Aplikasi Kompresi File dengan Algoritma Elias Gamma

Sukiman [1)] Tintin Chandra [2)]
STMIK IBBI Medan
Jl. Sei Deli No. 18 Medan, Telp. 061-4567111 Fax. 061-4527548
Email :sukiman.liu@gmail.com, tinuhnsbm@gmail.com

Abstrak

File merupakan tempat penyimpanan data dalam bentuk digital. Jumlah data yang disimpan pada file berbanding lurus dengan ukuran file itu sendiri. Ukuran file yang terlalu besar akan menjadi masalah bila file tersebut akan ditransfer atau dipertukarkan. Sebagai contoh file tidak muat pada media penyimpan seperti disket ataupun memperlambat proses download atau upload pada jaringan internet. Solusi untuk masalah ini biasanya file tersebut dibagi (split) menjadi ukuran yang lebih kecil. Cara lain adalah file tersebut dimampatkan sehingga ukurannya menjadi lebih kecil dari ukuran semula sehingga mempersingkat waktu ketika ditransmisi. Metode yang dipakai untuk memperkecil ukuran file adalah dengan algoritma Elias Gamma yang bekerja berdasarkan teknik pengkodean bilangan bulat menjadi bentuk biner yang lebih sederhana. Hasil adalah suatu perangkat lunak yang dapat melakukan kompresi dan dekompresi balik berdasarkan pada algoritma Elias Gamma pada semua jenis file yang belum dikompresi. Program juga dapat menampilkan besarnya rasio kompresi yang telah dikerjakan pada file.

Kata kunci: *kompresi data,, algoritma Elias Gamma*

Abstract

File represent data repository in the form of is digital. Amount of kept data at file compare diametrical of the size itself file. File size measure which too big will become the problem when the file will be transferred or commuted for. For example file do not load at depositor media like sketch and or slow down process of download or of upload at network of internet. Solution to the problem of this usually the file divided split become smaller size measure. Way of other is the file compressed so that its size measure become smaller than size measure initialy so that take a short cut time when transmission. Method weared to minimize file size measure is with algorithm of Elias laboring Gamma pursuant to integer code technique become binary form which more simple. Result is a software able to conduct decompression and kompresi return pursuant at algorithm of Elias Gamma at all of file type which not yet compress. Program also can present the level of ratio of kompresi which have been done at file.

Keywords*: data compression, Elias Gamma Algorithm*

1. Pendahuluan

Kompresi data atau juga dikenal sebagai pemadatan data adalah teknik yang dipakai untuk mengurangi data atau memperkecil data menjadi bentuk data lain dimana data tersebut diubah menjadi simbol yang lebih sederhana. Kompresi data dilakukan untuk mereduksi ukuran data atau *file*. Dengan melakukan kompresi atau pemadatan data maka ukuran *file* atau data akan lebih kecil sehingga dapat mengurangi waktu transmisi sewaktu data dikirim dan tidak banyak menghabiskan ruang media penyimpan, sedangkan dekompresi data merupakan kebalikan dari kompresi data yaitu mengembalikan data ke bentuk dan ukurannya semula. Proses dekompresi secara harfiah merupakan proses yang dilakukan bila data hasil kompresi ingin dikembalikan ke ukuran dan bentuknya semula. Tujuan dekompresi data karena data yang telah dikompresi tidak dapat dipakai secara langsung melainkan harus melalui suatu proses untuk mengembalikan ke ukuran semula.

File-file hasil pengolahan data komputer yang dikirimkan melalui internet dimana kebanyakan *file-file* tersebut berukuran besar sehingga memerlukan waktu yang lama untuk proses *upload* dan *download*. Untuk solusinya adalah *file-file* tersebut dapat dimampatkan terlebih dahulu sebelum dikirim sehingga ukurannya menjadi lebih kecil dan mempersingkat proses *upload* maupun *download*. Salah satu jenis dari *universal code* yang efisien dan cepat dimana tidak seperti metode kompresi lainnya, yaitu *Elias Gamma yang* langsung dapat melakukan proses *encode* dan *decode* tanpa adanya tabel ataupun

operasi yang sulit.

Terdapat banyak metode ataupun algoritma yang dapat digunakan untuk memadatkan data dimana secara umum metode-metode ini terbagi atas dua kategori yaitu *lossless* dan *lossy*. Metode *lossless* banyak diterapkan untuk memadatkan data yang tidak boleh menghilangkan data asli sehingga data dapat dikembalikan seperti semula. Terdapat banyak metode kompresi *lossless* di antaranya Huffman, LZ77, LZW, LZSS, LBE, dan lain-lain. Metode-metode tersebut mempunyai rasio kompresi dan kecepatan proses yang bervariasi. Salah satu jenis metode *lossless* yang bekerja dengan prinsip *universal code* adalah *Elias Gamma*. Metode *Elias Gamma* bekerja dengan cara membentuk suatu untaian kode *integer* dimana setiap data yang dibentuk akan didekompresi ulang. Pada kode ini, jumlah bit 0 sebelum bit 1 pertama yang muncul (*unary code*), menunjukkan berapa bit lagi yang harus diambil. Kode ini juga dapat di*decode* tanpa melakukan operasi yang rumit, karena setalah angka 1 ditemukan, maka jumlah dari bit yang harus diambil dapat diketahui. [2]

Berdasarkan tipe peta kode yang digunakan untuk mengubah pesan awal (isi file input) menjadi sekumpulan *codeword*, metode kompresi terbagi menjadi dua kelompok, yaitu

a. Metode statik :

Menggunakan peta kode yang selalu sama. Metode ini membutuhkan dua fase (*two-pass*): Fase pertama untuk menghitung probabilitas kemunculan tiap simbol/karakter dan menentukan peta kodenya, dan fase kedua untuk mengubah pesan menjadi kumpulan kode yang akan ditransmisikan. Contoh: algoritma Huffman statik.

b. Metode dinamik (adaptif) :

Menggunakan peta kode yang dapat berubah dari waktu ke waktu. Metode ini disebut adaptif karena peta kode mampu beradaptasi terhadap perubahan karakteristik isi file selama proses kompresi berlangsung. Metode ini bersifat *onepass*, karena hanya diperlukan satu kali pembacaan terhadap isi file. Contoh: algoritma LZW dan DMC.[8]

Pengkodean data dilakukan secara *on the fly*, bersamaan dengan proses penambahan *string* baru ke dalam *dictionary*. Pada algoritma ini, setiap *byte* hasil proses kompresi LZW dianggap sebagai 12 *bit.* Sehingga *dictionary* yang dipakai adalah sebesar 4096 (212) dan dikurangi dengan 256 karakter ASCII. Karena indeks dimulai dari nol, kisaran indeks *dictionary*-nya dimulai dari 256 sampai dengan 4095. [7]

Dalam kompresi data, tujuan utama yang perlu diperhatikan adalah rasio kompresi yang semakin baik, dan proses kompresi dan pengembalian yang semakin cepat. Rasio kompresi secara matematis dapat ditulis sebagai berikut:

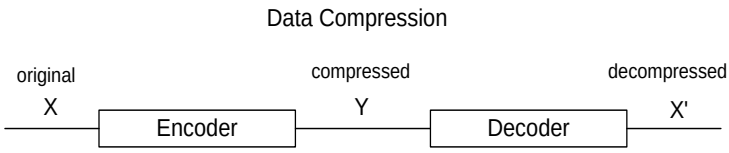
$$Rasio\ Kompresi = \frac{Output\ Stream}{Input\ Stream}$$

Compression rate atau laju kompresi adalah laju dari data yang dikompresi. Secara tipikal, satuannya adalah *bits*/*sample*, bits/*character*, *bits*/*pixel*, atau *bits*/ *second*. *Compression ratio* atau rasio kompresi adalah rasio atau perbandingan antara ukuran atau laju data yang dikompresi dengan ukuran atau laju dari data asli. [3]

Terdapat dua jenis utama dalam aplikasi kompresi data: transmisi dan penyimpanan. Suatu contoh dari yang terlebih dahulu adalah *speech compression* untuk transmisi secara *real time* melalui jaringan digital selular. Contoh untuk kasus yang kedua adalah kompresi *file* (contoh seperti program *DriveSpace* dan *DoubleSpace*) [1].

Jenis algoritma kompresi terbagi atas *lossless compression* dan *lossy compression*. Pada *lossy compression* ada data yang hilang tetapi tidak banyak setelah data dikompresi. Contoh standar yang menggunakan jenis *lossy compression* adalah JPEG (*Joint Photographic Experts Group*) sebagai standar *image* gambar atau *still image*, MPEG (*Motion Picture Experts Group*) untuk *audio video* sepeti Video CD, MP3 (*MPEG-1 Layer 3*) untuk *audio*. Data hasil kompresi dengan *lossy* compression jika dikembalikan maka hasilnya tidak akan sama persis lagi dengan data orisinil. Berbeda dengan *lossy compression*, pada *lossless compression* tidak ada data yang hilang setelah proses kompresi dan data dapat dikembalikan seperti data semula. Contoh standar yang menggunakan jenis ini adalah *Gzip, Unix Compress, WinZip, GIF* (*Graphic Interchange Form*at) untuk *still image* dan *Morse Code*.

Metode kompresi untuk jenis kompresi *lossless* diantaranya: *RLE, LZ78, Huffman*, *Splay Tree*, dan LZW. Sedangkan untuk jenis kompresi *lossy*, metode yang banyak digunakan antara lain: *Differential Modulation, Adaptive Coding* dan *Discrete Cosine Transform* (DCT). [2]



Gambar 1, Skema Proses Kompresi dan Dekompresi
Sumber: David Solomon, 2004: 20

X, Y, X' = String
Lossless Compression : X = X'
Lossy Compression : X!= X'
Compression Ratio: |Y|/|X| dimana |X| adalah jumlah *bit* dalam X dan |Y| adalah jumlah *bit* dalam Y.ecara umum, kebanyakan *prefix code* untuk bilangan integer tidak mampu lagi mengganti bilangan integer yang besar dengan *codeword*, seperti suatu kode dapat digunakan secara efisien untuk berkomunikasi dalam suatu pesan. *Universal code* umumnya digunakan untuk distribusi peluang yang diketahui secara tepat dan tidak ada *universal code* yang diketahui optimal untuk setiap distribusi yang digunakan dalam prakteknya.

Salah satu dari contoh *universal code* adalah Elias Gamma. Elias Gamma hanya dapat digunakan untuk mengkodekan bilangan bulat positif. Elias Gamma kebanyakan digunakan ketika pengkodean *integer* yang mempunyai *upper-bound* tidak dapat diperiksa sebelumnya. Adapun aturan untuk mengkodekan sebuah bilangan dengan menggunakan *Elias Gamma* adalah sebagai berikut:

1. Tulis bilangan tersebut dalam bentuk biner
2. Kurangkan 1 dari jumlah *bit* yang ditulis pada langkah pertama dan tambahkan sesuai dengan banyaknya bilangan nol. Proses yang ekivalen untuk menyatakan proses yang pada *point* nomor dua adalah sebagai berikut:
   a. Pisahkan *integer* menjadi pangkat 2 tertinggi (2^N) yang dapat dan ditampungnya sisakan digit biner N dari integer tersebut.
   b. Kodekan N dalam bentuk unar, jika N adalah nol maka diikuti oleh satu.
   c. Tambahakan sisa digit biner N untuk merepresentasikan N

Untuk melakukan proses *decode* suatu integer dengan menggunakan Elias Gamma dapat dilakukan dengan cara sebagai berikut:

a. Lakukan pembacaan dan perhitungan jumlah 0 dari stream hingga mencapai angka 1 pertama. Ini disebut dengan langkah perhitungan nilai nol *N*.
b. Pertimbangan satu yang telah dicapai sebagai *digit* pertama dari *integer*, dengan sebuah nilai 2^N, baca sisa dari N *digit* dari *integer*.

Pengkodean Gamma tidak melakukan pengkodean pada bilangan bulat nol ataupun negatif. Salah satu cara untuk menangani nol adalah menambahkan 1 sebelum pengkodeaan dan kemudian mengurangi dengan 1 setelah dilakukan proses *decoding*. Salah yang lain adalah memberi *prefix* pada semua kode bukan nol dengan 1 dan kemudian kode nol sebagai suatu 0 tunggal. Salah satu cara untuk mengkodekan semua integer adalah membentuk suatu bijeksi, yaitu pemetaan bilangan bulat (0, 1, -1, 2, -2, 3, -3, ...) hingga (1, 2, 3, 4, 5, 6, 7, ...) sebelum pengkodean dimulai.

Banyak algoritma kompresi menggunakan nilai bilangan bulat, seperti RLE yang melakukan penggantian terhadap karakter yang berulang dengan kode tertentu dan LZ77 melakukan pencocokan panjang *string* yang berdasarkan atas posisi *offset*. Setiap algoritma perlu untuk merepresentasikan nilai integer yang memberikan kelebihan jika dapat diatur pengurangan jumlah bit diperlukan untuk hal ini. Dari penjelasan ini tampak mengenai pengkode dari bilangan-bilangan buat ini sagat penting. Metode *encoding* apa yang dipilih bergantung kepada distribusi dan jangkuan dari nilai. Jika nilai yang terdistribusi rata-rata pada jangkauan seluruhnya, suatu representasi biner langsung merupakan pilihan yang optimal. Jumlah *bit* yang diperlukan tentunya juga bergantugn kepada jangkauan ini. Jika jangkuan bukan merupakan pangkat dua, beberapa pengaturan dapat dilakukan kepada kode untuk mendapatkan teoretikal optimum log2 (*range*) bit per nilai. [4]

Jika tidak terdapat batas atas terdefinisi pada nilai integer tersebut, kode biner langsung tidak dapat digunakan dan salah dari pengkodean Elias seperti Elias Gamma harus dipilih. Elias Gamma mengasumsikan bahwa nilai bilangan bulat yang kecil lebih dapat diproses. Pada kenyataan ini mengasumsikan (atau kelebihan dari) suatu distribusi yang proporsional. Nilai yang menggunakan *n bit* harus dua kali lebih menguntungkan dibandingkan nilai yang menggunakan *n* + 1 bit. Dalam pengkodean ini jumlah bit nol sebelum bit satu pertama (suatu kode *unary*) mendefinisikan seberapa banyak bit untuk didapatkan. Kode dapat dipertimbangkan sebagai pohon Huffman tetap. Sehingga dapat dihasilkan pohon

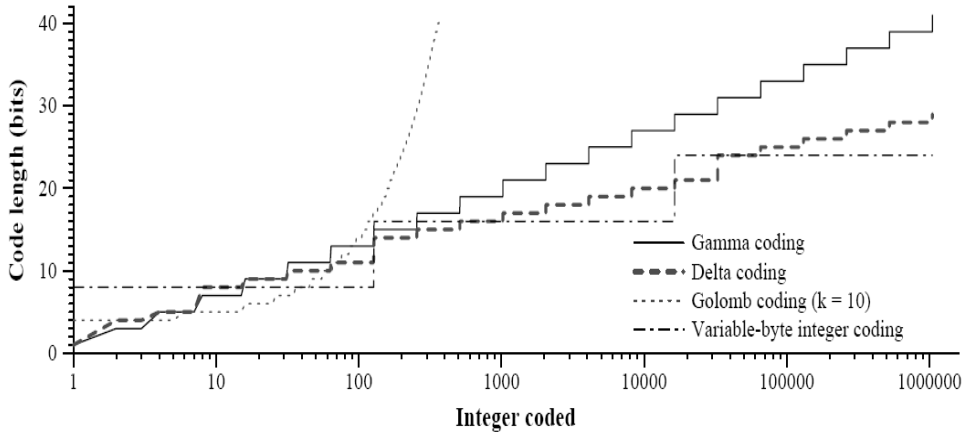
Huffman dari asumsi nilai distribusi dan mendapatkan nilai yang sangat mirip. Kode ini juga secara langsung dapat di-*decode* tanpa adanya tabel ataupun operasi yang sulit, karena sekali satu *bit* pertama ditemukan, panjang dari *code word* secara cepat dapat diketahui. *Bit* nol yang mengikutinya akan secara langung dikodekan menjadi nilai.

2. Metode Penelitian

Kompresi terdiri atas dua tahapan, pemodelan dan pengkodean. Suatu model untuk data yang akan dikompresi merepresentasikan suatu simbol tertentu dalam data menyertakan informasi seperti frekuensi mengenai setiap simbol. Pengkodean merupakan proses menghasilkan representasi data terkompresi, menggunakan model untuk menentukan suatu kode untuk setiap simbol. Skema pengkodean yang efisien berhubungan dengan kode pendek hingga simbol umum dan kode panjang hingga simbol yang jarang dipakai, dengan tujuan untuk mengoptimasi panjang kode. Jika suatu representasi *variable-byte* yang terdiri atas 7 dalam setiap *byte* digunakan untuk mengkodekan *integer*, dengan *least significant bit* diset menjadi 0 pada *byte* terakhir, atau 1 jika terdapat *byte* yang mengikutinya. Dengan cara seperti ini, *integer* kecil direpresentasikan secara efisien, sebagai contoh, 135 direpresentasikan dalam dua *byte* karena nilai ini terletak di antara *range* $[2^7 \ldots 2^{14}]$, sebagai 00000011 00001110, ini dibaca sebagai 00000010000111 dengan menghilangkan least significant *bit* dari setiap *byte* dan menyisakan 14 *bit*. Kode *variable – byte* untuk *integer* terpilih dalam *range* 1–30. Suatu aplikasi khusus merupakan pengkodean indeks dan mengubah daftar *offset file* untuk indeks terbalik.

Ketika menyimpan *array* besar dari *integer*, variable-*byte integer* secara umum tidak menghasilkan efesien seperti halnya dalam skema *bit* variable. Namun, ketika mneyimpan hanya beberapa *integer*, skema variable-*byte* yang telah diatur *byte*-nya hampir mempunyai persyaratan ruang yang sama dengan skema variable-*bit* yang ditambahkan untuk penyimpanan *byte*. *Integer* Variable-*byte* juga bekerja dengan baik dengan kumpulan data dimana struktur data tidak dikenal dan berbeda dengan skema *variable bit* tidak dapat secara selektif diterapkan. Lebih lanjut, *integer* variable-*byte* juga memperlukan beberapa operasi CPU untuk proses *decode*.

Suatu deskripsi *pseudo code* dari algoritma pengkodean variable-*byte* ditujukkan pada Gambar 2. Dalam algoritma ini *integer* dikodekan menjadi *integer* v. *Integer* yang terkodekan diterima kembali dengan memproses *byte* dari *array* A, dimana setiap *byte* disimpan dalam suatu *integer byte* tunggal *i*, sementara *least significnat bit* adalah satu.



Gambar 2 Perbandingan antara Beberapa Metode Pengkodean

Elias coding merupakan metode non parametik dari pengkodean *integer* sebagai contoh digunakan pada teks ukuran besar pada indeks database dan aplikasi khusus. Elias coding, seperti halnya dengan skema lainnya, memungkinkan pengkodean ambigu dari *integer* dan tidak memerlukan pemisah antara setiap *integer* dari *array* yang disimpan. Terdapat dua metode pengkodean Elias, yaitu *Gamma* dan *Delta* Coding.

Pada kode *Elias Gamma*, suatu *integer* positif x direpresentasikan oleh $1 + [\log_2 x]$ dalam bentuk unary (sehingga $[\text{Log}_2 x]$ 0 *bit* diikuti oleh 1-*bit*), diikuti oleh representasi *bit* dari x tanpa mengandung

most significant bit. Sehingga 9 direpresentasikan sebagai 0001001, karean [1 + log2 9] = 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*. Pengkodeaan *Gamma* hanya efisien untuk *integer* kecil tetapi tidak cocok untuk *integer* yang besar, dimana kode terparameter dari *Elias Code* yang lain adalah *Delta code* lebih cocok digunakan.

Kode Elias *Delta* menghasilkan bentuk lebih panjang dibandingkan dengan kode *Gamma* untuk *integer* yang kecil, tetapi untuk *integer* yang besar seperti nilai dalam dokumen dalam suatu indeks yang besar, maka situasi seperti ini menjadi terbalik. Untuk suatu *integer* x, kode *Delta* menyimpan representasi kode *Gamma* $1 + \text{Log}_2 x$, diikuti dengan representasi biner dari *x* dikurangi dengan *most significant bit*.

Integer terpilih untuk kode Elias mempuyai jangkauan 1-30 seperti ditunjukkan pada Tabel 3.1. Perbandingan dari panjang code dalam *bit* untuk *integer* dalam jangkauan 1 hingga sekitar 1 juta dikodekan dengan *Gamma*, *Delta*, pengkodean variable-*byte* dan kode *Golomb*. Tabel 1 Representasi dari *Integer* Terpilih dalam Jangkauan 1−30 sebagai *Integer* Tidak Terkompresi 8-*bit*

Tabel 1. Representasi dari Integer Terpilih

Decimal	Uncompressed	Elias Gamma	Elias Delta	Golomb ($k = 3$)	Golomb ($k = 10$)	Variable-byte
1	00000001	1	1	1 10	1 001	0000001 0
2	00000010	0 10	0 100	1 11	1 010	0000010 0
3	00000011	0 11	0 101	01 0	1 011	0000011 0
4	00000100	00 100	0 1100	01 10	1 100	0000100 0
5	00000101	00 101	0 1101	01 11	1 101	0000101 0
6	00000110	00 110	0 1110	001 0	1 1100	0000110 0
7	00000111	00 111	0 1111	001 10	1 1101	0000111 0
8	00001000	000 1000	00 100000	001 11	1 1110	0001000 0
9	00001001	000 1001	00 100001	0001 0	1 1111	0001001 0
10	00001010	000 1010	00 100010	0001 10	01 000	0001010 0
11	00001011	000 1011	00 100011	0001 11	01 001	0001011 0
12	00001100	000 1100	00 100100	00001 0	01 010	0001100 0
13	00001101	000 1101	00 100101	00001 10	01 011	0001101 0
14	00001110	000 1110	00 100110	00001 11	01 100	0001110 0
15	00001111	000 1111	00 100111	000001 0	01 101	0001111 0
16	00010000	0000 10000	00 1010000	000001 10	01 1100	0010000 0
20	00010100	0000 10100	00 1010100	0000001 11	001 000	0010100 0
25	00011010	0000 11001	00 1011001	000000001 10	001 101	0011010 0
30	00011110	0000 11110	00 1011110	00000000001 0	0001 000	0011110 0

2.1. Kompresi dengan *Elias Gamma*

Pengkodean *Elias Gamma* untuk tujuan kompresi hanya efektif untuk bilangan *integer* yang kecil. Dari Tabel 3.1 terlihat bahwa pengkodean *Elias Gamma* ini hanya cocok diterapkan untuk bilangan desimal 1 hingga 15 karena pengkodean kelima belas bilangan ini hanya memerlukan jumlah *bit* 1 hingga 7 *bit* sehingga efisiensi penyimpanan di dapat.

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.

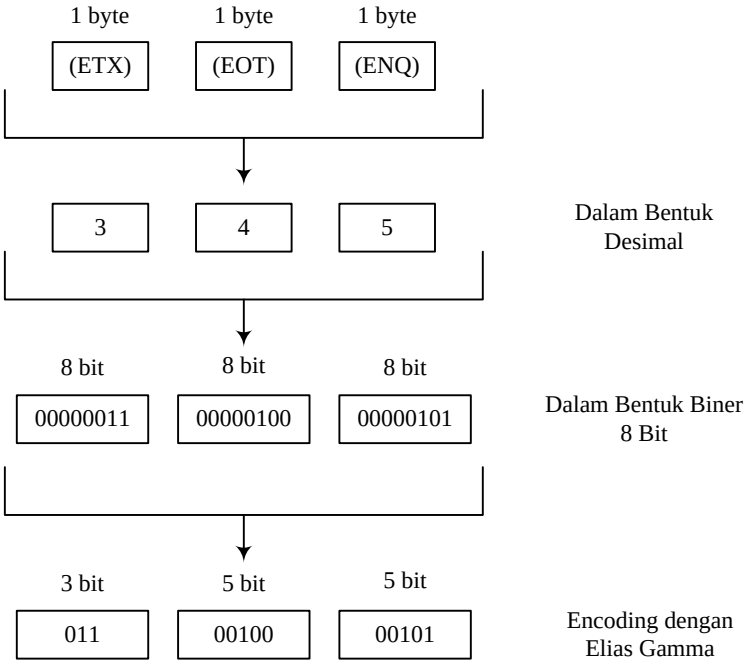
Dari kedua jenis *Elias Code*, *Elias Gamma* merupakan jenis yang paling sederhana. Cara kerja dari pengkodean *Gamma* dimana menggunakan bilangan *integer* $x = 13$. Untuk mengkodekan sebuah bilangan natural $x \in N = \{1, 2, 3, ...\}$, secara mudah dilakukan dengan menuliskan bentuk representasi biner diikuti dengan jumlah nol yang tentukan dengan rumus $[\text{Log}_2 x]$ dimana tanda [ ] berarti hasil diambil hanya merupakan bilangan bulatnya. Dengan cara seperti ini juga $[\text{Log}_2(x)] + 1$ merupakan jumlah digit yang diperlukan untuk menuliskan *x* dalam bentuk biner.

Sebagai contoh, untuk x = 13, representasi binernya adalah 1101, dan $[\text{Log}_2 13] = 3$ sehingga *code word Elias Gamma* adalah $C_\gamma(13) = 0001101$.

Dengan cara seperti ini mudah untuk memverifikasi kode secara *instant* yaitu dengan mempelajari prosedur *decoding*. *Decoder* dimulai dengan menghitung jumlah, misalkan *n* dari jumlah nol yang terdapat dibagian awal dari *code word*, jika *n* bernilai nol, maka bilang yang dikodekan adalah 1; jika *n* bukan merupakan bilangan nol, maka *decoder* membaca digit biner $n + 1$ dan mengkodekannya dengan bilangan biner yang cocok. Sebagai contoh di atas *code word* 000**1101** berarti terdapat tiga buah

nol di depan *code word* sehingga *n* = 3 sehingga pembacaan dilakukan atas 4 *bit* berikutnya dapat dihasilkan *bit* 1101 yang bernilai 13.

Proses kompresi dengan memanfaatkan *Elias Gamma* misalkan pada pembacaan suatu isi *file* terdapat *string* (ETX) (EOT) (ENQ) yang mempunyai nilai desimal berturut-turut adalah 2,3,5 proses kompresinya adalah melakukan pembacaan secara per *byte* dan mengubah ke dalam bentuk *code word* dengan *Elias Gamma*.



Gambar 3. Cara Kerja *Elias Gamma*

Dari hasil yang didapat terdapat pengurangan menjadi 13 *bit* dari 24 *bit* semula dengan hasil seperti ini dikatakan telah terjadi reduksi *bit* dalam representasi data. hasil di atas memberikan rasio di atas sebagai berikut:

$$\text{Rasio Kompr} = \frac{\text{Jumlah Bit}}{\text{Jumlah Bit}}$$

$$\text{Rasio Kompr} = \frac{24}{13}$$

$$\text{Rasio Kompr} = \frac{24}{}$$

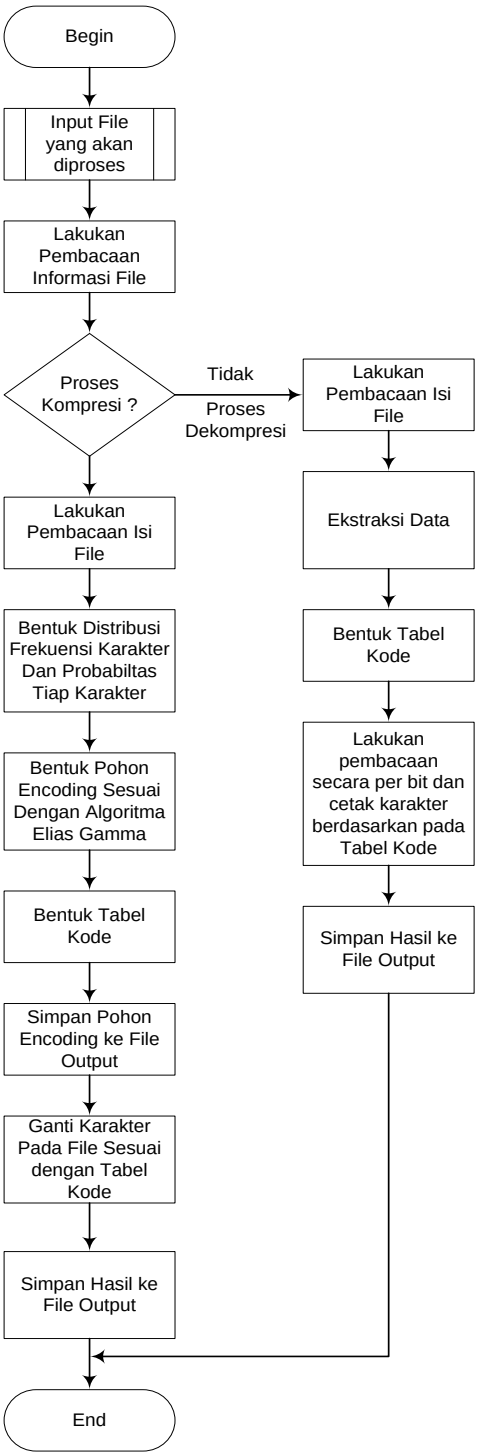
2.2. Dekompresi dengan *Elias Gamma*

Proses dekompresi dengan *Elias Gamma* dapat dilakukan secara langsung tanpa harus menggunakan suatu tabel. Langkah pertama proses penelusuran akan menemukan angka 0 sehingga sehingga n = 1 pembacaan dilakukan terhadap n + 1 sehingga dua *bit* berikutnya diambil mendapatkan *byte* pertama = 011. Penelusuran dilakukan terhadap dua nol berikutnya sehingga didapat hasil *n* = 2 pembacaan dilakukan terhadap tiga *bit* berikutnya sehingga didapat *byte* kedua = 00100. Kemudian pembacaan mendapatkan dua buah nol dan dilakukan pembacaan terhadap tiga *bit* berikutnya sehingga didapat *byte* ketiga = 00101.

Ketiga *byte* tersebut di-decode balik dengan menggunakan *Elias Gamma* sehingga didapat nilai desimal 3, 4, 5 kembali yang merupakan representasi dari *string* (ETX) (EOT) (ENQ). Algoritma program kompresi ini dapat dijabarkan dalam bentuk sebuah diagram alir. Langkah pertama sebelum melakukan proses kompresi dan dekompresi *file* adalah menginput *file* yang akan diproses. *File* yang diinput dapat berupa *file* biner ataupun *file* teks. Setelah itu program akan membaca informasi *file* berupa ukuran *file* dalam satu *byte*, tanggal dan waktu pembuatan *file*, attribut *file*, *path* tempat *file* diinput. Berikutnya adalah penentuan proses oleh *user* apakah memilih untuk mengkompresi *file* tersebut atau sebaliknya

Jika dipilih proses kompresi maka pertama sekali seluruh isi *file* dibaca secara per *byte* untuk membentuk sebuah tabel distribusi karakter yang berjumlah 256 buah (sesuai dengan tabel kode ASCII). Tiap karakter akan dicacah frekuensinya pada *file* tersebut. Berikutnya adalah menentukan probabilitas tiap karakter dan kemudian membentuk pohon *encoding* dengan algoritma *Shannon-Fano* untuk mencari tabel kode tiap karakter. Setelah itu pohon *encoding* yang dibuat dalam bentuk *array* akan disimpan pada *file output*.

Selanjutnya tiap karakter akan dikodekan sesuai dengan tabel kode yang dihasilkan dari pohon *encoding* dan hasilnya disimpan ke *file output* seperti ditunjukkan pada gambar 4.



Gambar 4. Diagram Alir Program

Untuk proses dekompresi maka akan dilakukan pembacaan seluruh isi *file*. Kemudian hasil pohon *encoding* dari proses kompresi akan dibaca dan dibentuk kembali. Isi *file* yang akan didekompresi dibaca secara per *bit* dan dilakukan proses pengembalian karakter berdasarkan pohon *encoding*. Untuk tiap karakter yang ditemukan akan disimpan ke *file output*.

Langkah pertama sebelum melakukan proses kompresi dan dekompresi *file* adalah meng-*input file* yang akan diproses. *File* yang di-*input* dapat berupa *file* biner ataupun *file* teks. Setelah itu program akan membaca informasi *file* berupa ukuran *file* dalam satu *byte*, tanggal dan waktu pembuatan *file*, *attribut file*, *path* tempat *file* di-*input*. Berikutnya adalah penentuan proses oleh *user* apakah memilih untuk mengkompresi *file* tersebut atau sebaliknya

```
Prosedur encoding dari elias Gamma Encode(karakter source, karakter dest)
{
    Bentuk IntReader dari intreader(source);
    Bentuk BitWriter dari bitwriter(dest);
    Lakukan selama (intreader.hasLeft())
    {
     Set num = intreader.getInt();
     Set l = log2(num);
     Untuk (nilai a=0; a < l; setiap nilai a bertambah 1)
     {
        Jika bitwriter.putBit bernilai (false); //tempatkan 0
                untuk mengindikasikan seberapa banyak bit yang harus
                ditempatkan
                Jika bitwriter.putBit bernilai (true);//tandai akhir bit
                dengan nol
     Untuk (nilai a=0; a < l; setiap nilai a bertambah
                1//Tuliskan output dalam bentuk biner
     {
              Jika (num dan 1 << a)
                Set bitwriter.putBit sama dengan (true);
              Jika Tidak
                Set bitwriter.putBit sama dengan (false);
     }
    }
    Lakukan prosedur intreader.close();
    Lakukan prosedur bitwriter.close();
}

Prosedur elias Gamma Decode(karakter source, karakter dest)
{
    Bentuk BitReader dari bitreader(source);
    Bentuk BitWriter dari bitwriter(dest);
    Set numberBits = 0;
    Lakukan selama (bitreader.hasLeft())
    {
      Lakukan selama (bitreader.getBit() Tidak Sama dengan
              False|| Set bitreader.hasLeft())numberBits = 1 +
              numberBits;
      Set current = 0;
      Untuk (nilai a=0; a < numberBits; untuk setiap nilai
              bertambah 1) //Baca setiap nilai bit
      {
         Jika (bitreader.getBit())
           current = current + 1 << a;
      }
      //Tulis dalam bentuk nilai 32 bit
       Current = current or ( 1 << numberBits ) ;
  Untuk (nilai a=0; a < 32; setiap nilai a bertambah 1) //Baca
```

```
            jumlah bit
      {
        Jika (current And (1 << a))
          Set bitwriter.putBit  = true;
        Jika Tidak
          Set bitwriter.putBit = false;
      }
    }
}
```

3. Hasil dan Analisis

Untuk pengujian program ini dilakukan terhadap beberapa jenis *file* seperti *file text*, *file audio*, *file grafik*, dan *file document Word* baik untuk proses kompresi dan dekompresi. Pengujian terhadap *file* sama dilakukan 3 (tiga) kali untuk mendapatkan hasil waktu proses dan hasilnya kemudian dirata-ratakan.

Tabel 2. Tabel Hasil Pengujian Kompresi Algoritma Elias Gamma

Jenis File	Ukuran File (byte)	Ukuran File Setelah Dikompresi (byte)	Rasio Kompresi	Waktu Proses (ms)			
				Test I	Test II	Test III	Rata-Rata
Uji1.doc	31.232	11.839	37,91%	20	15	16	17,00
Uji2.doc	41.984	25.004	59,56%	32	15	16	21,00
Uji1.wav	119.384	113.169	94,79%	46	47	32	41,67
Uji2.wav	171.100	161.777	94,55%	63	47	62	57,33
Uji1.bmp	196.664	195.108	99,21%	62	63	62	62,33
Uji2.bmp	12.406	5.057	40,76%	28	25	27	26,67
Uji1.txt	65.536	10.476	15,99%	16	16	15	15,67
Uji2.txt	22.421	21.538	94.55%	16	16	15	15,67

Dari hasil pengujian antara algoritma Elias Gamma dengan algoritma Huffman terlihat bahwa algoritam Elias Gamma mempunyai rasio kompresi yang lebih besar dibandingkan dengan algoritma Huffman. Hal ini berarti algoritma Elias Gamma dapat mereduksi data lebih banyak dibandingkan dengan algoritma Huffman sehingga dapat disimpulkan bahwa algoritma Elias Gamma lebih unggul dibandingkan dengan algoritma Huffman.

Tabel 3. Tabel Hasil Pengujian Dekompresi Elias Gamma

Jenis File	Ukuran File (byte)	Ukuran File Setelah Didekompresi (byte)	Rasio Kompresi	Waktu Proses (ms)			
				Test I	Test II	Test III	Rata-Rata
Uji1.doc.elg	11.839	31.232	263,81%	16	15	16	15,67
Uji2.doc.elg	25.004	41.984	167,91%	26	25	26	25,67
Uji1.wav.elg	113.169	119.384	105,49%	31	30	32	31,00
Uji2.wav.elg	161.777	171.100	105,76%	31	32	30	31,00
Uji1.bmp.elg	195.108	196.664	100.8%	47	48	46	47,00
Uji2.bmp.elg	5.057	12.406	245,32%	15	16	15	15,33
Uji1.txt.elg	10.476	65.536	625,58%	16	17	16	16,33
Uji2.txt.elg	21.538	22.421	104,10%	25	25	26	25,33

Setelah dilakukan proses kompresi terhadap *file* tersebut maka ukuran *file* menjadi 1.241 *byte* dengan rasio kompresi sebesar 55,45% dengan waktu proses selama 16 milisekon, 15 milisekon, dan 16 milisekon. Berarti rata-rata waktu prosesnya adalah 15,67 milisekon. File hasil akan mempunyai ekstensi *.ELG.

Dari hasil pengujian yang didapat diketahui bahwa jenis *file* tidak berpengaruh pada rasio kompresi. Rasio kompresi hanya berpengaruh pada isi *file*, semakin banyak perulangan karakter pada sebuah *file* maka rasio kompresinya semakin kecil yang berarti ukuran *file* hasil kompresi semakin kecil pula.

Waktu proses baik kompresi ataupun dekompresi *file* bergantung pada ukuran *file*, semakin besar

ukuran *file* maka semakin lama waktu yang diperlukan untuk memprosesnya. Proses dekompresi lebih cepat dibandingkan dengan proses kompresi hal ini disebabkan oleh pada proses kompresi dilakukan pembacaan isi *file* secara per *byte* untuk menentukan frekuensi karakter setelah itu dilanjutkan lagi dengan proses pembentukan pohon *encoding* baru dan penyimpanan pohon *encoding* ke *file* hasil, kemudian dikompresi sedangkan pada proses dekompresi hanya dilakukan pembentukan kembali pohon *encoding* yang telah disimpan serta karakter hasil kompresi.

4. Kesimpulan

Dari hasil dan analisis, maka dapat disimpulkan bahwa tingkat rasio kompresi dari algortima Elias Gamma mempunyai rentang 60% – 80%. Pengujian baik untuk proses kompresi dan dekompresi menunjukkan kecepatan proses rata-rata cukup cepat karena Elias Gamma melakukan proses *encoding* dalam proses kompresinya. Waktu yang dibutuhkan untuk kompresi tidak bergantung pada besar ukuran file dan begitu pula waktu dekompresi tidak tergantung pada waku kompresi. Secara rata-rata waktu kompresi membutuhkan waktu lebih daripada waktu dekompresi. Besarnya rasio kompresi tergantung kepada jumlah bit per sample dan sample rate. Proses kompresi dengan menggunakan Elias Gamma mempunyai keterbatasan karena hanya dapat merepresentasikan 7 karakter pertama dengan jumlah *bit* di bawah 8 *bit*.

Referensi

Buku Teks :

[1.] Ferrianto, Gozali, 2008, *Analisis Perbandingan Kompresi Data Dengan Teknik Arithmatic Coding dengan Run Length Encoding*, Jurnal Ilmiah: Volume 4, Nomor 1, Agustus 2004, Halaman 37-52, Jurusan Teknik Elektro, Universitas Trisakti, Jakarta.

[2.] Haryanto. R.I., Kompresi Data Dengan Algoritma *Huffman* Dan Perbandingannya Dengan Algoritma Lzw Dan Dmc, Makalah Struktur Diskrit, Teknik Informatika ITB, Desember 2009,

[3.] Merdiyan, M., Indarto, W., (2005), Implementasi Algoritma run length, half byte dan Huffman untuk kompresi file, diakses dari http://journal.uii.ac.id/index.php/Snati/article/viewFile/1391/1171.

[4.] Nelson, M. and Gailly, J. L., 2002, The Data Compression Book, Second Edition, M & T Books.

[5.] Salomon, D., 2004, Data Compression: The Complete Reference, Second Edition, Springer.

[6.] Subarkah. A.F., Rancang bangun aplikasi kompresi file Menggunakan metode lzw berbasis java, skripsi jurusan teknik informatika, Universitas Islam Negeri (UIN) Maulana Malik Ibrahim, Malang, 2010.

[7.] Suyanto, 2008, Tree (Pohon), Praktikum Struktur Data 1, Universitas Guna Dharma.

[8.] Wea. K.S.M., Willy Sudiarto R., Antonius Rachmat C, Aplikasi Player Untuk Menjalankan File Wave Yang Terkompresi Dengan Metode Huffman, Jurnal Informatika, Volume 6 Nomor 1, April 2010, hal 4, Program Studi Teknik Informatika, Universitas Kristen Duta Wacana, Yogyakarta.