Perbandingan Kinerja Berbagai Algoritma Kompresi pada Berbagai Tipe File

Obbie Hadrian – NIM : 13506027
Jurusan Teknik Informatika ITB, Bandung, email: obbie@students.itb.ac.id

Abstrak

Dari berbagai algoritma kompresi yang ada dipilih 3 algoritma kompresi yang dianggap mewakili berbagai karakteristik algoritma kompresi, yaitu algoritma Huffman, LZW (Lempel-Ziv-Welch), dan DMC (Dynamic Markov Compression) untuk dibandingkan kinerjanya. Perbandingan kinerja ketiga algoritma tersebut diukur berdasarkan rasio hasil kompresi dan kecepatan kompresi saat diujikan terhadap 12 golongan kasus. Disimpulkan pada bidang rasio hasil kompresi, secara rata-rata DMC merupakan yang terbaik dan Huffman merupakan yang terburuk, sedangkan dari sisi kecepatan kompresi, LZW merupakan yang terbaik dan DMC merupakan yang terburuk. Terdapat beberapa jenis file yang tidak tepat untuk dikompresi dengan metode tertentu karena file hasil kompresinya justru berukuran lebih besar.

Kata Kunci : rasio hasil kompresi, kecepatan kompresi, algoritma kompresi.

1. Pendahuluan

Kompresi ialah proses pengubahan sekumpulan data menjadi suatu bentuk kode untuk menghemat kebutuhan tempat penyimpanan dan waktu untuk transmisi data [1]. Saat ini terdapat berbagai tipe algoritma kompresi [2,3], antara lain: Huffman, 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.

Berdasarkan tipe peta kode yang digunakan untuk mengubah pesan awal (isi file input) menjadi sekumpulan codeword, metode kompresi terbagi menjadi dua kelompok, yaitu :

- Metode statik : menggunakan peta kode yang selalu sama. Metode ini membutuhkan dua fase (two-pass): fase pertama untuk menghitung probabilitas kemunculan tiap simbol/karakter dan menentukan peta kodenya, dan fase kedua untuk mengubah pesan menjadi kumpulan kode yang akan ditransmisikan.
  Contoh: algoritma Huffman statik.

- Metode dinamik (adaptif) : menggunakan peta kode yang dapat berubah dari waktu ke waktu. Metode ini disebut adaptif karena peta kode mampu beradaptasi terhadap perubahan karakteristik isi file selama proses kompresi berlangsung. Metode ini bersifat onepass, karena hanya diperlukan satu kali pembacaan terhadap isi file.
  Contoh: algoritma LZW dan DMC.

Berdasarkan teknik pengkodean/pengubahan simbol yang digunakan, metode kompresi dapat dibagi ke dalam tiga kategori, yaitu :

- Metode symbolwise : menghitung peluang kemunculan dari tiap simbol dalam file input, lalu mengkodekan satu simbol dalam satu waktu, dimana simbol yang lebih sering muncul diberi kode lebih pendek dibandingkan simbol yang lebih jarang muncul, contoh: algoritma Huffman.

- Metode dictionary : menggantikan karakter/fragmen dalam file input dengan indeks lokasi dari karakter/fragmen tersebut dalam sebuah kamus (dictionary), contoh: algoritma LZW.

- Metode predictive : menggunakan model finite-context atau finite-state untuk memprediksi distribusi probabilitas dari simbol-simbol selanjutnya; contoh: algoritma DMC.

Ada beberapa faktor yang sering menjadi pertimbangan dalam memilih suatu metode kompresi yang tepat, yaitu kecepatan kompresi, sumber daya yang dibutuhkan (memori, kecepatan PC), ukuran file hasil kompresi, besarnya redundansi, dan kompleksitas algoritma. Tidak ada metode kompresi yang paling efektif untuk semua jenis file.

Dalam penelitian ini, diimplementasikan tiga buah metode kompresi, yaitu algoritma Huffman, LZW, dan DMC, yang masing-masing mewakili sebuah kategori teknik pengkodean, dalam bentuk sebuah perangkat lunak. Ketiga metode ini diujikan untuk mengkompresi dan mendekompresi berbagai tipe dan

ukuran file yang berbeda. Lalu dilakukan analisis statistik untuk membandingkan kinerja setiap metode berdasarkan dua faktor, yaitu rasio/perbandingan ukuran file hasil kompresi terhadap file asli dan kecepatan kompresinya.

2. Dasar Teori

Dalam makalah ini, algoritma kompresi yang dibandingkan ada 3, yaitu:

1. Algoritma Huffman
2. Algoritma LZW
3. Algoritma DMC

Setiap algoritma ini memiliki keunggulan dan kekurangannya masing-masing serta karakteristik tersendiri.

2.1 Algoritma Huffman

Algoritma Huffman, yang dibuat oleh seorang mahasiswa MIT bernama David Huffman, merupakan salah satu metode yang 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.

Algoritma Huffman secara lengkap:

1. Pass pertama
   Baca (scan) file input dari awal hingga akhir untuk menghitung frekuensi kemunculan tiap karakter dalam file. n <- jumlah semua karakter dalam file input. T <- daftar semua karakter dan nilai peluang kemunculannya dalam file input. Tiap karakter menjadi node daun pada pohon Huffman.

2. Pass kedua
   Ulangi sebanyak (n -1) kali :
   i. Item m1 dan m2 <- dua subset dalam T dengan nilai peluang yang terkecil.
   ii. Gantikan m1 dan m2 dengan sebuah item {m1,m2} dalam T, dimana nilai peluang dari item yang baru ini adalah penjumlahan dari nilai peluang m1 dan m2.
   iii. Buat node baru {m1, m2} sebagai father node dari node m1 dan m2 dalam pohon Huffman.

3. T sekarang tinggal berisi satu item, dan item ini sekaligus menjadi node akar pohon Huffman.

Panjang kode untuk suatu simbol adalah jumlah berapa kali simbol tersebut bergabung dengan item lain dalam T.

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

01000001 01000010 01000001 0100001
A B A C

0100001 01000100 01000001
C D A

Untuk mengurangi jumlah bit yang dibutuhkan, panjang kode untuk tiap karakter dapat dipersingkat, terutama untuk karakter yang frekuensi kemunculannya besar. Pada *string* di atas, frekuensi kemunculan A = 3, B = 1, C = 2, dan D = 1, sehingga dengan menggunakan algoritma di atas diperoleh kode Huffman seperti pada Tabel 1.

Tabel 1. Kode Huffman untuk "ABACCDA"

Simbol	Frekuensi	Peluang	Kode Huffman
A	3	3/7	0
B	1	1/7	110
C	2	2/7	10
D	1	1/7	111

Dengan menggunakan kode Huffman ini, *string* "ABACCDA" 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 dekompresi dapat dilakukan dengan mudah. Contoh: saat membaca kode bit pertama dalam rangkaian bit "**0**11001010110", 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".

2.2 Algoritma LZW

Lempel-Ziv-Welch (LZW) adalah algoritma kompresi lossless universal yang diciptakan Abraham Lempel, Jacob Ziv, dan Terry Welch. Diciptakan oleh Welch pada 1984 sebagai implementasi dari pengembangan algoritma LZ78 yang diciptakan oleh Lempel dan Ziv pada 1978. Algoritma ini dirancang untuk cepat dalam

implementasi teapi biasanya tidak optimal karena hanya melakukan analisis terbatas pada data. Algoritma ini melakukan kompresi dengan menggunakan *dictionary*, di mana fragmen-fragmen teks digantikan dengan indeks yang diperoleh dari sebuah "kamus". Prinsip sejenis juga digunakan dalam kode Braille, di mana kode-kode khusus digunakan untuk merepresentasikan kata-kata yang ada.

Pendekatan ini bersifat adaptif dan efektif karena banyak karakter dapat dikodekan dengan mengacu pada string yang telah muncul sebelumnya dalam teks. Prinsip kompresi tercapai jika referensi dalam bentuk pointer dapat disimpan dalam jumlah bit yang lebih sedikit dibandingkan string aslinya.

Algoritma LZW secara lengkap:

1. *Dictionary* diinisialisasi dengan semua karakter dasar yang ada : {'A'..'Z','a'..'z','0'..'9'}.

2. *P* <- karakter pertama dalam *stream* karakter.

3. C <- karakter berikutnya dalam stream karakter.

4. Apakah string (P + C) terdapat dalam dictionary ?
   - Jika ya, maka P <- P + C (gabungkan P dan C menjadi string baru).
   - Jika tidak, maka :
     i. Output sebuah kode untuk menggantikan string P.
     ii. Tambahkan string (P + C) ke dalam dictionary dan berikan nomor/kode berikutnya yang belum digunakan dalam dictionary untuk string tersebut.
     iii. P <- C.

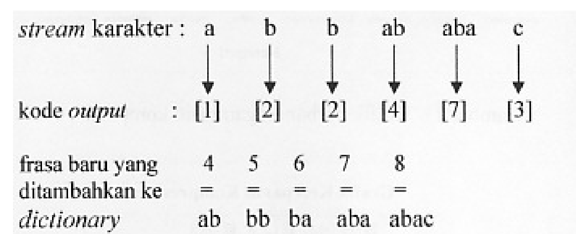
5. Apakah masih ada karakter berikutnya dalam stream karakter ?
   i. Jika ya, maka kembali ke langkah 2.
   ii. Jika tidak, maka output kode yang menggantikan string P, lalu terminasi proses (stop).

Sebagai contoh, string "ABBABABAC" akan dikompresi dengan LZW. Isi dictionary pada awal proses diset dengan tiga karakter dasar yang ada: "A", "B", dan "C". Tahapan proses kompresi ditunjukkan pada Tabel 2.

Tabel 2. Tahapan proses kompresi LZW

Langkah	Posisi	Karakter	*Dictionary*	Output
1	1	A	[4] A B	[1]
2	2	B	[5] B B	[2]
3	3	B	[6] B A	[2]
4	4	A	[7] A B A	[4]
5	6	C	[8] A B AC	[7]
6	9	C	---	[3]

Kolom posisi menyatakan posisi sekarang dari stream karakter dan kolom karakter menyatakan karakter yang terdapat pada posisi tersebut. Kolom dictionary menyatakan string baru yang sudah ditambahkan ke dalam dictionary dan nomor indeks untuk string tersebut ditulis dalam kurung siku. Kolom output menyatakan kode output yang dihasilkan oleh langkah kompresi. Hasil proses kompresi ditunjukkan pada Gambar 1.



Gambar 1. Hasil Proses Kompresi

Proses dekompresi pada LZW dilakukan dengan prinsip yang sama seperti proses kompresi. Algoritma diberikan sebagai berikut:

1. *Dictionary* diinisialisasi dengan semua karakter dasar yang ada : {'A'..'Z','a'..'z','0'..'9'}.

2. *CW* <- kode pertama dari stream kode (menunjuk ke salah satu karakter dasar).

3. Lihat dictionary dan output string dari kode tersebut (string.CW) ke stream karakter.

4. PW <- CW; CW <- kode berikutnya dari stream kode.

5. Apakah string.CW terdapat dalam dictionary ?
   - Jika ada, maka :
     1. output string.CW ke stream karakter
     2. P <- string.PW
     3. C <- karakter pertama dari string.CW
     4. tambahkan string (P+C) ke dalam dictionary

- Jika tidak, maka:
    1. P <- string.PW
    2. C <- karakter pertama dari string.CW
    3. output string (P+C) ke stream karakter dan tambahkan string tersebut ke dalam dictionary (sekarang berkorespondensi dengan CW);

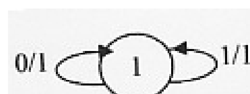
6. Apakah terdapat kode lagi di stream kode ?
    - Jika ya, maka kembali ke langkah 4.
    - Jika tidak, maka terminasi proses (*stop*).

Tabel 4. Tahapan proses dekompresi LZW

Langkah	Posisi	Output	*Dictionary*
1	1	A	---
2	2	B	[4] A B
3	2	B	[5] B B
4	4	A B	[6] B A
5	7	A B A	[7] A B A
6	3	C	[8] A B AC

2.3 Algoritma DMC

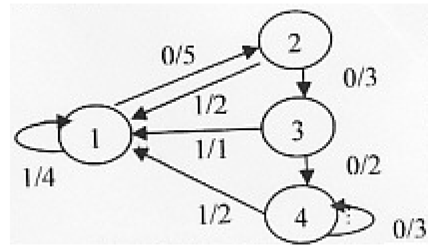
Algoritma DMC merupakan teknik pemodelan yang didasarkan pada model finite-state (model Markov). DMC membuat probabilitas dari karakter biner berikutnya dengan menggunakan pemodelan finite-state, dengan model awal berupa mesin FSA dengan transisi 0/1 dan 1/1, seperti ditunjukkan pada Gambar 2.



Gambar 2. Model awal DMC [2]

DMC merupakan teknik kompresi yang adaptif, karena struktur mesin finite-state berubah seiring dengan pemrosesan file. Kemampuan kompresinya tergolong amat baik, meskipun waktu komputasi yang dibutuhkan lebih besar dibandingkan metode lain [2].

Pada DMC, simbol alfabet input diproses per bit, bukan per byte. Setiap output transisi menandakan berapa banyak simbol tersebut muncul. Penghitungan tersebut dipakai untuk memperkirakan probabilitas dari transisi. Contoh: pada Gambar 3, transisi yang keluar dari state 1 diberi label 0/5, artinya bit 0 di state 1 terjadi sebanyak 5 kali.



Gambar 3. Sebuah model yang diciptakan oleh DMC [2]

Secara umum, transisi ditandai dengan 0/p atau 1/q dimana p dan q menunjukkan jumlah transisi dari state dengan input 0 atau 1. Nilai probabilitas bahwa input selanjutnya bernilai 0 adalah p/(p+q) dan input selanjutnya bernilai 1 adalah q/(p+q). Lalu bila bit sesudahnya ternyata bernilai 0, jumlah bit 0 di transisi sekarang ditambah satu menjadi p+1. Begitu pula bila bit sesudahnya ternyata bernilai 1, jumlah bit 1 di transisi sekarang ditambah satu menjadi q+1.

Algoritma kompresi DMC diberikan pada sebagai berikut:

1. s <- 1 /* jumlah state sekarang */

2. t <-1 1 /* state sekarang */

3. T[1][0] = T[1][1] <- 1 /* model inisialisasi */

4. C[1][0] = C[1][1] <- 1 /* inisialisasi untuk menghindari masalah frekuensi nol */

5. Untuk setiap input bit e :
    i. u <- t
    ii. t <- T[u][e] /*ikuti transisi*/
    iii. Kodekan e dengan probabilitas : C[u][e] / (C[u][0] + C[u][1])
    iv. C[u][e] <- C[u][e]+1
    v. Jika ambang batas cloning tercapai, maka :
        - s <- s + 1 /* state baru t' */
        - T[u][e] <- s ; T[s][0] <- T[t][0] ; T[s][1] <- T[t][1]
        - Pindahkan beberapa dari C[t] ke C[s]

Masalah tidak terdapatnya kemunculan suatu bit pada state dapat diatasi dengan menginisialisasi model awal state dengan satu. Probabilitas dihitung menggunakan frekuensi relatif dari dua transisi yang keluar dari state yang baru.

Jika frekuensi transisi dari suatu state t ke state sebelumnya, yaitu state u, sangat tinggi, maka state t dapat di-cloning. Ambang batas nilai cloning harus disetujui oleh encoder dan decoder. State yang di-cloning diberi simbol t' (lihat Gambar 4). Aturan

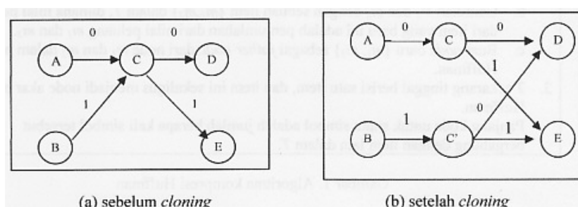
cloning adalah sebagai berikut :

- Semua transisi dari state u dikirim ke state t'. Semua transisi dari state lain ke state t tidak berubah.

- Jumlah transisi yang keluar dari t' harus mempunyai rasio yang sama (antara 0 dan 1) dengan jumlah transisi yang keluar dari t.

- Jumlah transisi yang keluar dari t dan t' diatur supaya mempunyai nilai yang sama dengan jumlah transisi yang masuk [2].

Metode DMC yang diterapkan dalam penelitian ini bertipe dinamik, dimana hanya dilakukan satu kali pembacaan terhadap file input. Kalkulasi dilakukan secara on the fly (proses perhitungan probabilitas dilakukan bersamaan dengan pengkodean data).



Gambar 4. Model Markov sebelum dan setelah *cloning.*

3. Implementasi dan Pengujian

Untuk menghasilkan pengukuran kinerja yang valid, ketiga algoritma di atas dikompilasi menggunakan compiler bahasa pemrograman yang sama (C++ Builder 5.0) dengan setting optimasi yang sama. Implementasi dan pengujian perangkat lunak ini dilakukan dalam lingkungan perangkat keras sbb.: prosesor Intel Pentium IV 2.4 GHz, memori 256MB RAM, harddisk Seagate 80 GB, dan motherboard Intel D850GB. Kecepatan kompresi dari sebuah algoritma ditentukan dari ukuran file input dibagi dengan waktu komputasi yang dibutuhkan, dan ditulis dalam satuan KByte/sec.

3.1 Kasus Uji

Dalam pengujian ini, digunakan 12 golongan besar kasus uji yang dipandang cukup memadai untuk mewakili sebagian besar tipe file yang ada, yaitu:

1. Calgary Corpus, merupakan kasus uji benchmark yang berisi koleksi dari 14 teks, yang sudah secara luas digunakan untuk mengevaluasi metode kompresi. Corpus ini dapat di-download di: ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus.

2. Canterbury Corpus, ditujukan untuk menggantikan Calgary Corpus yang telah berumur 10 tahun lebih. Corpus ini dapat di-download di http://corpus.canterbury.ac.nz/ftp/large.zip.

3. File aplikasi (Excel, Acrobat Reader, Flash, Corel Draw, PowerPoint, Font Window, dan Help)

4. File yang telah dikompresi sebelumnya.

5. File object/file biner (file com, file sistem, file hasil kompilasi C, Pascal, dan Java, file DLL)

6. File basis data (Access, DBase, Paradox, MySQL)

7. File executable, baik dalam lingkungan DOS maupun Windows

8. File gambar: file jpg, file bitmap, file gif, file png, file ani, dan file wmf.

9. File multimedia (file asf, file mpeg/mpg, file mp3, file mov, file midi, file avi, file wav)

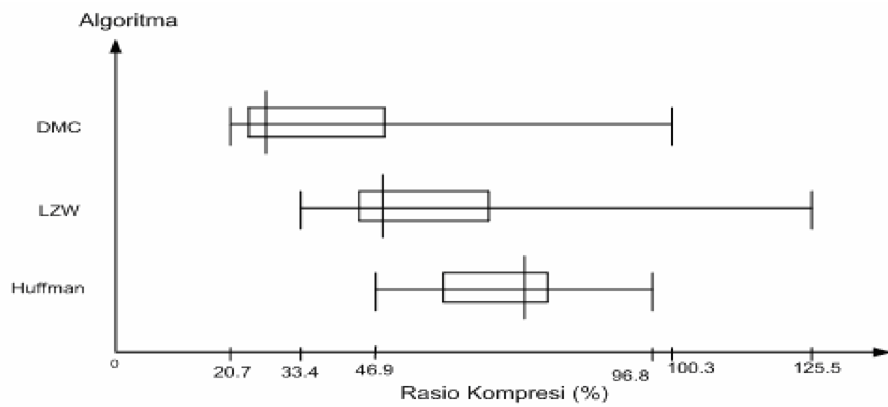
10. File source code (pascal, html, c, cpp, java, prolog, css, vbs, js, xml, php, lisp)

11. File teks (file rtf, file doc, file inf, file txt, file ini – konfigurasi Windows)

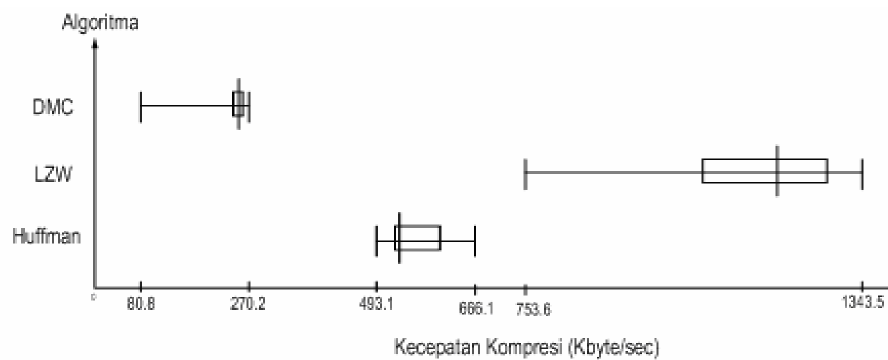
12. File pada sistem operasi Unix (berekstensi 1 – file help, file kernel, file program)

4. Hasil dan Pembahasan

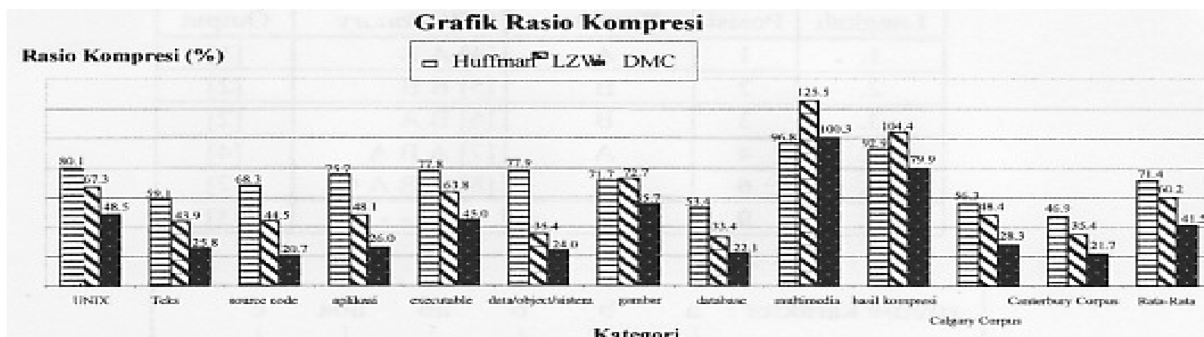
Hasil pengukuran statistik terhadap rasio hasil kompresi dan kecepatan kompresi dari ketiga metode di atas pada semua kasus uji dirangkum dalam bentuk box plot pada Gambar 5 dan 6. Grafik perbandingan rasio kompresi dan kecepatan kompresi dari ketiga metode tersebut dalam masing-masing kasus uji ditunjukkan secara lengkap dalam Gambar 7 dan 8.



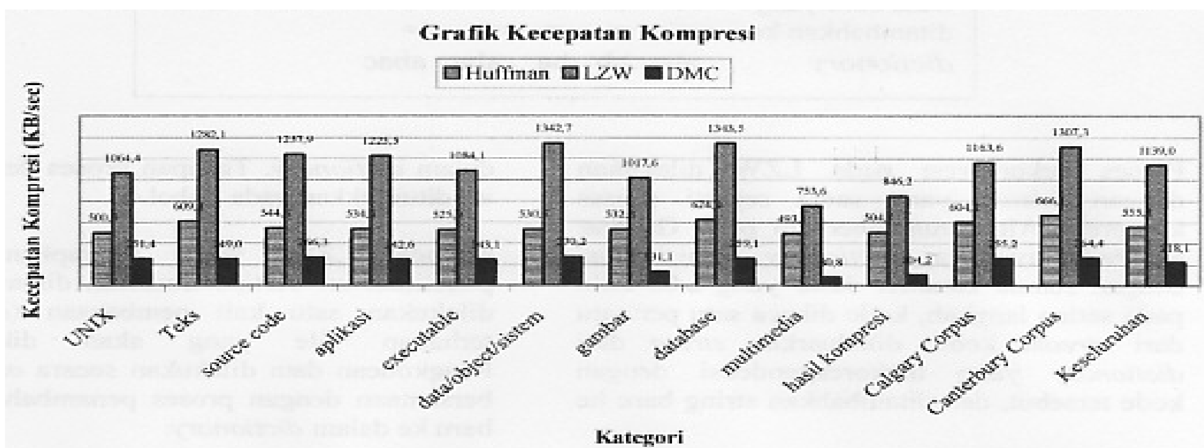
Gambar 5. *Box Plot* rasio kompresi



Gambar 6. *Box Plot* kecepatan kompresi



Gambar 7. Grafik perbandingan rasio kompresi



Gambar 8. Grafik perbandingan kecepatan kompresi

5. Kesimpulan

Dari penelitian ini dapat disimpulkan beberapa hal mengenai perbandingan kinerja ketiga metode kompresi yang telah diimplementasikan, yaitu :

1. Secara rata-rata algoritma DMC menghasilkan rasio file hasil kompresi yang terbaik (41.5% ± 25.9), diikuti algoritma LZW (60.2% ± 28.9) dan terakhir algoritma Huffman (71.4% ± 15.4).

2. Untuk kategori file teks, source code, file aplikasi, dan file basis data, DMC memberikan hasil kompresi yang baik sekali. Sedangkan untuk file multimedia, hasil kompresinya buruk (dapat > 100 %), karena pada umumnya file multimedia merupakan file hasil kompresi juga, dan hasil kompresi DMC terhadap file yang telah terkompresi sebelumnya memang kurang baik.

3. Hasil kompresi Huffman lebih baik dibandingkan LZW hanya pada kasus file biner, file multimedia, file gambar, dan file hasil kompresi. Algoritma Huffman memberikan hasil yang relatif hampir sama untuk setiap kasus uji, sedangkan LZW memberikan hasil kompresi yang buruk (dapat > 100%) untuk file multimedia dan file hasil kompresi.

4. Secara rata-rata algoritma LZW membutuhkan waktu kompresi yang tersingkat (kecepatan kompresinya = 1139 KByte/sec ± 192,5), diikuti oleh algoritma Huffman (555,8 KByte/sec ± 55,8), dan terakhir DMC (218,1 KByte/sec ± 69,4). DMC mengorbankan kecepatan kompresi untuk mendapatkan rasio hasil kompresi yang baik. File yang berukuran sangat besar membutuhkan waktu yang sangat lama bila dikompresi dengan DMC (contoh: file multimedia dengan ukuran 59 MB membutuhkan waktu kompresi 12,3 menit).

5. Kecepatan kompresi algoritma LZW secara signifikan berkurang pada file UNIX, file executable, file gambar, file multimedia, dan file hasil kompresi. Kecepatan kompresi algoritma DMC berkurang pada file gambar dan file hasil kompresi, bahkan untuk file multimedia kecepatan kompresi berkurang lebih dari sepertiga kalinya dibandingkan kecepatan kompresi rata-rata. Kecepatan kompresi algoritma Huffman hampir merata untuk semua kategori file.

6. Daftar Referensi

[1] Free On-line Dictionary of Computing, http://www.foldoc.org/
[27 Desember 2007]

[2] Witten, I.H, et al., “Managing Gigabytes”, Van Nostrand Reinhold, New York, 1994.

[3] Ben Zhao, http://www.cs.cmu.edu/afs/cs/project/pscico-guyb/294/class-notes/all/02.ps
[27 Desember 2007]

[4] http://en.wikipedia.org/wiki/Lempel-Ziv-Welch
[27 Desember 2007]

[5] http://www.rasip.fer.hr/research/compress/algorithms/fund/lz/LZW.htm [28 Desember 2007]

[6 University of Calgary, http://links.uwaterloo.ca/calgary.corpus.html
[29 Desember 2008]

[7] University of Canterbury,“The Canterbury Corpus” , http://corpus.canterbury.ac.nz
[29 Desember 2008]

[8] Cormack, G.V., Horspool, R.N., “Data Compression Using Dynamic Markov Compression”, University of Waterloo and University of Victoria, 1986.