Kompresi dan Dekompresi File Dengan Algoritma LZSS

Kian Wie[1)] Adang Risuta[2)]
STMIK IBBI Medan
Kampus Topaz, Jalan Damar No. 9G Medan [1), 2)]
Telepon (061) 415 - 3375
Email: kianwie@yahoo.com.sg[1]

Abstrak

File merupakan data digital yang berisi informasi-informasi. Ukuran *file* yang terlalu besar akan menjadi masalah bila *file* tersebut akan ditransfer atau dipertukarkan. Untuk itu dibutuhkan cara tertentu untuk memperkecil ukuran *file*.

Salah satu caranya untuk masalah di atas adalah *file* tersebut dipadatkan melalui proses kompresi sehingga ukurannya menjadi lebih kecil dari ukuran semula dan mempersingkat waktu ketika ditransmisi. Salah satu metode yang dapat dipakai untuk mengkompresi ukuran *file* adalah dengan metode LZSS (*Lempel Ziv Storer-Szymanski*) yang bekerja berdasarkan penempatan karakter pada *dictionary.*

Hasil dari tulisan ini adalah suatu perangkat lunak yang dapat melakukan kompresi dan dekompresi balik berdasarkan pada algoritma LZSS pada semua jenis *file* yang belum dikompresi. Program juga dapat menampilkan besarnya rasio kompresi, rasio dekompresi, dan lama proses yang telah dikerjakan pada *file* serta mempunyai fasilitas proteksi dengan menginput *password* untuk mengamankan *file* hasil kompresi. Hasil pengujian dengan metode kompresi LZSS juga dapat menampilkan rentang rasio reduksi.

Kata Kunci*: File, Kompresi, LZSS*

Abstract

A digital data file that contains the information. File size that is too large will be a problem if the files are to be transferred or exchanged. That requires a certain way to reduce the file size. One way for the above problem is the file is compressed by the compression process so that its size becomes smaller than the original size and shorten the time when it is transmitted.

One method that can be used to compress the file size is the method LZSS (Lempel Ziv Storer-Szymanski) that works by placing the characters in the dictionary.

The results of this paper is a software that can compress and decompress back based on LZSS algorithm on all types of files are not compressed. The program can also display the amount of compression ratio, compression ratio, and a long process that has been done on the file and having to enter the password protection facility for secure file compression results. Results of testing with LZSS compression method can also display range reduction ratio.

Keywords*: File, Compression, LZSS*

1. Pendahuluan

Kompresi data atau dikenal juga sebagai pemampatan 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 sehingga selain dapat menghemat media penyimpanan dan memudahkan transfer data. Selain itu pada suatu *file* banyak terdapat redudansi data serta untuk mempersingkat waktu transmisi sewaktu *file* tersebut dikirim atau di-*download* melalui jaringan Internet.

Terdapat banyak metode kompresi *lossless* diantaranya Huffman, LZ77, LZ78, LZW, LZSS, LBE, dan lain-lain. Metode-metode tersebut mempunyai rasio kompresi dan kecepatan proses yang bervariasi. Metode LZSS (Lempel Ziv Storer-Szymanski) dapat bekerja cepat. LZSS menghasilkan kode-kode untuk karakter tanpa perlu membentuk pohon biner seperti pada Huffman melainkan dengan cara membentuk ”*dictionary*”. Sedangkan perbedaannya dengan LZ77 dimana ”*dictionary*” harus dibentuk setiap data akan didekompresi ulang sedangkan LZSS tidak diperlukan cara demikian. Pada LZSS hanya suatu urutan dengan panjang minimum tertentu yang akan dikodekan, jika penggantian ini menawarkan efek kompresi yang lebih besar. 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. Selain itu pemampatan data merupakan salah satu isu penting untuk mempermudah transmisi data maka terdapat banyak program-program yang berfungsi untuk memampatkan data.

Disebabkan luasnya permasalahan yang ada, maka peneliti memberikan batasan masalah antara lain:

Kinerja dari algoritma kompresi ditunjukkan dalam bentuk rasio kompresi dan kecepatan proses (waktu eksekusi) baik untuk kompresi dan dekompresi

2. Program dapat memproteksi *file* hasil kompresi dengan menyisipkan *password* maksimum 7 (tujuh) karakter.
3. Program tidak dapat melakukan pembagian *file* (*spanning*) hasil kompresi menjadi beberapa *file-file* bagian
4. Jenis *file* yang dapat dikompresi adalah *file* *.DOC, *file* *.TXT, *file* *.WAV, *file* *.BMP, *.HTML, *.EXE, kecuali jenis *file* yang telah terkompresi.
5. Tidak dapat melakukan kompresi pada *folder*.
6. Perancangan dan pembuatan perangkat lunak ini menggunakan bahasa *Microsoft Visual Basic 6.0*.

2. Metode Penelitian

Kompresi data dilakukan untuk mereduksi ukuran data atau *file*. Dengan melakukan kompresi atau pemadatan data maka ukuran *file* atau data akan lebih kecil sehingga dapat mengurangi waktu transmisi sewaktu data dikirim dan tidak banyak banyak menghabiskan ruang media penyimpan. (Salomon, 2004: 25)

Dalam makalahnya di tahun 1948, "**A Mathematical Theory of Communication**", Claude E. Shannon merumuskan teori kompresi data. Shannon membuktikan adanya batas dasar (*fundamental limit*) pada kompresi data jenis *lossless*. Batas ini, disebut dengan *entropy rate* dan dinyatakan dengan simbol *H*. Nilai eksak dari *H* bergantung pada informasi data sumber, lebih terperinci lagi, tergantung pada statistikal alami dari data sumber. Adalah mungkin untuk mengkompresi data sumber dalam suatu bentuk *lossless*, dengan laju kompresi (*compression rate*) mendekati *H*. Perhitungan secara matematis memungkinkan ini dilakukan lebih baik dari nilai *H*.

Shannon juga mengembangkan teori mengenai kompresi data *lossy*. Ini lebih dikenal sebagai *rate-distortion theory*. Pada kompresi data *lossy*, proses dekompresi data tidak menghasilkan data yang sama persis dengan data aslinya. Selain itu, jumlah *distortion* atau nilai *D* dapat ditoleransi. Shannon menunjukkan bahwa, untuk data sumber (dengan semua properti statistikal yang diketahui) dengan memberikan pengukuran distorsi, terdapat sebuah fungsi R(*D*) yang disebut dengan *rate-distortion function*. Pada teori ini dikemukakan jika D bersifat toleransi terhadap jumlah distorsi, maka R(*D*) adalah kemungkinan terbaik dari laju kompresi.

Ketika kompresi *lossless* (berarti tidak terdapat distorsi atau $D = 0$), kemungkinan laju kompresi terbaik adalah R(0) = H (untuk sumber sumber alphabet yang terbatas). Dengan kata lain, laju kompresi terbaik yang mungkin adalah *entropy rate*. Dalam pengertian ini, teori *rate-distortion* adalah suatu penyamarataan dari teori kompresi data *lossless*, dimana dimulai dari tidak ada distorsi ($D = 0$) hingga terdapat beberapa distorsi ($D > 0$).

Teori kompresi data *lossless* dan teori *rate-distortion* dikenal secara kolektif sebagai teori pengkodean sumber (*source coding theory*). Teori pengkodean sumber menyatakan batas fundamental pada unjuk kerja dari seluruh algoritma kompresi data. Teori tersebut sendiri tidak dinyatakan secara tepat bagaimana merancang dan mengimplementasikan algoritma tersebut. Bagaimana pun juga algoritma tersebut menyediakan beberapa petunjuk dan panduan untuk memperoleh unjuk kerja yang optimal. (Salomon, 2004: 30).

Dalam kompresi data *lossless*, data yang dikompresi dan didekompresi mempunyai replikasi yang sama dengan data asli. Sedangkan pada kompresi data *lossy*, data yang didekompresi dapat berbeda dari data asli.

2.1. Model

Algoritma LZSS (*Lempel-Ziv-Storer-Szymanski)* merupakan perubahan dari algoritma LZ77 yang dilakukan oleh James Storer dan Thomas Szymanski pada tahun 1982. LZSS juga merupakan salah satu bagian dari pemampatan model kamus (*dictionary compression*), yang menggunakan simbol baru untuk menggantikan *string* data. *Output* LZSS terdiri dari sebuah *literal byte* atau pasangan (*offset*, *len*). Literal *byte* merupakan input *byte* sederhana yang dikopikan secara langsung ke *output*. Pasangan (*offset*,*len*) menjelaskan proses kembali (*back*) ke jarak *offset* suatu *input* data yang nantinya akan ditemukan sebuah *match length* (*len*) dari data tersebut. (Nelson, 2002: 35).

LZSS merupakan teknik pengkodean *dictionary*. Tidak seperti Huffman *coding*, dimana bertujuan untuk mengurangi jumlah rata-rata bit yang diperlukan untuk merepresentasi suatu simbol, LZSS bertujuan untuk mengganti suatu *string* simbol dengan suatu referensi dari suatu lokasi *dictionary* yang mempunyai *string* yang sama. Ini ditujukan pada referensi *dictionary* yang harus lebih pendek dari string yang digantikannya. Perbedaan utama antara LZ77 yang merupakan metode yang menurunkan LZSS adalah LZ77 selalu menghasilkan *output* pasangan *offset*/*length*, bahkan jika terjadi *match* meskipun hanya satu *byte* (dalam kasus ini menggunakan lebih dari 8 bit untuk merepresentasikan satu *byte*) jadi LZSS menggunakan trik yang lain untuk meningkatkannya. LZSS menggunakan *bit flags* yang hanya satu bit untuk memberitahukan data apa berikutnya yaitu: suatu literal (satu *byte*) ataupun satu pasang dari *offset*/*length*. (Nelson, 2002: 35).

2.2. Analisis

1. Algoritma *Kompresi*

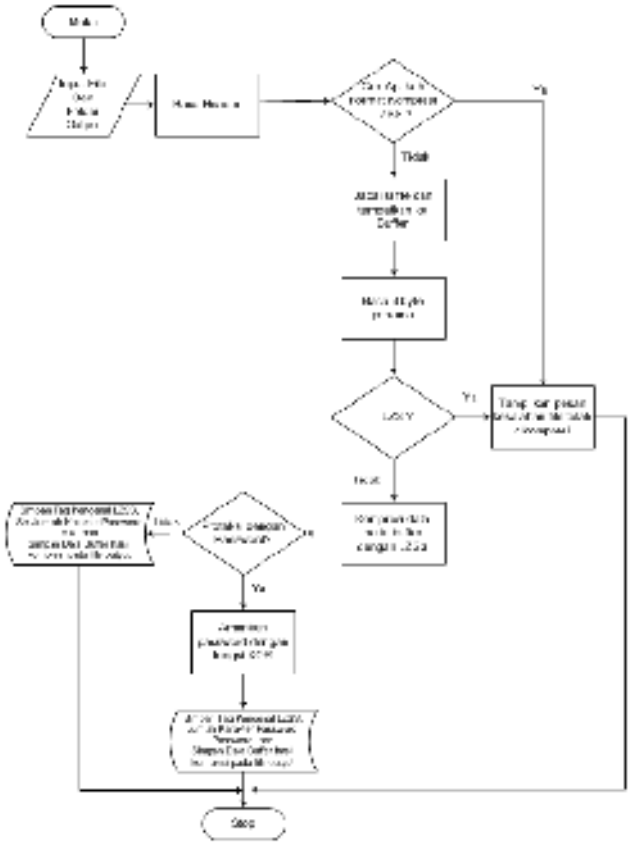
Kompresi LZSS mencari suatu solusi untuk beberapa *bottleneck* dan masalah unjuk kerja yang terdapat pada algoritma LZ77. Algoritma LZSS membuat dua perubahan besar dari cara kerjanya. Perubahan pertama adalah cara *text window* diatur. Pada LZ77, frase dalam *text window* disimpan sebagai suatu *block* tunggal bersambung dari teks, dengan tidak ada organisasi lainnya pada bagian atasnya. LZSS masih menyimpan teks dalam *window* bersambung, tetapi algoritma ini juga membentuk struktur data tambahan yang meningkatkan organisasi dari setiap frase-frase.

Setiap frase dilewatkan melalui suatu *look-ahead buffer* dan dikodekan dalam bagian *text windows*, LZSS menambahkan frase dalam struktur pohon. Pada implementasinya suatu *binary search tree*. Dengan mengurutkan frase-frase menjadi suatu *tree* seperti ini, maka waktu yang diperlukan untuk mencari frase *string* terpanjang pada proporsional dari ukuran *window* dan panjang frase. Selain itu, LZSS proporsi ini sesuai dengan logaritma basis 2 dari ukuran *window* dikalikan dengan panjang frase.

Penyimpanan yang dibentuk dengan menggunakan suatu *tree* tidak hanya membuat kompresi menjadi lebih efisien, tetapi juga memperbesar ukuran *window*. Menggandakan ukuran dari *text window* hanya menyebabkan suatu peningkatkan yang kecil dalam waktu kompresi, dimana sebelum digandakan.

Perubahan kedua berdasarkan atas *output token* aktual dari algoritma kompresi. Proses *recalling* dari *output token* LZ77 terdiri suatu *offset* frase, suatu *match length*, dan karakter yang diikuti oleh suatu frase. Ini berarti bahwa LZ77 dipaksa dengan *pointer* lainnya dengan *plain* karakter, dan tidak bergantung pada kealamian dari teks *input*.

Algoritma Proses Kompresi dapat dilihat pada Gambar 1.



Gambar 1. Algoritma Proses Kompresi

2. Algoritma Dekompresi

LZSS selain mengizinkan *pointer* dan karakter selama bebas dicampuradukkan. Pada awalnya sebagai contoh, algoritma kompresi LZSS mungkin tidak akan menemukan setiap frase yang cocok untuk menghasilkan selusin atau input simbol pertama. Di bawah sistem LZ77, *encoder* masih mempunyai *output* suatu *dummy match position* dengan panjang nol dari setiap simbol yang dihasilkannya. (Nelson, 2002: 36).

Selain itu LZSS menggunakan suatu bit tunggal sebagai suatu *prefix* untuk setiap *output token* untuk mengindikasikan apakah ini merupakan suatu pasangan *offset*/*length* ataupun suatu simbol tunggal untuk dihasilkan. Ketika menghasilkan beberapa karakter tunggal, metode ini mengurangi *overhead* dari kemungkinan beberapa *byte* per karakter menurun hingga suatu *byte* tunggal per karakter.

Algoritma Proses Dekompresi dapat dilihat pada Gambar 2.

`

Gambar 2. Algoritma Proses DeKompresi

2.3. Perancangan

Pada program nantinya akan ditambahkan fasilitas *password* untuk mengamankan *file* yang telah dikompresi. Panjang *password* maksimum adalah 7 karakter dengan input semua jenis karakter dan bersifat *case sensitive* (huruf besar dan kecil dibedakan). Untuk keamanannya *password* disimpan ke *file output* dengan menggunakan fungsi XOR <*Password Length*> untuk membalikkan bit ditambahkan dengan fungsi XOR sekali lagi dengan nilai x XOR xx dimana x adalah posisi karakter. *Password* ini akan otomatis dibaca oleh program pada saat dekompresi balik. Pada saat *user* tidak mengetikkan karakter pada *field password* maka nilai otomatis akan diset menjadi nol yang berarti *file* output tidak mengandung *password*. Agar lebih jelas bagaimana cara enkripsi ini akan diberikan contoh seperti ini:

Misalkan *Password*: "**1234567**" berarti panjang *password* adalah 7 karakter
Untuk $n = 1 \rightarrow$ 49 XOR 1 XOR 11 = 59 (Nilai ASCII 1 adalah 49)
Untuk $n = 2 \rightarrow$ 50 XOR 2 XOR 22 = 38 (Nilai ASCII 2 adalah 50)
Untuk $n = 3 \rightarrow$ 51 XOR 3 XOR 33 = 17 (Nilai ASCII 3 adalah 51)
Untuk $n = 4 \rightarrow$ 52 XOR 4 XOR 44 = 28 (Nilai ASCII 4 adalah 52)
Untuk $n = 5 \rightarrow$ 53 XOR 5 XOR 55 = 7 (Nilai ASCII 5 adalah 53)
Untuk $n = 6 \rightarrow$ 54 XOR 6 XOR 66 = 114 (Nilai ASCII 6 adalah 54)
Untuk $n = 7 \rightarrow$ 55 XOR 7 XOR 77 = 125 (Nilai ASCII 7 adalah 55)

Berarti karakter terenkripsi yang akan disimpan ke dalam *file* adalah: ";&DC1FSBELr}".

Proses sebaliknya sebanyak dekripsi dilakukan maka dilakukan perbandingan *password* yang di-*input* dengan *password* yang disimpan dalam *file*. Adapun proses dekripsi baliknya adalah sebagai berikut:

Password dalam *file* ";&DC1FSBELr}"; panjang *password* adalah 7 karakter.
Untuk $n = 1 \rightarrow$ 59 XOR 1 XOR 11 = 49 (Nilai ASCII ; adalah 59)
Untuk $n = 2 \rightarrow$ 38 XOR 2 XOR 22 = 50 (Nilai ASCII & adalah 38)
Untuk $n = 3 \rightarrow$ 17 XOR 3 XOR 33 = 51 (Nilai ASCII DC1 adalah 17)
Untuk $n = 4 \rightarrow$ 28 XOR 4 XOR 44 = 52 (Nilai ASCII 4 FS adalah 28)
Untuk $n = 5 \rightarrow$ 7 XOR 5 XOR 55 = 53 (Nilai ASCII BEL adalah 7)
Untuk $n = 6 \rightarrow$ 114 XOR 6 XOR 66 = 54 (Nilai ASCII r adalah 114)
Untuk $n = 7 \rightarrow$ 125 XOR 7 XOR 77 = 55 (Nilai ASCII } adalah 125)

Berarti karakter yang didekripsi balik didapat kembali *string*: "**1234567**".

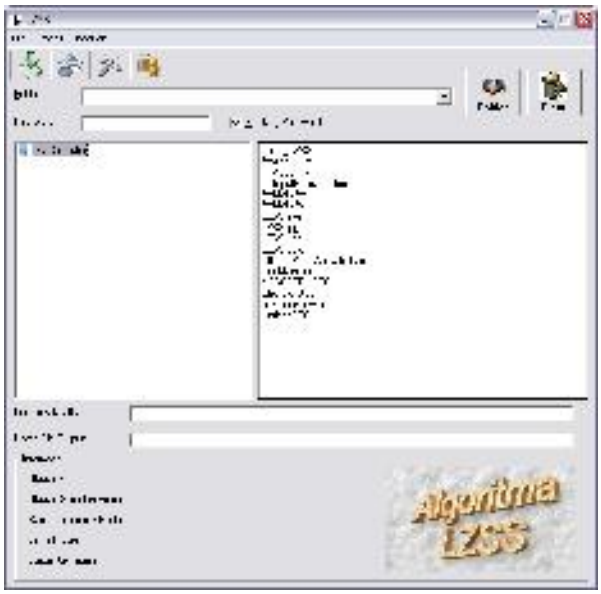
Sebagai catatan pada penerapan *password* digunakan pada saat proses kompresi hanya untuk tujuan verifikasi yaitu bagi *user* yang berhak dan mengetahui *password* lah yang dapat mendekompresi balik suatu *file* yang telah dikompresi dengan program yang dirancang. *Password* ini akan ditambahkan pada bagian belakang *file* hasil.

Untuk membedakan *file* hasil kompresi dengan *file* asli maka program akan menambahkan ekstensi tambahan yaitu ".LZS". Bila *file* tersebut didekompresi kembali maka ekstensi tambahan ini akan otomatis dibuang.

2.4. Implementasi

Antarmuka program ini cukup sederhana dimana pada bagian sisi kiri merupakan *tree view* untuk memilih *folder* yang nantinya semua *file* pada *folder* ini akan ditampilkan *File List* di sebelah kanan atas. Selanjutnya adalah memilih satu *file* dengan mengklik pada *file* tersebut. Program secara otomatis akan menampilkan *file* yang dipilih tersebut pada *text box file* yang dipilih.

Berikutnya adalah menentukan *folder output* dengan cara mengisi pada *combo box Folder* atau mengklik pada tombol di ujungnya. Kemudian program akan memunculkan sebuah kotak *dialog* untuk memilih *folder*. Setelah itu nama *file output* akan secara otomatis diberikan dan ditambahkan ekstensi *.LZS.



Gambar 3 Tampilan Program

Untuk melakukan proses kompresi atau dekompresi dapat mengklik pada tombol “ ” atau “ ”. Jika *user* memasukkan *password* pada saat kompresi *password* tersebut akan disimpan pada *file* hasil dan akan digunakan kembali sebagai konfirmasi saat *file* tersebut didekompresi balik. Panjang maksimum *password* adalah 7 karakter. Bagian *check box* “*Masking Password*” pada saat *user* memasukkan *password* akan digantikan dengan tanda asterik. Program tidak menyimpan *password* apabila *text box* pada bagian ini dikosongkan.

Berikutnya adalah pada tampilan utama ditekan tombol “ ” maka akan ditampilkan *form* seperti di atas. Penekanan tombol “OK” maka *form* ini disembunyikan dan fokus dikembalikan ke *form* utama.

3. Hasil dan Diskusi

Untuk mengetahui hasil pengujian algoritma LZSS yang telah diimplementasikan dapat dilihat pada Tabel 1. Secara keseluruhan ada 20 *file* yang diujicobakan dalam penelitian ini. Program pemampatan *file* dengan menggunakan metode LZSS ini dikembangkan dengan menggunakan bahasa pemrograman *Visual Basic* dan *output* dari sistem ini berupa *file* terkompresi yang disimpan dalam *file* .lzs. Hasil dekompresi dikembalikan ke bentuk asalnya melalui modul dekompresi LZSS, dengan perbandingan antara *file* hasil dekompresi dengan *file* asli adalah sama. Pada setiap percobaan dilakukan pengamatan terhadap kebutuhan ruang penyimpanan, analisis kecepatan proses program. Hasil dari penelitian menunjukkan bahwa rasio pemampatan untuk kelima jenis *file* berbeda-beda tergantung jenis dan ukuran *file*. Sedangkan lama waktu proses bergantung kepada ukuran *file* yang diproses.

Tabel 1. Tabel Pengujian Hasil Kompresi

No.	Jenis File	Ukuran File (byte)	Ukuran File Output (byte)	Rasio Kompresi	Lama Proses (ms)
1.	Text (*.TXT)	1.404	948	67,52 %	16
2	Text (*.TXT)	9.213	6.139	66,63 %	58
3	Wave (*.WAV)	8.400	5.639	67,10 %	60
4.	Wave (*.WAV)	10.026	8.208	81, 87 %	47
5.	BMP (*.BMP)	22.070	20.129	91,21 %	125
6	BMP (*.BMP)	22.080	11.262	50,99 %	125
7.	Excel Worksheet (*.XLS)	22.016	18.378	80, 55%	132
8.	Excel Worksheet (*.XLS)	36.538	31.795	87,02 %	188
9.	Power Point (*.PPT)	36.614	32.769	89,50 %	190
10.	Power Point (*.PPT)	36.636	30.756	83,95 %	192
11	Microsoft Word (*.DOC)	55.296	48.627	87,93 %	264
12.	Microsoft Word (*.DOC)	80.856	70.725	87,47 %	406
13.	Access (*.MDB)	179.704	141.211	78,58 %	735
14	Access (*.MDB)	282.668	241.861	85,58 %	1.265
15	Executable (*.EXE)	424.644	368.269	86,73 %	1.781
16.	Executable (*.EXE)	455.547	405.267	88,96%	1.785
17	HTML (*.HTML)	2.456	2.014	82,00%	25
18	HTML (*.HTML)	1.647	1.336	81,19%	15
19.	Windows Metafile (*.WMF)	5.689	4.952	80,74%	31
20.	Windows Metafile (*.WMF)	6.890	5.678	82,41%	30

Tabel 2. Tabel Pengujian Hasil Dekompresi

No.	Nama File	Ukuran File (byte)	Ukuran File Output (byte)	Rasio Dekompresi	Lama Proses (ms)
1.	Text (*.TXT)	948	1.404	148,10%	15
2	Text (*.TXT)	4.139	9.213	126,53%	78
3	Wave (*.WAV)	5.639	8.400	148,94%	60
4.	Wave (*.WAV)	8.208	10.026	122,15%	48
5	BMP (*.BMP)	20.129	22.070	109,64%	125
6.	BMP (*.BMP)	11.262	22.080	196,00%	127
7	Excel Worksheet (*.XLS)	18.378	22.016	119,79%	110
8.	Excel Worksheet (*.XLS)	31.795	36.538	114,92%	186
9	Power Point (*.PPT)	32.769	36.614	111,73%	188
10	Power Point (*.PPT)	30.756	36.636	119,12%	189
11	Microsoft Word (*.DOC)	48.627	55.296	114,00%	261
12.	Microsoft Word (*.DOC)	70.725	80.856	114,32%	400
13.	Access (*.MDB)	141.211	179.704	127,26%	734
14	Access (*.MDB)	241.861	282.668	116,82%	1.260
15	Executable (*.EXE)	368.269	424.644	115,30%	1.780
16	Executable (*.EXE)	405.267	455.547	112,40%	1.736
17.	HTML (*.HTML)	2.014	2.456	121,95%	20
18	HTML (*.HTML)	1.336	1.647	123,17%	10
19.	Windows Metafile (*.WMF)	4.592	5.689	123,83%	30
20	Windows Metafile (*.WMF)	5.678	6.890	121,34%	31

Penembangan ke deapan diharapkan hasil kompresi dapat dibuka oleh program kompresi seperi winrar dan winzip.

4. Kesimpulan dan Saran

4.1. Kesimpulan

Berdasarkan pengujian dari bab sebelumnya yang telah dilakukan maka dapat diambil beberapa kesimpulan sebagai berikut:

1. LZSS merupakan algoritma kompresi yang termasuk dalam *dictionary compressor*. 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*.

2. Dari hasil pengujian yang dilakukan untuk beberapa jenis *file* biner bila dilakukan kompresi sebanyak dua kali berturut-turut akan menghasilkan ukuran *file* yang sedikit lebih besar dari hasil kompresi pertama. Hal ini disebabkan oleh penyimpanan data *dictionary* untuk kedua kalinya pada *file output* kedua.
3. Lama waktu kompresi dan dekompresi tergantung pada spesifikasi komputer yang digunakan. Semakin besar ukuran *file* berarti lama proses juga semakin lama.
4. Seperti halnya dengan algoritma *dictionary* lainnya jika ditemukan kasus terburuk dimana tidak terdapat perulangan *string* yang banyak maka ukuran kompresi *file* bukannya menjadi lebih kecil tetapi semakin besar. Hal ini disebabkan oleh pada *file output*-nya juga ikut disimpan isi dari *dictionary* hasil pembacaan *file*.
5. Tujuan dari penelitian ini adalah menghasilkan perangkat lunak kompresi dengan memanfaatkan algoritma LZSS dimana setelah dilakukan pengujian dapat melakukan kompresi data sehingga dapat dikatakan tujuan penelitian tersebut tercapai.
6. Dalam hal manfaat, dengan adanya tulisan ini dapat diketahui teknik dan cara kerja LZSS serta teknik perancangan program LZSS sehingga dapat menambah wawasan pembaca dan peneliti.
7. Adapun keunggulan dari program ini adalah mampu melakukan kompresi pada *file* biner dan teks dengan menggunakan algoritma LZSS.

4.2. Saran

Untuk pengembangan lebih lanjut program kompresi ini, maka dapat diberikan beberapa saran sebagai berikut:

1. Metode ini lebih disarankan untuk digunakan pada jenis *file* teks dan *bitmap* dengan variasi warna yang seragam.
2. Melakukan studi perbandingan rasio kompresi, waktu eksekusi, dan rasio reduksi antara beberapa metode kompresi.
3. Merancang aplikasi agar dapat mengkompresi seluruh *file* dalam satu *folder*.

Daftar Pustaka

[1] Anonim, 2010, **Pengenalan Struktur Data dan Algoritma**, Modul Kuliah, PDF Version,http://lecturer.eepisits.edu/~entin/Struktur%20Data%20&%20Algoritma/buku/Data%20Structure%20-%20Bab%201.pdf, tanggal akses 21 Juli 2011.

[2] **ASCII**, **http://www.ascii.com**, tanggal akses 10 Agustus 2011.

[3] Nelson, M. and J. L. Gailly, 2002, **The Data Compression Book**, Second Edition, M & T Books.

[4] Rinaldi, 2010, **Pohon**, www.informatika.org/~rinaldi/Matdis/Pohon-2.doc.

[5] Salomon, D., **Data Compression: The Complete Reference**, Second Edition, Springer, 2004.

[6] Shannon, C. E., **A Mathematical Theory of Communication**, The Bell System Technical Journal, Vol. 27, July, October, 1948.

[7] Wardoyo, Irwan, Peri Kusdinar, dan Irvan Hasbi Taufik, **Kompresi Teks**, Jurusan Teknik Informatika, Sekolah Tinggi Teknologi Telkom, Bandung.

[8] Visual Basic, **http://id.wikipedia.org/wiki/Visual_Basic**, tanggal akses 02 Oktober 2011.