Journal of Computing and Informatics Research
ISSN 2808-375X (Media Online)
Vol 1, No 2, Maret 2022
Hal 50−55

Penerapan Metode Self Delimiting Codes Dalam Kompresi File

Chyintia Berliana Simbolon

Fakultas Ilmu Komputer dan Teknologi Informasi, Program Studi Teknik Informatika, Universitas Budi Darma, Medan, Indonesia
Email: schyntiaberliana@gmail.com

Abstrak−Kompresi data merupakan suatu teknik untuk memperkecil jumlah ukuran data (hasil kompresi) dari data aslinya. Pemampatan data umumnya diterapkan pada mesin computer, hal ini dilakukan karena setiap symbol yang dimunculkan pada computer memiliki nilai bit yang berbeda. Analisa dilakukan dalam proses cara kerja dari Algoritma Self Delimiting Codes. Teknik kompresi ini adalah untuk membangun kode pembatas-sendiri, kode-kode yang memiliki panjang variabel dan dapat diterjemahkan secara jelas menghasilkan keluaran algoritma berupa string biner atau kode yang menterjemahkan setiap string masukan agar string tersebut mempunyai jumlah bit yang sedikit dibandingkan dengan string yang tidak dikompres.

Kata Kunci: Kompresi; Algoritma Self Delimitiding Codes; File Audio

Abstrack−Data compression is a technique to reduce the amount of data size (compression results) from the original data. Data compression is generally applied to computer machines, this is done because each symbol that appears on the computer has a different bit value. The analysis is carried out in the process of working of the Self Delimiting Codes Algorithm. This compression technique is to build self-delimiting code, codes that have variable length and can be clearly translated to produce an algorithm output in the form of a binary string or code that translates each input string so that the string has a small number of bits compared to an uncompressed string.

Keywords: Compression; Self Delimitiding Codes Algorithm; Audio File

1. PENDAHULUAN

Kemajuan teknologi informasi telah berkembang pesat, sehingga kebutuhan akan pendukung-pendukung di dalamnya menjadi hal yang tidak bisa dihindari. Seperti kebutuhan akan media penyimpanan menjadi hal yang sangat dibutuhkan dalam sistem komunikasi. Dalam bidang teknologi informasi, komunikasi data sangat sering dilakukan. Komunikasi data berhubungan erat dengan pengiriman ataupun dalam media penyimpanan. Sistem komunikasi, data atau informasi tidak hanya berupa teks, tetapi dalam bentuk gambar, audio dan video. Salah satu informasi yang disimpan dalam bentuk *file* yaitu audio. Berbagai macam fasilitas teknologi terus dikembangkan agar masyarakat dapat melakukan pertukaran informasi dalam bentuk teks, gambar, audio dan video dengan baik. Semakin tingginya permintaan informasi yang *real-time*, menjadikan perpindahan data harus semakin cepat dan akurat. Hal tersebut menjadi masalah, dimana jalur komunikasi di Indonesia khususnya internet masih berada dalam kategori lambat dan sering bermasalah, dengan demikian data yang berukuran kecil akan dipilih karena akan lebih cepat dikirim dan lebih menghemat tempat penyimpanan.

File audio atau file suara yang berformat (mp3) sangat banyak digunakan hingga saat ini terutama file tersebut adalah file lagu. Banyak file audio berformat mp3 itu mengakibatkan sedikitnya ruang penyimpanan yang tersisa di memori handphone(hp) maupun memori laptop. Maka untuk menghemat ruang penyimpanan, perlu dilakukan proses kompresi agar ukuran file audio tersebut menjadi lebih kecil dan juga dapat mempengaruhi kecepatan dalam proses pengiriman file audio karena ukurannya menjadi lebih kecil.

Kompresi data adalah proses yang mengkonversi sebuah masukan berupa aliran data (*the source* atau data asli mentah) menjadi suatu aliran data lain (*the Output*, aliran bit atau aliran sudah dikompres) yang memiliki ukuran lebih kecil. Berdasarkan adanya kemungkinan data setelah dikompres dapat direkonstruksi kembali ke data yang asli, teknik kompresi data dibagi menjadi dua bagian yaitu kompresi *lossless* dan kompresi *Lossy*. Kompresi *lossless* memungkinkan data dapat dikembalikan ke data yang asli secara utuh atau tanpa ada informasi yang hilang dalam data tersebut. Sedangkan kompresi *Lossy* tidak dapat mengembalikan data yang telah dikompres secara utuh dari data yang asli pada saat proses dekompresi[1].

Adapun tujuan dari kompresi data adalah untuk mengurangi ukuran file sebelum menyimpan atau memindahkan data ke dalam media penyimpanan. Salah satu algoritma kompresi yang dapat digunakan adalah algoritma *Self Delimiting Codes* yang merupakan suatu algoritma yang membutuhkan masukan kata bersama dengan distribusi probabilitas untuk setiap masukan dan dikonstruksi sebagai pohon probabilitas yang akan diasosiasikan dengan huruf dari abjad masukan, algoritma ini juga dapat mengkompresi suatu file yang berukuran besar hingga memperkecil ukuran file tersebut.

Pada proses pengiriman data, kompresi juga bermanfaat pada penyimpanan data di dalam media sekunder. Kompresi bertujuan untuk mengurangi jumlah data yang digunakan untuk mewakili isi file teks, gambar, audio, dan video tanpa mengurangi kualitas data aslinya. Kompresi dilakukan dengan mengurangi jumlah bit yang diperlukan untuk menyimpan atau mengirimkan media digital tersebut[2].

Selain itu, penelitian yang dilakukan oleh Annisa Diah Mutiara, Sutardi dan Rahmat Ramadhan mengenai kompresi data dengan judul Aplikasi Kompresi *File* Audio Menggunakan Algoritma *Arithmetic Coding* menyimpulkan bahwa sistem terdapat tahap kompresi dan dekompresi. Tahap kompresi bertujuan untuk untuk memampatkan ukuran file audio, sedangkan tahap dekompresi bertujuan untuk mengembalikan ukuran *file* audio ke ukuran semula [3].

2. METODOLOGI PENELITIAN

2.1 Kompresi

Kompresi merupakan teknik memperkecil file yang berukuran besar dan mengurangi kebutuhan ruang penyimpanan. Proses kompresi merupakan proses yang mendekati pada minimisasi jumlah bit untuk representasi digital seperti gambar, audio, dan video, yang menghasilkan ukuran data yang lebih kecil namun tetap menjaga kuantitas informasi dalam data tersebut. Kompresi audio adalah proses memperkecil atau meminimalisasi jumlah tiap bit yang merepresentasikan suatu audio dengan data yang menjadi lebih kecil[1]. Berdasarkan kemungkinan data yang sudah dikompresi dapat dikembalikan ke bentuk aslinya atau data sebelum dikompresi

2.2 Algoritma *Self Delimiting Codes*

Untuk membangun self-delimiting kode, kode yang memiliki panjang variabel dan dapat diterjemahkan secara jelas[5].

1. Gandakan setiap bit dari pesan asli, sehingga pesan menjadi satu set pasangan bit identik, lalu tambahkan sepasang bit yang berbeda. Dengan demikian, pesan 01001 menjadi bitstring 00 | 11 | 00 | 00 | 11 | 01. Ini sederhana tetapi jelas terlalu panjang. Itu juga rapuh, karena satu bit yang buruk akan membingungkan decoder (komputer atau manusia). Variasi dari teknik ini mendahului setiap bit angka dengan bit intercalary 1, kecuali bit terakhir, yang didahului dengan 0. Jadi, 01001 menjadi ¯10¯11¯10¯10¯01. Kita juga bisa konsentrasikan bit antar kalkulus bersama dan minta mereka diikuti oleh jumlahnya, seperti pada 11110 | 01001 (yang merupakan nomor itu sendiri didahului dengan kode unary-nya).
2. Siapkan tajuk dengan panjang pesan dan tambahkan ke pesan. Ukuran tajuk tergantung pada ukuran pesan, jadi tajuk seharusnya membuat self-delimiting menggunakan metode 1 di atas. Dengan demikian, pesan 6-bit 010011 menjadi tajuk 00 | 11 | 11 | 00 | 01 diikuti oleh 010011. Tampaknya hasilnya masih sangat panjang (16 bit untuk mengkodekan enam bit), tetapi ini karena pesan kami sangat singkat. Diberikan 1 juta pesan bit, panjangnya membutuhkan 20 bit. Header pembatas diri adalah 42 bit panjang, meningkatkan panjang pesan asli sebesar 0,0042%.
3. Jika pesannya sangat panjang (triliunan bit) sundulannya mungkin terlalu panjang. Dalam kasus seperti itu, kita dapat membuat header itu sendiri membatasi diri dengan menulisnya dalam format mentah dan sebelumnya dengan tajuknya sendiri, yang dibuat sendiri dengan metode 1.
4. Sekarang jelas bahwa mungkin ada sejumlah header. Header pertama adalah membuat self-delimiting dengan metode 1, dan semua header lainnya digabungkan ke dalamnya format mentah. Komponen terakhir adalah pesan biner asli (sangat panjang).
5. Integer panjang variabel desimal dapat direpresentasikan dalam basis 15 (quindecimal) sebagai serangkaian nibble (masing-masing kelompok terdiri dari empat bit), di mana setiap nibble adalah basis-15 digit (mis., antara 0 dan14) dan nibble terakhir mengandung 16 = 11112. Metode ini terkadang disebut sebagai kode nibble atau byte coding. Tabel 3.25 mencantumkan beberapa contoh.
6. Variasi pada kode nibble dimulai dengan representasi biner dari integer n (atau n - 1), tambahkan dengan nol sampai jumlah bit habis dibagi 3, pisahkan ke dalam kelompok masing-masing tiga bit, dan awali setiap grup dengan 0, kecuali kelompok paling kiri (atau sebagai alternatif, paling kanan), yang diawali dengan 1. Panjang kode ini untuk bilangan bulat n adalah 4 (log2 n) / 3, sehingga sangat ideal untuk distribusi formulir.
7. Ini adalah distribusi kuasa hukum dengan parameter 3/4. Perpanjangan alami dari ini kode untuk grup k-bit. Kode semacam itu cocok dengan distribusi kuasa hukum formulir.
8. Jika data yang akan dikompresi terdiri dari sejumlah besar bilangan bulat positif kecil, maka skema pengemasan yang selaras kata mungkin memberikan kompresi yang baik (meskipun bukan yang terbaik) dikombinasikan dengan decoding cepat. Idenya adalah untuk mengemas beberapa bilangan bulat menjadi panjang tetap bidang kata komputer. Jadi, jika ukuran kata adalah 32 bit, 28 bit dapat dipartisi menjadi beberapa bidang k-bit sedangkan empat bit sisanya menjadi pemilih yang menunjukkan nilai k.
9. Metode yang dijelaskan di sini adalah karena yang menggunakannya untuk kompres indeks terbalik. Bilangan bulat yang mereka kompres positif dan kecil karena mereka perbedaan dari pointer berurutan yang ada dalam urutan. Penulis menggambarkan tiga skema pengemasan, yang hanya yang pertama, dijuluki simple-9, dibahas di sini.
10. Simple-9 mengemas beberapa bilangan bulat kecil menjadi 28 bit kata 32-bit, meninggalkan tersisa empat bit sebagai pemilih. Jika 28 bilangan bulat berikutnya untuk dikompresi semua miliki nilai 1 atau 2, maka masing- masing bisa muat dalam satu bit, sehingga memungkinkan untuk mengemas 28 bilangan bulat dalam 28 bit. Jika 14 bilangan bulat berikutnya semuanya memiliki nilai 1, 2, 3, atau 4, maka masing-masing cocok dalam 2-bit bidang dan 14 bilangan bulat dapat dikemas dalam 28 bit. Di sisi lain, jika integer berikutnya kebetulan lebih besar dari 214 = 16, 384, maka seluruh 28 bit harus dikhususkan untuk itu, dan kata 32-bit hanya berisi bilangan bulat ini. Pilihan 28 sangat kebetulan, karena 28 dapat dibagi dengan 1, 2, 3, 4, 5, 7, 9, 14, dan itu sendiri. Dengan demikian, kata 32-bit dikemas secara sederhana-9 dapat dipartisi dalam sembilan cara. Tabel 2.18 mencantumkan sembilan partisi ini dan menunjukkan bahwa paling banyak tiga bit terbuang (dalam baris e).

Tabel 1. Ringkasan Kode Simple-9.

Pemilih	Jumlah Kode	Panjang kode	Bit yang tidak digunakan
A	28	1	0
B	14	2	0
C	9	3	1
D	7	4	0
E	5	5	3
F	4	7	0
G	3	9	1
H	2	14	0
I	1	28	0

Dengan 14 bilangan bulat 4, 6, 1, 1, 3, 5, 1, 7, 1, 13, 20, 1, 12, dan 20, kami menyandikan yang pertama sembilan bilangan bulat sebagai c | 011 | 101 | 000 | 000 | 010 | 100 | 000 | 110 | 000 | b dan lima bilangan bulat berikut as e |01100 | 10011 | 00000 | 01011 | 10011 | bbb, dengan total 64 bit, di mana setiap b mengindikasikan sebuah bit yang tidak digunakan. Pencetus metode ini menunjukkan bahwa penggunaan kode Golomb akan mengompresi 14 bilangan bulat menjadi 58 bit, tetapi kehilangan kompresi kecil efisiensi simple-9 seringkali lebih dari dikompensasi oleh kecepatan decoding. Sekali empat bit paling kiri dari kata 32-bit diperiksa dan nilai pemilih ditentukan, 28 bit sisanya dapat dibongkar dengan beberapa operasi sederhana. Mengalokasikan empat bit untuk pemilih agak boros, karena hanya sembilan bit 16 nilai yang mungkin digunakan, tetapi fleksibilitas dari kode simple-9 adalah hasil dari banyak (sembilan) faktor dari 28. Adalah mungkin untuk menyerahkan satu nilai pemilih, memotong ukuran pemilih menjadi tiga bit dan meningkatkan segmen data menjadi 29 bit, tetapi 29 adalah bilangan prima, jadi 29- segmen bit tidak dapat dipartisi menjadi bidang dengan panjang yang sama. Penulis mengusulkan pembagian kata 32-bit menjadi pemilih 2-bit dan segmen 30-bit untuk mengemas data. Bilangan bulat 30 memiliki 10 faktor, sehingga tabel dari kode-10 sederhana, mirip dengan Tabel 2.18, akan memiliki 10 baris. Namun, bidang pemilih hanya dapat menentukan empat nilai yang berbeda, itulah sebabnya kode yang dihasilkan (tidak dijelaskan di sini) lebih kompleks dan dilambangkan dengan relatif-10, bukannya simple-10.

3. HASIL DAN PEMBAHASAN

Berdasarkan penelitian ini, akan dilakukan analisa dan perancangan perangkat lunak pengkompresian *file* audio dengan menggunakan algoritma *Self Delimiting codes*. Algoritma *Self Delimiting codes* merupakan salah satu teknik kompresi lossless yang dapat memperkecil suatu data berdasarkan dengan frekuensi karakter pada objek yang akan dilakukan proses kompresi. Penggunaan algoritma *Self Delimiting codes* akan dilakukan berdasarkan karakter yang sering muncul dan akan memiliki jumlah bit terkecil berdasarkan kode *Self Delimiting codes*, sedangkan karakter yang paling sedikit muncul akan memiliki jumlah bit terpanjang. Tahapan analisa terhadap suatu sistem dilakukan sebelum tahapan perancangan dilakukan.

Dalam hal ini penerapan algoritma *Self Delimiting codes* dilakukan terhadap *file* audio yang memiliki ukuran yang relatif besar. Dengan begitu proses kompresi data sangat diperlukan. Kompresi data merupakan teknik pemampatan data, sehingga diperoleh ukuran *file* yang lebih kecil dari aslinya. *File* audio memiliki ukuran relatif besar, semakin lama durasi *file* audio maka semakin besar alokasi penyimpanan yang dibutuhkan.

Dalam melakukan kompresi *file* audio sebelumnya harus dilakukan analisa terhadap file audio. *File* audio yang akan dianalisa yaitu *file* audio berformat *MP3*. Format *file* MP3 merupakan format yang minim kompresi, sehingga ukurannya cukup besar dan memerlukan pengkompresan *file*. Adapun tujuan dari analisa terhadap sistem yang akan dirancang yaitu untuk mengetahui dan merumuskan kebutuhan dari sistem serta membantu meminimalisir sumber daya yang belebih.

3.1 Penerapan Algoritma *Self Delimiting Codes*

Dalam penelitian ini,akan membahas 2 proses utama yaitu proses kompresi dan proses dekompresi, penulis akan mengkompresi audio dengan menggunakan algoritma *Self Delimiting codes)* merupakan salah satu algoritma yang termasuk di dalam metode *lossless* data. Sebelum *file* dikompresi, terlebih dahulu dilakukan pembacaan biner yang terdapat pada *file* audio untuk mendapatkan data berupa data biner. Membaca biner yang terdapat pada *file* gambar menggunakan aplikasi *Binery Viewer* untuk mencari nilai biner pada file audio. Berikut adalah contoh file audio yang akan di kompresi dan didekompresi

Tabel 2. Tampilan Nilai File Audio Berdasarkan *Biner Viewer*

Nilai File Audio	**Nilai Biner**
49	01001001
44	01000100
33	00110011
04	00000100

Nilai File Audio	Nilai Biner
00	00000000
00	00000000
00	00000000
00	00000000
00	00000000
12	00010010
00	00000000
00	00000000
03	00000011
6D	01101101
61	01100001

Berdasarkan pada tabel di atas maka didapatkan nilai *heksa* sebagai berikut: 49,44,33,04,00,00,00,00,00,12,00,00,03,6D,61,6A. Nilai Heksa ini dimasukan kedalam tabel untuk dilakukan pembacaan frekuensi. Pembacaan frekuensi dilakukan dengan menghitung jumlah nilai yang sama di setiap nilai *heksa* yang muncul. Adapun pembacaan frekuensi dapat dilihat pada tabel 3 berikut.

Tabel 3. Tabel Frekuensi Kemunculan Nilai

Heksa	Biner	Frekuensi
49	01001001	1
44	01000100	1
33	00110011	1
04	00000100	1
00	00000000	7
12	00010010	1
03	00000011	1
6D	01101101	1
61	01100001	1
6A	01101010	1
Jumlah		16

Berdasarkan pada tabel 3, didapatkan beberapa nilai *heksa* yang sama. Sebelum proses kompresi, langkah awal adalah membaca nilai *heksa* kemudian membuat tabel nilai *heksa* yang diurutkan dari nilai frekuensi terbesar (nilai *heksa* yang sama) ke terkecil. Urutan nilai *heksa* dapat dilihat pada tabel 4.

Tabel 4. *Heksa* yang Belum Dikompresi

Nilai Heksa	Binary	Bit	Frek	Bit x Frek
00	00000000	8	7	56
49	01001001	8	1	8
44	01000100	8	1	8
33	00110011	8	1	8
04	00000100	8	1	8
12	00010010	8	1	8
03	00000011	8	1	8
6D	01101101	8	1	8
61	01100001	8	1	8
6A	01101010	8	1	8
Total bit	128bit			

Langkah selanjutnya adalah dengan membentuk tabel kode *Self Delimiting codes.* Aturan dalam pembentukan kode bilangan dengan menggunakan *Self Delimiting codes* dapat dilihat pada sub landasan teori bab sebelumnya. Adapun kode *Self Delimiting codes)* dapat dilihat pada tabel 5 berikut.

Tabel 5. kode *Self Delimiting codes*

Pemilih	Jumlah kode	Panjang kode	Bit yang tidak digunakan
A	128	1	0
B	64	2	0
C	32	4	0
D	16	8	0
E	8	16	0
F	4	32	0

Pemilih	Jumlah kode	Panjang kode	Bit yang tidak digunakan
G	2	64	0
H	1	128	0

Langkah selanjutnya adalah melakukan proses kompresi berdasarkan tabel 5 sebagai berikut:

1. Proses kompresi dilakukan dengan cara memilih selector B dengan jumlah kode 64 dan panjang kode 2 dan bit yang tidak digunakan adalah 0 dan hasilnya dapat dilihat sebagai berikut:
   a. 01001001 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 0111
   b. 01000100 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 010001
   c. 00110011 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 1111
   d. 00000100 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 000001
   e. 00000000 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 00000000
   f. 00000000 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 00000000
   g. 00000000 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 00000000
   h. 00000000 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 00000000
2. Proses kompresi dilakukan dengan cara memilih selector B dengan jumlah kode 64 dan panjang kode 2 dan bit yang tidak digunakan adalah 0 dan hasilnya dapat dilihat sebagai berikut:
   a. 00000000 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 00000000
   b. 00010010 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 000110
   c. 00000000 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 00000000
   d. 00000000 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 00000000
   e. 00000011 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 000011
   f. 01101101 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 01101101
   g. 01100001 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 01100001
   h. 01101010 ➔ panjang kode 2 dengan bit yang tidak digunakan adalah 0 maka *code word* nya adalah 01101010

Hasil encode dari algoritma *Self Delimiting codes* pada Nilai heksa **"49,44,33,04,00,00,00,00,00,12,00,00,03,6D,61,6A"** dapat dilihat pada tabel 6 .

Tabel 6. Heksa yang sudah dikompresi dengan algoritma *Self Delimiting codes*

No	*Hexa decimal*	*Biner*	Panjang kode SDC	Bit yang tidak digunakan	*Code Word*	*Panjang Code Word*
1	49	01001001	2	0	0111	4
2	44	01000100	2	0	010001	6
3	33	00110011	2	0	1111	4
4	04	00000100	2	0	000001	6
5	00	00000000	2	0	00000000	8
6	00	00000000	2	0	00000000	8
7	00	00000000	2	0	00000000	8
8	00	00000000	2	0	00000000	8
9	00	00000000	2	0	00000000	8
10	12	00010010	2	0	000110	6
11	00	00000000	2	0	00000000	8
12	00	00000000	2	0	00000000	8
13	03	00000011	2	0	000011	6
14	6D	01101101	2	0	01101101	8
15	61	01100001	2	0	01100001	8
16	6A	01101010	2	0	01101010	8
Jumlah digit *biner*						112bit

Selanjutnya proses dekompresi yang dilakukan adalah menganalisa keseluruhan bit hasil dari kompresi sebelumnya. Adapun bit keseluruhan hasil dekompresi dapat dilihat pada tabel 7 berikut:

Tabel 7. Proses Dekompresi

No	*Code Word*	Pengembalian Kode	Karakter Awal
1	0111	2,2➔0	01001001
2	010001	2➔0	01000100
3	1111	2,2➔0	00110011
4	000001	2➔0	00000100
5	00000000	2➔0	00000000
6	00000000	2➔0	00000000

No	*Code Word*	Pengembalian Kode	Karakter Awal
7	00000000	2➔0	00000000
8	00000000	2➔0	00000000
9	00000000	2➔0	00000000
10	000110	2➔0	00010010
11	00000000	2➔0	00000000
12	00000000	2➔0	00000000
13	000011	2➔0	00000011
14	01101101	2➔0	01101101
15	01100001	2➔0	01100001
16	01101010	2➔0	01101010

Dari tabel 7 dapat dilihat bahwa data yang telah di kompresi dapat dikembalikan lagi menjadi data awal sebelum dilakukan proses kompresi.

4. KESIMPULAN

Berdasarkan penelitian yang telah dilakukan oleh penulis maka kesimpulan yang diambil adalah Hasil implementasi yang dilakukan dalam mengompresi *file* MP3 dimulai dengan mengambil nilai biner dari *file* audio MP3, dimana nilai biner tersebut yang akan dihitung untuk mengompresi *file* MP3. Hasil analisa mengkompresi *file* MP3 menggunakan algoritma *Self Delimiting Codes* berhasil mengompresi *file* yang sebelumnya sebesar 6,20 MB menjadi 6,18, MB.

REFERENCES

[1] R. O. Finola, "Penerapan Algoritma Interpolative Coding Untuk Kompresi File Audio," KOMIK (Konferensi Nas. Teknol. Inf. dan Komputer), vol. 3, no. 1, pp. 378–384, 2019, doi: 10.30865/komik.v3i1.1616.
[2] T. S. M.kom, E. M. M.Kom, D. V. Suhartono, O. D. N. MT, and W. M.Kom, Teori Pengolahan Citra Digital. Yogyakarta: C.v Andi Offset, 2009.
[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, doi: 10.30865/komik.v2i1.946.
[4] P. . Drs. Suarga, M.Sc., M.Math., Algoritma dan Pemrograman. Makassar, 2012.
[5] D. Salomon, "Handbook of Data Compression," Fifth Edit., Springer London Dordrecht Heidelberg New York: Springer, 2010.
[6] R. A. S. and M. Shalahuddin, Rekayasa Perangkat Lunak. Bandung, 2016.