul di-encoding dengan rangkaian bit yang lebih panjang. Teknik kompresi algoritma Huffman mampu memberikan penghematan pemakaian memori sampai 30%. Algoritma Huffman mempunyai kompleksitas O(n log n) untuk himpunan dengan n karakter.

Kata kunci: Algoritma Huffman, pohon HKeyword, karakter, kompresi

Pendahuluan

Data yang berukuran besar seringkali sulit untuk disimpan dalam media penyimpanan yang berukuran terbatas. Karena alasan itu, data yang besar sebelum disimpan biasanya dimampatkan terlebih dahulu agar ukurannya lebih kecil dari semula.

Salah satu algoritma yang sering digunakan dalam teknik pemampatan data adalah kode huffman. Teknik ini mampu memampatkan sampai dengan tiga puluh persen dari ukuran semula. Tetapi proses decoding string biner menjadi data kembali masih kurang efisien. Karena itu, kami mencoba membahas salah satu

algoritma yang mirip dengan algoritma Huffman tetapi lebih efisien yaitu algoritma Huffman Kanonik.

Teks adalah kumpulan dari karakter -karakter atau string yang menjadi satu kesatun. Teks yang memuat banyak karakter didalamnya selalu menimbulkan masalah pada media penyimpanan dan kecepatan waktu pada saat transmisi data. Media penyimpanan yang terbatas, membuat semua orang mencoba berpikir untuk menemukan sebuah cara yang dapat digunakan untuk mengompres teks.

Walaupun pada saat ini terdapat banyak algoritma untuk mengompres data termasuk teks, seperti LIFO, LZHUF, LZ77 dan variannya (LZ78, LZW, GZIP), Dynamic Markov Compression (DMC), Block -Sorting Lossless, Run-Length, Shannon-Fano, Arithmetic, PPM (Prediction by Partial Matching), Burrows-Wheeler Block Sorting, dan Half Byte. Namun penulis menggunakan algoritma Huffman, karena algoritma ini banyak digunakan dan mudah diimplementasikan dalam proses pengompresan teks.

Kompresi adalah proses pengubahan sekumpulan data menjadi bentuk kode dengan tujuan untuk menghemat kebutuhan tempat penyimpanan dan waktu untuk transmisi data.

Dengan menggunakan algoritma Huffman, proses pengompresan teks dilakukan dengan menggunakan prinsip pengkodean, yaitu tiap karakter dikodekan dengan rangkaian beberapa bit sehingga menghasilkan hasil yang lebih optimal.

Tujuan dari penulisan makalah ini adalah untuk mengetahui keefektifan algoritma Huffman dalam kompresi teks dan memaparkan cara-cara mengompresi teks dengan menggunakan algoritma Huffman.

Algoritma Huffman

Algoritma Huffman, yang dibuat oleh seorang mahasiswa MIT bernama David Huffman pada tahun 1952, merupakan salah satu metode paling lama dan paling terkenal dalam kompresi teks.

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.

Berdasarkan tipe peta kode yang digunakan untuk mengubah pesan awal (isi data yang diinputkan) menjadi sekumpulan codeword, algoritma Huffman termasuk kedalam kelas algoritma yang menggunakan metode statik. Metoda statik adalah metoda yang selalu menggunakan peta kode yang sama, metoda ini membutuhkan dua fase (two-pass): fase pertama untuk menghitung probabilitas kemunculan tiap simbol dan menentukan peta kodenya, dan fase kedua untuk mengubah pesan menjadi kumpulan kode yang akan di taransmisikan, Sedangkan berdasarkan teknik pengkodean simbol yang digunakan, algoritma Huffman menggunakan metode symbolwise. Metoda symbolwise adalah metode yang menghitung peluang kemunculan dari setiap simbol dalam satu waktu, dimana simbol yang lebih sering muncul diberi kode lebih pendek dibandingkan simbol yang jarang muncul.

Pembentukan Pohon Huffman

Kode Huffman pada dasarnya merupakan kode prefiks (prefix code). Kode prefiks adalah himpunan yang berisi sekumpulan kode biner, dimana pada kode prefik ini tidak ada kode biner yang menjadi awal bagi kode biner yang lain. Kode prefix biasanya direpresentasikan sebagai pohon biner yang diberikan nilai atau label . Untuk cabang kiri pada pohon biner 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 [3] :

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.

2. Terapkan strategi algoritma greedy sebagai 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.

3. Ulangi langkah 2 sampai hanya tersisa satu buah pohon Huffman. Agar pemilihan dua pohon yang akan digabungkan berlangsung cepat, maka semua yang ada selalu terurut menaik berdasarkan frekuensi.

Sebagai contoh, dalam kode ASCII string 7 huruf "ABACCDA" membutuhkan representasi 7 × 8 bit = 56 bit (7 byte), dengan rincian sebagai berikut:

A = 01000001

B = 01000010

A = 01000001

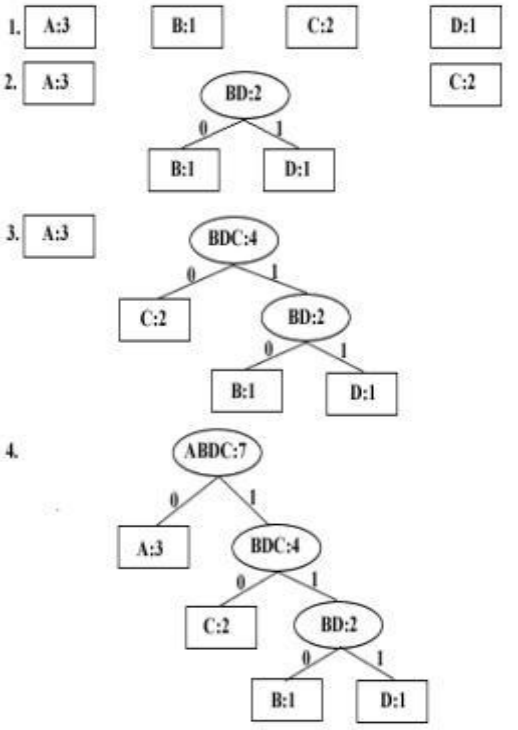
C = 01000011

C = 01000011

D = 01000100

A = 01000001

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



Gambar 1. Pohon Huffman untuk Karakter "ABACCDA"

Proses Encoding

Aturan-aturan proses encoding pada algoritma Huffman Kanonik adalah sebagai berikut:

1. Panjang kode untuk suatu simpul adalah sebesar aras+1 simpul tersebut.
2. String biner simpul paling dalam yang terletak paling kiri diberi nilai 0 semuanya. Untuk simpul berikutnya(bergeser dari kiri ke kanan) string binernya naik satu nilai dari simpul sebelumnya.

3. Apabila semua simpul pada kedalaman yang sama telah di-encode, maka proses encoding dilanjutkan ke aras yang lebih rendah. Hanya saja string binernya tidak dimulai dengan semuanya 0. String biner simpul paling kiri dimulai dengan kode baru. Kode baru itu merupakan kenaikan 1 nilai dari string biner simpul yang terakhir di-encode namun biner paling belakangnya dihilangkan sehingga panjang kodenya berkurang satu. Simpul berikutnya diencode engan string binernya naik satu nilai(bergeser dari kiri ke kanan).
4. Proses berhenti bila telah mencapai akar.

Terdapat pohon Huffman sebagai berikut: ((((B,F),A),E),(((G,C),D),H)) dengan kedalaman 4

Karakter	*String* Biner Huffman Biasa	*String* Biner Huffman Kanonik	Aras
A	001	010	2
B	0000	0000	3
C	1001	0001	3
D	101	011	2
E	01	10	1
F	0001	0010	3
G	1000	0011	3
H	11	11	1

Pada pohon Huffman tersebut, karakter 'B' terletak pada simpul paling dalam dan paling kiri dan diberi kode 0000. Karakter 'C' terletak di sebelah kanan 'B' pada aras yang sama sehingga diberi kode 0001 yang lebih satu nilai dari 0000. Lalu karakter 'F' dan 'G' diberi kode 0010 dan 0011 dengan cara yang sama. Untuk karakter 'A' yang berada pada aras yang lebih rendah diberi kode yang lebih satu nilai dari 0011 yaitu 0100 namun biner belakangnya dihilangkan menjadi 010. Karakter 'D' yang berada pada aras yang sama dengan 'A' diberi kode 011 yang lebih satu nilai dari 010. Demikian seterusnya hingga semua karakter telah di-encode.

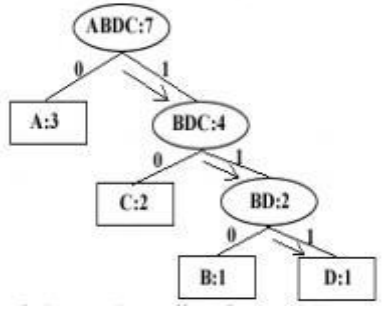
Proses Decoding

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 mendecoding 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 dibacadengan 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”



Gambar 2. Proses Decoding dengan Menggunakan Pohon Huffman

Setelah kita telusuri dari akar, maka kita akan menemukan bahwa string yang mempunyai kode Huffman “111” adalah karakter D. Cara yang kedua adalah dengan menggunakan tabel kode Huffman. Sebagai contoh kita akan menggunakan kode Huffman pada Tabel 1 untuk merepresentasikan string “ABACCDA”. Dengan

menggunakan Tabel 1 string tersebut akan direpresentasikan menjadi rangkaian bit : 0 110 0 10 10 1110. 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".

Kompleksitas Algoritma Huffman

Algoritma Huffman mempunyai kompleksitas waktu O(n log n), karena dalam melakukan sekali proses itersi pada saat penggabungan dua buah pohon yang mempunyai frekuensi terkecil pada sebuah akar membutuhkan waktu O (log n), dan proses itu dilakukan berkali-kali sampai hanya tersisa satu buah pohon Huffman itu berarti dilakukan sebanyak n kali[4].

Pengujian Algoritma Huffman

Pada pengujian digunakan, kita akan menencoding sebuah teks yang berisi 100.000 string, diantaranya 45.000 karakter 'g', 13.000 karakter 'o', 12.000 karakter 'p', 16.000 karakter 'h', 9.000 karakter 'e', dan 5.000 karakter 'r' dengan menggunakan 3 cara, yaitu dengan menggunakan kode ASCII , kode 3-bit dan kode Huffman. Setelah itu ketiga kode tersebut akan dibandingkan satu sama lainnya.

a. Kode ASCII

Karakter	ASCII	Biner
g	103	1100111
o	111	1101111
p	112	1110000
h	104	1101000
e	101	1100101
r	114	1110010

Tabel 3. Kode ASCII untuk karakter "g,o,p,h,e,r,"

Untuk meng-encoding teks tersebut kita membutuhkan sebanyak :

- Untuk karakter 'g'

  4.5000 x 8 bit (1100111) = 360.000 bit
- Untuk karakter 'o'

  13.000 x 8bit (1101111) = 104.000 bit
- Untuk karakter 'p'

  12.000 x 8bit (1110000) = 96.000 bit
- Untuk karakter 'h'

  16.000 x 8bit (1101 000) = 128.000 bit
- Untuk karakter 'e'

  9.000 x 8bit (1100101) = 72.000 bit
- Untuk karakter 'r'

  5.000 x 8bit (1110010) = 40.000 bit

  jumlah = 800.000 bit

b. 3-bitKode

Karakter	Kode	String Biner
g	0	000
o	1	001
p	2	010
h	3	011
e	4	100
r	5	101

Tabel 4. 3-bit Kode untuk karakter "g,o,p,h,e,r"

Untuk meng-encoding teks tersebut kita membutuhkan sebanyak :

- Untuk karakter 'g'

45.000 x 3 bit (000) = 135.000 bit

- Untuk karakter ‘o’

13.000 x 3bit (001) = 39 .000 bit

- Untuk karakter ‘p’

12.000 x 3bit (010) = 36.000 bit

- Untuk karakter ‘h’

16.000 x 3bit (011) = 48 .000 bit

- Untuk karakter ‘e’

9.000 x 3bit (100) = 27 .000 bit

- Untuk karakter ‘r’

5.000 x 3bit (101) = 15.000 bit

jumlah = 300.000 bit

c. Kode Huffman

Karakter	Frekuensi	Peluang	Kode Huffman
g	45000	3/13	0
o	13000	3/13	101
p	12000	1/13	100
h	16000	1/13	111
e	9000	1/13	1101
r	5000	1/13	1100

Tabel 5. Kode Huffman untuk Karakter “g,o,p,h,e,r”,

Untuk meng-encoding teks tersebut kita membutuhkan sebanyak :

- Untuk karakter ‘g’

45.000 x 1 bit (0) = 45 .000 bit

- Untuk karakter ‘o’

13.000 x 3bit (101) = 39 .000 bit

- Untuk karakter ‘p’

12.000 x 3bit (110) = 36 .000 bit

- Untuk karakter ‘h’

16.000 x 3bit (111) = 48.000 bit

- Untuk karakter ‘e’

9.000 x 4bit (1101) = 36.000 bit

- Untuk karakter ‘r’

5.000 x 4bit (1100) = 20.000 bit

jumlah = 224.000 bit

Dari data diatas kita dapat lihat bahwa dengan menggunakan kode ASCII untuk meng-encoding teks tersebut membutuhkan 800.000 bit, sedangkan dengan menggunakan 3-bit kode dibutuhkan 300.000 bit dan dengan menggunakan kode Huffman hanya membutuhkan 224.000. Dengan menggunakan data tersebut maka dapat kita lihat bahwa dengan menggunakan algoritma Huffman dapat mengompres teks sebesar 70% dibandingkan kita menggunakan kode ASCII dan sebesar 25,3% dibandingkan kita menggunakan 3-bit kode.

Kesimpulan

Algoritma Huffman adalah salah satu algoritma kompresi, yang banyak digunakan dalam kompresi teks. Dari ketiga teknik tersebut, proses kompresi data menggunakan Algoritma Huffman memiliki rasio kompresi yang paling tinggi, tetapi memiliki kompleksitas di dalam penyunsunan pohon Huffman. Algoritma Huffman adalah salah satu algoritma yang menggunakan prinsip algoritma greedy dalam penyusunan pohon Huffman. Dari hasil pengujian yang dilakukan, algoritma Huffman dapat mengompres teks sebesar 70% jika dibandingkan dengan menggunakan kode ASCII dan sebesar 25,3% jika dibandingkan dengan kita menggunakan 3-bit kode.

Referensi

[1]. Blelloch, G.E., 2001, Introduction to Data Compression. Computer Science Department, Carnegie Mellon University.

[2]. Cormen, T.H, Charles E.L., Ronald L.R., and Clifford S. 2001. Introduction to Algoritms. Second Edition. London: McGraw-Hill Book Company.

[3]. Haryanto, R.I., 2009. Kompresi Data dengan Algoritma Huffman dan Perbandingannya dengan Algoritma LZW dan DMC, Makalah IF2091 Strategi Algoritmik.

[4]. Schindler, M., Practical Huffman Coding. http://www.compressconsult.com/huffman/ Diakses 25 Januari 2011.

[5]. Silalahi, B.P., Julio A., Danny D.S. Perbandingan Algoritma Huffman Statik dengan Algoritma Huffman Adaptif pada Kompresi Teks. Jurnal Penelitian.

[6]. Wikimedia, Huffman Coding. . http://en.wikipedia.org/wiki/Huffman_coding. Diakses 27 Januari 2011.

[7]. Irwan Wardoyo, Peri Kusdinar, , Irvan Hasbi Taufik, KOMPRESI TEKS dengan MENGGUNAKAN ALGORITMA HUFFMAN, http://erizal.files.wordpress.com/2007/10/kompresi-teks-dengan-menggunakan-algoritma-huffman.pdf, diakses pada tanggal 28 Januari 2011

[8]. Yavta Mabaklini Ginting, Bemby Bantara Narendra, Pemampatan Data dengan Algoritma Huffman Kanonik, http://www.informatika.org/~rinaldi/Stmik/Makalah/MakalahStmik13.pdf, Diakses pada tanggal 30 Januari 2011.