SNATI
Seminar Nasional
Aplikasi Teknologi Informasi 2004
Yogyakarta, 19 Juni 2004

Kompresi Data Menggunakan Algoritme Huffman

Julio Adisantoso, Danny Dimas Sulistio, Bib Paruhum Silalahi
Departemen Ilmu Komputer Institut Pertanian Bogor
Kampus IPB Baranangsiang, Jalan Raya Pajajaran Bogor
Telp./Faks. (0251) 356653
e-mail: julio@bima.ipb.ac.id

Abstract

Text compression algorithms are normally defined in terms of a source alphabet of 8-bit ASCII codes. Huffman algorithm is the most popular methods of text compression. This research used static and adaptif Huffman algorithms to compress text data, and also compare it. Variation of character occurs will decrease compression ratio. Iteration time of static Huffman algorithm for compress and decompress is faster than adaptif Huffman algorithm, but performance of adaptif Huffman algorithm is best.

Keywords: *text compression, static and adaptif huffman algorithm.*

1. Pendahuluan

Pada saat ini kebutuhan akan informasi sangatlah diperlukan oleh masyarakat umum. Dengan semakin banyaknya informasi yang perlu disimpan secara digital, secara automatis akan meningkatkan keperluan untuk menyediakan media penyimpanan data yang lebih besar lagi. Oleh karena itu diperlukan suatu alternatif mekanisme penyimpanan data sehingga dengan media penyimpanan yang ada, kita dapat menyimpan data sebanyak-banyaknya.

Pemampatan atau kompresi data merupakan salah satu metode untuk memperkecil ruang penyimpanan data pada suatu media penyimpanan. Selain berguna dalam penyimpanan data, kompresi data dapat membantu memperkecil ukuran data yang ditransmisikan di dalam suatu media jaringan, seperti internet sehingga memperkecil waktu transfer data.

Salah satu metode kompresi data yang telah banyak dikenal ialah dengan menggunakan metode Huffman. Algoritma Huffman merupakan salah satu pelopor lahirnya kompresi data, sehingga ukuran data yang perlu disimpan menjadi lebih kecil dibandingkan dengan ukuran data sebenarnya.

Penelitian mengenai kompresi terhadap data teks menggunakan algoritma Huffman telah dilakukan sebelumnya oleh Hutasoit (2001) mengenai pengaruh n-gram dalam pembentukan kode Huffman pada file teks berbahasa Indonesia dan Layungsari (2003) mengenai implementasi kompresi multi tahap menggunakan algoritma Huffman pada file teks. Kesimpulan dari Hutasoit (2001) adalah pengaruh n-gram dapat menghasilkan rasio kompresi yang lebih baik yang ditunjukan dengan rasio kompresi untuk *tree* yang monogram lebih kecil dibandingkan dengan *tree* digram. Sedangkan Layungsari (2003) menyimpulkan bahwa hasil kompresi file teks menggunakan kompresi multi tahap Huffman menunjukan hasil yang tidak memuaskan. Hal ini dikarenakan file hasil kompresi oleh kompresi tahap pertama telah menghasilkan kode prefiks yang optimal.

Penelitian yang menelaah tentang pengembangan algoritme Huffman itu sendiri belum banyak dilakukan. Hal ini diperlukan agar semakin sempurna metode kompresi data yang dilakukan. Salah satu pengembangan algoritme Huffman statik adalah algoritme adaptif. Oleh karena itu penelitian ini bertujuan untuk menelaah algoritma Huffman adaptif dibanding dengan algoritma Huffman statik pada mekanisme kompresi data berbasis teks.

2. Landasan Teori

Kompresi data dapat dilihat sebagai kumpulan teori informasi dimana tujuan utamanya adalah untuk memperkecil ukuran data yang akan ditransmisikan (Lelewer,1987). Secara sederhana, karakteristik dari kompresi data dapat dianalogikan sebagai sebuah proses untuk mengubah sebuah string yang merupakan kumpulan karakter menjadi sebuah string yang baru dengan informasi yang sama namun dengan lebar atau ukuran yang lebih kecil. Suatu cara untuk mengembalikan data yang telah terkompresi menjadi seperti sedia kala dikenal dengan istilah dekompresi data.

Secara garis besar kompresi data dapat dikelompokkan ke dalam dua metode yaitu metode kompresi *lossy* dan metode kompresi *lossless*. Metode kompresi *lossy* adalah suatu metode kompresi data dimana data yang telah terkompresi apabila dikembalikan ke dalam bentuk semula akan terdapat beberapa informasi yang hilang. Contoh dari metode ini adalah pemampatan data gambar dan suara. Sedangkan metode kompresi *lossless* merupakan suatu metode kompresi data dimana data yang telah terkompresi akan dapat dikembalikan seperti semula tanpa ada informasi yang hilang. Implementasi dari metode ini yaitu pada pemampatan data teks atau dokumen ASCII.

2.1 Model Kompresi Data

Pada metode kompresi *lossless* terdapat dua buah model utama yaitu metode *lossless* dengan menggunakan model statistik dan model kamus. Pada model statistik, kompresi data diawali dengan melakukan perhitungan setiap karakter yang ada di dalam file, kemudian dengan statistik karakter yang ada akan dilakukan pengkodean karakter dengan representasi lain. Representasi tersebut apabila dibandingkan dengan karakter asli diharapkan akan lebih kecil ukurannya. Model ini digunakan pada algoritma Huffman dan algoritma Aritmatik.

Sedangkan untuk model kamus, kompresi data diawali dengan melakukan perhitungan setiap string yang terdapat di dalam file. String-string tersebut akan disusun seperti sebuah indeks dan string-string tersebut akan disimbolkan dengan sebuah representasi yang unik. Model ini digunakan pada algoritma Lempel-Ziv dan turunannya (LZW,LZSS, LZRW, dan sebagainya).

Pada setiap model kompresi, dilakukan proses konversi simbol dari representasi string menjadi representasi kode lain, dan sebaliknya. Ada dua alat konversi dalam hal ini, yaitu mesin enkoder dan mesin dekoder. Mesin enkoder merupakan sebuah alat yang menjalankan suatu mekanisme untuk melakukan konversi simbol dari suatu representasi string menjadi suatu representasi kode lainnya yang bersifat unik. Mekanisme yang dilakukan oleh alat ini sering dikenal dengan istilah pengkodean (*encoding*). Sedangkan mesin dekoder merupakan sebuah alat yang menjalankan suatu mekanisme untuk mengembalikan representasi unik dari suatu string menjadi sebuah representasi string seperti sedia kala. Mekanisme yang dilakukan oleh alat ini sering dikenal dengan istilah pendekodean (*decoding*).

2.2 Algoritme Huffman

Algoritma Huffman diperkenalkan oleh D.A. Huffman pada *paper* yang ia tulis sebagai salah satu tugas kuliahnya di MIT pada tahun 1950. Algoritma ini merupakan pengembangan lebih lanjut dari algoritma kompresi yang dilakukan oleh Claude Shannon dan R.M. Fano pada tahun yang sama. Ide dasar dari algoritma ini adalah membuat kode dengan representasi bit yang lebih pendek untuk karakter ASCII yang sering muncul didalam file dan membuat kode dengan representasi bit yang lebih panjang untuk karakter ASCII yang jarang muncul didalam file. Di dalam perkembangannya algoritma Huffman terpecah menjadi dua buah kategori yaitu statik dan adaptif.

2.2.1 Algoritme Huffman Statik

Algoritma Huffman Statik menggunakan kemungkinan kemunculan dari setiap karakter yang telah ditetapkan pada awal pengkodean dan kemungkinan kemunculan karakter tersebut juga dapat diketahui baik oleh enkoder maupun dekoder.

Cara kerja dari algoritma Huffman Statik untuk mengkompresi data yaitu dengan cara sebagai berikut:

a. Hitung jumlah karakter yang muncul di dalam file, sehingga masing-masing karakter yang ada akan memiliki bobot sesuai dengan banyaknya karakter tersebut didalam file. Kemudian susunlah suatu pohon biner yang dibangun berdasarkan bobot karakter yang ada. Inti dari pembangunan pohon biner adalah menggabungkan dua buah karakter dengan tingkat kemunculan (bobot) yang lebih kecil, kemudian membangkitkan satu buah node *parent* yang memiliki bobot gabungan dari kedua karakter tersebut.
b. Lakukan pengkodean (*encoding*) karakter yang ada didalam file menjadi suatu representasi bit sesuai dengan urutan pada pohon biner yang telah dibangun.

2.2.2 Algoritme Huffman Adaptif

Algoritma Huffman Adaptif adalah algoritme dengan kemungkinan kemunculan dari setiap simbol tidak dapat ditentukan dengan pasti selama pengkodean. Hal ini disebabkan oleh perubahan pengkodean secara dinamis berdasarkan frekuensi dari simbol yang telah diolah sebelumnya.

Algoritma Huffman Adaptif merupakan pengembangan lebih lanjut dari algoritma Huffman. Ide dasar dari algoritma ini adalah meringkas tahapan algoritma Huffman tanpa perlu menghitung jumlah karakter keseluruhan dalam membangun pohon biner. Algoritma ini dikembangkan oleh trio Faller, Gallager, dan Knuth dan kemudian dikembangkan lebih lanjut oleh Vitter, sehingga tercipta dua buah algorima Huffman adaptif yaitu algoritma FGK dan algoritma V. Secara umum algoritma Huffman Adaptif adalah sebagai berikut:

1. Pada mesin enkoder:

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

2. Pada mesin dekoder:

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

3. Metode Penelitian

Secara umum kompresi data menggunakan algoritma Huffman statik diawali dengan membaca dahulu file teks yang akan dikompresi. Hal ini dilakukan untuk mendapatkan informasi mengenai jumlah dari masing-masing karakter yang terdapat didalam file. Setelah itu kita akan membangun sebuah pohon biner yang dibangun dengan kriteria yang telah dijelaskan sebelumnya. Kemudian kita akan mengubah representasi dari masing-masing karakter yang ada di dalam file sesuai dengan representasi yang dihasilkan oleh pohon biner yang dituliskan pada file baru. Proses penulisan pada file hasil kompresi diawali dengan penulisan pohon biner yang kemudian disambung dengan isi file yang telah dikodekan.

Untuk dekompresi data, yang pertama kali dilakukan adalah membangun pohon Huffman yang merepresentasikan statistik data. Kemudian dilakukan pembacaan setiap bit sampai ditemukan node *leaf* yang sesuai. Hal ini dilakukan terus menerus sampai bit terakhir.

Sedangkan proses kompresi data dengan menggunakan algoritma Huffman Adaptif diawali dengan membentuk sebuah pohon biner yang berisi sebuah node NYT dengan bobot nol. Kemudian dilakukan pembacaan file sumber setiap karakter. Apabila ditemui sebuah karakter yang belum didefinisikan didalam pohon biner, maka kode NYT akan dituliskan kedalam file baru disambung dengan kode ASCII dari karakter baru tersebut. Setelah itu akan dibuatkan sebuah node yang berisi informasi tentang karakter baru dengan bobot satu dan kemudian pohon biner dimodifikasi ulang. Setelah itu dibaca karakter selanjutnya sampai akhir file. Pada saat dibaca karakter yang telah didefinisikan pada pohon, maka bobot karakter tersebut ditambah satu, kemudian pohon akan dimodifikasi ulang. Modifikasi dari pohon biner berlangsung selama proses pembacaan karakter. Sehingga pada file baru tidak perlu tambahkan informasi mengenai pohon biner.

3.1 Bahan Penelitian

Data yang digunakan adalah data teks dengan berbagai ukuran file, yaitu kurang dari 20.000 *byte*, 20.000 - 40.000 *byte*, dan lebih dari 40.000 *byte*. File teks tersebut berisi potongan tajuk rencana harian sebuah media cetak *online* (Media Indonesia) yang beredar antara bulan Juni sampai November 2003. Selain itu digunakan sebuah mekanisme untuk membangkitkan karakter dengan rentang bervariasi dari satu buah variasi karakter.

3.2 Analisis Data

Dalam penelitian ini dilakukan beberapa pengujian yang antara lain:

a. Melihat pola rasio pemampatan dan lamanya eksekusi algoritme dengan menggunakan data yang berasal dari potongan artikel.
b. Melakukan simulasi dengan menggunakan file teks yang hanya memiliki sebuah variasi karakter kemudian dilihat pola rasio pemampatan dan lamanya eksekusi dalam rentang 1.000 *byte*, 2.000 *byte*, 3.000 *byte* sampai 100.000 *byte*.
c. Mengamati rasio dan lamanya eksekusi algoritma pada file yang memiliki lima buah karakter yang berbeda, dan 256 karakter yang berbeda.
d. Penarikan kesimpulan menggunakan uji Analisis Keragaman (ANOVA)

Percobaan yang telah dirancang selanjutnya diuji coba dan dilakukan analisis terhadap kinerja algoritma yang dibatasi dengan melihat rasio pemampatan dan waktu proses kompresi. Rasio pemampatan (R) didefinisikan sebagai:

$$R = \frac{f_a - f_b}{f_a}$$

dimana f_a adalah ukuran file awal, dan f_b adalah ukuran file hasil kompresi.

4. Hasil Penelitian

4.1 Ukuran File

Lama waktu kompresi file berukuran kurang dari 20.000 byte oleh Huffman Statik berkisar antara 0,05 detik sampai 0,06 detik, sedangkan Huffman adaptif memerlukan waktu sekitar 0,11 detik sampai 0,13 detik dengan pola menaik untuk ukuran file yang semakin besar.

Rasio kompresi kedua algoritma menunjukan nilai yang fluktuatif. Hal ini disebabkan perbedaan banyaknya variasi karakter yang muncul dari kesepuluh file yang digunakan. Pada

lamanya dekompresi Huffman statik memerlukan waktu antara 0,035 detik sampai 0,04 detik, sedangkan Huffman adaptif memerlukan waktu antara 0,105 detik sampai 0,12 detik.

Untuk file berukuran antara 20.000 – 40.000 bytes, lamanya waktu kompresi yang diperlukan oleh Huffman Statik berkisar antara 0,05 detik sampai 0,06 detik, sedangkan Huffman adaptif memerlukan waktu sekitar 0,11 detik sampai 0,13 detik dengan pola menaik untuk ukuran file yang semakin besar. Rasio kompresi kedua algoritma menunjukan nilai yang fluktuatif. Hal ini disebabkan perbedaan banyaknya variasi karakter yang muncul dari kesepuluh file yang digunakan. Pada lamanya dekompresi Huffman statik memerlukan waktu antara 0,035 detik sampai 0,04 detik, sedangkan Huffman adaptif memerlukan waktu antara 0,105 detik sampai 0,12 detik.

Sedangkan untuk file berukuran lebih dari 40.000 bytes, hasil kompresi kurang lebih sama dengan percobaan sebelumnya yaitu iterasi yang diperlukan oleh algoritma Huffman adaptif lebih lambat dibandingkan algoritma Huffman statik, namun dengan rasio kompresi yang lebih baik.

Tabel 1 menunjukkan bahwa waktu iterasi untuk kompresi dan dekompresi Huffman statik lebih cepat dibandingkan waktu iterasi kompresi dan dekompresi Huffman adaptif. Namun rasio pemampatan Huffman adaptif lebih besar dibandingkan rasio pemampatan Huffman Statik. Hal ini menjadikan file kompresi yang dihasilkan oleh Huffman Adaptif akan lebih kecil ukurannya dibandingkan dengan file kompresi yang dihasilkan oleh Huffman Statik. Selain itu dapat kita lihat semakin besar ukuran file yang akan dikompresi berimpikasi pada lama iterasi yang semakin lama.

Tabel 1. Rata-rata lama dan rasio kompresi

	Ukuran file	***Kompresi***		***Dekompresi***	
		Lama(s)	***Rasio(%)***	***Lama(s)***	***Rasio(%)***
Huffman Statik	< 20 KB	0,0564441	42,6445	0,03934281	-74,363
	20 s/d 40 KB	0,0656619	43,1081	0,04878656	-75,777
	> 40 KB	0,0763654	43,2218	0,05559875	-76,13
Huffman Adaptif	< 20 KB	0,1235163	43,3365	0,11193219	-76,493
	20 s/d 40 KB	0,2234394	43,4659	0,18995875	-76,889
	> 40 KB	0,3226191	43,4665	0,26700437	-76,893

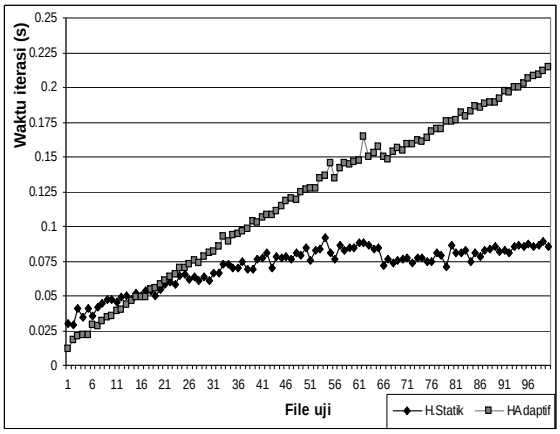
4.2 Variasi Karakter

Tujuan dari percobaan ini adalah untuk mendapatkan rasio dan lamanya waktu eksekusi dari kedua algoritma untuk file yang memiliki karakter dengan peluang kemunculan adalah satu. Hasil iterasi kedua algoritma pada saat ditampilkan akan dibedakan dengan warna, pada hasil iterasi algoritma Huffman Statik ditunjukkan dengan warna hitam dan untuk hasil algoritma Huffman Adaptif ditunjukkan dengan warna abu tua.

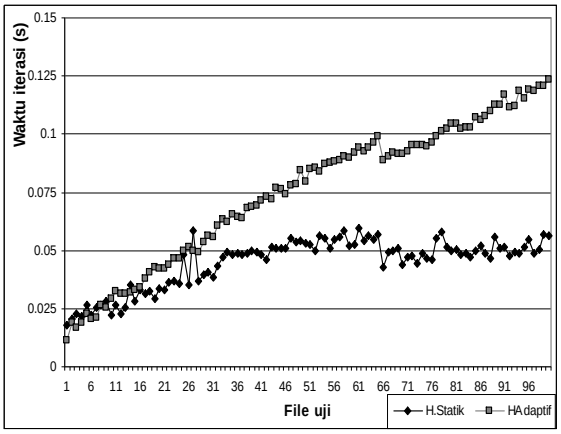
Lama iterasi untuk kompresi dan dekompresi Huffman Statik untuk satu variasi karakter cenderung lebih cepat dibandingkan dengan Huffman Adaptif, walaupun pada rentang file 1.000 *byte* sampai 20.000 *byte* masih terjadi fluktuasi disebabkan faktor ukuran file yang terlalu kecil dan iterasi yang cukup cepat (Gambar 1 dan 2). Sedangkan rasio kompresi Huffman Adaptif lebih baik dibandingkan rasio kompresi Huffman Statik, dengan kecenderungan semakin stabil seiring semakin besarnya ukuran file (Gambar 3).

Pada kompresi file yang berisi 5 karakter terjadi suatu anomali (Tabel 2), dimana hasil kompresi yang seharusnya mengecil menjadi membesar. Pada algoritma Huffman Statik hal ini disebabkan penambahan karakter diawal, berupa *header* terkompresi tidaknya file (4 *byte*), 1 *byte* kode CRC, 4 *byte* untuk menyimpan ukuran file sumber, dan 2 *byte* untuk menyimpan banyaknya karakter. Ditambah kebutuhan untuk menyimpan tree berupa 2 *byte* untuk masing-masing karakter, disambung representasi *tree* dan representasi hasil kompresi.

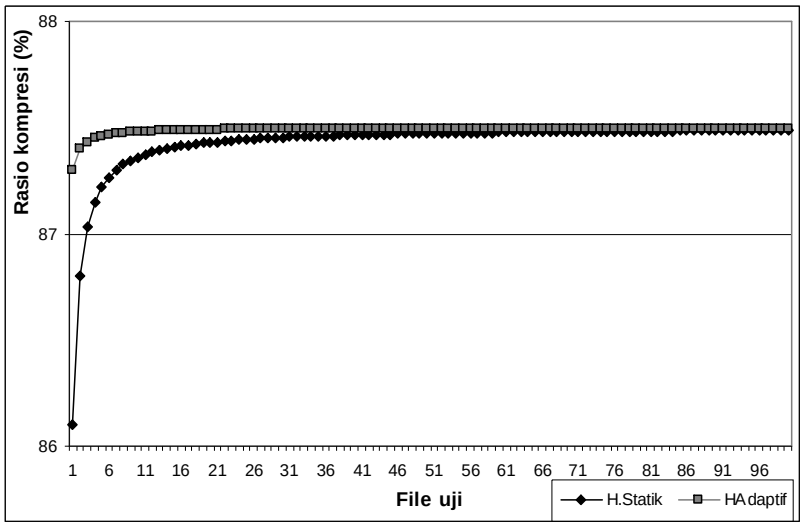
Sedangkan pada algoritma Huffman Adaptif mengalami pembesaran ukuran file karena selain menyimpan setiap variasi karakter yang muncul, akan disimpan pula hasil pengkodean berdasarkan *tree*.



Gambar 1. Lama kompresi kedua algoritma pada satu karakter



Gambar 2. Lama dekompresi kedua algoritma pada satu karakter



Gambar 3. Rasio kompresi kedua algoritma pada satu karakter

Karena *tree* yang dibentuk pada Huffman Adaptif tidak perlu disimpan, hal ini menjadikan rasio kompresi Huffman Adaptif lebih baik dibandingkan dengan rasio kompresi Huffman Statik.

Pada percobaan dengan file berisi 256 karakter terjadi pula anomali seperti halnya yang terjadi pada percobaan 5 karakter. Seperti yang dapat dilihat pada Tabel 3, rasio kompresi kedua algoritma menunjukan nilai negatif yang berarti hasil kompresi mengalami pembengkakan ukuran.

Tabel 2. Kompresi file 5 buah karakter

Huffman Statik	Lama iterasi (detik)	0,01328
	Rasio pampat (%)	-400
	Ukuran hasil (byte)	25
Huffman Adaptif	Lama iterasi (detik)	0,008969
	Rasio pampat (%)	-40
	Ukuran hasil (byte)	7

Tabel 3. Kompresi file 256 karakter

Huffman Statik	Lama iterasi (detik)	0,025
	Rasio pampat (%)	-304,297
	Ukuran hasil (byte)	1035
Huffman Adaptif	Lama iterasi (detik)	0,011344
	Rasio pampat (%)	-100,781
	Ukuran hasil (byte)	514

Pada algoritma Huffman Statik lebih disebabkan karena besarnya tempat yang diperlukan untuk menyimpan statistik karakter. Karena pada percobaan ini digunakan 256 karakter yang berbeda, maka secara otomatis akan diperlukan minimal 512 *byte* untuk menyimpan informasi karakter. Hal ini belum termasuk penyimpanan *header*, dan penyimpanan hasil pengkodean file sumber. Sedangkan pada algoritma huffman statik selain menyimpan 256 *byte* yang merepresentasikan karakter yang berbeda, perlu juga disimpan hasil pengkodean file sumber.

5. Kesimpulan

Semakin variatif karakter yang muncul, akan memperkecil rasio kompresi yang dihasilkan, baik oleh algoritma Huffman Statik, maupun pada algoritma Huffman Adaptif. Rasio kompresi terbaik terjadi pada file yang memiliki karakter dengan peluang kemunculan mendekati 1 (bobot karakter hampir sebesar ukuran file). Rasio terbaik yang didapat selama penelitian yaitu sebesar 87,489 %. Rasio kompresi terburuk terjadi pada file yang memiliki variasi karakter besar dan pada file yang berukuran kecil. Hal ini ditandai dengan terjadinya pembesaran ukuran file hasil dibandingkan ukuran file awal.

Dengan menggunakan metode kompresi apapun memiliki *trade off*, dimana kita akan diberi pilihan hasil maksimum dengan iterasi yang lama atau iterasi yang sangat cepat namun dengan hasil yang biasa.

Hasil percobaan yang terdapat pada pembahasan menunjukan bahwa waktu iterasi yang diperlukan oleh algoritma Huffman Statik untuk melakukan kompresi dan dekompersi adalah cenderung lebih kecil dibandingkan dengan yang dilakukan oleh algoritma Huffman Adaptif. Namun untuk hasil kompresi terlihat unjuk kerja Huffman Adaptif adalah lebih baik dibandingkan Huffman Statik.

6. Saran

Pada saat mengimplementasikan algoritma Huffman Adaptif digunakan prosedur yang menjadikan kompleksitas algoritma Huffman Adaptif menjadi O(n.m). Seharusnya kompleksitas Huffman adaptif hampir sama dengan algoritma Huffman Statis yaitu O(n lg m). Pengembangan selanjutnya untuk perbandingan kompresi Huffman statik dan adaptif ini dapat diterapkan pada mekanisme pengiriman paket data terkompresi pada media jaringan.

Daftar Pustaka

Cormen, T.H., Leiserson, Charles, E., dan Rivest, R.L. (1990). *Introduction to Algorithms*. Mc Graw Hill Company.

Huffman, D.A. (1952). *A Method for the Construction of Minimum-Redundancy Codes*. Proc. IRE, vol. 40, pp. 1098–1101, Sept. 1952.

Hutasoit, Y.E. (2001). Huffman Coding Untuk Kompresi Data Teks Berbahasa Indonesia. Skripsi. Jurusan Ilmu Komputer Fakultas MIPA IPB. Bogor, Indonesia.

Layungsari. (2003). Penyempurnaan dan Implementasi *Software* Kompresi Multi Tahap Menggunakan Huffman *Coding*. Skripsi. Jurusan Ilmu Komputer Fakultas MIPA IPB. Bogor, Indonesia.

Lelewer, D.A. dan Hirschberg D.S. (1987). *Data Compression.* ACM Computing Surveys 19(1987) 261-296.

Moore, D.S. (1994). *The Basic Practice of Statistics*. ISBN 0-7167-2628-9. W.H. Freeman and Company. New York

Vitter, J.S. (1987). *Design and Analysis of Dynamic Huffman Codes.* Journal of ACM, 344 (October 1987),825-845.

Vitter, J.S. (1989). *ALGORITHM 673:Dynamic Huffman Codes.* ACM Transactions on Mathematical Software, 15(2), 158-167.