PENGGABUNGAN ALGORITMA *LEMPEL-ZIV-WELCH* (LZW) DAN ALGORITMA *HUFFMAN KANONIK* UNTUK KOMPRESI *SHORT MESSAGE SERVICES* (SMS)

SKRIPSI

oleh:

SURYA BUDI PRABOWO

0410963052-96

PROGRAM STUDI ILMU KOMPUTER

JURUSAN MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS BRAWIJAYA

MALANG

2011

PENGGABUNGAN ALGORITMA *LEMPEL-ZIV-WELCH* (LZW) DAN ALGORITMA *HUFFMAN KANONIK* UNTUK KOMPRESI *SHORT MESSAGE SERVICES* (SMS)

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar
Sarjana dalam bidang Ilmu Komputer

oleh:
SURYA BUDI PRABOWO
0410963052-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2011

LEMBAR PENGESAHAN SKRIPSI

PENGGABUNGAN ALGORITMA *LEMPEL-ZIV-WELCH* (LZW) DAN ALGORITMA *HUFFMAN KANONIK* UNTUK KOMPRESI *SHORT MESSAGE SERVICE* (SMS)

oleh :

SURYA BUDI PRABOWO

0410963052 - 96

Telah dipertahankan di depan Majelis Penguji
pada tanggal 18 Agustus 2011
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana Komputer dalam bidang Ilmu Komputer

Pembimbing I	Pembimbing II
Edy Santoso,SSi.,M.Kom	**Drs. Marji. MT.**
NIP. 197404142003121004	**NIP. 196708011992031001**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Abdul Rouf Alghofari, MSc.
NIP. 196709071992031001

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**: Surya Budi Prabowo**
NIM	**: 0410963052**
Jurusan	**: Matematika**
Penulis skripsi berjudul	**:Penggabungan Algoritma Lempel Ziv Welch (LZW) dan Algoritma Huffman Kanonik untuk kompresi Short Message Service (SMS).**

Dengan ini menyatakan bahwa :

1. **Isi dari skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam skripsi ini.**
2. **Apabila dikemudian hari ternyata skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 18 Agustus 2011
Yang menyatakan,

(Surya Budi P.)
NIM. 0410963052-96

BAB I
PENDAHULUAN

1.1 Latar Belakang

Kemajuan teknologi yang cukup cepat pada saat ini membuat kebutuhan akan informasi dan komunikasi semakin tinggi namun mudah didapatkan. *Handphone* (telepon genggam) merupakan salah satu piranti komunikasi dan informasi, yang menjadi media masyarakat untuk bertukar informasi maupun untuk berkomunikasi, baik komunikasi suara maupun teks. Salah satu layanan *handphone* yang digemari oleh masyarakat adalah *Short Message Services* (SMS) karena dianggap mudah dan murah. Namun, masing-masing operator memiliki tarif yang bervariasi sehingga membingungkan masyarakat operator seluler mana yang memberikan layanan terbaik dengan tarif yang termurah. Sebagai perbandingan, tarif SMS semua operator di Indonesia dapat dilihat pada tabel 1.1 :

Tabel 1.1. Perbandingan tarif SMS berbagai operator di Indonesia

Operator Selular	Tarif SMS
AS	Rp 150,- / sms
XL	Rp 150,- / sms
Three	Rp 55,- / sms
AXIS	Rp 100,- / sms
IM3	Rp 100,- / sms
Mentari	Rp 50,- / sms
Esia	Rp 1,- / karakter
Flexi	Rp 49,- / sms
Simpati	Rp 150,- / sms

Sebuah pesan SMS maksimal terdiri dari 140 bytes, dengan kata lain sebuah pesan bisa memuat 140 karakter 8-bit, 160 karakter 7-bit atau 70 karakter 16-bit untuk bahasa Jepang, Bahasa Mandarin dan Bahasa Korea yang memakai *Hanzi* (Aksara *Kanji/Hanja*). Dalam melakukan pengiriman pesan SMS seorang pengguna dapat mengirim pesan lebih dari 140 byte, tetapi untuk itu seorang pengguna harus membayar lebih dari sekali. Hal ini terjadi karena pesan yang dikirimkan terdiri lebih dari satu halaman sehingga proses pengiriman pesan akan dilakukan sebanyak jumlah halaman

yang ada, jumlah halaman sesuai dengan isi SMS yang diketikkan (Heri,2008).

Salah satu cara untuk menghemat biaya tiap SMS yaitu dengan menyingkat isi pesan tersebut agar muat dalam satu halaman. Hal ini cukup merepotkan bagi beberapa pengguna, karena tidak semua pengguna *handphone* mengerti tentang arti singkatan – singkatan dalam 'bahasa' SMS sehingga diperlukan sebuah aplikasi untuk mengkompresi isi pesan teks SMS tersebut agar dapat memuat karakter SMS yang lebih banyak dalam satu halaman. Kompresi merupakan proses pengubahan sekumpulan data menjadi suatu bentuk kode untuk menghemat kebutuhan tempat penyimpanan dan waktu untuk transmisi data (Linawati, 2004).

Penelitian kompresi SMS dengan menggunakan algoritma *Huffman Kanonik* dan *Lempel Ziv Welch (LZW)* pernah dilakukan oleh Heri Purwanto (2008). Aplikasi kompresi SMS dengan menggunakan algoritma *Huffman* kanonik secara rata-rata mampu melakukan kompresi teks SMS dengan rasio kompresi sebesar 36.02% dengan tingkat keberhasilan mereduksi jumlah halaman SMS sebesar 75%. Apabila menggunakan algoritma LZW aplikasi mampu mengkompresi teks SMS dengan rasio kompresi sebesar 21.90% dengan tingkat keberhasilan mereduksi jumlah halaman SMS sebesar 18.37%.(Heri Purwanto, 2008)

Berdasarkan pada hasil penelitian diatas, akan dibuat suatu aplikasi kompresi SMS menggunakan penggabungan algoritma *Lempel Ziv Welch (LZW)* dan *Huffman Kanonik* dengan harapan agar hasil kompresi SMS yang dihasilkan lebih optimal.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang di atas, maka dapat dirumuskan beberapa permasalahan, antara lain :

1. Bagaimana implementasi metode *Lempel Ziv Welch (LZW)* dan *Huffman Kanonik* untuk kompresi *Short Message Service* (SMS).
2. Bagaimana rasio kompresi penggabungan metode *Lempel Ziv Welch (LZW)* dan *Huffman Kanonik* untuk kompresi SMS.

1.3 Batasan Masalah

Dari permasalahan diatas, berikut ini diberikan batasan untuk menghindari melebarnya masalah yang akan diselesaikan:

1. SMS yang dikompresi adalah SMS yang menggunakan karakter standar ASCII 7 bit.

2. Karakter yang akan dikompresi adalah huruf latin (besar dan kecil), angka latin, dan beberapa tanda baca yang umum.
3. Jumlah tabel *Huffman Kanonik* yang dibuat dibatasi satu tabel.

1.4 Tujuan

Tujuan yang ingin dicapai dalam skripsi ini adalah:

1. Mengimplementasikan algoritma *Lempel Ziv Welch (LZW)* dan *Huffman Kanonik* pada kompresi *Short Messagge Service* (SMS).
2. Menganalisa rasio kompresi algoritma *Lempel Ziv Welch (LZW)* dan *Huffman Kanonik* dalam melakukan kompresi SMS.

1.5 Manfaat

Manfaat yang akan dicapai dari skripsi ini adalah dihasilkannya perangkat lunak yang dapat melakukan kompresi SMS, sehingga dapat menampung lebih banyak karakter dalam satu halaman SMS.

1.6 Metodologi

Metodologi yang digunakan dalam penyusunan skripsi ini adalah sebagai berikut :

1. Studi literatur
   Studi ini dilakukan dengan cara mencari sekaligus mempelajari beberapa literatur dan artikel mengenai kompresi SMS, *Huffman Kanonik* dan *Lempel Ziv Welch (LZW)*
2. Pendefinisian dan Analisis Masalah
   Mendefinisikan dan menganalisis masalah untuk mencari solusi yang tepat.
3. Perancangan dan implementasi perangkat lunak
   Membuat perancangan model perangkat lunak dan mengimplementasikan hasil rancangan tersebut dengan membuat perangkat lunak kompresi dan dekompresi teks SMS menggunakan *Huffman Kanonik* dan *Lempel Ziv Welch (LZW)*.
4. Uji coba dan analisa hasil implementasi
   Menguji perangkat lunak, kemudian menganalisa hasil dari implementasi kompresi SMS sesuai dengan tujuan yang dirumuskan sebelumnya, untuk kemudian dievaluasi dan disempurnakan.

1.7 Sistematika Penulisan

BAB I : PENDAHULUAN

Dalam bab ini dijelaskan akan pentingnya kompresi teks khususnya untuk menghemat biaya pengiriman SMS yang termuat dalam latar belakang, perumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian, dan sistematika penulisan.

BAB II : DASAR TEORI

Dalam bab ini dijelaskan dasar-dasar teori yang menjadi acuan dalam proses pembuatan aplikasi kompresi teks.

BAB III : METODE DAN PERANCANGAN

Dalam bab ini dijelaskan metode yang digunakan dan tahapan-tahapan perhitungan kompresi masing-masing algoritma.

BAB IV : IMPLEMENTASI DAN PEMBAHASAN

Dalam bab ini dijelaskan mengenai implementasi program, uji coba dan analisanya.

BAB V : PENUTUP

Dalam bab ini berisi kesimpulan dan saran..

DAFTAR PUSTAKA

LAMPIRAN

BAB II
TINJAUAN PUSTAKA

Bab ini mencantumkan beberapa tinjauan pustaka dan referensi-referensi yang berkaitan dengan penelitian ini. Pada tahap pertama akan dijelaskan mengenai *Short Message Service* (SMS) sebagai aplikasi untuk dilakukannya kompresi. Kemudian dijelaskan juga tentang pengkodean standar dari SMS yang menggunakan kode ASCII.

Pada tahap kedua akan dijelaskan mengenai algoritma yang akan dipakai untuk melakukan kompresi. Sedangkan pada tahap ketiga dijelaskan mengenai kompresi *Huffman Kanonik* dan *Lempel Ziv Welch (LZW)* beserta proses-proses yang dilakukan, dan pada tahap terakhir akan diberikan rumus rasio kompresi untuk pengujian yang akan dilakukan.

2.1 *Short Message Service* (SMS)

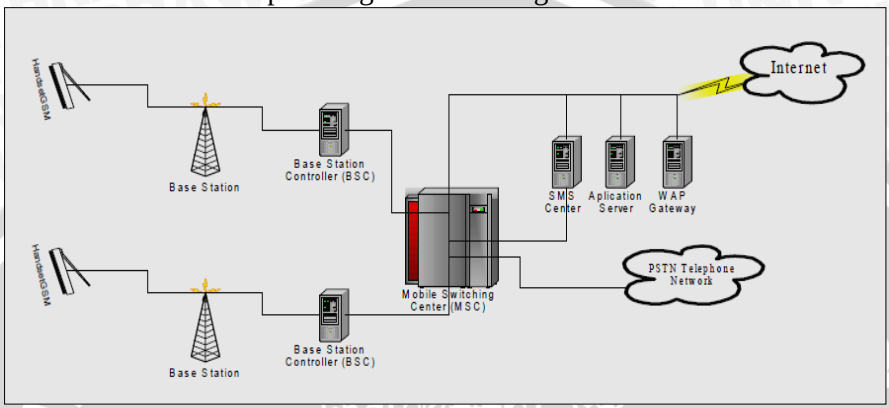
SMS (*Short Message Service*) merupakan fasilitas standar dari *Global System for Mobile Comunication* (GSM). Fasilitas ini dipakai untuk mengirim dan menerima pesan dalam bentuk teks ke dan dari sebuah ponsel.

Beberapa karakteristik SMS adalah :

- Sebuah pesan singkat terdiri atas 160 karakter yang terdiri dari atas huruf atau angka. Juga dapat mendukung pesan *non-text*, seperti *format binary*.
- Prinsip kerjanya adalah "menyimpan" dan "menyampaikan" pesan (*store and forward message*). Dengan kata lain pesan tidak langsung dikirimkan ke penerima, tetapi disimpan dahulu di SMS*Centre.*
- Memiliki ciri dalam hal konfirmasi pengiriman pesan, yaitu pesan yang dikirimkan tidak secara sederhana dikirimkan dan dipercayai akan disampaikan dengan selamat. Namun pengirim pesan dapat pula menerima pesan balik yang memberitahukan apakah pesan telah terkirim atau gagal.
- Pesan dapat dikirim dan diterima secara simultan dengan panggilan jenis layanan GSM lain.

Sebagai bagian dari system GSM, SMS adalah layanan yang sebenarnya merupakan *bearer service* atau paket pengirim dari data

GSM.Bearer service ini bekerja pada lapisan fisik yang merupakan lapisan terbawah dari protokol aplikasi data GSM.Arsitektur GSM itu sendiri adalah seperti tergambar dalam gambar 2.1 :



Gambar 2.1 Arsitektur GSM

SMS adalah type *asynchronous message* yang pengiriman datanya dilakukan dengan mekanisme protokol *store and forward.* Hal ini berarti bahwa pengirim dan penerima SMS tidak perlu berada dalam status berhubungan (*connected/online*) satu sama lain,ketika akan saling bertukar pesan SMS.Pengiriman pesan melalui *store and forward* berarti pengirim pesan SMS dan nomor telpon tujuan dan kemudian mengirimkannya (*store*) ke *server* SMS (SMS-*Center*) yang kemudian bertanggung jawab untuk mengirimkan pesan tersebut (*forward*) ke nomor telpon tujuan.Hal ini mirip dengan mekanisme *store and forward* pada protokol SMTP yang digunakan dalam pengiriman *e-mail* internet.Keuntungan mekanisme *store and forward* pada SMS adalah penerima tidak perlu dalam status *online* ketika ada pengirim yang bermaksud mengirim pesan kepadanya, karena pesan akan dikirim oleh pengirim ke SMSC yang kemudian dapat menunggu untuk meneruskan pesan tersebut ke penerima ketika ia siap dan dalam status *online* dilain waktu. Pada waktu pesan SMS telah terkirim dan diterima oleh SMSC, pengirim akan menerima pesan singkat (konfirmasi) bahwa pesan telah dikirim (*message sent*).

Sebuah SMS terdiri atas 160 karakter untuk skema 7 bit, dan ada yang memakai skema 8 bit berjumlah 140 karakter, atau 70 karakter memakai skema 16 bit (Bambang, 2007). Maksud skema disini adalah pemakaian jenis tabel ASCII, jika suatu operator selular

memakai skema 7 bit berarti tabel yang digunakan adalah tabel ASCII 7 bit. Untuk operator-operator yang berkembang di Indonesia saat ini kebanyakan memakai skema 7 bit. Penggabungan SMS dan kompresi SMS (terdiri lebih dari 160 karakter) telah dikembangkan. Tetapi fitur-fitur ini belum diimplementasikan oleh semua jaringan operator selular di dunia.

2.2 ASCII 7 bit

Penelitian ini akan membahas tentang SMS yang menggunakan kode ASCII 7 bit. Selain itu karena metode Huffman memodifikasi dan merubah rangkaian bit kode ASCII, maka akan dijelaskan mengenai prinsip penggunaan dan pengkodean ASCII 7 bit pada karakter.

Kode Standar Amerika untuk Pertukaran Informasi atau ASCII (*American Standard Code for Information Interchange*) merupakan suatu standar internasional berupa rangkaian bit kode untuk mewakili teks, agar bisa dimengerti oleh banyak model komputer (Ravi, 2008).

Ada dua jenis kode yang sering digunakan, yaitu ASCII- 8 bit dan ASCII- 7 bit. Kode yang terdapat pada ASCII-8 bit jauh lebih lengkap dari ASCII-7 bit. Menurut Tino ASCII-7 bit mempunyai kombinasi kode $2^7 = 127$, dengan ketentuan sebagai berikut:

1. 26 kode untuk huruf kapital *(upper case)* dari A –Z.
2. 26 kode untuk huruf kecil *(lower case)* dari a –z.
3. 10 digit desimal dari 0 –9.
4. 34 karakter kontrol untuk informasi status operasi komputer.
5. 32 karakter khusus *(special characters)*.

Kode ASCII dengan nilai kode 0 sampai dengan 31 dan 127 termasuk dalam status karakter-karakter kontrol yang tidak dapat dicetak (*non-printable characters*). Contohnya, tabel ASCII karakter 28 mewakili fungsi "*file separator*", atau karakter 8 yang mewakili *backspace*.

Kode ke 32 adalah karakter *spasi*, menandakan pemisah antara kata, yang dihasilkan oleh *space-bar* dari *keyboard*. Kode 33 sampai 126 dikenal sebagai karakter-karakter yang dapat di cetak (*printable characters*), terdiri dari beberapa huruf, angka, tanda baca, dan beberapa simbol. Tabel 2.1 adalah beberapa contoh karakter ASCII 7 bit yang merupakan *printable characters* (Ravi, 2008).

Tabel 2.1 Kode ASCII-7 Bit

Binary	Oct	Dec	Hex	**Value**

011 0000	60	48	30	0
011 0001	61	49	31	1
011 0010	62	50	32	2
011 0011	63	51	33	3
011 0100	64	52	34	4
011 1010	72	58	3A	:
011 1010	73	59	3B	;
011 1100	74	60	3C	<
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D

2.3 Frekuensi Pemakaian Huruf Dalam Bahasa Indonesia

Penelitian ini memakai *Huffman* dengan tabel yang tetap, oleh sebab itu akan diberikan beberapa data mengenai pemakaian huruf untuk mendukung analisis dari pembentukan tabel *Huffman* yang akan dibuat pada bab 3.

Menurut penelitian yang dilakukan Aysar (2008) pada 2 artikel bahasa Indonesia yang berbeda, didapatkan data mengenai frekuensi pemakaian huruf ditunjukkan pada Tabel 2.2 :

Tabel 2.2 Data Frekuensi Pemakaian Huruf I

Rangking	File Indonesia01.txt (sumber tvone.co.id)		File Indonesia02.txt (sumber kompas.com)	
	Huruf	**Frekuensi**	**Huruf**	**Frekuensi**
1	a	690	a	634
2	e	137	n	130
3	n	127	e	126
4	i	104	r	109
5	t	80	u	87
6	u	76	m	85
7	k	72	i	84
8	r	69	t	59
9	d	66	s	57
10	m	59	k	50

Pada penelitian yang dilakukan Galih (2008) terhadap 3 artikel bahasa Indonesia yang berbeda, didapatkan data mengenai frekuensi pemakaian huruf ditunjukkan pada Tabel 2.3 :

Tabel 2.3 Data Frekuensi Pemakaian Huruf II

Rangking	File 1 (sumber detikinet.com)		File 2 (sumber detikinet.com)		File 3 (sumber detikinet.com)	
	Huruf	**Frek**	**Huruf**	**Frek**	**Huruf**	**Frek**
1	a	112	a	137	a	134
2	i	73	i	76	n	79
3	n	60	n	61	e	59
4	e	57	e	59	i	57
5	s	55	s	54	s	43

Menurut *American Cryptogram Association* (2005) menyebutkan data mengenai frekuensi pemakaian huruf dalam bahasa Indonesia adalah seperti ditunjukkan pada Tabel 2.4 :

Tabel 2.4 Data Frekuensi Pemakaian Huruf III

Rangking	**Huruf**	**Frekuensi**
1	a	22.72%
2	n	8.79%
3	e	8.68%
4	i	6.00%
5	k	5.52%
6	r	5.27%
7	u	4.94%
8	t	4.85%
9	s	4.70%
10	d	4.54%
11	m	4.14%
12	p	3.16%
13	b	3.10%
14	ng	3.03%
15	h	2.70%
16	l	2.53%
17	y	1.63%
18	o	1.20%

19	g	0.98%
20	j	0.68%
21	w	0.39%
22	c	0.32%
23	f	0.13%
24	v	0.02%

2.4 Kompresi

Kompresi atau *compression* adalah proses pemampatan ukuran sebuah data. Sebuah data terkadang memiliki sebuah informasi yang sama dan berulang-ulang. Sebuah informasi yang sama dan berulang-ulang membuat ukuran sebuah data menjadi besar. Pada transmisi data (*data transmision*), sebuah data berukuran besar akan membutuhkan waktu *transfer* lebih lama dibandingkan data berukuran kecil. Pada *file*, sebuah *file* yang berukuran besar menyita banyak ruangan pada media penyimpanan. Oleh karena itu, untuk menghindari masalah tersebut, maka dilakukan kompresi data. Dengan mengunakan algoritma-algoritma dan teknik-teknik tertentu, informasi yang sama dan berulang-ulang tersebut dikodekan sedemikian rupa sehingga data tersebut menjadi berukuran lebih kecil. *File* atau data yang sudah dikompres agar bisa digunakan kembali harus dikembalikan lagi seperti semula. Proses pengembalian sebuah *file* yang terkompres menjadi seperti *file* aslinya disebut dekompresi atau *decompression*.

Jenis kompresi data berdasarkan *mode* penerimaan data oleh manusia:

1. *Dialoque Mode* yaitu proses penerimaan data dimana pengirim dan penerima seakan berdialog (*real time*), seperti pada contoh *video conference*. Kompresi data pada *mode* ini harus berada dalam batas penglihatan dan pendengaran manusia. Waktu tunda (*delay*) tidak boleh lebih dari 150 ms, dimana 50 ms untuk proses kompresi dan dekompresi dan 100 ms untuk mentransmisikan data dalam jaringan.
2. *Retrieval Mode* yaitu proses penerimaan data tidak dilakukan secara *real time*. *Retrieval Mode* dapat dilakukan secara *fast forward* maupun *fast rewind* di *client*, dan dapat dilakukan *random access* terhadap data dan dapat bersifat interaktif *(*Anynomous, 2005).

Jenis kompresi data berdasarkan output :

1. *Lossy Compression*
   - Teknik kompresi dimana terdapat data yang hilang selama proses kompresi. Artinya data hasil dekompresi tidak sama dengan data sebelum kompresi namun sudah cukup untuk digunakan. Contoh: Mp3, *streaming media*, JPEG, MPEG, dan WMA.
   - Kelebihannya yaitu ukuran file lebih kecil dibanding *loseless* namun masih tetap memenuhi syarat untuk digunakan.
   - Prinsip dasar dari *Lossy Compression* adalah membuang bagian-bagian data yang tidak digunakan, tidak dirasakan, tidak begitu dilihat oleh manusia sehingga manusia masih beranggapan bahwa data tersebut masih bisa digunakan walaupun sudah dikompresi.

2. *Loseless Compression*
   - Teknik kompresi dimana data hasil kompresi dapat didekompres lagi dan hasilnya tepat sama seperti data sebelum proses kompresi. Contoh aplikasi: ZIP, RAR, GZIP, 7-Zip
   - Data hasil dekompresi tetap sama dengan data sebelum dikompres, artinya tidak terdapat data yang hilang atau data kembali seperti sebelum proses kompresi.

Untuk *Huffman Kanonik Coding* yang digunakan dalam penelitian ini termasuk dalam metode *Loseless Compression (*Anynomous, 2005).

2.5 Algoritma Huffman Kanonik

Kode *Huffman kanonik* merupakan suatu bentuk variasi dari kode *Huffman* yang memiliki sifat dapat dideskripsikan dengan sangat mampat (*compactly described*). (Anonymous,2006)

Hal ini disebabkan kode *Huffman kanonik* memiliki aturan-aturan penulisan yang baku sehingga beberapa hal dapat disepakati dan tidak perlu dituliskan pada deskripsi kode. Contohnya adalah kode harus terurut berdasarkan urutan simbol, panjang kode suatu simbol harus sesuai dengan aras simpul simbol tersebut pada pohon *Huffman*, dll. Deskripsi kode yang *compact* membuat kode ini lebih efisien dalam beberapa hal dibandingkan dengan kode Huffman.

Algoritma *huffman* akan menggunakan tabel yang menyimpan frekuensi kemunculan dari masing-masing simbol yang digunakan

dalam file tersebut dan kemudian mengkodekannya dalam bentuk biner (Liliana, Lipesik, V.J., 2006,). Pada penelitian ini, untuk melakukan proses enkoding dengan menggunakan algoritma *Huffman kanonik* proses pembuatan pohon *Huffman* tidak akan dilakukan. Hal ini dilakukan karena pihak penerima pesan (*decoder*) akan mengalami kesulitan untuk men-dekode pesan jika informasi pohon *Huffman* tersebut tidak ikut dikirimkan. Penambahan informasi mengenai pohon *Huffman* akan memerlukan tempat tersendiri sehingga proses kompresi menjadi kurang efektif. Hal ini diperkuat dengan hasil penelitian Liliana dan Lipesik V.J yang menyimpulkan bahwa proses kompresi file kurang berhasil jika isi file terlalu sedikit sehingga ukuran file asli bisa menjadi lebih kecil dari ukuran file hasil kompresi karena file kompresi masih harus menyimpan *Huffman tree*-nya.

2.5.1 Proses Pembentukan Kode Huffman Kanonik

Pada penelitian ini proses kompresi akan diterapkan pada teks SMS yang mempunyai kapasitas kecil yaitu 140 *byte* untuk setiap halaman SMS. Hal ini sesuai dengan kesimpulan yang dihasilkan pada penelitian Liliana dan Lipesik V.J (2006) dan dengan dihilangkannya proses pembuatan pohon *Huffman* maka hal ini dapat menghemat waktu proses sehingga proses enkoding dapat dilakukan lebih cepat. Untuk menggantikan informasi mengenai pohon *Huffman* tersebut maka dibuat suatu tabel, tabel *Huffman*, yang berisi kode-kode *Huffman* kanonik dari karakter-karakter SMS *default* yang akan digunakan. Tabel ini bersifat statis dan akan digunakan oleh aplikasi, baik aplikasi pengirim (*encoder*) maupun penerima (*decoder*), sebagai acuan untuk melakukan proses *encoding* / *decoding* terhadap teks SMS.

2.5.1.2 Proses *Encoding*

Tabel *Huffman* akan digunakan sebagai acuan untuk melakukan proses kompresi. Tabel *Huffman* dibuat dengan cara menentukan kode *Huffman* dari karakter-karakter SMS tersebut. Penentuan kode *Huffman* ini dilakukan dengan cara memberi kode yang pendek untuk karakter yang sering diakses begitu pun sebaliknya. Contoh tabel kode *Huffman* dapat dilihat pada tabel 2.5 berikut :

Tabel 2.5 Kode Huffman

Huruf	Kode Huffman	Angka	Panjang Kode
A	001	1	3
B	0000	0	4
C	1001	9	4
D	101	5	3
E	01	1	2
F	0001	1	4
G	1000	8	4
H	11	3	2

Dari data tabel kode *Huffman* yang telah dibentuk diatas, huruf A memiliki 3 bit kode, huruf B 4 bit kode,dan seterusnya. Untuk membentuk suatu kode Huffman Kanonik,maka langkah awal yang harus dilakukan adalah mengurutkan data tersebut sesuai dengan panjang bit terbesar diikuti ($l_{max} \ldots l_{min}$) dengan urutan huruf *alphabet* sesuai dengan kode ASCII. Sehingga hasil pengurutannya ditunjukkan pada tabel 2.6 :

Tabel 2.6 Kode Huffman setelah pengurutan

Huruf	Kode Huffman	Angka	Panjang Kode
B	0000	0	4
C	1001	9	4
F	0001	1	4
G	1000	8	4
A	001	1	3
D	101	5	3
E	01	1	2
H	11	3	2

Setelah terbentuk kode *Huffman* yang telah dilakukan pengurutan, langkah berikutnya adalah melakukan transformasi menjadi kode *Huffman kanonik.* Secara garis besar data yang dibutuhkan pada tabel *Huffman* meliputi data-data berikut :

a. Huruf, *string* dari karakter SMS dan akan diurutkan berdasarkan nilai ASCII dari yang terkecil sampai yang terbesar. Tujuan dari pengurutan ini adalah untuk memudahkan dalam melakukan pencarian huruf pada saat melakukan proses enkoding *Huffman Kanonik.*
b. Kode *Huffman Kanonik*, menyatakan kode *Huffman Kanonik* yang dibentuk dari kode *Huffman* yang ada.
c. Angka, menyatakan representasi bilangan yang dibentuk oleh bit bit pada kode *Huffman kanonik* untuk karakter SMS tersebut. Kolom angka diperlukan untuk memudahkan saat melakukan proses dekoding *Huffman* Kanonik.
d. Panjang Kode, menyatakan panjang/lebar bit dari kode *Huffman kanonik* yang telah dibentuk.

Langkah-langkah proses *encoding* pada algoritma Huffman Kanonik adalah sebagai berikut :

1. Panjang kode untuk suatu simpul adalah sebesar aras+1 simpul tersebut. *String* biner simpul paling dalam yang terletak paling kiri diberi nilai 0 semuanya. Untuk simpul berikutnya (bergeser dari kiri ke kanan) *string* binernya naik satu nilai dari simpul sebelumnya.
2. Apabila semua simpul pada kedalaman yang sama telah di-*encode*, maka proses *encoding* dilanjutkan ke aras yang lebih rendah. Hanya saja *string* binernya tidak dimulai dengan semuanya 0. *String* biner simpul paling kiri dimulai dengan kode baru. Kode baru itu merupakan kenaikan 1 nilai dari *string* biner simpul yang terakhir di*encode* namun biner paling belakangnya dihilangkan sehingga panjang kodenya berkurang satu.
3. Simpul berikutnya di-*encode* dengan *string* binernya naik satu nilai (bergeser dari kiri ke kanan). Proses akan berhenti bila telah mencapai akar.

Hasil transformasi kode *Huffman* menjadi kode *Huffman Kanonik* ditunjukkan pada tabel 2.7 :

Tabel 2.7 Transformasi Kode *Huffman* ke *Huffman Kanonik*

Huruf	Kode Huffman	Kode Huffman Kanonik	Angka	Panjang Kode
B	0000	0000	0	4
C	1001	0001	1	4
F	0001	0010	2	4
G	1000	0011	3	4
A	001	010	2	3
D	101	011	3	3
E	01	10	2	2
H	11	11	3	2

2.5.1.3 Proses *Decoding*

Pada proses encoding, *string* teks SMS yang akan dikompres akan dikodekan setiap karakternya berdasarkan tabel *Huffman Kanonik* yang telah dibentuk. Sebagai contoh *string* "CABE", berdasarkan tabel 2.6, akan dikodekan menjadi "0001010000010 ".

Dalam melakukan proses dekoding maka hal pertama yang harus dilakukan adalah membaca bit-bit tersebut berdasarkan panjang kode yang paling panjang. Berdasarkan tabel 2.6, panjang kode terpanjang adalah 4 dengan kode biner "0001 ". Setelah dilakukan pembacaan bit maka langkah berikutnya adalah melakukan konversi terhadap kode tersebut menjadi bentuk decimal. Kode "0001 " jika direpresentasikan ke dalam bentuk desimal adalah 1 (2^0). Setelah didapat bentuk desimal maka dilakukan pencarian pada tabel 2.6 apakah kode dengan panjang 4 dan representasi desimal 1 terdapat di dalam tabel atau tidak. Pada table 2.6, panjang kode 4 dan representasi decimal 1 merupakan pemetaan dari huruf C. Jika data tersebut tidak ditemukan maka lakukan pergeseran sebanyak 1 bit sehingga kini data menjadi "000" dan akan direpresentasikan sebagai 0. Lakukan pencarian data pada tabel 2.6 sampai data berhasil ditemukan. Proses ini dilakukan sampai semua bit telah berhasil diterjemahkan.

2.6 Algoritma Lempel Ziv Welch (LZW)

Algoritma *Lempel Ziv Welch (LZW)* menggunakan teknik *dictionary* dalam kompresinya. Dimana string karakter digantikan oleh kode tabel yang dibuat setiap ada string yang masuk. Tabel

dibuat untuk referensi masukan string selanjutnya. Algoritma LZW melakukan kompresi dengan mengunakan kode tabel 256 hingga 4095 untuk mengkodekan pasangan *byte* atau *string*. Dengan metode ini banyak string yang dapat dikodekan dengan mengacu pada *string* yang telah muncul sebelumnya dalam teks. Namun, pada penelitian ini digunakan kode tabel 10 hingga 4095 agar tidak terjadi kerancuan pada saat *decoding.*

2.6.1 Proses *Encoding*

Algoritma *encoding* LZW menurut Mark Nelson (1989) dijelaskan pada gambar 2.2 berikut :

```
STRING = get input character
  WHILE there are still input characters DO
      CHARACTER = get input character
      IF STRING+CHARACTER is in the string table THEN
         STRING = STRING+character
      ELSE
         output the code for STRING
         add STRING+CHARACTER to the string table
         STRING = CHARACTER
      END of IF
  END of WHILE
  output the code for STRING
```

Gambar 2.2 Algoritma *Encoding LZW*

Sebagai contoh dari penerapan Algoritma LZW, berikut diberikan sebuah kata "Rain/in/Spain". Isi *dictionary* pada awal proses diisi dengan data kosong, karakter R disimpan pada variable K. Kemudian data berikut yaitu "a" disimpan pada K. Lalu W + K dibandingkan pada tabel, jika tidak ada maka diberikan output yang disimpan oleh W yaitu "R", selanjutnya W + K ditambahkan ke dalam tabel sehingga table berisi "Ra" dan diberi kode. Ulangi hingga karaketer terakhir, seperti pada Tabel 2.8

Tabel 2.8 *Dictionary* LZW

No	W	K	Simbol	Index	Output
1	NULL	R			
2	R	a	Ra	10	R
3	a	i	ai	11	a
4	i	n	in	12	i
5	n	/	n/	13	n

6	/	i	/i	14	/
7	i	n		(12)	
8	n	/	in/	15	12
9	/	S	/S	16	/
10	S	p	Sp	17	S
11	p	a	pa	18	p
12	a	i		(11)	
13	i	n	ain	19	11
14	n	NULL			n

Sehingga didapat output text yang berbentuk "Rain/12/Sp11n". Dari perbandingan input dan output dapat diketahui bahwa terjadi pemampatan yaitu dari text "Rain/in/Spain" yang besarnya ditunjukkan pada tabel 2.9 :

Tabel 2.9 Tabel Perhitungan Tanpa Menggunakan Kompresi LZW

No	Jumlah Huruf	Bit / Karakter	Total
1.	13	7	91

Dimana hasil dari kompresinya adalah "Rain/258/Sp257n" yang besarnya ditunjukkan pada tabel 2.10 :

Tabel 2.10 Tabel Perhitungan Hasil Kompresi Menggunakan LZW

No	Jumlah Huruf	Bit / Karakter	Total
1.	9	7	63
2.	2	9	18
Jumlah			81

Ini adalah contoh sederhana, apabila data berukuran lebih besar dan mempunyai redundansi yang besar juga maka hasil pemampatan juga jauh lebih kecil dan rasio kompresi bertambah besar.

2.6.2 Proses *Decoding*

Algoritma *decoding* LZW menurut Mark Nelson (1989) dijelaskan pada gambar 2.3 berikut :

```
Read OLD_CODE
output OLD_CODE
CHARACTER = OLD_CODE
WHILE there are still input characters DO
   Read NEW_CODE
   IF NEW_CODE is not in the translation table THEN
      STRING = get translation of OLD_CODE
      STRING = STRING+CHARACTER
   ELSE
      STRING = get translation of NEW_CODE
   END of IF
   output STRING
   CHARACTER = first character in STRING
   add OLD_CODE + CHARACTER to the translation table
   OLD_CODE = NEW_CODE
END of WHILE
```

Gambar 2.3 Algoritma *Decoding LZW*

Untuk lebih jelasnya diberikan contoh berdasarkan data hasil *encoding* diatas yaitu "Rain/12/Sp11n". Dari hasil ini akan di *decode* sehingga akan dihasilkan kumpulan karakter seperti semula yaitu "Rain/in/Spain" . Dengan menggunakan algoritma *decoding LZW* diatas maka akan didapat hasil seperti yang terlihat pada tabel 2.11 berikut :

Tabel 2.11 *Decoding LZW*

No	W	K	Simbol	Index	Output
1		R			R
2	R	a	Ra	10	a
3	a	i	ai	11	i
4	i	n	in	12	n
5	n	/	n/	13	/
6	/	12	/i	14	in
7	in	/	in/	15	/
8	/	S	/S	16	S
9	S	p	Sp	17	p
10	p	11	pa	18	ai
11	ai	n	ain	19	n

Dari tabel 2.11 terlihat bahwa hasil *decoding* dengan algoritma LZW menghasilkan nilai keluaran yang sama dengan nilai semula yaitu "Rain/in/Spain" dan simbol-simbol yang terdapat pada

kamus hasil dekoding mempunyai nilai yang sama dengan yang terdapat pada kamus hasil *encoding.*

2.7 Rasio Kompresi

Rasio kompresi adalah perbandingan ukuran file teks sebelum di kompresi dan sesudah di kompresi (Sayood, K. 2000). Untuk menghitung perbandingannya, bisa dihitung dengan rumus berikut :

$$\text{Rasio} = \frac{\text{Ukuran file teks sebelum dikompresi}}{\text{Ukuran file teks setelah dikompresi}}$$

$$\text{\% Kompresi} = \frac{(A-K)}{A} \times 100\%$$

Keterangan :

A = Ukuran file teks asli
K = Ukuran file teks kompresi

BAB III
METODOLOGI DAN PERANCANGAN

Pada bab metodologi dan perancangan ini akan dibahas metode perancangan yang digunakan dan langkah – langkah yang dilakukan untuk melakukan kompresi SMS dengan menggunakan algoritma *Lempel Ziv Welch (LZW)* dan *Huffman Kanonik*.

Langkah – langkah yang akan dilakukan adalah sebagai berikut:

1. Mempelajari literatur yang terkait dengan kompresi SMS yang telah dibahas pada bab II.
2. Menganalisa perangkat lunak dan metode yang digunakan pada penelitian sebelumnya, kemudian merancang sistem untuk dikembangkan guna memperbaiki kekurangan pada metode tersebut.
3. Melakukan perancangan sistem kompresi SMS dan mengimplementasikan menjadi perangkat lunak.
4. Melakukan uji coba sistem dengan mengirim dan menerima pesan SMS yang telah dikompresi melalui sistem.
5. Mengevaluasi tingkat keberhasilan sistem dalam melakukan kompresi SMS

Langkah-langkah yang dilakukan dapat dilihat pada Gambar 3.1:

Gambar 3.1 Langkah – langkah yang dilakukan.

3.1 Analisa Sistem

Pada subbab analisis sistem ini akan dijelaskan mengenai deksripsi sistem, dan batasan sistem.

3.1.1 Deskripsi Sistem

Sistem yang akan dibangun berupa aplikasi SMS pada *Handphone* yang dapat melakukan kompresi SMS, mengirim SMS, menerima SMS dan kemudian men-*dekompres*-nya untuk mengembalikan file seperti semula. Proses kompresi dilakukan sebelum SMS dikirim, yang bertujuan agar *size* dari SMS dapat lebih kecil sehingga mengurangi biaya pengiriman SMS. Perangkat lunak yang dibangun ditujukan bagi pengguna telepon genggam khususnya pengguna SMS.

Ketika *user* memasuki perangkat lunak maka proses yang terjadi adalah :

- *User* memasukkan beberapa huruf ASCII atau angka non negatif, dimana terdapat beberapa huruf yang sama secara berurutan.
- *User* dapat menentukan atau memillih akan melakukan pengiriman SMS dengan menggunakan kompresi atau dengan pengiriman biasa (tanpa kompresi).
- Jika *user* memilih dengan menggunakan kompresi, maka kompresi diawali dengan menggunakan *encoding Lempel Ziv Welch (LZW)* dilanjutkan dengan *encoding Huffman Kanonik.*
- Kemudian oleh perangkat lunak, teks tersebut akan diproses untuk dikompresi atau didekompresi.
- Proses yang akan digunakan seperti yang telah dibahas dalam bab 2.

3.1.2 Batasan Sistem

Batasan sistem yang akan dikembangkan adalah:

1. Karakter yang akan dikompresi adalah karakter-karakter umum yang digunakan untuk SMS.
2. Analisa pemakaian karakter pada SMS memakai bahasa sehari-hari dalam bahasa Indonesia
3. Jumlah tabel *Huffman Kanonik* yang dibuat sebatas satu tabel.

Langkah-langkah yang dilakukan sistem secara umum dijelaskan pada Gambar 3.2 :

Gambar 3.2 Gambaran Umum Sistem

3.2 Perancangan Tabel Kode *Huffman Kanonik*

3.2.1 Analisa Pemakaian Karakter

SMS menggunakan kode ASCII 7 bit yang mempunyai variasi karakter sebanyak 128, sedangkan untuk karakter-karakter yang dipakai dalam SMS sebanyak 95, oleh karena itu dalam pembentukan *Huffman tree* yang akan dilakukan memakai 95 karakter tersebut, yang terdiri dari :

1. 26 kode untuk huruf kapital *(upper case)* dari A –Z.
2. 26 kode untuk huruf kecil *(lower case)* dari a –z.
3. 10 digit desimal dari 0 – 9.
4. 34 karakter tanda baca.

Karena tidak semua karakter umum dipakai dalam penulisan SMS. Pemakaian ke 95 karakter dalam penggunaan SMS dapat dianalisis. Analisa pemakaian karakter pada SMS diasumsikan sebagai berikut :

1. Prioritas pertama adalah huruf kecil (a-z), karena hampir selalu digunakan dalam setiap penulisan SMS.
2. Prioritas kedua adalah tanda baca yang sering muncul dalam SMS, seperti : ” ”(spasi). ”.”(titik) dan ”,”(koma).
3. Prioritas ketiga adalah, angka (0-9), dan huruf besar (A-Z), karena jika dilihat dari teks SMS, penggunaan huruf besar hanya terdapat di awal kalimat atau setelah titik.
4. Prioritas terakhir adalah tanda baca yang sangat jarang terpakai dalam penulisan SMS, seperti : ”#”, ”~”, ”^”.

3.2.2 Analisa Penggunaan Algoritma Lempel Ziv-Welch

Algoritma ini melakukan kompresi dengan menggunakan kamus, di mana fragmen-fragmen teks digantikan dengan indeks yang diperoleh dari sebuah "kamus". Pendekatan ini bersifat adaptif dan efektif karena banyak karakter dapat dikodekan dengan mengacu pada string yang telah muncul sebelumnya dalam teks. Prinsip kompresi tercapai jika referensi dalam bentuk pointer dapat disimpan dalam jumlah bit yang lebih sedikit dibandingkan string aslinya.

3.2.3 Perancangan Tabel *Huffman Kanonik*

Pada penelitian Arya Wibisana (2008) sebelumnya, telah dibentuk tabel-tabel Huffman biasa berdasarkan *Huffman Tree* yang dibentuk atas hasil analisa pemakaian huruf pada SMS yang mengacu pada dasar teori yang telah ditulis pada bab 2. Dari kode-kode Huffman biasa tersebut, akan dilakukan transformasi ke kode *Huffman Kanonik*. Kode-kode *Huffman* "biasa" tersebut dibagi menjadi empat bagian sesuai dengan analisa karakter-karakternya :

1. Awalan 0, untuk karakter-karakter dengan prioritas utama, karena pada awalan 0, kompresi bit per karakter paling banyak dilakukan. Sesuai dengan prioritas yang telah dianalisis sebelumnya, awalan 0 ini ditempatkan pada huruf kecil yang sering sekali dipakai pada penulisan SMS.
2. Awalan 100, untuk karakter-karakter dengan prioritas sedang, dengan kata lain kemungkinan pemakaian karakter-karakter ini ada, tetapi tidak sering digunakan. Dari awalan 100 akan dibentuk 16 kode karakter yang terdiri dari :
   1) 10 angka (dari 0 sampai 9)
   2) 6 tanda baca yang sering dipakai dalam penulisan.
      yaitu: "(", ")", "?", "!", "-", ".".
3. Awalan 101, untuk huruf-huruf besar dari A sampai Z.
4. Awalan 110, untuk karakter-karakter sisa yang kesemuanya merupakan tanda baca.

3.2.3.1 Pembentukan Tabel *Huffman Kanonik*

Pada penelitian Arya Wibisana (2008), pembentukan tabel *Huffman* dengan awalan 0 dilakukan sesuai dengan prioritas penggunaan huruf mengacu pada teori di bab 2 ditunjukkan pada tabel 3.1 :

Tabel 3.1 Tabel *Huffman* I awalan 0

Pembentukan Tabel *Huffman* dengan awalan 100 digunakan untuk angka dan beberapa tanda baca sesuai dengan prioritas penggunaannya ditunjukkan pada tabel 3.2 :

Karakter	Awalan	Badan	Kode
a	0	0010	00010
b		10000	010000
c		111010	0111010
d		10001	010001
e		0010	00101
f		111011	0111011
g		10010	010010
h		10011	010011
i		0011	00011
j		10100	010100
k		10101	010101
l		10110	010110
m		10111	010111
n		0110	00110
o		0100	00100
p		11000	011000
q		111100	0111100
r		11001	011001
s		0111	00111
t		11010	011010
u		0101	00101
v		111101	0111101
w		11011	011011
x		111110	0111110
y		11100	011100
z		111111	0111111
spasi		0000	00000

Tabel 3.2 Tabel *Huffman* I awalan 100

0	100	0000	1000000
1		0001	1000001
2		0010	1000010
3		0011	1000011
4		0100	1000100
5		0101	1000101
6		0110	1000110
7		0111	1000111
8		1000	1001000
9		1001	1001001
(		1010	1001010
)		1011	1001011
?		1100	1001100
!		1101	1001101
-		1110	1001110
:		1111	1001111

Awalan 101 ditempatkan untuk huruf – huruf besar dari A sampai Z, hasil pembentukan tabel *Huffman* dengan awalan 101 ditunjukkan pada tabel 3.3 :

Tabel 3.3 Tabel *Huffman* I awalan 101

Karakter	Awalan	Badan	Kode
A	101	0001	101000
B		10000	10110000
C		111010	101111010
D		10001	10110001
E		0001	1010001
F		111011	101111011
G		10010	10110010
H		10011	10110011
I		0011	1010011
J		10100	10110100
K		10101	10110101
L		10110	10110110
M		10111	10110111

N		0110	1010110
O		0100	1010100
P		11000	10111000
Q		111100	101111100
R		11001	10111001
S		0111	1010111
T		11010	10111010
U		0101	1010101
V		111101	101111101
W		11011	10111011
X		111110	101111110
Y		11100	10111100
Z		111111	101111111
Karakter Lain		0000	1010000

Awalan 110 ditempatkan untuk karakter-karakter sisa yang kesemuanya merupakan tanda baca. Hasil pembentukan tabel *Huffman* dengan awalan 101 ditunjukkan pada tabel 3.4 :

Tabel 3.4 Tabel Huffman I awalan 110

Karakter	Awalan	Badan	Kode
.	110	000	110000
,		001	110001
"		01000	11001000
#		01001	11001001
$		01010	11001010
%		01011	11001011
&		01100	11001100
'		01101	11001101
*		01110	11001110
+		01111	11001111
/		10000	11010000
:		10001	11010001
≤		10010	11010010
=		10011	11010011
≥		10100	11010100

@		10101	11010101
[		10110	11010110
\		10111	11010111
]		11000	11011000
^		11001	11011001
_		11010	11011010
`		11011	11011011
{		11100	11011100
|		11101	11011101
}		11110	11011110
~		11111	11011111

Setelah tabel – tabel kode *Huffman* telah terbentuk, maka langkah selanjutnya adalah mentransformasikan kode *Huffman* tersebut ke kode *Huffman Kanonik* sesuai dengan teori yang telah dibahas pada bab 2. Hasil dari transformasi tersebut ditunjukkan pada tabel 3.5 :

Tabel 3.5 Hasil Transformasi Kode Huffman ke Kode *Huffman Kanonik*

Karakter	Kode Huffman	Kode *Huffman Kanonik*	Angka	Panjang Kode
C	101111010	000000000	0	9
F	101111011	000000001	1	9
Q	101111100	000000010	2	9
V	101111101	000000011	3	9
X	101111110	000000100	4	9
Z	101111111	000000101	5	9
"	11001000	00000011	3	8
#	11001001	00000100	4	8
$	11001010	00000101	5	8
%	11001011	00000110	6	8
&	11001100	00000111	7	8
'	11001101	00001000	8	8
*	11001110	00001001	9	8
+	11001111	00001010	10	8
/	11010000	00001011	11	8

;	11010001	00001100	12	8
≤	11010010	00001101	13	8
=	11010011	00001110	14	8
≥	11010100	00001111	15	8
@	11010101	00010000	16	8
B	10110000	00010001	17	8
D	10110001	00010010	18	8
G	10110010	00010011	19	8
H	10110011	00010100	20	8
J	10110100	00010101	21	8
K	10110101	00010110	22	8
L	10110110	00010111	23	8
M	10110111	00011000	24	8
P	10111000	00011001	25	8
R	10111001	00011010	26	8
T	10111010	00011011	27	8
W	10111011	00011100	28	8
Y	10111100	00011101	29	8
(	1001010	0001111	15	7
)	1001011	0010000	16	7
?	1001100	0010001	17	7
!	1001101	0010010	18	7
-	1001110	0010011	19	7
:	1001111	0010100	20	7
0	1000000	0010101	21	7
1	1000001	0010110	22	7
2	1000010	0010111	23	7
3	1000011	0011000	24	7
4	1000100	0011001	25	7
5	1000101	0011010	26	7
6	1000110	0011011	27	7
7	1000111	0011100	28	7
8	1001000	0011101	29	7
9	1001001	0011110	30	7
A	1010001	0011111	31	7
E	1010001	0100000	32	7
I	1010011	0100001	33	7
N	1010110	0100010	34	7

O	1010100	0100011	35	7
S	1010111	0100100	36	7
U	1010101	0100101	37	7
enter	1010000	0100110	38	7
c	0111010	0100111	39	7
f	0111011	0101000	40	7
q	0111100	0101001	41	7
v	0111101	0101010	42	7
x	0111110	0101011	43	7
z	0111111	0101100	44	7
.	110000	010111	23	6
,	110001	011000	24	6
b	010000	011001	25	6
d	010001	011010	26	6
g	010010	011011	27	6
h	010011	011100	28	6
j	010100	011101	29	6
k	010101	011110	30	6
l	010110	011111	31	6
m	010111	100000	32	6
p	011000	100001	33	6
r	011001	100010	34	6
t	011010	100011	35	6
w	011011	100100	36	6
y	011100	100101	37	6
spasi	00000	10011	19	5
a	00010	10100	20	5
e	00101	10101	21	5
i	00011	10110	22	5
n	00110	10111	23	5
o	00100	11000	24	5
s	00111	11001	25	5
u	00101	11010	26	5

3.3 Perancangan Sistem

Pada subbab ini akan dijelaskan mengenai berbagai proses yang terjadi dalam membangun sistem kompresi SMS, antara lain

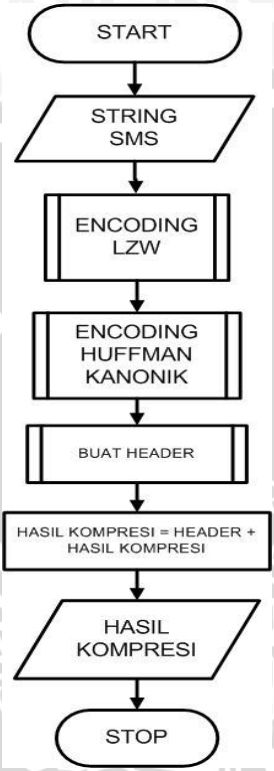
membahas tentang proses kompresi, proses dekompresi beserta *flowchart*-nya.

3.3.1 Perancangan Proses Kompresi

Proses kompresi dilakukan sebelum pesan SMS akan dikirim. Setelah pengguna SMS selesai menulis pesan, pesan akan dihitung dengan memakai tabel yang telah dibentuk sebelumnya. Langkah-langkah yang dilakukan adalah sebagai berikut:

1. Input *string* teks SMS
2. Inisialisasi panjang teks SMS
3. *Encoding* diawali dengan melakukan kompresi menggunakan *LZW* kemudian hasil kompresinya digunakan untuk kompresi dengan menggunakan *Huffman Kanonik.*
4. Hasil *encoding* gabungan tersebut kemudian diberikan *header* berupa 3 bit pertama sebagai penyimpan nilai banyaknya '0' yang ditambahkan di akhir pesan, 10 bit berikutnya merupakan panjang biner indeks LZW, dan byte berikutnya merupakan indeks LZW.
5. Output hasil kompresi.

Proses kompresi teks SMS secara keseluruhan bisa dilihat pada Gambar 3.3 :



Gambar 3.3 *Flowchart* Proses Kompresi

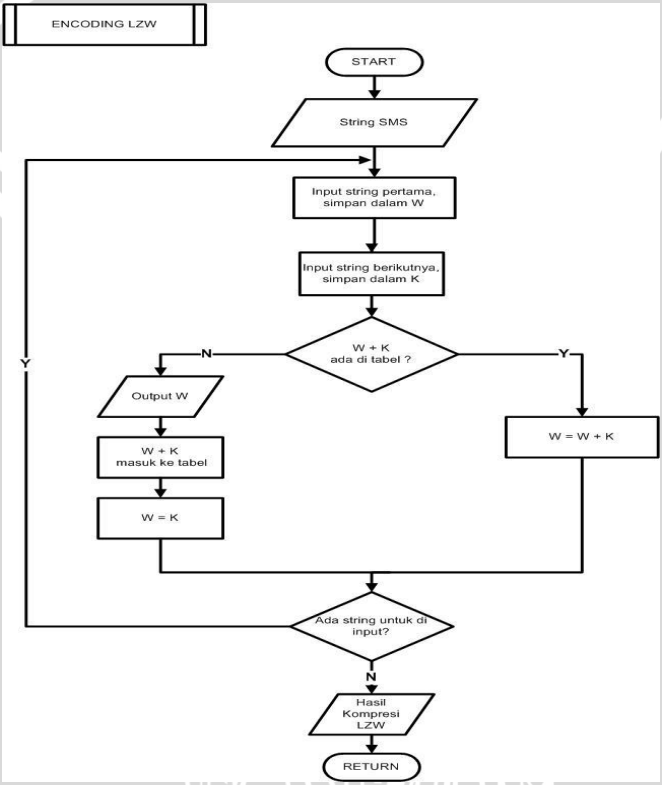
3.3.1.1 Algoritma Encoding Lempel Ziv Welch (LZW)

Berikut adalah penjelasan proses *encoding* algoritma *LZW* :

1. Input string SMS yang akan dikompresi
2. W ← karakter pertama dalam *stream* karakter.
3. K ← karakter berikutnya dalam *stream* karakter.
4. Periksa apakah string (W+K) terdapat dalam *dictionary* ?
   - Jika ya, maka W ← W+K menjadi string baru.
   - Jika tidak, maka :
     - *Output* sebuah kode untuk menggantikan *string* W.
     - Tambahkan *string* (W+K) ke dalam *dictionary* dan berikan nomor / kode berikutnya yang belum digunakan dalam *dictionary* untuk *string* tersebut.
     - W ← K
5. Apakah masih ada karakter berikutnya dalam *stream* karakter ?

- Jika ya, kembali ke langkah 2.
- Jika tidak, maka *output* kode yang menggantikan *string* W, lalu terminasi proses (stop).

Proses *encoding* dengan menggunakan algoritma *Lempel Ziv Welch (LZW)* secara keseluruhan dijelaskan pada Gambar 3.4 :



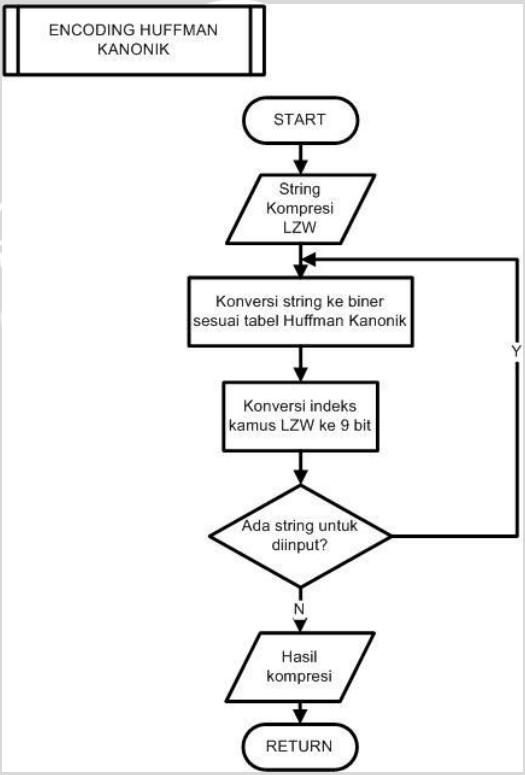
Gambar 3.4 *Flowchart* Proses *Encoding* LZW

3.3.1.2 Algoritma Encoding *Huffman Kanonik*

Berikut adalah penjelasan proses *encoding Huffman Kanonik* :

1. Input data *string* kompresi LZW.
2. Lakukan konversi hasil *encoding* LZW ke bentuk biner sesuai dengan tabel *Huffman Kanonik*.
3. Untuk indeks kamus *LZW*, konversikan ke bentuk 9 bit.
4. Semua dilakukan hingga semua karakter berhasil di*encoding*.
5. Output hasil_kompresi yang telah dilakukan.

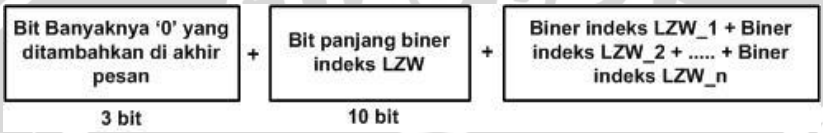
Proses *encoding* dengan menggunakan algoritma *Huffman Kanonik* seluruhnya dapat dilihat pada gambar 3.5 :



Gambar 3.5 *Flowchart* Proses *Encoding Huffman Kanonik*

3.3.1.3 Proses Pembuatan *Header*

Pembuatan *header* digunakan saat proses dekompresi, dengan tujuan untuk menginisialisasi panjang *byte* utama dan indeks *LZW* yang dipakai dalam proses kompresi. Header diawali dengan 3 bit pertama sebagai penyimpan nilai banyaknya '0' yang ditambahkan di akhir pesan, 10 bit panjang biner indeks LZW dan indeks kamus *LZW* yang telah dicari pada proses kompresi. Struktur dari header dapat dilihat pada Gambar 3.6 berikut :



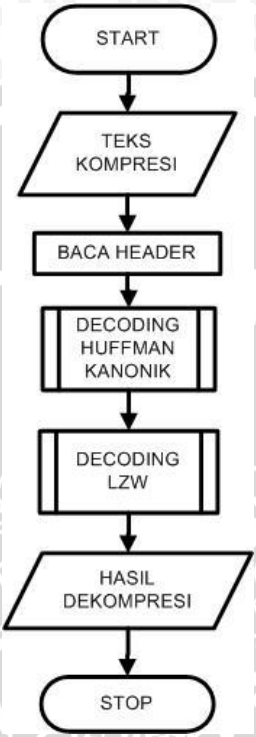
Gambar 3.6 Struktur *header*

3.3.2 Perancangan Proses Penguraian Teks (Decoding)

Langkah-langkah yang dilakukan adalah sebagai berikut:

1. Input *string* teks kompresi
2. Lakukan pembacaan *header* pesan kompresi.
3. Lakukan proses *decoding* dengan menggunakan *Huffman Kanonik* terlebih dahulu.
4. Pada proses *decoding* dengan *Huffman Kanonik*, lakukan pencocokan dengan tabel *Huffman Kanonik* hingga karakter terakhir.
5. Setelah selesai melakukan *decoding* dengan menggunakan *Huffman Kanonik*, lanjutkan *decoding* dengan menggunakan *LZW*.
6. Simpan hasil output

Proses penguraian teks secara keseluruhan bisa dilihat pada Gambar 3.7 :



Gambar 3.7 *Flowchart* Proses *Decoding*

3.3.2.1 Proses Pembacaan *Header*

Berikut ini merupakan penjelasan dari proses pembacaan *header:*

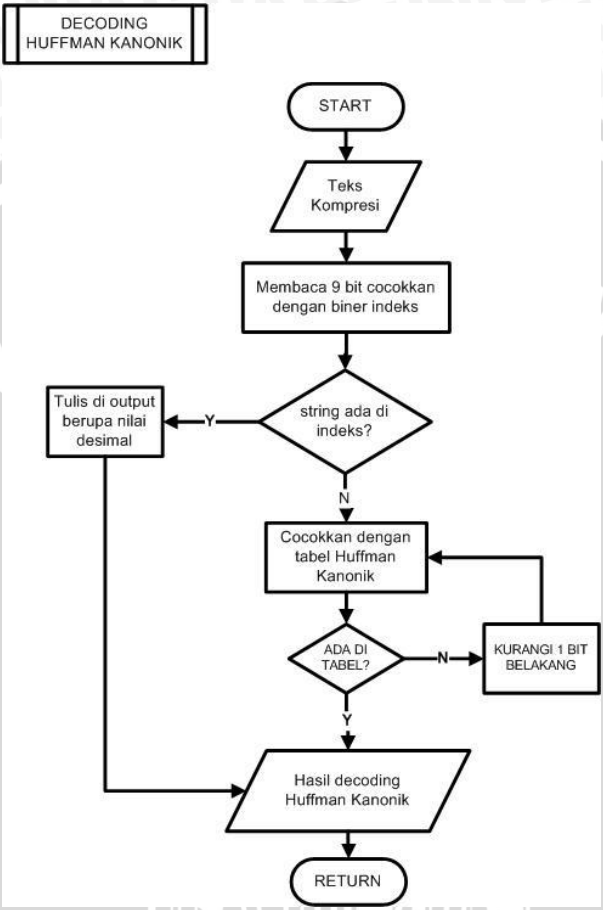
1. Lakukan pembacaan 3 bit pertama yang merupakan bit penyimpan nilai banyaknya '0' yang ditambahkan di akhir pesan
2. Kemudian baca 10 bit berikutnya, yang merupakan nilai dari panjang biner indeks LZW. Byte berikutnya merupakan byte biner indeks LZW.
3. Pisahkan dengan *byte* berikutnya yang merupakan *byte* pesan, agar tidak mengganggu proses *decoding*.

3.3.2.2 Algoritma Decoding *Huffman Kanonik*

Berikut ini merupakan penjelasan dari proses *decoding* menggunakan algoritma *Huffman Kanonik* :

1. Input teks SMS yang akan di dekompresi.
2. Lakukan pembacaan 9 bit pertama, lalu cocokkan dengan biner indeks.
3. Jika kode ada di indeks maka tulis di *output* berupa nilai desimal dari biner tersebut.
4. Jika tidak ada di indeks, lakukan pencocokan dengan tabel *Huffman Kanonik* yang sudah dibentuk.
5. Jika kode tidak ada maka lakukan pembacaan bit kembali, dengan mengurangi panjang kode sebanyak 1 bit dan cocokkan dengan tabel.
6. Lakukan proses diatas hingga semua kode didekompresi.

Proses *decoding* dengan menggunakan algoritma *Huffman Kanonik* secara keseluruhan dapat dilihat pada gambar 3.8 sebagai berikut:



Gambar 3.8 *Flowchart* Proses *Decoding Huffman kanonik*.

3.3.2.2 Algoritma Decoding Lempel Ziv Welch (LZW)

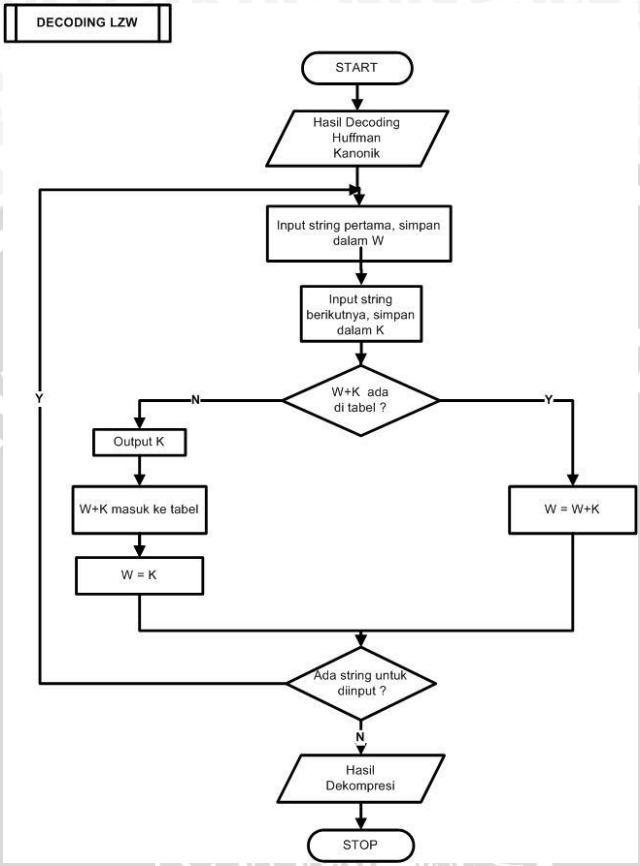
Berikut adalah penjelasan proses *decoding* algoritma LZW :

1. *Input* hasil *decoding* Huffman Kanonik.
2. *W* ← karakter pertama dari *stream* kode.
3. *K* ← karakter kedua dari *stream* kode.
4. Periksa apakah string (W+K) terdapat dalam *dictionary* ?
   - Jika ya, maka W ← W+K menjadi string baru.
   - Jika tidak, maka :
   - *Output* sebuah kode untuk menggantikan *string* K.

- Tambahkan *string* (W+K) ke dalam *dictionary* dan berikan nomor / kode berikutnya yang belum digunakan dalam *dictionary* untuk *string* tersebut.
- W ← K
- Jika K berupa indeks, maka W+K = W+W sesuai kode indeks
- Jika W berupa indeks, maka W=W+K sesuai indeks.

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

Proses *decoding* dengan menggunakan algoritma *Lempel Ziv Welch (LZW)* secara keseluruhan dijelaskan pada Gambar 3.9 :



Gambar 3.7 *Flowchart* Proses *Decoding* LZW

3.4 Perancangan Pengujian

Pengujian dilakukan untuk mengetahui seberapa jauh pengaruh proses kompresi terhadap jumlah karakter dan jumlah halaman SMS yang dihasilkan. Pengujian yang dilakukan dalam sistem ini ditinjau dari tingkat kompresi dan tingkat akurasi dalam melakukan kompresi teks dengan menggunakan algoritma gabungan algoritma *Lempel Ziv Welch (LZW)* dan *Huffman Kanonik.*

3.4.1 Data dan Bahan Pengujian

Pengujian dilakukan pada 20 teks SMS yang mempunyai jumlah karakter lebih dari 160 karakter (1 halaman) dan variasi pemakaian kata yang berbeda, dari ke 20 teks SMS tersebut akan dibagi menjadi 4 sesuai dengan jumlah karakter dan variasi kata. Pembagiannya adalah sebagai berikut :

1. SMS 1-5 jumlah karakter 1-160 (1 halaman SMS)
2. SMS 6-10 jumlah karakter 161-305 (2 halaman SMS)
3. SMS 11-15 jumlah karakter 306- 457 (3 halaman SMS)

3.4.2 Pengujian Rasio Kompresi

Pengujian yang akan dilakukan adalah dengan menggunakan algoritma *Huffman Kanonik*, algoritma *LZW*, serta penggabungan kedua algoritma tersebut. Dimana tiap algoritma tersebut memiliki cara perhitungan yang berbeda.

Tabel yang digunakan untuk pengujian rasio kompresi dengan masing–masing metode ditunjukkan seperti pada tabel di bawah ini :

Tabel 3.6 Tabel biner *Huffman Kanonik*

Huruf	Frekuensi	Kode Huffman Kanonik	Panjang Kode	Total

Tabel 3.7 Tabel perhitungan *Huffman Kanonik*

Jumlah Huruf	7 Bit / Huruf	Ukuran

Tabel 3.8 Tabel *Dictionary LZW*

W	K	Simbol	Index	Output

Setelah melakukan kompresi teks dengan menggunakan gabungan kedua algoritma maka prosentase dari pengkompresian dengan menggunakan algoritma tersebut adalah :

$$\% \text{ Kompresi} = \frac{(A-K)}{A} \times 100\%$$

Keterangan :
A = Ukuran file teks asli
K = Ukuran file teks kompresi

3.5 Contoh Perhitungan

Diketahui terdapat teks SMS dengan panjang 49, seperti ditunjukkan pada gambar di bawah ini :

Spongebob adalah sahabat anak-anak sepanjang masa

Untuk melakukan kompresi dengan menggunakan algoritma *Lempel Ziv Welch (LZW),* dibuat suatu *dictionary* seperti pada tabel 3.9 sebagai berikut :

Tabel 3.9 Dictionary LZW

W	K	Simbol	Indeks	Output
	S			
S	p	sp	10	S
p	o	po	11	p
o	n	on	12	o
n	g	ng	13	n
g	e	ge	14	g
e	b	eb	15	e
b	o	bo	16	b
o	b	ob	17	o
b	spasi	b spasi	18	b
spasi	a	spasi a	19	spasi
a	d	ad	20	a
d	a	da	21	d
a	l	al	22	a
l	a	la	23	l
a	h	ah	24	a
h	spasi	h spasi	25	h
spasi	s	spasi s	26	Spasi
s	a	sa	27	s
a	h	ah	(24)	

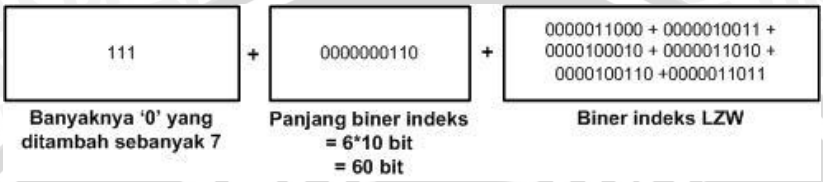
ah	a	aha	28	24
a	b	ab	29	a
b	a	ba	30	b
a	t	at	31	a
t	spasi	t spasi	32	t
spasi	a	spasi a	(19)	
spasi a	n	spasi an	33	19
n	a	na	34	n
a	k	ak	35	a
k	-	k-	36	k
-	a	-a	37	-
a	n	an	38	a
n	a	na	(34)	
na	k	nak	39	34
k	spasi	k spasi	40	k
spasi	s	spasi s	(26)	
spasi s	e	spasi se	41	26
e	p	ep	42	e
p	a	pa	43	p
a	n	an	(38)	
an	j	anj	44	38
j	a	ja	45	j
a	n	an	(38)	
an	g	ang	46	38
g	spasi	g spasi	47	g
spasi	m	spasi m	48	spasi
m	a	ma	49	m
a	s	as	50	a
s	a	sa	(27)	
a				27

Dari kompresi diatas, didapat hasil ouput teks:

“Spongebob adalah s24abat19nak-a34k26ep38j38g ma27”

Tahap berikutnya yaitu mengkonversi hasil *encoding* LZW tersebut ke bentuk *Huffman Kanonik* sesuai dengan tabel yang sudah dibentuk.. Indeks dari kamus LZW dikonversi ke bentuk biner 10 bit, hasil dari *encoding* tersebut kemudian ditambahkan *header*. Untuk

menginisialisasi panjang biner utama dan biner indeks pada proses dekompresi maka akan dibuat *header*, *header* pada contoh digambarkan pada Gambar 3.9 :



Gambar 3.10 *Header*

Teks terdiri dari 49 karakter dengan panjang masing-masing karakter adalah 7 bit. Panjang teks biner yang dihasilkan sebanyak 343 bit. Setelah dilakukan kompresi, panjang biner pada setiap karakter akan berbeda, dari ke-49 karakter, dihasilkan panjang teks biner sebanyak 264 bit ditambah panjang header 73 bit menjadi total 337 bit. Hasil teks biner yang sudah dikompresi ditunjukkan pada Gambar 3.11 :

```
1110000000110000001100000000100110000100010000
0000110100000100110000001101101001001000011100
0001011101101110101011001110000110011001110100
0001101010100011111101000111001001111001000000
0110001010001100110100100011000001001110111100
0100011110001001110100000010001001111000000010
1010101011000010000100110011101000010011001100
0111001110000010100000001101100000000
```

Dari contoh didapatkan data sebagai berikut: terdapat teks ASCII awal 49 karakter dengan panjang bit sebesar 343 bit dan panjang teks akhir sebanyak 43 karakter dengan panjang bit sebesar 337 bit maka didapatkan rasio :

% Kompresi (bit) = ((343-337)/343)* 100%
= 1,74 %

% Kompresi (karakter) = ((49-43)/49)*100%
= 12,24%

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

4.1 Lingkungan Implementasi

Lingkungan implementasi yang akan dijelaskan dalam subbab ini adalah lingkungan implementasi perangkat keras dan perangkat lunak.

4.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem kompresi SMS adalah sebagai berikut :

- Handphone
  1. *Support* Java.
  2. *Support Mobile Information Device Profile* 2.0 (MIDP 2.0)
  3. 64 Mb *Phone Memory*.
- Personal Computer
  1. Prosesor Intel Core 2 Duo 2,0 Ghz
  2. Memori 2048 Mb
  3. Harddisk dengan kapasitas 250 GB
  4. Monitor 14,1”
  5. Keyboard
  6. Mouse

4.1.2 Lingkungan Perangkat Lunak

Lingkungan perangkat lunak yang digunakan dalam pengembangan sistem kompresi SMS ini adalah dengan menggunakan :

1. Sistem operasi Windows 7 32 bit.
2. Java (TM) SE Runtime Environment 6
3. Java Micro Edition SDK 3.0
4. Netbeans IDE 6.7.1
5. Microsoft Excel.

4.2 Implementasi Program

Berdasarkan analisa dan perancangan proses yang terdapat pada subbab 3.3, maka pada subbab ini akan dijelaskan implementasi proses-proses tersebut.

4.2.1 Implementasi Kompresi / *Encoding*

Sesuai pada subbab 3.3, urutan proses kompresi yang akan dilakukan yaitu *encoding* dengan menggunakan Algoritma *Lempel Ziv Welch (LZW)*, kemudian dilanjutkan dengan *encoding* menggunakan Algoritma *Huffman kanonik*.

4.2.1.1 *Encoding Lempel Ziv Welch (LZW)*

Dalam melakukan kompresi menggunakan algoritma *Lempel Ziv Welch (LZW)*, langkah untuk melakukan *encoding* seperti pada Gambar 4.1

```
public DictionaryLZW(String sourceTeks) {
    this.sourceTeks = sourceTeks;

    tabelLZW = new ArrayList<structDictLZW>();
    int ind = 10;
    for (int i = 1; i < sourceTeks.length(); i++) {
      if (tabelLZW.isEmpty()) {
        structDictLZW rowLZW = new structDictLZW();
        rowLZW.W = Character.toString(sourceTeks.charAt(0));
        rowLZW.K = Character.toString(sourceTeks.charAt(1));
        rowLZW.Symbol = rowLZW.W + rowLZW.K;
        rowLZW.Index = 10;
        rowLZW.output = rowLZW.W;

        tabelLZW.add(rowLZW);
        } else {
        boolean found = false;

        structDictLZW rowLZW = new structDictLZW();

        rowLZW.K = Character.toString(sourceTeks.charAt(i));

        String bufPost = tabelLZW.get(tabelLZW.size() - 1).Symbol;
        int indexPost = findSymbol(bufPost);
        if (indexPost >= 0) {
          rowLZW.W = bufPost;
          rowLZW.Symbol = rowLZW.W + rowLZW.K;
         int indexCur = findSymbol(rowLZW.Symbol);
          if (indexCur >= 0) {
            rowLZW.Index = tabelLZW.get(indexCur).Index;
            rowLZW.output = "";
          } else {
            ind++;
            rowLZW.Index = ind;
            rowLZW.output = tabelLZW.get(tabelLZW.size() - 1).Index;
```

Gambar 4.1 *Encoding Lempel Ziv Welch (LZW)*

```
else {
                rowLZW.W = Character.toString(sourceTeks.charAt(i - 1));
                rowLZW.Symbol = rowLZW.W + rowLZW.K;

                int indexCur = findSymbol(rowLZW.Symbol);
                if (indexCur >= 0) {
                   rowLZW.Index = tabelLZW.get(indexCur).Index;
                   rowLZW.output = "";
                } else {
                   ind++;
                   rowLZW.Index = ind;
                   rowLZW.output = rowLZW.W;
                }
             }
             tabelLZW.add(rowLZW);
          }
       }
       structDictLZW rowLZW = new structDictLZW();
       rowLZW.K = "";
       rowLZW.W = Character.toString(sourceTeks.charAt(sourceTeks.length() - 1));
       rowLZW.Symbol = "";
       rowLZW.Index = 0;
       if (tabelLZW.get(tabelLZW.size() - 1).output.toString().equals("")) {
          rowLZW.output = tabelLZW.get(tabelLZW.size() - 1).Index;
       }
       else{
          rowLZW.output=rowLZW.W;
       }
             tabelLZW.add(rowLZW);
    }

private int findSymbol(String str) {
       int index = 0;

       boolean found = false;
       for (int i = 0; i < tabelLZW.size() - 1; i++) {
          if (str.equals(tabelLZW.get(i).Symbol)) {
             index = i;
             found = true;
             break;
          }
       }
       if (!found) {
          index = -1;
       }
       return index;
    }
```

Gambar 4.1 *Encoding Lempel Ziv Welch (LZW) (lanjutan)*

```
public void printTable() {
    System.out.println("Panjang tabel = " + tabelLZW.size());
    for (int i = 0; i < tabelLZW.size(); i++) {
       System.out.println(tabelLZW.get(i).W + " " + tabelLZW.get(i).K + " "
            + tabelLZW.get(i).Symbol + " " + tabelLZW.get(i).Index + " "
            + tabelLZW.get(i).output);
    }
    System.out.println(" output teks");
    for (int j = 0; j < tabelLZW.size(); j++) {
       System.out.print(tabelLZW.get(j).output.toString());
    }
    System.out.println("");
  }
```

Gambar 4.1 *Encoding Lempel Ziv Welch (LZW) (lanjutan)*

4.2.1.2 *Encoding Huffman Kanonik*

Pada tahap ini, dari hasil *encoding* menggunakan metode *LZW* kemudian dilanjutkan melakukan *encoding* menggunakan *Huffman Kanonik*, langkah untuk melakukan kompresi / *encoding* seperti pada Gambar 4.2

```
public void lzw2huff() {
    System.out.println("LZW to Huff");
    ascii ascu = new ascii();
    tableHuffman TBH = new tableHuffman();
    int sisa, mod, bin = 0;

    String buf[] = new String[9];
    StringBuffer result = new StringBuffer();
    boolean loop = true;
    ArrayList<Object> temp = new ArrayList<Object>();
ArrayList<Object> binerIndek = new ArrayList<Object>();
    ArrayList<String> bufStr = new ArrayList<String>();
    for (int a = 0; a < tabelLZW.size(); a++) {
       if (tabelLZW.get(a).output.toString().equals("")) {
       } else {
          try {
             bufStr.add(tabelLZW.get(a).output.toString());
          } catch (Exception exc) {
             System.out.println(exc);
          }
       }
    }
```

Gambar 4.2 *Encoding Huffman Kanonik*

```
boolean ketemu = false;
      boolean kembar;
      String hufman[] = new String[bufStr.size()];
      String huff2bin[] = new String[bufStr.size()];
      ArrayList<Integer> cekInpDec = new ArrayList<Integer>();

      for (int i = 0; i < bufStr.size(); i++) {
         hufman[i] = bufStr.get(i);
      }
      for (int d = 0; d < hufman.length; d++) {
         result = new StringBuffer();
         ketemu = false;
         for (int e = 0; e < TBH.karakter.length; e++) {
            if (hufman[d].equals(Character.toString(TBH.karakter[e]))) { // ada di tabel
ambil dari tabel
               ketemu = true;
               huff2bin[d] = TBH.hufman_kanonik[e];
            }
         }
         if (ketemu == false) {
            String binary = "";
            String jd9 = "";
            int inpDec = Integer.parseInt(hufman[d]);
            binary = Integer.toBinaryString(inpDec);
            if (binary.length() < 9) {
               for (int ku = binary.length(); ku < 9; ku++) {
                  jd9 = jd9 + 0;
               }
            }
            binary = jd9 + binary;
            huff2bin[d] = binary;
            if (binerIndek.isEmpty()) {
               binerIndek.add(binary);
               System.out.println("masih kosong,isi aja bro");
            } else {
               kembar = false;
               for (int y = 0; y < binerIndek.size(); y++) {
                  String bufff = (String) binerIndek.get(y);
                  if (bufff.equals(binary)) {
                     kembar = true;
                  }
               }
               if (kembar == false) {
                  binerIndek.add(binary);
                  System.out.println("add " + binary);
               }
            }
         }
      }
            System.out.println("isi biner indek " + binerIndek);
```

Gambar 4.2 *Encoding Huffman Kanonik (lanjutan)*

```
System.out.println("s = " + s);
                    binerIndek.add(s);
                } else {
                    for (int r = 0; r < cekInpDec.size(); r++) {
                        if (cekInpDec.get(r) == tempInp) {
                            found = true;
                            break;
                        }
                    }
                    if (found == false) {
                        cekInpDec.add(tempInp);
                        String b = "";
                        for (int q = temp.size() - 1; q >= 0; q--) {
                            b += temp.get(q).toString();
                        }
                        binerIndek.add(b);
                    }
                }
            } catch (Exception ee3) {
                System.out.println(ee3);
            }
        }
    }
```

Gambar 4.2 *Encoding Huffman Kanonik (lanjutan)*

4.2.1.3 Pembuatan *Header*

Setelah didapatkan hasil *encoding* dari gabungan kedua algoritma, tahap berikutnya adalah membuat *header* yang berisi bit penyimpan panjang tambahan '0', bit panjang biner indeks dan biner indeks LZW. *Header* ditempatkan diawal teks SMS. Prosedur untuk melakukan pembuatan ditunjukkan pada Gambar 4.3.

```
int totalBit = 0;

    for (int c = 0; c < huff2bin.length; c++) {
        System.out.println(c + ". " + hufman[c] + " : " + huff2bin[c]);
        totalBit += huff2bin[c].length();
    }
    String isiSMS = "";
    for (int y = 0; y < huff2bin.length; y++) {
        isiSMS = isiSMS + huff2bin[y];
    }
```

Gambar 4.3 Prosedur Pembuatan *Header*

```
int penambahNOL = 3;
    int pnjngbindek = binerIndek.size();
    String bitbindek = Integer.toBinaryString(pnjngbindek);
    if (bitbindek.length() <= 9) { //buat nambahi nol yang 10 bit
       int tambahan = 10 - bitbindek.length();
       String tambahane = "0";

       for (int a = 0; a < tambahan; a++) {
          //bitbindek.add(tambahane);
          bitbindek = tambahane + bitbindek;
       }
    }
    int panjangheadersementara = totalBit + 3 + binerIndek.size() * 10 + 10;
    penambahNOL = 8 - (panjangheadersementara % 8);
    String tmbhkannNol = Integer.toBinaryString(penambahNOL);
    System.out.println(tmbhkannNol);
    if (tmbhkannNol.length() < 3) { //buat nambahi nol yang 3 bit
       int tambahanNul = 3 - tmbhkannNol.length();
       String tambahNol = "0";
       System.out.println(tambahanNul);
       for (int a = 0; a < tambahanNul; a++) {
          tmbhkannNol = tambahNol + tmbhkannNol;
       }
    }
    String binerIndeknya = "";
    for (int h = 0; h < binerIndek.size(); h++) {
       binerIndeknya = binerIndeknya + binerIndek.get(h);
    }
```

Gambar 4.3 Prosedur Pembuatan *Header (lanjutan)*

4.2.2 Implementasi Dekompresi / *Decoding*

Pada subbab 3.4 inputan dari proses dekompresi adalah teks SMS dalam bentuk biner yang telah terkompresi, tujuan proses ini dilakukan adalah agar teks SMS dapat terbaca oleh *user*. Urutan proses pada tahap ini adalah sebagai berikut: pembacaan *header* , *decoding Huffman Kanonik*, kemudian decoding *LZW*.

4.2.2.1 Proses Pembacaan *Header*

Tahap ini akan membaca header pesan yang berupa 3 bit pertama sebagai penyimpan nilai banyaknya ‘0’ yang ditambahkan di akhir pesan, 10 bit berikutnya sebagai panjang biner indeks LZW dan byte berikutnya merupakan indeks LZW. Pembacaan ditunjukkan pada gambar 4.4 :

```
public void headBodynew() {
      String tigabit = "", sepuluhbit = "", binerindekk = "";
      int penambahNol = 0, pnjng10bit = 0, indek = 0, indekbodi = 0;

      for (int f = 0; f <= 2; f++) {
         tigabit = tigabit + sourceTeks.charAt(f);
      }
      System.out.println("panjang 3 bit = " + tigabit);
      penambahNol = Integer.parseInt(tigabit, 2);
      System.out.println("nol yg dibuang " + penambahNol);
      indek = 2;
      for (int s = 0; s <= 9; s++) {
         sepuluhbit = sepuluhbit + sourceTeks.charAt(s + 3);
      }
      System.out.println("10bit biner indek " + sepuluhbit);
      pnjng10bit = Integer.parseInt(sepuluhbit, 2);
      System.out.println("panjang 10 bit biner indek " + pnjng10bit);
      indek = 13;
      int panjangheaderindek = pnjng10bit * 10;
      for (int b = 0; b < panjangheaderindek; b++) {
         binerindekk = binerindekk + sourceTeks.charAt(b + indek);
      }
binerIndek = binerindekk;
      System.out.println("biner indek " + binerindekk + " , char = " +
binerindekk.length());
      indek = indek + panjangheaderindek;
      System.out.println("indek header antara 0 - " + indek);
      int batasbawah = (sourceTeks.length() - 1) - penambahNol;
      for (int g = (indek); g <= batasbawah; g++) {
         body = body + sourceTeks.charAt(g);
      }
      System.out.println("body = " + body);
   }
```

Gambar 4.4 Proses Pembacaan *Header*

Setelah pembacaan header ini, header kemudian dibuang agar tidak mengganggu proses decoding berikutnya. Sedangkan untuk bit penanda dan indeks kamus LZW dipisahkan dari *byte* pesan.

4.2.2.2 *Decoding Huffman Kanonik*

Pada tahap ini, akan dilakukan pembacaan *byte* pesan menggunakan algoritma *Huffman Kanonik* .Langkah pertama yaitu membaca 9 bit pertama, cocokkan dengan indeks kamus LZW, jika cocok outputkan nilai desimalnya, jika tidak cocok, lakukan

pencocokan dengan tabel *Huffman Kanonik.* Pada pencocokan dengan tabel, apabila bit tersebut tidak cocok, maka kurangi 1 bit belakang, lakukan pencocokan kembali. Langkah ini dilakukan hingga semua *byte* terbaca. Proses *decoding Huffman Kanonik* ini dapat dilihat pada gambar 4.5 :

```
public void bin2kanon() {
        System.out.println("header = " + header);
        ArrayList<String> headTemp = new ArrayList<String>();
        binerIndek = "";
        System.out.println("panjang header = " + header.length());

        for (int y = 5; y < header.length() - 5; y++) {
           binerIndek = binerIndek + Character.toString(header.charAt(y));
        }
        String tep;
        for (int u = 0; u < binerIndek.length() - 5; u++) {
           tep = new String();
           if ((u % 9 == 0)) {
              for (int r = u; r < u + 9; r++) {
                 tep = tep + Character.toString(binerIndek.charAt(r));
              }
              headTemp.add(tep);
              System.out.println("biner indek dari headtemp = " + tep);
           }
        }
String cob = body;
        long rem;
        String jadidesiml, buffInd, buffCur;
        int indek = cob.length() - 1;
        int awal = 0;
        String buffTemp, buffHk;
        int cur = 0;
        int panKode = 0;
        Boolean ket9bit = false;
        Boolean ketTabel = false;
        tableHuffman TH = new tableHuffman();
        buffInd = new String();
        buffHk = new String();
        System.out.println(headTemp);
while (awal < cob.length() - 9) {
           for (int i = 0; i <= 5; i++) {
              buffTemp = new String();
              buffInd = new String();
              buffHk = new String();
              jadidesiml = new String();
              for (int j = awal; j < (awal + 9) - i; j++) {
                 buffTemp += cob.charAt(j);
                 cur = j;
              }
```

Gambar 4.5 Proses *Decoding Huffman kanonik*

```
for (int h = 0; h < headTemp.size(); h++) {
                buffInd = headTemp.get(h);
                if (buffTemp.equals(buffInd)) {
                    ket9bit = true;
                } else {
                    for (int g = 0; g < TH.hufman_kanonik.length; g++) {
                        if (buffTemp.equals(TH.hufman_kanonik[g])) {
                            buffHk = Character.toString(TH.karakter[g]);
                            panKode = TH.panj_kode[g];
                            ketTabel = true;
                        }
                    }
                }
            }
            if ((ketTabel == true) && (ket9bit == false)) {
                System.out.println("ketemu " + buffHk + " " + buffTemp + " indeks ke " +
awal);
                mau_ahir += buffHk;
                awal = awal + panKode;
                ketTabel = false;
            }
            if ((ket9bit == true) && (ketTabel == false)) {
                long num = Long.parseLong(buffTemp);
                while (num > 0) {
                    rem = num % 10;
                    num = num / 10;
                    if (rem != 0 && rem != 1) {
                        System.exit(0);
                    }
                }
                int desim = Integer.parseInt(buffTemp, 2);
                System.out.println("ketemu " + buffTemp + " desimal " + desim + " indeks
ke " + awal);
                mau_ahir += desim;
                awal = awal + 9;
                ket9bit = false;
            }
        }
    }
    System.out.println("awal " + awal);
    System.out.println("last indek " + (cob.length() - 1));

    String last = "";

    for (int b = awal; b <= cob.length() - 1; b++) {
        last += cob.charAt(b);
    }
    System.out.print(last + " " + last.length());
```

Gambar 4.5 Proses *Decoding Huffman kanonik (*lanjutan*)*

```
if (last.length() == 9) {
        for (int hu = 0; hu < headTemp.size(); hu++) {
           buffInd = headTemp.get(hu);
           if (buffInd.equals(last)) {
              long num = Long.parseLong(last);
              while (num > 0) {
                 rem = num % 10;
                 num = num / 10;
                 if (rem != 0 && rem != 1) {
                    System.exit(0);
                 }
              }
              int desim = Integer.parseInt(last, 2);
              System.out.println(desim);
              mau_ahir += desim;
           }
        }
     } else {
        for (int f = 0; f < TH.hufman_kanonik.length; f++) {
           if (last.equals(TH.hufman_kanonik[f])) {
              buffHk = Character.toString(TH.karakter[f]);
              mau_ahir+=buffHk;
           }
        }
     }

  }
```

Gambar 4.5 Proses *Decoding Huffman kanonik (*lanjutan*)*

Setelah proses *decoding* dengan *Huffman Kanonik* selesai, maka langkah berikutnya yaitu *decoding* dengan menggunakan algoritma *Lempel-Ziv-Welch (LZW).*

4.2.2.3 *Decoding Lempel-Ziv-Welch (LZW)*

Pada tahap ini, dari hasil *decoding Huffman Kanonik* kemudian akan dilakukan *decoding* dengan algoritma *Lempel-Ziv-Welch (LZW). Decoding* dilakukan dengan cara membuat tabel kamus *LZW* seperti pada saat *encoding.* Proses *decoding LZW* dapat dilihat pada gambar 4.6 berikut :

```
public class parseStr {

    String text;
    ArrayList<Character> charSeq;
    ArrayList<Integer> intSeq;
    Struct LzwCode;

    {
        int kodeInt;
        char kodeChar;
    }

    public parseStr(String text) {
        this.text = text;
    }

    public String doParse() {
        System.out.println(text);

        charSeq = new ArrayList<Character>();
        intSeq = new ArrayList<Integer>();

        //ASCII bilangan 48-57
        boolean isLzw = false;
        String bufLzw = new String();
        for (int i = 0; i < text.length(); i++) {
            if (((int) (text.charAt(i)) >= 48) && ((int) (text.charAt(i)) <= 57)) {
                isLzw = true;
                bufLzw += text.charAt(i);
            } else {
                if (isLzw) {
                    System.out.println(bufLzw);
                    intSeq.add(Integer.parseInt(bufLzw));
                    charSeq.add('*');
                }
                bufLzw = new String();
                isLzw = false;
                charSeq.add(text.charAt(i));
            }
        }
            if (isLzw) {
            System.out.println(bufLzw);
            intSeq.add(Integer.parseInt(bufLzw));
            charSeq.add('*');
        }
```

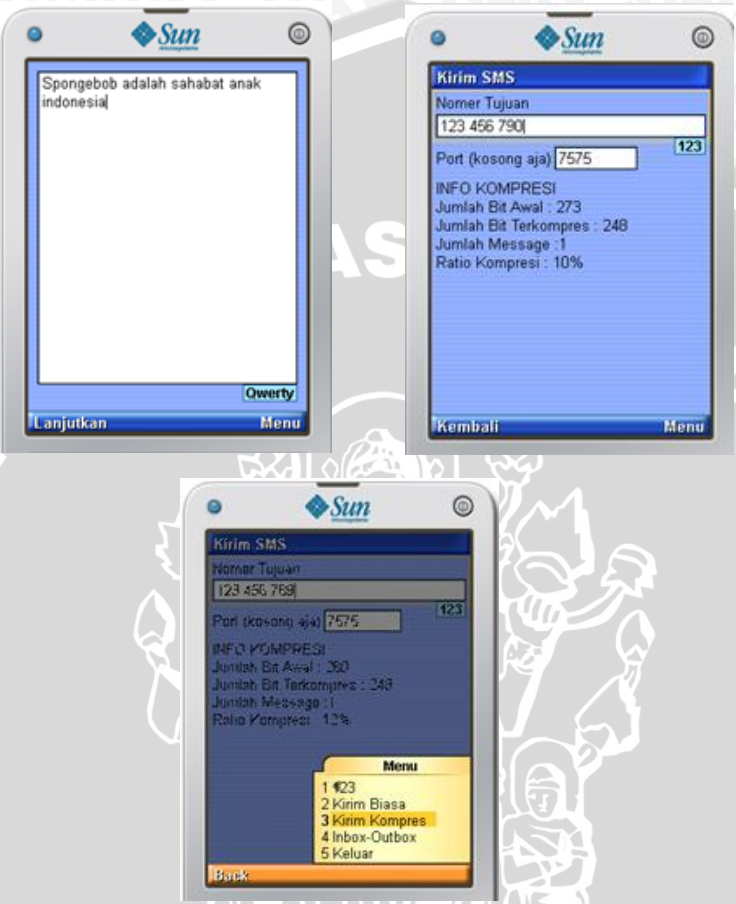
Gambar 4.6 Proses *Decoding Lempel-Ziv-Welch*

```
char[] chr2 = new char[intSeq.size()];

        for (int i = 0; i < intSeq.size(); i++) {
            System.out.print(intSeq.get(i) + " ");

        }

        String res = new String();
        System.out.println();
        int idx = 0;
        char c1;
        c1 = ' ';
        for (int i = 0; i < charSeq.size(); i++) {
            isLzw = false;
            if (charSeq.get(i) == '*') {
                charSeq.set(i, charSeq.get(intSeq.get(idx) - 10));
                chr2[idx] = charSeq.get(intSeq.get(idx) - 9);
                c1 = charSeq.get(intSeq.get(idx) - 9);
                System.out.println(c1 + " : " + charSeq.get(i));
                res += charSeq.get(i);
                res += c1;
                idx++;
                isLzw = true;
            } else {
                res += charSeq.get(i);
            }

        }

        return res;
    }
}
```

Gambar 4.6 Proses *Decoding Lempel-Ziv-Welch (lanjutan)*

4.3 Implementasi Antarmuka

Implementasi antarmuka dibuat menyerupai aplikasi SMS yang terdapat secara umum. Untuk tampilan pada uji coba kali ini menggunakan *emulator* Java Micro Edition SDK 3.0. Tampilan *form* utama dapat dilihat pada Gambar 4.7.



Gambar 4.7 Gambar form utama

Pada *form* utama, *user* dapat memasukkan teks SMS yang akan dikompresi kemudian tekan ‘lanjutkan’ untuk proses kompresi lebih lanjut. Pada *form* berikutnya akan tampak *field* untuk mengisi nomor tujuan dan info kompresi berupa jumlah bit awal pesan, jumlah bit terkompresi, jumlah halaman pesan, dan rasio kompresi dari pesan yang terkompresi. Untuk proses selanjutnya *user* memilih pilihan ‘Menu’ dan dapat memilih mengirim dengan kompresi atau tidak.

Gambar 4.8 Gambar form Inbox-Outbox

Pada sisi penerima, akan mendapat notifikasi adanya SMS baru yang masuk, *user* penerima dapat melihat isi pesan yang masuk pada pilihan 'Menu' dan memilih 'Inbox-Outbox'. Pada form Inbox-Outbox tersebut terdapat informasi nomor pengirim SMS tersebut dan isi pesan yang telah didekompresi.

4.4 Implementasi Sistem

4.4.1 Hasil Uji Prosentase Kompresi

Pengujian prosentase kompresi SMS dilakukan dengan 2 tahap pengujian. Pengujian pertama menggunakan aplikasi kompresi SMS yang sudah ada yaitu menggunakan metode Huffman . Hal ini dilakukan untuk mengetahui seberapa efektif penerapan algoritma *Lempel-Ziv-Welch* dan *Huffman kanonik* untuk kompresi SMS jika dibandingkan dengan aplikasi kompresi SMS yang sudah ada. Pengujian kedua menggunakan *Lempel-Ziv-Welch* dan *Huffman Kanonik*.

Pengujian dilakukan pada 15 teks SMS dengan panjang karakter dan jumlah halaman yang berbeda. Bagian pertama adalah teks SMS dengan panjang karakter antara 1-160 karakter (1 halaman SMS), bagian ke-dua 161-305 karakter (2 halaman SMS), dan bagian ke-tiga 306-457 karakter (3 halaman SMS). Pada pengujian ini, jumlah karakter paling sedikit 32 karakter dan paling banyak 419 karakter. Data Uji yang digunakan dalam pengujian rasio kompresi SMS ditunjukkan pada tabel 4.1 berikut :

Tabel 4.1 Tabel Data Uji

SMS ke-	Jumlah Karakter	Ukuran Bit	Jumlah halaman
1	37	259	1
2	32	224	1
3	54	378	1
4	155	1064	1
5	43	301	1
6	164	1148	2
7	169	1183	2
8	201	1407	2
9	226	1582	2
10	197	1379	2
11	222	1554	2
12	389	2723	3
13	308	2156	3
14	419	2933	3
15	324	2268	3

Untuk melakukan uji prosentase kompresi, pengujian akan dilakukan dengan menggunakan aplikasi kompresi SMS yang sudah ada dengan metode huffman dan kompresi menggunakan LZW+*Huffman Kanonik*. Hasil Kompresi SMS dapat dilihat pada tabel 4.2 :

Tabel 4.2 Perhitungan Data Uji

SMS ke-	Jumlah Karakter	Bit	Ukuran Setelah Dikompresi			
			Huffman		LZW+Huffman Kanonik	
			karakter	ukuran bit	karakter	ukuran bit
1	37	259	28	224	28	224
2	32	224	25	200	25	200
3	54	378	40	320	47	376
4	155	1064	112	896	134	1072
5	43	301	29	232	40	320
6	164	1148	102	816	120	960
7	169	1183	110	880	149	1192
8	201	1407	141	1128	175	1400
9	226	1582	152	1216	200	1600
10	197	1379	130	1040	178	1424
11	222	1554	163	1304	196	1568
12	389	2723	255	2040	343	2745
13	308	2156	201	1608	264	2112
14	419	2933	278	2224	354	2832
15	324	2268	207	1656	267	2136

Perhitungan prosentase kompresi yang akan dilakukan yaitu dibagi menjadi tiga tahap. Tahap pertama menghitung prosentase kompresi Aplikasi Kompresi Huffman, tahap kedua menghitung prosentase kompresi LZW+*Huffman Kanonik*, dan yang ke-tiga yaitu membandingkan hasil perhitungan prosentase kompresi keduanya. Hasil prosentase kompresi aplikasi Kompresi SMS dengan menggunakan Huffman dapat dilihat pada tabel 4.3 :

SMS ke-	Jumlah Karakter	Bit	Halaman	Ukuran setelah Dikompresi dengan Huffman			
				Karakter	Bit	Halaman	% Kompresi
1	37	259	1	28	224	1	24,32%
2	32	224	1	25	200	1	21,86%
3	54	378	1	40	320	1	25,93%
4	155	1064	1	112	896	1	27,74%
5	43	301	1	29	232	1	32,55%
6	164	1148	2	102	816	1	37,80%
7	169	1183	2	110	880	1	34,91%
8	201	1407	2	141	1128	2	29,85%
9	226	1582	2	152	1216	2	32,74%
10	197	1379	2	130	1040	1	34,01%
11	222	1554	2	163	1304	2	26,57%
12	389	2723	3	255	2040	2	34,45%
13	308	2156	3	201	1608	2	34,74%
14	419	2933	3	278	2224	2	33,65%
15	324	2268	3	207	1656	2	36,11%

Tabel 4.3 Prosentase Kompresi Aplikasi Huffman

Pada perhitungan prosentase kompresi menggunakan aplikasi kompresi SMS Huffman, rasio kompresi terbesar pada sms ke-6 dengan jumlah karakter SMS 164 karakter. Rata-rata rasio kompresi dengan menggunakan aplikasi ini yaitu 31,14%. Dengan rata-rata mereduksi halaman SMS sebesar 46,67%.

Perhitungan prosentase kompresi berikutnya yaitu kompresi menggunakan metode LZW+*Huffman Kanonik*. Hasil perhitungan prosentase kompresinya ditunjukkan pada tabel 4.4 berikut :

SMS ke-	Jumlah Karakter	Bit	Halaman	Ukuran setelah Dikompresi dengan LZW+Huffman Kanonik			
				Karakter	Bit	Halaman	% Kompresi
1	37	259	1	28	224	1	24,32%
2	32	224	1	25	200	1	21,88%
3	54	378	1	47	376	1	12,96%
4	155	1064	1	134	1072	1	13,54%
5	43	301	1	40	320	1	6,98%
6	164	1148	2	120	960	1	26,82%
7	169	1183	2	149	1192	1	11,83%
8	201	1407	2	175	1400	2	12,93%
9	226	1582	2	200	1600	2	11,50%
10	197	1379	2	178	1424	2	9,64%
11	222	1554	2	196	1568	2	11,71%
12	389	2723	3	343	2745	3	11,83%
13	308	2156	3	264	2112	2	14,28%
14	419	2933	3	354	2832	3	15,51%
15	324	2268	3	267	2136	2	17,59%

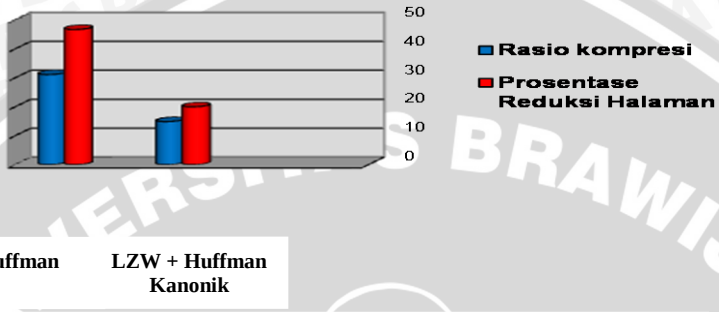
Tabel 4.4 Hasil Prosentase Kompresi LZW+*Huffman Kanonik*

Pada pengujian prosentase kompresi dengan menggunakan LZW+*Huffman Kanonik,* prosentase terbesar pada sms ke-6 dengan jumlah karakter SMS 164 karakter yaitu sebesar 26,82%. Rata-rata prosentase kompresi dengan menggunakan metode ini yaitu sebesar 14,89% dengan rata-rata mereduksi jumlah halaman SMS sebesar 20%.

4.4.2 Pembahasan dan Analisa Hasil

Berdasarkan perhitungan prosentase kompresi diatas, terjadi perbedaan rata-rata prosentase kompresi yang cukup signifikan pada kedua metode yang digunakan. Rata-rata prosentase kompresi dengan menggunakan aplikasi kompresi SMS Huffman adalah 35,72% dan rata-rata prosentase kompresi menggunakan metode LZW+*Huffman Kanonik* sebessar 14,89% . Dalam mereduksi jumlah halaman SMS, aplikasi kompresi SMS Huffman juga lebih baik dengan rata-rata mengurangi halaman SMS hingga 1 halaman. Sedangkan pada metode LZW+*Huffman Kanonik,* tidak semua SMS berhasil direduksi halamannya. Untuk melihat perbandingan

prosentase kompresi dari kedua metode tersebut ditunjukkan pada gambar 4.12 :



Gambar 4.12 Tingkat Rasio Kedua Metode

Berdasarkan pada hasil perhitungan prosentase diatas, metode Huffman memiliki rasio kompresi yang lebih tinggi daripada metode LZW dan *Huffman Kanonik*. Sehingga dapat disimpulkan bahwa metode penggabungan LZW dengan *Huffman Kanonik* tidak bisa memberikan hasil yang lebih efektif jika dibandingkan dengan aplikasi kompresi SMS yang sudah ada yaitu yang menggunakan metode Huffman.

Jika ditinjau dari pengiriman SMS jumlah karakter yang tersedia dalam satu kali pengiriman adalah 160 karakter, dengan metode kompresi LZW dan *Huffman Kanonik* jumlah rata-rata karakter dapat ditingkatkan dalam sekali pengirman yaitu sebesar :

Jumlah karakter rata-rata = K + (K * R)
= 160 + (160 * (14,89/100))
= 183 karakter

Keterangan :
K : Jumlah karakter SMS normal
R : Rata-rata prosentase Kompresi LZW+*Huffman Kanonik*

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan penelitian yang telah dilakukan, kesimpulan yang diperoleh antara lain:

1. Model kompresi metode *Lempel Ziv Welch* dan *Huffman Kanonik* dapat diimplementasikan pada kompresi *Short Message Service* (SMS).
2. Dari hasil uji coba, dapat disimpulkan bahwa penerapan penggabungan algoritma *Lempel-Ziv-Welch* dan *Huffman Kanonik* tidak dapat memberikan hasil yang lebih efektif bila dibandingkan dengan aplikasi kompresi SMS yang sudah ada yaitu menggunakan metode Huffman.
3. Berdasarkan pengujian yang telah dilakukan, diperoleh nilai rata-rata rasio kompresi dengan menggunakan gabungan metode *LZW* dan *Huffman Kanonik* sebesar 14,89% dan rata-rata mereduksi halaman SMS sebesar 20% atau bila ditinjau dari rata-rata rasio kompresi tersebut, jumlah karakter per halaman SMS adalah 183 karakter.

5.2 Saran

Saran yang diberikan untuk mengembangkan penelitian ini antara lain:

1. Melakukan penelitian lebih lanjut tentang analisa pemakaian karakter pada pembuatan *tree Huffman* biasa dengan data - data yang lebih banyak dan bervariasi sehingga didapatkan tabel *Huffman Kanonik* yang lebih baik.
2. Memperbanyak jumlah tabel *Huffman Kanonik* yang digunakan dengan variasi pemusatan kompresi yang berbeda-beda.

DAFTAR PUSTAKA

Aditya, Satrya. 2006. *Peningkatan Efisiensi Kode Huffman (Huffman Code) Dengan Menggunakan Kode Huffman Kanonik (Canonical Huffman Code).*Teknik Informatika ITB. Bandung.

American Cryptogram Association. 2005. *Letter Frequency Statistics*. http://www.cryptogram.org/cdb/words/frequency.html. Tanggal akses : 1 Desember 2010.

Anynomous. 2005. *Kompresi Dan Teks*. Fakultas Teknik Informatika Universitas Kristen Duta Wacana. http://lecturer.ukdw.ac.id/anton/download/multimedia6.pdf., Tanggal akses : 24 Oktober 2010.

Campos, Arturo. 1999. *Canonical Huffman*. http://www.arturocampos.com. Tanggal akses : 26 November 2010

Gerald R. Tamayo. 2008. *Lempel Ziv Welch (LZW) Compression.* http://compgt.googlepages.com.

J. Ziv and A. Lempel, "*Compression of Individual Sequences via Variable-rate Coding*," IEEE Transactions on Information Theory IT24, (1978), pp.530-536.

Liliana, Lipesik, V.J., 2006, *Pembuatan Perangkat Lunak Untuk Kompresi File Text Dengan Menggunakan Huffman Tree*, Fakultas Teknologi Industri Universitas Kristen Petra. Surabaya.

Linawati, Panggabean, H.P., 2004, *Perbandingan Kinerja Algoritma Kompresi Huffman, LZW, dan DMC pada Berbagai Tipe File*, INTEGRAL, Vol. 9 No. 1. Jurusan Ilmu Komputer FMIPA Universitas Katolik Parahyangan. Bandung.

Mabaklini, Yavta. 2005. *Pemampatan Data Dengan Algoritma Huffman Kanonik*. Teknik Informatika ITB. Bandung.

Nanda, Arif. 2008. *Aplikasi Pohon Dalam Teknik Kompresi Data Dengan Algoritma Huffman dan Algoritma Huffman Kanonik.* ITB.Bandung.

Nelson, Mark. 1989. LZW Data Compression. http://marknelson.us/code-use-policy/. Tanggal akses : 21 November 2010.

Purwanto, Heri.2008. *Aplikasi Kompresi SMS Teks (Short Message Service) Dengan Menggunakan Algoritma Huffman Kanonik dan LZW (Lempel-Ziv-Welch).* Universitas Gadjah Mada.Yogyakarta.

Ravi, K. 2008. *ASCII Table: 7-bit*. Dept. of Physiology University of Wisconsin. Madison. http://www.physiology.wisc.edu/comp/docs/ascii/. Tanggal akses : 25 November 2010.

R. N. Horspool, "*Improving LZW*," Proceedings of Data Compression Conference, DCC '91, IEEE Computer Society Press, April 1991, pp. 332-341.

Sayood, K. 2000. *Introduction To Data Compression, Second Edition*. Morgan Kauffman Publisher.

T. Welch, "*A Technique for High-Performance Data Compression*," Computer, June 1984.

Wibisana, Arya. 2008. *Kompresi Short Message Service Menggunakan Huffman Coding Dan Perulangan Bigram.* Ilmu Komputer Universitas Brawijaya. Malang.

LAMPIRAN

DATA LATIH YANG DIGUNAKAN

SMS ke-	Jumlah Karakter	Isi Pesan
1	37	kita rapat siang ini setelah duhur ya
2	32	saya sudah nyampe, bapak dimana?
3	54	she smart an independence, i don't think she needs me.
4	155	hehe da tidur? ya uda gapapa kok, pasti kamu baca sms ini waktu da pg hr,dimana matahari ud mw terbit, kl gt sekalian met pg aja deh. Enjoy your day then.
5	43	papa bertapa panas-panas padahal ada nanas.
6	164	kamuuuuuu udaaaaaa makaaaaaan beluuuuuum? soalnyaaaaaa akuuuuuu mauuuuuu beliiiiii makaaaaaan. nitiiiiiip gaaaaaak? habis iniiiiii akuuuuuu mauuuuuu berangkaaaaaat!
7	169	kemaren saya sudah coba hubungi, tapi tidak ada jawaban, coba langsung ditanya aja kekantornya. minta kejelasan tentang acara nikahannya. oya untuk tempatnya sudah oke?
8	201	halo halo...gimana kabarnya temanku ini sekarang? apakah sudah sukses? ataukah masih berjuang mengejar cita-cita masa muda dulu? hohoho...kasih kabar ya ke aku secepatnya, soalnya mau reunian nih. thx.
9	226	masalah tidak terdapatnya kemunculan suatu bit pada state dapat diatasi dengan menginisialisasi model awal state dengan satu. probabilitas dihtung menggunakan frekuensi relatif dari dua transisi yang keluer dari state yg baru.
10	197	Since all possible one character strings are already in the dictionary, each encoding step begins with a one character prefix, so the first string searched for in the dictionary has two characters.
11	222	Work has been done on lossy text compression on a word level, but to my knowledge it has not been done on the letter level as I described. With such a method, text files could meaningfully be compressed tremendous amounts.
12	389	Fungsi pencarian kata dipanggil saat proses kompresi, fungsi ini yang membedakan ketika akan mengkompresi karakter atau bigram. Fungsi mendapat masukkan dua karakter,yang mana dari kedua karakter ini akan dicocokkan dengan bigram-bigram yang telah dicari sebelumnnya. Jika dua karakter tersebut adalah bigram maka hasil kembalian dari fungsi ini adalah kode bigram kedua karakter tersebut.
13	308	sebenarbenarnya seharusnya kita itu berbuat benar dengan mata berbinar binar, agar kebenaran terlihat dari mata kita. bukankah kita diberikan mata yang indah dan bersinar

		saudaraku?senyumlah maka orang akan tersenyum padamu. sampaikanlah salamku pada teman-teman yang berjuang disana agar mereka bersemangat.
14	419	Pada dasarnya program kompresi sms tidak selalu berjalan mulus karena banyak sekali faktor yang mempengaruhi. hal-hal yang mempengaruhinya antara lain : banyaknya variasi huruf dalam string sms, kemudian menyebabkan panjangnya indeks LZW yang dikirimkan sehingga hasilnya kurang optimal. semoga nantinya ada yang bisa mengembangkan program ini sehingga bisa lebih sempurna dalam mengkompresi sms yang hendak dikirimkan.
15	324	puasa puasa ayo kita puasa, puasa puasa menahan lapar dan dahaga. puasa puasa ayo kita berpuasa, semoga semoga Allah merahmati kita semua. karena dengan puasa, badan lebih sehat dan jiwa lebih bersih. maka berpuasalah. kalau kamu tidak puasa maka pahala puasa orang lain akan dilipatgandakan. puasa yuk puasaaaaaaaaaaaa.....