Implementasi Algoritma Kompresi Data Huffman Untuk Memperkecil Ukuran *File* MP3 *Player*

Victor Amrizal

Staf Pengajar Fakultas Sains dan Teknologi
Universitas Islam Negeri Syarif Hidayatullah Jakarta
Tel : (021) 7493547 Fax : (021) 7493315
e-mail : ersebros@yahoo.com

ABSTRACT

Process to minimize file is by undertaking compression to that file. Text compression process aims to reduce symbol purpose repeat or character that arrange text by mengkodekan symbols or that character so room the need storage can be reduced and data Transfer time can faster. Text compression process can be done by coding segments of original text are next to be placed deep lexical. Process compression can be done by various algorithm media, amongst those Coding's Huffman that constitutes one of tech compression which involve frequency distribution a symbol to form unique code. Symbol frequency distribution will regard long its Huffman code, progressively frequent that symbol texts deep appearance therefore Huffman code length that resulting will get short. This method mengkodekan symbols or characters with binary treed help by merges two character emergence frequencies smallest until molded treed codes.

Keywords: *Huffman Coding, Data Compression, algorithm, and* MP3*.*

1. PENDAHULUAN

Teknologi komunikasi data melahirkan jaringan komputer terbesar di dunia yang disebut internet. Internet memiliki banyak fasilitas yang sangat berguna bagi pemakainya. Melalui internet proses pengumpulan informasi maupun penyebaran informasi menjadi lebih mudah dan cepat. Banyak organisasi yang menggunakan internet untuk mendistribusikan data dan informasi untuk digunakan dan dimanfaatkan oleh masyarakat luas.

Sebagian pengguna internet masih ada yang menggunakan jalur telepon sebagai medianya. Kecepatan maksimum jalur telepon analog untuk transmisi data digital adalah 56 Kbps. Dengan kecepatan seperti pengiriman data berukuran besar membutuhkan banyak waktu yang juga berarti membuat bengkak biaya. Makin besar ukuran data yang dikirim, makin lama waktu yang dibutuhkan, dan makin besar pula biaya yang harus dikeluarkan.

Salah satu cara untuk mempercepat transmisi data adalah dengan memperkecil ukuran data tersebut. Untuk keperluan tersebut, maka diperlukan suatu perangkat lunak yang dapat mengecilkan ukuran *file* tanpa mengubah isi aslinya.

Sekarang ini banyak perangkat lunak (aplikasi) untuk kompresi data yang sangat terkenal antara lain: *zip, rar, arj*, dan lain sebagainya. Aplikasi kompresi data tersebut memang efektif untuk melakukan kompresi terhadap beberapa jenis file, tetapi dengan aplikasi tersebut sangat beresiko terhadap keamanan data, karena format .zip dan .rar sudah sangat dikenal oleh orang banyak, apalagi jika data tersebut sifatnya sangat rahasia. Dan diawal penggunaannya pun, pemakai diharuskan untuk *registrasi* yaitu memasukkan nomor seri legal yang diperoleh dengan membeli produk aplikasi tersebut dengan biaya yang tidak murah agar dapat memanfaatkan semua fasilitas yang ada dan digunakan tanpa batas waktu tertentu. Walaupun tidak sedikit pula beberapa alamat disitus internet menyediakan nomor seri legalitas yang tidak resmi yang biasa disebut dengan membajak atau *crack*, tetapi hal tersebut merupakan salah satu bentuk pelanggaran undang-undang hak cipta yang belakangan ini sedang digencarkan oleh beberapa perusahaan teknologi informasi dunia. Dalam

penelitian ini, peneliti mengimplementasikan sebuah aplikasi kecil yang dapat digunakan untuk mengkompresi ukuran beberapa jenis *file* dan dapat dimanfaatkan oleh siapapun tanpa harus takut dicap sebagi pembajak perangkat lunak.

2. PEMBAHASAN

Analisis Permasalahan

Media penyimpanan elektronik seperti *harddisk*, *cd-rom*,disket atau media penyimpanan lainnya merupakan salah satu faktor pendukung sebuah komputer, yang berfungsi sebagai tempat menyimpan berbagai macam *file* atau berkas. *File* ini dapat bermacam-macam bentuk dan fungsi, mulai dari *file* yang dibutuhkan untuk proses sistem operasi sampai dengan *file* yang berupa dokumen-dokumen pribadi. Informasi, properti serta ukuran yang dimiliki oleh masing-masing *file* pun beragam. Semakin besar ukuran *file* dan semakin besar banyak *file* yang disimpan dalam media penyimpanan tersebut maka semakin besar pula kemungkinan berkurangnya dan menipisnya sisa kapasitas ruang penyimpanan di media tersebut.

Permasalahan lainnya yang sering dihadapi oleh para pengguna komputer adalah pada saat mereka membutuhkan atau pun bermaksud mengirim *file* dari dan ke suatu alamat di internet, misal pada surat elektronik (*e-mail*), mereka akan membutuhkan waktu yang lama untuk melakukan proses pengambilan dan pengiriman tersebut apabila ukuran yang dimiliki oleh *file*-nya besar.

Salah satu cara untuk mengatasi masalah tersebut adalah dengan memperkecil ukuran *file* yang diperlukan dengan menggunakan utiliti atau aplikasi kompresi *file*, khususnya *file* teks yang mempunyai redudansi yang relatif tinggi dibandingkan dengan format *file* lain.

Konsep Kompresi Data

Pada proses kompresi, pada umumnya, pertama kali yang dilakukan adalah membaca informasi yang dibutuhkan dari sebuah *file* sebagai proses inisialisasi. Informasi yang dimiliki oleh *file* dapat berupa tipe *file*, lokasi *file* dan yang utama adalah ukuran *file*.

Dari proses inisialisasi awal tersebut dapat didefinisikan informasi yang diperlukan dalam perancangan dan pengembangan aplikasi kompresi data yang akan dibuat, di antaranya, penamaan *file*, lokasi direktori asal *file*, ukuran besarnya *file*, lokasi direktori dan nama *file* hasil kompresi *file* tersebut disimpan. Proses selanjutnya adalah menelusuri dan membaca seluruh karakter dan frekuensi kemunculannya dalam *file*.

Metode Kompresi Data Huffman

Kompresi ialah proses pengubahan sekumpulan data menjadi suatu bentuk kode untuk menghemat tempat penyimpanan dan waktu untuk transmisi data. Metode Huffman yang diterapkan dalam penulisan ini adalah tipe statik, di mana dilakukan dua kali pembacaan (*two-pass*) terhadap *file* yang akan dikompresi, pertama untuk menghitung frekuensi kemunculan karakter dalam pembentukan pohon Huffman dan kedua untuk mengkodekan simbol dalam kode Huffman.

Metode ini mengkodekan simbol-simbol atau karakter dengan bantuan *binary tree* dengan cara menggabungkan dua buah frekuensi kemunculan karakter paling kecil hingga terbentuk pohon kode.

Proses Kompresi Data dengan Algoritma Huffman

Dasar pemikiran algoritma Huffman ini adalah bahwa setiap karakter ASCII biasanya diwakili oleh 8 bits. Jadi misalnya suatu *file* berisi deretan karakter "ABACAD" maka ukuran *file* tersebut adalah 6 x 8 bits =48 bit atau 6 *bytes*. Jika setiap karakter tersebut diberi kode lain misalnya A=1, B=00, C=010, dan D=011, berarti kita hanya perlu *file* dengan ukuran 11 bits (10010101011), yang perlu diperhatikan ialah bahwa kode-kode tersebut harus unik atau dengan kata lain suatu kode tidak dapat dibentuk dari kode-kode yang lain. Pada contoh sebelumnya, jika kode D kita ganti dengan 001, maka kode tersebut dapat dibentuk dari kode B ditambah dengan kode A yaitu 00 dan 1, tapi kode 011 tidak dapat dibentuk dari kode-kode yang lain. Selain itu karakter yang paling sering muncul, kodenya lebih kecil jumlah bitnya dibandingkan dengan karakter yang jarang muncul. Pada contoh sebelumnya, karakter A lebih sering muncul (3 kali), jadi kodenya dibuat lebih kecil jumlah bitnya dibanding karakter lain.

Untuk menentukan kode-kode dengan kriteria kode harus unik dan karakter yang sering muncul dibuat kecil jumlah bitnya, kita dapat menggunakan

algoritma Huffman untuk melakukan kompresi, dan membentuk *Huffman Tree* (pohon biner Huffman).

Sebagai contoh, sebuah *file* yang akan dimampatkan berisi karakter-karakter "PERKARA". Dalam kode ASCII masing-masing karakter dikodekan sebagai:

P = 50H = 01010000B
E = 45H = 01000101B
R = 52H = 01010010B
K = 4BH = 01001011B
A = 41H = 01000001B

Maka jika diubah dalam rangkaian bit, "PERKARA" menjadi:

01010000	01000101	01010010	01001011	01000001	01010010	01000001
P	E	R	K	A	R	A

yang berukuran 56 bit.

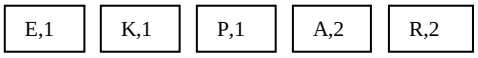
Proses kompresi dilakukan dengan langkah-langkah berikut:

a. Langkah pertama adalah menghitung frekuensi kemunculan masing-masing karakter dan mengurutkannya berdasarkan dari frekuensi terkecil kemunculan karakter. Jika dihitung pada contoh kata di atas, ternyata P muncul sebanyak 1 kali, E sebanyak 1 kali, R sebanyak 2 kali, K sebanyak 1 kali dan A sebanyak 2 kali.

E = 1
K = 1
P = 1
A = 2
R = 2

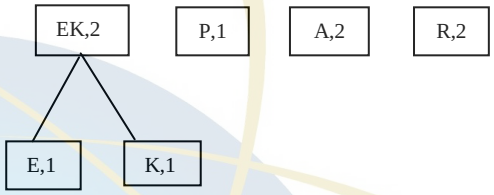
Untuk karakter yang memiliki frekuensi kemunculan sama seperti E, K dan P disusun menurut kode ASCII-nya, begitu pula A dan R.

b. Langkah kedua adalah membuat node masing-masing karakter beserta frekuensinya berdasarkan hasil dari pengurutan sebelumnya, maka didapat urutan *node* sebagai berikut:



Hasil Pengurutan Karakter

c. Langkah selanjutnya adalah dengan menggabungkan 2 *node* yang paling kiri (E dan K), lalu membuat *node* baru yang merupakan gabungan dua *node* tersebut, node gabungan ini akan memiliki cabang yang berisi 2 *node* yang digabungkan tersebut. Frekuensi dari *node* gabungan ini adalah jumlah frekuensi cabang-cabangnya.

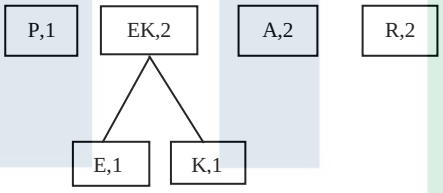


Gambar 1. Hasil Pengurutan Karakter

Penggabungan Dua *Node* Paling Kiri

Jika dilihat frekuensi tiap node pada level paling kiri, EK=2, P=1, A=2, dan R=2.

d. *Node-node* tersebut diurutkan kembali dari frekuensi terkecil, jadi *node* EK digeser ke sebelah kanan *node* P dan jika menggeser suatu *node* yang memiliki cabang, maka seluruh cabangnya diikutkan juga. Setelah diurutkan hasilnya akan menjadi sebagai berikut:

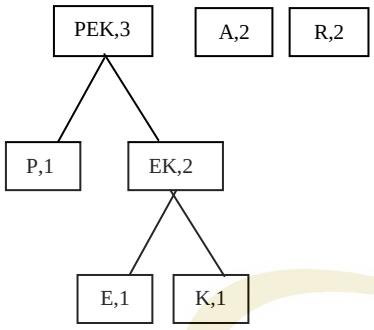


Gambar 2. Penggabungan 2 *Node*

Hasil Pengurutan Kembali

Setelah *node* paling kiri pada level diurutkan (**level berikutnya tidak perlu diurutkan**).

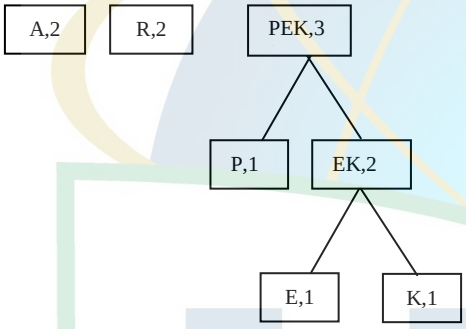
e. Berikutnya gabungkan kembali 2 *node* paling kiri seperti yang pernah dikerjakan sebelumnya. *Node* P digabung dengan node EK menjadi *node* PEK dengan frekuensi 3 dan akan menjadi seperti berikut ini:



Gambar 3. Hasil Pengurutan Kembali

Penggabungan Karakter Selanjutnya

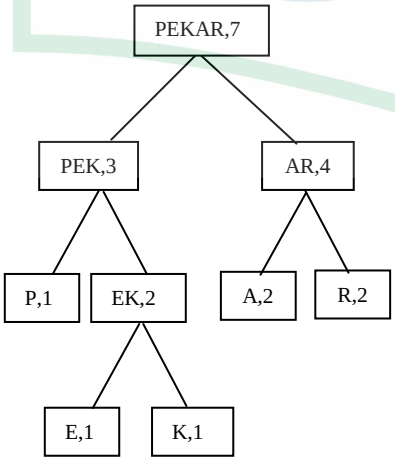
f. Kemudian diurutkan kembali menjadi:



Gambar 4. Penggabungan Karakter Berikutnya

Hasil Pengurutan Karakter Selanjutnya

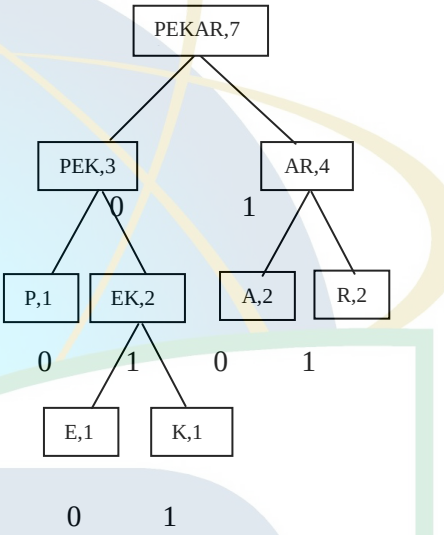
Demikian seterusnya sampai diperoleh pohon Huffman seperti berikut:



Gambar 5. Hasil Pengurutan Selanjutnya

Hasil Pembentukan Pohon Untuk Kata "PERKARA"

g. Setelah pohon Huffman terbentuk, kemudian diberikan tanda bit 0 untuk setiap cabang ke kiri dan bit 1 untuk setiap cabang ke kanan. Dari contoh sebelumnya maka didapat nilai bit untuk masing-masing cabang seperti Gambar 6.



Gambar 6. Hasil Pembentukan

Pemberian Bit 1 atau 0 pada Pohon

Setelah pohon Huffman terbentuk dan tiap *node*/cabangnya telah diberi angka bit penanda (0 atau 1), maka untuk mendapatkan kode Huffman masing-masing karakter, dapat ditelusuri karakter tersebut dari *node root* (yang paling atas: PEKAR) sampai ke *node* karakter tersebut dan kemudian disusun bit-bit yang dilaluinya.

Untuk mendapatkan kode Karakter E, dari *node* PEKAR kita harus menuju ke *node* PEK melalui bit 0 dan selanjutnya menuju ke *node* EK melalui bit 1, dilanjutkan ke *node* E melalui bit 0, jadi kode dari karakter E adalah 010.

Untuk mendapatkan kode Karakter K, dari *node* PEKAR kita harus menuju ke *node* PEK melalui bit 0 dan selanjutnya menuju ke *node* EK melalui bit 1,

dilanjutkan ke *node* K melalui bit 1, jadi kode dari karakter K adalah 011.

Untuk mendapatkan kode Karakter P, dari *node* PEKAR kita harus menuju ke *node* PEK melalui bit 0 dan selanjutnya menuju ke *node* P melalui bit 0, jadi kode dari karakter P adalah 00.

Untuk mendapatkan kode Karakter A, dari *node* PEKAR kita harus menuju ke *node* AR melalui bit 1 dan selanjutnya menuju ke node A melalui bit 0, jadi kode dari karakter A adalah 10.

Terakhir, untuk mendapatkan kode Karater R, dari *node* PEKAR kita harus menuju ke *node* AR melalui bit 1 dan selanjutnya menuju ke *node* R melalui bit 1, jadi kode dari karakter R adalah 11.

Hasil akhir kode Huffman dari contoh kata sebelumnya adalah:

```
E  = 010
K  = 011
P  = 00
A  = 10
R  = 11
```

Dengan kode ini, *file* yang berisi karakter-karakter "PERKARA" akan menjadi lebih kecil, yaitu:

00 010 11 011 10 11 10 = 16 bit
P E R K A R A

Dengan Algoritma Huffman berarti *file* ini dapat kita hemat sebanyak 56-16 = 40 bit.

Rancangan Prosedur dan Fungsi Program

Kegiatan pada tahap rancangan prosedur dan fungsi program ini meliputi pemrograman dan pengujian aplikasi.

Seperti yang telah dijelaskan bahwa tahapan dalam proses kompresi data algoritma Huffman adalah dengan membentuk pohon biner Huffman. Langkah awal untuk membentuk pohon biner ini adalah menyiapkan atau menyediakan suatu *array* (larik) dengan variabel inisial sebagai berikut:

```
//Algoritma inisialisasi awal (array)
1. inisialisasi uu, SigmaAscii,
   JumlahKarakterHuf,KarakterHuf
2. Selama uu <= 256
3.       SigmaAscii[uu] = 0
4.       JumlahKarakterHuf[uu] = 0
5.       KarakterHuf[uu] = 0
```

Pernyataan **SigmaAscii[uu]=0** dalam iterasi (*looping*) *uu* adalah inisialisasi kode ASCII untuk karakter yang nanti akan ditemukan pada file/berkas yang dikompresi dengan memberi inisial awal angka 0, begitu juga dengan pernyataan **JumlahKarakterHuf[uu]=0** adalah menyiapkan array untuk jumlah atau frekuensi karakter yang ditemui, dan pernyataan **KarakterHuf[uu]=0** adalah array yang disediakan untuk karakter yang ditemukan dalam proses. Seluruh *array* tersebut disediakan sebanyak 256 tempat, dengan asumsi bahwa karakter yang dibaca adalah karakter yang merupakan simbol kode karakter menurut ASCII (*American Standard Code for Information Interchange*).

Fungsi Pencarian dan Pencatat String

Proses Pencarian dan Pencatatan String adalah membaca seluruh karakter yang ada dalam file yang dikompres dan menyimpan jumlah tiap-tiap karakter pada posisi *array* yang sesuai dengan kode ASCII-nya. Apabila ada karakter sama ditemukan pada saat pembacaan maka jumlah karakter dalam *array* sebelumnya ditambah satu. Proses ini dinyatakan dalam prosedur iterasi berikut:

```
//Baca seluruh karaktaer yang ada dalam file
1. Selama I masih dalam UkuranFile
2.       Inbuf = nilai ASCII (karakter_i + 1)
3.       SigmaAscii[Inbuf] = SigmaAscii[Inbuf]+1
4. looping
```

- Proses iterasi ini adalah untuk pembacaan isi *file* sebanyak jumlah karakter yang ada dalam *file* tersebut.
- Karakter yang ditemukan diubah ke dalam bentuk integer nilai ASCII dengan pernyataan **Inbuf=nilai ASCII (karakter_i +1)** untuk kemudian disimpan dan dijumlahkan ke dalam array **SigmaAscii[]** pada posisi **Inbuf**.

1. Proses Pengurutan

Setelah seluruh karakter dalam *file* dibaca, diubah dan disimpan prosedur selanjutnya adalah pengurutan karakter-karakter berdasarkan dari frekuensi yang terkecil kemunculannya, apabila ada frekuensi karakter yang jumlahnya sama maka diurutkan menurut kode ASCII-nya. Fungsi ini dilakukan dengan algoritma sebagai berikut:

```
//Fungsi pengurutan karakter
1. Selama u <= 256          //Pengurutan
2.     AngkaTerkecil = 1000000
3.     Selama uu <= 256
4.         Jika (SigmaAscii[uu]>0) dan
(SigmaAscii[uu]<AngkaTerkecil) maka
5.             AngkaTerkecil=SigmaAscii[uu]
//catat jumlah karakter
6.             KarakterHuf[u]=uu
//catat kode ASCII
7.             JumlahKarakterHuf[u]=AngkaTerkecil
//catat jumlah karakter
```

Pada iterasi ini dilakukan 2 macam proses iterasi, yaitu:

- Iterasi pertama yaitu proses pencatatan karakter, iterasi u ini melakukan perulangan sebanyak 256 kali untuk menempatkan kode ASCII karakter yang ditemukan ke dalam posisi array **KarakterHuf[u]** dan mencatat frekuensi karakternya ke dalam array **JumlahKarakterHuf[u]** yang nantinya digunakan untuk perbandingan frekuensi dengan kode ASCII karakter lain.
- Iterasi kedua adalah perulangan uu sebanyak 256 kali untuk membaca seluruh isi array **SigmaAscii[]** (yang berisi kode ASCII karakter-karakter dalam file yang dibaca dari iterasi sebelumnya di atas), untuk kemudian dilakukan penempatan posisi karakter berdasarkan frekuensi terkecil.

Fungsi penempatan posisi karakter berdasarkan frekuensi yang terkecil adalah dilakukan dengan pengecekan kondisi apabila **SigmaAscii[uu]>0** dan apabila **SigmaAscii[uu]<Angka Terkecil** pada posisi iterasi uu, di mana parameter **Angka Terkecil** adalah nilai pembanding frekuensi karakter dengan frekuensi karakter sebelumnya dalam iterasi uu, artinya pada saat kondisi posisi array **SigmaAscii[uu]** nilainya lebih besar dari 0 dan nilainya lebih kecil dari **Angka Terkecil** sebelumnya, maka ambil isi array dan berikan nilainya pada parameter (**angka Terkecil=SigmaAscii[uu]**), kemudian simpan kode ASCII-nya ke dalam array **KarakterHuf[u]** pada posisi nilai iterasi u, dan simpan frekuensi karakter ke dalam array **JumlahKarakterHuf[u]** pada posisi u. Sebelum iterasi berlanjut, isi array **SigmaAscii[]** pada posisi **KarakterHuf[u]** diubah nilainya menjadi **nol** yang menandakan bahwa karakter tersebut telah dibaca dan disimpan di dalam array **KarakterHuf[]**. Parameter **KarakterLD0** adalah parameter yang digunakan untuk proses pencatatan jumlah level pada pohon Huffman, nilai ini ditentukan dengan penjumlahan **KarakterLD0=KarakterLD0+1.**

2. Proses Pembentukan Pohon Huffman.

Tahap awal proses pembentukan pohon Huffman adalah dengan menentukan jumlah level (tingkatan) *tree*-nya.. Proses tersebut dibentuk dengan iterasi sebagai berikut:

```
//Iterasi untuk penentuan jumlah level pohon Huffman
1. Selama u masih dalam KarakterLDO
2. LevelTree[u]=1
3. KarTree[u]=Huruf dari ASCII KarakterHuf[u]
   //Simpan karakter
4. FrekTree[u]=JumlahKarakterHuf[u]
```

Jumlah level *tree* ditentukan pada array **LevelTree[]** dengan iterasi sebanyak jumlah nilai pada parameter **KarakterLD0.** Jadi pada contoh sebelumnya untuk file yang berisi kalimat "PERKARA" ada 4 level untuk *tree*-nya. Statement **KarTree[u]=Huruf dari ASCII KarakterHuf[u]** berfungsi sebagai pencatat karakter ke dalam array **KarTree[]** pada posisi u dengan merubah kode ASCII yang ada pada array **KarakterHuf[u]** menjadi bentuk karakter. Misal pada contoh kata di atas kode pada posisi u = 41 dalam array **KarakterHuf[41]** maka karakter yang disimpan adalah "A", atau pada posisi u = 80 (**KarakterHuf[80]** maka karakter yang disimpan adalah "P", dan seterusnya.

3. Proses Pembentukan dan Penggabungan *Node* Pohon

Seperti yang telah dijelaskan sebelumnya bahwa proses pembentukan node-node (cabang) dalam pohon Huffman adalah dengan menggabungkan dua *node* yang paling kiri dan membuat *node* baru yang merupakan gabungan dari dua *node* tersebut, di mana node gabungan ini memiliki cabang pada level di bawahnya yang berisi dua *node* yang digabungkan tersebut.

Pertama yang dilakukan dalam proses ini adalah membuat variabel untuk menentukan jumlah level yaitu: **JumTree** **KarakterLD0**, di mana nilai

KarakterLD0 adalah hasil dari penjumlahan pada iterasi sebelumnya.

Selanjutnya adalah melakukan iterasi sebanyak jumlah karakter.

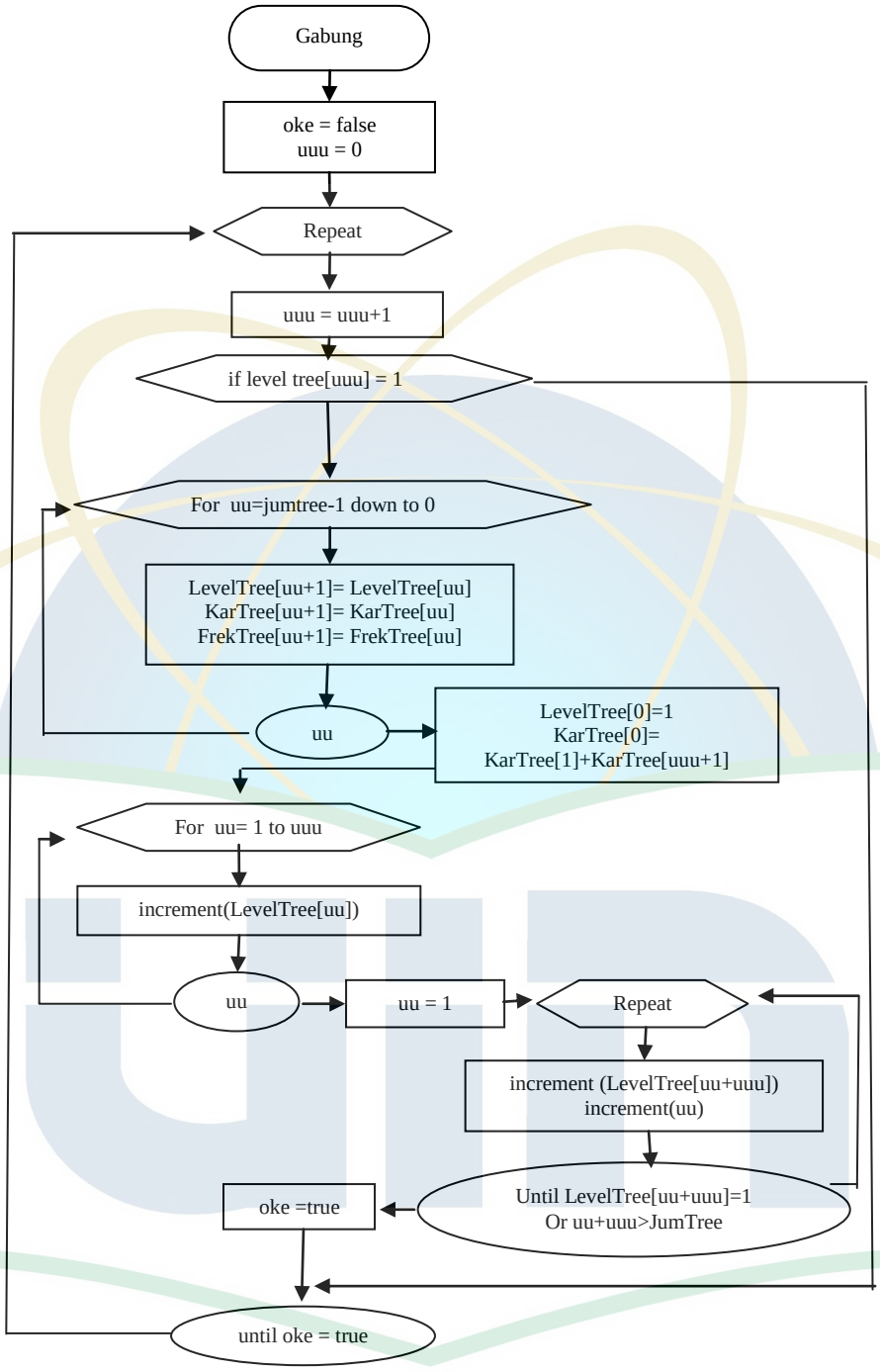
Fungsi iterasi ini adalah untuk pengurutan pembentukan pohon, dan penggabungan node. Dalam iterasi ini dilakukan iterasi lain, yaitu:

a. Iterasi pertama adalah untuk menggabungkan dua *node* paling kiri dan penentuan posisi level *node-node* tersebut di dalam pohon. Lihat algoritma tersebut:

```
Selama u masih dalam KarakterLDO  //Iterasi
sebanyak jml Karakter
  ...
  ...
Looping
```

```
//Prosedur iterasi penggabungan node
1.  oke = false
2.  uuu = 0
3.  Selama oke belum true
4.    uuu = uuu+1
5.    Jika LevelTree[uuu]=1 maka
6.      uu = jumptree
7.      Selama uu > 0                    //Geser
          posisi array
8.        LevelTree[uu+1]=LevelTree[uu]
9.        KarTree[uu+1]=KarTree[uu]
10.       FrekTree[uu+1]=FrekTree[uu]
11. looping uu
12. LevelTree[0]=1
13. KarTree[0]=KarTree[1]+KarTree[uuu+1]
          //Gabungkan 2 karakter
14. FrekTree[0]=FrekTree[1]+FrekTree[uuu+1] //
          jumlahkan Frekuensi
15. JumTree = JumTree+1
16. Selama uu masih dalam uuu
17.       LevelTree[uu] = LevelTree[uu+1]
18. looping uu
19. uu=1
20. Ulangi
21.       increment(LevelTree[uu+uuu])
22.       increment(uu)
```

Dalam prosedur ini dilakukan beberapa tahap penggabungan, yaitu menggeser posisi karakter dalam array, menggabungkan karakter dan menjumlahkan frekuensinya, kemudian menentukan posisi level *node* karakter tersebut. Jika dibentuk dalam *flowchart* maka dapat dilihat pada gambar berikut



Gambar 8. *Flowchart* Prosedur

Penggabungan *Node*

- Dalam iterasi ini, sebelum karakter-karakter digabungkan terlebih dahulu dilakukan iterasi **for uu=JumTree-1 downto 0** untuk menyiapkan *array* pada posisi 0 sebagai tempat karakter hasil dari penggabungan, yaitu dengan menggeser posisi array satu tingkat lebih tinggi dari posisi sebelumnya.
- Selanjutnya adalah penelusuran isi dari array **KarTree[]** dan merubah isinya pada posisi 0 dengan karakter hasil penggabungan dari isi array pada posisi 1 dan isi array pada posisi uuu+1 (**KarTree[0]= KarTree[1]+KarTree[uuu+1])**, begitu juga dengan penelusuran frekuensi karakternya, pada posisi 0 dengan array **FrekTree[]** isi array adalah jumlah frekuensi karakter yang telah dijumlahkan dari nilai array pada posisi1 dan posisi uuu+1 (**FrekTree[0]=FrekTree[1]+FrekTree[uuu+1])** .
- *Statement* dalam iterasi **for uu 1 to uuu** adalah inisialisasi level tiap cabang (node) sesuai dengan posisi urutan karakter yang tercatat dalam array **KarTree[].**

b. Iterasi kedua, yaitu membandingkan frekuensi *node* pada posisi kiri dengan posisi kanan, jika posisi kiri lebih besar daripada posisi kanan maka tentukan nilai pergeseran node tersebut. Lebih jelasnya lihat algoritma berikut:

```
//iterasi perbandingan frekuensi node
1. FrekSource=FrekTree[0]
2. NoAlih=0
3. Selama uuu masih dalam Jumtree
4.    FrekTarget=FrekTree[uuu]    //bandingkan
          frekeunsi 2 karakter
5.    Jika (FrekSource>FrekTarget) dan
          (LevelTree[uuu]=1 maka
6.        NoAlih=uuu
```

Pertama, catat frekuensi karakter dalam array **FrekTree[0]** ke dalam variabel frekuensi sumber(**FrekSource**), kemudian lakukan iterasi uuu sebanyak jumlah *tree*. Bandingkan nilai frekuensi sumber dengan frekuensi target(**FrekTarget**), jika kondisi sumber lebih besar dari kondisi target (nilai *array* **FrekTree[]** pada posisi uuu) dan *node* adalah *parent* maka simpan jumlah peralihan/pergeseran ke dalam variabel **NoAlih=uuu.**

c. Iterasi ketiga, mengurutkan posisi node berdasarkan frekuensi yang terkecil, dengan membandingkan jumlah frekuensi node yang telah digabungkan dan dijumlahkan dengan *node* lainnya. Iterasi ini dijalankan dengan kondisi nilai pergeseran tidak sama dengan atau lebih besar dari 0. *Statement* ini dapat dilihat pada algoritma berikut:

```
//Fungsi pengurutan kembali node setelah digabungkan
1.  Jika NoAlih>0 maka
2.      uuu=0
3.      JumPindah=0
4.      Ulangi loop
5.      LevelTree[JumTree+uuu]=LevelTree[uuu]
6.      KarTree[JumTree+uuu]=KarTree[uuu]
7.      FrekTree[JumTree+uuu]=FrekTree[uuu]
8.      increment(JumPindah)
9.      increment(uuu)
10.     SampaiLevelTree[uuu]=1
11.     Ulangi loop
12.     LevelTree[uuu-JumPindah]=LevelTree[uuu]
13.     KarTree[uuu-JumPindah]=KarTree[uuu]
14.     FrekTree[uuu-JumPindah]=FrekTree[uuu]
15.     increment(uuu)
16.     Sampai (LevelTree[uuu]=1) dan
        (FrekTree[uuu]>=FrekSource)
17.     Selama uu masih dalam JumPindah do
18.     LevelTree[uuu-JumPindah+uu-1]=LevelTree[JumTree+uu-1]
19.     KarTree[uuu-JumPindah+uu-1]=KarTree[JumTree+uu-1]
20.     FrekTree[uuu-JumPindah+uu-1]=FrekTree[JumTree+uu-1]
```

Dalam algoritma ini dilakukan beberapa iterasi, yaitu:

- Iterasi untuk menyimpan karakter-karakter yang telah digabungkan untuk membuat cabang pada susunan level *node* dan karakternya pada pohon. Dalam iterasi ini juga ditentukan nilai jumlah perpindahan untuk proses pengurutan *node* selanjutnya (baris 4-10).
- Iterasi untuk mengubah posisi karakter dalam array **KarTree[]** sesuai nilai *node* dengan frekuensi yang terkecil. Iterasi ini diproses sampai dengan kondisi nilai frekuensi dalam *array* **FrekTree[uuu]** pada posisi uuu lebih besar atau sama dengan nilai frekuensi sumber. (baris 11-21).

4. Frekuensi Inisialisasi *File* Hasil Kompresi

Dalam aplikasi ini hasil pemampatan, pada awal data *file* diberi tanda atau inisial, sehingga sewaktu pengembalian ke *file* asli dapat dikenali apakah *file* tersebut benar merupakan hasil pemampatan dengan agoritma ini.

Pada aplikasi ini format pengenal *file* tersebut ditulis pada byte pertama, kedua dan ketiga dengan karakter H, U, dan F. karakter keempat, kelima dan keenam berisi informasi ukuran *file* asli dalam byte, 3 karakter ini berisi FFFFFF H atau 16.777.215 byte. Karakter ketujuh berisi informasi jumlah karakter yang memiliki kode Huffman atau dengan kata lain jumlah karakter yang frekuensi kemunculannya pada *file* asli lebih dari nol, jumlah tersebut dikurangi satu dan hasilnya disimpan pada karakter ketujuh pada *file* pemampatan.

Misalnya suatu *file* dengan ukuran 3.000 byte dan seluruh karakter ASCII terdapat pada *file* tersebut, jadi:

Karakter 1-3 : HUF
Karena 3.000 = BB8 H, maka :
Karakter 4-6 : 000B88

Karena seluruh karakter ASCII terdapat pada file tersebut, jadi jumlah karakter yang memiliki kode Huffman adalah 256 buah, 256-1=225=FF H maka:

Karakter 7 : FF

Maka 7 byte pertama isi *file* pemampatan adalah:

Mulai dari karakter ke delapan berisi daftar kode Huffman berturut-turut karakter (1 byte), kode Huffman (2 byte) dan panjang kode Huffman (1 byte). 4 byte berturutan ini diulang untuk seluruh karakter yang dikodekan. Selanjutnya *file* diisi hasil pemampatan dengan algoritma Huffman seperti yang telah dijelaskan sebelumnya. Proses pelaksanaan algoritma tersebut adalah sebagai berikut:

```
BufC[1]='H'
BufC[2]='U'
BufC[3]="F'
KodeTemp=Hasil dari KonvesiUkuran file ke Hex
BufC[4]=String dari nilai hex KodeTemp pada posisi 1
 BufC[5]=String dari nilai hex KodeTemp pada posisi 3
BufC[6]= String dari nilai hex KodeTemp pada posisi 5
BufC[7]=Jumlah karakter Huffman dikurangi 1
```

Untuk lebih jelasnya, pada contoh sebelumnya, yaitu *file* yang berisi karakter :"PERKARA". Jika *file* ini dimampatkan dengan metode Huffman, maka *file* hasil pemampatan akan didapat sebagai berikut:

Karakter 1-3 : HUF = 48 55 46 H

Karena *file* tersebut berukuran 7 byte, jadi:

Karakter 4-6 : 00 00 07 H.

Daftar kode Huffman telah dicari pada bab sebelumnya yaitu:

E= 010
K= 011
P= 00
A= 10
R= 11

Karena jumlah karakter yang memiliki kode Huffman ada 5 buah, maka 5-1=4.

Karakter 7 : 04 H
Karakter 8 : karakter E = 45 H.

Karena kode Huffman dari karakter E adalah 010, sedangkan tempat yang disediakan sebanyak 2 byte maka karakter 9 dan 10 menjadi 00000000 00000010 B.

Karakter 9 : 00000000 B = 00 H.
Karakter 10 : 00000010 B = 02 H.

Panjang kode Huffman dari karakter E adalah 3 bit (010), jadi:

Karakter 11 : 03 H.

Cara tersebut diulang untuk karaktar K, P, A dan R sehingga didapat:

Karakter 12 : karakter K = 4B H.
Karakter 13 : 00000000 B = 00 H.
Karakter 14: 00000011 B = 03 H.
Karakter 15 : 03 H.
Karakter 16: karakter P = 50.
Karakter 17 : 00000000 B = 00 H.
Karakter 18 : 00000000 B = 00 H
Karakter 19 : 02 H.
Karakter 20 : karakter A = 41 H.
Karakter 21 : 00000000 B = 00 H.
Karakter 22 : 00000010 B = 02 H.
Karakter 23 : 02 H.
Karakter 24 : karakter R =52 H.
Karakter 25 : 00000000 B = 00 H.
Karakter 26 : 00000011 B = 03 H.
Karakter 27 : 02 H.

Selanjutnya pemampatan karakter-karakter "PERKARA" adalah: 0001011011101110. Jika dipotong-potong menjadi 8 bit tiap bagian akan menjadi:

00010110 = 16 H.
11101110 = EE H.

Jadi: Karakter 28 : 16 H.
Karakter 29 : EE H.

File hasil pemampatan akan menjadi berukuran 29 byte yang dalam heksadesimal berisi:

48 55 46 00 00 07 04 45 00 02 03 4B 00 03 02 50 00 00 02 41 00 02 02 52 00 03 02 16 EE.

```
37.     KarTree[uu+1] ←KarTree[uu]
38.     FrekTree[uu+1] ←
FrekTree[uu]
39.  end for
40.  LevelTree[0] ←1
41.  KarTree[0] ←
KarTree[1]+KarTree[uuu+1]
//Gabungkan karakter
42.  FrekTree[0] ←
FrekTree[1]+FrekTree[uuu+1]
//Jumlahkan frekunsi
43.  increment(JumTree)
44.  for uu ←1 to uuu do
//Level cabang pohon
45.     Increment(levelTree[uu])
46.  end for uu
47.  uu ←1
//Target & cabang pohon target
48.  repeat
49.     increment(LevelTree[uu+uuu])
50.     increment(uu)
51.  until(LevelTree[uu+uuu]=1) or
(uu+uuu>JumTree)
```

	H	U	F	Chr (00 H)	Chr (0BH)	Chr (88H)	Chr (FFH)	…
Bit ke-	1	2	3	4	5	6	7	8…

Jika dibandingkan dengan *file* aslinya yang berukuran 7 byte, hasil ini bukan menjadi lebih kecil, akan tetapi menambah byte sebesar 29-7=22 byte. Hal ini wajar dalam *file* yang berukuran sangat kecil seperti ini, tapi dalam *file-file* yang berukuran besar, algoritma Huffman ini sangat efektif. Dari keseluruhan proses algoritma tersebut berikut *list* lengkap *pseudocode* untuk proses kompresi data algoritma Huffman:

D. Dekompresi Hasil Pemampatan

Dekompresi adalah proses pengembalian *file* dari hasil pemampatan menjadi bentuk aslinya kembali. Dalam aplikasi ini proses pengembalian *file* mengacu kembali kepada kode Huffman yang telah dihasilkan. Proses pertama adalah memeriksa apakah *file* tersebut adalah hasil kompresi dengan algoritma Huffman, jika benar maka lakukan dekompresi. Lihat potongan algoritma berikut:

```
Jika Buf[1]+Buf[2]+Buf[3]='HUF' Maka lakukan dekompresi
```

Pada contoh sebelumnya hasil pemampatan adalah: 000101101100 1110 dengan Kode Huffman:

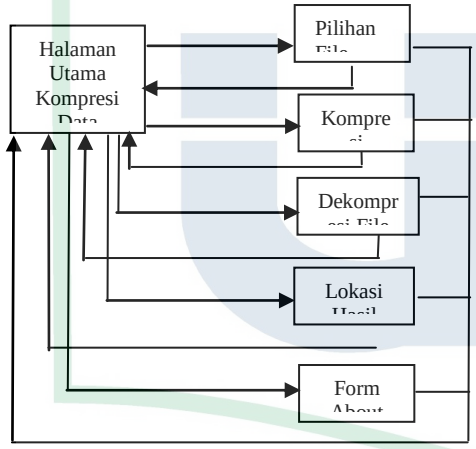
```
E=010   K=011   P=00   A=10   R=11
```

Fungsi pengembalian (dekompresi) dapat dilihat pada algoritma berikut:

```
1.  Inisialisasi i, ii, uu, BitMin
2.  Inisialisasi BitMin, KodeTem, FCSize
3.  KodeTem = (Buf[4]+Buf[5]+Buf[6])
4.  FCSize = hasil konvensi KodeTem ke string
5.  KodeTem = nilai hex Buf[7]
6.  KarakterLDO = hasil konvensi (nilai
    kodeTem+1)
7.  ii = 7
8.  BitMin = 256
9.  Selama uu masih dalam KarakterLDO
    //Kode Huffman
10. ii = ii+4;
11. KarakterHuf[uu-1] = nilai (Buf[ii-3])
    //byte ke-1
12. JumlahKarakterHuf[uu-1] = nilai(Buf[ii])  //byte
    ke-4
13. Jika JumlahKarakterHuf[uu-1] < BitMin maka
14. BitMin = JumlahKarakterHuf[uu-1]
15. i = Buf[ii-2]*256+Buf[ii-1]
    //byte ke-2 &3
16. KodeTem = Nilai Biner dari i
17. Jika panjang(KodeTem) <
    JumlahKarakterHuf[uu-1] maka
18. KodeTem = Nilai String dari
    (JumlahKarakterHuf[uu-1])
19. Panjang(KodeTem)+KodeTem
20. KodeHuf[uu-1] = KodeTem;
```

E. STD Rancangan Program

STD (*State Transition Diagram*) program kompresi data ini dapat ditunjukkan Gambar 9.



BukaFile_Klik
Open_klik
Kompresi_Klik
Ok_Klik
Dekompresi_Klik
Ok_Klik
Lokasi_Klik
Ok_Klik
About_Klik
Cancel_Klik

Gambar 9. STD Rancangan Program

F. Implementasi

Pada tahap implementasi ini dibahas sarana-sarana pendukung yang diperlukan dalam pengembangan program kompresi data, juga rancangan program yang dibuat.

1. Spesifikasi Perangkat Lunak dan Perangkat Keras.

Perangkat lunak yang digunakan untuk membuat dan mengembangkan program ini adalah sebagai berikut:

a. Microsoft Windows XP Profesional.
b. Borland Delphi 7.

Sedang perangkat keras yang digunakan adalah *Personal Computer* dengan spesifikasi sebagai berikut:

a. Processor Intel Pentium IV 2,66 GHz.
b. Motherboard Foxconn 648/661 series.
c. DDRAM 256MB PC-3200.
d. VGA AGP Radeon 9000 64MB.
e. *Harddisk* 80GB.
f. *Mouse* dan *Keyboard*.
g. Monitor resolusi 1024 x 768

2. Rancangan Program

WinZal adalah program untuk melakukan proses kompresi dan dekompresi data teks ini. *File* hasil kompresi dengan WinZal secara baku berekstensi **.qbo**. WinZal dapat digunakan untuk melakukan kompresi terhadap semua jenis *file*, dengan hasil kompres yang bervariasi tergantung jenis file tersebut.

Program ini dapat dijalankan secara mandiri, yaitu dengan cara mengklik dua kali pada *file* **WinZal.exe** dari *windows explorer* atau mengklik dua kali pada *shortcut file* tersebut, seperti pada *desktop* komputer. Dengan cara ini program dapat digunakan baik untuk kompresi maupun untuk dekompresi *file*.

Pada tampilan awal terdapat dua menu, yaitu menu **Pilihan** dan menu **Bantuan**, juga terdapat *Toolbar* yang berisi tombol-tombol di antaranya:

- Tombol **Buka File**, berfungsi untuk mencari dan memiih *file* yang akan dikompresi atau didekompresi.

- Tombol **Dekompresi**, berfungsi untuk melakukan proses pengembalian *file* ke jenis dan ukurannya semula.
- Tombol **Kompresi**, berfungsi untuk melakukan proses kompresi *file* yang dipilih.
- Tombol **About**, berfungsi untuk menampilkan kotak dialog penulis.
- Dan tombol **Keluar**, berfungsi untuk menutup dan keluar dari program.

Adapun fungsi untuk menu dan submenu pada program ini adalah sebagai berikut:

❖ Menu **Pilihan**.

Dalam menu Pilihan terdapat dua submenu pilihan, **Buka File** dan **Keluar.** Submenu Buka File digunakan untuk mencari file/berkas yang akan dikompresi ataupun yang akan didekompresi. Jika menu Buka *File* dipilih atau tombol Buka *File* diklik maka akan muncul kotak dialog untuk memilih *file*.

- Proses Kompresi *File*.

Untuk melakukan kompresi *file*, pertama pilih pada *file* yang dikehendaki lalu klik tombol *open* atau dapat langsung mengklik dua kali pada *file* tersebut, maka secara otomatis *file* yang dipilih akan tampil dalam *list file* yang akan dikompres. Dalam *list file* ini terdapat 3 kolom keterangan, kolom pertama menunjukkan nama *file*, kolom kedua lokasi *file* tersebut berada, dan kolom ketiga keterangan ukuran *file* dalam *byte*.

Langkah selanjutnya adalah dengan menekan/mengklik tombol **Kompresi**. Ketika tombol ini diklik maka akan muncul kotak dialog untuk input nama hasil kompresi. Pemakai dapat mengetikan nama *file*-nya pada kotak input, atau jika dikosongkan maka program secara otomatis akan menyimpan file hasil kompresi dengan nama hasil gabungan dari susunan ***WinZalddmmHns.qbo***, di mana **dd** adalah 2 digit tanggal, **mm** adalah 2 digit bulan dan **Hns (*Hour, minute, second*)** adalah jam yang berlaku pada sistem komputer saat kompresi diproses.

Setelah menentukan nama *file*, tekan tombol OK, maka program akan memproses *file* untuk dikompresi dan menyimpan *file* kompresi tersebut sesuai dengan nama yang diinput.

- Proses Dekompresi *File*.

Untuk proses Dekompresi pada program ini adalah dengan cara memilih *file* hasil kompresi, yaitu *file* yang berekstensi **.qbo**, pada saat kotak dialog untuk mencari dan memilih *file* terbuka. Untuk memudahkan pencarian *file* hasil kompresi, maka ketika kotak dialog tersebut muncul, pemakai dapat memilih pada combo *Files of type* dengan pilihan **Hasil Kompresi (*.qbo)**.

Langkah selanjutnya untuk dekompresi *file* ini adalah dengan mengklik tombol dekompresi pada tampilan utama. Kemudian program akan menampilkan kotak konfirmasi untuk menanyakan apakah *file* tersebut akan didekompresi dan disimpan pada direktori yang terpilih, jika tidak maka proses dekompresi akan dibatalkan.

❖ Menu **Bantuan**.

Pada Menu ini terdapat dua submenu yaitu:

- Menu **Petunjuk**, berfungsi untuk menampilkan bantuan cara untuk menggunakan program WinZal ini.
- Menu **About**, yaitu untuk menampilkan sekilas tentang profil peneliti dan pembuat program WinZal.

KESIMPULAN

Berdasarkan pembahasan tentang implementasi algoritma kompresi data Huffman untuk memperkecil ukuran *file* yang peneliti kembangkan, maka peneliti menarik kesimpulan sebagai berikut:

1. Salah satu cara untuk mempersingkat waktu *upload* dan *download* sebuah *file* dari dan ke suatu situs di internet yang terkoneksi melalui line telepon adalah dengan melakukan kompresi terhadap *file* tersebut.
2. Kompresi pada *file* juga dapat mengurangi ukuran file tersebut agar dapat menambah ruang pada media penyimpanan komputer seperti *harddisk*, disket, dan lain sebagainya.
3. Algoritma kompresi data Huffman mempunyai rasio kompresi yang cukup tinggi terutama bila digunakan untuk melakukan kompresi terhadap file teks.
4. Kerahasiaan data yang dikompres dengan menggunakan program ini lebih terjamin, karena yang dapat membaca isi file tersebut hanya pemakai yang memiliki dekompresornya (WinZal).

REFERENSI

Campos, S.E. Arturo. 2005. Huffman Introduction. [Online]. Tersedia: www.arturocampos.com/cp_ch3-0.html. [Agustus 2005].

Dipperstein, Michael. 2005. Huffman Code Discussion and Implementation. [Online]. Tersedia: www.michael.dipperstein.com/huffman/index.html. [Juli 2005].

Pranata, Antony. 2001. Pemrograman Borland Delphi, Edisi ke-3. Yogyakarta: Andi Yogyakarta.

Schindler, Michael. 2005. Practical Huffman Coding. [Online]. Tersedia: www.commpressconuslt.com/huffman.html. [Agustus 2005].

Sujaini, Herri. & Mulyani, Yessi. 2000. Algoritma Runlegth, Half-Byte & Huffman, Edisi I Bandung: Institut Teknologi Bandung.