MIND Journal | ISSN (p): 2528-0015 | ISSN (e): 2528-0902 | Vol. 2 | No. 1 | Halaman 68 - 81
DOI : https://doi.org/10.26760/mindjournal Januari 2017

Implementasi Algoritma Lzss pada Aplikasi Kompresi dan Dekompresi File Dokumen

JASMAN PARDEDE,MIRA MUSRINI B, LUQMAN YUDHIANTO

Jurusan Teknik Informatika, Fakultas Teknologi Industri
Institut Teknologi Nasional Bandung

Email: sangkuriang26@yahooo.com

Informasi yang bersifat dinamis dan berubah dalam hitungan detik harus dikirimkan saat itu juga agar informasi yang didapat sesuai dengan keadaan. Dokumen yang berukuran besar menimbulkan masalah bila kecepatan unggah pengirim lambat atau kecepatan unduh penerima lambat atau koneksi internet tidak stabil, informasi tidak bisa langsung diproses dikarenakan waktu transfer data yang terlampau lama. Kompresi file dibutuhkan agar dokumen yang dikirimkan dapat lebih kecil dari ukuran aslinya sehingga waktu pengiriman file dokumen tidak terlalu lama. Dalam penelitian ini dikembangkan aplikasi kompresi fileteks dengan menggunakan metode Lempel Ziv Storer Szymanski (LZSS) dengan menggunakan bahasa pemograman java. Metode Lempel Ziv Storer Szymanski mencari kesamaan string antara dictionary buffer dan look ahead buffer menghasilkan token yang ukuran lebih kecil dari string yang diwakili. Berdasarkan pengujian aplikasi, file teks yang berformat .doc menghasilkan rasio kompresi terbaik yaitu 30%. Metode Lempel Ziv Storer Szymanski bekerja dengan baik pada file teks yang isinya tidak memiliki gambar.

***Kata kunci** : Dokumen, Algoritma LZSS, Kompresi*

ABSTRACT

The information that dynamic and changing in seconds should be sent at that time so the information obtained realtime. Large document will cause problems when the sender upload speedwas slow or recipient download speedwas slow or theinternet connection unstable, the information can not be processed due data transfer timewas delayed. File compression is needed to reduce the sizeof document that will be sent to smaller than original sizeso the delivery time is not delayed. In this research will develop a document file compression application using Lempel Ziv Storer Szymanski (LZSS) using java programming language.Lempel Ziv Storer Szymanski method looking for similarities between the dictionary string buffer and look ahead buffer produces tokens that the smaller size of the string being represented. According to the application test, text file that .doc extension produces the best compression ratio that is 30%. Lempel Ziv Storer Szymanski works good on text file that not contained any picture.

***Keyword**: Document, LZSS Algorithm, Compression*

1. PENDAHULUAN

Informasi penting yang dikirim dari satu instansi ke instansi lain membutuhkan waktu pengiriman data yang cepat agar informasi yang didapat sesuai dengan keadaan yang terbaru. Isi dari dokumen tersebut berkembang tiap waktunya dan menghasilkan *file* dokumen yang semakin besar. Untuk dapat mengirim dokumen berukuran besar sesegera mungkin dengan kecepatan unggah yang lambat dibutuhkan suatu solusi yaitu dengan teknik kompresi data.

Kompresi data adalah suatu teknik mengubah data menjadi bentuk data lain dimana data tersebut diubah menjadi simbol yang lebih sederhana. Kompresi data mempunyai tujuan memperkecil ukuran data tanpa merubah isi dari data tersebut sehingga selain dapat menghemat media penyimpanan juga memudahkan transfer data.

Terdapat banyak metode kompresi*lossless*salah satu Metode *Lempel Ziv Storer Szymanski* (LZSS). LZSS menghasilkan kode-kode untuk karakter dengan cara membentuk "*dictionary*". Sedangkan perbedaannya dengan LZ77 dimana "*dictionary*" harus dibentuk setiap data didekompresi ulang sedangkan pada LZSS hanya suatu urutan dengan panjang minimum tertentu yang dikodekan. Metode LZSS menggunakan trik khusus yaitu menggunakan *bit flag* yang hanya satu bit saja yang memberitahukan data apa berikutnya dan untuk membedakan antara isi tidak terkompresi dengan menggunakan pointer. Ini tentunya berpengaruh pada waktu proses.

Oleh karena dirancang sebuah aplikasi yang dapat mengkompresi dan dekompres *file* dokumen menggunakan metode *Lempel Ziv Storer Szymanski* (LZSS)

2. METODOLOGI PENELITIAN

2.1. KompresiData

Kompresi berarti memampatkan mengecilkan ukuran.Kompresi data adalah proses mengkodekan informasi menggunakan bit atau *information-bearing* unit yang lain yang lebih rendah daripada representasi data yang tidak terkodekan dengan suatu sistem *encoding* tertentu. Contoh kompresi sederhana yang biasa dilakukan misalnya adalah menyingkat kata-kata yang sering digunakan tapi sudah memiliki konvensi umum.Misalnya : kata "yang" dikompres menjadi kata "yg". Kompresi data teks dan citra digital adalah proses untuk meminimalisasi jumlah bit yang merepresentasikan suatu data baik berupa teks maupun citra/gambar sehingga ukuran data menjadi lebih kecil. Pengiriman data hasil kompresi dapat dilakukan jika pihak pengirim atau pihak penerima memiliki aturan yang sama dalam hal kompresi data [1].

2.2 Lempel Ziv Storer Szymanski (LZSS)

Algoritma *Lempel Ziv Storer Symanski* (LZSS) adalah algoritma kompresi data *lossless* yang dimodifikasi dari LZ77, dinamai penciptanya James Storer dan Thomas Szymanski (yang dibangun di atas karya Abraham Lempel dan Jacob Ziv). Algoritma *Lempel Ziv Storer Symanski* (LZSS) salah satu kompresi urutan simbol data (misalnya, byte data) dengan mengidentifikasi urutan simbol yang berulang dalam masukan, dan menggantikan urutan-urutan simbol yang lebih kecil. Implementasi *Lempel Ziv Storer Symanski* (LZSS) dapat menyesuaikan jumlah bit yang dialokasikan dengan mengganti panjang ukuran byte (antara parameter lain) untuk mendapatkan kinerja kompresi yang cukup baik [2].

2.2.1 Algoritma *Lempel Ziv Storer Symanski* (LZSS)

Algoritma untuk *Lempel Ziv Storer Symanski* (LZSS) [2] adalah sebagai berikut :
1. Tempatkan posisi koding pada permulaan masukan *stream*.
2. Cari *longest match* pada *window* untuk *lookahead buffer*;
   a. P = pointer untuk mencocokan;
   b. L = panjang masukan yang cocok;
3. Apakah L > = MIN_LENGTH?
   a. Jika YA :keluarannya P dan bergerak ke posisi L karakter;
   b. Jika TIDAK : keluarannya karakter pertama pada *lookhead buffer* dan bergerak satu posisi karakter ke depan;
4. Jika karakter masukan *stream* banyak, ulangi langkah 2.

Window dan *Match Lengt*

Ada beberapa aturan agar sistem dapat bekerja yaitu:
a. Untuk *flag* mengidentifikasi sebuah literal atau cocok
b. Seberapa jauh data sebelumnya dapat dicocokan.
c. Jumlah bytes maksimum yangdapat dicocokan. Untuk aturan yang pertama hanya membutuhkan dua *flag* yaitu literal atau cocok, jadi hanya menggunakan bit tunggal. Jika menggunakan bit tunggal maka seluruh bytes tidak selalu ditulis tapi hanya sebagian saja. Sebelum proses pemampatan dilakukan, dibutuhkan suatu fungsi kode untuk membaca dan menulis jumlah variable bit dari atau ke data *stream*. Sebagai contoh : batas *offset* menggunakan 4 bits, jadi *range*-nya antara 0-31. Sedangkan untuk len menggunakan 3 bits jadi *range*- nya antara 0-7. Keluarannya ditulis menjadi (4,3) dalam byte menjadi "1 0100011". 1 bit pertama menunjukan *flag*, diikuti 4bits berikutnya yang menunjukan *offset*, kemudian 3bits berikutnya menunjukan *length* [2].

Canterbury Corpus

Corpus adalah kumpulan data standar yang digunakan dalam mengevaluasi terhadap suatu kompresi*loseless* secara empiris.*Canterbury Corpus* merupakan kumpulan corpus dari data yang sering digunakan di masa sekarang.*Canterbury Corpus* merupakan hasil riset seorang doktor yang bernama Jeff Ghilchrist dari Universitas Carleton, Ottawa, Kanada [3]. Berikut file teks yang diambil dari *Canterbury Corpus* yang digunakan dalam penelitian ini ditampilkan pada Tabel 1.

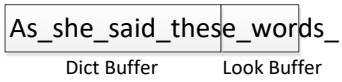
Tabel 1.File Teks Canterbury Corpus

Kode	File	Ukuran (byte)
C1	Alice29.txt	152089
C2	Asyoulik.txt	125179
C3	Lcet10.txt	426754
C4	1musk10.txt	1344739
C5	World95.txt	2993596

Studi Kasus

Pada studi kasus digunakan sebuah kalimat yang dikompresi algoritma LZSS yaitu "As she said these words"

Langkah ke 1 :
Menentukan besar *sliding windows*, diasumsikan *sliding windows* sebesar 21 *byte* dengan *16 byte dictionary buffer* dan 5 byte *dictionary buffer*. *Sliding windows* ditampilkan seperti

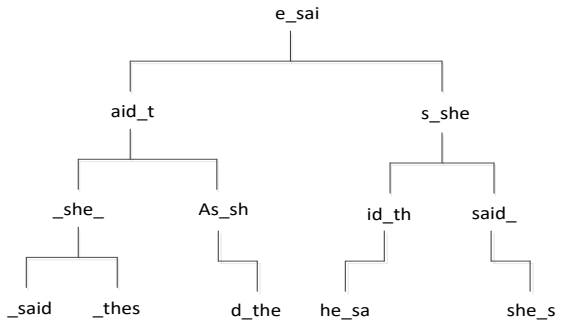


Langkah ke 2 :
Hitung banyak node yang dimasukkan pada *binary search tree*.Node dihitung dengan rumus.
Tnode = Panjang *Dictionary Buffer* – Panjang *Look Ahead Buffer* + 1
Tnode = 16 – 5 + 1 = 12 Node

Panjang karakter node sama dengan panjang *look ahead buffer* yaitu 5 *byte* (karakter). Setelah banyak node dan panjang node ditemukan maka dilakukan pemberian *offset* pada setiap node sebagai *pointer*. *Offset* dihitung dari sisi kanan *dictionary buffer*. Hasil pemberian *offset* pada *dictionary buffer* seperti

_thes	00
d_the	01
id_th	02
aid_t	03
said_	04
_said	05
e_sai	06
he_sa	07
she_s	08
she	09
s_she	10
As_sh	11

Langkah ke 3 :
Masukkan node yang ditemukan ke dalam *binary search tree* berdasarkan prinsip *lexicongraphic order*.Tinggi pohon didapat dari rumus $\log_2$Tnode dan hasil tinggi pohon 4 tingkat.*Binary search tree* yang terbentuk seperti pada pada`gambar 1`.

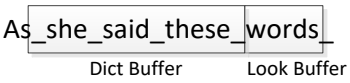


Gambar 1. Binary Search Tree

Langkah ke 4 :
Membandingkan karakter awal pada *look ahead buffer* dengan node yang ada pada *dictionary buffer*. Karakter awal pada awal *look aheadbuffer*adalah e dan terjadi cocok dengan node yang ada pada *dictionary buffer*. Pada *offset* 11 dan panjang cocok sebesar 2 karakter. Maka token dihasilkan ada (0,11,2) dimana 0 adalah *bit flag match*, 11 adalah *offset*, dan 2 adalah panjang karakter cocok. Maka kalimat yang terkompresi menjadi seperti.

As she said thes(0,11,2)words

Langkah ke 4:
Karena terjadi cocok maka *sliding windows* bergeser sepanjang cocok yaitu 2 karakter.*Sliding windows* pun menjadi seperti



Setelah *sliding windows* bergeser maka node yang ada *binary search tree* harus diperbaharui seperti dengan menghapus node As_sh dan s_she dan memasukkan node baru these dan hese_. Lalu lakukan langkah 2 kembali.
Proses kompresi terus berlangsung hingga pembacaan teks selesai dan *file* teks akan terkompresi.

Penghitungan Durasi Kompresi

Pada penghitungan durasi ini fungsi java yang digunakan adalah`System.currentTimeMillis()` yangberguna untuk mengambil waktu saat ini dalam satuan milidetik, dihitung sejak 1 Januri 1970 waktu GMT. Satu milidetik sama dengan 1 per 1000 detik. Keluarannya bertipe long.

Untuk menghitung waktu yang diperlukan untuk menjalankan suatu perintah, jalankan fungsi `System.currentTimeMillis()` sebelum dan sesudah suatu instruksi dijalankan. Perbedaannya adalah waktu yang diperlukan untuk menjalankan suatu instruksi. Berikut cara penulisan dalam program ditampilkan pada Gambar 2.

```
1  long startTime = System.currentTimeMillis();
2  --- run main class kompresi/dekompresi ---;
3  long endTime = System.currentTimeMillis();
4  totalTime = endTime - startTime;
```

Gambar 2, Penghitungan Durasi Kompresi

2.3 Analisis Pengumpulan Data

Pada pengumpulan data terdapat *file* teks yang digunakan dalam melakukan kompresi data dokumen. Kumpulan teks yang digunakan dalam mengecek kemampuan dari aplikasi yang dibangun diambil dari *Canterbury Corpus*seperti yang dijelaskan pada sub bab 2.5.

Selain *file* teks dari Canterbury Corpus, pengujian juga menggunakan *file* teks lain.*File* teks tersebut memiliki berbagai karakteristik. Jenis *file* teks yang digunakan adalah buku teknik dan buku non teknik.

File teks tersebut ditampilkan pada Tabel 2.

Tabel 2.File TeksLain

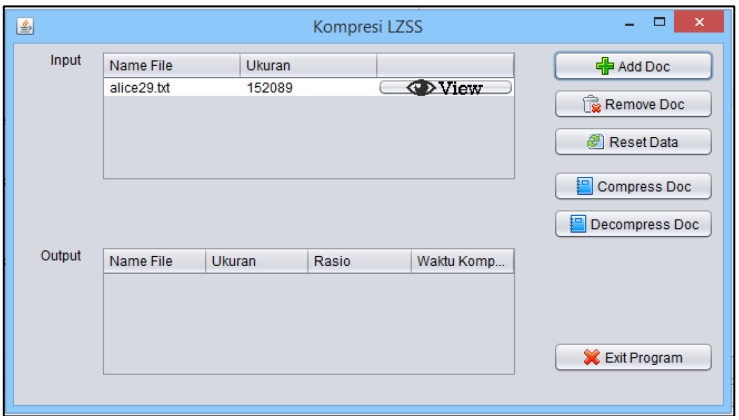
Kode	File	Ukuran (byte)
P1	11.IGIBook1.pdf	5562768
P2	b11.pdf	3832502
P3	First Course On Fuzzy Theory and Application.pdf	2403653
P4	JAVA3elatest.pdf	3596723
P5	Wiley.Interscience.Fuzzy.Expert.Systems.and.Fuzzy.Reasoning.Dec.2004.eBook-DDU.pdf	2926192
P6	Breadwinner.doc	284672
P7	frogs.doc	333824
P8	James.doc	59392
P9	Little_Women_NT.pdf	2173034
P10	the_case_book.doc	718848

2.4 Analisis Sistem

Setelah analisis pengumpulan dilakukan, tahap selanjutnya adalah merancang sebuah desain yang menjelaskan cara kerja sistem aplikasi kompresi *file* dengan metode LZSS ini.

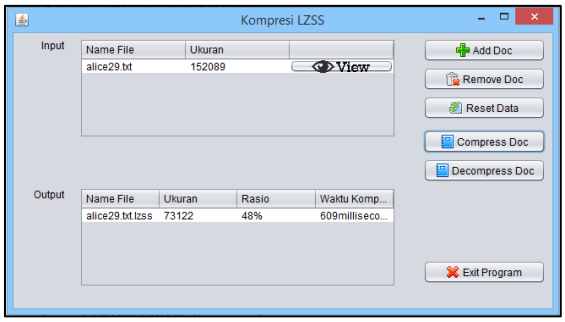
Tahap sistem kerja aplikasi kompresi ini adalah sebagai berikut :

a) Pengguna membuka aplikasi.
b) *File* teks dipilih oleh pengguna lalu ditampilkan pada aplikasi.



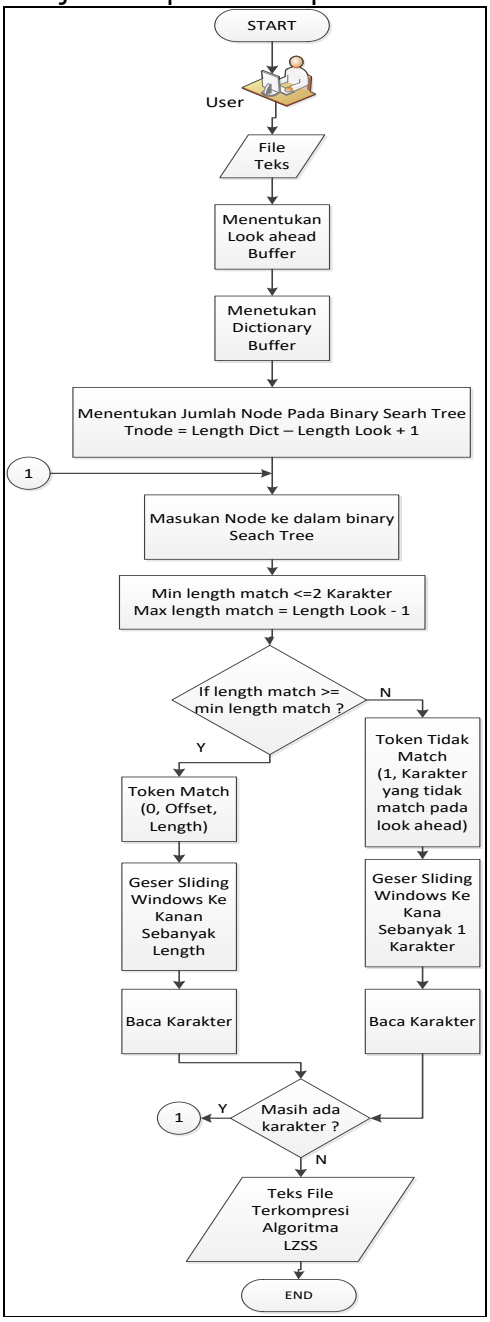
Gambar 3. Tampilan File Terpilih

c) Pengguna melakukan kompresi pada *file* teks yang sudah dipilih. Proses kompresi dilakukan seperti pada **studi kasus**.
d) Hasil keluaran dari proses kompresi adalah *file* teks terkompresi dengan format .lzss, rasio kompresi yang ditunjukkan dalam bentuk persen, dan durasi kompresi yang ditunjukkan dalam satuan milisekon.



Gambar 4 Tampilan Output

Analisis sistem sudah dilakukan, hasil analisis tersebut dipetakan dalam bentuk flowchart. Berikut gambaran sistem kerja dari aplikasi kompresi*file* ini ditampilkan pada`gambar 5`.

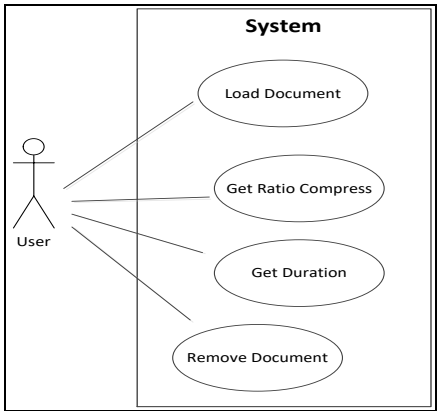


Gambar 5. Flowchart Sistem Kerja Aplikasi

3. ANALISIS DAN PEMBAHASAN

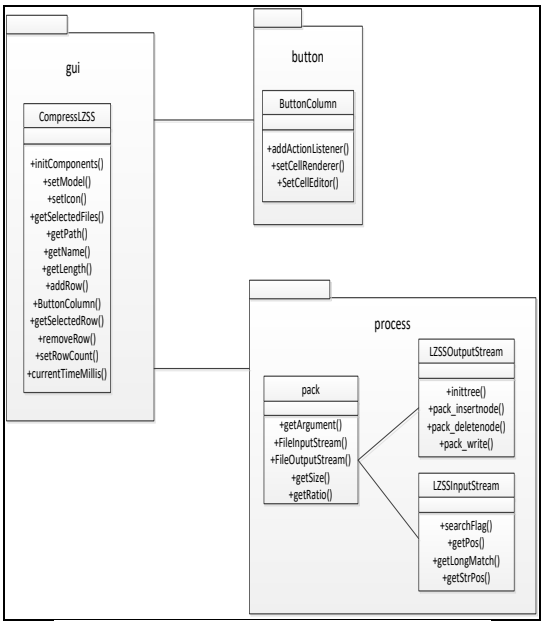
3.1 Perancangan Sistem

Perancangan sistem ini menggunakan UML.Berdasarakan analisis yang telah dilakukan. Berikut merupakan proses pengembangan yang dimodelkan dalam bentuk *use case diagram*. *Use case diagram* dari proses utama yang terjadi pada pengembangan aplikasi ditampilkan pada Gambar 6.



Gambar 6. Use Case Diagram Sistem

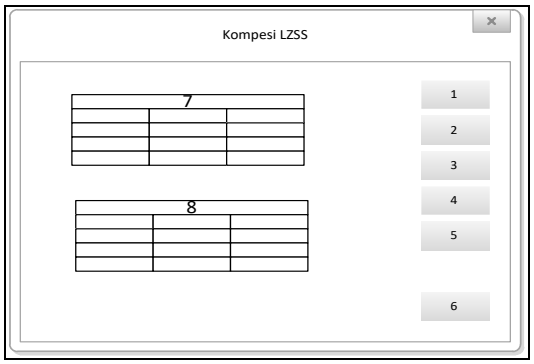
Setelah *use case diagram* dirancang. Berikut *class diagram* dari sistem yang telah dibangun, ditampilkan pada Gambar 7.



Gambar 7. Class Diagram Sistem

3.2 Perancangan *Layout*

Berikut perancangan *layout* untuk antar muka aplikasi yang telah dibangun ditampilkan pada Gambar 8.



Gambar 8 Tampilan Antar Muka Aplikasi

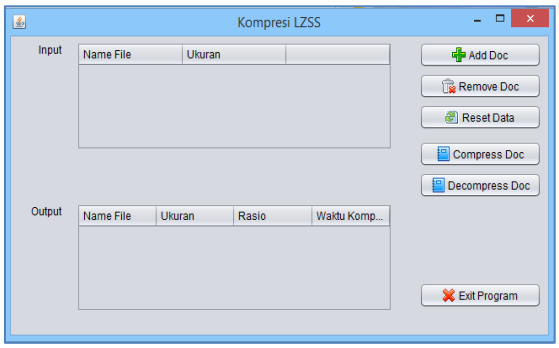
Dimana :

1. Tombol *Add Doc*
2. Tombol*Remove Doc*
3. Tombol*Reset Data*
4. Tombol *Compress Doc*
5. Tombol *Decompress Doc*
6. TombolKeluar Program
7. TabelDataMasukan
8. TabelDataKeluaran

3.3 Pengujian Fungsional

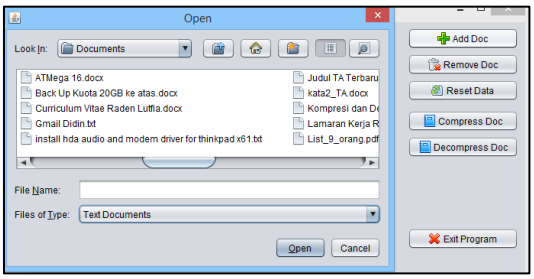
Setelah melakukan perancangan, tahap selanjutnya adalah implementasi sistem yang sudah dibangun.Berikut merupakan tampilan hasil dan pengujian fungsional aplikasi.

1. Tampilan *Form* Utama



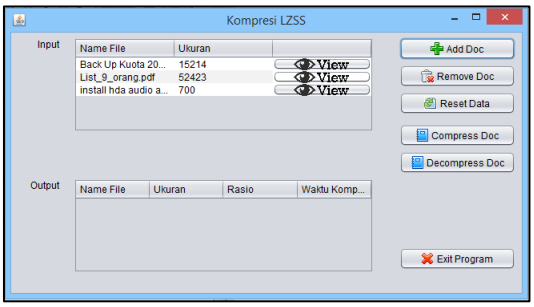
Gambar 9 Tampilan Form Utama

2. Tampilan Pengujian Tambah *File*
   Selanjutnya dilakukan penambahan *file* yang dikompresi dengan menekan tombol *add doc* seperti pada Gambar 10.



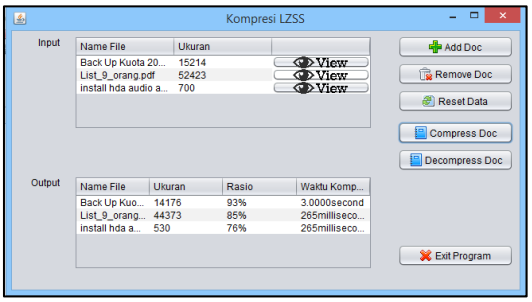
Gambar 10 Tampilan Pilih File

Setelah memilih pengguna menekan tombol*open*.Hasilnya seperti pada Gambar 11.



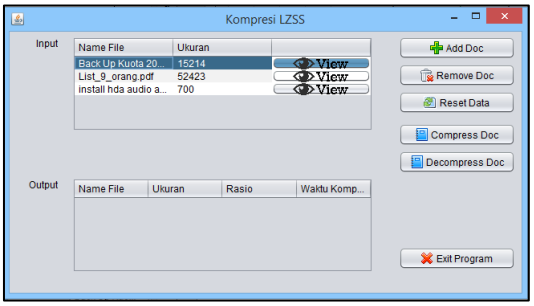
Gambar 11. Data Terpilih Ditampilkan

3. Tampilan Pengujian Nilai Rasio dan Durasi Kompresi
   Setelah *file* berhasil ditambahkan, untuk mendapatkan rasio dan durasi kompresi dengan menekan tombol *compress doc*. Hasilnya seperti pada `gambar 12`.



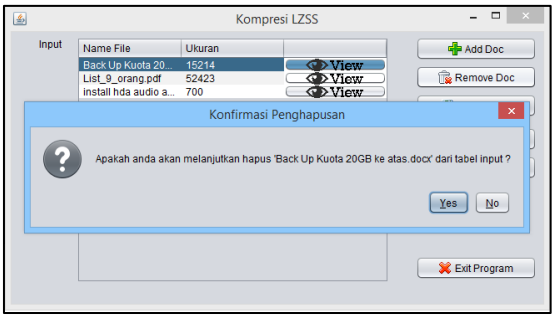
Gambar 12. Hasil Pengujian Nilai Rasio dan Durasi Kompresi

4. Tampilan Pengujian Hapus *File*
   Jika pengguna ingin menghapus *file* dari tabelmasukan, pertama pengguna menekan salah satu baris pada tabelmasukan lihat Gambar 13.



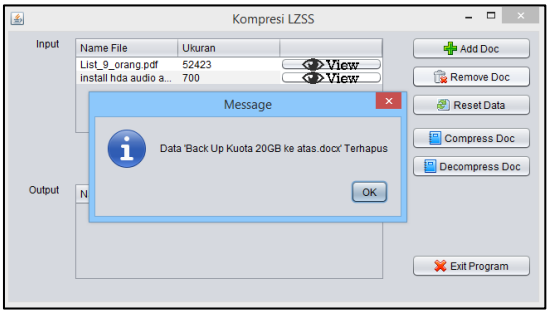
Gambar 13. Klik Data Yang Dihapus

Lalu *pengguna* menekan tombol*removedoc* dan peringatan penghapusan muncul lalu pilih *yes*lihat pada Gambar14.



Gambar 14. Peringatan Penghapusan

Maka *file* terpilih terhapus dan windows pemberitahuan bahwa *file*teks terpilih terhapus ditampilkan seperti pada Gambar 15.



Gambar 15. Data Terpilih Terhapus

3.4 Pengujian Sistem

Setelah melakukan pengujian secara fungsional, selanjutnya pengujian sistem dimana untuk mengetahui kemampuan aplikasi kompresi yang telah dibangun.Pengujian ini bertujuan untuk melihat hasil rasio kompresi dan durasi kompresi.Data yang dipakai dalam pengujian ada di Tabel 1 dan Tabel 2.Berikut adalah hasil dari pengujian dari masing-masing data.

3.4.1 Pengujian Rasio dan Durasi Kompresi pada *File* Teks Dari *Canterbury Corpus*

Hasil pengujian rasio dan durasikompresi ditunjukkan pada Tabel 3.

Tabel 3. Hasil Pengujian Rasio dan Durasi Kompresi Canterbury Corpus

Kode	Ukuran Sebelum Kompresi(byte)	Ukuran Sesudah Kompresi (byte)	Rasio (%)	Durasi (ms)
C1	152089	73122	48	875
C2	125179	65555	52	344
C3	426754	199727	47	1000
C4	1344739	665536	50	4000
C5	2993596	1580209	53	6000

Dari hasil pengujian pada Tabel 3, dapat kita lihat bahwa rasio kompresi LZSS terhadap *file* teks *Canterbury Corbus* tidak terlalu bervariasi. Hal tersebut dikarenakan kelima*file* teks tersebut memiliki karakteristik yang hampir sama yaitu sebuah karya tulis non ilmiah dan ukuran *file* tidak jauh berbeda.

Dari hasil pengujian pada Tabel 3, diketahui bahwa semakin besar *file*masukan semakin lama waktu pengompresian *file* teks tersebut.

3.4.2 Pengujian Rasio dan Durasi Kompresi pada *File* TeksLain

Hasil pengujian rasio dan durasi kompresi pada*file* teks lain ditunjukkan pada Tabel 4.

Tabel 4. Hasil Pengujian Rasio dan DurasiKompresiFile TeksLain

Kode	Ukuran Sebelum Kompresi (byte)	Ukuran Setelah Kompresi (byte)	Rasio (%)	Durasi (ms)
P1	5562768	4134071	74	6000
P2	3832502	2272557	59	6000
P3	2403653	1884993	78	3000
P4	3596723	2855648	79	5000
P5	2926192	1927907	66	3000
P6	284672	135258	48	937
P7	333824	107555	32	1000
P8	59392	17964	30	141
P9	2173034	1495575	69	1000
P10	718848	305332	42	1000

Dari hasil pengujian pada`tabel 4`, dapat kita lihat bahwa rasio kompresi LZSS terhadap *file* teks lainmemiliki nilai bervariasi. Hal tersebut dikarenakan kesepuluh*file* teks tersebut memiliki karakteristik yang berbeda dan ukuran *file*teks variatif.

Sama halnya dengan hasil pengujian pada`tabel 3`, diketahui bahwa semakin besar *file*masukansemakin lama waktu pengompresian*file* teks tersebut.

3.4.3 Pengujian Rasio dan Durasi Kompresi pada *File* Teks Berformat .txt

Pada pengujian ini diambil 3 *file* teks dari data yang sebelumnya diuji yaitu *alice29.txt*, *james.doc*, dan *frogs.doc*. *File* teks yang belum berformat .txt dikonversi terlebih dahulu. Berikut hasil pengujian rasio kompresi*file* teks berformat .txt ditunjukkan pada Tabel 5.

Tabel 5 Hasil Pengujian Rasio dan DurasiKompresiFile Teks Berformat .txt

File	Ukuran Sebelum Kompresi(byte)	Ukuran Setelah Kompresi(byte)	Rasio (%)	Durasi (ms)
alice29.txt	152089	73122	48	453
frogs.txt	183586	66269	36	515
james.txt	15433	8296	54	48

Dari Tabel 5 didapatkan bahwa hasil rasio kompresi LZSS terhadap *file* teks berformat .txt baik jika *file*masukannya berukuran besar seperti ditunjukkan *frogs.txt* menghasilkan rasio 36% dimana *file* asli teks tersebut sebesar 183586 byte.

Sama halnya dengan hasil pengujian pada Tabel 3 dan Tabel 4, diketahui bahwa semakin besar *file*masukan semakin lama waktu pengompresian*file* teks tersebut.

3.4.4 Pengujian Rasio dan Durasi Kompresi pada *File* Teks Berformat .doc

Berikut hasil pengujian rasio kompresi*file* teks berformat .doc ditunjukkan pada Tabel 6.

Tabel 6 Hasil Pengujian Rasio dan Durasi KompresiFile Teks Berformat .doc

File	Ukuran Sebelum Kompresi (byte)	Ukuran Sesudah Kompresi (byte)	Rasio (%)	Durasi (ms)
alice29.doc	242176	103305	43	610
frogs.doc	333824	107555	32	812
james.doc	59392	17964	30	156

Dari Tabel 6 didapatkan bahwa hasil rasio kompresi LZSS terhadap *file* teks berformat .doc mengalami peningkatan dibanding dengan kompresi*file* teks berformat .txt.

Sama halnya dengan hasil pengujian pada Tabel 3, Tabel 4dan Tabel 5, diketahui bahwa semakin besar *file* masukan semakin lama waktu pengompresian*file* teks tersebut.

3.4.5 Pengujian Rasio dan Durasi Kompresi pada *File* Teks Berformat .pdf

Berikut hasil pengujian rasio kompresi*file* teks berformat .pdf ditunjukkan pada Tabel 7.

Tabel 7 Hasil Pengujian Rasio dan Durasi KompresiFile Teks Berformat .pdf

File	Ukuran Sebelum Kompresi (byte)	Ukuran Setelah Kompresi (byte)	Rasio (%)	Durasi (ms)
alice29.pdf	546470	469032	86	1000
frogs.pdf	489807	449186	92	859
james.pdf	140115	145613	104	187

Sama halnya dengan hasil pengujian pada tabel 5, hasil rasio kompresi LZSS terhadap *file* teks berformat .pdf baik jika *file* masukannya berukuran besar seperti ditunjukkan *alice29.pdf* menghasilkan rasio 86% dimana *file* asli teks tersebut sebesar 546470 byte.

Sama halnya dengan hasil pengujian pada tabel 3,4,5dan6, diketahui bahwa semakin besar *file*masukan semakin lama waktu pengompresian*file* teks tersebut.

3.4.6 Pengujian Dekompresi pada *File* Teks Berformat .txt Terkompresi Algoritma LZSS

Pada pengujian dilakukan dekompresi dari *file* teks terkompresi algoritma LZSS ke *file* teks asal. Pengujian ini hanya dilakukan pada 3 *file* teks berformat yang sebelumnya diuji berdasarkan format. Berikut hasil dari dekompresi*file* teks ini ditampilkan pada tabel 8.

Tabel 8 Hasil Pengujian DekompresiFile Teks Berformat .txt terkompresi Algoritma LZSS

File	Ukuran Kompresi (byte)	Hasil Dekompresi (byte)	Durasi (ms)
alice29.txt	73122	152089	297
frogs.txt	66269	183586	265
james.txt	8296	15433	47

Dari tabel 8 diketahui bahwa *file* yang telah terkompresi dapat didekompresi ke *file* teks asal dengan baik dan ukuran*file* juga sama dengan *file* teks asal sebelum terkompresi.

3.4.7 Rata-rata rasio kompresi setiap pengujian

Berikut rata-rata rasio kompresi dari setiap pengujian ditampilkan pada tabel 9.

Tabel 9 Rata-Rata Rasio Kompresi

Pengujian Pada	Rata-Rata Rasio Kompresi
Canterbury	50%
File Teks Lain	57,7%
Format .txt	46%
Format .doc	35%
Format .pdf	94%

3.4.8 Rata-rata Durasi kompresi setiap pengujian

Berikut rata-rata durasikompresi dari setiap pengujian ditampilkan pada tabel 10.

Tabel 10 Rata-Rata Rasio Kompresi

Pengujian Pada	Rata-Rata Durasi Kompresi
Canterbury	2443,8 ms
File Teks Lain	2707,8 ms
Format .txt	427,34 ms
Format .doc	516 ms
Format .pdf	682 ms

4. KESIMPULAN

Berdasarkan hasil penelitian yang diperoleh dari pengujian aplikasi didapatkan kesimpulan sebagai berikut :

1. Aplikasi kompresi file menggunakan metode *Lempel Ziv Storer Szymanski* mampu mengompresi maksimaldengan rasio terbaik dari hasil pengujian didapat dari *file*masukan berformat *.doc yaitu *james.doc* dengan rasio kompresi 30%.
2. Kecepatan kompresi dipengaruhi oleh ukuran *file*masukannya. Jika *file* masukannya berukuran besar maka pengompresian memakan waktu yang lama merujuk pada tabel 3, 4,5, 6dan 7.
3. Dari hasil pengujian berbagai format, *file* teks berformat .doc menghasilkan rasio yang baik dengan rata-rata rasio kompresi 35% dan *file* teks berformat .txt menghasilkan durasi kompresi yang cepat dengan rata-rata durasi kompresisi 427,34 ms.
4. Dari hasil pengujian dekompresi diketahui bahwa *file* teks terkompresi dapat dikompresi ke *file* teks asli dengan baik. Ukuran*file* teks hasil dekompresisama dengan *file* teks asal sebelum terkompresi artinya kompresi LZZS yang digunakan benar *lossless*.

DAFTAR RUJUKAN

[1] D. Solomon, " Data Compression: the complete reference," Springer-Verlag: New York, 2004.

[2] J.A. Storer dan T.G. Szymaski, " Data compression via textual substitution, " Jurnal ACM on Information Theory, 1982.

[3] M. Powell, " Evaluating lossless compression method, " Technical Report, Department of Computer Science, University of Canterbury, NZ, 2004

[4] J. Gilchrist, "Archive Comparison Test, " Technical Report, 2002. http://compression.ca/