PENERAPAN ALGORITMA LEVENSTEIN PADA APLIKASI KOMPRESI FILE MP3

Tetti Purnama Sari, Surya Darma Nasution, Rivalri Kristianto Hondro

Program Studi Teknik Informatika STMIK Budi Darma, Medan, Indonesia
Email: [1]tettipurnama1610@gmail.com, [2]rivalryhondro@gmail.com

Abstrak

File yang dikompres memerlukan lebih sedikit ruang disk daripada file yang tidak dikompres, jadi mengompres bermanfaat untuk membuat salinan cadangan data dengan pengunaan ruang penyimpanan menjadi kecil atau untuk mengirimkan informasi melalui Internet menjadi lebih cepat. MP3 menjadikan format audio yang sering digunakan karena data yang disimpan menyerupai data yang asli pada saat direkam dan memiliki ukuran yang tidak terlalu besar dibandingkan format lain. Pengguna selain menyimpan file lagu pasti juga menyimpan video dan file-file lainnya dan pengguna juga menginginkan data dengan kualitas terbaik dan kuantitas (ukuran) yang minimum. Maka untuk membuat banyak ruang kosong dan memiliki ukuran data yang tidak besar pada media penyimpanan diperlukan metode kompresi yang memiliki arti untuk mempersingkat ukuran bit yang diperlukan suatu data. Algoritma yang digunakan dalam proses kompresi, diantaranya adalah algoritma Levenstein yang merupakan jenis kompresi lossless. Menerapkan algoritma Levenstein untuk proses kompresi penulis ingin mengetahui kinerja kompresi apabila dilakukan dengan mengkompresi file MP3, sehingga file MP3 yang berukuran besar akan dikompresi menjadi ukuran yang lebih kecil, sehingga proses transmisi yang dilakukan lebih cepat serta memperkecil lokasi penyimpanan data.

Kata kunci: MP3, File, Levenstein

Abstract

Compressed files require less disk space than files that are not compressed, so compressing is useful for backing up data by using storage space to be small or to send information over the Internet faster. MP3 makes the audio format often used because the data stored resembles the original data when recorded and has a size that is not too large compared to other formats. Users besides storing song files, they also store videos and other files and users also want the highest quality data and minimum quantity (size). So to make a lot of empty space and have a size of data that is not large on storage media, a compression method that means to shorten the size of the bits needed for data is needed. Algorithms used in the compression process, including the Levenstein algorithm which is a type of lossless compression. Applying the Levenstein algorithm to the compression process the author wants to know the performance of compression when done by compressing MP3 files, so that large MP3 files will be compressed into smaller sizes, so that the transmission process is carried out faster and reduces the data storage location.

Keywords: MP3, File, Levenstein

1. PENDAHULUAN

Teknologi pemampatan data atau dikenal dengan kompresi data adalah sebuah cara untuk memadatkan data sehingga hanya memerlukan ruangan penyimpanan lebih kecil, sehingga lebih efisien dalam menyimpannya atau mempersingkat waktu pertukaran data tersebut [1]. MPEG (Moving Picture Expert Group)-1 audio layer 3 atau yang lebih dikenal dengan MP3. MP3 Player bisa membuat seseorang mendengarkan lagu secara digital, tidak perlu menggunakan kaset atau CD seperti zaman dahulu dan memungkinkan jutaan orang di seluruh dunia untuk saling bertukar rekaman musik melalui komputer yang terhubung jaringan internet. MP3 format yang biasa dimainkan oleh MP3 Player adalah salah satu format audio yang paling sering digunakan dalam penyimpanan data audio.

MP3 menjadikan format audio yang sering digunakan karena data yang disimpan menyerupai data yang asli pada saat direkam dan memiliki ukuran yang tidak terlalu besar dibandingkan format lain. Untuk masalah kapasitas tentunya sekarang tidak masalah lagi karena kapasitas penyimpanan juga semakin besar, tetapi pengguna selain menyimpan file lagu pasti juga menyimpan video dan file-file lainnya dan pengguna juga menginginkan data dengan kualitas terbaik dan kuantitas (ukuran) yang minimum. Maka untuk membuat banyak ruang kosong dan memiliki ukuran data yang tidak besar pada media penyimpanan diperlukan metode kompresi yang memiliki arti untuk mempersingkat ukuran bit yang diperlukan suatu data. Salah satu penelitian dari Antoni, dkk pada tahun 2014 menganalisa bahwa dengan mengurangi ruang 100% dengan rasio kompresi dalam hal ini menghemat ruang yang dihasilkan oleh Levenstein code. Besarnya kompresi rasio tergantung pada jumlah set karakter dan jumlah kejadian dari masing-masing karakter[2].

Dari hasil penelitian sebelumnya yang dilakukan oleh Darno Willfrid Midukta Simamora, dkk pada tahun 2016 telah berhasil memproses kompresi file audio yang berektensi MP3 dengan menggunakan metode Run Length Encoding (RLE) didasarkan pada kenyataan bahwa pada hampir semua jenis data selalu terdapat pengulangan pada komponen data yang dimilikinya, misalnya di dalam suatu data audio akan terdapat pengulangan penggunaan angka 0 sampai 9 atau huruf a sampai huruf z. Langkah di dalam proses kompresi dengan algoritma Run-Length Encoding (RLE) yaitu dengan terlebih dahulu membuka file MP3 kemudian membaca header-header nilai sampel menggunakan aplikasi hex neo, lalu mengambil sampel MP3 ke-1 sampai ke-n. Dengan menggunakan metode Run Length Encoding (RLE) kualitas MP3 setelah kompresi hanya mengurangi sedikit kapasitas dan tidak mengubah file aslinya[3].

Saat ini banyak algoritma yang digunakan dalam proses kompresi, diantaranya adalah algoritma Levenstein yang merupakan jenis kompresi lossless. Dengan menerapkan algoritma Levenstein penulis ingin mengetahui kinerja kompresi apabila dilakukan dengan mengkompresi file MP3, sehingga file MP3 yang berukuran besar akan dikompresi menjadi ukuran yang lebih kecil, sehingga proses transmisi yang dilakukan lebih cepat serta memperkecil lokasi penyimpanan data..

2. TEORITIS

2.1 Kompresi

Proses kompresi merupakan proses mereduksi ukuran suatu data untuk menghasilkan representasi digital yang padat atau mampat (compact) namun tetap dapat mewakili kuantitas informasi yang terkandung pada data tersebut. Pada citra video, dan audio, kompresi mengarah pada minimalisasi jumlah bit rate untuk representasi digital. Pada beberapa literatur, istilah kompresi sering disebut juga source coding, data compression, bandwidth compression, dan signal compression[4].

Beberapa manfaat kompresi[5] adalah:

1. Waktu pengiriman data pada saluran komunikasi data lebih singkat. Contohnya pengiriman gambar dari fax, video conferencing, handphone, download dari internet, pengiriman data medis, pengiriman dari satelit, dan lain-lain.
2. Membutuhkan ruang memori dalam storage yang lebih sedikit dibandingkan dengan data yang tidak dimampatkan.

Ada dua teknik yang dapat dilakukan dalam melakukan kompresi[5] yaitu:

1. Lossless Compression
   Lossless Compression merupakan kompresi di mana dekompresi dari file yang terkompresi sama dengan file aslinya, tidak ada informasi yang hilang. Sayangnya, rasio kompresi file metode ini sangat rendah. Banyak aplikasi yang memerlukan kompresi tanpa cacat. Seperti pada aplikasi radiografi, kompresi hasil diagnosa medis atau gambar satelit, dimana kehilangan gambar sekecil apa pun akan menyebabkan hasil yang tak diharapkan. Contohnya Run Length Endocing (RLE), Entropy Encoding (huffman, aritmatika), dan Adaptive Dictionary Based (LZW).
2. Lossy Compression
   Lossy compression adalah kompresi file di mana hasil dekompresi dari file yang terkompresi tidak sama dengan file aslinya karena ada informasi yang hilang, tetapi masih bisa ditolerir oleh persepsi mata. Mata tidak dapat membedakan perubahan kecil pada gambar. Metode ini menghasilkan ratio kompresi yang lebih tinggi daripada metode transform coding, seperti transformasi fourier, wavelet dan lain-lain.

Kriteria yang dapat digunakan untuk mengukur pemampatan adalah[4] sebagai berikut:

1. Waktu kompresi dan waktu dekompresi
   Proses kompresi merupakan proses mengkodekan (encode) sehingga diperoleh file dengan representasi kebutuhan memori yang minimum. File data terkompresi disimpan dalam file dengan format tertentu. Sedangkan proses dekompresi adalah proses untuk menguraikan file data yang dimampatkaan untuk dikembalikan lagi (decoding) menjadi file data yang tidak mampat. Algoritma pemampatan yang baik adalah algoritma yang membutuhkan waktu untuk kompresi dan dekompresi paling sedikit (paling cepat).
2. Kebutuhan memori
   Metode kompresi yang baik adalah metode kompresi yang mampu mengompresi file data menjadi file yang berukuran paling minimal. Algoritma pemampatan yang baik akan menghasilkaan memori yang dibutuhkan untuk menyimpan hasil kompresi yang berkurang secara berarti. Biasanya semakin besar persentase pemampatan, semakin kecil kebutuhan memori yang diperlukan sehingga kualitas file data makin berkurang. Dan sebaliknya, semakin kecil presentase file data yang di mampatkan, semakin bagus kualitas hasil pemampatan tersebut.
3. Kualitas Pemampatan (fidelity)
   Metode kompresi yang baik adalah metode kompresi yang mampu mengembalikan file data hasil kompresi menjadi file data tanpa kehilangan informasi apapun. Kalau pun ada informasi yang hilang akibat pemampatan, sebaiknya hal itu ditekan seminimal mungkin. Biasanya semakin berkualitas hasil pemampatan, semakin besar

memori yang dibutuhkan. Sebaliknya, semakin jelek kualitas file data hasil pemampatan, semakin kecil kebutuhan memori yang disediakan.

4. Format Keluaran
   Format file data hasil pemampatan yang baik adalah yang cocok dengan kebutuhan pengiriman dan penyimpanan file.

Rasio kompresi adalah ukuran persentase data yang telah berhasil dimampatkan. Secara matematis rasio pemampatan data dituliskan sebagai berikut.

Rasio=[(hasil kompresi)/(Data Asli) ×100%]

Misalkan rasio kompresi adalah 25% artinya 25% dari data semula telah berhasil dimampatkan[5].

2.2 File MP3

Pada awalnya MPEG Audio Layer-3 banyak dipakai oleh para pengguna komputer. File-file MPEG Audio Layer-3 disimpan dengan ekstensi nama file MP3. Kemudian MPEG Audio Layer-3 selanjutnya banyak dikenal sebagai MP3. File MPEG terdiri dari bagian-bagian kecil yang disebut frame. Biasanya tiap frame dapat berdiri sendiri. Tiap frame memiliki header yang berisi informasi frame tersebut. Pada file MPEG tidak ada header file, karena itu memotong file MPEG bisa dilakukan dimana saja selama masih dalam batasan frame. Lain halnya pada MP3, beberapa frame bisa merupakan bagian yang saling tergantung. Untuk membaca informasi mengenai file MPEG dapat dilakukan dengan membaca header dari frame pertama. Tapi untuk file MPEG yang menggunakan variable bit rate / bitrate switching, informasi dari frame berubah-ubah. Dengan variable bit rate akan didapatkan file yang lebih kecil tanpa menurunkan kualitas suara [7].

2.3 Algoritma Levenstein

Algoritma levenstein code atau levenstein coding merupakan pengkoden universal untuk bilangan bulat non-negtif yang dikembangkan oleh Vladimir Levenshein pada tahun 1968. Algoritma ini tidak banyak diketahui atau kurang terkenal. Algoritma levenstein code merupakan algoritma yang prosesnya melalui tahapan-tahapan tertentu baik pada saat pengkodean maupun pembacaan sandi [6].

Proses pengkodean pada algoritma Levenstein kode. Kode nol pada Levenstein kode adalah satu 0. Untuk kode angka positif n, berikut adalah langkah encode-nya:

1. Atur jumlah varibel C menjadi 1.
2. Tuliskan representasi biner dari nomor tanpa awalan “1” ke kode awal.
3. Misalkan M adalah jumlah bit yang dituliskan pada langkah ke dua.
4. Jika M tidak sama dengan 0, tambahkan C dengan 1. Ulangi langkah ke dua, dengan M dimasukkan sebagai nomor baru.
5. Jika M = 0, tambahkan C “1” bit dan 0 ke awal kode.

Table *levenstein code* yang menunjukkn *18 kode levenstein* dapat dilihat pada tabel dibawah ini.

Tabel 1. Tabel Kode Levenstein

N	***Kode Levenstein***	***N***	***Kode Levenstein***
0	0	9	1110 1 001
1	10	10	1110 1 010
2	110 0	11	1110 1 011
3	110 1	12	1110 1 100
4	1110 0 00	13	1110 1 101
5	1110 0 01	14	1110 1 110
6	1110 0 10	15	1110 1 111
7	1110 0 11	16	11110 0 00 0000
8	1110 1 000	17	11110 0 00 0001

Tabel diatas merupakan daftar beberapa kode dari angka yang telah disandikan. Dari kode tersebut disisipkan spasi untuk menandakan bagian-bagian dari masing-masing kode *levenstein*. Sebagai contoh, dapat dibuktikan kode *levenstein* untuk angka 18 dan 19 adalah 11110 | 0 | 00 | 0010 dan 11110 | 0 | 00 | 0011, dan seterusnya.

Untuk tahapan *decoding* dapat dilakukan sebagai berikut:

1. Lakukan perhitungan pada jumlah bit C “1” sampai “0” ditemukan.
2. Jika hasil perhitungannya adalah C = 0, maka nilainya adalah nol, lalu berhenti. Jika tidak lanjutkan langkah ketiga.
3. Atur N = 1, dan ulangi langkah 4 (C – 1) kali.

Lakukan pembacaan bit N, tambahkan 1, lalu berikan nilai yang dihasilkan ke N (dengan demikian dapat menghilangkan nilai sebelumnya dari N). *String* yang diberikan ke N pada iterasi terakhir merupakan hasil dari pembacaan sandi.

Langkah-langkah kompresi dengan menggunakan algoritma *Levenstein Code* yaitu sebagai berikut[1] :

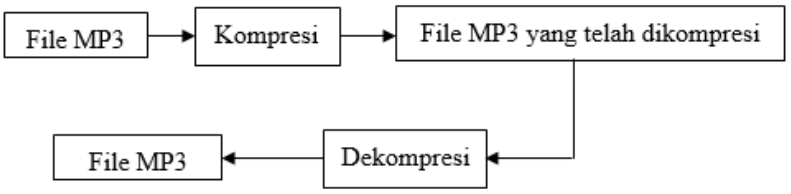
1. Memasukkan *file*.
2. Melakukan pembacaan isi *file*.
3. Mengurutkan dari karakter yang memiliki *frekuensi* terbesar (banyak nilai yang sama) ke terkecil.
4. Membentuk tabel kode *Levenstein*. *File* yang akan dikompresi diganti dengan kode yang terdapat pada tabel kode *Levenstein*. Setelah diganti hitung jumlah bit pada tiap nilai.
5. Melakukan hasil *string bit* kode *Levenstein* menjadi nilai *file*.
   Namun sebelum melakukan hasil *string bit* kode *Levenstein* menjadi nilai *file*, terlebih dahulu dilakukan pemeriksaan terhadap panjang *string bit*. Berikut adalah lagkah-langkah dalam pemeriksaan terhadap panjang *string bit*.
   a. Jika sisa bagi panjang *string bit* terhadap 8 adalah 0 maka tambahkan 00000001. Nyatakan dengan *bit* akhir.
   b. Jika sisa bagi panjang *string bit* terhadap 8 adalah *n* (1, 2, 3, 4, 5, 6, 7) maka tambahkan 0 sebanyak 7 – *n* + "1" di akhir strng bit. Nyatakan dengan L. Lalu tambahkan bilangan biner dari 9 – *n*. Nyatakan dengan *bit* akhir.

   Langkah-langkah dekompresi dengan menggunakan algoritma *Levenstein Code* yaitu sebagai berikut :

1. Memasukkan hasil *file* kompresi.
2. hasil *string bit* kode *Levenstein* yang menjadi nilai *file* dirubah kembali ke bentuk biner.
3. Mengembalikan biner menjadi *string bit* semula dengan melakukan pembacaan pada 8 *bit* terakhir, hasil pembacaan berupa bilangan desimal. Nyatakan hasil pembacaan dengan *n* lalu hilangkan *bit* pada bagian akhir sebanyak 7 + *n*.

3. ANALISA

Tahapan analisa terhadap kompresi dan dekompresi file MP3 pada penelitian ini dapat dilihat pada gambar berikut:



Gambar 1. Diagram Kompresi Dekompresi File MP3

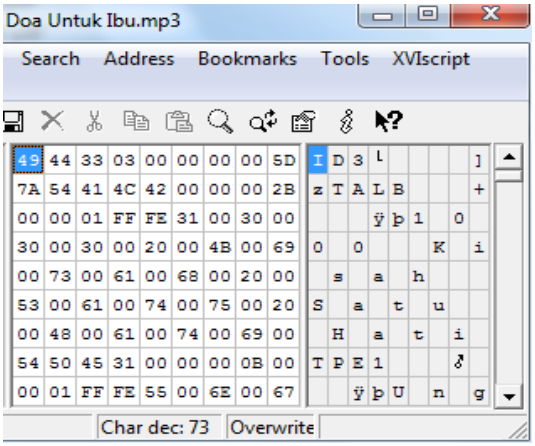
Keluaran dari hasil kompresi adalah sebuah file MP3 yang ukuranya kapasitas telah berubah dari kapasitas awal. Proses dekompresi adalah proses pengembalian pada kapasitas awal file MP3.

Melakukan kompresi file MP3 sebelumnya harus dilakukan analisa terhadap file MP3 yang akan dikompresi. Menganalisa file MP3 yang harus dilakukan adalah mengambil sample file MP3 dengan melakukan pembacaan file MP3. Pembacaan file MP3 dilakukan untuk mendapatkan nilai dari data pada sebuah file MP3 yang berupa bilangan hexadesimal. Proses hitungan manual nilai sample file MP3 diambil menggunakan sofware Hex Editor, kemudian data nilai hexadesimal MP3 di kompresi menggunakan algoritma Levenstein. Berikut informasi objek file MP3 yang akan diambil samplenya sebelum dilakukan kompresi.

Tabel 2. Informasi *File* MP3 *Sample*

Keterangan	
Jenis File	.MP3
Judul	Doa Untuk Ibu
Ukuran	4.524 KB
Durasi	4.48 Menit

1. Kompresi Berdasarkan Algoritma *Levenstein*
   a. Memasukkan *file*
      Dari *sample* MP3 di dapat nilai *hexedesimal* menggunakan bantuan *software hex editor* seperti pada gambar di bawah ini:



Gambar 2. Nilai *Hexadesimal File* MP3 *Sample*

Berdasarkan pada gambar di atas di dapati nilai *hexadesimal file* MP3 *sample*. Untuk keperluan hitungan manual hanya diambil *sample* nilai sebanyak 16 karakter nilai *hexadesimal file* MP3 *sample*. Nilai *hexadesimal* diambil dari sisi kiri atas sampai dengan bilangan ke 16.

b. Melakukan pembacaan isi *file*
Adapun bilangan *hexadesimal file* MP3 *sample* tersebut adalah 49, 44, 33, 03, 00, 00, 00, 00, 5D, 7A, 54, 41, 4C, 42, 00, 00.

c. Mengurutkan dari karakter yang memiliki *frekuensi* terbesar (banyak nilai yang sama) ke terkecil. Urutan bilangan *hexadesimal* dapat dilihat pada tabel di bawah ini :

Tabel 3. Nilai *Bit File* MP3 *Sample*

Nilai		*Bit*	*Frek*	*Bit x Frek*
Hex	*Biner*			
00	00000000	8	6	48
49	01001001	8	1	8
44	01000100	8	1	8
33	00110011	8	1	8
03	00000011	8	1	8
5D	01011101	8	1	8
7A	01111010	8	1	8
54	01010100	8	1	8
41	01000001	8	1	8
4C	01001100	8	1	8
42	01000010	8	1	8
Total				128 *bit*

Berdasarkan tabel di atas, satu nilai *hexadesimal* (karakter) bernilai delapan *bit* bilangan *biner*. Sehingga 16 bilangan *hexadesimal* mempunyai nilai *biner* sebanyak 128 *bit*. Untuk mengubah satuan menjadi *byte* maka jumlah keseluruhan bit dibagikan 8. Maka dapat dihasilkan 128 / 8 = 16 *byte*

d. Membentuk tabel kode *Levenstein*
File yang akan dikompresi diurutkan dengan kode *Levenstein* sesuai dengan tabel kode yang terdapat pada tabel 2.1. Setelah diurutkan hitung jumlah *bit* pada tiap nilai *hexadesimal.*

Tabel 4. Kompresi Nilai MP3 *Sample* Dengan Nilai *Levenstein Code*

N	*Nilai Hex*	*Levenstein Code*	*Bit*	*Frek*	*Bit x Frek*
0	00	0	1	6	6
1	49	10	2	1	2
2	44	110 0	4	1	4
3	33	110 1	4	1	4
4	03	1110 0 00	7	1	7
5	5D	1110 0 01	7	1	7
6	7A	1110 0 10	7	1	7

N	*Nilai Hex*	*Levenstein Code*	*Bit*	*Frek*	*Bit x Frek*
7	54	1110 0 11	7	1	7
8	41	1110 1 000	8	1	8
9	4C	1110 1 001	8	1	8
10	42	1110 1 010	8	1	8
Jumlah					68 *bit*

Dari perhitungan tabel diatas setelah dikompresi dengan menggunakan *Levenstein code* adalah 68 *bit*. Untuk diubah menjadi satuan *byte* maka dibagi 8. 68 / 8 = 8,5 *byte*.

e. Melakukan hasil *string bit* kode *Levenstein* menjadi nilai *file*.
Sebelum melakukan hasil *string bit* kode *Levenstein* menjadi nilai *file*, terlebih dahulu dilakukan pemeriksaan terhadap panjang *string bit.*
Pembentukan nilai *bit* baru hasil kompresi dari susunan nilai *hexadesimal* sebelum dikompresi yaitu 49, 44, 33, 03, 00, 00, 00, 00, 5D, 7A, 54, 41, 4C, 42, 00, 00 (tanpa tanda koma dan spasi) menjadi nilai *bit biner"*1011001101111000000001110001111001011100111111010001110100111101010000*"*. Melakukan pemeriksaan terhadap panjang *string bit*. Jika sisa bagi panjang *string bit* terhadap 8 adalah 0 maka tambahkan 00000001 dan nyatakan dengan *bit* akhir. Jika sisa bagi panjang *string bit* terhadap 8 adalah *n* (1, 2, 3, 4, 5, 6, 7) maka tambahkan 0 sebanyak 7 – *n* + "1" di akhir *string bit*, nyatakan dengan L. Lalu tambahkan bilangan *biner* dari 9 – *n*. Nyatakan dengan *bit* akhir. Total jumlah bit yang telah dikompresi adalah 68 *bit*, maka *N* = *MOD* (68,8) = 4, dimana L = 7 – 4 + "1" = 3, dari *bit* menjadi **0001**. *Bit* Akhir = 9 – 4 = 5, nilai binernya menjadi **00000101**. Sehingga *string bit* yang terbentuk adalah :
"1011001101111000000001110001111001011100111111010001110100111101010000**000100000101**"
Total *bit* sebelumnya setelah penambahan pada *string bit* kode *Levenstein* adalah 68+4+8= 80.
Hasil *string bit* kode *Levenstein* dipisah menjadi 8 bagian agar dapat diubah ke dalam nilai *hexadesimal* yaitu dapat dilihat pada tabel berikut:

Tabel 5. Nilai *Biner* dan *Hexadesimal* hasil kompresi keseluruhan

Biner	*Hexa*
10110011	B3
01111000	78
00000111	7
00011110	1E
01011100	5C
11111010	FA
00111010	3A
01111010	7A
10000001	81
00000101	5

Persentase ukuran data yang telah dikompresi dan didapat dari hasil perbandingan antara ukuran data setelah dikompresi dengan ukuran data sebelum dikompresi.

$$ukuran\ data\ sebelum\ dikompresi = 128$$
$$ukuran\ data\ setelah\ dikompresi = 80$$
$$Compression\ Ratio(Cr) = \frac{Ukuran\ data\ setelah\ dikompresi}{Ukuran\ data\ sebelum\ dikompresi}\ X\ 100\%$$
$$= \frac{80}{128}\ X\ 100\% = 62{,}5\%$$

2. Dekompresi Berdasarkan Algoritma *Levenstein*

a. Memasukkan hasil *file* kompresi
Adapun nilai *hexadesimal* dari hasil kompresi yaitu : B3, 78, 7, 1E, 5C, FA, 3A, 7A, 81, 5.

b. hasil *string bit* kode *Levenstein* yang menjadi nilai *file* dirubah kembali ke bentuk biner, yaitu :
"1011001101111000000001110001111001011100111111010001110100111101010000000100000101"

c. Mengembalikan biner menjadi *string bit* semula dengan melakukan pembacaan pada 8 *bit* terakhir, hasil pembacaan berupa bilangan desimal. Nyatakan hasil pembacaan dengan *n* lalu hilangkan *bit* pada bagian akhir sebanyak 7 + *n*. Hasil pengembalian *biner* menjadi *string bit* semula :
10110011011110000000011100011110010111001111110100011101001111010100000
11111010001110100111101010000**00010000010**1, 7 + *n* → 7 + 5 = 12

Desimal (n) = 5

Selanjutnya hapus 12 *bit* keseluruhan dari *bit string* terakhir sehingga menjadi seperti :
"10110011011110000000011100011110010111001111101000111010011110101000"
Selanjutnya adalah melakukan cek *bit* dari *bit* pertama dengan tabel kode *Levenstein* pada tabel 3.3 di atas. Jika ditemukan *bit* yang sesuai dengan tabel kode *Levenstein* di atas maka ubah nilai *string* yang sesuai. Sehingga didapati hasil seperti tabel di bawah ini :

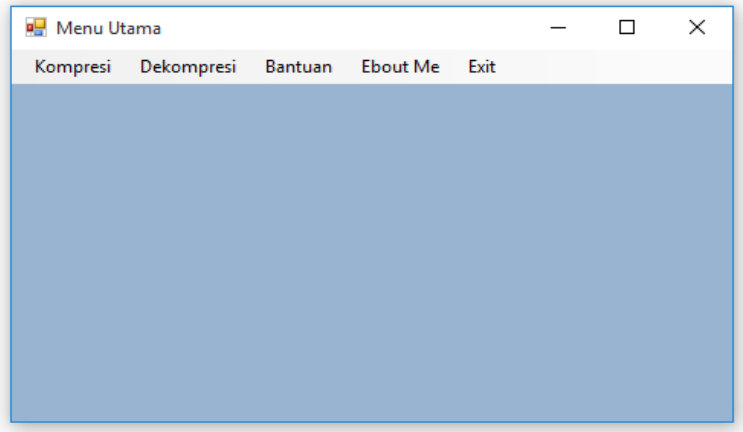
Tabel 6. Hasil Dekompresi Menggunakan *Levenstein Code*

Levenstein Code	Nilai *Hex*
10	49
1100	44
1101	33
1110000	03
0	00
0	00
0	00
0	00
1110001	5D
1110010	7A
1110011	54
11101000	41
11101001	4C
11101010	42
0	00
0	00

Berdasarkan hasil dekompresi di atas didapati nilai *hexadesimal* awal sebelum kompresi sebagai berikut, 49, 44, 33, 03, 00, 00, 00, 00, 5D, 7A, 54, 41, 4C, 42, 00, 00.

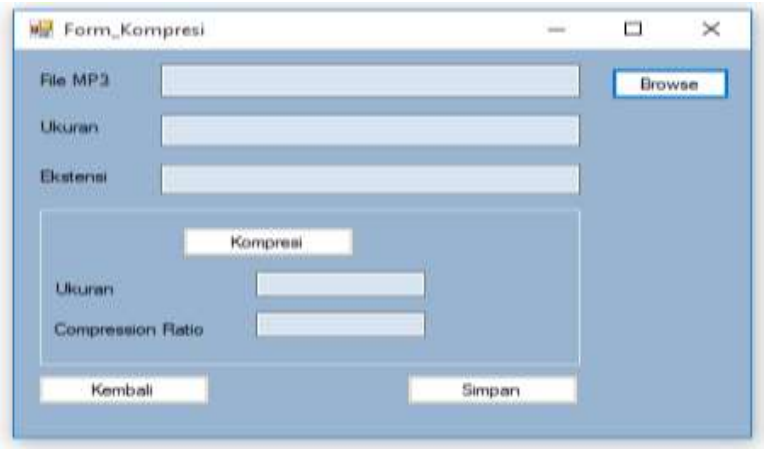
4. IMPLEMENTASI

1. Tampilan Form Utama



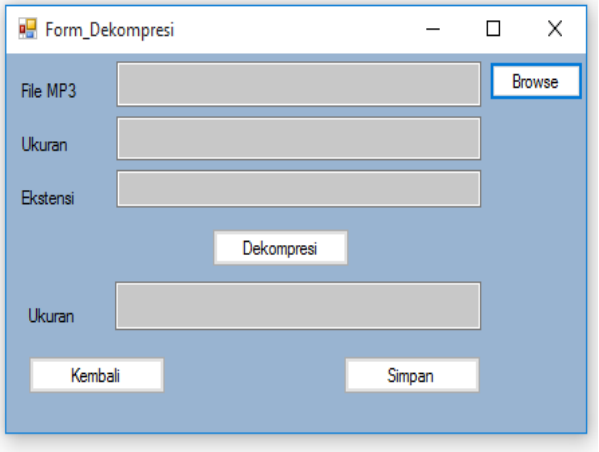
Gambar 3. Tampilan Antar Muka Form Menu Utama

2. Tampilan Form Kompresi



Gambar 4. Tampilan Antar Muka Form Kompresi

3. Tampilan Form Dekompresi



Gambar 5. Tampilan Antar Muka Form Dekompresi

Hasil pengujian terdiri dari hasil pengujian kompresi dan hasil pengujian dekompresi. Hasil pengujian berupa perbedaan antara *file* MP3 sebelum dan sesudah kompresi serta dekompresi.

1. Hasil Pengujian Kompresi *File* MP3
   Adapun hasil pengujian dari *file* MP3 yang dilakukan kompresi dapat dilihat pada tabel di bawah ini :

Tabel 7. Hasil Pengujian Kompresi *File* MP3

No.	Sebelum Kompresi			Sesudah Kompresi		
	Nama File MP3	Ukuran	Ekstensi	Nama File MP3	Ukuran	CR
1.	Ungu - Doa Untuk Ibu	4.524 MB	.mp3	Ungu - Doa Untuk Ibu	3.072 MB	67,9 %
2.	Seventeen-Ayah	4.36 MB	.mp3	Seventeen-Ayah	3.11 MB	71,3 %
3.	Melly Goeslaw - Gantung	3.41 MB	.mp3	Melly Goeslaw – Gantung	2.13 MB	62,4 %
4.	Opick - Rapuh	4.11 MB	.mp3	Opick-Rapuh	2.81 MB	68,3 %
5.	Utopia – Benci	3.93 MB	.mp3	Utopia - Benci	2.43 MB	61,8 %

Berdasarkan pada hasil pengujian pada tabel di atas, menunjukan bahwa hasil kompresi *file* MP3 memiliki perbedaan ukuran ketika *file* MP3 sebelum kompresi. Hal ini dapat dilihat dari *Compression Ratio*. Kecepatan dari kompresi *file* MP3 dipengaruhi oleh banyaknya ukuran awal.

2. Hasil Pengujian Dekompresi *File* MP3
   Adapun hasil pengujian dekompresi dari *file* MP3 yang sudah dilakukan kompresi dapat dilihat pada tabel di bawah ini:

Tabel 8. Hasil Pengujian Dekompresi *File* MP3

No	Sebelum Dekompresi			Sesudah Dekompresi	
	Nama File MP3	Ukuran	Ekstensi	Nama File MP3	Ukuran
1.	Ungu – Doa Untuk Ibu	3.072 MB	.mp3	Ungu – Doa Untuk Ibu	4.524 MB
2.	Seventeen - Ayah	3.11 MB	.mp3	Seventeen - Ayah	4.36 MB
3.	Melly Goeslaw - Gantung	2.13 MB	.mp3	Melly Goeslaw - Gantung	3.41 MB
4.	Opick - Rapuh	2.81 MB	.mp3	Opick - Rapuh	4.11 MB
5.	Utopia – Benci	2.43 MB	.mp3	Utopia – Benci	3.93 MB

Berdasarkan pada hasil pengujian pada tabel di atas, menunjukan bahwa hasil dekompresi mengembalikan ukuran *file* MP3 menjadi ukuran awal sebelum terjadinya kompresi. Kecepatan dari dekompresi *file* MP3 dipengaruhi oleh banyaknya ukuran kompresi.

5. DAFTAR PUSTAKA

[1] Shmilovici A.; Kahiri Y.; Ben-Gal I.; Hauser S. "Measuring the Efficiency of the Intraday Forex Market with a Universal Data Compression Algorithm" (PDF). Computational Economics, Vol. 33 (2), 131-154., 2009 Antoni, E. B. Nababan, and M. Zarlis, “Result Analysis of Text Data Compression On Elias Gamma Code, Elias Delta Code and Levenstein Code,” vol. 4, no. 2221–8386, 2014.
[2] D. Willfrid, M. Simamora, G. Ginting, and Y. Hasan, “IMPLEMENTASI ALGORITMA RUN LENGTH ENCODING PADA KOMPRESI FILE MP3,” vol. 3, no. 4, pp. 5–9, 2016.
[3] Darma Putra, Pengolahan Citra Digital. Yogyakarta: Andi, 2010.
[4] D. T. Sutoyo, S.Si, Teori pengolah citra digital. Yogyakarta: Andi, 2009.
[5] David Salomon and G. Motta, Handbook of Data Compression. 2010.
[6] F. Release and D. W. I. Hartanto, “Sekelumit tentang format MP3 dan MP3 Player,” pp. 3–6, 2005.
[7] Rosa A.S and M.Shalahuddin, Rekayasa Perangkat Lunak (Terstruktur dan Berorientasi Objek). Bandung: Informatika Bandung, 2016.
[8] Hendrayudi, Dasar-Dasar Pemrograman Microsoft Visual Basic 2008. Bandung: PT.SARANA TUTORIAL NURANI SEJAHTERA, 2010.
[9] Rachmad Hakim S, Visual Basic 2008 for Pemula Banget. Jakarta: PT Elex Media Komputindo, 2009.