Perbandingan Kompresi Data Dengan Algoritma Huffman Statik dan Adaptif

Timotius Triputra Safei (13509017)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509017@std.stei.itb.ac.id

***Abstract*—Makalah ini akan membahas permasalahan kompresi data. Terutama data komputer. Data pada komputer disimpan dalam bentuk biner. Dengan bertambahnya kecanggihan komputer saat ini, maka data yang perlu disimpan semakin banyak. Dengan begitu penyimpanan data harus efisien. Kompresi data yang dilakukan menggunakan algoritma Huffman. Algoritma in berdasarkan pada teorema graf, terutama pohon biner. Algoritma yang akan digunakan terdiri dari 2 jenis yaitu algoritma Huffman statik dan algoritma huffman adaptif. Sedangkan data yang akan disimpan bersifat umum. Akan tetapi untuk studi kasus dan contoh akan digunakan data bertipe string. Selain itu akan dibandingkan juga antara kedua jenis algoritma.**

***Kata Kunci*—kompresi data, algoritma huffman, pohon biner,string.**

***Abstract*—This paper will discuss about data copression problems, focusing on computer data. Data for computeris stored in biner. By today computer ability, data become larger than before. Therefore we need to save it efficiently. In this paper we will use Huffman Algorithm. Huffman algorithm is based in graf theory, especially binary tree. The algorithm that will be used is split into 2 kinds, static Huffman algorithm and adaptive Huffman algorithm. We will use general data, but for the case and example we will use string types. Besides that, we will compare both of algorthm.**

***Index Terms* —data compression, Huffman algorithm, binary tree, string.**

I. PENDAHULUAN

Pada masa kini semua peralatan yang digunakan adalah alat digital. Digital yang berarti taip perubahan adalah diskrit dan tidak kontinu. Pada perkembangannya digital masa kini lebih terutama dalam kode biner. Semua peralatan elektronik saat ini memakai sistem biner yaitu 1 atau 0. Dengan begitu dapat dibuat alat elektronik dengan menggunakan logika dasar.

Dengan digunakannya biner pada pengolahan data, maka semua data tersimpan dalam bentuk biner. Data dalam bentuk biner memiliki beberapa keunggulan, seperti dapat diolah dengan mudah dengan perangkat elektronik masa kini. Akan tetapi hal ini juga menyebabkan beberapa masalah.

Masalah yang dibahas dalam makalah ini adalah masalah kompresi data. Seperti yang kita ketahui, biner hanya dapat memberikan informasi dengan 2 pilihan logika tiap bit. Hal ini mengakibatkan data yang dapat disimpat memakan memori sebanyak n bit tergantung berapa banyak data yang ingin disimpan. Setiap bit hanya dapat menimpan 2 jenis logika sehingga untuk setiap n bit dapat memberikan 2^n jenis data.

Hal di atas mengakibatkan jumlah memori yang dibutuhkan cukup besar. Contohnya untuk bilangan. Dalam penulisan normal, desimal, 9 jenis angka dapat disimpan dalam 1 bit data. Akan tetapi dalam biner dibutuhkan 4 bit data karena 2^4 baru dapat memuat 16 jenis data.

Masalah kebutuhan memori ini perlu diselesaikan mengingat semakin berkembangnya jaman, makin besar pula data yang perlu disimpan. Hal ini berarti semakin besar pula kebutuhan memori untuk menyimpannya. Masalah ini dapat besarnya data yang perlu disimpan dapat disusahakan dengan 2 cara, yaitu menciptakan memori yang semakin besar dan mencoba untuk mengecilkan data.

Cara pertama yaitu dengan membuat memori yang semakin besar dianggap kurang efisien. Akan lebih baik jika kita memcoba mengkompresi data dan menyimpannya secaraq efisien. Karena itu diperlukan cara untuk melakukan kompresi data yang efisien. Salah satunya dengan menggunakan algoritma Huffman.

Algoritma ini berdasarkan pada graf terutama pohon biner. Dengan algoritma ini data yang dapat tetap disimpan dalam bentuk biner, tetapi dengan lebih efisien. Bit yang diperlukan dapat disesuaikan dengan bobot yang dimiliki dari total keseluruhan data. Dengan itu maka penyimpanan data dapat semakin efisien dan dapat mengurangi kebutuhan momori.

II. ALGORITMA HUFFMAN

Algoritma huffman adalah algoritma yang ditulis oleh D.A. Huffman untuk tugasnya yang berbentuk paper pada saat kuliah di MIT pada tahun 1950. Algoritma ini adalah pengembangan dari algoritma yang berfungsi sebagai algoritma kompresi yang di tulis R.M. Fano dan Claude Shannon. Algoritma ini memanfaatkan graf terutama

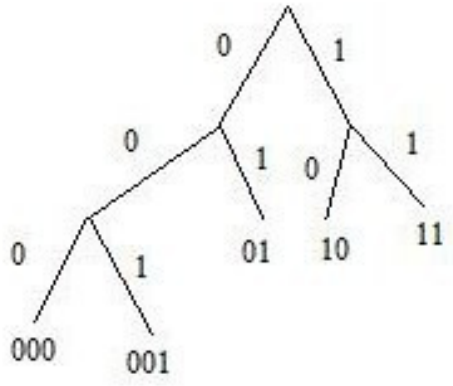
pohon biner.

Graf yang dapat dibuat sebagai pohon merentang dan pohon dapat membentuk pohon biner. Pohon biner sendiri sudah dimanfaatkan secara luas dan memiliki beberapa keunggulan.

Algoritma huffman juga menggunakan pohon biner sebagai dasar pembentukan kode. Pohon biner yang digunakan harus merupakan kumpulan dari kode awalan. Kode awalan adalah himpunan kode sedemikian sehingga tidak ada anggota kumpulan yang merupakan awalan dari anggota lainnya contohnya {000,001,01,10,11}. Contoh lain yang merupakan bukan kode awalan {1,00,01}. Himpunan tersebut bukan kode awalan karena dengan anggotanya dapat membentuk anggota lainnya sehingga dapat menimbulkan keambiguan.

Kode awalan mempunyai kode yang bersesuaian, setiap sisi diberi label 1 atau 0. Pemberian label tiap sisi haruslah taat asas, yaitu setiap sisi harus memiliki nilai sama. Misalkan semua sisi kiri harus diberi nilai 1 saja, atau 0 saja, dan begitu juga sebaliknya. Kode yang didapat berasal dari alamat tiap daun yang terbentuk..

Kode awalan digunakan untuk menghindari keambiguan, karena itu sangat berguna untuk kompresi data yang menggunakan panjang bit yang berbeda-beda. Dengan menggunakan kode awalan, data yang dibuat dalam bentuk biner tidak akan menghasilkan kode yang ambigu meskipun bit yang digunakan berbeda-beda untuk setian data.



Gambar 1 Pohon Biner untuk Kode Awalan

Kode Huffman atau algoritma huffman melakukan kompresi data juga berdasarkan adanya kode awalan. Kode Huffman membuat suatu kamus baru dengan penggunaan bit berbeda-beda untuk setiap data.

Kode Huffman digunakan, sebagai contoh, untuk kompresi dari arsip yang disimpan dalam bentuk kode ASCII. Kode ASCII menggunakan 8 bit untuk menyampaikan 1 karakter. Setiap data karakter dalam ASCII dikodekan menjadi 8 bit kode biner yang memiliki nilai angka tertentu.

Tabel I Contoh Kode ASCII

SIMBOL	KODE ASCII
A	01000001
B	01000010
C	01000011
D	01000100
E	01000101
F	01000110
G	01000111

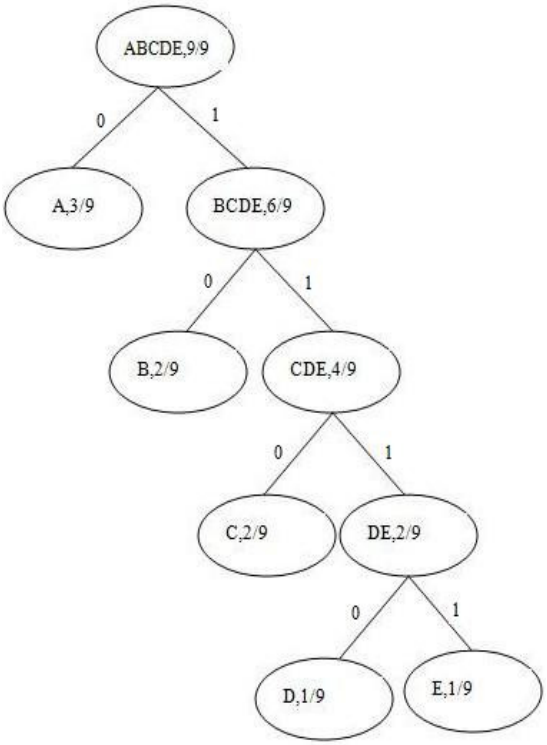
Dalam kode ASCII jumlah bit yang diperlukan adalah 8*n dengan n adalah jumlah karakter. Hal ini memerlukan memori yang cukup besar. Oleh karena itu keinginan untuk meminimalisasi keperluan memori terutama berdasarkan frekuensinya mendasari kemunculan kode Huffman.

Kode Huffman dibuat berdasarkan jumlah kemunculan karakter dalam data. Hal ini menyebabkan penyimpanan data lebih efisien karena data yang sering muncul menggunakan lebih sedikit jumlah bit.

Sebagai contoh untuk membentuk kata CDACBAEBA dengan menggunakan kode ASCII memerlukan 9*8 bit yaitu 81. Kodenya 010000110100010001000001010000110100001001000001010001010100001001000001. Akan tetapi dengan menggunakan kode Huffman dapat dibentuk kode yang lebih sederhana.

Tabel II Tabel Kode Huffman

SIMBOL	KEKERAPAN	PELUANG	KODE HUFFMAN
A	3	3/9	0
B	2	2/9	10
C	2	2/9	110
D	1	1/9	1110
E	1	1/9	1111



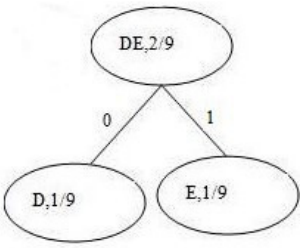
Gambar 2 Pohon Kode Huffman

Dari pohon yang terbentuk, dapat dibuat tabel II. Berdasarkan itu kita dapat membuat rangkaian bit baru berupa 110111001101001111100. Rangkaian bit baru tersebut hanya berjumlah 21 bit. Juah lebih ringan penggunaan memorinya daripa penggunaan tabel ASCII.

Dalam pembuatan kode Huffman ada beberapa cara, tetapi yang akan dibahas di sini hanya 2.

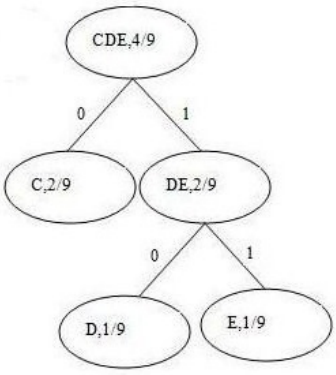
Yang pertama adalah algoritma Huffman statik. Cara ini melakukan perhitungan statistik kemunculan terlebih dahulu. Setelah itu melakukan kalkulasi dan pembuatan pohon biner. Berikut ini langkah-langkah membuat algoritma Huffman statik.

1. Lakukan perhitungan statistik kemunculan tiap karakter.
2. Pilih 2 simbol yang peluang kemunculannya paling kecil, seperti pada gambar 3.



Gambar 3 Pohon dengan 2 simbol

3. Selanjutnya pilih simbol baru yang peluangnya terkecil, setelah dikurangi 2 simbol tersebut seperti pada gambar 4.



Gambar 4 Pohon dengan 3 simbol

4. Lakukan prosedur yang sama terhadap simbol lainnya. Pada akhirnya akan terbentuk akar dari pohon untuk semua simbol dengan peluang 1.

Cara lain pembuatan kode Huffman adalah dengan algoritma Huffman adaptif. Dengan cara ini kita tidak perlu menghitung dahulu statistik secara keseluruhan. Dengan begitu kita dapat melakukan kompresi terhadap berbagai data yang tidak mungkin menghitung dahulu statistiknya, atau data yang tidak tersedia dari awal data statistiknya, seperti gambar dan audio.

Dengan cara ini kita dapat melakukan kompresi pada waktu sebenarnya karena kompresi dapat berjalan sesuai dengan pembuatan data, tidak perlu menunggu semua data selesai baru kemudian di hitung statistiknya. Untuk algoritma encodernya

```
Initialize_model();
while ((c = getc (input)) != eof)
{
        encode (c, output);
        update_model (c);
}
```

Sedangkan untuk dekodernya

```
Initialize_model();
while ((c= decode(input))!=eof)
{
        putc (c, output);
        update_model (c);
}
```

Yang menjadi kunci utama dalam penggunaan algoritma ini adalah *Initialization* yang sama antara encoder dan dekodernya. Selain itu yang harus dilakukan adalah meng-*update* secara terus menerus.

Yang dilakukan dalam *update* adalah meng-*increment* jumlahnya dan mengatur kembali pohon huffman.

Pada algoritma Huffman hal-hal yang mempengaruhi

hasil kompresi ada beberapa. Salah satunya adalah banyaknya variasi dari isi data yang akan dikompres. Semakin banyak variasinya akan mengakibatkan hasil kompresi tidak maksimal. Hal lain yang berpengaruh adalah perbandingan peluang dari isi data tersebut. Semakin besar perbedaan peluang yang ada akan mengakbatkan hasil kompresi dapat semakin efektif.

I I I. KOMPRESI DATA

Kompresi data dilakukan untuk memperkecil memori yang dibutuhkan untuk menimpan data. Kompresi dengan menggunakan algoritma Huffman adalah kompresi dengan sifat loss less, maksudnya kompresi dengan tidak membuat data-data tertentu. Jadi ketika hasil kompresi di-*decode* ulang, tidak ada data yang hilang dan dapat kembali sama persis seperti semula. Berbeda dengan kompresi *lossy.* Dengan cara tersebut, kompresi dilakukan dengan cara membuang data yang dianggap tidak diperlukan atau kurang berguna. Akan tetapi juka dikembalikan ke bentuk semula tidak dapat kembali ke bentuk awal. Contoh kompresi bentuk ini adalah kompresi gambar dan suara (audio).

Kompresi data yang cukup efisien adalah dengan menggunakan algoritma Huffman. Algoritma ini merubah data sesuai dengan rasio kemunculan. Algoritma Huffman dapat digunakan untuk kompresi data berbagai jenis data. Beberapa jenis data yang dapat dikompres dengan algoritma Huffman adalah string, gambar, audio dan lainnya.

Untuk pembahasan aplikasi kali ini yang akan digunakan adalah data berbentuk string dengan 2 jenis algoritma Huffman. Data yang digunakan berbentuk teks dengan beberapa ukuran berbeda. 20kB, 20-40 kB, dan lebih dari 40kB.

Yang akan dianalisis adalah waktu kompresi dan juga keefektifannya. Hasil pengujian dapat dilihat pada tabel III.

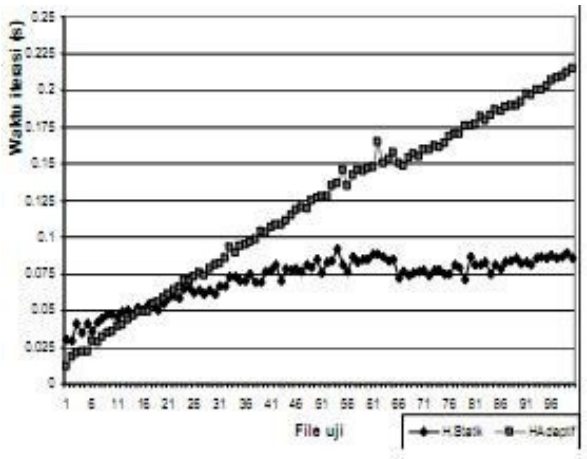
Tabel III Data lama dan rasio kompresi

	Ukuran File	Kompresi	
		Lama(s)	Rasio(%)
Huffman Statik	< 20kB	0,0564441	42,6445
	20-40kB	0,0656619	43,1081
	> 40kB	0,0763654	43,2218
Huffman Adaptif	< 20 kB	0,1235163	43,3365
	20-40kB	0,2234394	43,4659
	< 20 kB	0,3226191	43,4665

Dari tabel dapat terlihat semakin besar file yang di kompres tentu akan memerlukan waktu yang lebi lama pula. File tersebut berupa potongan artikel dari surat kabar. Selain itu bisa dilihat juga bahwa penggunaan algoritma Huffman adaptif memerlukan waktu yang lebih lama daripada algoritma Huffman statik.

Akan tetapi jika dilihat dari rasio hasilnya bisa dilihat bahwa algoritma huffman adaptif menghasilkan efisiensi yang lebig baik. Hal ini memungkan karena pada algoritma adaptif pohon tidak perlu disimpan, tetapi cukup di-*update* secara terus menerus. Sedangkan pada algoritma statik, setelah membentuk tabel dan kamus maka langsung dikompres. Oleh karena itu pada saat ini lebih banyak digunakan algoritma adaptif.

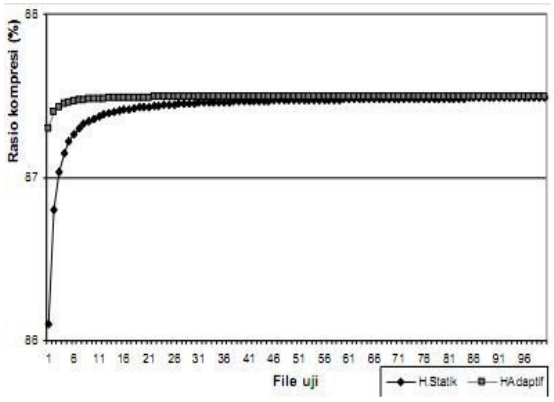
Algoritma adaptif juga berguna untuk file-file besar yang berbentuk selain teks. Meskipun perbedaan rasio tidak tertlalu terlihat mencolok, tetapi ika dihitung jumlah memori yang digunakan akan berdampak cukup besar. Perbandingan waktu dari kedua algoritma dapat dilihat pada gambar 4.



Gambar 4 Perbandingan Waktu

Dapat dilihat untuk penggunaan waktu algoritma adaptif lebih stabil meskipun terus meningkat. Akan tetapi pada file ukuran kecil (1-20kB) masih terjadi fluktuatif dikarenakan ukuran file masih kecil dan pada algoritma adaptif hanya perlu membaca dan meng-*update* sedikit file tanpa perlu membaca dan menghitung dahulu secara keseluruhan baru memulai proses koding, seperti pada algoritma statik.

Pada file yang berukuran besar, algoritma sttatik lebih cepat. Hal ini dikarenakan dengan jumlah variasi isi yang sama,semua variasi file sudah dihitung terlebih dahulu, sehingga setelah perhitungan selesai algoritma statik hanya mengubah file sesuai dengan kamus yang terbentuk. Sedangkan untuk algoritma adaptif, tetap perlu membaca dan meng-*update* data serta pohon yang sudah ada. Karena itu untuk file berukuran besar waktu yang dibutuhkan algoritma Adaptif lebih lama. Sedangkan untuk rasio akan diperlihatkan pada gambar 5.



Gambar 5 Perbandingan Rasio Kedua Algoritma

Untuk perbandingan rasio, algoritma adaptif memiliki efisiensi lebih bagus dan cenderung stabil. Berbeda dengan algoritma statik yang tetap memerlukan memori minimum yang lebih banyak karena perlu menyimpan hasil perhitungan statistik lebih dahulu.

Akan tetapi terdapat suatu anomali terhadap file yang berukuran kecil dan mempunyai variasi relatif kecil. Pada file-file tersebut malah terjadi pembengkakan ukuran. Hal ini disebabkan untuk file berukuran kecil, Algoritma malah memerlukan memori tambahan untuk menyimpan data-data yang diperlukan. Pada algoritma statik diperluakan memori untuk menimpan data statistik serta pohon yang sudah di bentuk, sehingga malah menambah memori meskipun terjadi efisiensi jumlah bit untuk data.

Begitu juga pada algoritma adaptif. Meski efisiensi pengompresan lebih baik,tetapi tetap membutuhkan memori tambahan untuk menyimpan header, data-data lain seperti ukuran file sumber, serta untuk menyimpan pohon sementara yang harus terus di-*update.*

Pada tabel selanjutnya akan diberikan data hasil dekompresi sebagai pembanding.

Tabel IV Lama dan rasio Dekompresi

	Ukuran File	Dekompresi	
		Lama(s)	Rasio(%)
Huffman Statik	< 20kB	0,03934281	-74,363
	20-40kB	0,04878656	-75,777
	> 40kB	0,05559875	-76,13
Huffman Adaptif	< 20 kB	0,11193219	-76,493
	20-40kB	0,18995875	-76,889
	< 20 kB	0,26700437	-76,893

Semua perbedaan tersebut dianalisis menggunakan komputer dengan kemampuan sama. Hal ini perlu dilakukan karena kemampuan komputer tetap berpengaruh terhadap hasil yang didapat. Komputer dengan kemampuan lebih baik tentu menghasilkan hasil yang lebih baik, terutama dari segi waktu.

V. KESIMPULAN

Dari pembahasan dan penjelasan di atas dapat disimpulkan:

1. Algoritma Huffman adalah algoritma yang cukup baik sebagai algoritma untuk kompresi data.
2. Algoritma Huffman statik memiliki keunggulan dari segi waktu dibanding adaptif.
3. Algoritma Huffman adaptif lebih baik dari segi rasio dibanding statik.

REFERENCES

[1] http://en.wikipedia.org/wiki/Adaptive_Huffman_coding waktu akses : 14 desember 2010

[2] http://en.wikipedia.org/wiki/Huffman_coding waktu akses : 14 desember 2010

[3] http://journal.uii.ac.id/index.php/Snati/article/viewFile/1822/1601 waktu akses : 16 desember 2010.

[4] Rinaldi, Munir, *Matematika Diskrit edisi III.* Bandung: Informatika Bandung, 2007.

[5] www.cs.cf.ac.uk/Dave/Multimedia/node212.html waktu akses : 16 Desember 2010

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, 17 Desember 2010

ttd

Timotius Triputra Safei (13509017)