PERBANDINGAN METODE LZ77, METODE HUFFMAN DAN METODE DEFLATE TERHADAP KOMPRESI DATA TEKS

Christian Puji Nugraha[1] criz4805@gmail.com
R. Gunawan Santosa[2] gunawan@ukdw.ac.id
Lukas Chrisantyo A.A.[3] lukaschris@gmail.com

Abstract

Data compression is a very important process in the world that has been vastly using digital files, such as for texts, images, sounds or videos. Those digital files has a varied size and often taking disk storage spaces. To overcome this problem, many experts created compression algorithms, both for lossy and lossless compression. This research discusses about testing of four lossless compression algorithms that applied for text files, such as LZ77, Static Huffman, LZ77 combined with Static Huffman, and Deflate. Performance comparison of the four algorithms is measured by obtaining the compression ratio. From the test results can be concluded that the Deflate algorithm is the best algorithm due to the use of multiple modes, i.e. uncompressed mode, LZ77 combined with Static Huffman mode, and LZ77 combined with Dynamic Huffman Coding mode. The results also showed that the Deflate algorithm can compress text files and generates an average compression ratio of 38.84%.

Kata kunci : deflate, LZ77, Huffman, kompresi teks

1. Pendahuluan

Sekarang perkembangan teknologi telah berkembang pesat, banyak data-data yang dahulu dibukukan sekarang disimpan menggunakan file digital. Ruang penyimpanan yang kecil menjadi masalah ketika file digital yang akan disimpan berukuran sangat besar melebihi ruang penyimpanan yang tersisa. Untuk mengatasi masalah tersebut, banyak para ahli telah menemukan berbagai macam algoritma untuk melakukan pemampatan data/kompresi data, diantaranya seperti LZMA, LZ77, LZ78, *Static Huffman*, *Adaptive Huffman*, *Shannon-Fano, Dynamic Markov Compression*, Deflate, LIFO, *Run-Length, Prediction by Partial Matching, Burrow-Wheeler Block Sorting,* dan masih banyak lainnya.

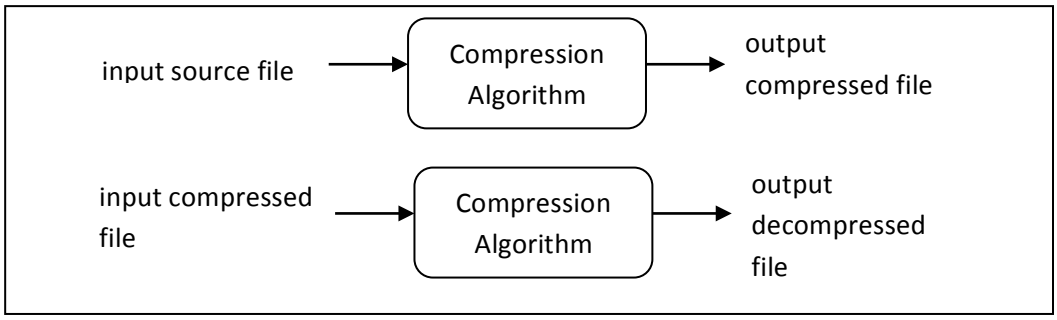
Penulis dalam penelitian ini hanya melakukan implementasi menggunakan algoritma LZ77, algoritma *Static Huffman*, penggabungan algoritma LZ77 dan *Static Huffman*, serta algoritma Deflate.

2. Landasan Teori

2.1. Pengertian Kompresi Data

Menurut Salomon, D. (2007) mengemukakan bahwa definisi kompresi data merupakan proses konversi dari aliran data input (sumber aliran data atau data asli) ke aliran data lain (output, aliran data bit atau data terkompresi) yang memiliki ukuran lebih kecil.

[1] *Program Studi Teknik Informatika, Fakultas Teknologi Informasi,Universitas Kristen Duta Wacana*
[2] *Program Studi Teknik Informatika, Fakultas Teknologi Informasi,Universitas Kristen Duta Wacana*
[3] *Program Studi Teknik Informatika, Fakultas Teknologi Informasi,Universitas Kristen Duta Wacana*



Gambar 1. Alur Proses Kompresi dan Dekompresi
Dikutip dari : Pu, I.M. (2006). Fundamental Data Compression.
London: Butterworth-Heinemann.

2.2. Jenis-Jenis Kompresi Data

Berdasarkan hasil proses kompresi data yang dihasilkan dan pengembalian hasil kompresi ke data sebelum terkena kompresinya, jenis kompresi data dibedakan menjadi dua yaitu Kompresi *Loseless* dan Kompresi *Lossy*.

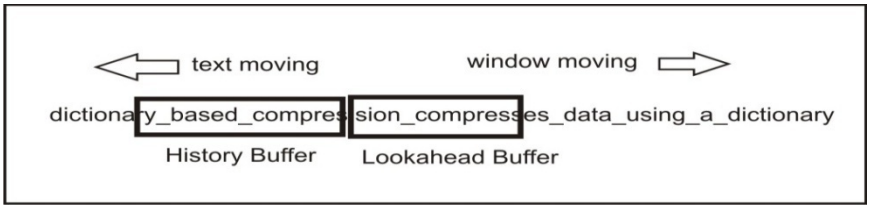
Kompresi *Loseless* merupakan jenis kompresi yang *reversible*, yakni data yang sudah terkena kompresi dimungkinkan untuk membentuk kembali tepat sama dengan data awal sebelum terkena proses kompresi. Jenis kompresi ini sering diterapkan dalam file teks.

Kompresi *Lossy* merupakan jenis kompresi irreversible, yakni yang menghasilkan data yang berbeda apabila dilakukan proses dekompresi. Data hasil dekompresinya mengalami kehilangan beberapa data yang tidak sesuai dengan data awal sebelum terkena proses kompresi, jenis kompresi ini sering diterapkan dalam file gambar, suara maupun video.

2.3. Algoritma LZ77

2.3.1 Definisi Algoritma LZ77

Algoritma LZ77 (Lempel Ziv 1977) merupakan algoritma kompresi yang bersifat *Loseless*, yang dikembangkan oleh Abraham Lempel dan Jacob Ziv pada tahun 1977. Metode Kompresi LZ77 merupakan metode kompresi *Sliding Windows Compression*, struktur data yang berupa *text windows* akan dibagi menjadi 2 bagian, terdiri dari teks yang sudah dikodekan (*history buffer*) dan bagian lain dari teks yang akan dikodekan (*lookahead buffer*). Ukuran *buffer* dari masing-masing ditetapkan sebelumnya dalam implementasi, *history buffer* akan memiliki panjang hingga beberapa ribu byte dan *lookahead buffer* memiliki panjang hanya puluhan byte (Salomon,D., 2007).



Gambar 2. Tampilan Jendela Kompresi LZ77
Dikutip dari : Pu, I.M. (2006). Fundamental Data Compression.
London: Butterworth-Heinemann.

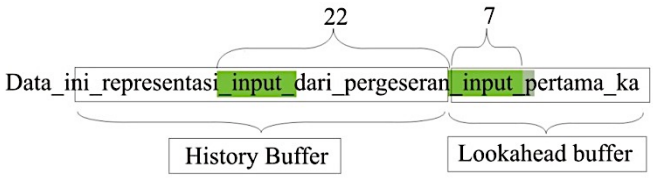
2.3.2 Tahap Kompresi Algoritma LZ77

Dalam melakukan kompresi LZ77, ada beberapa tahapan yang harus dilakukan. Pada awalnya harus ditentukan dulu besar *history buffer* dan besar *lookahead buffer*. Kemudian isikan barisan karakter inputan ke dalam ruang yang tersedia di *lookahead buffer*. Apabila *history buffer* masih kosong atau tidak ada karakter pola yang sama di dalam *lookahead buffer* dibandingkan dengan *history buffer*, maka keluarkan token dengan format *offset* bernilai 0, *length match* bernilai 0, lalu diikuti 1 karakter *missmatch* di awal *lookahead buffer*. Namun

apabila ditemukan pola maka keluarkan token dengan format *offset* bernilai jarak dari *history buffer* ditemukannya pola, *length match* bernilai panjang pola yang ditemukan sama, dan karakter *missmatch* diisi 1 karakter diluar dari pola yang ditemukan. Kemudian potong karakter yang sudah diubah menjadi token tersebut dari *lookahead buffer* dan diisikan ke dalam *history buffer*, yang menandakan karakter tersebut telah diproses kompresi. Isi *lookahead buffer* hingga penuh dari barisan inputan yang tersisa, dan lakukan proses pencocokan pola karakter seperti tadi hingga ditemukan *end of file*. Apabila sudah ditemukan *end of file* dan *lookahead buffer* sudah tidak terdapat karakter tersisa, maka akan didapatkan token hasil kompresi LZ77.

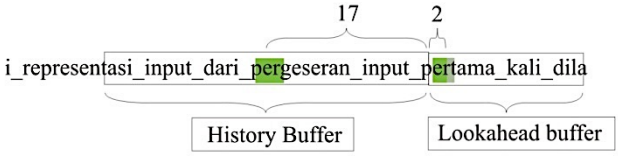
2.3.3 Contoh Penerapan Algoritma LZ77

Contoh penerapan kompresi pada LZ77 tepatnya tentang pembuatan token akan dijelaskan di sini. Diketahui terdapat barisan data *string* sebagai berikut: "Data_ini_representasi_input_dari_pergeseran_input_pertama_kali_dilakukan". Misalkan di barisan tersebut telah dilakukan proses kompresi hingga string "Data_ini_representasi_input_dari_pergeseran". Dengan kata lain, barisan string tersebut berada di *history buffer*, barisan sisanya akan berada di *lookahead buffer* dan data inputan. Mari perhatikan gambar 3, yang merupakan tampilan dari jendela LZ77 yang diterapkan.



Gambar 3. Jendela Kompresi LZ77

Dari gambar 3 tepatnya pada *lookahead buffer*, dapat dilihat terdapat pola kesamaan dengan *history buffer* sepanjang 7 karakter yaitu barisan string "_input_". Maka keluarkan nilai token dari pola yang ditemukan tersebut. Formatnya akan dituliskan seperti ini (22,7,p) yang berasal dari 22 yang merupakan nilai offset, 7 yang merupakan panjang pola, dan p adalah karakter missmatch. Kemudian dilakukan pergeseran jendela hingga menjadi terlihat seperti pada gambar 4.



Gambar 4. Pergeseran Jendela Kompresi LZ77

Dari gambar 4 terlihat *history buffer* dan *lookahead buffer* bergeser sepanjang token pola yang ditemukan pada gambar 3. Pada *lookahead buffer* juga diisi kembali sebanyak ruang yang tersedia, dengan mengambil data dari sisa inputan. Setelah itu dilakukan proses pembentukan token lagi, yang diketahui pada gambar 4 akan menghasilkan token (17,2,t). Lalu lakukan pergeseran dan pembentukan token lagi hingga *lookahead buffer* berisi *end of file*.

2.4 Algoritma Static Huffman

2.4.1 Definisi Algoritma Static Huffman

Algoritma Huffman pada awalnya diperkenalkan oleh David Huffman pada tahun 1952, yang mana metode ini merupakan metode yang paling terkenal di dalam kompresi teks. Metode Kompresi Huffman melakukan penganalisaan terlebih dahulu terhadap string masukan yang akan diproses kompresi, selanjutnya nanti akan dibuat pohon Huffman yang merupakan pohon biner dengan kode pengganti yang optimal untuk simbol-simbol dengan probabilitas kemunculan yang lebih tinggi (Sayood,K. 2003).

2.4.2 Tahap Kompresi Algoritma Static Huffman

Tahap kompresi algoritma Huffman dimulai dengan melakukan pembentukan pohon biner Huffman. Adapun langkah pembentukkan pohon Huffman sebagai berikut:

1. Baca seluruh karakter unik di dalam barisan inputan dan hitung frekuensi kemunculannya.
2. Urutkan karakter berdasarkan frekuensi yang terkecil hingga terbesar.
3. Ambil 2 karakter yang memiliki probabilitas frekuensi terkecil dan jadikan ke dalam 1 node. Dalam node tersebut nilai probabilitas frekuensi dari kedua karakter akan dijumlahkan.
4. Ulangi pengambilan 2 karakter seperti pada poin 3, hingga seluruh karakter dikenai proses tersebut dan menjadi pohon biner.
5. Berikan nilai setiap simpul kiri pohon biner dengan 0 dan setiap simpul kanan dengan 1.
6. Proses pembentukan pohon Huffman selesai dan untuk melakukan pembacaan dilakukan dari root atau puncak pohon hingga menemukan karakter yang dicari di daun.

2.4.3 Contoh Penerapan Algoritma Static Huffman

Contoh penerapan algoritma *Static Huffman* adalah sebagai berikut, misal terdapat barisan karakter “abibadibib”. Diketahui bahwa setiap karakter di dalam komputer dikodekan dalam 1 Byte atau sebesar 8 bit. Barisan karakter “abibadibib” tersebut terdapat 10 karakter maka dapat dihitung menjadi: 10 x 8bit = 80bit. Barisan karakter tersebut dapat dikatakan membutuhkan ruang penyimpanan sebesar 80 bit atau setara dengan 8 Byte. Karakter-karakter “abibadibib” tersebut memiliki rincian kode bit yang dikenali komputer sebagai berikut:

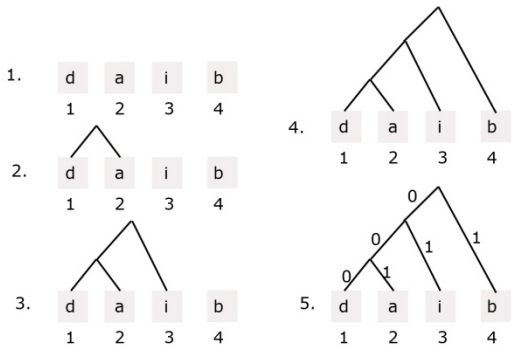
Tabel 1
Kode ASCII Representasi “abibadibib”

Karakter	Kode Representasi
a	01100001
b	01100010
i	01101001
b	01100010
a	01100001
d	01100100
i	01101001
b	01100010
i	01101001
b	01100010

Setelah mengetahui pemborosan bit tersebut, maka diterapkan kompresi Huffman pada barisan tersebut. Maka langkah yang perlu dilakukan adalah menerapkan langkah-langkah kompresi yaitu pembentukan pohon Huffman sebagai berikut:

1. Cek karakter unik dalam barisan dan hitung frekuensi kemunculannya. Urutkan dari frekuensi yang terkecil. Diketahui [d] memiliki 1 kemunculan, [a] memiliki 2 kemunculan, [i] memiliki 3 kemunculan, dan [b] memiliki 4 kemunculan.
2. Ambil 2 node terkecil yakni karakter [d] dan [a], gabung jadi 1 node [da].
3. Ambil 2 node terkecil yakni [da] dan [i], gabung jadi 1node [dai].
4. Ambil 2 node terkecil yakni [dai] dan [b], gabung jadi 1node [daib].
5. Beri nilai tiap simpul kiri dengan 0 dan tiap simpul kanan dengan 1.

Gambar langkah-langkah pembentukan pohon Huffman yang diterapkan pada barisan “abibadibib” dapat dilihat pada gambar 5.



Gambar 5. Pembentukan Pohon Huffman barisan "abibadibib"

Dengan menerapkan pohon Huffman akan menghasilkan penghematan terhadap bit yang seharusnya dimiliki tiap karakter yang ada pada barisan "abibadibib". Hal tersebut ditunjukkan pada tabel 2.

Tabel 2
Kode *Static Huffman* untuk "abibadibib"

Karakter	Kode Representasi
a	001
b	1
d	000
i	01

Dengan melakukan pengkodean ulang terhadap barisan karakter "abibadibib" maka akan didapatkan barisan bit: 0011011001000011011. Diketahui bahwa karakter [b] yang pada pengkodean ASCII membutuhkan ruang sebesar 8 bit sekarang hanya membutuhkan 1 bit, maka panjang bit karakter [b] yang terdapat di "abibadibib" awalnya sebesar 4*8bit = 32bit menjadi hanya sebesar 4*1bit = 4 bit saja. Baris karakter "abibadibib" yang mulanya membutuhkan ruang sebesar 80 bit sekarang hanya membutuhkan sebesar 19 bit.

2.5 Algoritma Penggabungan LZ77 dan Static Huffman

2.5.1 Definisi Algoritma Penggabungan LZ77 dan Static Huffman

Prinsip dari metode penggabungan kompresi LZ77 dan *Static Huffman* ini adalah melakukan 2 tahapan kompresi terhadap barisan karakter masukan. Proses yang pertama dilakukan adalah dengan melakukan kompresi LZ77 terhadap barisan karakter masukan. Diketahui bahwa algoritma kompresi LZ77 akan melakukan proses kompresi dengan melakukan *sliding window* terhadap barisan masukan dan kemudian mengeluarkan *token* hasil proses tersebut. Proses *sliding window* ini dilakukan hingga menemukan *end of file* atau bisa juga dikatakan telah didapatkan seluruh barisan token kompresi LZ77 dari barisan karakter masukan. Tahap selanjutnya adalah mengkodekan kembali barisan token yang dihasilkan pada kompresi LZ77 menggunakan algoritma kompresi *Static Huffman*. Proses pengkodean *token* tentu dengan cara mengambil tiap karakter pada *token* yang dihasilkan dalam proses kompresi LZ77. Selanjutnya akan dilakukan proses penghitungan frekuensi kemunculan tiap karakter yang ada pada barisan *token*. Setelah selesai melakukan penghitungan frekuensi kemunculan karakter, maka perlu dibuat pohon *Static Huffman* dari barisan *token*. Seluruh barisan *token* akan dikodekan ulang bitnya hingga menemukan *end of file*, dengan demikian akan didapatkan barisan bit terkompresi.

2.6 Algoritma Deflate

2.6.1 Definisi Algoritma Deflate

Kompresi Deflate awal mulanya merupakan konsep kompresi yang dibuat dan diimplementasikan oleh Philip Katz, salah satu implementasinya telah diterapkan di dalam

kompresi file Zip. Program yang dibuat oleh Philip Katz dan mengimplementasi algoritma kompresi Deflate adalah software PKZIP. Metode kompresi Deflate merupakan kombinasi dari variant LZ77 dan metode Huffman (Salomon, D. 2007). Algoritma kompresi Deflate ini bersifat *loseless* sama seperti algoritma kompresi LZ77 dan algoritma kompresi *Static Huffman*. Metode *encoding* pada Deflate telah dimuat dalam dokumen *Request For Comments* 1951. Ada 4 mode kompresi yang digunakan di dalam Deflate, namun mode keempat tidak digunakan. Adapun mode kompresi pada *compressor* Deflate yaitu:

1. Tidak dikompresi sama sekali. Kompresi dengan metode ini merupakan kompresi yang inputan langsung diberikan kepada output.
2. Kompresi dimulai dengan metode LZ77 dan diteruskan dengan pengkodean pohon Huffman. Pohon Huffman dalam mode ini merupakan pohon *Static Huffman*, jadi tidak memerlukan ruang ekstra untuk menyimpan pohon tersebut.
3. Kompresi dimulai dengan metode LZ77 dan diteruskan dengan pengkodean pohon Huffman. Pada mode ini terdapat perbedaan karena dirancang oleh kompresor dan disimpan bersama data-data yang dikompresi. Pohon Huffman ini disebut juga sebagai pohon Huffman dinamik.
4. *Reserved*.

2.6.2 Tahap Kompresi Algoritma Deflate

Adapun tahap kompresi Deflate dimulai dengan melakukan eliminasi karakter string kembar (implementasi algoritma kompresi LZ77) yang dilakukan dari hasil pembacaan tiap block yang diproses. Apabila pada awal block tidak ditemukan pola maka akan mengaktifkan mode 1 pada block tersebut, yakni mode tidak dikompresi. Ketika ditemukan pola karakter kembar maka dituliskan sebuah referensi berisi panjang sekuen dan jaraknya dari awal block. Kemungkinan panjang block adalah 2-258 Bytes dan kemungkinan jarak *sliding window*nya adalah 1-32.768 Bytes.

Tahap berikutnya adalah penggunaan pohon Huffman, yaitu mengganti setiap data yang sering muncul dengan simbol tertentu yang memiliki bit yang lebih pendek. Mode kedua menggunakan pohon *Static Huffman* yang artinya karakter yang dibutuhkan untuk melakukan dekompresi file tidak disertakan dalam file terkompresi. Mode ketiga adalah menggunakan *Dynamic Huffman Compression*, yang artinya harus mendapatkan seluruh karakter yang akan dilakukan untuk melakukan kompresi terhadap block. Karakter tersebut disimpan setelah data, dan sifat karakter *encode* ini adalah *fixed*. Penggolongan karakter *encode Dynamic Huffman Compression* dapat dilihat pada RFC 1951 (Deutsch,1996a). Nantinya akan didapatkan hasil tiap block kompresi yang merupakan hasil teks terkompresi.

3. Perancangan Program

Saat program mulai dijalankan, pengguna akan melakukan pemilihan metode kompresi/dekompresi yang akan digunakan, file input, file output dan parameter yang dibutuhkan metode kompresi tersebut. Pada saat menerima perintah kompresi/dekompresi, maka program akan melakukan proses pengolahan data input dan diproses dengan metode yang dipilih oleh pengguna. Setelah proses pengolahan data input selesai, program akan melakukan penyimpanan ke dalam file output serta akan menampilkan hasil analisis dari proses tersebut.

4. Hasil dan Pembahasan

Penelitian ini dibuat untuk meneliti perbandingan algoritma kompresi yang memiliki rasio kompresi yang efektif antara algoritma LZ77, algoritma *Static Huffman*, algoritma penggabungan LZ77 dan *Static Huffman*, serta algoritma Deflate. Secara garis besar terdapat dua pengujian yang dilakukan oleh penulis yakni pengujian kompresi terhadap pola karakter yang diambil 5 buah sampel file teks dan pengujian kompresi terhadap 20 file teks cerita atau novel. Adapun daftar pengujian pola karakter diterapkan pada spesifikasi pada tabel 3.

Tabel 3.
Daftar File Pengujian Pola

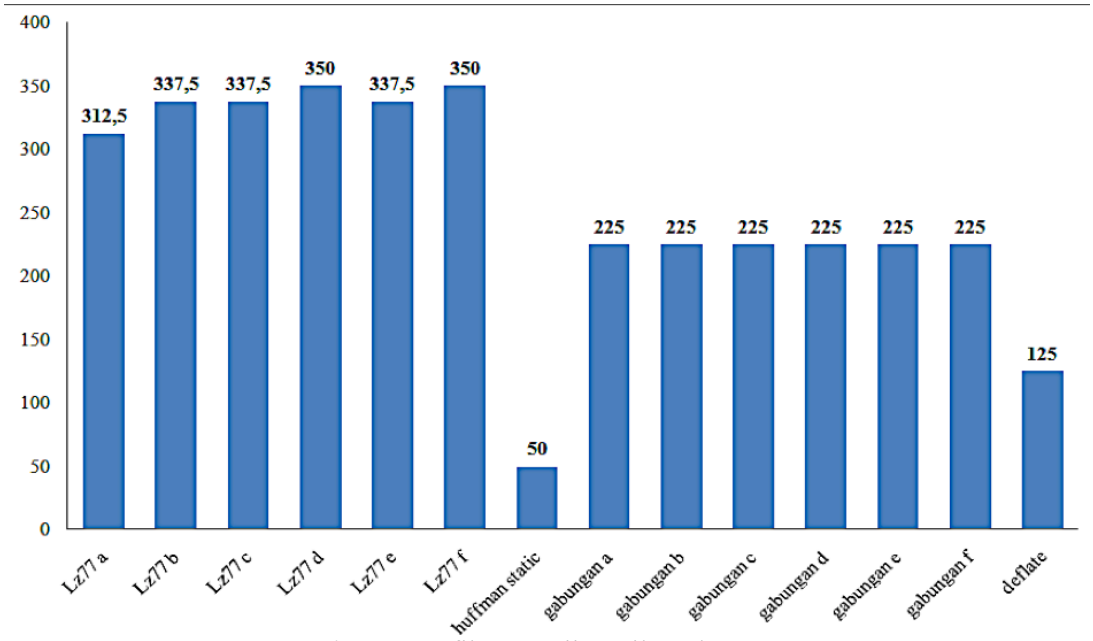
No	Nama File	Isi File	Pola	Karakter Unik (n)	Besar File (Bytes)
1	polaTest1.txt	Hanya berisi karakter unik. Isi barisan karakter tersebut adalah "tes_pola".	Tidak Sama	8	8
2	polaTest2.txt	Berisi karakter yang memiliki pola kesamaan dalam barisan masukan. Isi barisan karakternya adalah "tes_pola" yang diulang sebanyak 5.000 kali.	Sama	8	40.000
3	polaTest3.txt	Berisi 1 karakter unik yang memiliki pola kesamaan dalam barisan masukan. Isi karakternya adalah "a" yang diulang sebanyak 40.000 kali.	Sama	1	40.000
4	polaTest4.txt	Berisi karakter ASCII random sebanyak 40.000 karakter.	Tidak Sama	n>1 , n<=128	40.000
5	polaTest5.txt	Berisi 26 karakter abjad (a s/d z) yang berurutan dan diulang 1.500 kali.	Sama	26	39.000

Pengujian 5 teks pola maupun 20 file cerita tersebut akan diterapkan dengan spesifikasi pengujian yang telah penulis persiapkan dan ditunjukkan pada tabel 4. Hal tersebut dilakukan untuk mengetahui efektifitas dari kinerja masing-masing algoritma kompresi, yang mana diketahui untuk algoritma kompresi LZ77 membutuhkan 2 parameter inputan yaitu besar *history buffer* dan besar *lookahead buffer*.

Tabel 4.
Kamus Data File Pengujian

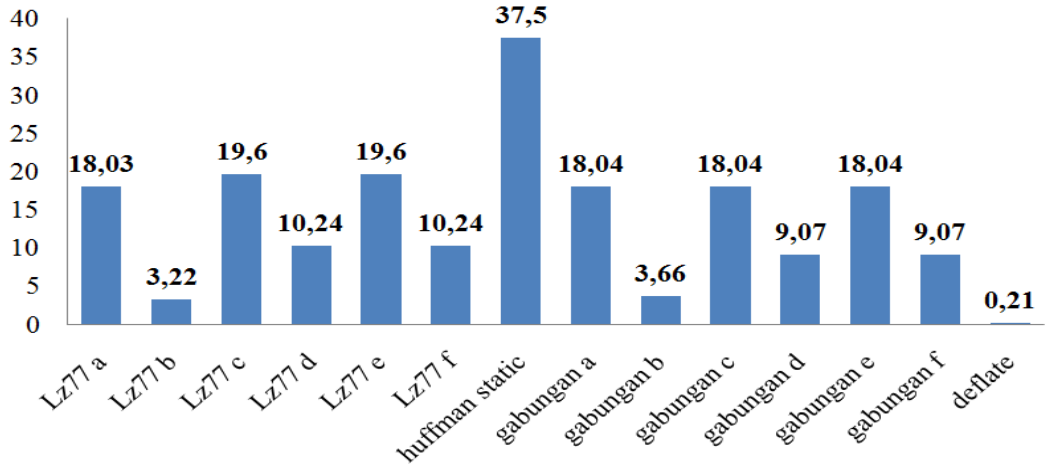
No	Jenis Test Kompresi	Parameter		Nama Alias Tambahan	File Hasil
		Besar History buffer	Besar Lookahead Buffer		
1	Kompresi LZ77	1000	16	A	.lz77
		1000	100	B	
		3000	16	C	
		3000	32	D	
		4000	16	E	
		4000	32	F	
2	Kompresi *Static Huffman*	-	-	-	.huf
3	Kompresi Gabungan LZ77 dan *Static Huffman*	1000	16	A	.lzHuf
		1000	100	B	
		3000	16	C	
		3000	32	D	
		4000	16	E	
		4000	32	F	
4	Kompresi Deflate	-	-	-	.Deflate

Dari hasil pengujian pada file polaTest1.txt, diketahui bahwa setiap algoritma yang menggunakan LZ77 dan Deflate menghasilkan rasio yang sangat besar. Hal tersebut dikarenakan tidak ditemukannya pola karakter yang sama di dalam file tersebut.



Gambar 6. Grafik Pengujian File polaTest1.txt

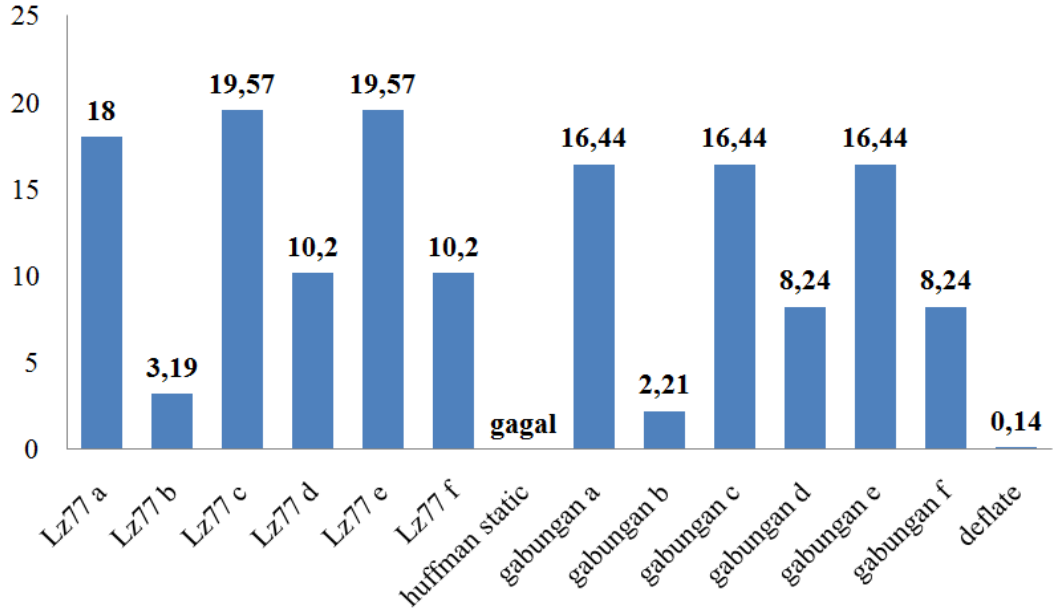
Dari hasil pengujian file polaTest2.txt, dapat dilihat bahwa untuk semua jenis kompresi yang menggunakan algoritma LZ77, gabungan LZ77 dan *Static Huffman*, dan Deflate menghasilkan rasio kompresi yang jauh lebih kecil dibandingkan *Static Huffman*. Hal tersebut terjadi dikarenakan ditemukannya karakter yang sama dalam barisan karakter tersebut. Apabila diperhatikan dengan seksama, dapat dilihat juga pengaruh penambahan besar *history buffer* maupun *lookahead buffer* pada kompresi menggunakan algoritma LZ77 akan membuat rasio kompresi menjadi jauh lebih baik. Dapat dikatakan, bila barisan karakter *history buffer* memiliki seluruh pola yang diminta dari *lookahead buffer* maka akan semakin sedikit *token* yang terbentuk dan tentu akan mempengaruhi besar file terkompresinya.



Gambar 7. Grafik Pengujian File polaTest2.txt

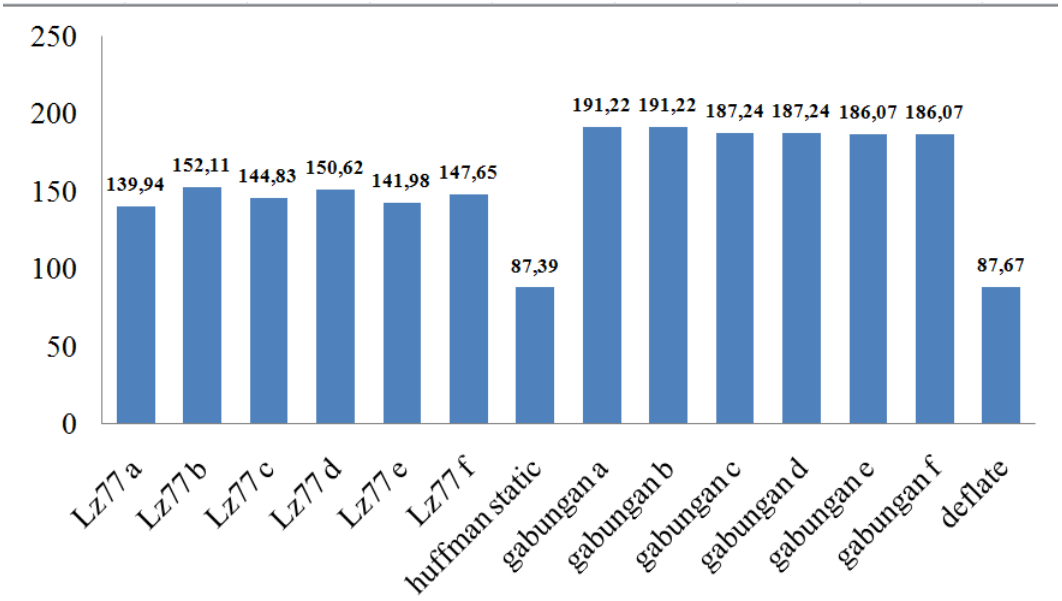
Dari hasil pengujian file polaTest3.txt, dapat dilihat bahwa hasil rasio kompresi yang menggunakan algoritma LZ77, gabungan LZ77 dan *Static Huffman*, serta Deflate memiliki hasil rasio yang hampir sama dengan pengujian file "polaTest2.txt" sebelumnya. Hal tersebut terjadi

karena pola yang sama selalu ditemukan. Namun pada kompresi yang hanya menggunakan algoritma *Static Huffman* menghasilkan kegagalan dalam kompresi, hal tersebut disebabkan pada implementasi yang penulis lakukan, pembentukan pohon *Static Huffman* minimal membutuhkan 2 karakter unik. Diketahui bahwa pada file "polaTest3.txt" hanya memiliki 1 karakter unik, jadi kegagalan disebabkan tidak terjadinya pembuatan pohon *Static Huffman*.



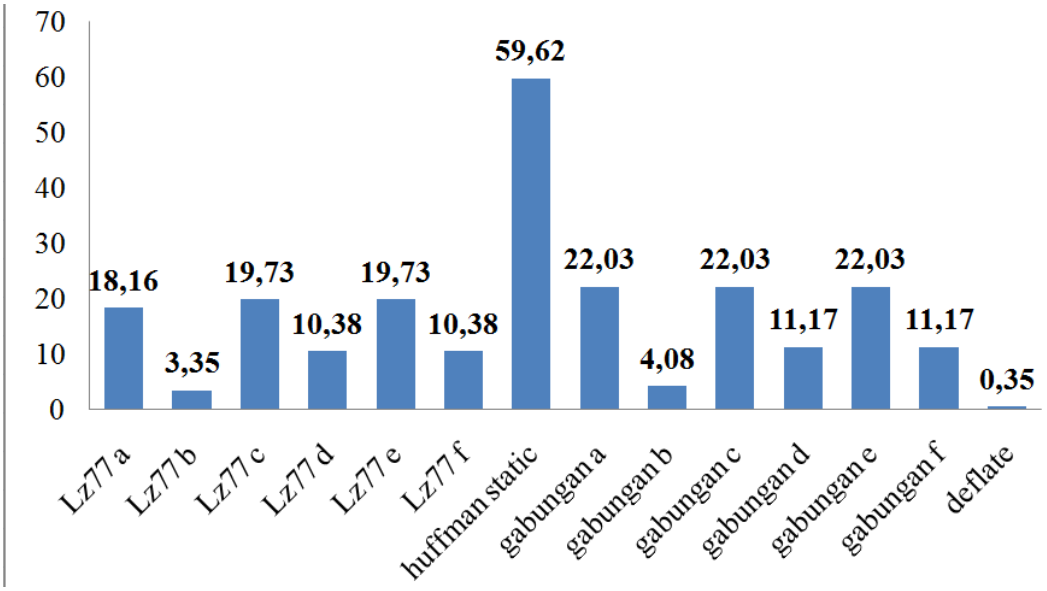
Gambar 8. Grafik Pengujian File polaTest3.txt

Dari hasil pengujian file polaTest4.txt, kompresi yang hanya menggunakan algoritma *Static Huffman* menghasilkan rasio kompresi yang terbaik seperti pada pengujian file "polaTest1.txt". Hal tersebut dikarenakan pengkodean nilai bit karakter yang menjadi berkurang dari awalnya 8 Byte setiap karakter menjadi bervariasi tergantung dari pembentukan pohon *Static Huffman*. Pada kompresi yang menerapkan algoritma Deflate juga mendapatkan rasio yang lebih baik jika dibandingkan dengan pengujian file "polaTest1.txt" sebelumnya. Hal ini terjadi dikarenakan ditemukannya pola perulangan dari file "polaTest4.txt" meskipun hanya sedikit saja, berbeda dengan file "polaTest4.txt" yang tidak ada pola perulangan karakter sama sekali. Untuk kompresi yang menerapkan algoritma LZ77 maupun gabungan LZ77 dan *Static Huffman* menghasilkan rasio yang masih besar, bahkan melebihi ukuran file aslinya. Bila dibandingkan pada hasil pengujian "polaTest1.txt" dan "polaTest4.txt", rasio kompresi pada "polaTest4.txt" jauh lebih baik. Rasio kompresi tersebut terjadi dikarenakan ditemukannya pola karakter yang sama, namun jumlahnya sedikit.



Gambar 9. Grafik Pengujian File polaTest4.txt

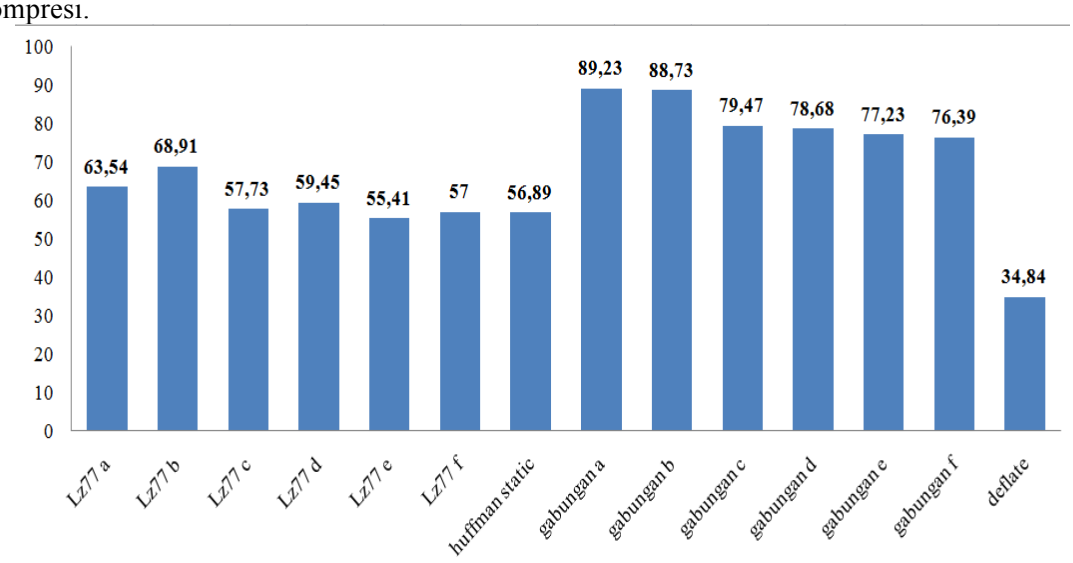
Dari hasil pengujian file polaTest5.txt, dapat dilihat bahwa hasil rasio yang dihasilkan tidak begitu jauh berbeda dengan pengujian file "polaTest2.txt", hal tersebut didapatkan pada kompresi yang menggunakan algoritma LZ77, gabungan LZ77 dan *Static Huffman*, serta Deflate mendapatkan pola yang diminta *lookahead buffer* pada *history buffer*.



Gambar 10. Grafik Pengujian File polaTest5.txt

Dari hasil pengujian file cerita, terlihat bahwa kompresi file teks menggunakan algoritma Deflate mendapatkan rasio kompresi yang terbaik, diikuti dibelakangnya *Static Huffman,* LZ77, dan yang terburuk adalah penggabungan LZ77 dan *Static Huffman*. Algoritma Deflate mendapatkan rasio yang terbaik karena menerapkan 4 mode block dalam melakukan kompresi file teks yaitu mode tidak dikompresi, kompresi LZ77 dan *Static Huffman*, kompresi LZ77 dan Huffman Dinamik, *reserved*. Hal tersebut dikarenakan *compressor* Deflate dapat memperhitungkan apabila hasil token dalam proses LZ77 akan membutuhkan banyak ruang bit dibandingkan karakternya, maka *compressor* Deflate secara langsung akan menggunakan mode tidak dikompresi dalam block tersebut. Masih juga terdapat 2 mode yang berperan penting yaitu

pohon *Static Huffman* maupun pohon Huffman Dinamik yang diterapkan dalam block data kompresi.



Gambar 11. Grafik Rasio Rata-Rata Kompresi Teks Cerita

Setelah seluruh pengujian kompresi file diterapkan, selanjutnya adalah pengujian pengembalian file terkompresi ke file teks. Hasil pengujiannya ditunjukkan pada Tabel 5.

Tabel 5.
Daftar Hasil Pengujian Dekompresi

No	Jenis Pengujian	Kompresi	Dekompresi
1	File Pengujian Kompresi Pola Teks	Berhasil	Berhasil dan sama seperti file mula.
		Gagal Kompresi pada file "polaTest3.txt" menggunakan Static Huffman	Tidak ada file yang tepat untuk didekompresi, tidak bisa untuk didekompresi seperti file mula.
2	File Pengujian Kompresi Teks Cerita	Berhasil	Berhasil dan sama seperti file mula.

5. Kesimpulan

Berdasarkan analisa yang dilakukan terhadap sistem yang dibuat penulis dan mengacu pada hasil pengamatan yang telah dilakukan pada proses pengujian, maka dapat diambil kesimpulan sebagai berikut :

1. Pada metode kompresi LZ77, semakin besar *history buffer* dan *lookahead buffer* akan membuat banyak kemungkinan pola teks yang ditemukan. Adanya banyak pola teks yang sama pada file sangat berpengaruh terhadap kecilnya ukuran hasil kompresi.
2. Pada metode kompresi *Static Huffman,* semakin sedikit karakter unik yang muncul dalam suatu teks akan menghasilkan file terkompresi yang semakin kecil hal ini dikarenakan pengubahan kode karakter menggunakan pohon Huffman. Dekompresi pada metode ini sangat berpengaruh dengan kamus karakter yang dihasilkan dari pohon Huffman yang dibuat pada proses kompresi.
3. Pada kompresi gabungan LZ77 dan *Static Huffman*, tidak menghasilkan file terkompresi dengan rasio yang lebih baik bila dibandingkan hanya menggunakan LZ77 ataupun *Static Huffman* saja. Hal tersebut dikarenakan *token* yang dihasilkan pada LZ77 memungkinkan adanya banyak karakter unik pada kompresi *Static Huffman*.

4. Pada kompresi Deflate, file terkompresi yang dihasilkan ukurannya jauh lebih kecil dibandingkan dengan kompresi LZ77, kompresi *Static Huffman*, maupun gabungan (LZ77 dan *Static Huffman*). Ukuran file terkompresi yang lebih kecil ini dipengaruhi dari *block* data yang terbagi menjadi 4 bagian yang diterapkan dalam proses kompresi yaitu mode tidak dikompresi, kompresi LZ77 dan *Static Huffman*, kompresi LZ77 dan Huffman Dinamik, serta *reserved*.
5. Dari ujicoba perbandingan antara hasil rasio kompresi Deflate dan kompresi gabungan (LZ77 dan *Static Huffman*), diketahui bahwa algoritma gabungan menghasilkan rasio yang jauh lebih buruk dibandingkan algoritma kompresi Deflate karena tidak menerapkan metode block data seperti yang ada pada algoritma Deflate.
6. Rasio rata-rata kompresi dari hasil implementasi sistem secara berturut-turut dari yang terbaik ke yang kurang adalah algoritma Deflate, algoritma *Static Huffman,* algoritma LZ77, dan algoritma gabungan (algoritma LZ77 dan *Static Huffman*).
7. File teks yang berhasil diolah menggunakan kompresi LZ77, *Static Huffman,* gabungan (LZ77 dan *Static Huffman*), maupun Deflate dapat dikembalikan sesuai dengan file aslinya.

Daftar Pustaka

Antaneus Feldspar. (2002). *An Explanation of the Deflate Algorithm.* Diakses pada tanggal 13 Oktober 2013 dari http://zlib.net/feldspar.html

Corneliussen, A., Poulsen, E., Silpakar, P., & Østeraa, T. (2009). *ZIP-file encoding & decoding using DEFLATE*. Diakses pada tanggal 25 April 2013 dari http://www.cvmt.dk/education/teaching/f09/VGIS8/MultiMediaData/ZIP-09gr840.pdf

Deutsch, L.P. (1996a). *Deflate Compressed Data Format Spesification Version 1.3*. Networking Working Group - RFC 1951.

Pu, Ida Mengyi. (2006). *Fundamental Data Compression.* London: Butterworth-Heinemann.

Sayood, K. (2003). *Lossless Compression Handbook*. California: Academic Press.

Salomon, D. (2007). *Data Compression The Complete Reference 4th Edition*. London: Springer-Verlag.

Salomon, D. & Motta, G. (2010). *Handbook of Data Compression 5th Edition*. London: Springer-Verlag.