KOMPRESI FILE TEKS MENGGUNAKAN KOMBINASI ALGORITMA *LIMPEL-ZIV-WELCH* (*LZW*) DAN *HUFFMAN*

Kurniadi Prajitno **Drs. Marji, MT** **Edy Santoso, SSi., M.Kom**
Program Studi Ilmu Komputer
Universitas Brawijaya Malang Pembimbing 1 Pembimbing 2

Abstrak : Dengan berkembangnya Teknologi komputer sekarang ini dapat dimanfaatkan untuk berbagai hal, salah satunya adalah menyimpan *file*. Semakin besar ukuran *file* yang disimpan di dalam komputer maka semakin besar pula tempat penyimpanannya. Semakin besar tempat penyimpanan yang dibutuhkan maka semakin besar pula biaya yang dikeluarkan. Selain itu *file* yang memiliki ukuran besar dapat menimbulkan masalah pada saat transmisi *file*. Oleh karena itu kemudian muncul metode-metode yang bertujuan untuk memampatkan *file* agar dapat menghemat tempat penyimpanan *file-file* tersebut, salah satunya dengan cara kompresi. Dengan adanya kompresi diharapkan dapat menghemat biaya serta waktu yang dikeluarkan guna menambah fasilitas media penyimpanan *file* pada komputer serta mempercepat proses transfer *file*.

1. PENDAHULUAN

Dengan berkembangnya Teknologi komputer di masa sekarang ini dapat dimanfaatkan untuk berbagai hal, salah satunya adalah menyimpan *file*. Semakin besar ukuran *file* yang akan di simpan di dalam komputer maka membutuhkan semakin besar pula tempat penyimpanannya. Semakin besar tempat penyimpanan yang dibutuhkan maka semakin besar pula biaya yang dikeluarkan. Selain itu *file* yang memiliki ukuran yang besar dapat menimbulkan masalah pada saat transmisi *file*. Oleh karena itu kemudian muncul metode-metode yang bertujuan untuk memampatkan *file* agar dapat menghemat tempat penyimpanan *file-file* tersebut, salah satunya dengan cara kompresi [6]. Dengan adanya kompresi diharapkan dapat menghemat biaya serta waktu yang dikeluarkan guna menambah fasilitas media penyimpanan *file* pada komputer serta mempercepat proses transfer *file*

Kompresi *file* berarti suatu teknik untuk memampatkan *file* agar diperoleh *file* dengan ukuran yang lebih kecil daripada ukuran aslinya sehingga lebih efisien dalam menyimpannya serta mempersingkat waktu pertukaran *file* tersebut [6]. Dengan kata lain, kompresi *file* sebenarnya adalah proses meminimalkan ukuran *file* atau berkas dengan mengurangi *file* yang berulang, karena umumnya pada sebuah *file* sering terjadi pengulangan.

Pada kompresi *file*, terdapat dua tipe macam kompresi, yaitu *lossless compression* dan *lossy compression*. Pada *lossless compresion*, semua informasi yang ada pada *file* akan kembali menjadi seperti aslinya, tidak ada informasi yang hilang. Teknik ini biasanya digunakan untuk dokumen-dokumen, *file executable*, dan lainnya. Jika kehilangan sebuah informasi merupakan hal yang fatal bagi *file-file* tersebut. Sedangkan pada *lossy compression*, tidak semua informasi yang ada akan kembali seperti semula. Hanya informasi-informasi inti yang dikembalikan. Hal ini terjadi, karena pada *lossy compression* informasi-informasi yang tidak berguna akan dihilangkan [7]. Walaupun ada informasi yang hilang, namun hal ini tidak terlalu disadari oleh pengguna. Teknik ini biasanya digunakan pada *file* video, gambar, suara yang dimana *file-file* tersebut biasanya berukuran besar.

Banyak teknik algoritma yang dapat digunakan untuk melakukan pengkompresian sebuah *file*, contohnya adalah algoritma *Huffman*, algoritma LZ (*Lempel-Ziv*), algoritma

DMC (*Dinamyc Markov Compression*), *Block-Shorting Lossless, Run-Lenght, Shannon-Fano, Arithmetic, PPM* (*Prediction by Partial Matching*), *Burrows-Wheeler Block Sorting*, dan *Half Byte*, dll [11].

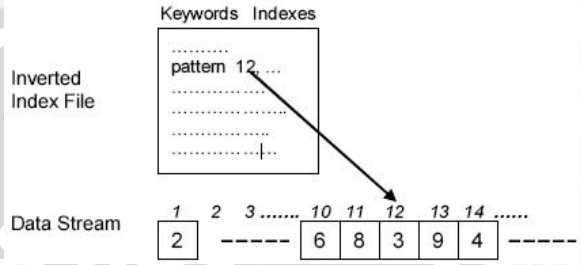
Beberapa software kompresi yang banyak digunakan para pengguna komputer dewasa ini diantaranya adalah WinZip (menghasilkan format .zip) dan WinRAR (menghasilkan format .rar). Selain itu, telah banyak penelitian yang dilakukan mengenai pengkompresian sebuah file dengan berbagai macam metode algoritma., misalnya Aplikasi Pohon Dalam Teknik Kompresi Data Dengan Algoritma *Huffman* dan Algoritma *Shannon-Fano* [9].

Ada beberapa faktor yang sering menjadi pertimbangan dalam memilih suatu metode kompresi yang tepat, yaitu kecepatan kompresi, sumber daya yang dibutuhkan (memori, kecepatan PC), ukuran *file* hasil kompresi. Berdasarkan uraian latar belakang di atas maka judul yang diambil penulis dalam penelitian tugas akhir ini adalah "**Kompresi *File* Menggunakan Kombinasi Algoritma Kompresi LZW (*Limpel-Ziv-Welch)* dan Huffman**". Penulis memilih Algoritma LZW dan Huffman karena berdasarkan penelitian sebelumnya algoritma Huffman dalam pengolahan *file* memungkinkan kompresi *file* rata-rata mencapai 45,8% tanpa menyebabkan adanya *file* yang hilang. Rasio kompresi mencapai maksimal saat mengkompresi *file* bertipe teks. Untuk algoritma LZW saat mengkompresi memungkinkan kompresi *file* rata-rata mencapai 51,6 % tanpa menyebabkan adanya *file* yang hilang. Rasio Kompresi tersebut mencapai maksimal saat mengkompresi *file* bertipe teks, sehingga penulis mencoba menggabungkan kedua algoritma ini karena kedua Algoritma tersebut memiliki rasio tertinggi dari beberapa metode kompresi file yang ada. Dan bedasarkan penelitian pula kompresi pertama dilakukan dengan LZW kemudian dilanjutkan dengan Huffman memberikan rasio kompresi yang cukup tinggi daripada dikompresi dengan Huffman kemudian dilanjutkan dengan LZW, sehingga dalam Tugas Akhir ini Penulis memilih kompresi pertama dilakukan dengan LZW dan selanjutnya dikompresi dengan algoritma Huffman [1].

2. TINJAUAN PUSTAKA

2.1 *Algoritma Limpel-Ziv-Welch*(*LZW*)

Algoritma LZW dikembangkan dari metode kompresi yang dibuat oleh Abraham Ziv dan Jacob Lampel pada tahun 1977 yaitu Algotitma LZ77. Kemudian Pada tahun 1978 Terry Welch



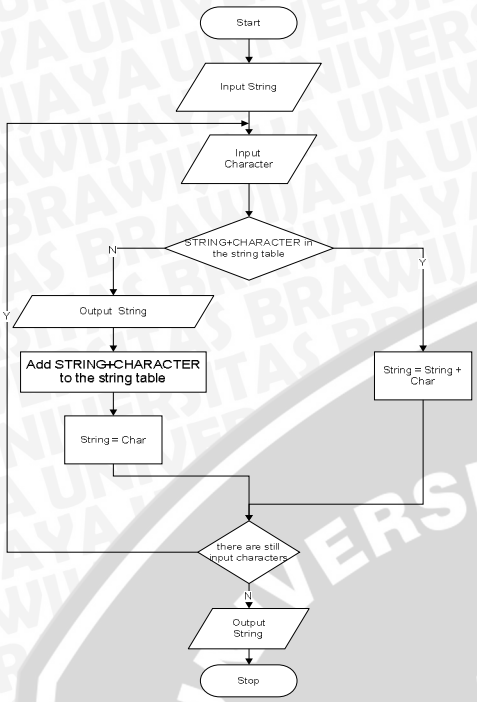
Gambar 2.1 Ilustrasi Kompresi LZW

Dalam bentuk yang sederhana Algoritma kompresi *LZW* dapat dilihat seperti di bawah ini.

```
STRING = get input character
WHILE there are still input characters Do
  CHARACTER = get input character
  if STRING+CHARACTER in the string table
  then
   STRING = STRING+CHARACTER
  ELSE
   OUTPUT the code for STRING
   Add STRING+CHARACTER to the string
   table
   STRING = CHARACTER
  END of IF
END of WHILE
OUTPUT the code for STRING
```

Algortima ini menggunakan teknik dictionary dalam kompresinya. Dimana string karakter digantikan oleh kode table yang dibuat setiap ada string yang masuk. Tabel dibuat untuk referensi masukan string selanjutnya. Ukuran tabel dictionary pada algoritma LZW asli adalah 4096 sampel atau 12 bit, dimana 256 sampel pertama digunakan untuk table karakter single (Extended ASCII), dan sisanya digunakan untuk pasangan karakter atau string dalam data input.

Algoritma LZW melakukan kompresi dengan mengunakan pendekatan yang bersifat adaptif dan efektif dimana kode table 256 hingga 4095 untuk mengkodekan pasangan byte atau string, karena metode ini banyak string yang dapat dikodekan dengan mengacu pada string yang telah muncul sebelumnya dalam teks.



Gambar 2.2 *Flowchart* Kompresi LZW [10]

Sebagai contoh, *String* ”ABBABABAC” akan dikompresi dengan LZW. Isi *dictionary* pada awal proses diset dengan 3 karakter dasar yang ada : ”A”,”B”,dan”C”. Tahapan proses kompresi ditujukan pada tabel 2.1. Kolom Posisi menyatakan posisi sekarang dari *steam* karakter dan kolom karakter menyatakan karakter yang terdapat pada posisi tersebut. Kolom *dictionary* menyatakan *string* baru yang sudah ditambahakan ke dalam *dictionary* dan nomor indeks untuk *string* tersebut ditulis dalam kurung siku.

Kolom *output* menyatakan kode *output* yang dihasilkan oleh langkah kompresi.

Tabel 2.1 Tahapan Proses Kompresi LZW

Langkah	Posisi	Karakter	*Dictionary*	*Output*
1	1	A	[4]AB	[1]
2	2	B	[5]BB	[2]
3	3	B	[6]BA	[2]
4	4	A	[7]ABA	[4]
5	6	A	[8]ABAC	[7]
6	9	C	---	[3]

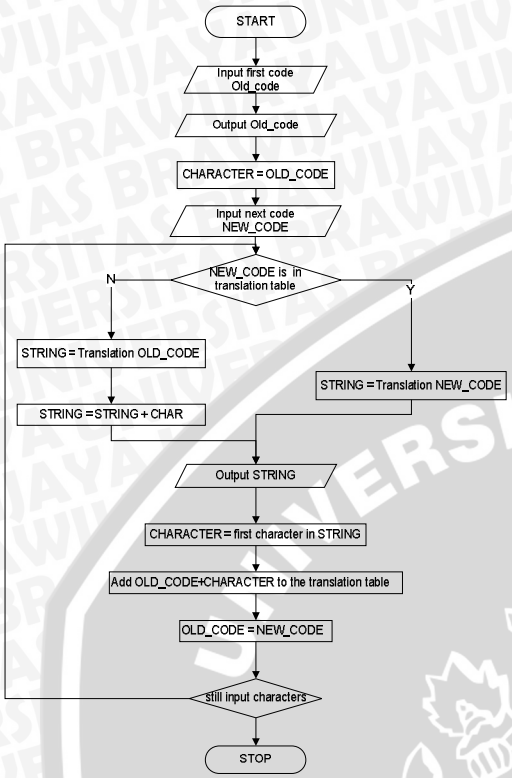
Proses dekompresi pada LZW dilakukan dengan prinsip yang sama seperti proses kompresi. Pada awalnya, *dictionary* diinisialisasi dengan semua karakter dasar yang sudah ada. Lalu pada setiap langkah, kode dibaca 1 per 1 dari *steam* kode, dikeluarkan *string* dari *dictionary* yang berkorespondensi dengan kode tersebut atau disebut juga dengan istilah translasi, dan ditambahkan *string* baru ke dalam *dictionary*[10].

Dalam bentuk yang paling sederhana Algoritma kompresi *LZW* dapat dilihat seperti di bawah ini.

```
Read OLD_CODE
Output OLD_CODE
CHARACTER = OLD_CODE
WHILE there are still input characters Do
  Read NEW_CODE
  If NEW_CODE is not in translation table
  Then
    STRING = get translation of OLD_CODE
    STRING = STRING + CHARACTER
  ELSE
    STRING = Get translation of NEW_CODE
  END IF
OUTPUT STRING
CHARACTER = first character in STRING
Add OLD_CODE+CHARACTER to the translation
table
OLD_CODE = NEW_CODE
END WHILE
```

Proses dekompresi pada LZW dilakukan dengan prinsip yang sama seperti proses kompresi. Pada awalnya, dictionary diinisialisasi dengan semua karakter dasar yang ada. Lalu pada setiap langkah, kode dibaca satu per satu dari stream kode, dikeluarkan string dari dictionary yang berkorespondensi dengan kode tersebut, dan ditambahkan string baru ke dalam dictionary

Algoritma dekompresi LZW dapat dilihat pada gambar 2.3



Gambar 2.3 *Flowchart* Dekompresi LZW [10]

2.2 Algoritma *Huffman*

Algoritma Huffman, yang dibuat oleh seorang mahasiswa MIT yang bernama David Huffman pada tahun 1953, merupakan salah satu metode paling lama dalam kompresi. Algoritma huffman menggunakan prinsip pengkodean yang mirip dengan kode morse, yaitu tiap karakter (simbol) dikodekan hanya dengan rangkaian beberapa *bit*, dimana karakter yang sering muncul dikodekan dengan rangkaian bit yang lebih pendek dan karakter yang jarang muncul dikodekan dengan rangkaian *bit* yang lebih panjang.

Berdasarkan tipe peta kode yang digunakan untuk mengubah pesan awal (isi data yang diinputkan) menjadi sekumpulan *codeword*, algoritma Huffman termasuk kedalam kelas algoritma yang menggunakan metode statik. Metoda statik adalah metoda yang selalu menggunakan peta kode yang
sama, metoda ini membutuhkan dua fase (two-pass): fase pertama untuk menghitung probabilitas kemunculan tiap simbol dan menentukan peta kodenya, dan fase kedua untuk mengubah pesan menjadi kumpulan kode yang akan di transmisikan [11].

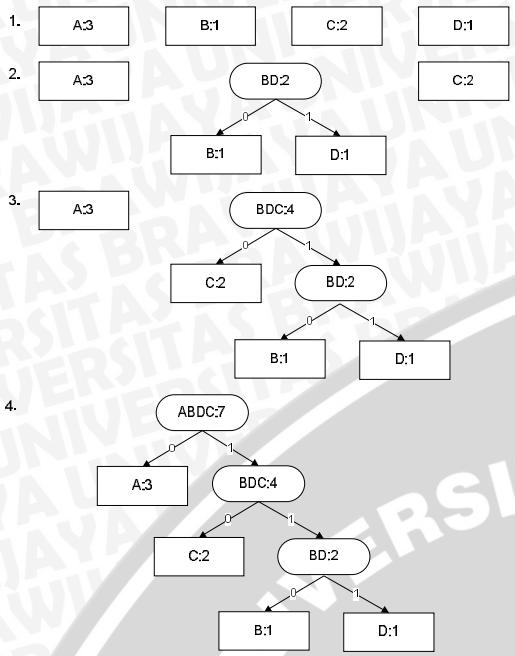
2.2.1 Penghitungan Frekuensi yang Muncul

Langkah awal sebelum melakukan kompresi dengan menggunakan algoritma *Huffman* menghitung frekuensi kemunculan karakter yang muncul dalam string. Setelah didapat jumlah frekuensi yang dikelompokkan sesuai dengan nilai ASCIInya, kemudian diurutkan dari yang terkecil sampai yang terbesar. Hal ini dimaksudkan untuk mempermudah pembuatan pohon Huffman

2.2.2 Pembentukan Pohon

Kode Huffman pada dasarnya merupakan kode prefiks (*prefix code*). Kode prefiks adalah himpunan yang berisi sekumpulan kode biner, dimana pada kode prefik ini tidak ada kode biner yang menjadi awal bagi kode biner yang lain. Kode prefiks biasanya direpresentasikan sebagai pohon biner yang diberikan nilai atau label. Untuk cabang kiri pada pohon biner diberi label 0, sedangkan pada cabang kanan pada pohon biner diberi label 1.Rangkaian bit yang terbentuk pada setiap lintasan dari akar ke daun merupakan kode prefiks untuk karakter yang berpadanan. Pohon biner ini biasa disebut pohon Huffman. Poses pembentukan pohon Huffman ditunjukkan pada gambar 2.4

Sebagai contoh, dalam kode ASCII string 7 huruf “ABACCDA” membutuhkan representasi 7 × 8 bit = 56 bit (7 byte), dengan rincian sebagai berikut:
A = 01000001 B = 01000010 A = 01000001 C = 01000011 C = 01000011 D = 01000100 A = 01000001
Pada string di atas, frekuensi kemunculan A = 3, B = 1, C = 2, dan D = 1



Gambar 2.4 Pohon Huffman untuk Karakter "ABACCDA"

2.2.3 Encoding

Encoding adalah cara menyusun string biner dari teks yang ada. Proses encoding untuk satu karakter dimulai dengan membuat pohon Huffman terlebih dahulu. Setelah itu, kode untuk satu karakter dibuat dengan menyusun nama string biner yang dibaca dari akar sampai ke daun pohon Huffman. Langkah-langkah untuk men-encoding suatu string biner adalah sebagai berikut

1. Tentukan karakter yang akan di-encoding
2. Mulai dari akar, baca setiap bit yang ada pada cabang yang bersesuaian sampai ketemu daun dimana karakter itu berada
3. Ulangi langkah 2 sampai seluruh karakter diencoding

Sebagai contoh kita dapat melihat tabel 2.2 yang merupakan kode *Huffman* untuk karakter "ABCD"

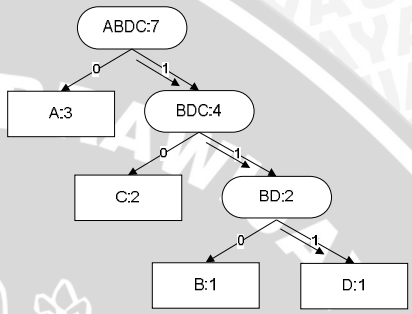
Tabel 2.2. Kode Huffman untuk Karakter "ABCD"

Karakter	String Biner Huffman
A	0
B	110
C	10
D	111

2.2.4 Proses Decoding

Decoding merupakan kebalikan dari encoding. Decoding berarti menyusun kembali data dari string biner menjadi sebuah karakter kembali. Decoding dapat dilakukan dengan dua cara, yang pertama dengan menggunakan pohon Huffman dan yang kedua dengan menggunakan tabel kode Huffman.

Gambar 2.5 menunjukan proses decoding dengan menggunakan pohon Huffman.



Gambar 2.5 Proses Decoding dengan Menggunakan Pohon Huffman

Setelah kita telusuri dari akar, maka kita akan menemukan bahwa string yang mempunyai kode Huffman "111" adalah karakter D. Cara yang kedua adalah dengan menggunakan tabel kode Huffman. Sebagai contoh kita akan menggunakan kode Huffman pada Tabel 2.2 untuk merepresentasikan string "ABACCDA". Dengan menggunakan Tabel 2.2 string tersebut akan direpresentasikan menjadi rangkaian bit : 0 110 0 10 10 1110. Jadi, jumlah bit yang dibutuhkan hanya 13 bit. Dari Tabel 2.2 tampak bahwa kode untuk sebuah simbol/karakter tidak boleh menjadi awalan dari kode simbol yang lain guna menghindari keraguan (ambiguitas) dalam proses dekompresi. Karena tiap kode Huffman yang dihasilkan unik, maka proses decoding dapat dilakukan dengan mudah. Contoh: saat membaca kode bit pertama dalam rangkaian bit "011001010110", yaitu bit "0", dapat langsung disimpulkan bahwa kode bit "0" merupakan pemetaan dari simbol "A". Kemudian baca kode bit selanjutnya, yaitu bit "1". Tidak ada kode Huffman "1", lalu baca kode bit selanjutnya, sehingga menjadi "11". Tidak ada juga kode Huffman "11", lalu baca lagi kode bit berikutnya, sehingga menjadi "110". Rangkaian kode bit "110" adalah pemetaan dari simbol "B" [11].

2.3 Rasio Kompresi

Proses kompresi adalah proses encoding yang menghasilkan data yang sudah dikompresi yang disebut aliran data encoded. Sebaliknya aliran data yang telah dikompresi harus dilakukan proses dekompresi untuk menghasilkan kembali alairan data yang asli. Karena proses dekompresi menghasilkan decoding dari aliran data yang sudah dikompresi maka hasilnya adalah aliran data decoded.

Tingkat pengurangan data yang dicapai sebagai hasil dari proses kompresi disebut rasio kompresi. Rasio ini merupakan perbandingan antara panjang data string asli dengan panjang data string yang sudah dikompresi, seperti dituliskan dalam persamaan 2.1

$$Rasio = \frac{\text{Ukuran file asli}}{\text{Ukuran file terkompresi}} \quad (2.1)$$

Jika dinyatakan dalam presentase maka dituliskan dalam persamaan berikut:

$$P = \left(1 - \frac{\text{Ukuran file terkompresi}}{\text{Ukuran file asli}}\right) * 100\%$$

Yang berarti ukuran *file* berkurang sebesar *P* (dalam persentase) dari ukuran semula. Semakin tinggi rasio tingkat suatu teknik kompresi data maka semakin efektif teknik kompresi tersebut. Pada saat dikompresi, rasio kompresi akan berselang-seling berdasarkan pengaruh data terhadap algortima yang digunakan. Dengan demikian maka yang perlu diperhatikan adalah rasio kompresi rata-rata. Bukan pada rasio yang dicapai pada suatu waktu tertentu. Pada umumnya algoritma yang baik dapat mencapai presentase rasio kompresi rata-rata 1,5 atau jika dalam presentase sebesar 33%, bahkan saat ini sudah banyak algoritma yang menghasilkan rasio kompresi rata-rata lebih besar dari dua [3].

3. METODOLOGI DAN PERANCANGAN

3.1 Deskripsi Umum Perangkat Lunak

Perangkat lunak yang akan dibuat adalah untuk seorang *user* yang ingin mengkompresi sebuah *file* teks, dimana *user* bisa mendapatkan hasil kompresi yang semaksimal mungkin guna menghemat tempat penyimpanan pada suatu media.

Maka perangkat lunak yang dibuat berupa sistem yang dapat mengkompres atau memampatkan suatu *file*. Metode yang digunakan dalam perangkat lunak ini adalah kombinasi metode kompresi *LZW(Limpel-Ziv-Welch)* dan *Huffman*.

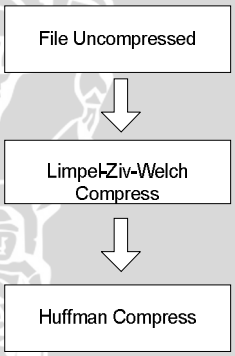
Ketika *user* memasuki perangkat lunak maka proses yang terjadi adalah :

- *User* memasukkan file yang dibuka melalui *Open Dialog File*.
- Kemudian oleh perangkat lunak, file tersebut akan diproses untuk di kompresi.

3.2 Perancangan Perangkat Lunak

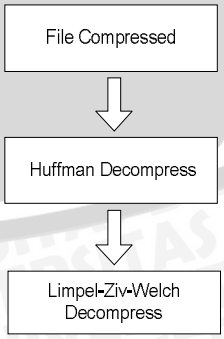
Berdasarkan analisa yang telah dilakukan, berikut ini akan dibahas mengenai arsitektur dan proses yang terjadi pada perangkat lunak yang akan dibangun.

Gambar 3.1 menjelaskan tentang proses kompresi secara keseluruhan dari kompresi *LZW* dan *Huffman*



Gambar 3.1 Proses Kompresi

Sedangkan Gambar 3.2 menjelaskan tentang proses dekompresi secara keseluruhan

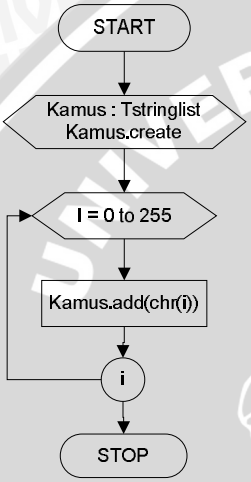


Gambar 3.2 Proses Dekompresi

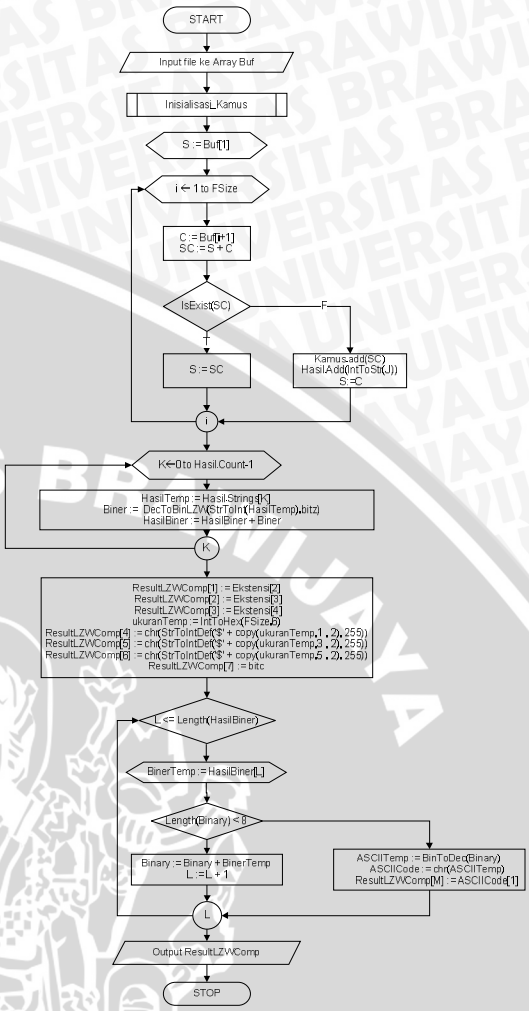
3.3 Perancangan Kompresi

3.3.1 Kompresi LZW

Algoritma kompresi LZW akan menggunakan kamus dimana rangkaian string akan digantikan dengan indeks yang diperoleh dari sebuah kamus. Hal ini berarti meng-enkode suatu string maka data yang akan disimpan adalah angka yang merepresentasikan indeks string tersebut pada kamus. Gambar 3.3 menunjukkan *flowchart* inisialisasi_kamus 0-255 dan Gambar 3.4 menunjukkan *flowchart* algoritma kompresi *LZW* (*Limpel-Ziv-Welch*).



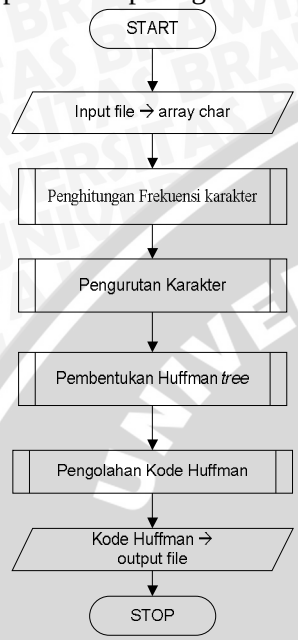
Gambar 3.3 *Flowchart* Proses Inisialisasi_Kamus



Gambar 3.4 *Flowchart* kompresi *LZW*(*Limpel-Ziv-Welch*)

3.3.2 Kompresi Huffman

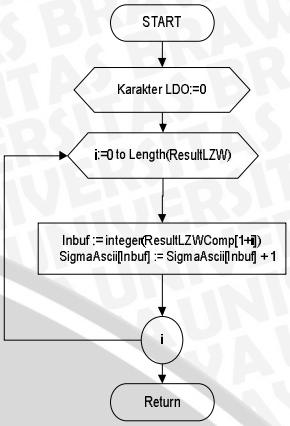
Setelah proses Kompresi dari Algoritma LZW (*Limpel-Ziv-Welch)*, Hasil output string kemudian di *encoding* dengan Algoritma Huffman. Berikut adalah *Flowchart* yang menggambarkan proses Algoritma kompresi *Huffman* dapat dilihat pada gambar 3.5



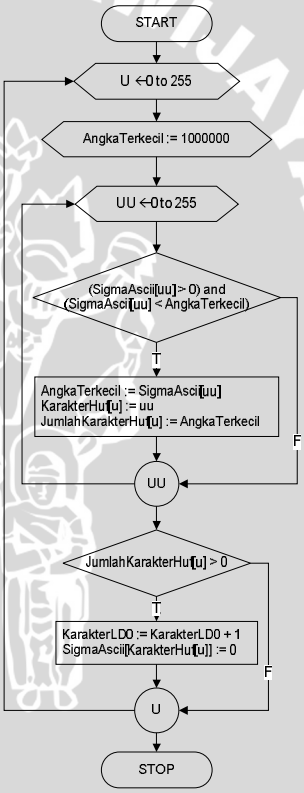
Gambar 3.5 *Flowchart* kompresi Huffman

Algoritma Kompresi Huffman ini dimulai dengan proses penginputan *file* yang merupakan hasil kompresi dari Algoritma *LZW* (*Limpel-Ziv-Welch)*.

Kemudian dilakukan penghitungan jumlah karakter yang muncul dan dikelompokkan sesuai dengan nilai ASCIInya. Setelah didapat jumlah karakter yang dikelompokkan sesuai dengan nilai ASCIInya, kemudian diurutkan dari yang terkecil sampai yang terbesar. Hal ini dimaksudkan untuk mempermudah pembuatan pohon Huffman. *Flowchart* Penghitungan frekuensi karakter dapat dilihat pada gambar 3.6. Sedangkan *flowchart* Pengurutan Karakter dapat dilihat pada gambar 3.7

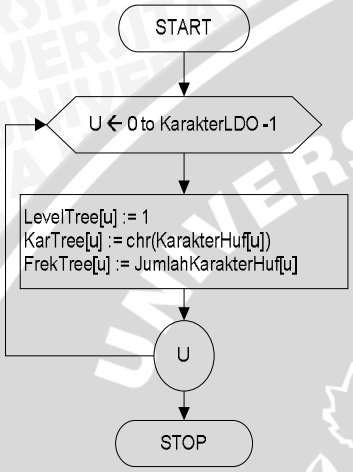


Gambar 3.6 *Flowchart* Penghitungan Frekuensi karakter



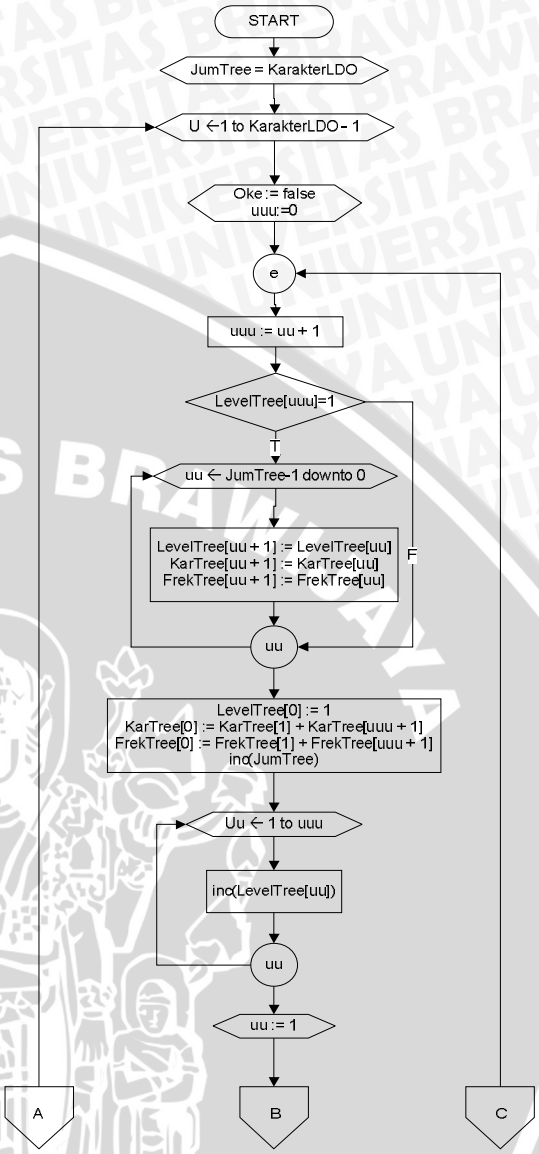
Gambar 3.7 *Flowchart* Pengurutan Karakter

Setelah dilakukan pengurutan karakter ASCII sesuai dengan nilai jumlahnya dari yang terkecil sampai terbesar maka mulai dilakukan inisialisasi *node* Huffman *Tree*. Semua karakter tersebut masing-masing dijadikan sebuah *node*, diberi level awal yang bernilai 1 dan masing-masing *node* akan menyimpan informasi karakter ASCII dan frekuensinya. *Flowchart* inisialisasi *node Huffman tree* dapat dilihat pada gambar 3.8

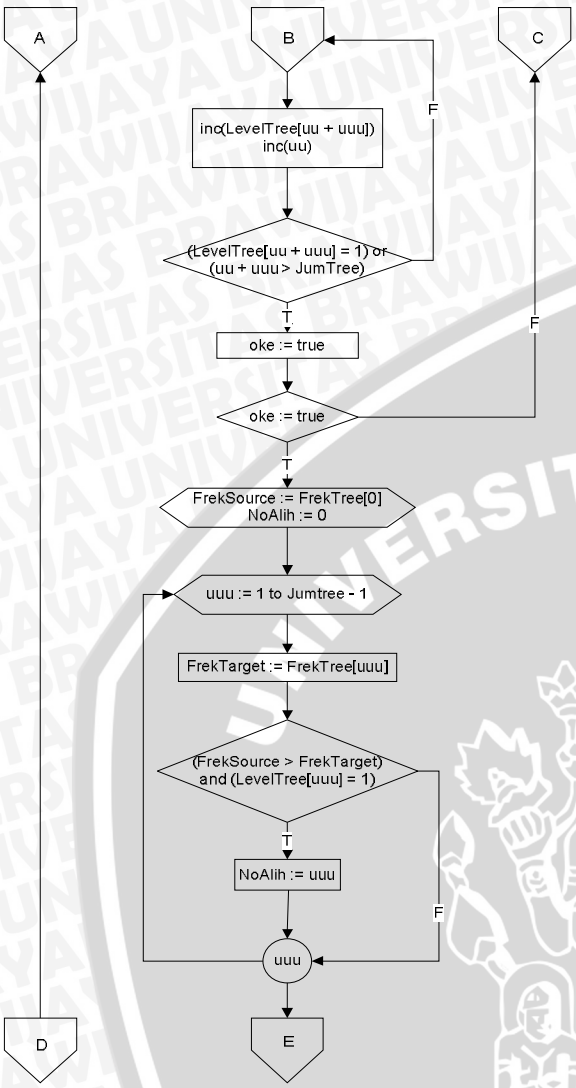


Gambar 3.8 *Flowchart* inisialisasi *node Huffman tree*

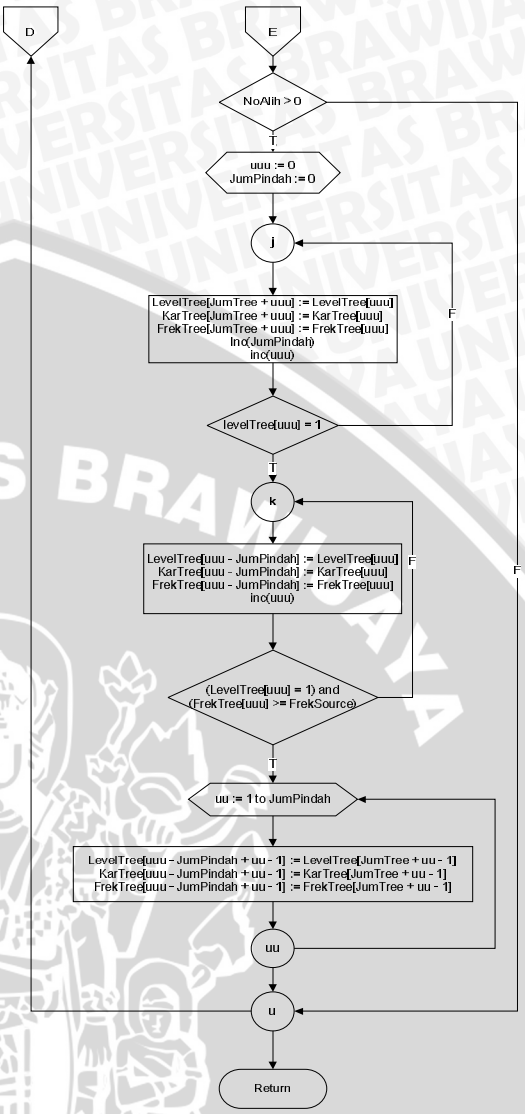
Huffman tree sudah siap dibentuk setelah proses penginisialisasi *node* Huffmannya selesai. Dalam proses pembentukan *Huffman tree*, terjadi dua proses utama yaitu : Pembentukan *node* baru dan pengurutan *node* yang lama dan baru sesuai dengan nilai frekuensinya. Proses pembentukan *node* baru, dimulai dengan dua *node* yang mempunyai frekuensi terkecil akan membentuk sebuah *node* baru dan *node* yang baru tersebut akan berisi karakter dari *node-node yang* membentuknya. Sedangkan frekuensi *node* yang baru bernilai jumlah dari frekuensi *node-node* yang membentuknya. Setelah dilakukan pembentukan *node* baru maka akan dilakukan pengurutan node sesuai dengan nilai frekuensi dari yang terkecil sampai yang terbesar. Hal itu dilakukan terus sampai terbentuk sebuah *node* saja. *Node* tersebut diistilahkan *root.Flowchart* Pembentukan *Huffman tree* dapat dilihat pada gambar 3.9.



Gambar 3.9 *Flowchart* Pembentukan *Huffman Tree*



Gambar 3.9 *Flowchart* Pembentukan *Huffman Tree* (Sambungan)



Gambar 3.9 *Flowchart* Pembentukan *Huffman Tree* (Sambungan)

Pembentukan Kode Huffman pada dasarnya merupakan kode prefiks (prefix code). Kode prefiks adalah himpunan yang berisi sekumpulan kode biner, dimana pada kode prefik ini tidak ada kode biner yang menjadi awal bagi kode biner yang lain. Kode prefiks biasanya direpresentasikan sebagai pohon biner yang diberikan nilai atau label. Untuk cabang kiri pada pohon biner diberi label 0, sedangkan pada cabang kanan pada pohon biner diberi label 1. Rangkaian bit yang terbentuk pada setiap lintasan dari akar ke daun merupakan kode prefiks untuk karakter yang berpadanan. Pohon biner ini biasa disebut pohon Huffman.

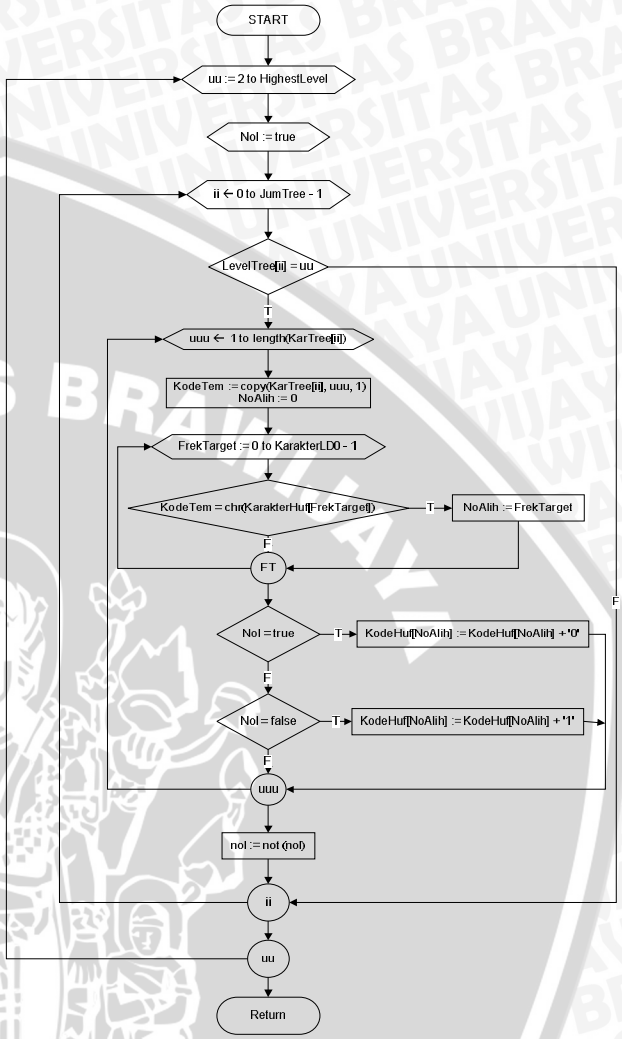
Langkah-langkah pembentukan Pohon *Huffman* :

1. Memilih 2 simpul dengan nilai frekuensi yang terkecil dalam simpul
2. Menggabungkan ke 2 nilai frekuensi tersebut dan nilai frekuensi dari gabungan tersebut merupakan jumlah dari nilai frekuensi dari 2 simpul
3. Memilih simpul lain untuk digabung seperti sebelumnya, simpul yang dipilih dapat berupa gabungan simpul-simpul sebelumnya ataupun simpul dari baru.
4. Ulangi langkah-langkah sebelumnya sampai hanya terdapat 1 simpul yang mewakili string yang dikodekan

Encoding merupakan proses penyusunan kode biner dari teks string yang ada. Proses encoding merupakan kelanjutan dari proses sebelumnya, yaitu pembuatan pohon Huffman. Langkah-langkah sebagai berikut

a. Menentukan karakter dalam string teks yang akan di-encoding terlebih dahulu.
b. Dimulai dari akar pohon Huffman, baca setiap satuan nilai biner yang terdapat dalam cabang ke bawah hingga simpul karakter yang diinginkan ditemukan.
c. Mengulangi langkah-langkah sebelumnya hingga semua karakter dalam teks string selesai diencoding.

Proses Pengolahan Kode *Huffman* dapat dilihat pada Gambar 3.10



Gambar 3.10 Pengolahan Kode H*uffman*

3.4 Pembentukan *Header File*

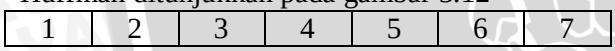
Header file merupakan bagian dari data yang akan disimpan ke dalam file terkompresi. *Header file* berisi bagian-bagian yang berfungsi sebagai pedoman atau kamus dalam proses dekompresi nantinya. Adapun *header file* hasil dari kompresi *LZW* ditunjukkan pada gambar 3.11 berikut:

Gambar 3.11 *Header File LZW*

Adapun keterangan untuk bagian-bagian pada gambar 3.4 adalah:

1. bagian satu berisi ekstensi file yang akan dikompresi atau file asli.
2. bagian dua berisi kode heksa yang berfungsi untuk menyimpan ukuran file asli yang akan dikompresi
3. bagian 3 berisi data hasil kompresi akhir dari kalimat ke dalam karakter ASCII

Sedangkan *header file* hasil dari kompresi Huffman ditunjukkan pada gambar 3.12



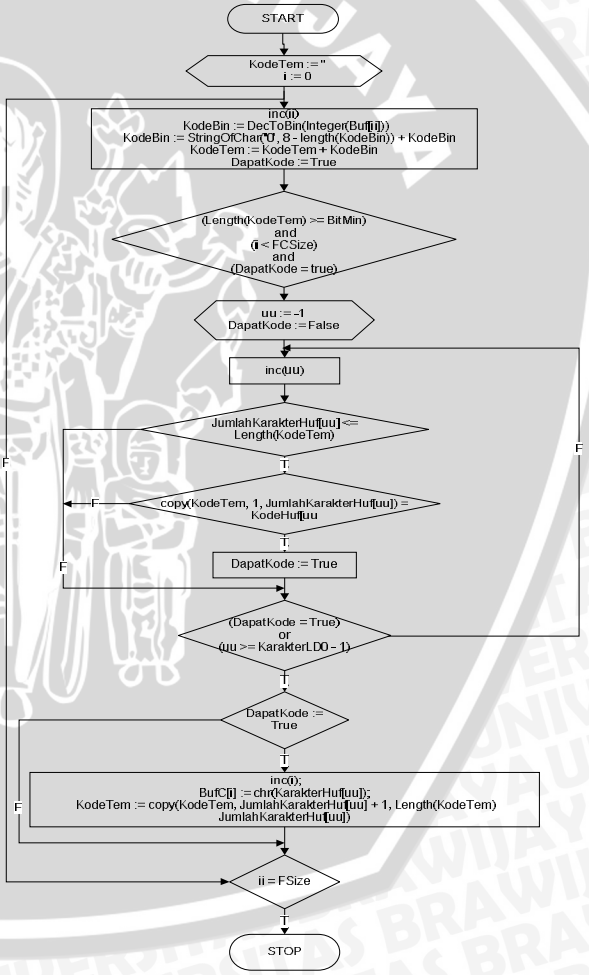
Gambar 3.12 *Header File Huffman*

1. Bagian satu berisi ekstensi file yang akan di kompresi yang merupakan hasil kompresi dari Algoritma *LZW* (*.lzw)
2. Bagian dua berisi kode heksa yang berfungsi untuk menyimpan ukuran file asli yang akan dikompresi
3. Bagian tiga berisi jumlah karakter Huffman
4. Bagian empat berisi karakter Huffman
5. Bagian lima berisi kode Biner pertama dari karakter Huffman
6. Bagian enam berisi kode Biner kedua dari karakter Huffman
7. Bagian tujuh berisi panjang dari kode Biner dari kode Huffman

3.5 Perancangan Dekompresi

3.5.1 Dekompresi Huffman

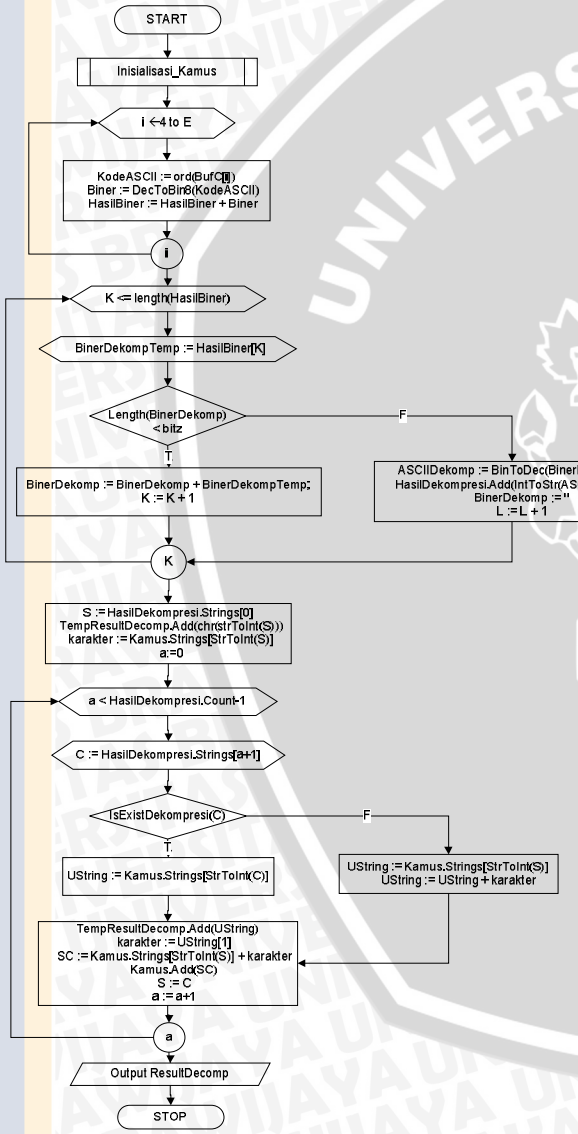
Karena tiap kode Huffman yang dihasilkan unik, maka proses dekompresi dapat dilakukan dengan mudah. Tiap *bit* yang dibaca dicocokan dengan daftar kode Huffman yang telah dibuat pada proses kompresi sebelumnya. Jika *bit* yang dibaca tidak terdapat pada daftar kode Huffman, maka *bit* tersebut digabungkan dengan *bit* selanjutnya, kemudian dicocokan kembali pada daftar kode. Jika kumpulan *bit* tersebut ada dalam daftar kode Huffman maka kumpulan *bit* tersebut dipetakan dengan karakter ASCII yang sesuai dengan kumpulan *bit* tersebut. *Flowchart* Algoritma dekompresi Huffman dapat dilihat pada Gambar 3.13



Gambar 3.13 *Flowchart* Dekompresi *Huffman*

3.5.2 Dekompresi LZW

Proses dekompresi pada LZW dilakukan dengan prinsip yang sama seperti proses kompresi. Pada awalnya, dictionary diinisialisasi dengan semua karakter dasar yang ada. Lalu pada setiap langkah, kode dibaca satu per satu dari stream kode, dikeluarkan string dari dictionary yang berkorespondensi dengan kode tersebut, dan ditambahkan string baru ke dalam dictionary. *Flowchart* Algoritma dekompresi *LZW (Limpel-Ziv-Welch)* dapat dilihat pada Gambar 3.14

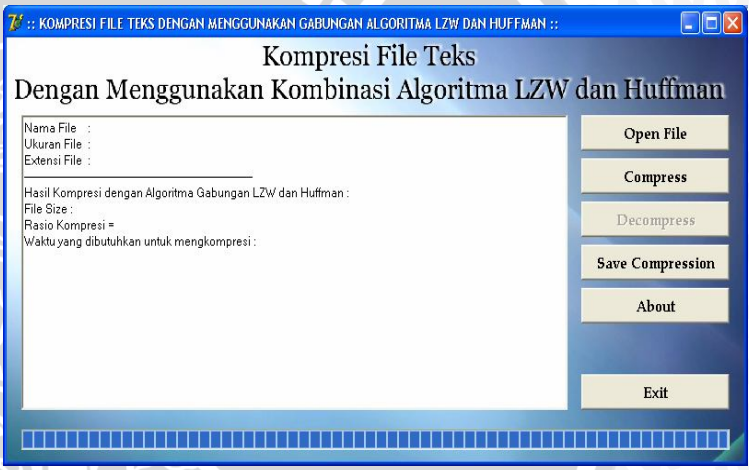


Gambar 3.14 *Flowchart* Dekompresi *LZW*

4. IMPLEMENTASI

Cakupan pembahasan pada bab ini meliputi implementasi sistem dan pembahasannya serta membahas uji coba dan evaluasi hasil. Implementasi sistem merupakan implementasi rancangan yang sudah dilakukan pada bab sebelumnya yaitu Bab 3. Sedangkan pembahasan adalah penjelasan dari masing-masing implementasi tersebut.

Uji coba dilakukan terhadap sistem pengkompresian sebuah file teks dengan menerapkan kombinasi Algoritma *LZW (Limpel-Ziv-Welch)* dan *Huffman*. Hasil dari uji coba ini akan digunakan untuk mengevaluasi tingkat keefektifan yang dilakukan oleh sistem yang telah dibuat.



Gambar 4.1 Rancangan antarmuka aplikasi Kompresi *File* Teks Menggunakan Kombinasi algoritma *LZW (Limpel-Ziv-Welch)* dan *Huffman*

4.1 Implementasi Uji Coba

4.1.1 Rancangan Evaluasi

Pengujian dilakukan terhadap 20 *file* yaitu masing-masing 5 *file* dengan ekstensi **.txt*, **.html*, **.pas* , dan **.doc*. Uji coba dilakukan terhadap berbagai ukuran file. Data *input* yang digunakan dalam uji coba ditunjukkan pada tabel 4.1 berikut.

Tabel 4.1 Data *input* untuk uji coba

Nama file	Ekstensi	Ukuran file (bytes)
file 1	*.*txt*	4.172
file 2	*.*txt*	6.571
file 3	*.*txt*	8.444
file 4	*.*txt*	19.937
file 5	*.*txt*	32.035
utama	*.*html*	16.937
Radio Nafiri FM 107.1	*.*html*	30.050
BCA	*.*html*	35.808
Indosat Integrated home	*.*html*	48.619
Home	*.*html*	80.370
Quick Sort	*.*pas*	1.400
Frekuensi Data	*.*pas*	1.774
ULZW	*.*pas*	11.880
Bigram	*.*pas*	26.372
Pemampatan	*.*pas*	26.918
Kata Pengantar	*.*doc*	25.088
Lembar pengesahan	*.*doc*	26.112
Pencuri-Pencuri Impian	*.*doc*	32.256
bca	*.*doc*	33.665
CV2009	*.*doc*	43.250

4.1.2 Analisa dan Evaluasi Hasil

Hasil dari uji coba terhadap sistem untuk proses kompresi file teks yang berekstensi *.*txt* ditunjukkan pada tabel 4.2. Uji coba proses kompresi untuk sebuah file teks telah menghasilkan sebuah file terkompresi dengan ekstensi *.*lzh*. Dari tabel 4.2 terlihat bahwa sistem dapat menghasilkan sebuah file dengan ukuran yang lebih kecil dibandingkan dengan ukuran file aslinya. Rasio kompresi rata-rata yang dicapai oleh sistem untuk file dengan ekstensi *.*txt* adalah sebesar 1,70 dengan persentase rata-rata sebesar 43,41%.

Tabel 4.2 Hasil Kompresi file ***.txt***

Nama file	Ukuran file asli (bytes)	Ukuran file terkompresi (bytes)	Waktu proses (s)	Rasio kompresi
file 1	4.172	3.348	0,8	1,246117
file 2	6.571	4.542	1,3	1,44672
file 3	8.444	3.845	1,19	2,196099
file 4	19.937	11.421	5,81	1,745644
file 5	32.035	17.112	16,24	1,872078

Hasil dari uji coba terhadap sistem untuk proses kompresi file teks yang berekstensi *.*html* ditunjukkan pada tabel 4.3. Rasio rata-rata yang dicapai oleh sistem untuk file berekstenai *.*html* adalah sebesar 2,28. dengan persentase rata-rata sebesar 58,08%. Tingkat rasio kompresi untuk sebuah file sangat dipengaruhi oleh komposisi data yang bersangkutan.

Tabel 4.3 Tabel hasil kompresi file ***.html***

Nama file	Ukuran file asli (bytes)	Ukuran file terkompresi (bytes)	Waktu proses (s)	Rasio kompresi
utama	16.937	9.862	5,41	1,7174
Radio Nafiri FM 107.1	30.050	15.732	14,18	1,91012
BCA	35.808	13.240	14,20	2,704532
Indosat Integrated home	48.619	22.259	33,93	2,18424
Home	80.370	27.682	67.94	2,903331

Hasil dari uji coba terhadap sistem untuk proses kompresi file yang berekstensi *.*pas* ditunjukkan pada tabel 4.4. Rasio kompresi rata-rata untuk file berekstensi *.*pas* yang dicapai oleh sistem adalah sebesar 1,61 dengan persentase rata-rata sebesar 50,59%. Proses kompresi terhadap sebuah data yang berukuran sangat kecil akan menghasilkan sebuah file dengan ukuran yang lebih besar dari aslinya. Hal ini terjadi karena pada saat konversi biner 12 *bit* kemudian dikelompokkan per 8 *bit* untuk mencari karakter ASCIInya pada saat kompresi LZW.

Tabel 4.4 Hasil Kompresi file ***.pas***

Nama file	Ukuran file asli (bytes)	Ukuran file terkompresi (bytes)	Waktu proses (s)	Rasio kompresi
Quick Sort	1.400	1.740	0.52	0,804598
Frekuensi Data	1.774	1.914	0,58	0,926855
ULZW	11.880	6.276	2.86	1,892925
Bigram	26.372	11.587	7.61	2,275999
Pemampatan	26.918	12.250	8.95	2,197388

Hasil uji coba sistem untuk proses kompresi file yang berekstensi **.doc* ditunjukkan pada tabel 4.2. Dari tabel 4.5 terlihat bahwa sistem menghasilkan sebuah file dengan ukuran yang lebih kecil dibandingkan dengan ukuran file aslinya. Rasio kompresi rata-rata untuk file dengan ekstensi **.doc* adalah sebesar 2,60 dengan persentase rata-rata sebesar 61,05%.

Tabel 4.5 Tabel hasil kompresi file ***.doc***

Nama file	Ukuran file asli (bytes)	Ukuran file terkompresi (bytes)	Waktu proses (ms)	Rasio kompresi
kata pengantar	25.088	11.001	7.23	2,28052
lembar pengesahan	26.112	8.538	6.04	3,058327
Pencuri-Pencuri Impian	32.256	12.089	10.40	2,668211
Bca	33.665	12.896	12.46	2,610499
CV2009	43.520	18.039	20.63	2,412551

Hasil uji coba untuk proses dekompresi ditunjukkan pada tabel 4.6, tabel 4.7, tabel 4.8 dan tabel 4.9. Proses dekompresi dari sistem terbukti dapat mengembalikan sebuah file **.lzh* menjadi seperti *file* aslinya. Waktu proses yang dibutuhkan oleh sistem sebenarnya sangatlah dipengaruhi oleh komposisi data yang bersangkutan serta perangkat keras yang digunakan oleh *user*.

Tabel 4.6 Hasil Dekompresi file ***.txt***

Nama file	Ukuran file terkompresi (bytes)	Ukuran file hasil dekompresi	Waktu proses (s)
file 1	3.348	4.172	0.38
file 2	4.542	6.571	0.51
file 3	3.845	8.444	0.36
file 4	11.421	19.937	1.16
file 5	17.112	32.035	2.03

Tabel 4.7 Hasil Dekompresi file ***.html***

Nama file	Ukuran file terkompresi (bytes)	Ukuran file hasil dekompresi	Waktu proses (s)
utama	9.862	16.937	1.23
Radio Nafiri FM 107.1	15.732	30.050	1.79
BCA	13.240	35.808	1.45
Indosat Integrated home	22.259	48.619	2.91
Home	27.682	80.370	4.12

Tabel 4.8 Tabel hasil dekompresi file ***.pas***

Nama file	Ukuran file terkompresi (bytes)	Ukuran file hasil dekompresi	Waktu proses (ms)
Quick Sort	1.740	1.400	0.16
Frekuensi Data	1.914	1.774	0.20
ULZW	6.276	11.880	0.65
Bigram	11.587	26.372	1.23
Pemampatan	12.250	26.918	1.30

Tabel 4.9 Tabel hasil dekompresi file ***.doc***

Nama file	Ukuran file terkompresi (bytes)	Ukuran file hasil dekompresi	Waktu proses (ms)
kata pengantar	11.001	25.088	1.05
lembar pengesahan	8.538	26.112	0.85
Pencuri-Pencuri Impian	12.089	32.256	1.32
Bca	12.896	33.665	1.38
CV2009	18.039	43.520	1.86

Ada beberapa kasus dimana hasil kompresi gabungan dari LZW dan Huffman lebih besar jika dibandingkan dengan hasil kompresi LZW sendiri atau Huffman sendiri. Hal ini terjadi jika terdapat karakter yang sama, berurutan dan dalam jumlah yang banyak . Sehingga pada saat hasil kompresi LZW yang merupakan inputan untuk kompresi Huffman, hasil kompresi dari Huffman tidak begitu memberikan hasil yang signifikan jika dibandingkan dengan di kompresi dengan algoritma masing-masing.

5. KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari implementasi dan pembahasan yang telah dilakukan pada pengerjaan Tugas Akhir ini, maka dapat diambil kesimpulan sebagai berikut :

1. Semakin besar tingkat rasio, maka ukuran file kompresi yang dihasilkan akan semakin kecil
2. Sistem Pengkompresian sebuah file dengan menerapkan kombinasi algoritma *LZW* (*Limpel-Ziv-Welch)* dan *Huffman* ini menghasilkan kompresi rata-rata sebesar 1,701331 (43,41%) untuk file **.txt*; 2,28 (58,08%) untuk file **.html*; 1,61 (50,59%) untuk file **.pas*; dan untuk file **.doc* sebesar 2,6 (61,05%)
3. Proses kompresi dari sebuah file sangat dipengaruhi oleh komposisi data yang di-*inputkan*. Hasil *output* dari proses kompresi tersimpan dengan nama dan direktori sesuai yang telah ditentukan oleh *user* dengan berekstensi **.lzh*. Proses dekompresi dari sistem yang dibuat telah dapat mengembalikan sebuah file terkompresi menjadi seperti file aslinya.
4. Ada beberapa kasus dimana hasil kompresi gabungan dari LZW dan Huffman lebih besar jika dibandingkan dengan hasil kompresi LZW atau Huffman. Hal ini terjadi jika terdapat karakter yang sama, berurutan dan dalam jumlah yang banyak

5.2 Saran

Beberapa saran untuk pengembangan lebih lanjut yang dapat diberikan oleh penulis adalah:

1. Melakukan pengembangan penggunaan kombinasi algoritma *Lempel Ziv Welch (LZW)* dan *Huffman* untuk kompresi tipe *file* selain teks
2. Melakukan Penelitian untuk membandingkan kombinasi algoritma *LZW*(*Limpel-Ziv-Welch*) dan *Huffman* dengan algoritma yang lainnya

DAFTAR PUSTAKA

[1]Al-Laham Muhammed dan El Emary M.M. Ibrahiem. 2007. *Comparative Study Between Various of Data Compression Techniques*. Faculty of Engineering. Al Ahliyya Amman University. Amman. Jordan

[2]Anton. 2005. *Kompresi dan Teks*. Fakultas Teknik Informatika. Universitas Kristen Duta Wacana.Salatiga

[3]Bloom, C. 2004. *Compression : Algorithms : Statistical Codes*. http://www.cbloom.com/algs/statisti.html Tanggal akses: 10Maret 2009

[4]Callista,N. 2007. *Aplikasi Greedy pada Algoritma Huffman untuk Kompresi Teks*. Program Studi Teknik Informatika. Sekolah Teknik Elektro dan Informatika. ITB. Bandung

[5]Candra,A. 2006. *Analisa Perbandingan Kerja Algoritma Kompresi Huffman, LZW(Limpel-Ziv-Welch), dan DMC(Dynamic Markov Compression)*. Jurusan Teknik Informatika . Fakultas Teknologi Industri. Universitas Kristen Petra. Surabaya

[6]Linawati. 2004. *Perbandingan Kinerja Algoritma Kompresi Huffman, LZW, dan DMC pada berbagai Tipe File*. Jurusan Ilmu Komputer. Fakultas MIPA. Universitas Katholik Parahyangan Bandung

[7]MDGR. 2008. *Pengantar Sistem Operasi Komputer*. *http://bebas.vlsm.org/v06/Kuliah/SistemOperasi/BUKU/SistemOperasi-4.X-2/ch24s12.html* Tanggal akses: 5 Maret 2009

[8]Nelson, Mark. 1989. *LZW Data Compression*. http://marknelson.us/1989/10/01/lzw-data-compression Tanggal akses : 8 Juni 2009

[9]Noprianto. 2004. *Adu Kemampuan Software Kompresi*. http://ilkom.unud.ac.id/infolinux/Tahun%202004/PDF%20LINUX%200804/24_Adu%20Software_08.pdf Tanggal akses: 11 Maret 2009

[10]Smith, Steven W. 2007. *The Scientist and Engineer's Guide to Digital Signal Processing*. http://dspguide.com/ch27/5.htm Tanggal akses : 9 Juni 2009

[11]Wardoyo,I. 2005. *Kompresi Teks dengan menggunakan Algoritma Huffman*. Jurusan Teknik Informatika. Sekolah Tinggi Teknologi Telkom. Bandung