SinkrOn
JURNAL & PENELITIAN TEKNIK INFORMATIKA
Publikasi Jurnal & Penelitian Teknik Informatika
Volume 2 Nomor 2, April 2018
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Analisa Perbandingan Rasio Kecepatan Kompresi Algoritma Dynamic Markov Compression Dan Huffman

Indra Kelana Jaya
Universitas Methodist Indonesia
Medan, Indonesia
indrakj_sagala@yahoo.com

Resianta Perangin-angin
Universitas Methodist Indonesia
Medan, Indonesia
resianta88@gmail.com

Abstract — Kompresi menjadi penting dikarenakan penyimpanan ruang yang terbatas, oleh karena itu kompresi merupakan satu-satu nya cara untuk meminimalisir percepatan overload size data, dalam penelitian ini dilakukan sebuah ujicoba menggunakan algoritm DMC dan Huffman dalam hal kompresi file, dimana dari 15 iterasi yang dilakukan didapat bahwasanya algoritma DMC untuk setiap iterasi yang dilakukan rata-rata mengkompres diatas 50% dari kapasistas aslinya sedangkan algoritma Huffman diatas 76% untuk hasil kompresi dari setiap file yang di ujikan, sedangkan untuk kecepatan sendiri ini berbanding lurus dengan besarnya file yang telah di kompresi untuk algoritma DMC sendiri apa bila file yang dikompresi masih dalam kapasistas kecil maka algoritma ini akan lebih cepat dibandingkan dengan algoritma Huffman, namun menjadi menarik dikarenakan apabila kompresi file menggunkan size yang cukup besar maka kecepatan kompresi menjadi lebih baik metode Huffman. Rasio ukuran *file* yang diperoleh dengan algoritma Huffman cukup tinggi berkisar minimal 76% Jadi dapat dikatakan dengan rasio kompresi ini algoritma Huffman sudah dikatakan baik dalam hal mengkompresi *file* khususnya *file* .Tingkat kompresi dipengaruhi oleh banyaknya nada yang sama dalam *file* .Kecepatan proses tidak bergantung pada data yang diproses tetapi berbanding lurus dengan ukuran *file* , artinya semakin besar ukuran *file* yang diproses maka semakin lama waktu prosesnya.Proses dekompresi lebih cepat dilakukan dibandingkan dengan proses kompresi karena pada proses dekompresi tidak dilakukan lagi proses pembentukan pohon Huffman dari data melainkan hanya langsung membaca dari tabel *code* pohon Huffman yang disimpan pada *file* sewaktu proses kompresi.
Kata kunci : *huffman, dmc, kompresi, perbandingan, analisa*

I. PENDAHULUAN

Berbicara masalah *source* atau ruang penyimpanan hal ini tentu lumrah bagi hampir semua orang di era digitalisasi ini, perkembangan dari masa ke masa dalam hal penyimpanan sangatlah pesat. Terlihat dari hadrdware-hardware penyimpan data digital sangatlah mumpuni dalam hal ruang yang mencapai ratusan giga byte bahkan mencapai tera byte untuk 1 unit hardware penyimpanan. Namun demikian seiring dengan perkembangan hardware penyimpanan yang sangat masif dalam hal ruang ini berbanding lurus dengan data digitalisasi yang juga sangat besar dan sangat mudah untuk mendapatkanya, dengan demikian sebesar apapun ruang penyimpanan yang kita punya ini hanya masalah waktu sampai ruang tersebut habis.

Kompresi data merupakan salah satu kajian di dalam ilmu komputer yang bertujuan untuk mengurangi ukuran *file* sebelum menyimpan atau memindahkan data tersebut ke dalam media penyimpanan. Salah satu teori yang cukup sederhana adalah dengan menggunakan metode Huffman.[1]

Oleh sebab itu perlu meminimalisasi data-data yang ada dari segi size atau ukuran file ataupun data tersebut. Kompresi merupakan proses untuk mengubah data menjadi sekumpulan kode untuk menghemat tempat penyimpanan dengan atau tanpa mengurangi kualitas dari citra serta mempercepat waktu transmisi data [2], berangkat dari pengertian ini kita memahami bahwasanya kompresi merupakan solusi untuk meminimalisir suatu ruang data terlalu cepat penuh.

Salah satu kegunaan kompresi adalah untuk memperkecil kapasitas kosong dalam memori media penyimpanan, agar kita tidak terlalu boros menggunakan media penyimpanan tersebut. [3], Pada kompresi data, terdapat dua tipe macam kompresi, yaitu lossless compression dan lossy compression. Pada lossless compression, semua informasi yang ada pada data akan kembali menjadi seperti aslinya dan tidak ada informasi yang hilang. Teknik ini biasanya digunakan untuk dokumen-dokumen, file executable, dan lainnya. [3], Kompresi data menjadi solusi untuk menangani atau setidaknya mengimbangi berkembangnya kebutuhan yang tidak terbendung tersebut. Kompresi sangat berguna ketika data terlalu besar tetapi perlu disimpan dalam tempat yang terbatas ataupun data akan dikirim melalui sebuah saluran komunikasi yang memiliki bandwidth terbatas. [4]

Ada banyak metode kompresi antara lain Lempel-Ziv Compression (LZ77, LZ78, LZW, GZIP), Dynamic Markov Compression (DMC), Block-Sorting Lossless, Run-Length, Shannon-Fano, Arithmetic, PPM (Prediction by Partial Matching), Burrows-Wheeler Block Sorting, Half Byte, Huffman Coding, dan masih banyak lainnya.[4]

Metode Huffman merupakan salah satu teknik kompresi data yang bersifat loseless. Metode ini menggunakan prinsip bahwa nilai derajat keabuan yang sering muncul di dalam citra akan dikodekan dengan jumlah bit yang lebih sedikit, sedangkan nilai keabuan yang munculnya sedikit (jarang) dikodekan dengan jumlah bit yang lebih panjang.[5]

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. [6]

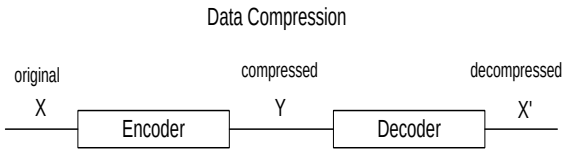
Namun untuk penelitian ini akan mencoba mengimplementasikan algoritma DMC dan Huffman untuk kompresi file dikarenakan bahwa metode DMC dapat disanding dengan aritmatika coding untuk mencapai suatu tingkat kompresi data yang sangat baik. [7], Sedagkan metode huffman menggunakan prinsip pengkodean yang 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.[8]

Oleh karena itu penelitian ini bertujuan melihat bagaimana kecepatan dan tingkat rasio dari hasil kompresi algoritma DMC dan Huffman dalam hal kompresi untuk file.

II. PEMBAHASAN

A. Sekema Kompresi File Model DMC

Dalam penelitian ini dirancang sebuah skema atau alur sistem untuk proses kompresi dan dekompresi file menggunakan algoritma DMC, untuk lebih jelasnya dapat melihat gambar 2.1



Gambar 1 Sekema kompresi dan dekompresi

Dimana :

X, Y, X' = String

Lossless Compression: X = X'

Lossy Compression: X != X'

Compression Ratio: |X| / |Y| dimana |X| adalah jumlah *bit* dalam dalam X dan |Y| adalah jumlah *bit* dalam Y

Algoritma ini menggunakan pengkodean aritme 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.

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 = 1 (jumlah state sekarang)
2. t = 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 :

Dimana :

u = t

t = T[u][e] (ikuti transisi)

Kodekan e dengan probabilitas : C[u][e] / (C[u][0] + C[u][1])

C[u][e] = C[u][e]+1

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]

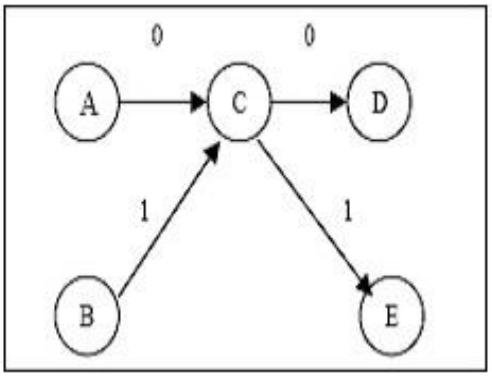
Masalah tidak terdapatnya kemunculan suatu bit pada state dapat diatasi dengan menginisialisasi model awal state dengan satu. Probabilitas dihitung menggunakan

frekuensi relatif dari dua transisi yang keluar dari state yang baru.

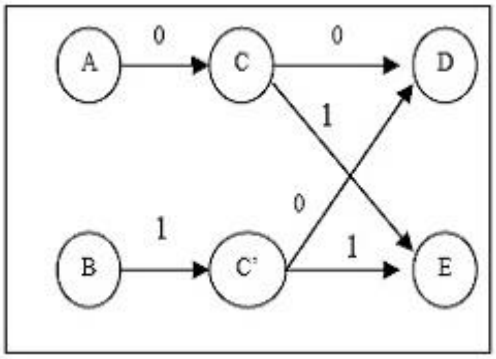
Jika frekuensi transisi dari suatu state t ke state sebelumnya, yaitu state u, sangat tinggi, maka state t dapat di-cloning. Ambang batas nilai cloning harus disetujui oleh encoder dan decoder. State yang di-cloning diberi simbol t' (lihat Gambar 2.2 dan 2.3).

Aturan cloning adalah sebagai berikut :

1. Semua transisi dari state u dikirim ke state t'. Semua transisi dari state lain ke state t tidak berubah.
2. Jumlah transisi yang keluar dari t' harus mempunyai rasio yang sama (antara 0 dan 1) dengan jumlah transisi yang keluar dari t.
3. Jumlah transisi yang keluar dari t dan t' diatur supaya mempunyai nilai yang sama dengan jumlah transisi yang masuk.



Gambar 2 Model DMC Sebelum Cloning



Gambar 3 Model DMC Sesudah Cloning

Dari sekema yang telah ada akan diterapkan untuk kompresi dan dekompresi file menggunakan algoritma DMC, akan dilihat kecepatan dan besaran kompresi yang dapat dihasilkan algoritma ini dalam kompresi file.

B. Sekema Model Kompresi Huffman

Ide dasar dari *encoding* Huffman adalah mencocokkan *code word* yang pendek pada blok input dengan kemungkinan yang terbesar dan *code word* yang panjang dengan kemungkinan terkecil. Konsep ini mirip dengan *Morse Code*.

Suatu *file* merupakan kumpulan dari karakter-karakter. Dalam suatu *file* tertentu suatu karakter dipakai lebih banyak daripada yang lain. Jumlah bit yang diperlukan untuk merepresentasikan tiap karakter bergantung pada jumlah karakter yang harus direpresentasikan. Dengan menggunakan satu bit maka dapat merepresentasikan dua buah karakter. Sebagai contoh 0 merepresentasikan karakter pertama dan 1 merepresentasikan karakter kedua. Dengan menggunakan dua bit maka dapat merepresentasikan 2^2 atau 4 buah karakter.

00 – karakter pertama
01 – karakter kedua
10 – karakter ketiga
11 – karakter keempat

Secara umum jika ingin merepresentasikan *n* buah karakter maka diperlukan 2^n bit untuk merepresentasikan satu karakter. Kode ASCII (American Standar Code for In*form*ation Interchange) menggunakan 7 bit untuk merepresentasikan sebuah karakter. Oleh karena 2^7 = 128 bit maka direpresentasikan dengan menggunakan kode ASCII.

Kode ASCII dan kode yang disebutkan di bagian atas untuk merepresentasikan karakter-karakter dikenal sebagai *fixed length codes*. Ini dikarenakan tiap karakter mempunyai panjang bit yang sama atau dengan kata lain jumlah bit yang diperlukan untuk merepresentasi tiap karakter sama. Pada kode ASCII setiap karakter memerlukan 7 bit. Dengan menggunakan *variable length codes* untuk tiap karakter maka dapat direduksi ukuran dari suatu *file*. Dengan menggantikan *code* yang lebih kecil untuk karakter-karakter yang lebih sering dipakai dan *code* yang lebih besar untuk karakter yang tidak sering dipakai, maka sebuah *file* dapat dikompresi.

Sebagai contoh misalkan sebuah *file* terdiri atas data berikut ini.

AAAAAAAAAABBBBBBBBCCCCCCDDD DDEE

Maka frekuensi atau banyaknya sebuah karakter muncul pada sebuah *file* adalah sebagai berikut.

Frekuensi dari A adalah 10
Frekuensi dari B adalah 8
Frekuensi dari C adalah 6
Frekuensi dari D adalah 5
Frekuensi dari E adalah 2

Jika tiap karakter direpresentasikan dengan menggunakan tiga buah bit maka jumlah bit yang diperlukan untuk menyimpan *file* ini adalah:

3 * 10 + 3 * 8 + 3 * 6 + 3 * 5 + 3 * 2 = 93 bit

Sekarang misalkan karakter-karakter di atas direpresentasikan seperti berikut ini.

A dengan code 11
B dengan code 10
C dengan code 00
D dengan code 011
E dengan code 010

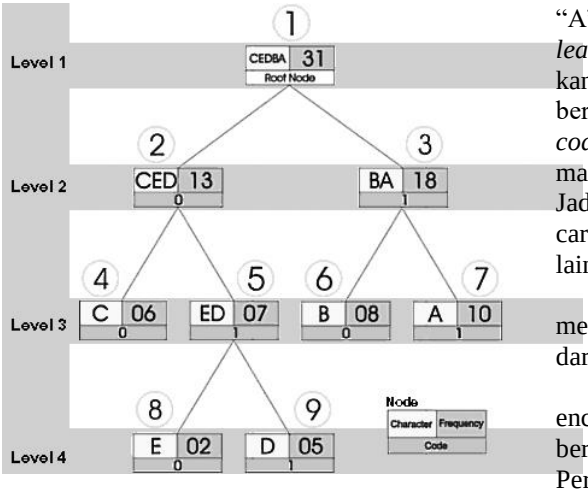
Maka ukuran *file* tersebut akan menjadi 2 * 10 + 2 * 8 + 2 * 6 + 3 * 5 = 69 bit. Tingkat kompresi dengan nilai tertentu telah dicapai saat ini. Secara umum nilai rasio kompresi dapat diperoleh dengan menggunakan cara atau metode seperti ini.

Seperti yang terlihat frekuensi karakter yang sering muncul digantikan dengan *code* yang lebih kecil sementara frekuensi karakter yang jarang muncul digantikan dengan *code* yang lebih besar. Salah satu kesulitan dengan menggunakan *variable-length code* adalah tidak dapat diketahui kapan dicapai akhir dari suatu karakter dalam pembacaan urutan bit ‘0’ dan ‘1’. Masalah ini dapat dipecahkan jika merancang kode sedemikian rupa bahwa tidak ada *code* yang sama persis dipakai kembali untuk karakter yang lain. Pada kasus di atas A direpresentasikan dengan 11. Tidak ada *code* yang lain dimulai dengan 11 lagi. Seperti halnya dengan C digantikan dengan *code* 00. Maka tidak ada *code* yang lain dengan 00. *Code* jenis seperti ini dikenal sebagai *prefix codes*.

Secara umum, metode untuk menggantikan suatu karakter *code* dengan menggunakan *variable-length prefix codes* yang mengambil keuntungan dari frekuensi relatif karakter pada teks untuk di-*encode* dikenal sebagai *encoding* Huffman.

Kode Huffman dapat direpresentasikan sebagai suatu pohon biner dimana daun-daun merupakan karakter yang akan di-*encode*. Pada setiap *non-leaf node* dari pohon terdiri atas kumpulan semua karakter pada daun yang berada di bawah *node*. Sebagai tambahan, tiap daun disimbolkan dengan suatu nilai (*weight*) yang merupakan frekuensi dari karakter, dan setiap *non-leaf node* terdiri atas sebuah *weight* yang merupakan jumlah dari semua *weight* dari daun-daun yang berada di

bawahnya. Contoh dari pohon Huffman dapat dilihat pada gambar 2 dibawah ini.



Gambar 2. Pohon Huffman

Sebagai catatan bila *n* karakter terdapat dalam sebuah *file* maka jumlah *node* pada pohon Huffman berjumlah $(2n - 1)$. Jika terdapat *n node* dalam sebuah pohon maka terdapat paling banyak $(n + 1)/2$ *level*, dan sekurang-kurangnya $\log_2(n + 1)$ *level*. Jumlah *level* pada sebuah pohon Huffman mengindikasikan panjang maksimum dari *code* yang diperlukan untuk merepresentasikan sebuah karakter.

Code length dari sebuah karakter mengindikasikan *level* dimana karakter tersebut berada. Jika *code length* dari sebuah karakter adalah *n* maka karakter tersebut berada pada *level* ke $(n + 1)$ dari pohon. Sebagai contoh *code length* dari karakter D adalah 011, maka *code length*-nya adalah 3. Oleh karena itu karakter tersebut harus berada pada *level* keempat dari pohon.

Code untuk tiap karakter diperoleh dengan memulai dari *node* akar dan bergerak turun ke daun yang merepresentasikan karakter. Ketika bergerak ke *node* kiri anak sebuah bit '0' ditambahkan pada code dan ketika bergerak ke sisi kanan *node* anak, bit '1' ditambahkan pada *code*.

Untuk memperoleh *code* dari karakter "A" dari pohon, pertama sekali dimulai dari *node* akar (*node* 1). Karena karakter "A" pada keturunan pada sisi kanan *node* anak (ini ditentukan dengan cara cabang yang mana yang akan diikuti dengan mengetes dan melihat apakah cabang tersebut merupakan *node* daun untuk karakter ataupun merupakan *ancestor*-nya) bergerak ke kanan dan menambahkan bit '1' pada code untuk karakter "A". Sekarang bila telah berada pada *node* 3, *leaf node* untuk karakter "A" berada pada kanan dari *node* tersebut, jadi sekali lagi bergerak ke kanan dan menambahkan '1' pada *code*-nya. Sekarang telah dicapai *node* 7 yang mana merupakan *leaf node* untuk karakter A. Jadi *code* untuk karakter "A" adalah 11. Jadi cara yang sama untuk *code* tiap karakter yang lain dapat diperoleh juga.

Seperti terlihat *code* dari karakter yang mempunyai frekuensi tertinggi lebih pendek dari pada *code* dengan frekuensi yang rendah.

Metode *encoding* ini meminimalkan encoding *variable-length character* berdasarkan pada frekuensi dari tiap karakter. Pertama, tiap karakter menjadi sebuah pohon trivial (*trivial tree*), dengan karakter hanya sebagai *node*. Frekuensi karakter merupakan frekuensi dari pohon. Jika dua pohon dengan frekuensi paling sedikit digabungkan dengan sebuah akar baru maka akan memberikan hasil jumlah dari frekuensi mereka. Ini akan berulang hingga semua karakter membentuk satu buah pohon. Satu kode bit merepresentasi tiap *level*. Jadi karakter yang berfrekuensi tinggi akan dekat dengan akar dan akan di-*encode* dengan beberapa bit, dan karakter yang jarang muncul akan jauh dari akar yang di-*encode* dengan panjang bit yang banyak.

Menggabungkan pohon-pohon dengan frekuensi sama halnya dengan menggabungkan urutan-urutan panjang data untuk mendapatkan hasil penggabungan yang optimal. Dikarenakan sebuah *node* dengan hanya satu anak tidaklah optimal, maka *encoding* Huffman merupakan satu pohon biner yang lengkap.

Sebagai catatan kasus terburuk dari *encoding* Huffman (atau sama halnya dengan encoding paling panjang Huffman untuk satu kumpulan karakter) adalah ketika distribusi dari frekuensi diikuti oleh bilangan Fibonacci.

Encoding Huffman optimal untuk meng-*encoding* karakter (satu karakter dengan satu *code word*) dan mudah untuk diprogram. Metode lain seperti Shannon-Fano merupakan kode *prefix* minimal. Sedangkan *arithmetic coding* saat ini yang paling bagus karena dapat mengalokasi sebagian kecil bit, tetapi lebih rumit.

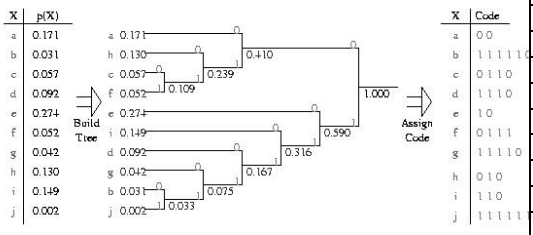
Kompleksitas waktu dari *encoding* Huffman adalah O(nlogn). Dengan

menggunakan sebuah *heap* untuk menyimpan nilai besaran dari tiap pohon, tiap iterasi memerlukan waktu O(logn) untuk memeriksa besaran paling kecil dan menyisipkan besaran yang baru, ini dengan asumsi bahwa terdapat O(n) iterasi untuk tiap item.

Seperti disebutkan di bagian atas *encoding* Huffman dirancang dengan menggabungkan sekaligus dua karakter yang paling sedikit kemungkinannya, dan perulangan proses ini hingga hanya ada satu karakter sisa. Kode pohon (*code tree*) akan dibuat dan encoding Huffman diperoleh dari label dari kode pohon. Contoh berikut mengenai proses pembentukan kode pohon.

X	p(X)
a	0.171
b	0.031
c	0.057
d	0.092
e	0.274
f	0.052
g	0.042
h	0.130
i	0.149
j	0.002

Gambar 3 Pembentukan Kode Statik Akhir



Gambar 4. Pembentukan Kode Pohon

Ada beberapa *point* yang harus diperhatikan mengenai pembentukan kode pohon ini, antara lain:

1. Tidak ada masalah bagaimana karakter-karakter tersebut diatur dan begitu juga dengan *code* akhir pohon diberi label (dengan '0' dan '1'). Bagian atas cabang diberi nilai '0' dan bagian bawah cabang diberi nilai '1'
2. *Encoding* Huffman bersifat unik
3. *Encoding* Huffman optimal dalam kasus tidak ada kehilangan dalam *fixed-to-variable length code* yang mempunyai sebuah nilai di bawah nilai rata-rata
4. *Rate* dari *code* di atas adalah 2,94 bit/karakter
5. Entropi urutan paling bawah adalah 2,88 bit/karakter

III. HASIL PENELITIAN

Hasil penelitian di dapat dengan melakukan 15 iterasi ujicoba kompresi untuk mengetahui hasil dari kompresi yang dilakukan oleh masing-masing metode, dari hasil tersebut akan dibandingkan rasio kompresi dan kecepatan kompresi dari masing-masing metode.

A. Hasil Algoritma DMC

Untuk dapat mengetahui hasil yang didapat dalam kompresi dan dekompresi ini maka akan dilakukan ujicoba sebanyak 15 kali untuk kompresi dan dekompresi.

Tabel 1. Hasil Ujicoba DMC

Uji coba	Nama File	Ukuran File (Byte)	Ukuran File Output (Byte)	Rasio Kompresi (%)	Lama Kompresi (s)
1.	Uji_1	2780	1456	52.4	8.08
2.	Uji_2	926	500	54.0	1.00
3.	Uji_3	1002	544	54.3	1.17
4.	Uji_4	2916	1539	52.8	9.12
5.	Uji_5	1904	1014	53.3	4.52
6.	Uji_6	2700	1407	52.1	7.65
7.	Uji_7	1448	779	53.8	1.98
8.	Uji_8	2895	1530	52.8	8.15
9.	Uji_9	1600	855	53.4	2.44
10.	Uji_10	2110	1119	53.0	4.16
11.	Uji_11	1872	944	50.4	3.24
12.	Uji_12	2351	1235	52.5	5.13
13.	Uji_13	2831	1476	52.1	7.47
14.	Uji_14	2640	1367	51.8	6.59
15.	Uji_15	4084	2107	51.6	16.17

Dari hasil 15 ujicoba kompresi yang dilakukan didapat rasio kompresi dan kecepatan kompresi yang di hasilkan, dari hasil tersebut jelas bahwasanya rasio dari kompresi yang dihasilkan yakni rata-rata diatas 50% dengan

kata lain besar output yang dihasilkan setelah dilakukan kompresi terhadap file maka akan mengurangi size dengan rata-rata 50%.

B. Hasil Algoritma Huffman

Tabel 2. Hasil Ujicoba Algoritma Huffman

Uji coba	Nama File	Ukuran File (Byte)	Ukuran File Output (Byte)	Rasio Kompresi (%)	Lama Kompresi (s)
1.	Uji_1	1404	948	67.5	16
2.	Uji_2	5212	4129	79.2	31
3.	Uji_3	6400	4819	75.3	62
4.	Uji_4	10026	8208	81.9	47
5.	Uji_5	22070	20129	91.2	125
6.	Uji_6	22580	17262	76.4	128
7.	Uji_7	22816	18378	80.5	132
8.	Uji_8	36538	31795	87.0	188
9.	Uji_9	36614	32769	89.5	190
10.	Uji_10	36636	30756	84.0	192
11.	Uji_11	55776	48627	87.2	266
12.	Uji_12	80856	70725	87.5	406
13.	Uji_13	179704	141211	78.6	735
14.	Uji_14	282608	247861	87.7	120
15.	Uji_15	424644	368269	86.7	178

C. Hasil Analisa

Dari hasil pengujian proses kompresi didapat bahwa rasio mempunyai *range* antara 67,52% untuk nilai terendah dan tertinggi 91,21%. Jika dicari hasil rasio kompresi tersebut secara rata-rata adalah sebesar 82,11%. Ini berarti ukuran *file* hasil adalah 0,8211 kali ukuran *file* semula dan pengurangan ukuran *file* sebesar (100% – 82,11%) = 17,89%. Nilai ini cukup terutama dalam mengkompresi *file* berukuran besar misalnya berukuran di atas 10 MB.

Dari hasil tersebut juga menunjukkan bahwa persentase kompresi atau dekompresi *file* tidak bergantung pada ukuran *file* melainkan bergantung pada isi data pada *file tersebut*. Semakin banyak perulangan data yang terdapat pada bagian *chunk* data *file* maka rasio kompresi akan semakin rendah.

Kecepatan kompresi memang tidak dilakukan pengujian tetapi dari beberapa pengujian yang dilakukan tingkat kecepatan baik untuk proses kompresi dan dekompresi berbanding lurus dengan ukuran *file* , artinya semakin besar ukuran *file* yang diproses maka semakin lama proses berlangsung.

Dari hasil 15 iterasi ujicoba yang dilakukan didapat hasil bahwasanya model huffman memiliki rasio rata-rata diatas 76% ini membuktikan bahwasanya dari segi rasio model huffman masih lebih tinggi dibandingkan dengan dmc sedangkan dari sisi kecepatan model dmc masih lebih unggul dalam hal kompresi ini.

IV. KESIMPULAN

Berdasarkan hasil dari ujicoba yang sebelumnya yang telah dilakukan maka dapat diambil beberapa kesimpulan sebagai berikut:

Penurunan rasio file algoritma Huffman ini cukup tinggi minimal 76% dari file asli yang telah di kompresi. Jadi dapat dikatakan dengan rasio kompresi ini algoritma Huffman sudah dikatakan baik dalam hal mengkompresi *file* khususnya *file* .

Tingkat kompresi dipengaruhi oleh banyaknya nada yang sama dalam *file* .

Kecepatan proses tidak bergantung pada data yang diproses tetapi berbanding lurus dengan ukuran *file* , artinya semakin besar ukuran *file* yang diproses maka semakin lama waktu prosesnya.

File yang telah dikompresi bila dilakukan proses kompresi sekali lagi maka ukuran *file* akan bertambah besar sedikit karena algoritma Huffman merupakan *optimal compression* jadi *file* yang dilakukan kompresi sebanyak dua kali maka proses terakhir tidak akan mereduksi ukuran *file* lagi. Terjadi pertambahan *byte* pada proses kompresi kedua kalinya karena program menyimpan struktur pohon Huffman dari hasil kompresi pertama.

Sedangkan untuk dari metode DMC sendiri hasil kompresi sangat tergantung pada besaran file nya, jika file nya terlalu bersar maka ini akan berpengaruh dalam hal kecepatan dan rasionya, dalam pengujian ini tidak ingin membandingan metode yang terbaik dalam hal kompresi namun ingin melihat hasil dari kecepatan dan rasio kompresi dari kedua metode ini.

REFERENCES

[1] S. Arysanti and L. I. L. Z., “ANALISIS METODE HUFFMAN UNTUK KOMPRESI DATA CITRA DAN TEKS PADA APLIKASI KOMPRESI DATA,” 2011. [Online]. Available: http://eprints.mdp.ac.id/101/. [Accessed: 30-Mar-2018].

[2] M. R. A. Neta, “Perbandingan Algoritma Kompresi Terhadap Objek Citra Menggunakan JAVA,” *Semantik 2013*, vol. 3, no. 1, pp. 224–230, Nov. 2013.

[3] M. I. Dzulhaq and A. A. Andayani, “Aplikasi Kompresi File Dengan Metode Lempel-Ziv-

Welchof," *JURNAL SISFOTEK GLOBAL*, vol. 4, no. 1, Mar. 2014.

[4] B. P. Dessyanto, C. R. Heru, and A. N. Muhammad, "APLIKASI KOMPRESI SMS BERBASIS JAVA ME DENGAN METODE KOMPRESI LZW-HUFFMAN," *Telematika*, no. 1, May 2011.

[5] A. WIDAGDO, "Implementasi Algoritma Metode Huffman Pada Kompresi Citra," s1, Universitas Muhammadiyah Surakarta, 2012.

[6] N. J. Tuturoong, "PERBANDINGAN RASIO DAN KECEPATAN KOMPRESI MENGGUNAKAN ALGORITMA HUFFMAN, LZW DAN DMC," *TEKNO*, vol. 8, no. 53, Aug. 2010.

[7] G. V. Cormack and R. N. S. Horspool, "Data Compression Using Dynamic Markov Modelling," *Comput J*, vol. 30, no. 6, pp. 541–550, Dec. 1987.

[8] A. Satyapratama, W. Widjianto, and M. Yunus, "ANALISIS PERBANDINGAN ALGORITMA LZW DAN HUFFMAN PADA KOMPRESI FILE GAMBAR BMP DAN PNG," *JURNAL TEKNOLOGI INFORMASI: Teori, Konsep, dan Implementasi*, vol. 6, no. 2, pp. 69–81, Oct. 2015.

[9] Shannon, C. E., A Mathematical Theory of Communication, The Bell System Technical Journal, Vol. 27, pp. 379 − 423, 623 − 656, July, October, 1948.