Journal of Computing and Informatics Research
ISSN 2808-375X (Media Online)
Vol 1, No 3, Juli 2022
Hal 82−87

Kompresi File Teks dengan Menggunakan Kombinasi Squitur dan Elias Gamma Code

Tulus Fransius Silaban

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

Abstrak−File teks memiliki ukuran yang besar sehingga mempengaruhi ruang penyimpanan dan proses pengiriman. ukuran inilah yang menjadi faktor berapa banyak ruang penyimpanan yang akan dipakai dan berapa lama waktu yang dibutuhkan untuk melakukan pengiriman file. Semakin panjang teks maka ukuran juga semakin besar. Ukuran yang besar dapat diatasi dengan cara melakukan proses kompresi. Kompresi dilakukan untuk mengurangi ukuran dari sebuah file. Algoritma kompresi yang dapat digunakan yaitu algoritma sequitur dan algoritma elias gamma code. Kelebihan algoritma sequitur adalah dengan mengganti 2 karakter berdampingan dengan 1 karakter terminal pada karakter set, sedangkan kelebihan algoritma elias gamma code adalah mengganti bit awal karakter dengan bit code elias gamma. Kedua algoritma tersebut akan dikombinasikan untuk menghasilkan ukurang yang lebih kecil dari ukurang sebelumnya.

Kata Kunci: Kompresi; File Teks; Kombinasi; Squiter; Elias Gamma Code

Abstract−Text files have a large size that affects the storage space and the sending process. This size is a factor in how much storage space will be used and how long it will take to send files. The longer the text, the larger the size. Large sizes can be overcome by doing the compression process. Compression is done to reduce the size of a file. Compression algorithms that can be used are sequitur algorithm and elias gamma code algorithm. The advantage of the sequitur algorithm is to replace 2 characters side by side with 1 terminal character in the character set, while the advantage of the elias gamma code algorithm is to replace the initial bit of the character with a gamma elias code bit. The two algorithms will be combined to produce a smaller size than the previous one.

Keywords: Compression; Text File; Combination; Squiter; Elias Gamma Code

1. PENDAHULUAN

Pada saat sekarang banyak sekali perangkat lunak yang digunakan untuk menangani masalah kompresi data. Dalam proses kompresi data, ada beberapa hal yang harus diperhatikan. Yaitu, *time process* (waktu yang berjalan pada saat data di kompresi dan dekompresi), *ratio* (ukuran data setelah di kompresi dan dekompresi), *completeness* (kelengkapan data setelah file-file tersebut di kompresi dan dekompresi), *spacesavings* (persentase selisih ukuran data setelah dikompresi dengan ukuran data sebelum di kompresi).

Algoritma *elias gamma code* merupakan memiliki proses kompresi dan dekompresi cukup cepat karena algoritma *elias gamma code* melakukan proses encoding pada saat proses kompresi berlangsung. Waktu kompresi tidak bergantung pada besarnya ukuran file dan waktu dekompresi tidak tergantung pada waktu kompresi. Waktu kompresi rata-rata menbutuhkan waktu lebih dari pada waktu dekompresi. Elias menambah panjang dalam unary (u) untuk membentuk tabel kode *elias gamma code*. Dalam kode berikutnya, Eγ ditambahkan pada panjang kode (M) dalam biner (β). algoritma *sequitur* beroperasi dengan menjalankan batasan diagram keunikan dan aturan kegunaan.

2. METODOLOGI PENELITIAN

2.1 Kompresi

Proses kompresi merupakan proses mereduksi ukuran suatu data untuk menghasilkan representasi digital yang padat atau mampat (*compact*) namun tetap dapat mewakili kuantitas informasi yang terkandung pada data tersebut. Istilah kompresi sering disebut juga *source coding*, data *compression*, *bandwidth compression*, dan *signal compression*

Data dan informasi adalah dua hal yang berbeda. Pada data terkandung suatu informasi. Namun tidak semua bagian data terkait dengan informasi tersebut atau pada suatu data terdapat bagian - bagian data yang berulang untuk mewakili informasi yang sama. Bagian data yang tidak terkait atau bagian data yang berulang tersebut disebut dengan data berlebihan (*redundancy* data). Tujuan daripada kompresi data adalah untuk mengurangi data yang berlebihan tersebut sehingga ukuran data menjadi lebih kecil dan lebih ringan dalam proses transmisi. Prinsip umum yang digunakan pada proses kompresi adalah mengurangi duplikasi data sehingga memori untuk merepresentasikan menjadi lebih sedikit dari pada representasi data semula.

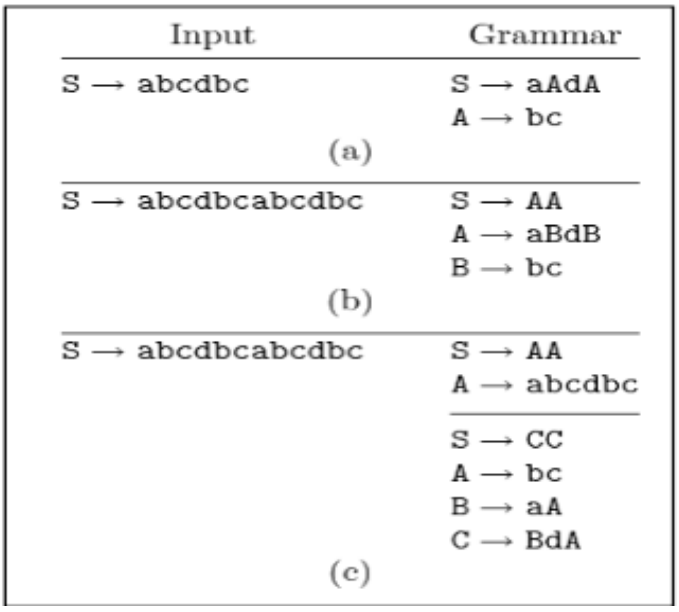
2.2 Algoritma *Sequitur*

Algoritma *sequitur* merupakan algoritma yang berdasarkan pada konsep tata bahasa bebas konteks (*context-free grammar*). Pada algoritma ini dikenal simbol non-terminal dan simbol terminal. Kedua jenis simbol tersebut merupakan unsur dari *production rules* (aturan-aturan yang digunakan untuk membangun sebuah kalimat). Sebuah simbol non-terminal diletakkan di sebelah kiri dan sebuah *string* simbol terminal dan non-terminal di sebelah kanan. Simbol non-terminal di sebelah kiri menjadi nama dari string yang di sebelah kanan. Algoritma *sequitur* membangun tata bahasanya dengan menggunakan 2 prinsip, yaitu :

1. Digram *Uniqueness*, tidak ada pasangan simbol yang berdekatan yang muncul lebih dari satu.
2. Semua aturan harus digunakan lebih dari satu kali dan aturan yang hanya digunakan sekali harus diabaikan atau ditiadakan.

Kedua aturan ini dapat dilihat pada gambar di mana terdapat 3 contoh string yang dikompresi menggunakan metode Sequitur. Input S sebelah kiri dari gambar bagian (a) sudah merupakan sebuah tata bahasa yang mengandung frase bc yang berulang sehingga frase bc dijadikan sebagai *production rule*. Frase bc kemudian diubah menjadi simbol nonterminal A. Perubahan tersebut menghasilkan dua aturan grammar, di mana aturan pertama merupakan input yang *redundancy*-nya dihilangkan dan aturan kedua merupakan simbol A yang menggantikan digram bc.

Pada gambar 1 bagian (b) input S sebelah kiri mengandung *redundancy* abcdbc yang kemudian diubah ke dalam simbol nonterminal A. Simbol nonterminal A mengandung *redundancy* bc yang juga diubah ke dalam simbol nonterminal B. Gambar 1. bagian (c) menunjukkan bagaimana prinsip kedua dari algoritma *sequitur*. Karena simbol nonterminal B yang mengandung simbol aA hanya muncul satu kali, maka simbol tersebuh diabaikan. Demikian pula dengan simbol C yang mengandung BdA yang juga hanya muncul sekali. Sehingga mengubah simbol abcdbc menjadi simbol nonterminal A adalah lebih optimal.



Gambar 1. Contoh Algoritma *Sequitur*

2.3 Algoritma Elias Gamma Code

Algoritma Elias Gamma Code adalah sistem pemampatan yang dikembangkan oleh Peter Elias yang digunakan untuk membuat kode dalam bentuk bilangan bulat positif. Contoh bilangan bulat yang dimaksud adalah $2^M \leq n < 2^{M+1}$. Algoritma kompresi ini telah lama ditemukan, cara kerja kompresi data yang digunakan dalam kompresi ini dibuat berdasarkan urutan dari posisi karakter yang akan dikompresi.

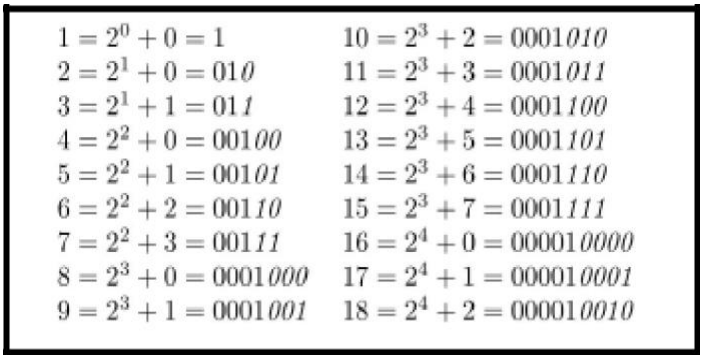
Pada kode Elias Gamma, suatu integer positif x direpresentasikan oleh 1 + dalam bentuk unary (sehingga 0 bit diikuti oleh 1-bit), diikuti oleh representasi bit dari x tanpa mengandung most significant bit. Sehingga 9 direpresentasikan sebagai 0001001, karena 1 + = 4, atau 0001 dalam unary, dan 9 adalah 001 dalam bentuk biner dengan most significant bit yang dihilangkan. Dengan cara ini, 1 direpresentasikan oleh 1, yang mewakili 1 bit.

Proses kompresi sendiri didasarkan pada bahwa isi *file* akan dibaca secara per *byte* (8 *bit*) sehingga menghasilkan nilai pembacaan antara 0 hingga 255. Dimana jika pembacaan mendapatkan nilai antara 1 – 7 maka akan diproses dengan menggunakan *Elias Gamma* dan bila tidak akan dipertahankan dalam dalam bentuk 8 *bit*. Pengkodean atas nilai antara 1–7 inilah yang menghasilkan efisiensi artinya semakin banyak data redundansi dalam rentang nilai 1–7 ini maka rasio kompresi yang didapat akan semakin besar.

Jika kode dari sebuah karakter bernilai n maka diandaikan $n = 2^M + L$, dimana M adalah pangkat tertinggi yang menghasilkan angka terdekat dengan nilai n yang dicari disimbolkan dengan $\beta(n)$ dan L adalah sisa dari $(n - 2^M)$ disimbolkan dengan $\alpha(n)$. Algoritma Elias Gamma Code dapat digunakan untuk pemampatan (kompresi) juga dapat digunakan untuk penirmampatan (dekompresi). Langkah-langkah kompresi algoritma elias gamma code :

1. Tentukan nilai M untuk pangkat yang paling mendekati nilai n yang dituliskan sebagai $\beta(n)$. Nilai ini disebut sebagai unary code, dimana jumlah nilai M ditulis menjadi angka 0 dan diakhiri dengan angka 1.
2. Dapatkan nilai L dengan mengurangi nilai n dengan nilai 2^M, nilai yang didapat diubah menjadi bilangan biner.
   Contoh : n = 13
   Nilai M yang tertinggi yang mendekati 13 adalah 3 karena $2^M = 2^3 = 8$, ubah menjadi bilangan unary menjadi 0001.

Sehingga dihasilkan L = 13 – 8 = 5, ubah nilai L menjadi nilai biner menjadi 101. Sehingga kode elias gamma dari 13 adalah 0001101. Agar lebih jelas berikut hasil pengkodean elias gamma code dapat dilihat pada gambar 2.



Gambar 2. Daftar Elias Gamma Code

Langkah-langkah dekompresi algoritma elias gamma code

1. Baca angka 0 dalam daftar kode yang ada hingga ditemukan angka 1. Jadikan jumlah angka 0 menjadi N
2. Selanjutnya baca nilai setelah N dan jadikan sebagai bilangan bulat L. Hitung $n = 2^N + L$.
   Contoh : 0001101

Jumlah angka 0 didepan sebelum angka 1 adalah 3, sehingga N = 3. Angka 101 adalah L dan diubah menjadi bilangan biner L = 5 Sehingga Nilai $n = 2^3 + 5 = 13$

3. HASIL DAN PEMBAHASAN

Analisa sistem merupakan tahap awal dalam sebuah penelitian yang bertujuan untuk mengetahui masalah terkait dalam pembuatan sebuah sistem untuk menghasilkan keluaran yang sesuai dengan kebutuhan pemakai dan pada bab ini mengemukakan tetang analisis masalah program yang akan dirancang dan rancangan program yang digunakan dalam penulisan ini.

3.1 Penerapan Algoritma Squitur

Sebelum melakukan proses kompresi menggunakan algoritma *squitur*, penulis menentukan *string* yang akan di gunakan. Misalkan *file* teks berisi *string* "KAKAK_DAN_KAKEK_MAKAN". Harus di ketahui terlebih dahulu bahwa *string* "KAKAK_DAN_KAKEK_MAKAN" terdiri dari 21 karakter, dimana 1 karakter tersebut sama dengan (=) 1 byte dan 1 byte sama dengan (=) 8 bit, maka total *string* sebelum dikompresi adalah 21 byte atau sama dengan (=) 168 bit. Untuk mengetahui ukuran dari *string* "KAKAK_DAN_KAKEK_MAKAN", dapat dilihat pada tabel sebagai berikut:

Tabel 1. Ukuran *String* Sebelum Dikompresi

Karakter	Frekuensi	ASCII Binary	Bit	Bit x Frekuensi
K	7	01001011	8	56
A	6	01000001	8	48
Sp (spasi)	3	00100000	8	24
N	2	01001110	8	16
D	1	01000100	8	8
E	1	01000101	8	8
M	1	01001101	8	8
	Jumlah bit x frekuensi			168 bit

File teks yang telah di ketahui *string* dan hasil jumlah dari setiap frekuensi kemunculannya, maka proses selanjutnya adalah proses kompresi menggunakan algoritma *sequitur*. Pada algoritma *sequitur*, *string* yang telah ada kemudian di proses untuk mendapat *string* yang telah ada kemudian diproses untuk mendapatkan *string* yang berduplikat atau dengan kata lain 2 karakter atau 2 huruf yang muncul lebih dari sekali. Kemudian karakter tersebut akan di periksa, jika ada pasangan yang muncul lebih dari satu kali maka pasangan tersebut diubah menjadi simbol non- terminal

***String* awal : KAKAK_DAN_KAKEK_MAKAN**

Seperti penjelasan di atas, kemudian akan di periksa apakah ada pasangan yang muncul lebih dari sekali. Dari *string* "KAKAK_DAN_KAKEK_MAKAN", pasangan karakter "KA" muncul sebanyak empat kali. Maka seperti prinsip kinerja algoritma *sequitur* ini, pasangan karakter "KA" akan di ubah menjadi simbol *non-terminal* yaitu menjadi "A". Maka terbentuklah sebuah *string* baru menjadi "AAK_DAN_AKEK_MAAN".

Kemudian dari *string* yang baru, akan di periksa kembali apakah ada pasangan karakter yang muncul lebih dari sekali. Setelah di periksa kembali ternyata pasangan karakter "AA" muncul lebih dari sekali yaitu muncul sebanyak dua kali. Maka dari itu, pasangan karakter "AA" akan di ubah menjadi simbol *non-terminal* yaitu simbol "B". Sehingga

membentuk *string* "BK_DAN_AKEK_MBN". Proses keseluruhan algoritma sequitur ini, dapat dimuat dalam bentuk tabel yang disajikan sebagai berikut :

Tabel 2. Proses Algoritma Sequitur

Proses	*String*	Diagaram Duplikat Pada String	Pembentukan Simbol (*Non-Terminal)*	Aturan (*Rule)*	String Yang Dihasilakan
1	KAKAK DAN KAKEK MAKAN	KA	A	A=KA	AAK DAN AKEK MAAN
2	AAK DAN AKEK MAAN	AA	A	A=KA B=AA	BK DAN AKEK MBN
3	BK DAN AKEK MBN	Null	A B	A=KA B=AA	-

Pada tabel 2, proses ketiga menunjukkan bahwa terdapat simbol *non-terminal.* Sehingga hasil dari simbol "A" adalah "KA" dan "B" adalah "AA". Jadi, *string* "KAKAK DAN KAKEK MAKAN" setelah diproses menggunakan algoritma *sequitur* menjadi *string* "BK_DAN_AKEK_MBN".

Proses dekompresi ini bertujuan agar *string* yang telah terkompresi kembali seperti semula saat sebelum di kompresi. Maka dari itu, setiap proses kompresi pasti selalu di dampingi dengan adanya dekompresi. Dari pernyataan ini, penulis akan mengurai bagaimana cara dekompresi algoritma *sequitur* dengan *string* yang telah di proses sebelumnya.

Telah di ketahui *string* yang sudah dikompresi yaitu *string* "BK_DAN_AKEK_MBN". Pada algoritma *sequitur*, proses algoritma ini memiliki aturan (*rule)* dimana di dalam aturan ini memiliki simbol A dan B. Setiap simbol memiliki pasangan karakter yang berulang. Untuk lebih jelas terhadap dekompresi algoritma *sequitur* dapat dilihat pada tabel berikut :

Tabel 3. Proses Dekompresi Algoritma Sequitur

Proses	String Kompresi	Aturan Rule	Keterangan Rule	String Setelah Dekompresi
1	"BK DAN AKEK MBN"	B = AA	Simbol "B" diganti dengan pasangan karakter "AA"	AAK DAN AKEK MAAN
2	AAK DAN AKEK MAAN	A = KA	Simbol "A" diganti dengan rule "KA"	KAKAK DAN KAKEK MAKAN

3.2 Penerapan Algoritma Elias Gamma Code

Algoritma Elias Gamma Code adalah sistem pemampatan yang dikembangkan oleh Peter Elias yang digunakan untuk membuat kode dalam bentuk bilangan bulat positif (Bagherzandi & Oktay, 2014). Contoh bilangan bulat yang dimaksud adalah $2^M \leq n < 2^{M+1}$.

Algoritma kompresi ini telah lama ditemukan, cara kerja kompresi data yang digunakan dalam kompresi ini dibuat berdasarkan urutan dari posisi karakter yang akan dikompresi. Setelah melakukan proses kompresi menggunakan algoritma *sequitur*, maka penulis akan melakukan kompresi file teks yang dimana *string* yang akan digunakan adalah hasil dari kompresi file teks *sequitur* yaitu "BK_DAN_AKEK_MBN".

Frekuensi kemunculan masing-masing karakter adalah :

B = 2 K = 3 SPASI = 3 D = 1 A = 2 N = 2 M = 1 E = 1

Setelah karakter dan frekuensi kemunculan tiap karakter, maka selanjutnya akan mengetahui pengukuran hasil karakter sebelum dikompresi.

Tabel 4. Ukuran *String* Sebelum Dikompresi

Karakter	Frekuensi	ASCII Binari	Bit	Bit x Frekuensi
B	2	01000010	8	16
K	3	01001011	8	24
SPASI	3	00100000	8	24
D	1	01000100	8	8
A	2	01000001	8	16
N	2	01001110	8	16
M	1	01001101	8	8
E	1	01000101	8	8
Jumlah Bit x Frekuensi				120 Bit

Selanjutnya, melakukan proses pengurutan karakter berdasarkan frekuensi kemunculannya yang diurutkan dari frekuensi terbesar ke kemunculan terkecil. Jika terdapat lebih dari suatu karakter dengan frekuensi kemunculan yang sama, maka diurutkan berdasarkan abjad. Untuk hasil pengurutan karakter dan frekuensi karakter dapat dilihat pada tabel sebagai berikut :

Tabel 5. Hasil Pengurutan karakter *Set* Kompresi *Elias Gamma Code* Berdasarkan Frekuensi Kemunculan

Nomor	Karakter	Frekuensi
1	Spasi	3
2	K	3
3	B	2
4	A	2
5	N	2
6	D	1
7	M	1
8	E	1
Jumlah		15

Langkah selanjutkan adalah masing-masing karakter pada table 3.3.1 di atas diganti dengan kode yang terdapat pada tabel kode *elias gamma*. Setelah diganti, hitung jumlah *bit* untuk tiap karakter.

Tabel 6. Hasil Pergantian Karakter Berdasarkan Kode *Elias Gamma*

No	Karakter	Frekuensi Karakter	Kode Elias Gamma Code	Bit	Bit x Frekuensi
1	K	3	1	1	3
2	Spasi	3	010	3	9
3	B	2	011	3	6
4	A	2	00100	5	10
5	N	2	00101	5	10
6	D	1	00110	5	5
7	M	1	00111	5	5
8	E	1	0001000	7	7
Jumlah Bit x Frekuensi					55 Bit

Tahap selanjutnya adalah menyusun kembali kode-kode yang telah dibuat pada tabel 3.3.2 diatas sesuai dengan posisi karakter pada *string* teks file materi yang dikompresi adalah "BK_DAN_AKEK_MBN", maka susunan kode-kode nya adalah :

011	1	010	00110	00100	010	00100	1	0001000	1	010	00111	011	00101
B	K	spasi	D	A	spasi	A	K	E	K	spasi	M	B	N

Gabungkan seluruh kode masing-masing karakter diatas dimulai dari karakter pertama hingga terakhir, sehingga diperoleh *string bit* sebagai berikut sebanyak 55 bit :

01110100011000100001010100010010001000101000111011001 01

Setelah diperoleh *string bit* maka, lakukan pengelompokan biner-biner hasil kompresi dengan jumlah bit perkelompok adalah 8 kemudian komversi menjadi karakter.

01110100 01100010 00010101 00010010 00100010 00111011 00101

Karena 55 bit tidak habis dibagi 8 maka akan dilakukan penambahan bit yaitu padding dengan ketentuan :

1. Jika sisa bagi panjang *string bit* terhadap 8 adalah 0 maka tambahkan 00000001. Nyatakan dengan bit akhir.
2. 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 binner dari 9-n. Nyatakan dengan bit akhir.

Karena sisa string bit n=5 maka akan di tambahakan 0 sebanyak 7- 5+"1"=2+"1"=001 lalu tambahakan flagging dengan rumus 9-n=9-5=4 lalu di ubah ke binner menjadi 00000100 maka padding dan falgging di tambakan ke string bit diatas menjadi

01110100 01100010 00010101 00010010 00100010 00111011 00101**00100000100**

Biner-biner diatas dikonversi menjadi karakter, sehingga di hasilkan karakter **tb ";)** karakter yang di hasilkan inilah (karakter hasil kompresi) yang dijadikan *record database* untuk menyimpan materi dengan teks"BK_DAN_AKEK_MBN" (contoh di atas).

Berdasarkan hasil kompresi, maka diperoleh total bit data sebelum dikompresi adalah sebanyak 120 bit dan ukuran setelah di kompresi menjadi 55 bit maka, kinerja dari kompresi berdasarkan algoritma *elias gamma code* adalah sebagai berikut :

Compression (Cr) = $\frac{Ukuran\ Data\ Setelah\ Dikompresi}{Ukuran\ Data\ Sebelum\ Dikompresi} = \frac{55}{120} * 100\%$ = 45,83%

Redundancy (Rd) = 100% - Cr = 100% - 45,83% = 54,17%

Langkah pertama dalam melakukan dekompresi file adalah dengan memasukkan file hasil kompresi. Saat file dimasukkan *Timer* akan mulai melakukan pencatatan waktu yang dilanjutkan dengan melakukan generate terhadap isi file ke *binary*. Hasil *generata* isi file menjadi *binary* dapat dilihat sebagai berikut :

01110100 01100010 00010101 00010010 00100010 00111011 00101**001 00000100**

Selanjutnya adalah dengan mengembalikkan *binary* menjadi *string* menjadi *string bit* semula. Untuk mengembalikkan *binary* menjadi *string bit* semula dapat dilakukan melalui langkah-langkah berikut ini :

1. Lakukan pembacaan pada 8 bit terakhir, hasil pembacaan berupa bilangan desimal. Nyatakan hasil pembacaan dengan *n.*
2. Hilangkan bit pada bagian akhir sebanyak 7 + n - "1".

Hasil pengembalian binary menjadi *string bit* semula dapat dilihat di bawah ini :

01110100 01100010 00010101 00010010 00100010 00111011 00101

Setelah diperoleh *string bit* seperti semula, langkah selanjutnya adalah dengan menggantikan kode pada *string bit* berdasarkan diatas agar diperoleh isi dokumen seperti sebelum mengalami kompresi. Berikut adalah langkah-langkah untuk mengganti *string bit* berdasarkan tabel kode *Elias Gamma Code* :

1. Lakukan pembacaan *string bit* dari awal hingga ketemu 1. Catat posisi angka 1 dan nyatakan sebagai p. Nyatakan jumlah 0 dengan n.
2. Lanjutkan pembacaan *string bit* setelah angka 1 sebanyak n.
3. Ganti kode hasil pembacaan dengan karakter berdasarkan diatas.

Hasil penggantian *string bit* berdasarkan diatas ataupun hasil dekompresi dengan kode *Elias Gamma Code* dapat dilihat di bawah ini :

String bit 011101000110001000010101000100100010001010000111011001 01

Hasil dekompresi diatas akan disimpan pada *file output* lalu *Timer* akan berhenti melakukan pencatatan waktu. Hasil pencatatan waktu oleh *Timer* akan dijadikan sebagai pembanding kecepatan dekompresi.
Langkah-langkah dekompresi dengan algoritma *Elias Gamma Code*, dapat dilihat di bawah ini.

4. KESIMPULAN

Penelitian ini menghasilkan beberapa kesimpulan sebagai berikut Berdasarkan prosedur kompresi dengan menggunakan algoritma Sequitur dan Elias Gamma Code telah berhasil melakukan proses kompresi file teks sehingga proses kompresi dapat berjalan sesuai dengan teknik kompresi. Berdasarkan penerapan algoritma yang memiliki ukuran besar dapat dikompres menjadi ukuran yang lebih kecil. Algoritma kompresi dipengaruhi oleh jumlah karakter dan variasi karakter Berdasarkan hasil pengujian menggunakan parameter waktu kompresi dan dekompresi waktu yang dibutuhkan algoritma Elias Gamma Code lebih tinggi dibandingkan algoritma Sequitur

REFERENCES

[1] K. Data, T. Dengan, and M. Algoritma, "Darnita, Kompresi Data Teks Dengan Menggunakan Algoritma Sequitur 104," vol. 8, pp. 104–113, 2019.
[2] A. Mukti, S. D. Nasution, R. Limbong, M. Nevill-manning, and I. Witlen, "IMPLEMENTASI ALGORITMA SEQUITUR UNTUK KOMPRESI SHORT MESSAGE SERVICE (SMS) BERBASIS ANDROID," vol. 17, pp. 362–365, 2018.
[3] T. Chandra, "Aplikasi Kompresi File dengan Algoritma Elias Gamma," no. 18.
[4] K. Data, T. Dengan, and M. Algoritma, "Darnita, Kompresi Data Teks Dengan Menggunakan Algoritma Sequitur 104," vol. 8, pp. 104–113, 2019.
[5] F. Gram et al., "Analisis Perbandingan Kompresi dan Dekompresi Menggunakan Algoritma Shannon-," vol. 3, no. 3, pp. 5197–5204, 2016.