Kompresi Data dengan Algoritma *Huffman* dan Perbandingannya dengan Algoritma LZW dan DMC

Roy Indra Haryanto - 13508026

Fakultas Sekolah Teknik Elektro dan Informatika
Program Studi Teknik Informatika
Institut Teknologi Bandung
Jalan Ganesha No. 10, Bandung, 40132
e-mail: royindrah@gmail.com

ABSTRAK

Makalah ini membahas tentang kompresi data dengan algoritma Huffman dan membandingkannya dengan kompresi data menggunakan algoritma lain seperti algoritma LZW (*Lempel-Ziv-Welch)* dan DZW *(Dynamic Markov Compression).* Dalam makalah ini dijelaskan mengenai cara kerja algoritma Huffman, LZW dan DMC dalam melakukan kompresi data. Kompresi data merupakan proses yang penting dalam dunia informatika dan algoritma Huffman merupakan salah satu algoritma yang telah digunakan secara luas untuk mengompresi data dalam berbagai bahasa pemrograman.

Kata kunci: Algoritma Huffman, Kata *LZW (*Lempel-Ziv-Welch), *DZW (Dynamic Markov Compression*, kompresi data.

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], 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 :

(a) 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.

(b) 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.

2. METODE

2.1 Algoritma – algoritma dalam Kompresi Data

Dalam pengkompresian data, ada bermacam – macam algoritma yang dapat digunakan. Dalam upabab kali ini, akan dijelaskan tentang algoritma *Huffman, LZW*, dan *DMC.*

2.1.1 Algoritma Huffman

Algoritma Huffman ditemukan oleh David Huffman pada tahun 1952. Algoritma ini menggunakan pengkodean yang mirip dengan kode Morse. Berdasarkan tipe kode yang digunakan algoritma Huffman termasuk metode statistic. Sedangkan berdasarkan teknik pengkodeannya menggunakan metode symbolwise. Algoritma Huffman merupakan salah satu algoritma yang digunakan untuk mengompres teks. Algoritma Huffman secara lengkap [5] :

1. Pilih dua simbol dengan peluang (*probability*) paling kecil (pada contoh di atas simbol *B* dan *D*). Kedua simbol tadi dikombinasikan sebagai simpul orangtua dari simbol *B* dan *D* sehingga menjadi simbol *BD*

dengan peluang 1/7 + 1/7 = 2/7, yaitu jumlah peluang kedua anaknya.
2. Selanjutnya, pilih dua simbol berikutnya, termasuk simbol baru, yang mempunyai peluang terkecil.
3. Ulangi langkah 1 dan 2 sampai seluruh simbol habis.

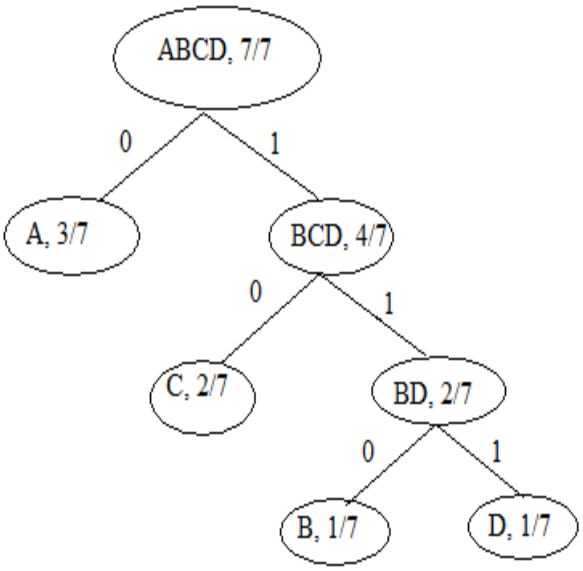
Sebagai contoh, dalam kode ASCII *string* 7 huruf "ABACCDA" membutuhkan representasi 7 × 8 bit = 56 bit (7 byte), dengan rincian sebagai berikut:

01000001 01000010 01000001 01000011 01000011
A B A C C

01000100 01000001
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.

Gambar 1. Pohon Huffman untuk "ABACCDA"



Tabel 1 Kode Huffman

Karakter	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 yang seharusnya dibutuhkan 56 bit.

Untuk menguraikan kembali data yang sudah dikodekan sebelumnya dengan algoritma *Huffman*, dapat digunakan cara sebagai berikut :

1. Baca bit pertama dari string biner masukan
2. Lakukan traversal pada pohon Huffman mulai dari akar sesuai dengan bit yang dibaca. Jika bit yang dibaca adalah 0 maka baca anak kiri, tetapi jika bit yang dibaca adalah 1 maka baca anak kanan.
3. Jika anak dari pohon bukan daun (simpul tanpa anak) maka baca bit berikutnya dari string biner masukan.
4. Hal ini diulang (traversal) hingga ditemukan daun.
5. Pada daun tersebut simbol ditemukan dan proses penguraian kode selesai.
6. Proses penguraian kode ini dilakukan hingga keseluruhan string biner masukan diproses.

2.1.2 Algoritma LZW (*Lempel-Ziv-Welch)*

Algoritma LZW dikembangkan oleh Terry A.Welch dari metode kompresi sebelumnya yang ditemukan oleh Abraham Lempel dan Jacob Ziv pada tahun 1977. Algortima ini menggunakan teknik dictionary dalam kompresinya. Dimana string karakter digantikan oleh kode table yang dibuat setiap ada string yang masuk. Tabel dibuat untuk referensi masukan string selanjutnya. Ukuran tabel dictionary pada algoritma LZW asli adalah 4096 sampel atau 12 bit, dimana 256 sampel pertama digunakan untuk table karakter single (Extended ASCII), dan sisanya digunakan untuk pasangan karakter atau string dalam data input. [4]

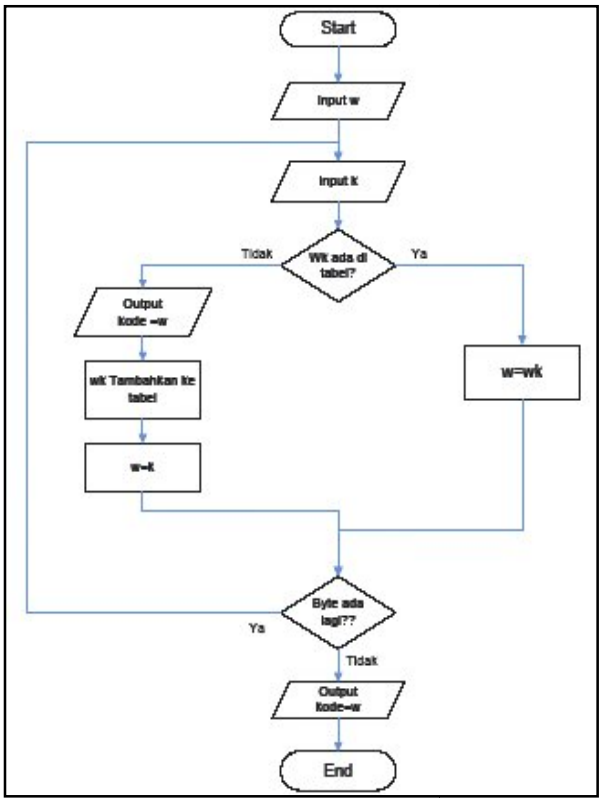
Algoritma LZW melakukan kompresi dengan mengunakan kode table 256 hingga 4095 untuk mengkodekan pasangan byte atau string. Dengan metode ini banyak string yang dapat dikodekan dengan mengacu pada string yang telah muncul sebelumnya dalam teks.

Algoritma kompresi LZW secara lengkap :

1. *KAMUS* diinisialisasi dengan semua karakter dasar yang ada : {'A'..'Z','a'..'z','0'..'9'}.
2. *W* ← karakter pertama dalam *stream* karakter.
3. *K* ← karakter berikutnya dalam *stream* karakter.
4. Lakukan pengecekan apakah (W+K) terdapat dalam *KAMUS*
   - Jika ya, maka *W* ← *W* + *K* (gabungkan *W* dan *K* menjadi *string* baru).
   - Jika tidak, maka :
     - ✓*Output* sebuah kode untuk menggantikan *string W*.
     - ✓Tambahkan *string* (*W*+ *K*) ke dalam *dictionary* dan berikan nomor/kode berikutnya yang belum digunakan dalam *dictionary* untuk *string* tersebut.
     - ✓W ← K.

- Lakukan pengecekan apakah masih ada karakter berikutnya dalam *stream* karakter
  - ✓ Jika ya, maka kembali ke langkah 2.
  - ✓ Jika tidak, maka *output* kode yang menggantikan *string W*, lalu terminasi proses (*stop*).

Gambar 2. Flowchart Algoritma LZW



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 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	A	[8] A B A C	[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 2.

Gambar 3. Hasil Proses Kompresi

stream karakter :	a	b	b	ab	aba	c
	↓	↓	↓	↓	↓	↓
kode *output* :	[1]	[2]	[2]	[4]	[7]	[3]
frasa baru yang ditambahkan ke *dictionary*	4 = ab	5 = bb	6 = ba	7 = aba	8 = abac	

Proses dekompresi data pada algoritma LZW tidak jauh berbeda dengan proses kompresinya. Pada dekompresi LZW, juga dibuat tabel dictionary dari data input kompresi, sehingga tidak diperlukan penyertaan tabel dictionary ke dalam data kompresi. Berikut algoritma dekompresi LZW :

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 :
     - ✓ *Output string.CW* ke *stream* karakter
     - ✓ *P* ← *string.PW*
     - ✓ *C* ← karakter pertama dari *string.CW*
     - ✓ Tambahkan *string* (*P*+*C*) ke dalam *dictionary*
   - Jika tidak, maka :
     - ✓ *P* ← *string.PW*
     - ✓ *C* ← karakter pertama dari *string.PW*
     - ✓ *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*).

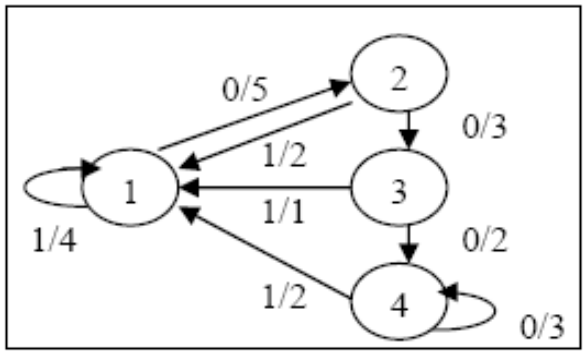
2.1.3 Algoritma DMC

Algoritma *DMC* (*Dynamic Markov Compression*) adalah algoritma kompresi data lossless dikembangkan oleh Gordon Cormack dan Nigel Horspool. Algoritma ini menggunakan pengkodean aritmetika mirip dengan prediksi oleh pencocokan sebagian (PPM), kecuali bahwa input diperkirakan satu bit pada satu waktu (bukan dari satu byte pada suatu waktu). DMC memiliki rasio kompresi yang baik dan kecepatan moderat, mirip dengan PPM, tapi memerlukan sedikit lebih banyak memori dan tidak diterapkan secara luas. Beberapa implementasi baru-

baru ini mencakup program kompresi eksperimental pengait oleh Nania Francesco Antonio, ocamyd oleh Frank Schwellinger, dan sebagai submodel di paq8l oleh Matt Mahoney. Ini didasarkan pada pelaksanaan tahun 1993 di C oleh Gordon Cormack.

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 4. Sebuah model yang diciptakan oleh metode DMC



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 :

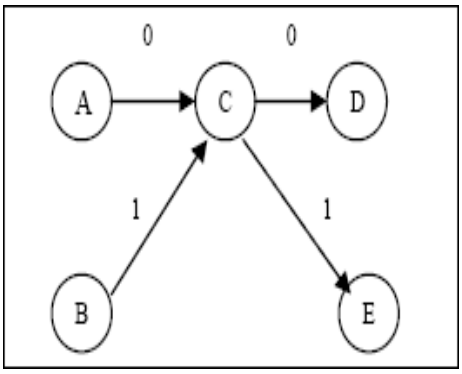
1. $s \leftarrow 1$ (jumlah state sekarang)
2. $t \leftarrow 1$ (state sekarang)
3. $T[1][0] = T[1][1] \leftarrow 1$ (model inisialisasi)
4. $C[1][0] = C[1][1] \leftarrow 1$ (inisialisasi untuk menghindari masalah frekuensi nol)
5. Untuk setiap input bit *e* :
   - $u \leftarrow t$
   - $t \leftarrow T[u][e]$ (ikuti transisi)
   - Kodekan *e* dengan probabilitas : $C[u][e] / (C[u][0] + C[u][1])$
   - $C[u][e] \leftarrow C[u][e]+1$
   - Jika ambang batas *cloning* tercapai, maka :
     - ✓ $s \leftarrow s + 1$ (state baru *t*')
     - ✓ $T[u][e] \leftarrow s$; $T[s][0] \leftarrow T[t][0]$; $T[s][1] \leftarrow T[t][1]$
     - ✓ 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.

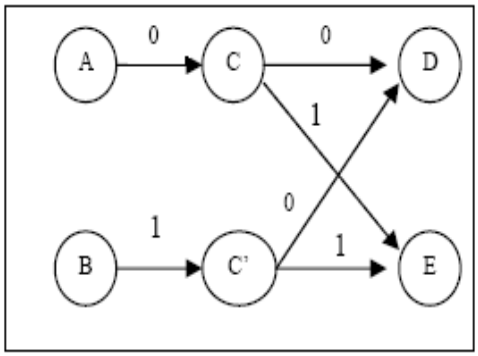
Jika frekuensi transisi dari suatu *state t* ke *state* sebelumnya, yaitu *state u*, sangat tinggi, maka *state t* dapat di-*cloning*. Ambang batas nilai *cloning* harus disetujui oleh *encoder* dan *decoder*. State yang di-*cloning* diberi simbol *t*' (lihat Gambar 4 dan 5). Aturan *cloning* adalah 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].

Gambar 5. Model Markov sebelum cloning



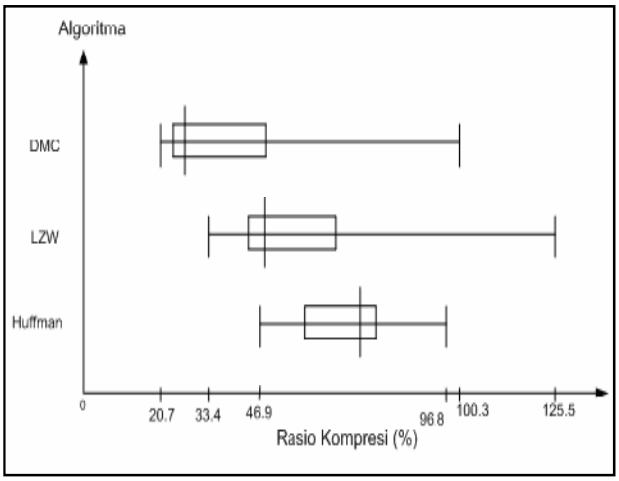
Gambar 6. Model Markov setelah cloning



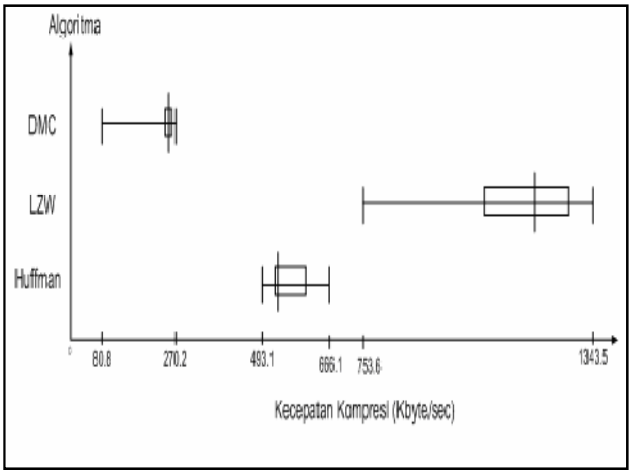
2.2 Perbandingan Kinerja Algoritma *Huffman* dengan Algoritma *LZW* dan *DMC*

Jika kinerja algoritma Huffman dibandingkan dengan Algoritma LZW dan DMC, maka akan diperoleh hasil seperti dibawah ini [6] :

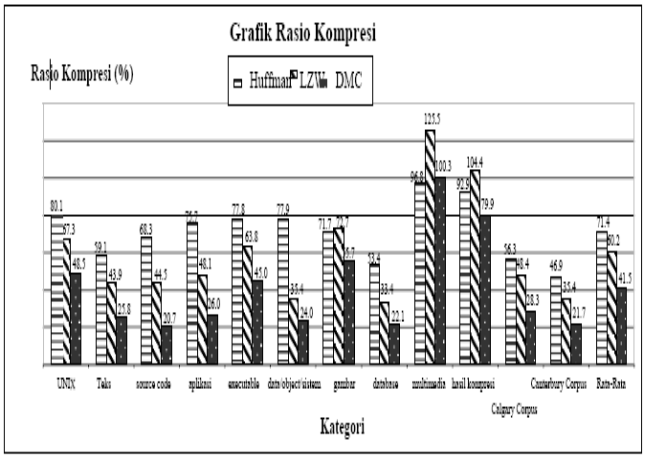
Gambar 7. *Box Plot* Rasio Kompresi Algoritma Huffman, LZW dan DMC



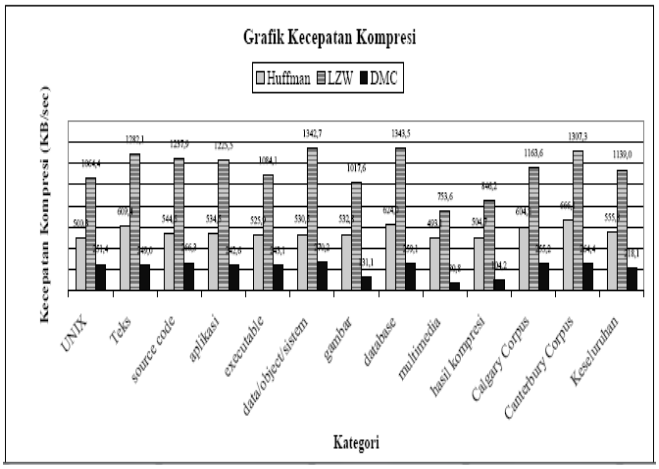
Gambar 8. *Box Plot* Kecepatan Kompresi Algoritma Huffman, LZW dan DMC



Gambar 9. Grafik Perbandingan Rasio Kompresi Algoritma Huffman, LZW dan DMC



Gambar 10. Grafik Perbandingan Kecepatan Kompresi Algoritma Huffman, LZW dan DMC



Dari grafik di atas, dapat kita lihat bahwa 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).

Dan dari grafik di atas juga, dapat kita lihat bahwa 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.

3. KESIMPULAN

Dari makalah ini, dapat diambil kesimpulan sebagai berikut :

1. Algoritma *Huffman* dapat digunakan sebagai dasar untuk kompresi data, dan pengaplikasiannya cukup mudah serta dapat digunakan dalam berbagai jenis data.
2. 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)
3. 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.
4. Jika dibandingkan dengan algoritma LZW dan DMC, dalam kompresi data, algoritma Huffman masih kalah dalam hal rasio kompresi data maupun kecepatan kompresinya.

4. UCAPAN TERIMA KASIH

Makalah ini dibuat untuk memenuhi tugas mata kuliah Struktur Diskrit. Pertama – tama Penulis mengucapkan terima kasih kepada dosen pengajar mata kuliah Struktur Diskrit yang telah memberikan ilmu yang sangat bermanfaat untuk penulis. Tanpa ilmu yang diberikan, tidak mungkin penulis mampu menulis makalah ini.

Penulis juga memberikan ucapan terima kasih kepada setiap pihak yang telah membantu dalam menyelesaikan makalah ini. Terutama kepada setiap institusi pendidikan yang telah bersedia memberikan sarana dalam internet untuk memberikan data yang berhubungan dengan makalah ini.

Ucapan terima kasih ini penulis berikan karena tanpa bantuan mereka semua, penulis tidak mungkin dapat menyelesaikan makalah ini.

REFERENSI

[1] Howe, D., *"Free Online Dictionary of Computing"*, http://www.foldoc.org/
Tanggal Akses 18 Desember 2009 21.27

[2] Ben Zhao, *et al.*, "*Algorithm in the Real World - (Compression) Scribe Notes"*, http://www-2.cs.cmu.edu/~guyb/realworld/class-notes/all/
Tanggal Akses 18 Desember 2009 22.35

[3]http://www.ittelkom.ac.id/library/index.php?view=article&catid=20%3Ainformatika&id=188%3Aalgoritma-huffman-dan-shannon-fano&option=com_content&Itemid=15
Tanggal Akses 18 Desember 2009 23.05

[4]http://www.ittelkom.ac.id/library/index.php?view=article&catid=20%3Ainformatika&id=478%3Aalgoritma-lzw-lemple-ziv-welch&option=com_content&Itemid=15
Tanggal Akses 18 Desember 2009 23.16

[5] Rinaldi Munir, 2003, Diktat Kuliah Matematika Diskrit, Penerbit ITB.

[6] Linawati. "Perbandingan Kinerja Algoritma Kompresi *Huffman, LZW*, dan *DMC* pada Berbagai Tipe File", 2004.