Jurnal Ilmiah Sinus (JIS) Vol : 22, No. 1, Januari 2024
ISSN (Print) : 1693-1173 , ISSN (Online): 2548-4028

Implementasi Kompresi File MP3 dengan Menerapkan Algoritma Levenstein

Lenita Cahya Anggraeni[1)], Ahmad Fashiha Hastawan[2)], Djuniadi[3)]
[1,2,3)] Pendidikan Teknik Informatika dan Komputer, Universitas Negeri Semarang
[1)] lenitacahya305@students.unnes.ac.id, [2)] ahmad.fashiha@mail.unnes.ac.id, [3)]djuniadi@mail.unnes.ac.id

ABSTRACT

MP3 is an audio format that is often used because the data stored in MP3 format resembles the actual data when recorded.Storage space requirements will also rise if you keep a large number of MP3 files. Because of this, MP3 files must be compressed in order to prevent the storage space from filling up too quickly. This is because the compressed data will be lower in size. The Lossless compression method can be used to compress audio. The author uses the Levenstein Algorithm to implement audio compression in MP3 format. Levenstein's algorithm compresses data without any loss. When a file is compressed using lossless compression, the resulting file has the exact same size as the original file upon decompression. The author will be able to assess the effectiveness of MP3 file compression by using the Levenstein Algorithm to compress MP3 files. This will allow for faster transmission times and less data storage space consumption because the compressed MP3 file will shrink from its initial large size. After compression, the MP3 file's size—which was originally 128 bits—was reduced to 104 bits, yielding an 81.25% compression ratio. The purpose of this computation is to determine whether the compressed files are similar enough to be either decompressed or restored to their original size.

Keywords: *Compression, Levenstein Algorithm, MP3*

I. PENDAHULUAN

Perkembangan teknologi dan informasi sangatlah pesat. Informasi terdiri dari berbagai jenis, antara lain yaitu citra, audio, dan video. Audio merupakan suara yang diciptakan oleh getaran suatu benda. Audio biasanya digunakan sebagai efek suara pada suatu video, film, ataupun *games.* Audio juga dapat digunakan untuk diperdengarkan pada aplikasi *music player*. Audio merupakan format file untuk menyimpan data audio digital pada sistem komputer. Data audio digital pada sistem komputer dapat disimpan dengan kompresi maupun tanpa kompresi.

MP3 merupakan salah satu format audio yang cukup banyak digunakan. Data yang disimpan dengan format MP3 hasilnya serupa dengan data sebenarnya saat direkam, karena itu banyak yang menggunakan audio dengan format MP3. Format MP3 sering digunakan karena memiliki ukuran yang cukup kecil jika dibandingkan dengan format audio yang lain. Akan tetapi, ukuran file MP3 cukup besar apabila dibandingkan dengan ukuran file dokumen biasa. Dengan demikian, jika semakin banyak file MP3 yang disimpan, maka akan semakin banyak pula ruang penyimpanan yang dibutuhkan. Untuk mengatasi penuhnya ruang penyimpanan maka perlu dilakukan kompresi pada file. Kompresi adalah suatu cara memampatkan data agar ukuran bit yang diperlukan suatu data dalam memori penyimpanan menjadi tidak terlalu besar.

Penulis mengimplementasikan kompresi audio berformat MP3 dengan menerapkan Algoritma Levenstein. Algoritma Levenstein merupakan salah satu kompresi *lossless. Lossless compression* adalah proses kompresi dimana file yang dikompresi jika dilakukan dekompresi akan menghasilkan file dengan ukuran yang sama persis dengan file awal. Semakin besar frekuensi kejadian karakter pada Algoritma Levenstein maka semakin kecil hasil kompresinya. Selain itu, Algoritma Levenstein cocok untuk digunakan mengkompresi data yang memiliki banyak pengulangan. Penulis melakukan penelitian terkait kompresi *file* MP3 dengan Algoritma Levenstein untuk mengetahui bagaimana

kompresi berpengaruh pada kinerja file MP3, memampatkan file besar menjadi lebih kecil untuk mempercepat proses transmisi dan mengurangi ruang penyimpanan yang dibutuhkan. Selain itu, penelitian ini dilakukan untuk menguji *file* hasil kompresi yang dapat dikembalikan lagi ukurannya seperti semula atau dapat didekompresi.

II. TINJAUAN PUSTAKA

1.1 Kompresi

Kompresi adalah suatu proses yang dilakukan dengan tujuan untuk mengurangi ukuran suatu data dan menciptakan representasi digital yang padat tetapi tetap dapat mewakili informasi-informasi yang ada dalam data (Purnama Sari et al., 2018). Kompresi file dapat dikatakan sebagai aspek penting dalam dunia teknologi informasi. File audio dan video merupakan jenis file yang berekstensi MP3 dan MP4. File audio dan video memiliki ukuran file yang cukup besar, karena itu diperlukan ruang penyimpanan lebih untuk menampung file audio dan video. Ukuran file berpengaruh terhadap kecepatan dalam proses pertukaran informasi, karena itu diperlukan kompresi file untuk mereduksi ukuran file tersebut (Telaumbanua et al., 2022).

Pada kompresi file audio dilakukan pemadatan pada isi file agar ukurannya berkurang, namun kualitas isi file tersebut tetap terjaga. Metode kompresi dibedakan menjadi dua, yaitu *lossless compression* dan *lossy compression* (Handayani, 2021).

1. *Lossless Compression*
   Lossless compression adalah suatu metode kompresi yang dilakukan tanpa menghapus data sebelumnya, sehingga data hasil kompresi dapat diubah lagi menjadi data aslinya (dekompresi). Metode ini menghasilkan rasio kompresi yang sangat rendah (Finola, 2019).
2. *Lossy Compression*
   Lossy compression adalah suatu metode kompresi yang tidak dapat mengembalikan data hasil kompresi menjadi data awal sebelum kompresi. Ketika dilakukan dekompresi akan terdapat beberapa bagian data yang hilang atau tidak sama seperti data asli sebelum kompresi. Rasio kompresi yang dihasilkan oleh *lossy compression* lebih tinggi daripada *lossless compression* (Finola, 2019).

2.2 Algoritma Levenstein

Algoritma Levenstein merupakan suatu kode universal pada bilangan bulat positif. Algoritma Levenstein dikembangkan oleh Vladimir Levenstein pada tahun 1968. Algoritma ini merupakan suatu algoritma yang pada saat enkripsi maupun pada saat deskripsi melewati tahapan-tahapan tertentu (Ramadhana, 2021).

Proses pengkodean pada Algoritma Levenstein yaitu 1 (satu) dan 0 (nol) (Purnama Sari et al., 2018). Berikut langkah-langkah encode untuk kode angka positif n:

1. Mengatur jumlah variabel C menjadi 1.
2. Menuliskankan representasi biner dari nomor tanpa awalan “1” ke kode awal.
3. Buat pemisalan M adalah jumlah bit yang dituliskan pada langkah kedua.
4. Apabila M tidak sama dengan 0, maka tambahkan C dengan 1. Ulangi langkah kedua, dengan memasukkan M sebagai nomor baru.

 DOI : https://doi.org/10.30646/sinus.v21i2.752

5. Jika M = 0, tambahkan C "1" bit dan 0 ke awal kode.

Tabel 1. Tabel Kode Levenstein

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

Tabel kode levenstein di atas adalah beberapa daftar dari kode-kode angka yang telah disandikan. Pada setiap kode angka disisipkan spasi yang menjadi penanda pada masing-masing bagian dari kode levenstein (Iqbal, 2019).
Untuk langkah-langkah *decoding* yaitu sebagai berikut:

1. Hitang jumlah bit C "1" sampai "0" ditemukan.
2. Apabila perhitungannya menghasilkan C = 0, maka nilainya yaitu nol, maka berhenti. Apabila tidak, lanjutkan langkah ketiga.
3. Set N = 1, ulangi langkah 4 (C – 1) kali.

Selanjutnya melakukan pembacaan pada bit N, lalu menambahkan angka 1, dan kemudian berikan nilai yang dihasilkan ke N sehingga dapat menghilangkan nilai sebelumnya dari N. Hasil dari pembacaan sandi yaitu string yang diberikan ke N pada iterasi terakhir.

2.3 MP3

MP3 merupakan salah satu format audio yang biasanya digunakan untuk menyimpan file-file musik audiobooks dalam hard drive. Data yang disimpan dengan format MP3 hampir sama dengan data asli saat direkam (Nasional et al., 2023). Moving Pictur Expert Group (MPEG) merupakan pengembang format audio MP3. MP3 menggunakan audio layer 3 yang mana umumnya dipergunakan untuk menyimpan file-file musik serta audiobooks di dalam hard drive. Format audio MP3 dapat menyajikan suara dengan kualitas yang mendekati CD stereo dengan 16-bit (Silitonga & Nasution, 2022).

2.4 Penelitian Terdahulu

Pada penelitian sebelumnya yang berjudul "Penerapan Algoritma Levenstein pada Aplikasi Kompresi File Mp3" yang dilakukan oleh Tetti Purnama Sari (Purnama Sari et al., 2018) didapatkan hasil kompresi *file* MP3 yang memiliki perbedaan ukuran dengan file

MP3 sebelum kompresi, hal ini dapat dilihat dari *Compression Ratio* yang didapatkan. Kecepatan proses kompresi *file* MP3 dipengaruhi oleh banyaknya ukuran awal. Sementara itu, pada pengujian dekompresi didapatkan *file* MP3 dengan ukuran yang sama seperti ukuran awal *file* MP3 sebelum dilakukan kompresi. Banyaknya ukuran kompresi memengaruhi kecepatan proses dekompresi *file* MP3.

III. METODE PENELITIAN

Metode yang digunakan oleh penulis pada penelitian ini yaitu metode *lossless compression*. *Lossless compression* adalah suatu metode kompresi yang mana data pada file sebelumnya tidak dihilangkan sehingga data yang telah dikompresi dapat dikembalikan lagi menjadi data asli sebelum dikompresi. Metode *lossless compression* ini memiliki rasio kompresi yang rendah.

Tahap-tahap yang dilakukan pada penelitian adalah sebagai berikut:

3.1 Input File MP3

File MP3 yang akan dikompresi dimasukkan ke *software* HxD atau Hex Editor untuk menampilkan nilai heksadesimal pada file MP3.

3.2 Melakukan Perhitungan

Perhitungan dilakukan secara manual, di mana proses perhitungan nilai sampel file MP3 secara manual diambil menggunakan software HxD atau Hex Editor, selanjutnya mengambil sampel bilangan desimal untuk perhitungan, membuat tabel kode levenstein, dan membentuk hasil string kode levenstein menjadi nilai file. Sementara tahapan perhitungan dekompresi dilakukan dengan memasukkan nilai heksadesimal hasil kompresi, mengubah hasil string bit kode levenstein yang menjadi nilai file menjadi bentuk biner dan melakukan pembacaan pada 8 bit terakhir, kemudian mengembalikan bentuk biner menjadi string bit semula.

IV. HASIL DAN PEMBAHASAN

Implementasi file MP3 dengan menerapkan Algoritma Levenstein diawali dengan melakukan analisis terhadap sampel file MP3 yang akan dikompresi. Tahap analisis dilakukan dengan memilih salah satu audio berformat MP3 dan kemudian melakukan pembacaan terhadap audio MP3 tersebut. Setelah melakukan pembacaan terhadap sampel file MP3 selanjutnya melakukan perhitungan secara manual. Proses perhitungan manual nilai sampel file MP3 diambil melalui software HxD atau Hex Editor untuk menampilkan karakter heksadesimal dari MP3 dan selanjutnya data nilai heksadesimal MP3 dikompresi dengan menerapkan Algoritma Levenstein. Implementasi kompresi audio MP3 menggunakan Algoritma Levenstein dilakukan dengan menggunakan sampel file MP3 sebagai berikut.

Tabel 2. Informasi Sampel File MP3

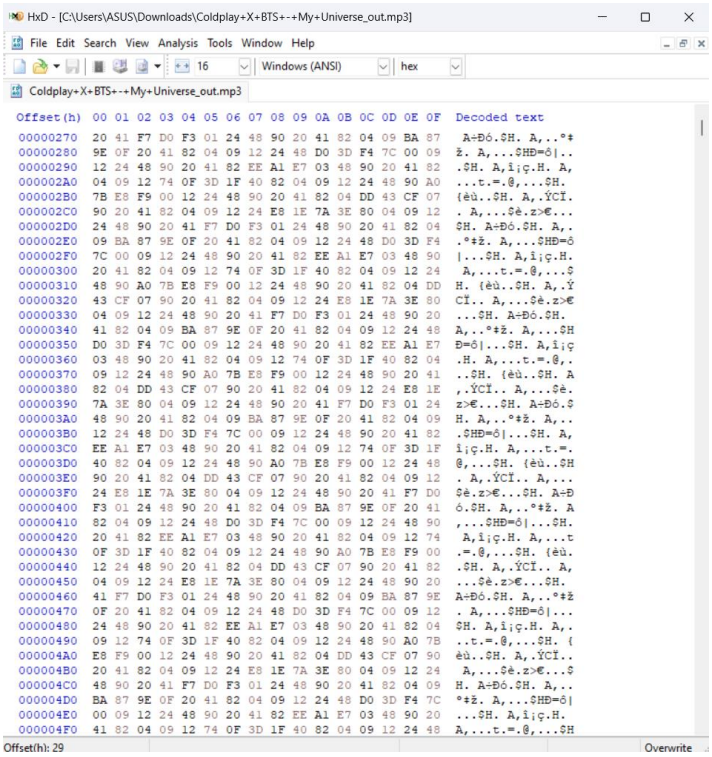
Jenis File	.MP3
Judul File	Coldplay+X+BTS+-+My+Universe_out
Ukuran	2.33 MB
Durasi	3.48 Menit

4.1 Kompresi Berdasarkan Algoritma Levenstein

Tahapan kompresi dengan menerapkan Algoritma Levenstein adalah sebagai berikut:

a. Memasukkan sampel file MP3 ke *software* HxD.

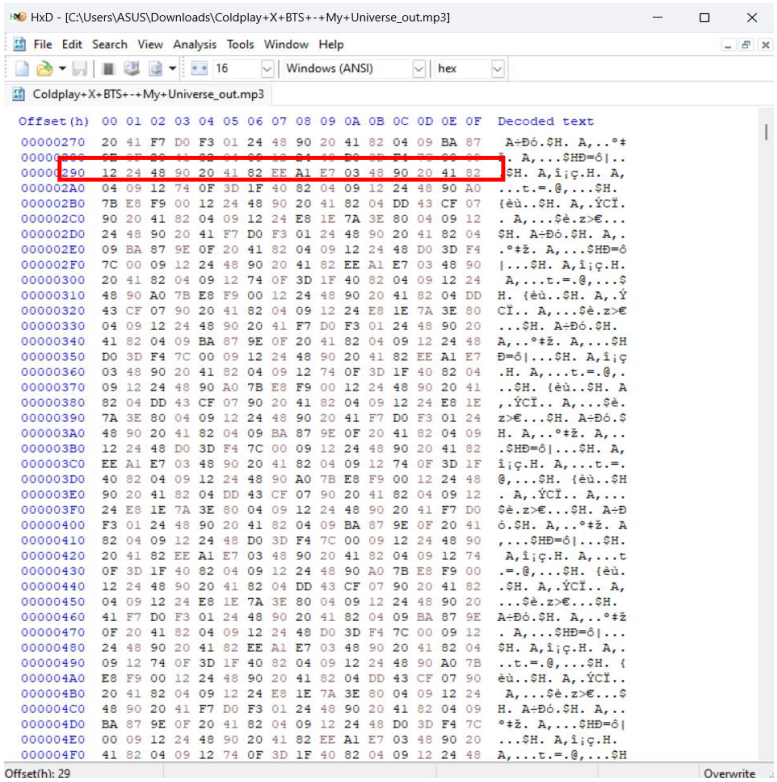
 DOI : https://doi.org/10.30646/sinus.v21i2.752



Gambar 1. Nilai Heksadesimal File MP3 Sampel

Dari gambar di atas didapatkan nilai heksadesimal pada sampel file MP3 sampel. Dan selanjutnya diambil sampel nilai heksadesimal sebanyak 16 karakter pada file MP3 untuk melakukan perhitungan secara manual.

b. Melakukan pembacaan isi file.



Gambar 2. Bilangan Heksadesimal yang Digunakan Untuk Perhitungan

Bilangan heksadesimal dari file MP3 yang diambil sebagai sampel untuk perhitungan adalah 20, 41, F7, D0, F3, 01, 24, 48, 90, 20, 41, 82, 04, 09, BA, 87.

c. Mengurutkan karakter heksadesimal dari yang memiliki frekuensi terbesar ke terkecil. Urutan bilangan heksadesimal dari frekuensi terbesar ke terkecil yaitu sebagai berikut.

Tabel 3. Nilai Bit File Sampel MP3

Nilai		Bit	Frekuensi	Bit x Frekuensi
Heksadesimal	Biner			
20	00100000	8	2	16
41	01000001	8	2	16
F7	11110111	8	1	8
D0	11010000	8	1	8
F3	11110011	8	1	8
01	00000001	8	1	8
24	00100100	8	1	8
48	01001000	8	1	8
90	10010000	8	1	8
82	10000010	8	1	8
04	00000100	8	1	8
09	00001001	8	1	8
BA	10111010	8	1	8
87	10000111	8	1	8
Total				128 bit

Dari tabel di atas, setiap satu karakter hekadesimal memiliki nilai delapan bit bilangan biner. Dengan demikian, nilai biner 16 bilangan heksadesimal yaitu sebanyak 128 bit. Ubah satuan bit menjadi *byte* dengan membagi jumlah total bit dengan 8. Sehingga menjadi 128 / 8 = 16 *byte.*

d. Membuat tabel kode Levenstein.
Ubah nilai heksadesimal file MP3 yang akan dikompresi dengan kode yang terdapat pada tabel kode Levenstein. Kemudian hitung jumlah bit pada setiap nilai.

Tabel 4. Kompresi Nilai Sampel MP3 dengan Kode Levenstein

N	Nilai Heksadesimal	Kode Lavenstein	Bit	Frekuensi	Bit x Frekuensi
0	20	0	1	2	2
1	41	10	2	2	4
2	F7	110 0	4	1	4
3	D0	110 1	4	1	4
4	F3	1110 0 00	7	1	7
5	01	1110 0 01	7	1	7
6	24	1110 0 10	7	1	7
7	48	1110 0 11	7	1	7
8	90	1110 0 000	8	1	8
9	82	1110 1 001	8	1	8
10	04	1110 1 010	8	1	8
11	09	1110 1 011	8	1	8
12	BA	1110 1 100	8	1	8
13	87	1110 1 101	8	1	8
Jumlah					90 bit

 DOI : https://doi.org/10.30646/sinus.v21i2.752

Dari hasil perhitungan pada tabel diatas, jumlah bit file MP3 setelah dikompresi dengan menggunakan kode Levenstein hasilnya yaitu 90 bit. Setelah diubah menjadi satuan *byte* hasilnya menjadi 11,25 *byte*.

e. Membentuk hasil string bit kode Levenstein menjadi nilai file.
Untuk membentuk nilai bit baru hasil kompresi yang diambil dari deretan nilai heksadesimal sebelum dikompresi yaitu 20, 41, F7, D0, F3, 01, 24, 48, 90, 20, 41, 82, 04, 09, BA, 87. Diubah menjadi nilai bit biner sehingga seperti berikut:
"01011001101111000011100011110010111001111100000010111010011110101011
10101111101100111011011".

Selanjutnya, melakukan pemeriksaan panjang string bit. Adapun langkah-langkah dalam pemeriksaan terhadap panjang string bit yaitu sebagai berikut.
1) Apabila panjang string bit habis dibagi dengan 8 atau bersisa 0 jika dibagi 8 maka tambahkan 00000001. Nyatakan dengan bit akhir.
2) Apabila panjang string bit dibagi dengan 8 bersisa n (1, 2, 3, 4, 5, 6, 7) maka sisipkan 0 sebanyak 7 – n + "1" pada string bit akhir, nyatakan dengan L. Kemudian tambahkan bilangan biner dari 9 – n dan nyatakan dengan bit akhir.

Total jumlah bit yang telah dikompresi yaitu 90 bit, sehingga n = MOD (90,8) = 2, dimana L = 7 – 2 + "1" = 5, dari bit menjadi 000001. Bit Akhir = 9 – 2 = 7, nilai binernya menjadi 00000111. Dengan demikian, string bit yang terbentuk yaitu:
"01011001101111000011100011110010111001111100000010111010011110101011
1010111110110011101101000001000001111"
Jumlah total bit sebelumnya setelah dilakukan penambahan pada string bit kode Levenstein yaitu 90+6+8 = 104.
Pisahkan hasil string bit kode Levenstein menjadi 8 bagian supaya mudah untuk diubah ke dalam nilai heksadesimal. Hasilnya dapat dapat dilihat pada tabel berikut.

Tabel 5. Nilai Biner dan Heksadesimal Hasil Kompresi Keseluruhan

Biner	Heksadesimal
01011001	59
10111100	BC
00111000	38
11110010	F2
11100111	E7
11000000	C0
10111010	BA
01111010	7A
10111010	BA
11111011	FB
00111011	3B
01000001	41
00000111	7

Selanjutnya yaitu menghitung *ratio of compression* (RC) dan *compression ratio* (CR).
$ukuran\ data\ sebelum\ dikompresi$ = 128
$ukuran\ data\ setelah\ dikompresi$ = 104

Ratio of Compression (RC)

Ratio of Compression (RC) adalah perhitungan perbandingan antara ukuran data hasil kompresi dengan ukuran data asli atau data sebelum dikompresi.

$$\textit{Ratio of Compression (RC)} = \frac{\textit{Ukuran data setelah dikompresi}}{\textit{ukuran data sebelum dikompresi}} \quad (1)$$

$$= \frac{104}{128}$$

$$= 0.8125$$

Compression Ratio (CR)

Compression Ratio (CR) merupakan proses perhitungan ukuran data hasil kompresi dibagi dengan ukuran data asli dan kemudian dikalikan dengan presentase 100%.

$$\textit{Compression Ratio (CR)} = \frac{\textit{Ukuran data setelah dikompresi}}{\textit{ukuran data sebelum dikompresi}} X\ 100\% \quad (2)$$

$$= \frac{104}{128} X\ 100\%$$

$$= 81.25\%$$

4.2 Dekompresi Berdasarkan Algoritma Levenstein

Tahapan-tahapan dekompresi dengan menerapkan Algoritma Levenstein adalah sebagai berikut :

a. Memasukkan nilai heksaadesimal hasil kompresi.
   Adapun nilai heksaadesimal dari hasil kompresi yaitu : 59, BC, 38, F2, E7, C0, BA, 7A, BA, FB, 3B, 41, 7.

b. Mengubah hasil string bit kode Levenstein yang menjadi nilai file menjadi bentuk biner, hasilnya yaitu sebagai berikut.
   "01011001101111000011100011110010111001111100000010111010011110101011101011111011001110110100000100000111"

c. Bentuk biner dikembalikan menjadi string bit semula dan lakukan pembacaan pada 8 bit terakhir. Ubah 8 bit terakhir tersebut menjadi bilangan desimal. Hasil pembacaan dinyatakan dengan n kemudian hilangkan bit pada bagian akhir sebanyak 7 + n.
   0101100110111100001110001111001011100111110000001011101001111010101110101111101100111011010*000001**00000111***

   n = 00000111 = **7** (desimal)

   → 7 + n → 7 + 7 = 14

   Selanjutnya 14 bit keseluruhan dari bit string terakhir dihilangkan, dan hasilnya menjadi seperti berikut:
   "01011001101111000011100011110010111001111100000010111010011110101011101011111011001110110"

   Kemudian lakukan pengecekan bit mulai dari bit pertama dengan memperhatikan tabel kode Levenstein pada tabel 4 di atas. Apabila bit yang sesuai dengan tabel kode

 DOI : https://doi.org/10.30646/sinus.v21i2.752

Levenstein ditemukan, maka ubah nilai string yang sesuai. Dengan demikian, didapatkan hasil sebagai berikut.

Tabel 6. Hasil Dekompresi dengan Menggunakan Kode Levenstein

Kode Levenstein	Nilai Heksadesimal
0	20
10	41
1100	F7
1101	D0
1110000	F3
1110001	01
1110010	24
1110011	48
11100000	90
0	20
10	41
11101001	82
11101010	04
11101011	09
11101100	BA
11101101	87

Dari hasil dekompresi di atas didapatkan nilai heksadesimal awal sebelum kompresi yaitu: 20, 41, F7, D0, F3, 01, 24, 48, 90, 20, 41, 82, 04, 09, BA, 87.

V. KESIMPULAN DAN SARAN

5.1 KESIMPULAN

Algoritma Levenstein merupakan salah satu kompresi lossless yaitu proses kompresinya dimana *file* yang dikompresi jika dilakukan dekompresi akan menghasilkan *file* dengan ukuran yang sama persis dengan *file* awal. Algoritma Levenstein dapat digunakan untuk mengompresi audio yang berekstensi .mp3. Dari hasil pengujian Algoritma Levenstein nilai *Ratio of Compression* (RC) yang didapatkan sebesar 81.25 dan nilai *Compression ratio* (CR) yang didapatkan sebesar 81.25%. Perhitungan tersebut dilakukan untuk menguji kesamaan *file* hasil kompresi dan dapat dikembalikan lagi ukurannya seperti semula atau dapat didekompresi.

5.2 SARAN

Untuk melengkapi penelitian selanjutnya, dapat dilakukan penambahan metode kompresi dan membandingkannya dengan metode kompresi yang penulis gunakan dalam penelitian ini.

DAFTAR PUSTAKA

Finola, R. O. (2019). Penerapan Algoritma Interpolative Coding Untuk Kompresi File Audio. *KOMIK (Konferensi Nasional Teknologi Informasi Dan Komputer)*, *3*(1), 378–384. https://doi.org/10.30865/komik.v3i1.1616

Handayani, R. (2021). *Perancangan Aplikasi Kompresi File Audio Menerapkan Algoritma Universal Codes*. *5*, 324–330. https://doi.org/10.30865/komik.v5i1.3735

Iqbal, D. (2019). Implementasi Algoritma Levenstein Untuk Kompresi File Video Pada Aplikasi Chatting Berbasis Android. *KOMIK (Konferensi Nasional Teknologi Informasi Dan Komputer)*, *3*(1), 266–273. https://doi.org/10.30865/komik.v3i1.1601

Nasional, S., Elektro, T., Informasi, S., & Informatika, T. (2023). *Implementasi Algoritma Levenshtein untuk Kompresi File Audio*. 314–320.

Purnama Sari, T., Darma Nasution, S., & Kristianto Hondro, R. (2018). *KOMIK (Konferensi Nasional Teknologi Informasi dan Komputer) PENERAPAN ALGORITMA LEVENSTEIN PADA APLIKASI KOMPRESI FILE MP3*. *2*. http://ejurnal.stmik-budidarma.ac.id/index.php/komik

Ramadhana, B. (2021). *Implementasi Kombinasi Algoritma Fibonacci Codes Dan Levenstein Codes Untuk Kompresi File Pdf*. *8*(2), 67–71.

Silitonga, S. H., & Nasution, S. D. (2022). Implementasi Algoritma Boldi-Vigna Codes Untuk Kompresi File Audio pada Aplikasi Pemutar Audio Berbasis Web. *KOMIK (Konferensi Nasional Teknologi Informasi Dan Komputer)*, *6*(November), 586–595. https://doi.org/10.30865/komik.v6i1.5754

Telaumbanua, L. Y., Bu, E., & Ulfa, K. (2022). Implementasi Algoritma Code-Excited Linear Prediction (CELP) pada Kompresi File Audio. *KOMIK (Konferensi Nasional Teknologi Informasi Dan Komputer)*, *6*(November), 317–321. https://doi.org/10.30865/komik.v6i1.5701.