KOMPRESI DATA

BAGIAN I : KONSEP DASAR

A. PENDAHULUAN

Teknik kompresi data ada dua yaitu :

(1) *lossy data-compression* dan;

(2) *lossless data-compression*.

Lossy data-compression masih memperbolehkan adanya kesalahan dalam proses kompresi atau dekompresi, selama kesalahan proses tersebut tidak terlalu mengubah pola pokok dari data yang dikompres. Teknik kompresi ini biasanya digunakan untuk kompresi data gambar dan suara.

Lossless data-compression digunakan jika kesalahan tidak boleh terjadi sama sekali. Artinya data sebelum dikompres harus sama dengan data hasil dekompresi. Teknik ini biasanya digunakan untuk rekaman database, file-file word prosessing dan data-data lain yang memerlukan proses kompresi dan dekompresi akurasi tinggi.

Kompresi data merupakan penerapan Teori Informasi yang merupakan cabang ilmu matematika. Teori ini muncul pada akhir 1940 dan dikembangkan pertama kali oleh Claude Shannon di Laboratorium Bell. Dorongan pengembangan ilmu ini adalah berdasarkan beberapa pertanyaan tentang apa itu informasi , termasuk bagaimana cara penyimpanan dan pengiriman pesan (informasi).

Dalam Teori Informasi digunakan istilah "*entropy*" yang diambil dari istilah Ilmu Thermodinamika. Makna dari entropy sebetulnya menunjukkan ketidakteraturan sesuatu (dalam hal ini informasi). Semakin tinggi nilai entropy sebuah *message* (informasi) berarti semakin banyak simbul yang muncul dalam informasi tersebut. Dengan demikian nilai entropy berkaitan dengan probabilitas simbul dari sebuah informasi.

Nilai entropy dari sebuah simbul didefinisikan dengan logaritma negatif dari probabilitasnya. Untuk menentukan banyaknya bit dari simbul yang digunakan dalam informasi tersebut dapat dinyatakan dengan logaristma berbasis 2 yaitu :

$$Jumlah\ bit\ dari\ Simbul = -\log_2 (Probablilitas\ Simbul)$$

Sedangkan entropy keseluruhan informasi merupakan jumlah entropy dari masing-masing simbul.

Sebagai gambaran misalkan dalam sebuah tulisan, probabilitas huruf "e" adalah 1/16. Maka jumlah bit untuk kode huruf tersebut adalah 4 bit. Sedangkan kalau menggunakan kode ASCII jumlah bit yang digunakan adalah 8-bit.

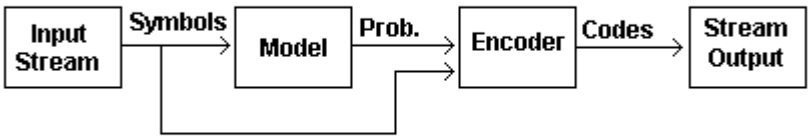
Dalam perhitungan entropy untuk kompresi data berbeda dengan perhitungan entropy dalam thermodinamika. Untuk kompresi data kita menggunakan bilangan yang tidak absolut, artinya tergantung cara memandang dalam memperoleh nilai probabilitas sebuah simbul.Untuk orde-0 kita mengabaikan simbul-simbul yang telah muncul sebelumnya. Atau dengan kata lain setiap simbul yang akan muncul memiliki nilai probabilitas yang sama. Model ini disebut dengan *zero-memory source* (sumber tanpa memori). Nilai entropy untuk zero-memory source yang menhasilkan *S* buah simbul dapat dihitung dengan persamaan :

$$H(S) = \sum_{S} P(Si) \log_2 \frac{1}{P(Si)}$$

dimana *P(Si)* adalah probabilitas setiap simbul.

B. PEMODELAN DAN PENGKODEAN DALAM KOMPRESI DATA

Pada umumnya proses kompresi data meliputi pembacaan simbul, mengubah kode untuk tiap-tiap simbul kemudian menuliskan simbul-simbul dengan kode yang baru. Jika proses kompresi berjalan efektif, maka hasil file yang diperoleh dapat lebih kecil dari file aslinya. Efektif tidaknya sistem kompresi data tergantung dari pemodelan dan pengkodean yang digunakan. Model dan Kode yang digunakan dalam kompresi data dibentuk berdasarkan nilai probabilitas tiap-tiap simbul.



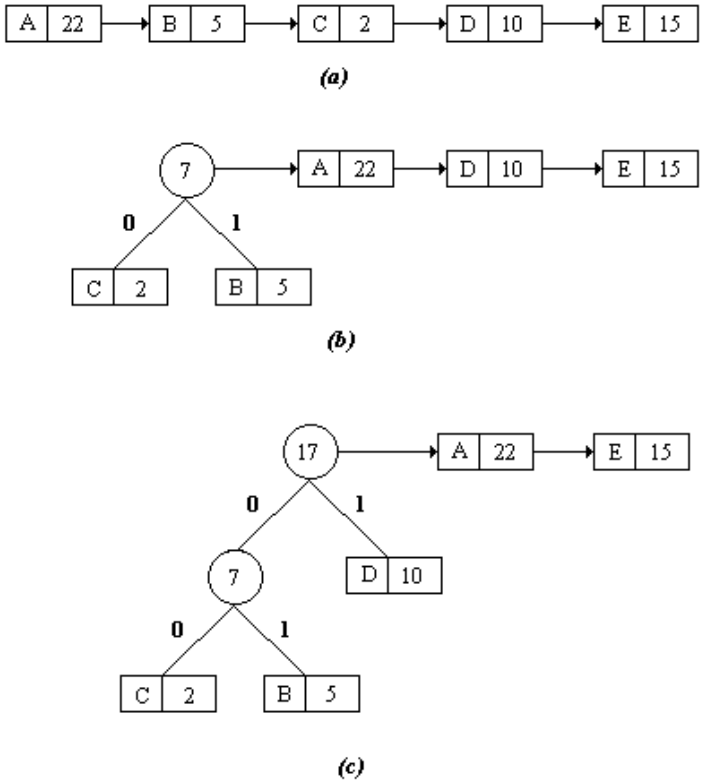
Gambar-1. Model Statistik dengan Huffman Encoder

C. ALGORITMA PENGKODEAN HUFFMAN

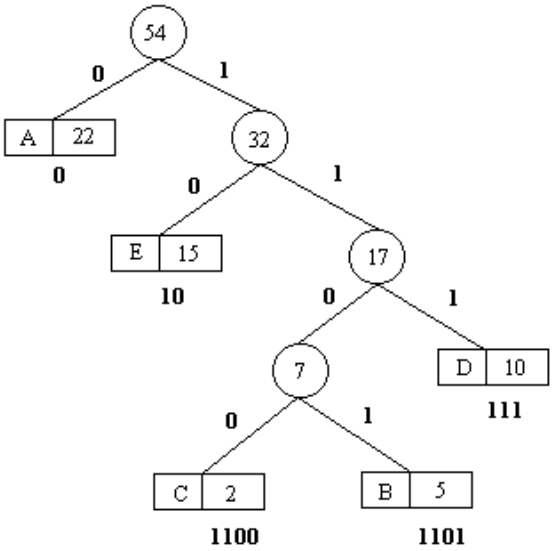
Algoritma pengkodean Huffman sebetulnya hampir sama dengan algoritma pengkodean Shannon-Fano. Yaitu simbul yang mempunyai probabilitas paling besar diberi kode paling pendek (jumlah bit kode sedikit) dan simbul dengan probabilitas paling kecil akan memperoleh kode paling panjang (jumlah bit kode banyak). Kode tersebut diperoleh dengan cara memyusun sebuah pohon Huffman untuk masing-masing simbul berdasarkan nilai probabilitasnya. Simbul yang memiliki probabilitas terbesar akan dekat dengan root (sehingga memiliki kode terpendek) dan simbul yang memiliki probabilitas terkecil akan terletak jauh dari root (sehingga memiliki kode terpanjang).

Pohon Huffman merupakan pohon-biner (*binary tree*). Sedangkan Algoritma penyusunan pohon beserta pengkodeannya adalah sebagai berikut :

- *Berdasarkan daftar simbul dan probabilitas, buat dua buah node dengan frekuensi paling kecil.*
- *Buat node parent dari node tersebut dengan bobot parent merupakan jumlah dari probabilitas kedua node anak tersebut.*
- *Masukkan node parent tersebut beserta bobotnya ke dalam daftar, dan kemudian kedua node anak beserta probabilitasnya dihapus dari daftar.*
- *Salah satu node anak dijadikan jalur (dilihat dari node parent) untuk pengkodean 0 sedangkan lainnya digunakan untuk jalur pengkodean 1.*



Gambar-2. Langkah-langkah Pembentukan Pohon Huffman



Gambar-3. Pohon Huffman yang terbentuk beserta kode simbul

- *Langkah-langkah di atas diulang sampai hanya tinggal satu node bebas yang tersisa dalam daftar (dengan bobot 1) dan digunakan sebagai akar pohon tersebut.*

Suatu contoh dalam daftar terdiri dari 5 huruf beserta frekuensinya (dapat dilihat pada *Gambar-2a*.). Nilai frekuensi tiap-tiap simbul dapat juga dinyatakan dalam nilai probabilitas, dengan cara membagi masing-masing nilai frekuensi dengan jumlah seluruh simbul yang muncul dalam pesan (bilangan yang terdapat dalam akar pohon).

Hasil algoritma *pertama, kedua* dan *ketiga* seperti ditunjukkan *Gambar-2b*. Dalam membuat dua buah node untuk algoritma pertama, node dengan frekuensi terkecil selalu diletakkan di sebelah kiri. Algoritma *keempat* (penentuan jalur untuk kode 0 dan 1) diperlihatkan pada *Gambar-2b* atau *Gambar-2c*. Sedangkan pohon Huffman yang terbentuk ditunjukkan pada *Gambar-3*.

D. ALGORITMA PENGKODEAN ELIAS

Algoritma Elias Encoder menggunakan rumus logaritma berbasis 2 terhadap nomor urut simbul dalam membuat kodenya. Nomor urut ini dibentuk berdasarkan probabilitas simbul. Simbul yang memiliki propabilitas terbesar akan memiliki nomor urut terkecil dan simbul yang memiliki probabilitas terkecil akan memiliki nomor urut terbesar. Dengan demikian simbul yang memiliki nomor urut terkecil akan diberi kode pendek dan simbul yang memiliki nomor urut terbesar akan diberi kode panjang sesuai dengan nilai logaritmanya.

Kode Elias terdiri dari dua bagian yaitu *prefix* yang terdiri dari sejumlah deretan bit ‘0’ diikuti dengan sejumlah deretan bit akhiran. Aturan pembuatan kode untuk sebuah integer ***n*** dari nomor simbul adalah :

- Banyaknya bit ‘0’ dalam prefix dihitung dengan persamaan $\log_2(n+1)$
- Deretan bit akhiran merupakan bilangan biner dari $n + 1$.

Contoh :

Nomor	Kode
0	1
1	010
2	011
3	00100
4	00101

5	00110
6	00111
7	0001000

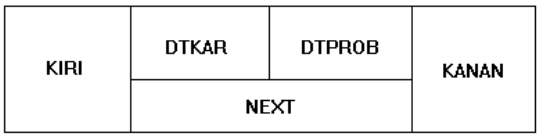
Dalam algoritma Elias Encoder ini proses pertama yang dilakukan untuk keperluan pembuatan kode adalah mencari probabilitas masing-masing simbul kemudian mengurutkan simbul-simbul tersebut berdasarkan probabilitasnya. Sehingga diperoleh simbul yang memiliki probabilitas terbesar menempati nomor paling kecil dan simbul yang memiliki probabilitas paling kecil menempati nomor paling besar.

BAGIAN II : REALISASI PROGRAM

A. ALGORITMA POHON HUFFMAN

Program kompresi data menggunakan Pohon Huffman ini dibuat dengan bahasa Turbo Pascal 7.0. Untuk keperluan penyusunan pohon digunakan stuktur data pointer (struktur data dinamis) karena dirasa lebih menguntungkan. Untuk lebih meningkatkan efisiensi pengunaan pointer maka setiap pointer harus memiliki 3 medan pointer selain 2 medan untuk data (karakter dan probabilitas). Fungsi ketiga medan pointer ini adalah satu medan digunakan untuk menunjuk pointer berikutnya dalam pembuatan senarai berantai, sedangkan dua medan lainnya digunakan untuk menunjuk cabang kiri dan kanan setelah menjadi bentuk pohon.

Bentuk komponen pointer tersebut dapat digambarkan sebagai berikut :



Gambar-4. Struktur komponen pointer untuk pohon Huffman

Dalam struktur di atas, medan *DTKAR* sebagai tempat data simbul sedangkan *DTPROB* sebagai tempat data probabilits dari simbul tersebut. Medan *KIRI* adlah pointer yang akan menunjuk cabang kiri, medan *KANAN* adalah pointer yang akan menunjuk cabang kanan, dan medan *NEXT* adalah pointer yang

digunakan untuk menggandeng komponen dalam bentuk senarai berantai (sebelum pembuatan pohon Huffman).

Deklarasi type berdasarkan struktur komponen di atas adalah:

```
type Node       = ^Simpul;
Simpul          = record
                    dtkar       : char;
                    dtprob      : real;
                    Kiri, Kanan,
                    Next        : Node
                    end;
```

PROGRAM KOMPRESI

Program kompresi dibagi menjadi 3 bagian utama yaitu :

- Proses Membaca Berkas.
- Proses Pengkodean Huffman Tree.
- Proses Penulisan Hasil Kompresi.

1. Proses Membaca File.

Proses ini dilakukan oleh procedure BACA_FILE. Dalam proses ini dilakukan pencarian simbul-simbul yang muncul beserta probabilitasnya. Masing-masing simbul dibaca dalam bentuk byte, kemudian ditransfer kedalam bentuk nilai ASCII dan disimpan dalam sebuah array data karakter dengan indek 0..255. Sedangkan nilai probabilitasnya disimpan dalam sebuah array bilangan dengan indek karakter yang bersangkutan.

Pembacaan file dalam procedure BACA-FILE dilakukan setiap 1 kbyte (1024 byte).Sedangkan algoritma yang digunakan adalah sebagai berikut :

1. *Buka berkas yang akan dibaca.*
2. *(Inisialisasi): pencacah simbul=0;*
3. *Untuk I=0 sampai 1024 (1 Kbyte) lakukan langkah 4;*
4. *Lakukan pengetesan apakah posisi berkas = eof; jika "Ya" keluar dari procedure; jika "Tidak" lakukan langkah 5 sampai 8*
5. *Baca satu byte dari berkas, masukkan ke dt[J];*
6. *Naikkan pencacah jumlah simbul;*
7. *Untuk J=0 sampai 255 cari simbul yang sesuai untuk dt[J];jika dt[J]=j tentukan dtT[char(j)]=dtT[char(j)]+1;*

8. *Ulangi langkah 3 (untuk 1 Kbyte berikutnya) selama posisi berkas <> eof.*
9. *Tutup berkas;*
10. *Selesai.*

Realisasi program procedure BACA_FILE dapat dilihat pada Lampiran A.1. Jika program ini dijalankan dengan file coba.dat sebagai file inputnya maka akan ditampilkan simbul-simbul dalam file tersebut beserta probabilitasnya.

```
a    = 0.40
b    = 0.04
c    = 0.30
d    = 0.10
e    = 0.06
f    = 0.10
```

2. Proses Pembuatan Pohon Huffman.

Fungsi prosedur ini adalah untuk menyusun pohon Huffman berdasarkan probabilitas simbul, kemudian membuat kode berdasarkan pohon tersebut. Procedure pohon Huffman terdiri dari beberapa procedure yaitu:

1. BUAT_LIST_PELUANG
2. CARI_PELUANG_TERKECIL
3. SUSUN_POHON
4. ENCRYPT (Pembuatan Kode).

2.1 Algoritma Procedure BUAT_LIST_PELUANG

Prosedur ini digunakan untuk membuat senarai berantai (linked-list) dari tabel simbul beserta probabilitasnya yang telah disediakan oleh prosedur BACA_FILE. Masukan prosedur ini adalah variabel *Kal* (array probabilitas dengan indeks karakter) dan keluarannya adalah pointer yang menunjuk pada simpul kepala dari senarai berantai *H*.

Sebelum menjalankan prosedur BUAT_LIST_PELUANG, prosedur HUFFMAN memanggil prosedur DUMMY(Kepala) yang artinya membuat pointer boneka dengan nama Kepala. Pointer ini selanjutnya akan digunakan proses.

Algoritma BUAT_LIST_PELUANG adalah :

1.*Inisialisasi: Tentukan Bantu = H;*

2.*Untuk I=0 sampai 255 kerjakan langkah 3;*

3.(Uji probabilitas setiap karakter:

Tes : Apakah Kar[chr(I)] <> 0 ?

Jika "Ya" kerjakan langkah 4 sampai langkah 6;

Jika "Tidak" ulangi langkah 3 untuk I berikutnya;

4.*Panggil procedure DUMMY dengan argumen Bantu^.Next;*

5.(Mengisi simpul) :

Tentukan : Bantu^.Next.dtkar = chr(I);

Bantu^.Next.dtprob = Kar[chr(I)] ;

6.*Tentukan Bantu = Bantu^.Next;*

Selesai.

Dari algoritma tersebut maka karakter yang probabilitasnya sama dengan nol (berarti simbul tersebut tidak muncul) maka tidak akan dimasukkan ke dalam tabel. Sehingga akan menghemat tabel karakter dan probabilitasnya.

Setelah semua data karakter dan peluangnya dimasukkan ke dalam senarai berantai oleh prosedur BUAT_LIST_PELUANG, maka kemudian dilakukan pencarian peluang terkecil dengan prosedur CARI_PELUANG_TERKECIL.

2.2 Prosedur CARI_PELUANG_TERKECIL

Prosedur ini digunakan untuk mencari dua simpul yang berisi peluang terkecil. Masukannya adalah pointer yang menunjuk ke simpul kepala, *H*. Keluaran adalah pointer yang menunjuk ke simpul dengan frekuensi terkecil kedua, *Dua*.

Algoritma mencari dua peluang terkecil adalah sebagai berikut :

1. *(Mencari peluang terkecil pertama).*

Tentukan : Satu = H^.Next, Bantu = Satu^.Next;

2. *Kerjakan langkah 3 dan 4 selama Bantu <> Nil;*
3. *Tes: apakah Satu^.dtprob > Bantu^.dtprob?*

Jika "Ya", tentukan Satu = Bantu;

4. *Tentukan : Bantu = Bantu ^.Next;*
5. *(Mencari peluang terkecil kedua).*

Tes: apakah Satu = H^.Next?

Jika "Ya" tentukan Dua = Satu^.Next;

Jika "Tidak" tentukan Dua = H^.Next;

6. *Tentukan Bantu = Dua^.Next;*
7. *Kerjakan langkah 8 dan 9 selama Bantu <> Nil;*
8. *Tes apakah (Dua^.dtprob > Bantu^.dtprob) dan (Bantu <> Satu)?*

Jika "Ya" tentukan Dua = Bantu;

9. *Tentukan Bantu = Bantu^.Next;*

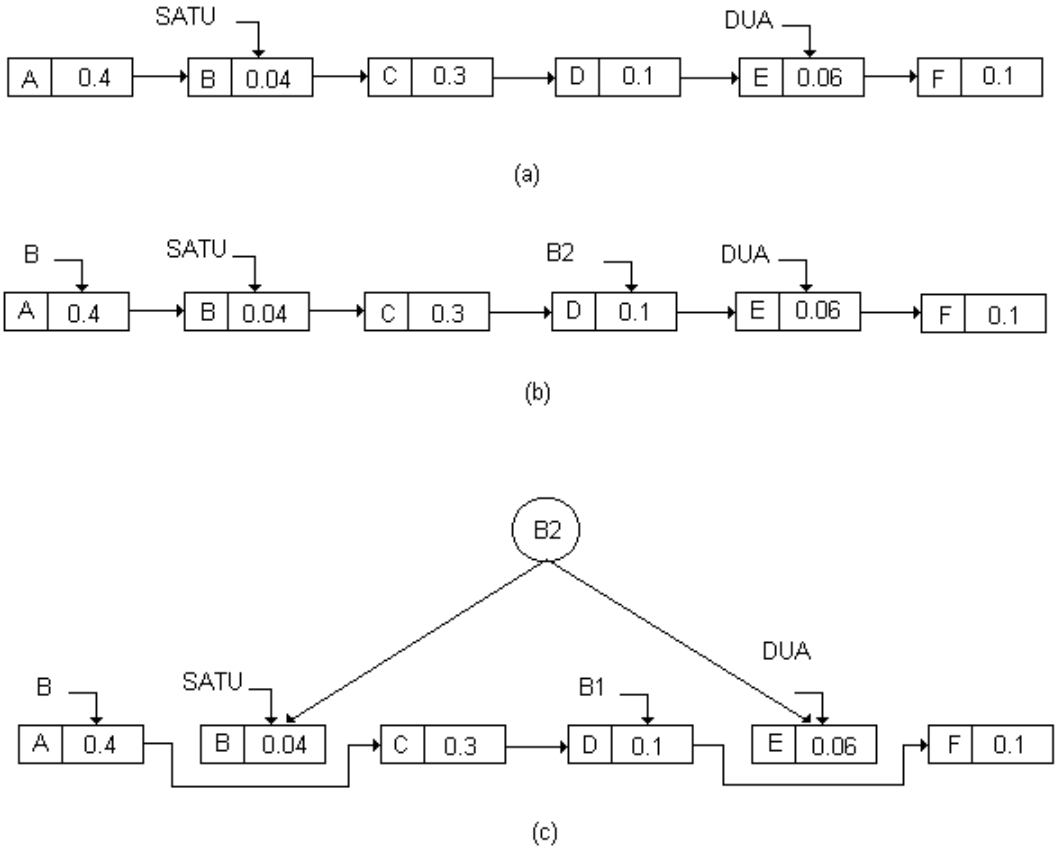
2.3 Prosedur SUSUN_POHON

Dalam algoritma SUSUN_POHON ini selain pointer *Satu,* yang menunjuk ke simpul peluang terkecil pertama, *Dua* yang menunjuk ke simpul peluang terkecil kedua, digunakan pointer *B* dan *B1*. Pointer *B* digunakan untuk menunjuk ke simpul sebelum yang ditunjuk oleh simpul *Satu*, pointer *B1* digunakan untuk menunjuk ke simpul sebelum simpul yang ditunjuk oleh simpul *Dua*. Untuk lebih jelasnya dapat dilihat pada *Gambar-5*.

Dalam program penentuan letak pointer B dan B1 diimplementasikan dalam fungsi *SAMBUNGAN.*

Berdasarkan *Gambar-5,* ada dua titik kritis yang dapat menyebabkan kesalahan. Titik kritis yang pertama apabila pointer *Dua* terletak persis sesudah pointer *Satu*, sehingga pointer *B1* akan menunjuk pointer ke simpul yang sama seperti halnya pointer *Satu*. Titik kritis kedua, apabila pointer *Dua* terletak persis sebelum pointer *Satu*, sehingga pointer *B* menunjuk simpul yang sama seperti halnya pointer *Dua*.

Dengan memperhatikan titik kritis tersebut, maka sebelum proses pembentukan pohon Huffman dilanjutkan harus dilakukan pengetesan terlebih dahulu apakah kita sedang dihadapkan dari salah satu titik kritis tersebut atau tidak. Algoritma penyusunan pohon Huffman adalah sebagai berikut :



Gambar-5. Ilustrasi pembentukan Pohon Huffman

1. *(Menentukan letak pointer B dan B_1);*

Tentukan: *B = SAMBUNGAN(H,SATU)*, *dan*

B_1=SAMBUNGAN(H,Dua);

2. *(Alokasi tempat untuk menjumlah dua peluang).*

Panggil prosedur DUMMY dengan parameter B_2.

3. *(Menyambung cabang kiri dan kanan).*

Tentukan: *B_2^.dtprob = Satu^.dtprob + Dua^.dtprob,*

B_2^.K_iri = Satu, dan

B_2^.K_anan = Dua.

4. *(Tes titik kritis pertama).*

Tes : apakah B^.Next[^.Next = Dua?

Jika "Ya" tentukan B^.Next = Dua^.Next;

Jika "Tidak" tentukan B^.Next = Satu^.Next;

5. *(Tes titik kritis kedua).*

Tes: apakah B1^.Next^.Next = Satu?

Jika "Ya" tentukan B1^.Next = Satu^.Next;

Jika "Tidak" tentukan B1^.Next = Dua^.Next.

6. *(Memindahkan letak simpul yang ditunjuk oleh B2)*

Tentukan: *B2^.Next = H^.Next, dan*

H^.Next = B2.

7. *Selesai.*

2.4 Prosedur ENCRYPT.

Prosedur Encrypt bersama-sama dengan prosedur Prefiks berfungsi untuk membuat kode Huffman berdasarkan pohon Huffman yang telah dibuat. Kode Huffman tersebut diujudkan dalam bentuk string dari '0' dan '1' dan diartikan sebagai bilangan biner, meskipun pada saat penulisan file hasil kompresi kode tersebut diubah dari bentuk bilangan biner menjadi bilangan bertype byte.

Pembuatan kode Huffman adalah dengan membuat daftar simbul seperti yang dihasilkan pada saat mencari probabilitas simbul kemudian dilakukan penelusuran simbul dalam pohon Huffman. Penelusuran selalu dimulai dari ujung akar. Langkah dalam penelusuran untuk cabang kiri dan kanan itulah yang akan menentukan kode Huffman. Jika mengujungi cabang kiri diartikan '0' dan mengunjungi cabang kanan diartikan '1'.

3. Prosedur Penulisan Hasil Kompresi

Prosedur ini menyediakan dua fungsi utama yang berkaitan dengan penulisan file hasil kompresi. Yaitu penulisan **header** dan **badan** file.

3.1 Header

Header file hasil kompresi terdiri dari empat bagian yaitu :

- *informasi jumlah byte file asli*
- *informasi banyak simbul yang terdapat dalam file asli*
- *informasi simbul-simbul yang dalam file asli*
- *informasi probabilitas tiap-tiap simbul dalam file asli*

Informasi besarnya file asal digunakan sebagai ukuran pencacah pada saat dilakukan proses dekompresi file hasil kompresi tersebut. Dengan adanya informasi ini maka program akan mudah dalam menentukan kapan harus menyelesaikan pembacaan data untuk dekompresi file hasil kompresi.

Informasi jumlah simbul digunakan untuk menentukan pencacah yang berhubungan dengan daftar simbul dan daftar probabilitas agar tidak terjadi kesalahan akses data dari file hasil kompresi. Atau dengan kata lain informasi tersebut digunakan untuk menentukan besarnya array/pointer yang mencacat simbul dan probabilitasnya.

Informasi simbul dan *informasi probabilitas* simbul digunakan pembuatan pohon Huffman kembali dalam proses dekompresi.

3.2 Badan File

Badan file terdiri dari data byte hasil kompresi berdasarkan kode Huffman yang dibuat. Setiap byte dalam badan file ini merupakan potongan (karena diujudkan dalam byte berarti sebanyak 8 bit) dari deretan simbul file asal yang telah diubah menjadi deretan kode. Karena setiap kode dari masing-masing simbul tidak sama panjangnya maka perlu dilakukan kerja keras dan hati-hati pada saat melakukan proses dekompresi.

Dalam melakukan proses penulisan hasil kompresi ini maka algoritma yang digunakan adalah :

1. *Siapkan informasi-informasi header file dan tulis ke dalam file hasil kompresi;*
2. *Baca file asal byte demi byte dan cari kode yang cocok dari tabel kode Huffman yang terlah dibuat (berdasarkan prosedur Encrypt);*
3. *Apabila deretan kode sudah mencapai 8-bit maka dilakukan penulisan data 8-bit tersebut, jika tidak lakukan langkah 2;*
4. *Ulangi proses 2 sampai 3 hingga file asal telah terbaca semua.*

Dalam penulisan file hasil kompresi ini agar proses dapat berlangsung dengan cepat maka pembacaan file asal dan penulisan file hasil dilakukan secara blok.

PROGRAM DEKOMPRESI

Dalam program dekompresi ini proses dibagi menjadi tiga bagian :

- Proses Membaca Berkas.
- Proses Pembuatan Pohon Huffman.
- Proses Penulisan Hasil Dekompresi.

1. Proses Membaca Berkas

Proses membaca berkas/file pada program dekomprefsi ini berbeda dengan proses pembacaan berkas/file pada program kompresi sebelumnya. Pembacaan berkas/file hasil kompresi dalam program dekompresi ini harus mengikuti aturan yang digunakan dalam proses penulisan file hasil kompresi pada program kompresi di atas.

Karena berkas/file hasil kompresi terdiri dari header dan badan file maka pembacaannya juga disesuaikan. Yaitu pertama-tama membaca header untuk memperoleh informasi besar file asli, informasi banyaknya simbul, informasi simbul dan informasi probabilitas tiap-tiap simbul. Berdasarkan hasil pembacaan tersebut maka baru dapat dilakukan proses selanjutnya.

2. Proses Pembuatan Pohon Huffman

Berdasarkan informasi-informasi yang diperoleh pada proses pembacaan berkas, maka dapat disusun Pohon Huffman untuk keperluan dekompresi. Prosedur-prosedur yang digunakan sama dengan proses pembuatan pohon Huffman pada Program Kompresi. Yaitu :

1. BUAT_LIST_PELUANG
2. CARI_PELUANG_TERKECIL
3. SUSUN_POHON

Dalam proses pembuatan pohon Huffman ini tidak ada prosedur Decrypt sebagai lawan dari prosedur Encrypt dari program kompresi. Hal ini disebabkan prosedur Decrypt akan langsung digunakan dalam prosedur dekompresi nanti. Jadi ada sedikit perbedaan dalam proses pembuatan pohon Huffman dalam program kompresi dengan program dekompresi.

3. Proses Penulisan Hasil Dekompresi

Setelah selesai melakukan proses pertama dan kedua (baca header dan membuat pohon Huffman) maka dilakukan proses berikutnya yaitu proses dekompresi dan penulisan hasilnya.

Dalam proses dekompresi dan penulisan hasilnya ini akan selalu memanggil prosedur Decrypt untuk mencari simbul yang sesuai dengan aliran bit-bit dari pembacaan file hasil kompresi. Bit-bit tersebut akan digunakan sebagai petunjuk dalam mengunjungi pohon Huffman sampai ditemukan sebuah simbul.

Seperi dalam proses pembuatan kode Huffman yaitu kunjungan kekiri akan menghasil kode '0' dan kunjungan ke kanan akan menghasilkan kode '1', maka dalam proses Decrypt ini juga sama. Yaitu jika memperoleh bit '0' maka harus mengunjungi cabang kiri dan jika diperoleh bit '1' akan mengunjungi cabang kanan Kunjungan akan berakhir pada saat tidak ada cabang kiri ataupun cabang kanan, yang berarti telah ditemui sebuah simbul. Simbul tersebut kemudian ditampung dalam array terlebih dahulu. Hal ini dimaksudkan untuk mempercepat proses, karena penulisan file dilakukan secara blok.

Algoritma yang digunakan dalam proses Decrypt ini adalah :

1. *Inisialisai pencacah besar file;*
2. *Baca file hasil kompresi byte demi byte;*
3. *Berdasarkan bit-bit dalam byte tersebut kunjungi pohon Huffman dengan prosedur Decrypt, sampai tidak ada cabang kiri maupun cabang kanan.*

Tulis simbul yang diperoleh dan naikkan pencacah besar file dengan 1;

4. *Lakukan langkah 1 dan 2 sampai blok penulisan file terpenuhi, lakukan penulisan file hasil dekompresi;*
5. *Ulangi langkah 1 sampai 4 hingga pencacah besar file menunjukkan angka sama dengan informasi besar file.*
6. *Selesai*

B. ALGORITMA ELIAS ENCODER

Seperti telah dijelaskan pada Bagian I, bahwa pengkodean Elias menggunakan rumus logaritma terhadap nomor urut suatu simbul. Sehingga perbedaan yang nyata dari program kompresi dan dekompresi dengan algoritma Elias Encoder ini adalah proses sorting simbul berdasarkan probabilitasnya dan

cara pembuatan kode untuk tiap-tiap simbul. Proses lainnya seperti : membaca file untuk mengetahui macam simbul dan probabilitasnya (dari file yang akan dikompres) serta proses penulisan hasil kompresi menjadi file terkompres tidak berbeda dengan prosedur-prosedur yang digunakan dalam program kompresi dengan algoritma pohon Huffman.

Suatu hal penting yang perlu diperhatikan dalam program ini adalah : Karena dalam bahasa pemrograman Pascal hanya memiliki fungsi aritmatika logaritma natural, sedangkan fungsi logaritma yang diperlukan dalam pengkodean Elias menggunakan logaritma berbasis 2 maka perlu dibuatkan fungsi baru untuk keperluan tersebut. Fungsi yang dimaksud diujudkan dalam program fungsi :

```
function log(x:integer):string;
```

Fungsi tersebut akan mengolah data integer (nomor urut dari simbul) menjadi deretan bit ‘0’ sebagai prefiks sekaligus menambahkan sufiks pengkodean Elias.

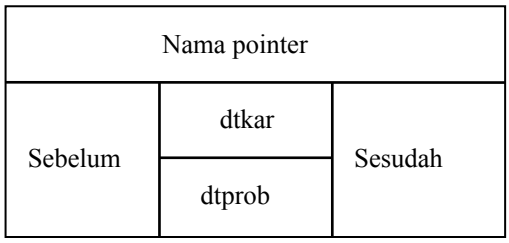
PROGRAM KOMPRESI

Secara garis besar proses kompresi dibagi menjadi empat bagian proses yaitu :

- Proses Membaca Berkas
- Proses Mengurutkan Simbul Berdasarkan Probabilitasnya
- Proses Membuat Kode Elias
- Proses Penulisan Hasil Kompresi

1. Proses Membaca Berkas/File

Algoritma proses pertama dan proses keempat hampir sama dengan algoritma yang digunakan pada program kompresi Pohon Huffman. Untuk proses pertama perbedaan hanya terletak pada struktur data yang digunakan. Karena pointer yang digunakan hanya untuk membentuk senarai berantai linear (tidak berbentuk pohon) maka bentuk pointernya adalah sebagai berikut :



Setelah melakukan pendataan macam simbul dan probabilitasnya maka dilakukan proses optimasi tabel yaitu dengan menggunakan prosedur SELEKSI_SIMBUL. Prosedur ini akan menghapus karakter yang probabilitasnya sama dengan nol dari tabel simbul dan probabilitas.

Untuk keperluan proses sorting, maka simbul dan probabilitas dimasukkan dalam struktur data pointer dengan memanggil prosedur MASUK_HEAP.

2. Proses Mengurutkan Simbul Berdasarkan Probabilitasnya

Proses ini berfungsi untuk mengurutkan simbul berdasarkan probabilitasnya sehingga simbul yang memiliki probabilitas besar akan memiliki nomor urut paling kecil sedangkan simbul dengan probabilitas paling kecil akan memiliki nomor urut paling besar.

Banyak algoritma sorting yang dapat digunakan dalam pengurutan simbul ini. Dalam program ini digunakan algoritma Chain Record Sort yang memiliki kecepatan tinggi dalam melakukan sorting meskipun memori yang dipakai menjadi lebih besar. Dengan metode Chain Record Sort ini masing-masing elemen yang akan diurutkan tetap pada posisinya masing-masing (tidak akan dipindahkan). Dengan pointer yang ada maka cukup merubah arah penunjuk sesuai dengan peringkat elemen yang kita bandingkan. Dengan demikian letak masing-masing elemen tidak berubah. Proses sorting ini dilakukan dengan memanggil prosedur URUTKAN. Berdasarkan senarai berantai yang telah dibentuk oleh prosedur MASUK_HEAP maka alogritma sorting adalah sebagai berikut :

1. Inisialisasi :

Tentukan Akhir :=Awal;

2. Tentukan pointer Kepala disebelah kanan pointer Akhir :

Kepala := Awal^.Sesudah;

3. *Tentukani : Awal^.Sesudah := Nil;*
4. *Jika Kepala sama dengan Nil maka proses SELESAI, jika tidak sama dengan Nil maka lakukan langkah berikut ini:*
5. *Tentukan J:=Kepala^.Sesudah;*
6. *Jika Kepala^.dtprob <= Akhir^.dtprob maka lakukan langkah 7, jika tidak lakukan langkah 8 dan selanjutnya;*
7. *Letakkan letakkan pointer Kepala disebelah kanan pointer Akhir dan ;*

- jadikan pointer Akhir tadi menjadi pointer Awal;

- jadikan pointer Kepala menjadi pointer Akhir;

- Ulangi langkah 2 sampai 6.

8. *J:= Akhir^.sebelum;*

- Selama Kepala^.dtprob >= J^.dtprob dan J <> Nil maka tentukan: J:= J^.Sebelum;

9. *Jika J= Nil maka Kepala merupakan pointer awal paling kiri;*
10. *Jika J <> Nil maka letakkan J disebelah kiri Kepala;*
11. *Ulangi langkah 4 sampai dengan langkah 11.*

Setelah dilakukan sorting maka bentuk senarai berantai menjadi urut dimana simbul dengan probabilitas paling besar akan ditunjuk paling awal dan simbul dengan probabilitas paling kecil akan ditunjuk paling akhir dalam senarai tersebut. Dengan demikian dapat kita buat sebuah array yang berisi urutan hasil sorting tersebut. Nomor array inilah yang akan kita gunakan sebagai bahan pembuatan kode Elias.

3. Proses Pembuatan Kode Elias

Proses pengkodean Elias dilakukan dengan memanggil prosedur ELIAS dimana terdapat fungsi **log** yang dapat melakukan operasi aritmatik logaritma berbasis dua dari sebuah nilai integer sekaligus mengubah nilai integer tersebut menjadi kode elias. Algoritma untuk pengkodean Elias adalah :

1. *Tentukan x:= nomor simbul pertama;*
2. *Buat Prefiks berupa deretan bit '0' sebanyak $\log_2 (x+1)$, tampung ke variabel awal;*

3. *Buat Sufiks yang merupakan bilangan biner dari (x+1), tampung dalam variabel akhir;*
4. *Gabungkan menjadi kode Elias :*
5. *Kode[x] := awal + akhir;*
6. *Ulangi langkah 2 sampai 5 untuk nilai x berikutnya sebanyak jumlah_simbul;*
7. *Selesai.*

Setelah proses ini selesai maka dilakukan pembacaan file yang akan dikompres byte demi byte dan diubah simbulnya sesuai dengan kode yang sudah dibuat.

4. Proses Penulisan Hasil Kompresi.

Prosedur ini menyediakan dua fungsi utama yang berkaitan dengan penulisan file hasil kompresi. Yaitu penulisan **header** dan **badan** file.

4.1 Header

Header file hasil kompresi terdiri dari empat bagian yaitu :

- *informasi jumlah byte file asli*
- *informasi banyak simbul yang terdapat dalam file asli*
- *informasi simbul-simbul yang dalam file asli*

Informasi besarnya file asal digunakan sebagai ukuran pencacah pada saat dilakukan proses dekompresi file hasil kompresi tersebut. Dengan adanya informasi ini maka program akan mudah dalam menentukan kapan harus menyelesaikan pembacaan data untuk dekompresi file hasil kompresi.

Informasi jumlah simbul digunakan untuk menentukan pencacah yang berhubungan dengan daftar simbul dan daftar probabilitas agar tidak terjadi kesalahan akses data dari file hasil kompresi. Atau dengan kata lain informasi tersebut digunakan untuk menentukan besarnya array/pointer yang mencacat simbul.

Informasi simbul digunakan pembuatan kode Elias kembali dalam proses dekompresi.

Dalam header file kompresi Elias ini tidak perlu mencantumkan informasi peluang tiap simbul, karena urutan simbul tersebut sudah dapat digunakan untuk membuat kode Elias dalam proses Dekompresi nantinya.

4.2 Badan File

Badan file ini sama dengan badan file program kompresi Huffman. Yaitu menyimpan byte demi byte hasil pengkodean tiap simbul. Setiap byte dalam file ini merupakan potongan (karena diujudkan dalam byte berarti sebanyak 8 bit) dari deretan simbul file asal yang telah diubah menjadi deretan kode. Karena setiap kode dari masing-masing simbul tidak sama panjangnya maka perlu dilakukan kerja keras dan hati-hati pada saat melakukan proses dekompresi.

Dalam melakukan proses penulisan hasil kompresi ini maka algoritma yang digunakan adalah :

1. *Siapkan informasi-informasi header file dan tulis ke dalam file hasil kompresi;*
2. *Baca file asal byte demi byte dan cari kode yang cocok dari tabel kode Elias yang terlah dibuat (berdasarkan prosedur Elias);*
3. *Apabila deretan kode sudah mencapai 8-bit maka dilakukan penulisan data 8-bit tersebut, jika tidak lakukan langkah 2;*
4. *Ulangi proses 2 sampai 3 hingga file asal telah terbaca semua.*

Dalam penulisan file hasil kompresi ini agar proses dapat berlangsung dengan cepat maka pembacaan file asal dan penulisan file hasil dilakukan secara blok.

PROGRAM DEKOMPRESI

Dalam program dekompresi ini proses dibagi menjadi tiga bagian :

- Proses Membaca Berkas.
- Proses Pembuatan Kode Elias.
- Proses Penulisan Hasil Dekompresi.

1. Proses Membaca Berkas

Proses membaca berkas/file pada program dekomprefsi Elias ini hampir sama dengan program Huffman dimana pembacaan harus mengikuti aturan sesuai dengan bentuk file yang diproses pada saat penulisan hasil kompresi sebelumnya. Yaitu pertama-tama membaca header untuk memperoleh informasi

besar file asli, informasi banyaknya simbul dan informasi simbul. Berdasarkan hasil pembacaan tersebut maka baru dapat dilakukan proses selanjutnya.

2. Proses Pembuatan Kode Elias

Berdasarkan informasi-informasi yang diperoleh pada proses pembacaan berkas, maka dapat disusun kode Elias seperti pada program kompresi. Karena simbul-simbul yang disimpan sudah urut dari probabilitas terbesar sampai probabilitas terkecil, maka nomor urutan tersebut langsung dapat dipakai untuk membuat kode Eliasnya.

3. Proses Penulisan Hasil Dekompresi

Setelah selesai melakukan proses pertama dan kedua (baca header dan membuat kode Elias) maka dilakukan proses berikutnya yaitu proses dekompresi dan penulisan hasilnya.

Dalam proses dekompresi proses pencarian kode yang cocok dengan pembacaan byte demi byte dari badan file akan lebih mudah dilakukan, karena kode tersebut sebenarnya dapat ditebak berdasarkan prefiknya dan nilai prefiks ini akan menentukan panjangnya deretan sufik dari kode tersebut. Sehingga proses perulangan (loop) menjadi lebih sedikit.

Simbul-simbul yang diperoleh dari hasil dekode tersebut kemudian ditampung dalam array terlebih dahulu. Hal ini dimaksudkan untuk mempercepat proses, karena penulisan file dilakukan secara blok.

C. PELENGKAP PROGRAM

Baik program Kompresi/Dekompresi Huffman maupun Elias dilengkapi prosedur tambahan seperti peringatan bahwa nama file yang akan dikompres tidak ada dan nama file yang akan dikompres ternyata sama dengan nama file hasil kompresi yang semuanya dikemas dalam prosedur Header yang juga merupakan tempat komunikasi dengan pemakai.

BAGIAN III : KESIMPULAN

Dalam tugas ini telah berhasil dibuat program kompresi dengan algoritma Pengkodean Pohon Huffman dan algoritma Elias Encoder. Hasil yang diperoleh terhadap kinerja dan sifat-sifat kedua program tersebut adalah :

1. Kedua program kompresi tersebut hampir memiliki prosesntase kompresi yang sama.
2. Algoritma Elias Encoder memiliki kelebihan dibanding algoritma Pohon Huffman yaitu konfigurasi header file hasil kompresi lebih sederhana (tidak mencantumkan probabilitas tiap simbul); sederhana dalam pengkodean dan dalam proses pencarian simbul berdasarkan kode dapat ditebak secara cepat sehingga mempercepat proseas dekompresi.
3. Program kompresi dengan algoritma Pohon Huffman kadang-kadang dalam membuat kode tidak efektif yaitu menghasilkan kode terpendek lebih dari 2 atau 3 bit, sedangkan untuk algoritma Elias pasti ada kode yang panjangnya satu bit.