Aplikasi Kompresi SMS Teks (*Short Message Service*) Dengan Menggunakan Algoritma Huffman Kanonik dan LZW (Lempel-Ziv-Welch)

Heri Purwanto[1)] , Anny Kartika Sari[2)]
1) Mahasiswa Fakultas Matematika dan Ilmu Pengetahuan Alam, jurusan Ilmu Komputer, Universitas Gadjah Mada
2) Dosen tetap Fakultas Matematika dan Ilmu Pengetahuan Alam, jurusan Ilmu Komputer, Universitas Gadjah Mada
Gedung Selatan Sekip Unit III Yogyakarta, Telp: 0274-546194, Fax: 0274-546194
e-mail : **herpur@mail.ugm.ac.id, a_kartikasari@ugm.ac.id**

Abstrak

Aplikasi kompresi SMS merupakan aplikasi yang dibuat agar dapat melakukan proses kompresi dan dekompresi terhadap teks SMS untuk kemudian mengirimkan atau menerima pesan tersebut. Algoritma kompresi yang digunakan pada aplikasi ini adalah algoritma *Huffman* kanonik dan LZW dimana teks SMS yang akan dikompresi harus merupakan teks SMS *default* mengacu pada spesifikasi GSM 03.38. Pada saat mengirim pesan, aplikasi akan bertindak sebagai kompresor sedangkan pada saat menerima pesan aplikasi akan bertindak sebagai dekompresor. Aplikasi ini dibuat dengan menggunakan bahasa JAVA, yaitu J2ME, dan akan berjalan pada ponsel berbasis JAVA MIDP 2.0.

Aplikasi kompresi SMS secara keseluruhan mampu untuk melakukan proses kompresi dan dekompresi terhadap teks SMS. 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%.

Kata kunci: kompresi teks SMS, dekompresi teks SMS, *Huffman* kanonik, LZW.

1. Pendahuluan

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.

Dalam menulis pesan SMS, seorang pengguna biasa melakukannya dengan cara menyingkat isi pesan tersebut. Hal ini dilakukan selain karena kesulitan atau malas untuk mengetikkan isi pesan juga untuk menghemat tempat sehingga terkadang, isi pesan yang ada, harus di edit untuk menyesuaikan agar dapat termuat dalam satu halaman SMS. Hal ini tentu akan sedikit merepotkan para pengguna ketika harus melakukan pengiriman SMS sehingga perlu dibuat suatu aplikasi untuk menambah pemuatan karakter SMS yang akan dikirimkan dalam satu halaman, dengan cara mengkompresi isi pesan atau teks SMS tersebut.

Kompresi merupakan proses pengubahan sekumpulan data menjadi suatu bentuk kode untuk menghemat kebutuhan tempat penyimpanan dan waktu untuk transmisi data[1]. Contoh kompresi sederhana yang biasa dilakukan misalnya adalah menyingkat kata-kata yang sering digunakan tapi sudah memiliki konvensi umum. Misalnya kata “yang” dikompres menjadi kata “yg”. Saat ini terdapat berbagai tipe algoritma kompresi, antara lain: Huffman, LZ77 dan variannya (LZ78, LZW, GZIP), *Dynamic Markov Compression* (DMC), *Run-Length*, *Shannon-Fano* dan lain-lain. Kompresi data menjadi sangat penting karena dapat memperkecil kebutuhan penyimpanan data, mempercepat pengiriman data, dan memperkecil kebutuhan *bandwidth*.

Proses kompresi terhadap teks SMS dilakukan dengan cara melakukan proses enkoding dan dekoding terhadap data berupa teks SMS. Enkoding data dilakukan dengan cara menyusun *string* biner dari teks yang ada (mengkodekan informasi) menggunakan bit atau informasi lain yang lebih rendah daripada representasi data yang tidak terkodekan dengan suatu sistem enkoding tertentu. Proses dekoding merupakan kebalikan dari enkoding yang berarti menyusun kembali data dari *string* biner menjadi sebuah karakter kembali. Proses enkoding dilakukan terhadap data SMS yang akan dikirimkan ke ponsel tujuan sedangkan dekoding dilakukan terhadap data SMS yang diterima oleh ponsel penerima.

Dengan adanya proses kompresi terhadap teks SMS maka akan terjadi pemampatan terhadap data SMS sehingga dapat menghemat biaya pengiriman SMS (pulsa). Meskipun saat ini biaya pengiriman SMS semakin murah, dengan adanya perang tarif antar operator penyedia layanan SMS, tetapi tarif murah tersebut berlaku untuk sesama operator dan berlaku untuk pengiriman di dalam negeri saja. Selain itu dengan adanya proses enkoding dan dekoding terhadap teks SMS maka hal ini dapat memproteksi isi pesan SMS ketika terjadi penyadapan. Hal ini terjadi karena isi pesan akan sulit dibaca jika pesan tersebut belum dilakukan proses dekoding.

2. Short Message Service (SMS)

SMS (*Short Message Service*) adalah salah satu fasilitas standar dari GSM yang digunakan untuk mengirim dan menerima pesan berupa teks ke dan dari sebuah ponsel. Untuk dapat menggunakan fasilitas SMS, pengguna perlu melengkapi ponselnya dengan ponsel dan kartu SIM (*Subscriber Identity Module*) dari penyedia layanan GSM yang mendukung SMS. Sebuah pesan SMS tidak dikirimkan langsung dari ponsel pengirim ke ponsel penerima tetapi akan dikirimkan terlebih dahulu ke *SMS Center*. Ketika ponsel tujuan tidak aktif, sistem akan menunda pengiriman pesan ke ponsel tujuan sehingga ponsel tujuan aktif kembali. Apabila terjadi kegagalan pengiriman pesan yang bersifat sementara (misalnya: ponsel tujuan tidak aktif) akan dilakukan pengiriman ulang pesan,

kecuali bila diberlakukan aturan bahwa pesan yang telah melampaui batas waktu tertentu harus dihapus dan dinyatakan gagal terkirim.

Sebuah pesan SMS mempunyai panjang maksimal 140 *byte* atau 160 karakter dalam penyandian ASCII 7-bit, yang mana format ini merupakan format standar yang digunakan pada SMS[2]. Pesan SMS dalam penyandian 8-bit mempunyai panjang maksimal 140 karakter dan biasanya digunakan untuk mengirimkan pesan cerdas (*smart messaging*) seperti gambar atau *ringtone* dan pengiriman data melalui OTA (*over the air*) untuk melakukan setting WAP. Pesan yang dikirimkan dalam tulisan Arab, Korea, Cina atau tulisan lainnya dengan format penyandian 16-bit, maka penulisan SMS akan dibatasi hingga 70 karakter.

Sebuah pesan standar yang mengandung 160 karakter atau kurang akan dihitung sebagai satu SMS. Untuk sebuah *concatenated message* yang mengandung lebih dari 160 karakter, setiap 153 karakter akan dihitung sebagai satu SMS, karena 7 karakter lainnya digunakan sebagai penanda (tag) nomor bagian dari setiap bagian tersebut. Dengan *concatenated messaging*, walaupun pesan mengandung lebih dari 160 karakter, setiap SMS yang dikirimkan tetap terdiri dari 160 karakter, hanya saja dengan adanya penanda nomor bagian tadi, saat SMS tersebut diterima oleh ponsel yang mendukung *concatenated messaging*, maka beberapa SMS tadi akan langsung digabungkan menjadi satu pesan yang panjang [3].

3. Huffman Kanonik

Algoritma huffman akan menggunakan tabel yang menyimpan frekuensi kemunculan dari masing-masing simbol yang digunakan dalam file tersebut dan kemudian mengkodekannya dalam bentuk biner[4]. 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 (dekoder) 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 [4] 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.

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 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 (enkoder) maupun penerima (dekoder), sebagai acuan untuk melakukan proses enkoding/dekoding terhadap teks SMS.

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. Setelah terbentuk kode *Huffman* langkah berikutnya adalah melakukan transformasi menjadi kode *Huffman kanonik.* Secara garis besar data yang dibutuhkan pada tabel *Huffman* meliputi data-data berikut (tabel 1):

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.

Tabel 1 Tabel Huffman Kanonik Karakter SMS yang Sudah Dibentuk

Huruf	Kode Huffman Kanonik	Angka	Panjang Kode
o	011100	28	6
t	10110	22	5
i	1110	14	4
n	1111	15	4

Untuk melakukan proses enkoding terhadap teks SMS yang akan dikompres maka setiap karakter akan dikodekan berdasarkan tabel tersebut. Sebagai contoh *string* “toti“, berdasarkan tabel 1, akan dikodekan menjadi “10110011100101101110“. Untuk melakukan proses dekoding maka hal pertama yang harus dilakukan adalah membaca bit-bit tersebut berdasarkan panjang kode yang paling panjang, berdasarkan tabel 1 adalah 6 dan berdasarkan kode diatas akan di dapat “ 101100“. Setelah dilakukan pembacaan bit maka langkah berikutnya adalah melakukan konversi terhadap kode tersebut menjadi bentuk desimal, “101100“ akan direpresentasikan ke dalam bentuk desimal menjadi 44. Setelah didapat bentuk desimal maka lakukan pencarian pada tabel 1 apakah kode dengan panjang 6 dan representasi desimal 4 terdapat di dalam tabel atau tidak. Jika data tersebut tidak ditemukan maka lakukan pergeseran sebanyak 1 bit sehingga kini data menjadi “ 10110“ dan akan direpresentasikan sebagai 22. Lakukan pencarian data pada tabel 1 sampai data berhasil ditemukan. Proses ini dilakukan sampai semua bit telah berhasil diterjemahkan.

4. Lempel-Ziv-Welch (LZW)

Algoritma kompresi LZW akan menggunakan kamus dimana rangkaian string akan digantikan dengan indeks yang diperoleh dari sebuah kamus. Hal ini berarti untuk meng-enkode suatu string maka data yang akan disimpan adalah angka yang merepresentasikan indeks string tersebut pada kamus. Kamus yang digunakan untuk melakukan proses enkoding LZW adalah data yang berada pada tabel a dimana data yang akan digunakan hanya berupa indeks huruf pada tabel tersebut. Sebagai contoh huruf “i“ pada tabel 1 berada pada baris ke-3 sehingga akan dikodekan menjadi “11“ sebagai representasi biner dari 3.

Algoritma untuk melakukan proses enkoding dengan menggunakan algoritma LZW dapat dilihat pada gambar 1 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 1 Algoritma enkoding LZW [5]

Untuk lebih jelasnya diberikan contoh yaitu terdapat string “abababab” dengan kamus {‘a’, ‘b’} yang mana tiap simbol pada kamus tersebut mempunyai indeks 0 dan 1. Dengan mengacu pada algoritma kompresi LZW yang ada pada gambar 1 diatas maka proses enkoding dapat dilihat pada gambar 2 berikut.

STRING (S)	CHARACTER (CH)	Cek S+CH di Kamus	Kode Output (indeks S)	Indeks Simbol Baru di kamus	Simbol Baru
a	b	kosong	0	2	ab
b	a	kosong	0, 1	3	ba
a	b	ada	-	-	-
ab	a	kosong	0, 1, 2	4	aba
a	b	ada	-	-	-
ab	a	ada	-	-	-
aba	b	kosong	0, 1, 2, 4	5	abab
b	**EOF**	-	0, 1, 2, 4, 1	-	-

Gambar 2 Contoh proses enkoding algoritma LZW

Dari gambar tersebut hasil proses enkoding menghasilkan 0, 1, 2, 4, 1. Dari gambar 2 diatas terlihat bahwa indeks paling besar adalah 5 (“101”, jika dalam bentuk biner). Dengan mengasumsikan bahwa indeks simbol tidak lebih besar dari angka 7 (“111”, jika dalam bentuk biner) maka setiap indeks yang disimpan hanya membutuhkan 3 bit saja. Dari hasil enkoding diatas maka total bit yang dibutuhkan untuk menyimpan indeks data tersebut adalah sebanyak 3 x 5 = 15 bit. Hal ini jauh lebih kecil bila dibandingkan dengan data asli sebelum proses kompresi yaitu 8 x 8 = 64 bit.

Untuk melakukan proses dekoding pada algoritma LZW langkah-langkahnya dapat dilihat pada gambar 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 3 Algoritma dekoding LZW [5]

Untuk lebih jelasnya diberikan contoh berdasarkan data dari hasil enkoding LZW diatas yaitu: 0,1, 2, 4, 1. Dari kumpulan angka ini akan di dekode sehingga akan dihasilkan kumpulan karakter seperti semula yaitu “abababab”. Dengan menggunakan algoritma dekoding LZW diatas maka akan didapat hasil seperti yang terlihat pada gambar 4 berikut.

Old Code (OC)	New Code (NC)	Cek NC di Kamus	STRING (S)	CHARACTER (Huruf awal pada S)	OUTPUT (S)	Simbol Baru
0	-	-	-	a	**a**	-
0	1	ada	b	b	**a**b	ab (indeks = 2)
1	2	ada	ab	a	**ab**ab	ba (indeks = 3)
2	4	kosong	aba	a	**abab**aba	aba (indeks = 4)
4	1	ada	b	b	**abababa**b	abab (indeks = 5)
1	**EOF**	-	-	-	-	-

Gambar 4 Contoh proses dekoding algoritma LZW

Dari gambar 4 terlihat bahwa hasil dekoding dengan algoritma LZW menghasilkan nilai keluaran yang sama dengan nilai semula yaitu “abababab” dan simbol-simbol yang terdapat pada kamus hasil dekoding mempunyai nilai yang sama dengan yang terdapat pada kamus hasil enkoding.

5. Mengirim dan Menerima SMS Pada J2ME

Pada MIDP paket untuk menangani SMS ditangani oleh paket opsional yaitu WMA (*Wireless Messaging API*). Pada WMA 2.0 terdapat tiga bentuk pengiriman pesan yaitu [6]:

1. Pesan biner (*binary message*) yaitu pesan berbentuk biner yang dikirimkan melalui SMS. Pesan jenis ini menggunakan *encode* data 8-bit dengan jumlah data maksimum tiap halaman SMS sebesar 140 byte atau 133 byte jika nomor *port* disertakan.
2. Pesan teks (*text message*) yaitu pesan dalam bentuk teks yang dikirimkan melalui SMS. Pada pesan jenis ini jika data yang digunakan berada dalam format GSM 7-bit maka jumlah karakter maksimum dalam satu halaman SMS

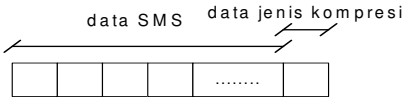
adalah sebanyak 160 karakter atau 152 karakter jika nomor *port* disertakan. Jika data yang digunakan berada dalam format UCS-2 maka jumlah karakter maksimum adalah sebanyak 70 karakter atau 66 karakter jika nomor *port* disertakan.

3. Pesan *multipart* (*multipart message*) yaitu pesan multi-media yang dikirimkan melalui MMS.

Untuk menerima pesan SMS, WMA menggunakan URL "sms://" sebagai pengenal protokol SMS. Di dalam WMA terdapat dua format pengalamatan untuk mengirim pesan, yaitu [7]:

1. Format **sms://<no piranti mobile>**. SMS yang dikirim menggunakan format ini akan langsung ditangkap oleh *inbox* dan tidak akan bisa diterima oleh aplikasi WMA.
2. Format **sms://<no piranti mobile>:<Port>.** Format ini berguna untuk mengirim SMS ke piranti *mobile* lain yang mengaktifkan aplikasi WMA sebagai penerima SMS. Port berguna untuk komunikasi antar aplikasi WMA. Apabila port tidak disertakan, maka SMS yang diterima akan masuk ke dalam *inbox* piranti *mobile* akibatnya WMA tidak akan pernah menerima SMS.

Penelitian ini menggunakan proses pengiriman dan penerimaan pesan dalam format biner. Proses kompresi/dekompresi terhadap teks SMS dilakukan dengan cara memanipulasi bit-bit pada karakter SMS yang akan dikirimkan sesuai dengan algoritma kompresi yang dipilih oleh *user*. Pada bagian akhir dari pesan yang dikirimkan akan disisipkan, satu bit/*byte* data, informasi mengenai jenis kompresi yang digunakan sehingga akan memudahkan aplikasi ketika akan melakukan proses dekompresi. Secara umum data SMS yang akan dikirimkan pada penelitian ini mempunyai format seperti yang ditunjukkan pada gambar 5 berikut.



Gambar 5 Format data SMS yang akan dikirimkan

6. Menyimpan Data Pada J2ME

MIDlet tidak menggunakan file sistem untuk menyimpan data, tetapi menyimpan semua informasi dalam sebuah memori *non-volatile* (memori tetap) yang disebut *Record Management System* (RMS). RMS merupakan suatu sistem manajemen penyimpanan *record* yang mengacu pada satu tabel dengan kumpulan *record*. Dalam RMS tidak terdapat *primary key* dan *foreign key* yang bisa didefinisikan. *Primary key* data pada RMS telah terdefinisi secara otomatis sebagai id *record* bertipe integer.

Record pada RMS disimpan sebagai *array* dari *byte* yang cara kerjanya berdasarkan *record* (baris data). RMS memiliki orientasi *record* basis data yang sederhana sehingga tidak mengenal *field* (kolom data) seperti database pada umumnya. Dalam RMS tidak bisa mengambil data per *field* per *record* seperti yang biasa dilakukan pada database yang umum, sehingga perlu dipetakan dahulu datanya.

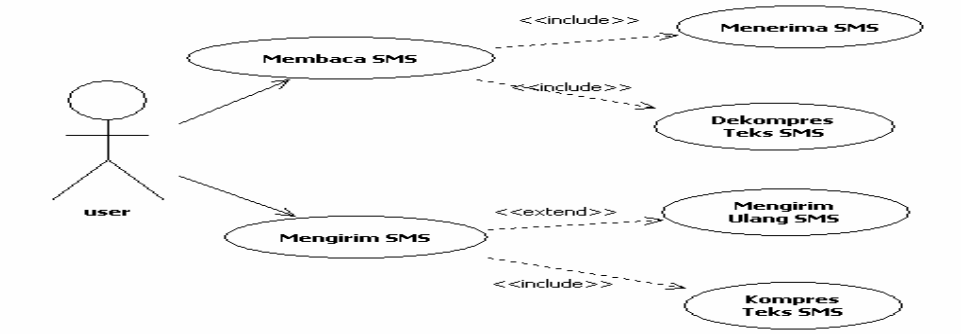
Paket RMS terdapat pada *javax.microedition.rms* yang memiliki beberapa *interface* dan satu kelas. Satu-satunya kelas yang digunakan untuk memanipulasi data adalah kelas *RecordStore*. Kelas dan *Interface* yang terdapat pada paket RMS adalah [7]:

1. Kelas *RecordStore*. Kelas ini digunakan untuk membuka, membaca, menulis, mengubah, menghapus dan menutup data RMS.
2. *Interface RecordEnumeration*. *Interface* ini digunakan untuk menelusuri data RMS akan tetapi tidak dapat digunakan untuk menambah, mengubah atau menghapus data RMS. Fungsi lain dari *interface* ini adalah untuk menyaring dan mengurutkan data.
3. *Interface RecordComparator*. *Interface* ini berguna untuk membandingkan dua record dengan metode *compare()* terimplementasi. Hasil dari perbandingan ini akan berupa konstanta integer statis: *PRECEDES* (parameter pertama bernilai lebih kecil), *EQUIVALENT* (kedua parameter bernilai sama) dan *FOLLOW* (parameter pertama bernilai lebih besar). Fungsi lain dari *interface* ini adalah menampilkan data yang urut pada *interface RecordEnumeration*.
4. *Interface RecordFilter*. *Interface* ini digunakan untuk menguji *record* dengan metode *matches()* terimplementasi yang akan mengembalikan nilai *boolean*: *true* (bila lulus uji) dan *false* (bila tidak lulus uji). Kegunaan lain dari *interface* ini yaitu untuk menyaring data pada *interface RecordEnumeration*.

J2ME tidak menyediakan API untuk mengakses inbox dan outbox ponsel. Dengan keterbatasan tersebut maka dibuat suatu *inbox* dan *outbox* buatan yang fungsinya hampir sama dengan *inbox* dan *outbox* pada ponsel. Pesan yang masuk atau keluar akan disimpan ke dalam suatu tabel di dalam RMS, khusus disediakan untuk menampung isi SMS yang akan diberi nama tabel *tInbox* (untuk menyimpan pesan yang ditampilkan pada *inbox* buatan) dan *tOutbox* (untuk menyimpan pesan yang ditampilkan pada *outbox* buatan).

7. Rancangan Sistem

Sistem kompresi yang akan dibangun dirancang untuk dapat melakukan proses kompresi dan dekompresi terhadap teks SMS yang akan dikirim atau diterima pada ponsel berbasis JAVA MIDP 2.0. Secara garis besar sistem kompresi SMS mempunyai fungsi utama untuk mengirim dan menerima SMS yang sudah terkompres dimana fungsi-fungsi utama pada sistem ini dapat dilihat pada gambar 6 *diagram use case* berikut.



Gambar 6 Use Case aplikasi kompresi SMS

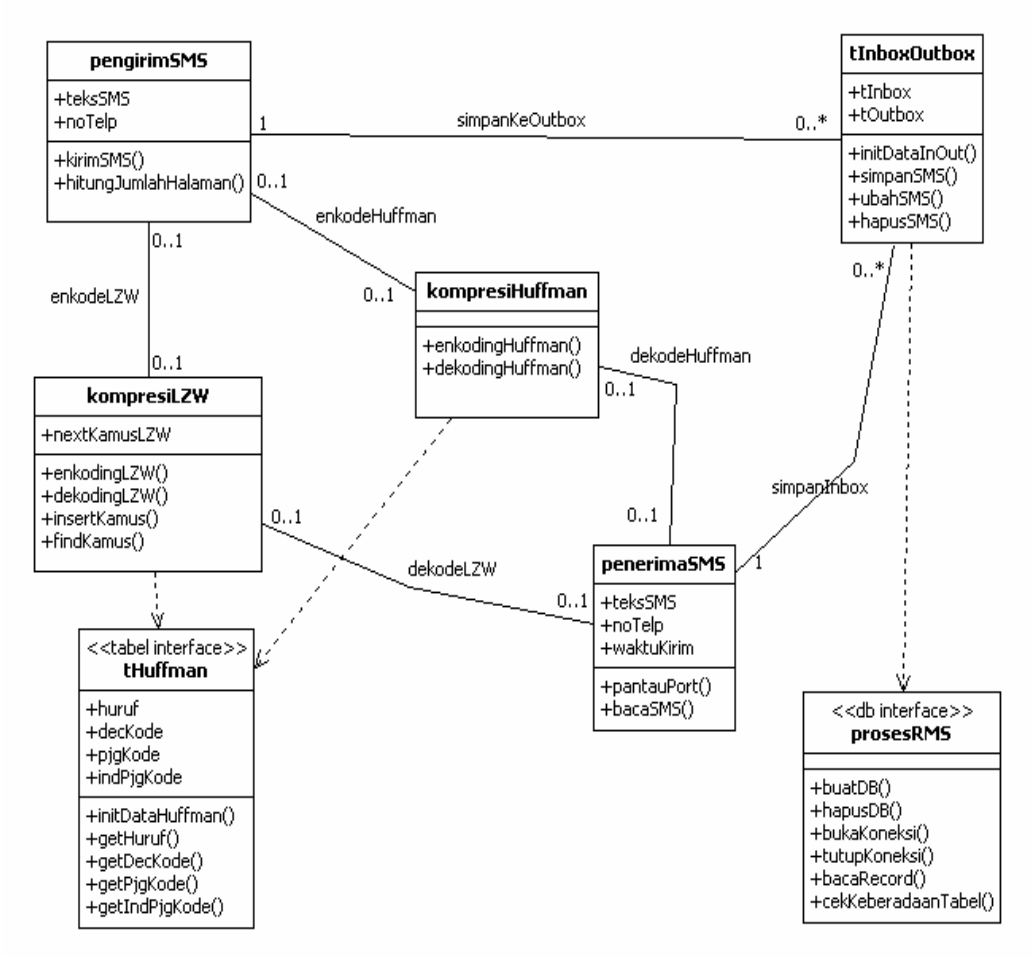
Skenario dari proses mengirim SMS dimulai pada saat user memasukkan isi pesan ke dalam sistem. Pada saat user akan mengirimkan pesan maka terlebih dahulu akan dilakukan proses kompresi terhadap pesan tersebut. Hal ini dilakukan untuk memampatkan teks SMS sehingga ukurannya menjadi lebih kecil daripada ukuran sebenarnya. Proses kompresi dilakukan dengan cara memanipulasi bit-bit penyusun karakter dengan menggunakan algoritma enkoding LZW atau Huffman Kanonik.

Untuk dapat membaca SMS maka sistem harus dapat menangkap atau menerima SMS yang ditujukan pada sistem. Dalam proses menerima SMS, ketika aplikasi dalam keadaan tidak aktif, AMS akan memberikan *alert* kepada user apakah aplikasi akan diaktifkan atau tidak untuk melakukan proses penerimaan SMS. Jika Aplikasi sudah diaktifkan, maka aplikasi akan melakukan proses penerimaan pesan untuk kemudian menampilkan pemberitahuan kepada user, berupa nada atau bentuk lainnya. Pada saat user akan membaca SMS maka pada pesan tersebut harus dilakukan proses dekompres terlebih dahulu. Hal ini dilakukan agar isi pesan dapat dipahami karena sebelum dikirimkan pesan tersebut telah mengalami proses kompresi terlebih dahulu. Algoritma dekompresi (dekoding) yang disediakan pada aplikasi ini terdapat dua pilihan algoritma yaitu Huffman Kanonik dan LZW.

Pada aplikasi kompresi SMS implementasi dari fungsi-fungsi yang dibutuhkan akan dibuat ke dalam bentuk kelas-kelas. Diagram kelas dari aplikasi kompresi SMS terdiri dari kelas pengirim SMS, penerima atau pembaca SMS, kompresi SMS (enkoding), dan dekompresi SMS (dekoding). Proses enkoding dan dekoding merupakan pasangan yang tidak bisa dipisahkan. Hal ini berarti jika suatu teks dilakukan proses enkoding dengan menggunakan algoritma Huffman Kanonik maka untuk melakukan proses dekoding harus juga dilakukan dengan menggunakan algoritma Huffman Kanonik.

Untuk melakukan proses kompresi dan dekompresi SMS pada aplikasi ini akan digunakan dua kelas yaitu kompresi Huffman dan kompresi LZW. Pada kedua kelas tersebut (kompresi Huffman dan kompresi LZW) akan terdapat suatu *method* untuk melakukan proses enkoding dan dekoding sesuai dengan algoritma yang digunakan. Pada aplikasi ini setiap SMS baik yang diterima atau yang dikirimkan akan disimpan ke dalam suatu tabel untuk kemudian ditampilkan pada inbox atau outbox buatan. Untuk itu akan ditambahkan satu kelas yaitu tInboxOutbox yang berfungsi untuk menangani hal-hal yang berkaitan dengan data pada inbox atau outbox buatan.

Pada diagram kelas ini akan terdapat dua buah *interface* yaitu prosesRMS dan tHuffman. Interface prosesRMS digunakan untuk mengimplementasikan hal-hal yang berkaitan dengan proses pada RMS. Interface tHuffman digunakan untuk melakukan inisialisasi terhadap tabel Huffman. Selain itu proses yang memiliki kesamaan yang digunakan pada kelas kompresiHuffman dan kelas kompresiLZW akan diimplementasikan pada *interface* ini. Diagram kelas aplikasi kompresi SMS dapat dilihat pada gambar 7 berikut.



Gambar 7 Diagram kelas aplikasi kompresi SMS

Dari gambar 7 terlihat bahwa pengirimSMS dapat menyimpan isi pesan pada tabel *tOutbox* beberapa kali. Namun, satu pesan yang disimpan di tabel *tOutbox* harus berasal dari satu kali proses pengiriman saja. Teks yang akan dikirimkan harus dilakukan proses kompresi terlebih dahulu. Namun, dari kedua metode kompresi yang disediakan, *user* harus memilih salah satu diantaranya secara manual. Hal ini menyebabkan satu pesan yang akan dikirimkan hanya dapat berasosiasi dengan kelas kompresiHuffman atau kelas kompresiLZW saja.

Pada saat menerima pesan, data SMS yang diterima akan disimpan pada tabel *tInbox*. Data SMS yang disimpan pada tabel ini harus berasal dari satu proses penerimaan SMS saja sedangkan penerimaSMS dapat menyimpan isi pesan beberapa kali ke dalam tabel tersebut. Untuk dapat mengetahui isi pesan yang dibaca, user harus melakukan proses dekompresi terlebih dahulu terhadap isi pesan yang diterima. Pesan yang akan di dekompresi hanya dapat menggunakan satu algoritma dekoding saja, sesuai dengan proses enkoding yang digunakan pada waktu mengirim SMS, oleh karena itu satu pesan yang diterima hanya dapat berasosiasi dengan kelas kompresiHuffman atau kelas kompresiLZW saja.

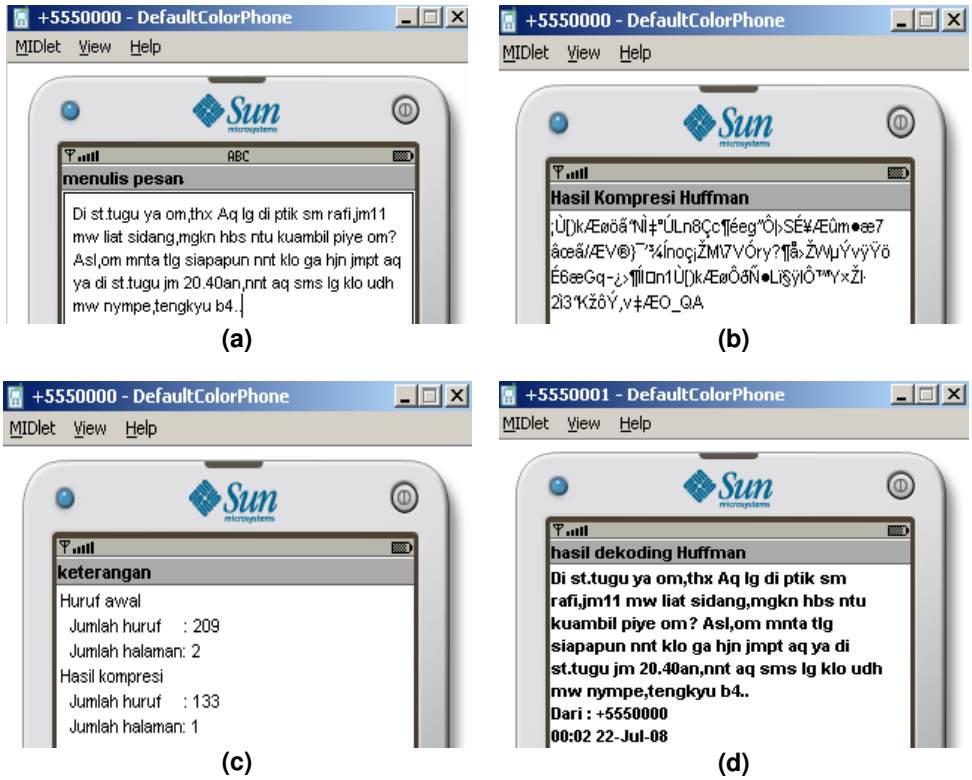
8. Pengujian Sistem

Pengujian dilakukan dengan tujuan untuk mengetahui seberapa jauh pengaruh proses kompresi terhadap jumlah karakter dan jumlah halaman SMS yang dihasilkan. Proses pengujian dilakukan dengan tujuan untuk menghitung rasio antara data sebelum dilakukan proses kompresi dengan data setelah dilakukan proses kompresi. Untuk menghitung rasio kompresi akan di persamaan 1 berikut[8]:

$$\text{Rasio} = (1 - (\text{compressed\_size} / \text{raw\_size})) * 100\% \quad (1)$$

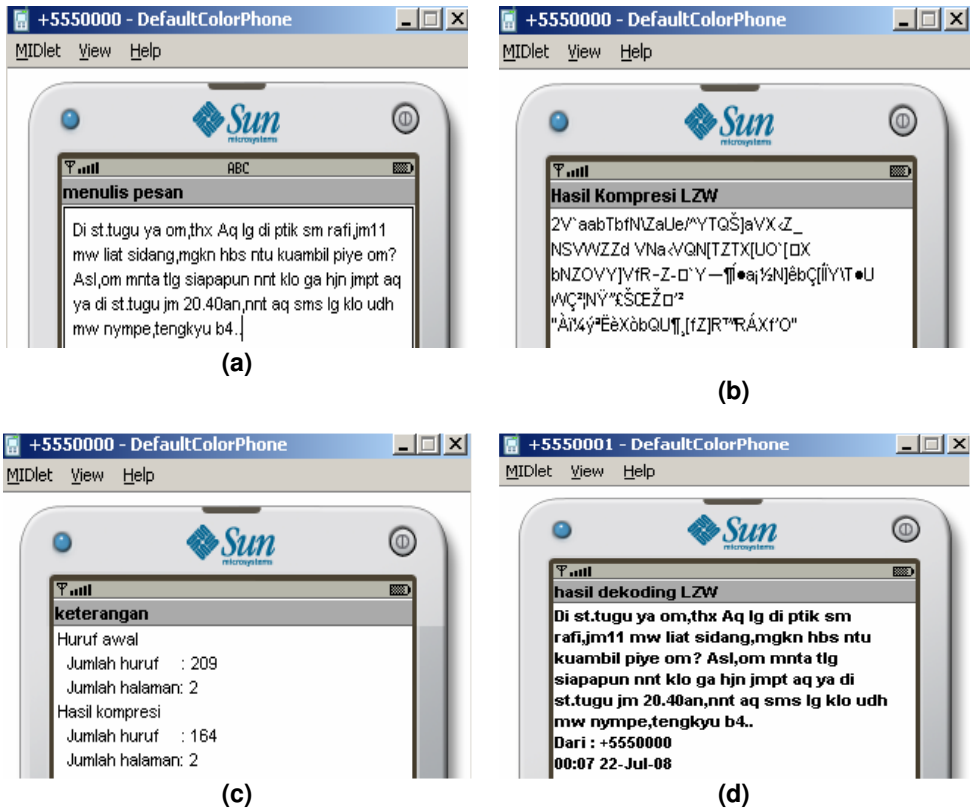
dimana compressed_size menyatakan kapasitas teks setelah dikompresi dan raw_size menyatakan kapasitas teks sebelum dikompresi (data mentah).

Proses pengujian dilakukan pada ponsel Motorola E398 dan Nokia N70 dan telah berjalan dengan baik. Untuk kepentingan *visual* maka hasil pengujian aplikasi pada laporan ini akan disajikan hasil pengujian dengan menggunakan emulator. Pengujian pertama dilakukan proses kompresi terhadap teks SMS dengan menggunakan algoritma Huffman Kanonik. Pada pengujian ini akan ditunjukkan teks SMS awal, teks SMS setelah dikompresi dan kesimpulan hasil kompresi yang meliputi jumlah karakter sebelum dan sesudah kompresi berikut jumlah halaman SMS-nya. Pengujian aplikasi terhadap teks SMS dengan menggunakan algoritma Huffman Kanonik dapat dilihat pada gambar 8 berikut.



Gambar 8 Proses kompresi SMS dengan menggunakan algoritma Huffman kanonik. (a) Keadaan awal teks SMS. (b) Teks setelah dikompresi. (c) Keterangan hasil kompresi. (d) Teks SMS setelah diterima dan dilakukan proses dekoding.

Proses pengujian kedua akan dilakukan proses kompresi teks SMS dengan menggunakan algoritma LZW (gambar 9).



Gambar 9 Proses kompresi SMS dengan menggunakan algoritma LZW. (a) Keadaan awal teks SMS. (b) Teks setelah dikompresi. (c) Keterangan hasil kompresi. (d) Teks SMS setelah diterima dan dilakukan proses dekoding.

9. Analisis Hasil Pengujian

Percobaan yang dilakukan pada gambar 8 menghasilkan rasio kompresi sebesar 36.36% dimana aplikasi berhasil mereduksi jumlah halaman SMS dari yang semula dua halaman menjadi satu halaman. Dari gambar tersebut juga terlihat bahwa isi pesan sebelum dan sesudah dikirimkan dari ponsel yang satu (ponsel dengan nomor 5550000) ke ponsel yang lain (ponsel dengan nomor 5550001) adalah sama. Percobaan pada gambar 9 rasio kompresi yang dihasilkan adalah sebesar 21.53%. Pada percobaan ini, isi pesan antara ponsel pengirim dan ponsel penerima adalah sama. Namun demikian, pada percobaan ini aplikasi mengalami kegagalan untuk mereduksi jumlah halaman SMS.

Berdasarkan percobaan yang dilakukan, algoritma LZW akan menghasilkan kompresi maksimal ketika data yang dikompres mempunyai variasi simbol yang sedikit dan komposisi string yang banyak berulang begitu juga

sebaliknya. Hal ini bisa dipahami karena semakin banyak data yang dimasukkan ke dalam kamus maka akan semakin besar pula indeks dari huruf tersebut pada kamus. Dengan membesarnya indeks huruf ini maka lebar data yang diperlukan untuk menyimpan data tersebut akan bertambah lebar pula. Algoritma kompresi Huffman kanonik akan menghasilkan nilai rasio kompresi yang maksimal ketika data yang akan dikompres terdiri dari karakter-karakter yang mempunyai panjang kode pendek pada tabel Huffman (tabel 1) begitu juga sebaliknya.

Berdasarkan percobaan yang dilakukan, secara umum algoritma kompresi Huffman kanonik berhasil mereduksi jumlah halaman SMS dengan tingkat keberhasilan mencapai 75% dari seluruh sampel percobaan yang dilakukan. Hal ini terjadi karena pada algoritma Huffman kanonik, karakter yang sering digunakan akan dikodekan dengan panjang kode yang pendek dan pada prakteknya karakter-karakter tersebut sering digunakan oleh user untuk menuliskan pesan SMS.

Algoritma kompresi LZW secara umum kurang berhasil mereduksi jumlah halaman SMS dengan tingkat keberhasilan kurang dari 20%. Berdasarkan hasil percobaan algoritma LZW akan menghasilkan nilai kompresi maksimal jika komposisi data terdiri dari string yang berulang. Hal ini bertolak belakang dengan keadaan sebenarnya dimana pemakaian karakter yang berulang sangat jarang digunakan oleh user. Berdasarkan keadaan tersebut maka algoritma LZW akan lebih baik digunakan jika data yang akan dikirimkan terdiri dari string yang berulang. Namun jika tidak, maka *user* sebaiknya menggunakan algoritma Huffman kanonik.

10. Kesimpulan

Berdasarkan hasil percobaan yang telah dilakukan maka dapat diambil kesimpulan sebagai berikut:

1. Algoritma kompresi Huffman kanonik dan LZW dapat dimanfaatkan untuk melakukan proses kompresi SMS.
2. Algoritma LZW menghasilkan kompresi yang baik ketika data yang dikompres mempunyai variasi simbol yang sedikit dan komposisi string yang banyak berulang. Algoritma kompresi Huffman kanonik akan menghasilkan kompresi yang baik ketika data yang akan dikompres terdiri dari karakter-karakter dengan panjang kode lebih pendek pada tabel Huffman, misal karakter a berdasarkan tabel a.
3. Algoritma kompresi Huffman kanonik mempunyai rasio kompresi rata-rata sebesar 36.02% sedangkan LZW sebesar 21.90%.
4. Kompresi teks SMS dengan menggunakan algoritma Huffman kanonik secara umum mampu mereduksi jumlah halaman SMS yang akan dikirimkan dengan tingkat keberhasilan mencapai 75%. Algoritma LZW secara umum belum mampu mereduksi jumlah halaman SMS yang akan dikirimkan dengan tingkat keberhasilan mencapai 18.37%

DAFTAR PUSTAKA

[1] 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. (home.unpar.ac.id/.../Volume%209/Integral%209%20No.%201/Perbandingan%20Kinerja%20Algoritma%20Kompresi.pdf, diakses terakhir 12 Januari 2008).

[2] Anonim, 2005, *The 7 Bit Default Alphabet*. (http://www.dreamfabric.com/sms/default_alphabet.html, diakses terakhir 17 Mei 2008).

[3] Setiawan, A., Sukanto, T., Nathan, P.S., 2006, *Perancangan dan Pembuatan Sistem Layanan SMS Untuk Biro Administrasi Akademik Universitas Kristen Petra*, Jurnal Informatika, Vol. 7 No. 1, Universitas Kristen Petra, Surabaya. (http://www.petra.ac.id/~puslit/journals/request.php?PublishedID=INF06070103, diakses terakhir 21 Maret 2008).

[4] Liliana, Lipesik, V.J., 2006, *Pembuatan Perangkat Lunak Untuk Kompresi File Text Dengan Menggunakan Huffman Tree*, Fakultas Teknologi Industri Universitas Kristen Petra, Surabaya. (http://fportfolio.petra.ac.id/user_files/03-024/kompresiFile.doc, diakses terakhir 1 Juli 2008)

[5] Nelson, M., 1989, *LZW Data Compression*, USA. (http://marknelson.us/1989/10/01/lzw-data-compression/, diakses terakhir 22 Maret 2008).

[6] Ortiz, C.E., 2005, *The Wireless Messaging API*, Sun Microsystem, Inc. (http://developers.sun.com/mobility/midp/articles/wma2/, diakses terakhir 19 Maret 2008).

[7] Mardiono, T., 2006, Membangun Solusi Mobile Business Dengan Java, Elex Media Komputindo, Jakarta.

[8] Nelson, M., Jean, L.G., 1995, The Data Compression Book Second Edition, IDG Books Worldwide, Inc, Cambridge.