

IMPLEMENTASI ALGORITMA *LEVENSTEIN* UNTUK KOMPRESI *FILE* VIDEO PADA APLIKASI *CHATTING* BERBASIS ANDROID

Dani Iqbal

Program Studi Teknik Informatika STMIK Budi Darma, Medan, Indonesia
Email : idniqbal@gmail.com

Abstrak

Teknologi berkembang sangat pesat seiring dengan kebutuhan masyarakat dalam memperoleh informasi secara cepat. Aplikasi *chatting* sebagai alat komunikasi yang dapat membantu berkomunikasi tanpa ada batasan waktu dan bebas dari hambatan. Tetapi dalam berkomunikasi melalui aplikasi *chatting* tidak bisa leluasa atau secepat dalam berkomunikasi pada umumnya. Dimana aplikasi *chatting* memiliki jumlah bit yang telah ditetapkan, sehingga dalam pengiriman video pada aplikasi *chatting* tidak dapat sekaligus melainkan dalam beberapa kali pengiriman. Adapun Solusi dalam permasalahan ini adalah bagaimana video tersebut dapat dikompresi guna untuk mempercepat proses pengiriman dan penyimpanan *file* video. kompresi pada *file* video dilakukan dengan memperkecil ukuran video dengan proses mengurangi bit pada video, akan tetapi tidak menghilangkan data informasi didalamnya. Dalam penelitian ini, algoritma yang digunakan adalah *Levenstein*, dengan menggunakan metode tersebut, hasil kompresi dari nilai *Levenstein* mempunyai hasil yang berbeda-beda dari setiap nilainya, dan hasil kompresi akan menguntungkan dalam melakukan pengiriman dan pemindahan *file* video.

Kata Kunci : *Kompresi Video, Algoritma Levenstein, Aplikasi Chatting*

Abstract

Technology is developing very rapidly along with the needs of the community in obtaining information quickly. Chat application as a communication tool that can help communicate without any time restrictions and is free from obstacles. But in communicating through chat applications can not be free or free in communicating in general. Where the chat application has a predetermined number of bits, so that in sending video on the chat application can not be at once but in several times the transmission. The solution to this problem is how the video can be compressed in order to speed up the process of sending and storing video files. compression on video files is done by reducing the size of the video by the process of reducing the bits in the video, but does not eliminate the data information in it. In this study, the algorithm used is *Levenstein*, using this method, the compression results from the *Levenstein* value have different results from each value, and the compression results will be advantageous in sending and transferring video files.

Keywords: Video Compression, Levenstein Algorithm, Chatting Application

1. PENDAHULUAN

Pada era revolusi industri 4.0 teknologi berkembang sangat pesat seiring dengan kebutuhan masyarakat dalam memperoleh informasi secara cepat. Sangat banyak media yang dapat mempercepat informasi di era sekarang ini. Salah satunya aplikasi *chatting* yang memegang peranan penting terhadap komunikasi di kehidupan manusia. Aplikasi *chatting* sebagai alat komunikasi yang dapat membantu berkomunikasi tanpa ada batasan waktu dan bebas dari hambatan. Tetapi dalam berkomunikasi melalui aplikasi *chatting* tidak bisa leluasa atau secepat dalam berkomunikasi pada umumnya. Dimana aplikasi *chatting* memiliki jumlah bit yang telah ditetapkan, sehingga dalam pengiriman *file* pada aplikasi *chatting* tidak dapat sekaligus atau dalam sekali pengiriman melainkan dalam beberapa kali pengiriman.

Dalam melakukan pengiriman *file* video pada aplikasi *chatting* sering terjadi masalah yaitu ukuran video yang terlalu besar sehingga *file* video yang akan dikirim tidak dapat dikirimkan kepada sipenerima. Untuk mengatasi masalah tersebut dapat menggunakan teknik kompresi untuk mengurangi ukurannya. Salah satu kegunaan teknik kompresi adalah untuk memperkecil kapasitas ruang dalam memori media penyimpanan, agar data didalam ruang penyimpanan tersebut dapat ditata dengan baik. Masalah tersebut dapat diatasi dengan melakukan proses teknik kompresi *file* video yang besar menjadi ukuran yang kecil atau bisa mengurangi ukuran bit yang terdapat pada setiap *file* video sehingga dapat menghemat wadah penyimpanan dan mempermudah proses pengiriman *file* video melalui aplikasi *chatting*.

Pada penelitian terdahulu yang dilakukan oleh Rizky Syahputra Pada Tahun 2016 telah menyimpulkan bahwa penelitian terhadap *file* video di perlukan untuk memperkecil ukuran *file* video sehingga mempermudah proses penyimpanan dan pengiraman *file* video melalui aplikasi *chatting* [1].

Pada penelitian terdahulu yang dilakukan oleh Tetti Purnama Sari, Surya Darma Nasution, Rivalri Kristianto Hondro melakukan penelitian terhadap algoritma *Levenstein* Berdasarkan pada hasil pengujian pada tersebut, menunjukan bahwa hasil dekompresi mengembalikan ukuran *file* menjadi ukuran awal sebelum terjadinya kompresi. Kecepatan dari dekompresi *file* dipengaruhi oleh banyaknya ukuran kompresi. [2].

Algoritma *levenstein code* atau *levenstein coding* merupakan satu jenis kompresi *Lossless* dimana dekompresi dari *file* yang terkompresi masih sama dengan *file* aslinya, tidak ada data yang hilang. Algoritma *levenstein code* merupakan algoritma yang prosesnya melalui tahapan-tahapan yang sudah ditentukan baik pada saat pengkodean maupun pembacaan sandi [3]. Dengan menerapkan algoritma *Levenstein* diharapkan dapat digunakan untuk mengkompresi *file* video, sehingga *file* video yang berukuran besar akan dikompresi menjadi ukuran yang lebih kecil dan proses pengiriman *file* video pada aplikasi *chatting* dapat dilakukan lebih cepat.

2. TEORITIS

2.1 Kompresi

Proses kompresi yaitu proses pengurangan ukuran data untuk menghasilkan suatu data digital yang padat atau mampat namun tidak mengurangi kualitas informasi yang ada pada data tersebut. Pada beberapa data digital seperti citra, audio, dan video, kompresi mengarah pada mengurangi jumlah *bit rate* untuk memampatkan data digital. Pada beberapa literatur, istilah kompresi sering disebut dengan *source coding*, *data compression*, *bandwidth compression*, dan *signal compression* [4]. Contoh kompresi sederhana yang biasa dilakukan misalnya adalah dengan menyingkat kata-kata pada pesan singkat untuk dikirimkan kepada orang lain.

Tujuan kompresi data adalah untuk mengurangi data yang berlebihan sehingga ukuran data dapat menjadi lebih kecil dan lebih mudah dalam melakukan proses pengiriman.

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

$$\text{Rasio} = \left[\frac{\text{hasil kompresi}}{\text{Data Asli}} \times 100\% \right]$$

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

2.2 Dekompresi

Dekompresi merupakan proses mengembalikan sebuah data yang sudah terkompresi. Sebuah data yang sudah dikompres tentunya harus dapat dikembalikan lagi ke bentuk aslinya. Untuk dapat merubah data yang terkompres diperlukan cara yang berbeda seperti pada waktu proses kompres dilaksanakan. Jadi pada saat dekompres terdapat catatan *header* yang berupa *byte-byte* yang berisi catatan mengenai isi dari *file* tersebut [6]. Data yang telah dikompresi, harus dikembalikan seperti semula untuk kemudian dibaca kembali, proses pengembalian ini disebut sebagai proses dekompresi. Sama seperti kompresi, dekompresi dibagi menjadi dua, yaitu *lossy* dimana *file* yang dihasilkan, tidak sama persis seperti sebelum *file* dikompres, dan *lossless* dimana *file* yang dihasilkan akan sama persis seperti *file* sebelum dikompres[7].

2.3 Aplikasi Chatting

Aplikasi *Chatting* merupakan suatu perangkat lunak (*software*) yang memfasilitasi pengiriman pesan singkat antara dua *user* atau lebih. Salah satu kelebihan dari aplikasi *chatting* adalah kemampuannya memungkinkan pengguna untuk bertukar pesan baik dalam jenis teks, video, audio atau gambar [8]. Banyak fitur dari Aplikasi *Chatting* yang dikembangkan pada saat ini, tidak hanya berfungsi sebagai pengiriman teks saja tetapi sudah memiliki fitur seperti, *voice call* dan *video call* yang bisa dilakukan secara bersama-sama. Fitur dari Aplikasi *Chatting* juga memungkinkan pengguna untuk menyimpan daftar orang yang dapat diajak berkomunikasi secara mudah dan cepat.

2.4 File MPEG (Moving Picture Expert Group)-4 (MP4)

Pada awalnya MPEG Video Layer-4 banyak dipakai oleh para pengguna komputer. *File-file* MPEG Video Layer-4 disimpan dengan ekstensi nama *file* MP4. Kemudian MPEG Audio Layer-4 selanjutnya banyak dikenal sebagai MP4. *File* MPEG terdiri dari 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.

2.5 Android

Android merupakan suatu sistem operasi *mobile* yang berbasis pada sistem operasi *linux*. Android menawarkan pendekatan yang menyeluruh dalam pengembangan aplikasi. Artinya, satu aplikasi android yang dibangun dapat berjalan di berbagai perangkat yang menggunakan sistem operasi Android baik itu *smartphone*, *smartwatch*, *tablet*, dan perangkat lainnya. Perkembangan teknologi Android yang begitu cepat juga tidak dapat dilepas dari peranan AOSP (*Android Open Source Project*) yang bertanggung jawab dalam pengembangan sistem operasi Android dan dipimpin langsung oleh Google [9].

2.6 Algoritma Levenstein

Algoritma *levenstein code* atau *levenstein coding* merupakan pengkodean *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[10].

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 variabel 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 menunjukkan 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 [11] :

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 langkah-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 *string 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.

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 DAN PEMBAHASAN

Dalam melakukan kompresi *file* MP4 sebelumnya harus dilakukan analisa terhadap *file* MP4 yang akan dikompresi. *File* MP4 merupakan *file* yang sangat dikenal dan paling populer diantara *file* yang lainnya. Dalam menganalisa *file* MP4 yang harus dilakukan adalah mengambil *sample file* MP4 dengan melakukan pembacaan *file* MP4. Pembacaan *file*

MP4 dilakukan untuk mendapatkan nilai dari data pada sebuah file MP4 yang berupa bilangan *hexadesimal*. Dalam proses hitungan manual nilai *sample file* MP4 diambil menggunakan *software Binary Viewer*, kemudian data nilai *hexadesimal* MP4 di kompresi menggunakan algoritma *Levenstein*. Berikut informasi objek file MP4 yang akan diambil *sampelnya* sebelum dilakukan kompresi.

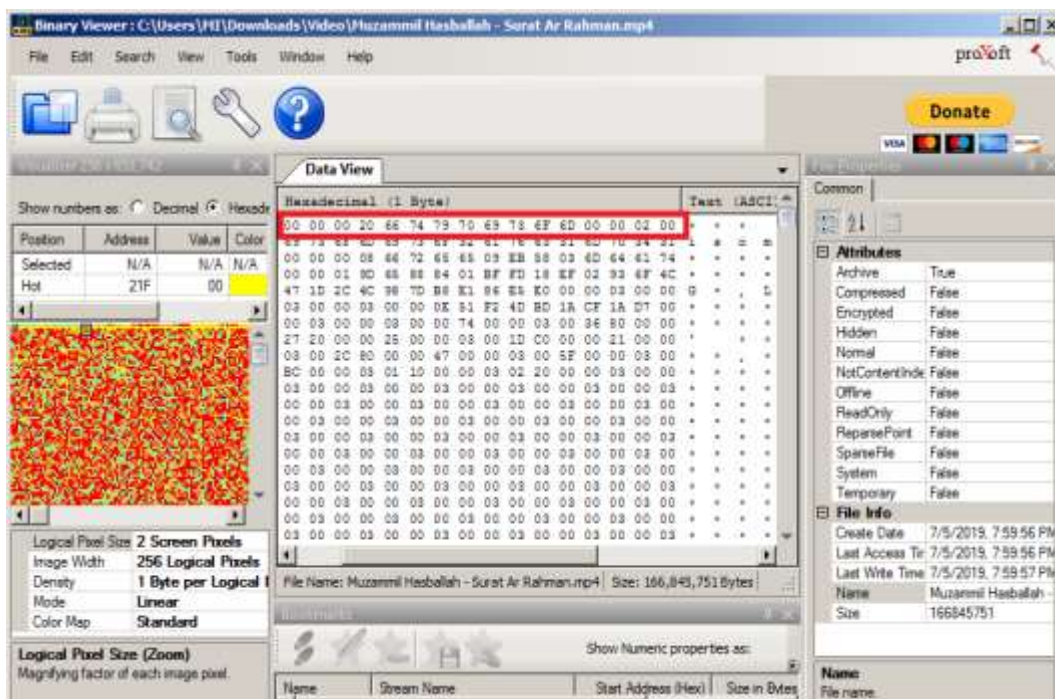
Tabel 2. Informasi File MP4 Sample

Keterangan	
Jenis File	.MP4
Judul	Surat Ar Rahman
Ukuran	159 MB
Durasi	16.15 Menit

1. Kompresi Berdasarkan Algoritma Levenstein

a. Memasukkan file

Dari *sample* MP4 di dapat nilai *hexadesimal* menggunakan bantuan *software binary viewer* seperti pada gambar di bawah ini:



Gambar 1. Nilai Hexadesimal File MP4 Sample

Berdasarkan pada gambar di atas di dapat nilai *hexadesimal file* MP4 *sample*. Untuk keperluan hitungan manual hanya diambil *sample* nilai sebanyak 16 karakter nilai *hexadesimal file* MP4 *sample*. Nilai *hexadesimal* diambil dari sisi kiri atas sampai dengan bilangan ke 16.

b. Melakukan pembacaan isi file

Adapun bilangan *hexadesimal file* MP4 *sample* tersebut adalah 00, 00, 00, 20, 66, 74, 79, 70, 69, 73, 6F, 6D, 00, 00, 02, 00.

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 MP4 Sample

Nilai		Bit	Frek	Bit x Frek
Hex	Biner			
00	00000000	8	6	48
20	00100000	8	1	8
66	01100110	8	1	8
74	01110100	8	1	8
79	01111001	8	1	8
70	01110000	8	1	8
69	01101001	8	1	8

73	01110011	8	1	8
6F	01101111	8	1	8
6D	01101101	8	1	8
02	00000010	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*

c. Membentuk tabel kode *Levenstein*

File yang akan dikompresi diurutkan dengan kode *Levenstein* sesuai dengan tabel kode yang terdapat pada tabel 4.. Setelah diurutkan hitung jumlah *bit* pada tiap nilai *hexadesimal*.

Tabel 4. Kompresi Nilai MP4 Sample Dengan Nilai *Levenstein*

<i>N</i>	Nilai Hex	<i>Levenstein Code</i>	<i>Bit</i>	<i>Frek</i>	<i>Bit x Frek</i>
0	00	0	1	6	6
1	20	10	2	1	2
2	66	110 0	4	1	4
3	74	110 1	4	1	4
4	79	1110 0 00	7	1	7
5	70	1110 0 01	7	1	7
6	69	1110 0 10	7	1	7
7	73	1110 0 11	7	1	7
8	6F	1110 1 000	8	1	8
9	6D	1110 1 001	8	1	8
10	02	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*.

d. 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 00, 00, 00, 20, 66, 74, 79, 70, 69, 73, 6F, 6D, 00, 00, 02, 00 (tanpa tanda koma dan spasi) menjadi nilai *bit biner* "00010110011011110000111000111100101110011111010001110100100111010100". 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 :

"00010110011011110000111000111100101110011111010001110100100111010100000100000101"

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

<i>Biner</i>	<i>Hexa</i>
00010110	16
01101111	6F
00001110	E
00111100	3C
10111001	B9
11110100	F4
01110100	74
10011101	9D
01000001	41
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

$$\text{Compression Ratio}(Cr) = \frac{\text{Ukuran data setelah dikompresi}}{\text{Ukuran data sebelum dikompresi}} \times 100\%$$

$$= \frac{80}{128} \times 100\% = 62,5\%$$

3.1 Proses Dekompresi

1. Memasukkan hasil *file* kompresi

Adapun nilai *hexadesimal* dari hasil kompresi yaitu : 16, 6F, E, 3C, B9, F4, 74, 9D, **41, 5**.

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

“00010110011011110000111000111100101110011111010001110100100111010100**000100000101**”

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 $7 + n$. Hasil pengembalian *biner* menjadi *string bit* semula :

00010110011011110000111000111100101110011111010001110100100111010100 **000100000101** $7 + n$ $7 + 5 = 12 \rightarrow$

Desimal (*n*) = 5

Selanjutnya hapus 12 *bit* keseluruhan dari *bit string* terakhir sehingga menjadi seperti :

“00010110011011110000111000111100101110011111010001110100100111010100”

Selanjutnya adalah melakukan cek *bit* dari *bit* pertama dengan tabel kode *Levenstein* pada tabel 6. di atas. Jika ditemukan *bit* yang sesuai dengan tabel kode *Levenstein* di atas maka ubah nilai *string* yang sesuai. Sehingga mendapatkan hasil seperti tabel di bawah ini :

Tabel 6. Hasil Dekompresi Menggunakan *Levenstein Code*

<i>Levenstein Code</i>	Nilai Hex
0	00
0	00
0	00
10	20
110 0	66
110 1	74
1110 0 00	79
1110 0 01	70
1110 0 10	69
1110 0 11	73
1110 1 000	6F
1110 1 001	6D
0	00
0	00
1110 1 010	02
0	00

Berdasarkan hasil dekomposisi di atas didapati nilai *hexadesimal* awal sebelum kompresi sebagai berikut, 00, 00, 00, 20, 66, 74, 79, 70, 69, 73, 6F, 6D, 00, 00, 02, 00.

4. IMPLEMENTASI

Pengujian sistem merupakan tahap mengidentifikasi hasil dari implementasi sistem aplikasi. Hasil pengujian terdiri dari hasil pengujian kompresi dan hasil pengujian dekomposisi. Hasil pengujian berupa perbedaan antara *file* MP4 sebelum dan sesudah kompresi serta dekomposisi.



Gambar 2. Form Pilih Video



Gambar 3. Form hasil pengiriman Kompresi

1. Hasil Pengujian Kompresi File MP4

Adapun hasil pengujian dari file MP4 yang dilakukan kompresi dapat dilihat pada tabel di bawah ini :

Tabel 1. Hasil Pengujian Kompresi File MP4

No.	Nama File MP4	Sebelum Kompresi		Sesudah Kompresi	
		Ukuran	Ekstensi	Nama File MP4	Ukuran
1.	Muzamil – Surah Ar-Rahman	159 MB	.MP4	Muzamil – Surah Ar-Rahman	146 MB
2.	Surga yang Tak di Rindukan (2015)	278 MB	.MP4	Surga yang Tak di Rindukan (2015)	252 MB
3.	Thank You Cinta 2014	299 MB	.MP4	Thank You Cinta 2014	279 MB

Berdasarkan pada hasil pengujian pada tabel di atas, menunjukkan bahwa hasil kompresi file MP4 memiliki perbedaan ukuran ketika file MP4 sebelum kompresi. Hal ini dapat dilihat dari *Compression Ratio*. Kecepatan dari kompresi file MP4 dipengaruhi oleh banyaknya ukuran awal.

2. Hasil Pengujian Dekompresi File MP4

Adapun hasil pengujian dekompresi dari file MP4 yang sudah dilakukan kompresi dapat dilihat pada tabel di bawah ini :

Tabel 2. Hasil Pengujian Dekompresi File MP4

No	Sebelum Dekompresi			Sesudah Dekompresi	
	Nama File MP4	Ukuran	Ekstensi	Nama File MP4	Ukuran
1.	Muzamil – Surah Ar-Rahman	146 MB	.MP4	Muzamil – Surah Ar-Rahman	159 MB
2.	Surga yang Tak di Rindukan (2015)	252 MB	.MP4	Surga yang Tak di Rindukan (2015)	278 MB
3.	Thank You Cinta 2014	279 MB	.MP4	Thank You Cinta 2014	299 MB

5. KESIMPULAN

Kesimpulan yang bisa diambil setelah melakukan kompresi file video dengan menerapkan algoritma *Levenstein Coding* adalah :

1. Prosedur dalam mengkompresi file MP4 dapat dilakukan dengan mencari nilai *hexadesimal* file MP4 dan kemudian diterapkan dengan menggunakan algoritma *Levenstein*.
2. Proses penerapan algoritma *Levenstein* dalam mengkompresi file MP4 memberikan hasil rasio kompresi hampir 15% sehingga dapat menghemat ruang dan mempercepat proses pengiriman file MP4 yang dihasilkan.

3. Penulis dapat membangun aplikasi *chatting* untuk kompresi *file* MP4 menggunakan algoritma *Levenstein* dengan menggunakan IDE Android Studio untuk memudahkan para pengguna dalam merancang tampilan program kompresi *file* MP4 yang diperoleh.

6. REFERENCES

- [1] R. Syahputra, "KOMPRESI FILE VIDEO MP4 DENGAN MENGGUNAKAN METODE," pp. 52–57, 2016.
- [2] N. M. Mancelina, "Penerapan Metode Perbandingan Eksponensial Dalam Menganalisa Kinerja Algoritma Fixed Length Binary Encoding Dengan Variable Length Binary Encoding," *J. INFOTEK*, vol. 2, no. 1, pp. 1–4, 2017.
- [3] T. P. Sari, S. D. Nasution, and R. K. Hondro, "Penerapan Algoritma Levenstein Pada Aplikasi Kompresi File Mp3," *KOMIK (Konferensi Nas. Teknol. Inf. dan Komputer)*, vol. 2, no. 1, 2018.
- [4] D. Putra, *Pengolahan Citra Digital*. Andi, 2010.
- [5] D. T. Sutoyo, S. Si., *Teori pengolahan citra digital*. Yogyakarta: Andi, 2009.
- [6] C. T. Utari, P. Studi, M. Teknik, U. S. Utara, and K. Citra, "IMPLEMENTASI ALGORITMA RUN LENGTH ENCODING UNTUK PERANCANGAN APLIKASI KOMPRESI DAN DEKOMPRESI," vol. V, no. 2, pp. 24–31, 2016.
- [7] E. Prayoga and K. M. Suryaningrum, "IMPLEMENTASI ALGORITMA HUFFMAN DAN RUN LENGTH ENCODING PADA APLIKASI KOMPRESI BERBASIS WEB," vol. IV, no. 2, pp. 92–101, 2018.
- [8] D. Iqbal, A. R. Panggabean, I. W. Sinaga, and T. Zebua, "Implementasi Algoritma RC4 + Untuk Mengamankan Pesan Teks Pada Aplikasi Chatting," pp. 916–920, 2019.
- [9] M. C. Seng Hansun, S. Si., *Pemrograman Android dengan Android Studio IDE*. Andi, 2018.
- [10] David Salomon and G. Motta, *Handbook of Data Compression*. 2010.
- [11] 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.