KOMIK (Konferensi Nasional Teknologi Informasi dan Komputer)
Volume 6, Nomor 1, November 2022
DOI: 10.30865/komik.v6i1.5754
ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 586-595

Implementasi Algoritma Boldi-Vigna Codes Untuk Kompresi File Audio Pada Aplikasi Pemutar Audio Berbasis Web

Sukma Hakiki Silitonga, Surya Darma Nasution

Fakultas Ilmu Komputer dan Teknologi Informasi, Program Studi Teknik Informatika, Universitas Budi Darma, Medan, Indonesia
Email: [1,*]sukmaahakiki27@gmail.com, [2]surya.darma.nasution1@gmail.com

Abstrak-Kapasitas penyimpanan yang besar saat ini sangat penting dikarenakan data yang harus disimpan semakin lama semakin banyak, ukuran suatu file audio terkadang relatif berukuran besar, karna semakin panjang durasi suatu audio maka semakin besar pula ukuran file audio tersebut. Besarnya suatu ukuran *file* yang akan disimpan sangat berpengaruh pada ruang penyimpanan, artinya jika ukuran *file* yang akan disimpan berukuran besar maka dibutuhkan ruang penyimpanan yang cukup besar juga untuk menampung data tersebut. Adapun solusi dalam mengatasi masalah seperti ini adalah dengan melakukan teknik kompresi data. Kompresi data bertujuan untuk memperkecil sebuah ukuran *file* yang dapat mengurangi ukuran bit yang ada pada setiap *file audio*, akan tetapi tidak menghilangkan data informasi didalamnya. Dengan melakukan kompresi, data yang besar akan berkurang ukurannya sehingga dapat menghemat ruang penyimpanan. Dalam penelitian ini algoritma yang digunakan adalah *algoritma boldi-vigna codes*, dengan menggunakan algoritma tersebut, hasil kompresi dari setiap kode ζ mempunyai hasil yang berbeda-beda dari setiap nilainya, sehingga hasil kompresi akan menguntungkan.

Kata kunci: Kompresi; File Audio Mp3; Algoritma Boldi-Vigna Codes; Kapasitas

Abstract- Large storage capacities are currently very important because more and more data must be stored, the size of an audio file is sometimes relatively large, because the longer the duration of an audio, the larger the size of the audio file. The size of a file to be stored greatly affects storage space, meaning that if the file size to be stored is large, a large enough storage space is also needed to accommodate the data. The solution to overcome problems like this is to perform data compression techniques. Data compression aims to reduce a file size which can reduce the bit size in each audio file, but does not eliminate the information data in it. By compressing, large data will be reduced in size so as to save storage space. In this study the algorithm used is the boldi-vigna codes algorithm, by using this algorithm, the compression results of each ζ code have different results for each value, so the compression results will be profitable.

Keyword: Compression; File Audio Mp3; Boldi-Vigna Codes; Capacity

1. PENDAHULUAN

Kapasitas penyimpanan yang besar saat ini sangat penting dikarenakan data yang harus disimpan semakin lama semakin banyak, apalagi untuk seorang konten *kreator* atau *editor* video yang sangat membutuhkan kapasitas penyimpanan yang besar untuk menyimpan data data yang dimilikinya, juga untuk seorang penggemar musik untuk menyimpan *file* musik yang disukainya. *MP3* adalah suatu format *audio* yang sering digunakan untuk menyimpan *file-file* musik *audiobooks* dalam *hard drive*, juga karena data yang tersimpan hampir sama dengan data asli pada saat direkam. *Audio* juga dapat digunakan sebagai efek suara pada sebuah konten, berbagai *film* dan *games*, atau hanya untuk diperdengarkan pada aplikasi *music player*. Untuk menyimpan suatu *file audio*, tentu diperlukan sebuah media penyimpanan yang dapat menampung *file audio* tersebut.

Besarnya suatu ukuran *file* yang akan disimpan sangat berpengaruh pada ruang penyimpanan, artinya jika ukuran *file* yang akan disimpan berukuran besar maka dibutuhkan ruang penyimpanan yang cukup besar juga untuk menampung data tersebut. Dalam hal ini proses penyimpanan data juga bisa saja gagal dilakukan jika *file* yang akan disimpan memiliki ukuran yang lebih besar dari pada ruang penyimpanan yang tersedia, maka dibutuhkan sebuah teknik untuk mengatasi masalah tersebut yaitu teknik kompresi. kompresi bermaksud untuk mengekecilkan bentuk *file* sehingga saat melakukan proses pemindahan atau dalam penyimpanan suatu *file* akan menjadi lebih mudah atau efisien.

Salah satu algoritma kompresi adalah algoritma *Boldi-Vigna codes* yang mana pada tahun 2021 algortima *Boldi-Vigna codes* ini pernah dibandingkan dengan algoritma *Elias Delta Code* oleh Muhammad Abrar pada kompresi file *audio* berektensi **.wav*, dimana dalam pengujiannya didapatkan hasil bahwa algoritma *Boldi-Vigna Codes* lebih baik dalam melakukan kompresi dengan nilai rata-rata *Ration of Compression* sebesar 69,562%, *Compression Ratio* sebesar 1,458 dan *Space Saving* sebesar 30,428%[1].

Dari hasil penelitian lainnya yang dilakukan oleh Puja Samba Hasmita dkk pada tahun 2021 telah berhasil memproses kompresi *file audio* pada aplikasi lagu rohani yang berektensi *mp3* dengan menggunakan metode *Puncture Elias Code*, dimana saat proses kompresi *file* yang diinput berektensi **.mp3* dan menghasilkan *output* berektensi **.ipc* dan ukuran *file audio* pada aplikasi lagu rohani yang dikompresi sehingga dapat menghasilkan rasio rata-rata sebanyak 65,63%[2].

2. METODOLOGI PENELITIAN

2.1. Kompresi Data

File kompresi dimaksudkan sebagai data yang digabungkan merupakan satu dengan makna untuk memperoleh ukuran data yang kian kecil dibandingkan pada data aslinya[3]. Data yang dikompresi memungkinkan *file* lebih cepat ketika di-download dan lebih banyak data yang tersimpan dalam media penyimpanan eksternal.

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 586-595

2.1.1 Teknik Kompresi

Terdapat dua teknik yang dapat dilakukan pada saat melakukan kompresi[4]:

a. .Lossless

Kompresi *lossless* didasarkan pada prsoses pengurangan redudansi dan mempunyai penekanan pada cara yang lebih efisien saat proses *encoding* data image. Pada metode kompresi, kehilangan informasi dikatakan hamper hamper tidak terjadi, sehingga kompresi mempunyai hasil sama persis dengan informasi aslinya.

b..Lossy

Kompresi *lossy* didasarkan pada strategi pengurangan informasi yang tidak relevan, terutama saat proses *encode*. Transformasi kompresi *lossy* menghasilkan reduksi ukuran *file* yang sangat baik dibandingkan dengan metode *lossless*..

2.1.2. Rasio Kompresi (Compression Ratio)

Pada suatu teknik yang digunakan untuk mengkompresi data terdapat beberapa faktor yang dapat digunakan untuk menentukan kualitas data yang dikompresi, antara lain[5]:

a. *Ratio of Compretion* (RC)

Ratio of compression merupakan nilai perbandingan antara ukuran bit data sebelum dikompresi dengan ukuran bit data yang telah dikompresi. Secara sistematis dapat dituliskan sebagai berikut :

$$Rc = \frac{ukuran\ data\ sebelum\ kompresi}{ukuran\ data\ setelah\ kompresi} \quad \text{..... (1)}$$

b. *Compretion Ratio* (CR)

Compression ratio merupakan persentase perbandingan antara data yang sudah dikompresi dengan data yang belum dikompresi. Secara sistematis dapat dituliskan sebagai berikut :

$$Cr = \frac{ukuran\ data\ setelah\ kompresi}{ukuran\ data\ sebelum\ kompresi} \times 100\% \quad \text{.....(2)}$$

c. *Redundancy* (Rd)

Redundancy merupakan hasil penilaian nilai rasio dengan hasil nilai rasio kompresi. Secara sistematis dapat dituliskan sebagai berikut :

$$Rd = \frac{ukuran\ sebelum\ kompresi - ukuran\ setelah\ kompresi}{ukuran\ data\ sebelum\ kompresi} \times 100\% \quad \text{.....(3)}$$

d. *Space Saving* (Ss)

Space saving merupakan selisih antara data yang belum dikompresi dengan besar data yang sudah dikompresi. Secara sistematis dapat dituliskan sebagai berikut :

$$ss = 100\% - Cr \quad \text{.....(4)}$$

2.1.3. Dekompresi

Dekompresi merupakan proses mengembalikan sebuah data yang sudah terkompresi. Sebuah data yang telah dikompres tentunyaharus bisa dikembalikan juga kebentuk aslinya. Untuk mampu merubah data yang terkompres dibutuhkan cara yang berlainan seperti dalam waktu proses kompres dilaksanakan[6].

2.3. Algoritma Boldi-Vigna Codes

Algoritma *Boldi-Vigna Codes* merupakan algoritma kompresi *lossless* yang diperkenalkan oleh Paolo Boldi dan Sebastiano Vigna sebagai bagian dari *Variable-Length code* yang merupakan pilihan terbaik untuk kompresi. Dan algoritma *boldi-vigna* dikembangkan kembali pada suatu studi *WebGraph* dan sekarang disebut dengan *World Wide Web* (WWW) yang mana pada saat itu bertujuan untuk memperkecil ukuran pada *WebGraph* tersebut. Kode *zeta boldi-vigna* dimulai dengan bilangan bulat positif k yang menjadi faktor penyusutan kode. Himpunan semua bilangan bulat positif dibagi menjadi *interval* [2^0, 2^k - 1], [2^k, 2^{2k} - 1], [2^{2k}, 2^{3k} - 1], dengan bentuk umumnya adalah [2^{hk},$2^{(h-k)}$ - 1]. Panjang setiap *interval* dinyatakan dalam bentuk $2^{(h+1)k}$ - 2^{hk}[7].

Contoh :

Diketahui nilai N = 5, dan K = 4 (karena menggunakan *boldi-vigna* ζ_4) dan h = 0. Pertama lihat *unary code* dari h dengan rumus h + 1 = 0 + 1 = 1 dan unary code 1 adalah 1 (*unary code* yang digunakan tipe *reverse*)

Tabel 1 Tabel *Unary Code Reseve*

N	*Unary code reserve*
1	1
2	01
3	001
4	0001
...	...

Selanjutnya hitung nilai X dengan rumus n - 2^{hk}

$$X = n - 2^{hk}$$
$$= 5 - 2^{0.4}$$

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 586-595

$= 5 - 1$
$= 4$

Selanjutnya hitung nilai Z dengan rumus $2^{(h+1)k} - 2^{hk}$

$z = 2^{(0+1)4} - 2^{0.4}$
$= 2^4 - 2^0$
$= 16 - 1 = 15$

Selanjutnya hitung nilai S dengan rumus

$$S = \frac{\log(Z)}{\log(2)}$$

$$S = \frac{\log 15}{\log 2} = 4$$

Selanjutnya, untuk mencari nilai *codeword* ada 2 ketentuan yaitu :

1) Jika $x < 2^s - z$, maka x diencode sebagai kode biner sebanyak s-1 bit.
2) Jika $x \geq 2^s - z$, maka $(2^s + x - z)$ diencode sebagai kode biner sebanyak s bit.

Maka ketentuan yang pertama:

$x < 2^s - z$
$4 < 2^4 - 15$
$4 < 16 - 15$
$4 < 1$(tidak terpenuhi)

Jika ketentuan yang pertama tidak memenuhi maka dilakukan ketentuan yang kedua :

$4 \geq 2^s - z$
$4 \geq 2^4 - 15$
$4 \geq 16 - 15$
$4 \geq 1$(memenuhi)

Maka dilakukan $(2^s + x - z)$
$= 2^s + x - z$
$= 2^4 + 4 - 15$
$= 16 + 4 - 15 = 5$, diencode sebagai biner sebanyak s bit

Artinya nilai biner dari 5 diambil sebanyak 4 digit (s = 4) yang mana nilai biner 5 = 0101

Maka :

Unary code 1 = 1
Biner 5 = 0101
Boldi-vigna ζ_4 n 5 = 10101

Untuk lebih jelanya dapat dilihat pada tabel kode *boldi-vigna (ζ)* dapat dilihat pada tabel berikut.

Tabel 2 tabel kode boldi-vigna (ζ) codes

N	ζ_1	ζ_2	ζ_3	ζ_4
1	10	10	100	1000
2	110	110	1010	10010
3	011	111	1011	10011
4	00100	01000	1100	10100
5	00101	01001	1101	10101
6	00110	01010	1110	10110
7	00111	01011	1111	10111
8	0001000	011000	0100000	11000
9	0001001	011001	0100001	11001
10	0001010	011010	0100010	11010
11	0001011	011011	0100011	11011
12	0001100	011100	0100100	11100
13	0001101	011101	0100101	11101
14	0001110	011110	0100110	11110
15	0001111	011111	0100111	11111
16	000010000	00100000	01010000	010000111

2.4. File Audio Mp3

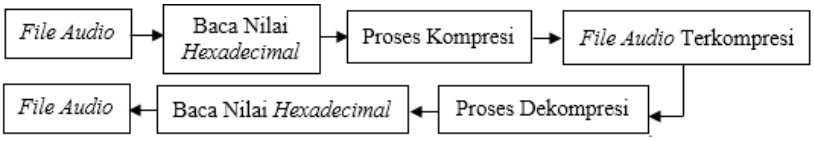
MP3 merupakan format *audio* yang dikembangkan oleh *Moving Pictur Expert Group* (MPEG). Format file ini menggunakan *audio* layer 3 yang secara umum digunakan untuk menyimpan *file-file* musik dan *audiobooks* dalam *hard drive*. Format *file mp3* mampu memberikan kualitas suara yang mendekati *CD stereo* dengan 16-bit. *MP3* mengalami kejayaan pada tahun 1995, dimana semakin banyak *file MP3* tersedia di internet dan popularitasnya semakin terdongkrak karena kualitasnya dan kapasitas yang menjadi relatif sangat kecil[8][9][10][11][12][13].

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 586-595

3. HASIL DAN PEMBAHASAN

3.1. Analisa

Kapasitas penyimpanan yang besar saat ini sangat penting dikarenakan data yang harus disimpan semakin lama semakin banyak, apalagi untuk seorang konten *kreator* atau *editor* video yang sangat membutuhkan kapasitas penyimpanan yang besar untuk menyimpan data data yang dimilikinya, juga untuk seorang penggemar musik untuk menyimpan *file* musik yang disukainya. *Audio* juga dapat digunakan sebagai efek suara pada sebuah konten, berbagai *film* dan *games*, atau hanya untuk diperdengarkan pada aplikasi *music player*. *File-file* yang biasanya disimpan oleh seorang konten kreator bukan hanya musik mp3 melainkan ada *file* gambar dan juga *file* video yang harus disimpannya. Sehingga terkadang seorang konten kreator perlu melakukan kompresi data pada *file-file* yang dimilikinya dengan tujuan memperkecil ukuran *file*, terutama pada *file mp3.* penelitian ini, bertujuan untuk melakukan analisa dari cara kerja dalam perancangan perangkat lunak pengkompresian file audio dengan menggunakan algoritma boldi-vigna codes.



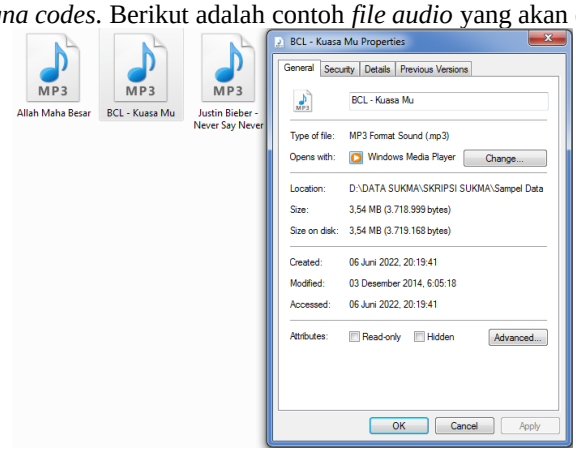
Gambar 1 Prosedur Kompresi Dekompresi File audio

Berdasarkan gambar yang ada diatas proses awal analisa yang dilakukan yaitu dengan mencari *file audio* yang akan dikompresi, kemudian mengubahnya ke nilai *hexadecimal* dengan bantuan *software hex editor* (HXD), kemudian nilai *hexadecimal* pada *file audio* tersebut akan dilakukan proses kompresi dan diperoleh *file* hasil kompresi. Proses dekompresi adalah proses yang mengembalikan kapasitas awal dari file gambar sebelumnya.

3.1.1. Contoh Kasus

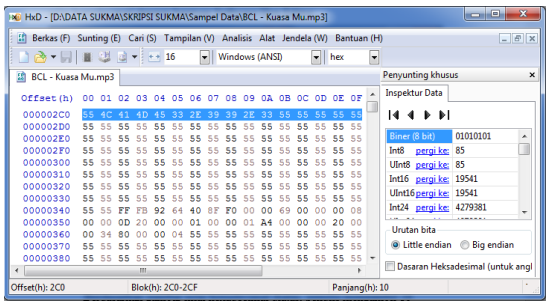
Untuk melakukan hitungan manual nilai *hexadesimal* sampel *file audio* didapatkan dengan bantuan *software hex editor* (HXD) kemudian nilai *hexadesimal file audio* tersebut dilakukan proses kompresi menggunakan algotima *boldi-vigna codes.*

Sebelum *file audio* dikompresi, yang harus dilakukan terlebih dahulu adalah mencari *file audio* yang akan dikompesi menggunakan algoritma *boldi-vigna codes.* Berikut adalah contoh *file audio* yang akan dikompresi dan didekompresi



Gambar 2 sampel file audio

Berdasarkan gambar diatas, maka *file audio* tersebut diubah terlebih dahulu menjadi nilai *hexadecimal* dengan bantuan *Software Hex Editor* (HXD).



Gambar 3 nilai hexadesimal sampel file audio

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 586-595

Berdasarkan gambar nilai *hexadecimal* diatas, nilai *hexadesimal file audio* mp3 yang digunakan sebanyak 16 karakter untuk dilakukan hitungan manual. Adapun nilai *hexadecimal* tersebut adalah 55, 4C, 41, 4D, 45, 33, 2E, 39, 39, 2E, 33, 55, 55, 55, 55, 55.

a. Kompresi Berdasarkan Algoritma Boldi-Vigna Codes
Untuk memulai proses kompresi, urutkan terlebih dahulu nilai *hexadecimal* berdasarkan frekuensi (banyak nilai yang sama) dan dihitung nilai bitnya, nilai frekuensi paling banyak berada diurutan pertama. Urutan bilangan *hexadesimal* dapat dilihat pada tabel berikut.

Tabel 3 Pengurutan Nilai *Hexadecimal*

No	Nilai Hexadecimal	Nilai Biner	Bit	Frekuensi	B x F
1	55	01010101	8	6	48
2	33	00110011	8	2	16
3	2E	00101110	8	2	16
4	39	00111001	8	2	16
5	4C	01001100	8	1	8
6	41	01000001	8	1	8
7	4D	01001101	8	1	8
8	45	01000101	8	1	8
		Total Bit			128

Setelah nilai *hexadecimal* diurutkan berdasarkan frekuensi kemunculan dan didapatkan nilai biner serta total bitnya, maka selanjutnya menghitung bit menggunakan kode algoritma *boldi-vigna*. Aturan dalam pembentukan kode *boldi-vigna codes* dapat dilihat pada sub sebelumnya yaitu lanasan teori. Adapun kode **ζ** *boldi-vigna* dapat dilihat pada tabel berikut.

Tabel 4 tabel kode boldi-vigna (ζ) codes

N	ζ_1	ζ_2	ζ_3	ζ_4
1	10	10	100	1000
2	110	110	1010	10010
3	011	111	1011	10011
4	00100	01000	1100	10100
5	00101	01001	1101	10101
6	00110	01010	1110	10110
7	00111	01011	1111	10111
8	0001000	011000	0100000	11000
9	0001001	011001	0100001	11001
10	0001010	011010	0100010	11010
11	0001011	011011	0100011	11011
12	0001100	011100	0100100	11100
13	0001101	011101	0100101	11101
14	0001110	011110	0100110	11110
15	0001111	011111	0100111	11111
16	000010000	00100000	01010000	010000111

Langkah selanjutnya adalah melakukan kompresi nilai *hexadesimal* dari sampel *file audio* dengan kode *boldi-vigna* diatas. Adapun proses penghitungan bit pada nilai *hexadecimal file audio* tersebut dengan menggunakan kode *boldi-vigna* sebagai berikut :

1. Untuk *Boldi-Vigna* **(ζ_1)**
Untuk penghitungan bit dan pengurutan kode pada nilai *hexadecimal file audio* menggunakan algoritma *boldi-vigna* **(ζ_1)** dapat dilihat pada tabel berikut :

Tabel 5 Pengurutan Kode *Boldi-Vigna* **(ζ_1)**

No	Nilai *Hexadecimal*	*Boldi-Vigna* **(ζ_1)**	Bit	Frekuensi	B x F
1	55	10	2	6	12
2	33	110	3	2	6
3	2E	011	3	2	6
4	39	00100	5	2	10
5	4C	00101	5	1	5
6	41	00110	5	1	5
7	4D	00111	5	1	5
8	45	0001000	7	1	7
		Total Bit			56

Langkah selanjutnya adalah menyusun kembali *codeword* masing-masing sesuai dengan nilai *hexadecimal*.

Tabel 6 *Codeword* Masing-Masing Nilai Hexadecimal

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 586-595

55	**4C**	**41**	**4D**	**45**	**33**	**2E**	**39**
10	00101	00110	00111	0001000	110	011	00100
39	**2E**	**33**	**55**	**55**	**55**	**55**	**55**
00100	011	110	10	10	10	10	10

Setelah disusun kembali *codeword* masing-masing nilai *hexadecimal*, maka selanjutnya adalah menggabungkan semua *codeword* menjadi string bit (tanpa tanda koma dan spasi).
"10001010011000111000100011001100100001000111101010101010**"** dengan total 56 bit. Selanjutnya lakukan pembagian bit string menjadi per 8 bit untuk mengubah kembali ke karakter baru, dan per karter adalah 8 bit panjangnya. **"**10001010 01100011 10001000 11001100 10000100 01111010 10101010**"** Maka setelah dilakukan pembagian bit akan menyisahkan 0 (Bisa dengan menggunakan 56 mod 8 = 0). Dan dinyatakan dengan simbol n, maka n = 0. Ada beberapa langkah untuk melakukan pemeriksaan string bit yaitu :

a) Jika sisa bagi = 0, maka ditambahkan **"00000001"** diakhir bit dan dinyataan sebagai bit akhir.
b) Jika sisa bagi adalah (1,2,3,4,5,6,7), maka tambahkan **"0"** sebanyak **(7 – n + "1")** diakhir string bit dan dinyatakan sebagai *padding*. Lalu tambahkan bilangan biner dari hasil **(9 - n)** di akhir bit dan dinyatakan sebagai *flagging*.

Padding merupakan penambahan bit data yang telah dikompresi agar seluruh jumlah bit data tersebut habis dibagi 8 Sedangkan *flagging* merupakan penambahan 8 bit bilangan biner setelah *padding* yang dimana untuk mempermudan dalam membaca bit-bit hasil kompresi pada saat proses dekompresi.
Berdasarkan hasil perhitungan sisa bagi sebelumnya didapat hasil n = 0, maka untuk pemeriksaan string bit menggunakan langkah pertama, yang mana jika sisa bagi = 0 maka ditambahkan **"00000001"** dan dinyatakan sebagai bit akhir. **"**10001010011000111000100011001100100001000111101010101010**00000001"**Maka setelah dilakukan penambahan bit string, total bit sekarang menajdi **64** bit.
Langkah selanjutnya adalah membagi *string* bit yang telah ditambah *padding* dan *flagging* menjadi per 8 bit.

Tabel 7 Pengelompokan Bit

10001010	01100011	10001000	11001100
10000100	01111010	10101010	00000001

Lalu merubah nilai biner ke nilai desimal untuk mengetahui karakter baru yang dihasilkan dari tabel kode ASCII. Sehingga menghasilkan karakter karakter baru yaitu "**ŠCˆÌ†Zª**"

Dari hasil kompresi **(ζ_1)** diatas, maka untuk mengetahui tingkat kinerja algoritma *boldi-vigna* **(ζ_1)** akan dilakukan perhitungan rasio kompresi untuk mengetahui berapa rasio kompresi pengurangan ukuran yang didapat.
Ukuran sebelum dikompresi → 128
Ukuran setelah dikompresi → 64

1) Ratio of Compretion (RC)
$$Rc = \frac{ukuran\ data\ sebelum\ kompresi}{ukuran\ data\ setelah\ kompresi}$$
$$Rc = \frac{128}{64} = 2$$
2) Compretion Ratio (CR)
$$Cr = \frac{ukuran\ data\ setelah\ kompresi}{ukuran\ data\ sebelum\ kompresi} \times 100\%$$
$$Cr = \frac{64}{128} \times 100\% = 50\ \%$$
3) Space Saving (SS)
$$ss = 100\% - Cr$$
$$ss = 100\% - 50\% = 50\ \%$$

2. Untuk *Boldi Vigna* **(ζ_2)**
Untuk penghitungan bit dan pengurutn kode pada nilai *hexadecimal file audio* menggunakan algoritma *boldi-vigna* (ζ_2) dapat dilihat pada tabel berikut :

Tabel 8 Pengurutan Kode *Boldi-Vigna* (ζ_2)

No	Nilai *Hexadecimal*	*Boldi-Vigna* **(ζ_2)**	Bit	Frekuensi	B x F
1	55	10	2	6	12
2	33	110	3	2	6
3	2E	111	3	2	6
4	39	01000	5	2	10
5	4C	01001	5	1	5
6	41	01010	5	1	5
7	4D	01011	5	1	5
8	45	011000	6	1	6

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 586-595

Total Bit	55

Langkah selanjutnya adalah menyusun kembali *codeword* masing-masing sesuai dengan nilai *hexadecimal*.

Tabel 9 *Codeword* Masing-Masing Nilai *Hexadecimal*

55	**4C**	**41**	**4D**	**45**	**33**	**2E**	**39**
10	01001	01010	01011	011000	110	111	01000
39	**2E**	**33**	**55**	**55**	**55**	**55**	**55**
01000	111	110	10	10	10	10	10

Setelah disusun kembali *codeword* masing-masing nilai *hexadecimal*, maka selanjutnya adalah menggabungkan semua *codeword* menjadi *string* bit (tanpa tanda koma dan spasi). **"**1001001010100101101100011011101000010001111101010101010**"** dengan total 55 bit.

Selanjutnya lakukan pembagian bit *string* menjadi per 8 bit untuk mengubah kembali ke karakter baru, dan per karter adalah 8 bit panjangnya. **"**10010010 10100101 10110001 10111010 00010001 11110101 0101010**"** Maka setelah dilakukan pembagian bit akan menyisahkan 7 (Bisa dengan menggunakan 55 mod 8 = 7). Dan dinyatakan dengan simbol n, maka n = 7.

Berdasarkan hasil perhitungan sisa bagi sebelumnya didapat hasil n = 7, maka untuk pemeriksaan *string* bit menggunakan langkah kedua, yang mana jika sisa bagi = (1,2,3,4,5,6,7) maka ditambahkan **"0"** sebanyak (7 – n + "1"). Maka $7 - 7 = 0$. Artinya jika hasil n = 0 maka tinggal ditambah angka 1 diakhir bit, dan dinyatakan sebagai *padding*. Maka menjadi **"**1001001010100101101100011011101000010001111101010101010**1".**

Lalu tambahkan bilangan biner dari hasil (9 - n). Maka $9 - 7 = 2$ dan biner dari 2 adalah 00000010 dan dinyatakan sebagai *flagging* menjadi : **"**1001001010100101101100011011101000010001111101010101010**10 0000010"** total bit sekarang menajdi **64** bit. Langkah selanjutnya adalah membagi *string* bit yang telah ditambah *padding* dan *flagging* menjadi per 8 bit.

Tabel 10 Pengelompokan Bit

10010010	10100101	10110001	10111010	00010001	11110101	01010101	00000010

Lalu merubah nilai biner ke nilai desimal untuk mengetahui karakter baru yang dihasilkan dari tabel kode ASCII. Sehingga menghasilkan karakter karakter baru yaitu "**¥±ºDC1ÕU**"

Dari hasil kompresi **(ζ_2)** diatas, maka untuk mengetahui tingkat kinerja algoritma *boldi-vigna* **(ζ_2)** akan dilakukan perhitungan rasio kompresi untuk mengetahui berapa rasio kompresi pengurangan ukuran yang didapat.

Ukuran sebelum dikompresi → 128

Ukuran setelah dikompresi → 64

a. Ratio of Compretion (RC)

$$Rc = \frac{ukuran\ data\ sebelum\ kompresi}{ukuran\ data\ setelah\ kompresi}$$

$$Rc = \frac{128}{64} = 2$$

b. Compretion Ratio (CR)

$$Cr = \frac{ukuran\ data\ setelah\ kompresi}{ukuran\ data\ sebelum\ kompresi} \times 100\%$$

$$Cr = \frac{64}{128} \times 100\% = 50\ \%$$

c. Space Saving (SS)

$$ss = 100\% - Cr$$

$$ss = 100\% - 50\% = 50\ \%$$

3. Untuk *Boldi-Vigna* **(ζ_3)**

Untuk penghitungan bit dan pengurutn kode pada nilai *hexadecimal file audio* menggunakan algoritma *boldi-vigna* **(ζ_3)** dapat dilihat pada tabel berikut :

Tabel 11 Pengurutan Kode *Boldi-Vigna* **(ζ_3)**

No	Nilai *Hexadecimal*	*Boldi-Vigna* **(ζ_3)**	Bit	Frekuensi	B x F
1	55	100	3	6	18
2	33	1010	4	2	8
3	2E	1011	4	2	8
4	39	1100	4	2	8
5	4C	1101	4	1	4
6	41	1110	4	1	4
7	4D	1111	4	1	4
8	45	0100000	7	1	7
		Total Bit			61

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 586-595

Langkah selanjutnya adalah menyusun kembali *codeword* masing-masing sesuai dengan nilai *hexadecimal*.

Tabel 12 Codeword Masing-Masing Nilai *Hexadecimal*

55	**4C**	**41**	**4D**	**45**	**33**	**2E**	**39**
100	1101	1110	1111	0100000	1010	1011	1100
39	**2E**	**33**	**55**	**55**	**55**	**55**	**55**
1100	1011	1010	100	100	100	100	100

Setelah disusun kembali *codeword* masing-masing nilai *hexadecimal*, maka selanjutnya adalah menggabungkan semua *codeword* menjadi *string* bit (tanpa tanda koma dan spasi). **"**10011011110111101000001010101111001100101111010 100100100100100**"** dengan total 61 bit.

Selanjutnya lakukan pembagian bit *string* menjadi per 8 bit untuk mengubah kembali ke karakter baru, dan per karter adalah 8 bit panjangnya. **"**10011011 11011110 10000010 10101111 00110010 11101010 01001001 00100**"** Maka setelah dilakukan pembagian bit akan menyisahkan 5 (Bisa dengan menggunakan 61 mod 8 = 5). Dan dinyatakan dengan simbol n, maka n = 5.

Berdasarkan hasil perhitungan sisa bagi sebelumnya didapat hasil n = 5, maka untuk pemeriksaan *string* bit menggunakan langkah kedua, yang mana jika sisa bagi = (1,2,3,4,5,6,7) maka ditambahkan **"0"** sebanyak (7 – n + "1"). Maka $7 - 5 = 2$.

Maka "0" ditambahkan sebanyak 2 dan ditambah angka 1 diakhir bit, dan dinyatakan sebagai *padding*. Maka menjadi : **"**10011011110111101000001010101 11100110010111010100100100100100**001".** Lalu tambahkan bilangan biner dari hasil (9 - n). Maka $9 - 5 = 4$ dan biner dari 4 adalah 00000100 dan dinyatakan sebagai *flagging*, menjadi : **"**10011011110111101000001010101111001100101111010100100100100100**00100000100**. total bit sekarang menajdi **72** bit. Langkah selanjutnya adalah membagi *string* bit yang telah ditambah padiding dan *flagging* menjadi per 8 bit.

Tabel 13 Pengelompokan Bit

10011011	11011110	10000010	10101111	00110010
11101010	01001001	00100001	00000100	

Lalu merubah nilai biner ke nilai desimal untuk mengetahui karakter baru yang dihasilkan dari tabel kode ASCII. Sehingga menghasilkan karakter karakter baru yaitu "**›Þ,¯2ÊI!**"

Dari hasil kompresi **(ζ_3)** diatas, maka untuk mengetahui tingkat kinerja algoritma *boldi-vigna* **(ζ_3)** akan dilakukan perhitungan rasio kompresi untuk mengetahui berapa rasio kompresi pengurangan ukuran yang didapat.

Ukuran sebelum dikompresi → 128

Ukuran setelah dikompresi → 72

a) Ratio of Compretion (RC)

$$Rc = \frac{ukuran\ data\ sebelum\ kompresi}{ukuran\ data\ setelah\ kompresi}$$

$$Rc = \frac{128}{72} = 1.77$$

b) Compretion Ratio (CR)

$$Cr = \frac{ukuran\ data\ setelah\ kompresi}{ukuran\ data\ sebelum\ kompresi} \times 100\%$$

$$Cr = \frac{72}{128} \times 100\% = 56.25\ \%$$

c) Space Saving (SS)

$$ss = 100\% - Cr$$

$$ss = 100\% - 56.25\% = 43.75\ \%$$

4. Untuk *Boldi-Vigna* **(ζ_4)**

Untuk penghitungan bit dan pengurutn kode pada nilai *hexadecimal file audio* menggunakan algoritma *boldi-vigna* **(ζ_4)** dapat dilihat pada tabel berikut :

Tabel 14 Pengurutan Kode *Boldi-Vigna* **(ζ_4)**

No	Nilai *Hexadecimal*	*Boldi-Vigna* **(ζ_4)**	Bit	Frekuensi	B x F
1	55	1000	4	6	24
2	33	10010	5	2	10
3	2E	10011	5	2	10
4	39	10100	5	2	10
5	4C	10101	5	1	5
6	41	10110	5	1	5
7	4D	10111	5	1	5
8	45	11000	5	1	5
		Total Bit			74

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 586-595

Langkah selanjutnya adalah menyusun kembali *codeword* masing-masing sesuai dengan nilai *hexadecimal*.

Tabel 15 *Codeword* Masing-Masing Nilai *Hexadecimal*

55	**4C**	**41**	**4D**
1000	10101	10110	10111
45	**33**	**2E**	**39**
11000	10010	10011	10100
39	**2E**	**33**	**55**
10100	10011	10010	1000
55	**55**	**55**	**55**
1000	1000	1000	1000

Setelah disusun kembali *codeword* masing-masing nilai *hexadecimal*, maka selanjutnya adalah menggabungkan semua *codeword* menjadi *string* bit (tanpa tanda koma dan spasi).**"**10001010110110101111100010010100111010010100100 11100101000100010001000 1000**"** dengan total 74 bit.

Selanjutnya lakukan pembagian bit *string* menjadi per 8 bit untuk mengubah kembali ke karakter baru, dan per karter adalah 8 bit panjangnya. **"**10001010 11011010 11111000 10010100 11101001 01001001 11001010 00100010 00100010 00**"** Maka setelah dilakukan pembagian bit akan menyisahkan 2 (Bisa dengan menggunakan 74 mod 8 = 2), dan dinyatakan dengan simbol n, maka n = 2.

Berdasarkan hasil perhitungan sisa bagi sebelumnya didapat hasil n = 2, maka untuk pemeriksaan *string* bit menggunakan langkah kedua, yang mana jika sisa bagi = (1,2,3,4,5,6,7) maka ditambahkan **"0"** sebanyak (7 – n + "1"). Maka $7 - 2 = 5$ Maka "0" ditambahkan sebanyak 5 dan ditambah angka 1 diakhir bit, dan dinyatakan *padding* **"**10001010110110101111100010010100111010010100100111001010001000100010001000**000001".** Lalu tambahkan bilangan biner dari hasil (9 - n). Maka $9 - 2 = 7$ dan biner dari 7 adalah 00000111 dan dinyatakan sebagai *flagging*. Setelah dilakukan penambahan *padding* dan *flagging*, Maka menjadi **"**10001010110110101111100010010100111010010100100111001010001000100010001000**00000100000111.** Maka setelah dilakukan penambahan bit string, total bit sekarang menajdi **88** bit. Langkah selanjutnya adalah membagi *string* bit yang telah ditambah *padding* dan *flagging* menjadi per 8 bit.

Tabel 16 Pengelompokan Bit

10001010	11011010	11111000	10010100	11101001
01001001	11001010	00100010	00000001	00000111

Lalu merubah nilai biner ke nilai desimal untuk mengetahui karakter baru yang dihasilkan dari tabel kode ASCII. Sehingga menghasilkan karakter karakter baru yaitu "**ŠÚØ"ÉIÊ"**"

Dari hasil kompresi (**ζ_4**) diatas, maka untuk mengetahui tingkat kinerja algoritma *boldi-vigna* (**ζ_4**), maka akan dilakukan perhitungan rasio kompresi untuk mengetahui berapa rasio kompresi pengurangan ukuran yang didapat.

Ukuran sebelum dikompresi → 128

Ukuran setelah dikompresi → 88

a) Ratio of Compretion (RC)

$$Rc = \frac{ukuran\ data\ sebelum\ kompresi}{ukuran\ data\ setelah\ kompresi}$$

$$Rc = \frac{128}{88} = 1.45$$

b) Compretion Ratio (CR)

$$Cr = \frac{ukuran\ data\ setelah\ kompresi}{ukuran\ data\ sebelum\ kompresi} \times 100\%$$

$$Cr = \frac{88}{128} \times 100\% = 68.75\ \%$$

c) Space Saving (SS)

$$ss = 100\% - Cr$$

$$ss = 100\% - 68.75\% = 31.25\ \%$$

2. Dekompresi Berdasarkan Algoritma *Boldi-Vigna Codes*

Proses dekompresi pada *file audio* yang telah dikompresi dapat dilakukan dengan mengembalikan karakter-karakter aneh menjadi bilangan *hexadesimal* kemudian diubah menjadi *string* bit semula.

1) Untuk Nilai *Boldi-Vigna* (**ζ_1**)

File hasil kompresi *boldi-vigna* (**ζ_1**) yaitu : "**ŠCˆÌ†Zª**" kembali ke string masing-masing. Lalu, analisa karakter-karakter tersebut kedalam bentuk bilangan biner dan bilangan desimal sehingga menghasilkan bit hasil kompresi sebelumnya. Adapun bit hasil analisa pada karakter-karakter tersebut dapat dilihat pada tabel dibawah ini:

Tabel 17 Nilai Biner dan Desimal Karakter

No	karakter	*Decimal*	Biner
1	Š	138	10001010
2	C	99	01100011

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 586-595

3	ˆ	136	10001000
4	Ì	204	11001100
5	†	134	10000100
6	Z	122	01111010
7	ª	170	10101010
8		1	00000001

Berdasarkan pada tabel diatas, gabungkan seluruh nilai biner menjadi *string* bit (tanpa tanda koma dan spasi). "1000101001100011100010001100110010000100011111010101010100000001" dengan total 64 bit.

Selanjutnya lakukan pengurangan bit untuk kembali menjadi bit *string* semula dengan menghilangkan *padding* dan *flagging* dengan melakukan pembacaan 8 bit akhir lalu ubah ke desimal dan dinyatakan dengan n. Kemudian gunakan rumus 7 + n untuk menghilangkan *padding* dan *flagging* agar kembali ke string bit awal.

"1000101001100011100010001100110010000100011111010101010100000001" (bit akhir = 00000001)

N = 00000001 = 1

Lalu gunakan rumus 7 + n untuk menghilangkan *padding* dan *flagging*

7 + n = 7 + 1 = 8

Maka hilangkan string bit sebanyak 8 bit terakhir, sehingga string bit sekarang menjadi : "1000101001100011100010001100110010000100011111010101010" dengan total 56 bit. Selanjutnya lakukan pembacaan *string* bit dari kiri ke kanan, sehingga menemukan nilai *hexadesimal* yang sesuai dengan *codeword* algoritma *boldi-vigna* **(ζ_1)** sehingga kembali ke string awal yaitu "55, 4C, 41, 4D, 45, 33, 2E, 39, 39, 2E, 33, 55, 55, 55, 55, 55"

4. KESIMPULAN

Berdasarkan hasil analisa yang telah dilakukan, maka penulis mengambil kesimpulan sebagai berikut :

a. Proses awal yang dilakukan dalam mengkompresi *file audio* menggunakan algoritma *Boldi-Vigna Codes* yaitu dengan mencari *file audio* yang akan dikompresi, kemudian mengubahnya ke nilai *hexadecimal* dengan bantuan *software hex editor* (HXD), dan nilai *hexadecimal* tersebut akan dilakukan proses kompresi dan diperoleh hasil kompresi.
b. Berdasarkan hitungan manual dengan menggunakan algoritma *Boldi-Vigna Codes* telah berhasil melakukan proses kompresi *file audio* berekstensi *.mp3, yang mana pada *boldi-vigna* ζ_1 menghasilkan rasio kompresi sebesar 50%, *boldi-vigna* ζ_2 menghasilkan rasio kompresi sebesar 50%, *boldi-vigna* ζ_3 menghasilkan rasio kompresi sebesar 56.25% dan *boldi-vigna* ζ_4 menghasilkan rasio kompresi sebesar 68.75%.
c. Aplikasi pemutar *audio* berbasis web dirancang menggunakan *Microsoft Visual Studio Code*.

REFERENCES

[1] R. T. M. Sitepu, "UNIVERSITAS SUMATERA UTARA Poliklinik UNIVERSITAS SUMATERA UTARA," *J. Pembang. Wil. Kota*, vol. 1, no. 3, pp. 82–91, 2020.
[2] P. S. Hasmita and C. F. Sianturi, "Implementasi Algoritma Punctured Elias Code Untuk Kompresi File Audio Pada Aplikasi Lagu Rohani," vol. 10, pp. 46–50, 2021.
[3] C. S. Kim and C. C. J. Kuo, *Data Compression*, vol. 1. 2011. doi: 10.1002/9781118256053.ch13.
[4] A. M. Silitonga, S. D. Nasution, and P. Ramadhani, "Implementasi Algoritma Subexponential Code Untuk Kompresi File Gambar," *KOMIK (Konferensi Nas. Teknol. Inf. dan Komputer)*, vol. 3, no. 1, pp. 340–348, 2019, doi: 10.30865/komik.v3i1.1611.
[5] L. Belakang and M. Kemajuan, "BAB I PENDAHULUAN".
[6] D. Iqbal, "IMPLEMENTASI ALGORITMA LEVENSTEIN UNTUK KOMPRESI FILE VIDEO," vol. 3, pp. 266–273, 2019, doi: 10.30865/komik.v3i1.1601.
[7] D. Salomon and G. Motta, *Handbook of Data Compression 5th*, vol. 53, no. 9. 2019.
[8] P. Kuncara, "Macam-macam format file audio beserta kelebihan dan kekurangan." p. 1, 2017. [Online]. Available: https://klikhost.com/macam-macam-format-file-audio-beserta-kelebihan-dan-kekurangan/
[9] C. S. Kim and C. C. J. Kuo, *Data Compression*, vol. 1. 2011.
[10] Y. Darnita, K. Khairunnisyah, and H. Mubarak, "Kompresi Data Teks Dengan Menggunakan Algoritma Sequitur," *Sistemasi*, vol. 8, no. 1, p. 104, 2019, doi: 10.32520/stmsi.v8i1.429.
[11] D. Sinurat, "Universitas Sumatera Utara Poliklinik Universitas Sumatera Utara," *J. Pembang. Wil. Kota*, vol. 1, no. 3, pp. 82–91, 2018.
[12] D. Pratiwi and T. Zebua, "Analisis Perbandingan Kinerja Algoritma Fixed Length Binary Encoding Dan Algoritma Elias Gamma Code Dalam Kompresi File Teks," *KOMIK (Konferensi Nas. Teknol. Inf. dan Komputer)*, vol. 3, no. 1, pp. 424–430, 2019, doi: 10.30865/komik.v3i1.1623.
[13] S. Nainggolan, I. Pendahuluan, and A. A. G. Codes, "Analisa Perbandingan Algoritma Goldbach Codes Dengan Algoritma Dynamic Markov Compression (Dmc) Pada Kompresi File Teks Menggunakan," vol. 6, no. Dmc, pp. 395–399, 2019.