ANALISA KODE HUFFMAN UNTUK KOMPRESI DATA TEKS

Timothy John Pattiasina, ST., M.Kom.*

ABSTRAK

Huffman Algorithm adalah sa1ah satu algoritma kompresi tertua yang disusun oleh David Huffman pada tahun 1952. Algoritma tersebut digunakan untuk membuat kompresi jenis *loss compression*, yaitu pemampatan data dimana tidak satu *byte* pun hilang sehingga data tersebut utuh dan disimpan sesuai dengan aslinya. Prinsip kerja algoritma Huffman adalah mengkodekan setiap karakter ke dalam representasi bit. Representasi bit untuk setiap karakter berbeda satu sama lain berdasarkan frekuensi kemunculan karakter. Semakin sering karakter tersebut muncul, maka semakin pendek panjang representasi bit nya. Sebaliknya bila semakin jarang frekuensi karakter muncul, maka semakin panjang representasi bit untuk karakter tersebut Teknik kompresi algoritma Huffman mampu memberikan penghematan pemakaian memori sampai 30%. Algoritma Huffman mempunyai kompleksitas 0 (n log n) untuk himpunan dengan n karakter.

Kata Kunci: *Huffman Algorithm, Lossy Compression¸ Representasi Bit*

1. PENDAHULUAN

Kompresi ialah proses pengubahan sekumpulan data menjadi suatu bentuk kode untuk menghemat kebutuhan tempat penyimpanan dan waktu untuk transmisi data. Saat ini terdapat berbagai tipe algoritma kompresi, diantaranya Huffman, LIFO, LZHUF, LZ77 dan variannya, (LZ78, LZW, GZIP), *Dynamic Markov Compression* (DMC), *Block-Sorting Lossless*, *Run- Length*, *Shannon-Fano*, *Arithmetic*, PPM (*Prediction by Partial Matching*), *Burrows-Wheeler Block Sorting*, dan *Half Byte*.

1.1. Latar Belakang

Teks adalah kumpulan dari karakter –karakter atau *string* yang menjadi satu kesatun. Teks yang memuat banyak karakter didalamnya selalu menimbulkan masalah pada media penyimpanan dan kecepatan waktu pada saat transmisi data. Media penyimpanan yang terbatas, membuat semua orang mencoba berpikir untuk menemukan sebuah cara yang dapat digunakan untuk mengompres teks.

Algoritma Huffman, yang dibuat oleh seorang mahasiswa MIT bernama David Huffman, merupakan salah satu metode paling lama dan paling terkenal dalam kompresi teks. 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 pendek dan karakter yang jarang muncul dikodekan dengan rangkaian bit yang lebih panjang.

1.2. Perumusan Masalah

Berdasarkan latar belakang diatas, maka dapat ditarik sutu rumusan masalah sebagai berikut:

1. Bagaimanakah cara kerja kode huffman dalam kompresi teks?

* Staf Pengajar Program Studi S1-Teknik Informatika IKADO

2. Bagaimanakah cara kerja kode huffman dalam mencari solusi dari kompresi teks?

1.3. Tujuan dan Manfaat Penelitian

Penelitian yang dilakukan ini memiliki beberapa tujuan, diantaranya:

- Untuk mengetahui cara kerja kode Huffman dalam melakukan kompresi data teks
- Untuk mengetahui cara kerja kode Huffman dalam mencari solusi dari kompresi data teks.

Sedangkan manfaat dari penelitian yang dilakukan ini adalah:

- Mengetahui sistematika kode Huffman dalam kompresi data teks.
- Menambah wawasan baru tentang teknik dalam kompresi data teks.

2. TINJAUAN PUSTAKA

Dalam penelitian ini dijabarkan mengenai tinjauan pustaka yang menjadi dasar dilakukannya penelitian ini. Dimana seluruh tinjauan pustaka mencakup semua hal terkait dengan penelitian yang dilakukan.

2.1. Encoding dan Decoding

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 *di-encoding*

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. Langkah-langkah *men-decoding* suatu *string* biner dengan menggunakan pohon Huffman adalah sebagai berikut:

1. Baca sebuah *bit* dari *string* biner.
2. Mulai dari akar
3. Untuk setiap *bit* pada langkah 1, lakukan traversal pada cabang yang bersesuaian.
4. Ulangi langkah 1, 2 dan 3 sampai bertemu daun. Kodekan rangkaian *bit* yang telah dibaca dengan karakter di daun.
5. Ulangi dari langkah 1 sampai semua *bit* di dalam *string* habis

2.2. Huffman Tree

Kode Huffman digunakan secara luas dan sangat efektif untuk kompresi data. Bisa menghemat 20% - 90% dari ukuran semula, tergantung tipe karakter yang akan dikompresi. Algoritma huffman menggunakan tabel yang menyimpan frekuensi kemunculan dari masing-masing simbol yang digunakan dalam *file* tersebut dan kemudian mengkodekannya dalam.bentuk biner.

Pertama-tama buat tabel frekuensi dari semua simbol atau karakter yang muncul dalam suatu *file* teks. Kemudian diurutkan mulai dari simbol dengan frekuensi paling sedikit sampai simbol dengan frekuensi paling banyak. Pembentukan *Huffman tree* dimulai dari dua simbol paling depan untuk dijadikan anak kiri dan anak kanan dari *tree* yang terbentuk dan frekuensinya dijumlahkan. Setelah itu langkah berikutnya diurutkan kembali berdasarkan frekuensi yang baru. Demikian dilakukan terus menerus sampai semua simbol terbentuk menjadi *tree*.

Dengan algoritma di atas maka terbentuklah suatu *tree* yang menyimpan simbol - simbol sedemikian rupa sehingga simbol dengan frekuensi kemunculan yang paling banyak akan ditempatkan di kedalaman yang paling dangkal, sedangkan simbol-simbol dengan frekuensi kemunculan yang paling sedikit akan ditempatkan di kedalaman yang paling dalam. Setelah proses dari pembuatan atau pembentukan *tree* selesai, maka *tree* siap digunakan untuk langkah selanjutnya yaitu membentuk kode.

2.3. Kompresi Data

Kompresi data adalah proses *encoding* suatu informasi dalam data yang menjadi rangkaian *bit* yang lebih pendek daripada rangkaian *bit* yang diperlukan dari data tersebut sebelum dikompresi dengan menggunakan metode *encoding* tertentu.

Keuntungan menggunakan kompresi data adalah memiliki penghematan tempat pada media penyimpanan (misalnya *Hard Disk* dan DVD *(Digital Versatile Disc))*, dan penghematan *bandwidth* (lebar pita) pada pengiriman data. Namun kompresi data juga memiliki sisi negatif. Bila data yang terkompresi ingin dibaca, perlu dilakukan proses dekompresi terlebih dahulu, pada pengiriman sebuah data, penerima data harus mengerti proses *encoding* **dalam** data terkompresi yang dikirim (memiliki perangkat lunak *(software)* yang dapat mendekompresinya).

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

a. Metode Statik
   Metode statik adalah metode yang menggunakan peta kode yang selalu sama. Metode ini membutuhkan dua fase *(twopass).* Fase yang pertama adalah untuk menghitung probabilitas kemunculan tiap simbol/karakter dan menentukan peta kodenya, dan fase yang kedua adalah untuk mengubah pesan menjadi kumpulan-kumpulan kode yang akan ditransmisikan. Contoh: algoritma Huffman statik.
b. Metode Dinamik (adaptif)
   Metode dinamik (adaptif) adalah metode yang menggunakan peta kode yang dapat berubah dari waktu ke waktu. Metode ini disebut adaptif karena peta kode mampu beradaptasi terhadap perubahan karakteristik isi *file* selama proses kompresi berlangsung. Metode ini bersifat *onepass*, karena hanya diperlukan satu kali pembacaan terhadap isi *file.* Contoh: algoritma LZW dan DMC.

2.4. Pengkodean Huffman Dinamis

Pengkodean Huffman dinamis, atau disebut juga pengkodean Huffman adaptif, pertama kali disusun oleh Faller dan Gallager. Kemudian, D. E. Knuth melakukan peningkatan terhadap algoritma tersebut dan menghasilkan algoritma baru yang disebut algoritma FGK. Versi terbaru dari pengkodean Huffman dinamis dideskripsikan oleh Vitter pada tahun 1987, yang disebut sebagai Algoritma V.

2.4.1. Proses *Encoding* dan *Decoding*

Proses *encoding* dan *decoding* dalam pengkodean Huffman dinamis menginisialisasi pohon Huffman dengan sebuah pohon bersimpul tunggal yang berkorespondensi dengan sebuah karakter artifisial, ditunjukkan dengan simbol ART. Bobot dari simpul tunggal ini adalah 1. berikut adalah proses dari *encoding* dan *decoding:*

1. Proses *encoding:*
   Setiap pembacaan simbol a dari teks sumber, *codeword*-nya dalam pohon dikirimkan. Namun, hal ini hanya dilakukan jika a telah muncul sebelumnya. Jika tidak, kode dari ART dikirimkan diikuti oleh *codeword* asli dari a. Kemudian, pohon tersebut dimodifikasi sebagai berikut:
   Jika a belum pernah muncul sebelumnya, sebuah simpul *internal* dibuat dan kedua simpul anaknya berisi a dan ART. Kemudian, pohon tersebut diperbaharui untuk mendapat pohon Huffman dari teks yang sudah dibaca.
   Input.
   Alphabet A = {a[1], a[2], ..., a[n]}memakai simbol alphabet dengan ukuran n. Set C = {c[1], c[2], ..., c[n]} dengan konfigurasi nilai simbol, c[i] = nilai (a[i]), 1 <= i <= n.

   Output.
   Code H(A,C) = {h[1], h[2], ..., h[n]} dengan konfigurasi (biner) *codewords*, dimana h[i] adalah *codeword* dari a[i], 1 <= i <= n.

   Goal.
   Let S(H) = sum (c[i] * length (h[i])) (1 <= i <= n) akan mengembang dari batas awal kode H. Harus: S(H) <= S(T) untuk semua kode T(A,C).

   Contoh:

   Sample-1

Input	Alphabet	a	b	c	d	e	f	g	h	i	
	Costs	**10**	**15**	**5**	**15**	**20**	**5**	**15**	**30**	**5**	
Output	Codewords	**001**	**010**	**00000**	**011**	**101**	**00001**	**100**	**11**	**0001**	
	Weighted path length	**10*3**	**15*3**	**5*5**	**15*3**	**20*3**	**5*5**	**15*3**	**30*2**	**5*4**	**= 355**

   Sample-2

Input	Alphabet	a	b	c	d	e	f	g	h	i	
	Costs	**3**	**21**	**2**	**1**	**8**	**34**	**1**	**13**	**5**	
Output	Codewords	**000001**	**01**	**0000001**	**00000000**	**0001**	**1**	**00000001**	**001**	**00001**	
	Weighted path length	**3*6**	**21*2**	**2*7**	**1*8**	**8*4**	**34*1**	**1*8**	**13*3**	**5*5**	**= 220**

2. Proses *Decoding*
   Pada waktu *decoding*, teks hasil pemampatan *di-parse* dengan menggunakan pohon pengkodean. Simpul saat ini diinisialisasi oleh akar seperti algoritma *encoding*, kemudian pohon tersebut tumbuh secara simetris. Setiap sebuah 0 dibaca dari berkas hasil kompresi, pohon tumbuh mengikuti ke kiri Jika 1 yang dibaca, pohon tumbuh ke kanan. Ketika simpul terkini adalah daun,

simpul tersebut terasosiasi dengan simbol yang ditulis di *output file* dan pohon diperbaharui sehingga menjadi persis sama seperti seperti pohon pada proses *encoding*.

2.4.2. Proses Updating

Selama proses *encoding* dan *decoding*, pohon sementara harus diperbaharui untuk mendapatkan frekuensi simbol yang benar. Ketika karakter baru diketahui bobotnya, bobot daun yang berasosiasi dengan karakter tersebut bertambah satu, dan bobot dari simpul-simpul diatasnya dimodifikasi. Kebenaran pohon yang terbentuk dilihat dari *siblings property* pohon tersebut.

Proses *updating* bekerja sebagai berikut:

1. Bobot dari daun n yang berkorespondensi dengan a ditambah 1
2. Jika *siblings property*-nya tidak sesuai lagi, maka simpul n ditukar dengan simpul terdekat, m (m<n) sehingga bobot (m) < bobot (n).
3. Simpul-simpul tersusun menurun sesuai bobotnya.

2.4.3. Langkah-Langkah Pengkodean

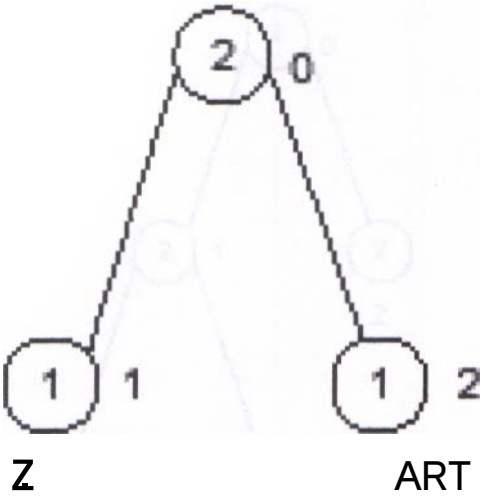
Langkah-langkah pengkodean pada algoritma pengkodean Huffman dinamis adalah sebagai berikut:

y = ZBZH {teks masukan)

inisialisasi pohon:

Gambar Inisialisasi pohon

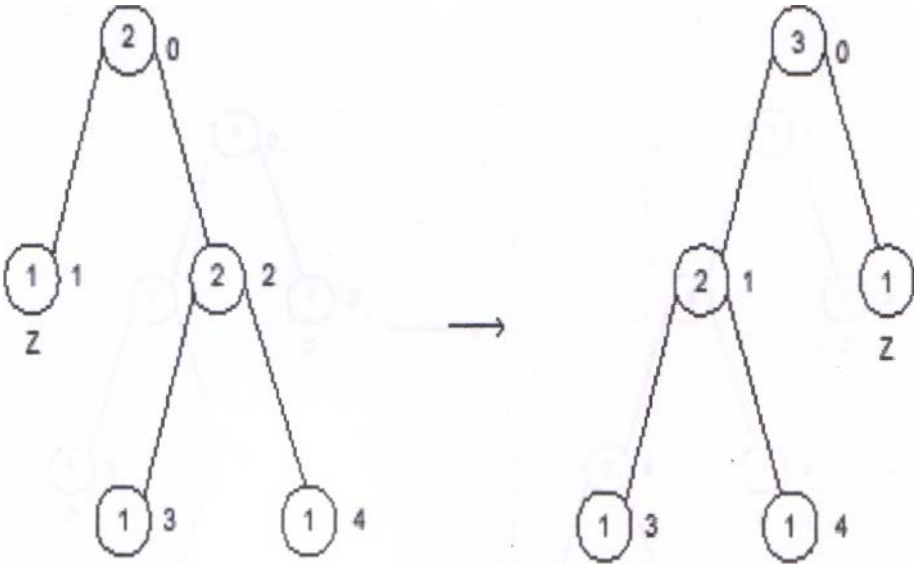
next = Z {simbol selanjutnya Z}



Gambar Simbol Z dari inisialisasi pohon

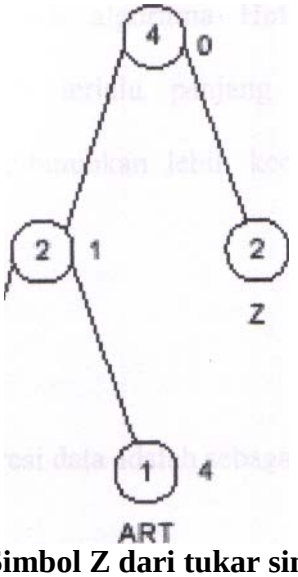
next = B

tukarSimpul(1,2)



Gambar Tukar simpul (1,2)

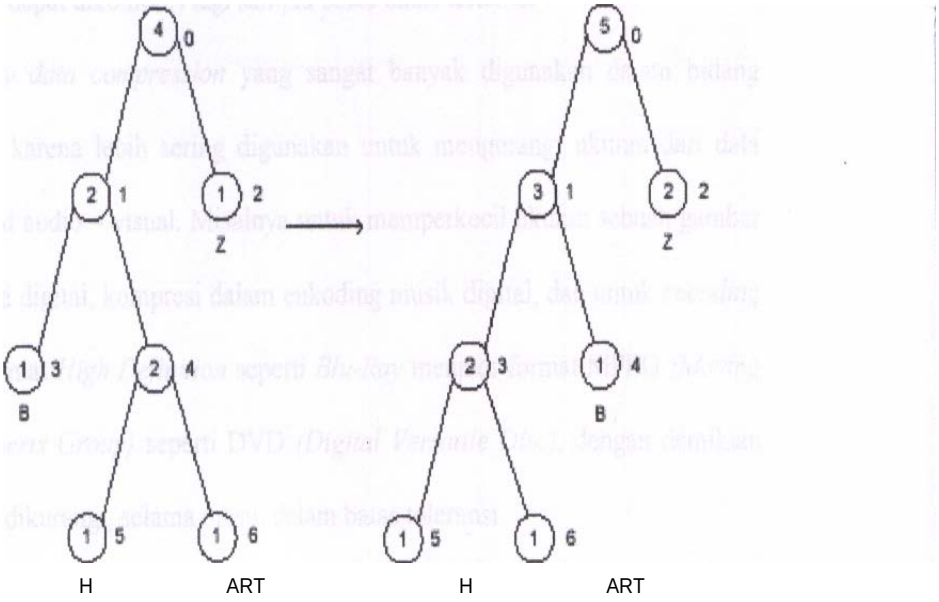
next = Z



Gambar Simbol Z dari tukar simpul (1,2)

next = h

tukarSimpul(3,4)



Gambar Tukar Simpul (3,4)

Dari hasil pengkodean di atas, panjang kode untuk tiap karakter tidak lebih sedikit daripada yang dihasilkan oleh algoritma Huffman statis. Hal ini disebabkan teks yang dibaca tidak terlalu panjang. Namun dari proses pengkodeannya, jelas waktu yang dibutuhkan lebih kecil, karena pembacaan berkas hanya dilakukan satu kali.

3. METODE ANALISA

Pada bab ini, akan dijabarkan metode yang digunakan di dalam penelitian ini, serta prosedur-prosedur dalam pengumpulan dan analisa data.

3.1. Metode Analisis

Penelitian tentang teknik dalam kompresi data ini difokuskan pada data teks yang dilakukan dengan mengumpulkan beberapa *paper* dari internet dengan cara mengunduh *(download) paper* tersebut. Kumpulan-kumpulan *paper* yang diambil dari internet kemudian dipelajari, serta dianalisa dan dirangkum sebagai bahan acuan dari penelitian ini.

3.2. Prosedur Pengumpulan Data

Penelitian ini menggunakan prosedur pengumpulan data yang dijabarkan pada poin-poin di bawah ini:

1. Mencari artikel-artikel yang membahas segala sesuatu tentang teknik kompresi data teks terutama yang membahas tentang metode algoritma huffman dalam mengkompresi data teks. Artikel-artikel ini dicari pada internet, karena algoritma ini tidak dijabarkan dalam artikel-artikel majalah dan sebagainya.
2. Mencari buku yang membahas tentang teknik kompresi data teks khususnya yang membahas tentang metode algoritma huffman dalam menglcompresi data teks. Buku yang dicari berupa *e-book* yang berformat PDF tetapi tidak mengurangi isi dari *e-book* dengan isi dari buku aslinya. Hal ini dilakukan

untuk mempermudah dalam menganalisa tentang teknik kompresi data teks dengan metode algoritma Huffman.

3. Bertanya kepada pembuat artikel-artikel tersebut melalui forum tanya jawab. Forum tanya jawab dilakukan di Internet melalui situs dimana kita mengunduh *(Men-download)* artikel tersebut.

3.3. Prosedur Analisis Data

Setelah melakukan prosedur pengumpulan data, yang dilakukan selanjutnya adalah menganalisa data. Prosedur analisa data yang diperoleh dari pengumpulan data, dapat dijabarkan pada poin-poin di bawah ini:

a. Mempelajari dan memahami isi dari artikel-artikel yang diperoleh, untuk kemudian merangkumnya sebagai suatu teori yang mendukung penelitian ini.
b. Mempelajari dan memahami teori-teori teknik untuk mengkompresi data teks terutama dengan metode algoritma huffman pada buku yang kemudian merupakan suatu landasan teori untuk pelaksanaan penelitian.
c. Memahami jawaban-jawaban yang diberikan oleh pembuat artikel pada forum tanya jawab di internet. Jawaban-jawaban dari pembuat artikel ini sangat membantu dalam penelitian ini.

4. ANALISA ALGORITMA

Pada bab analisa algoritma ini menjelaskan mengenai algoritma huffman yang dipakai dalam teknik kompresi data teks. Sehingga dapat mengetahui cara kerja dari kompresi data dengan melihat dari analisa algoritmanya.

4.1. Algoritma Huffman

Pemikiran algoritma ini adalah bahwa setiap karakter ASCII *(american standard for information interchange)* 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 di beri 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 diatas 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 diusahakan lebih kecil jumlah *bit-nya* dibandingkan dengan karakter yang jarang muncul. Pada contoh di atas karakter A lebih sering muncul (3 kali), jadi kodenya dibuat lebih kecil jumlah *bit-nya* dibanding karakter lain.

4.1.1. Pembentukan Pohon Huffman

Kode huffman pada dasarnya merupakan kode prefiks *(prefix code).* Kode prefiks adalah himpunan yang berisi sekumpulan kode biner, dimana pada kode prefiks 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 dalam pembentukan pohon biner adalah sebagai berikut:

1. Baca semua karakter di dalam teks untuk menghitung frekuensi kemunculan setiap karakter. Setiap karakter penyusun teks dinyatakan sebagai pohon bersimpul tunggal. Setiap simpul di-assign dengan frekuensi kemunculan karakter tersebut.
2. Terapkan strategi algoritma greedy. Gabungkan dua buah pohon yang mempunyai frekuensi terkecil pada sebuah akar. Setelah digabungkan akar tersebut akan mempunyai frekuensi yang merupakan jumlah dari frekuensi dua buah pohon-pohon penyusunnya.
3. Ulangi langkah 2 sampai hanya tersisa satu buah pohon Huffman. Agar pemilihan dua pohon yang akan digabungkan berlangsung cepat, maka semua yang ada selalu terurut vertikal naik berdasarkan frekuensi.

4.1.2. Proses Encoding

Langkah-langkah untuk *menencoding* suatu string biner adalah sebagai berikut:

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

4.1.3. Proses Decoding

Langkah-langkah untuk *menencoding* suatu string biner adalah sebagai berikut:

a. Baca sebuah bit dari string biner.
b. Mulai dari akar.
c. Untuk setiap .hit pada langkah 1, lakukan tranversal pada cabang yang bersesuaian
d. Ulangi langkah 1, 2 dan 3 sampai bertemu daun. Kodekan rangkaian bit yang telah dibaca dengan karakter di daun.
e. Ulangi dari langkah 1 sampai semua bit didalam string habis.

Cara yang kedua adalah dengan menggunakan tabel kode huffman.Untuk proses pengembalian ke file aslinya, kita harus mengacu kembali kepada kode Huffman yang telah dihasilkan.

4.1.4. Kompleksitas Algoritma Huffman

Algoritma Huffman mempunyai kompleksitas waktu 0(n *log* n), karena algoritma Huffman dalam melakukan sekali proses iterasi pada saat penggabungan dua buah pohon yang mempunyai frekuensi terkecil pada sebuah akar membutuhkan waktu *0(log* n), dan proses itu dilakukan berkali-kali sampai hanya tersisa satu buah pohon Huffman itu berarti dilakukan sebanyak n kali..

4.2. Algoritma Greedy

Algoritma greedy adalah salah satu algoritma yang digunakan untuk menyelesaikan persoalan optimasi, artinya persoalan yang menuntut pencarian solusi *optimum*, baik masalah maksimasi *(maximization)* atau minimasi *(minimization)*. Algoritma greedy adalah algoritma yang memecahkan masalah langkah per langkah, pada setiap langkahnya algoritma greedy melakukan hal-hal sebagai berikut :

1. Mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip *"take what you can get"*).
2. Berharap bahwa dengan memilih *optimum local* pada setiap langkah akan berakhir dengan *optimum global.*

5. UJI COBA

Pada bab uji coba ini dilakukan pengujiaan algoritma Huffman dalam melakukan kompresi data teks. Dan juga pengujian kompresi dari beberapa *file* dengan variasi karakter.

5.1. Pengujian Algoritma Huffman

Pada pengujian, akan dilakukan *encoding* sebuah teks yang berisi 100.000 *string*, diantaranya 45.000 karakter 'g', 13.000 karakter 'o', 12.000 karakter `p', 16.000 karakter 'h', 9.000 karakter `e', dan 5.000 karakter 'r' dengan menggunakan 3 cara, yaitu dengan menggunakan kode ASCII , kode *3-hit* dan kode Huffman. Setelah itu ketiga kode tersebut akan dibandingkan satu dengan yang lain.

- Hasil Kode Huffman

Tabel Kode Huffman untuk karakter "gopher"

Karakter	Frekeuensi	Peluang	Kode Huffman
g	45.000	45/100	0
o	13.000	13/100	101
p	12.000	12/100	100
h	16.000	16/100	111
e	9.000	9/100	1101
r	5.000	5/100	1100

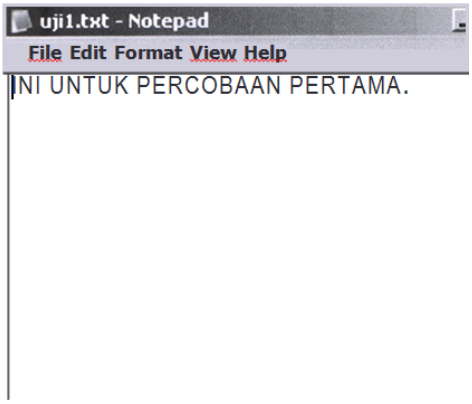
Keterangan:

untuk karakter ‘g’
45.000 x 1 *bit (0)* = 45.000 *bit*
untuk karakter 'o'
13.000 x *3bit* (101) = 39.000 *bit*
untuk karakter `p'
12.000 x *3bit* (110) = 36.000 *bit*
untuk karakter 'h'
16.000 x *3bit (111)* = 48.000 *bit*
untuk karakter `e'
9.000 x *4bit (1101)* = 36.000 *bit*
untuk karakter 'r'
5.000 x 4bit (1100)= *20.000* bit

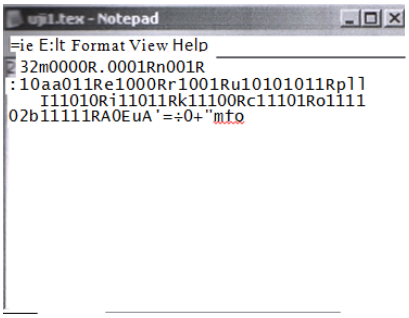
Jumlah *bit* yang digunakan dalam pembentukan kode Huffman untuk karakter" g,o,p,h,e,r, "= 45.000 + 39.000 + 36.000 + 48.000 + 36.000 + 20.000 = 224.000 *bit.*

5.2. Pengujian Kompresi dari File

Uji coba dirancang untuk mengetahui hasil kompresi apakah dugaan sebelumnya, dimana jika ada file yang dikompres berukuran kecil, maka hasil kompresnya lebih besar ukurannya daripada file asli.



Gambar File asal uji1.txt



Gambar File hasil Kompresi dari File uji1.txt

Dari ujicoba yang dilakukan, tidak ada pengurangan ukuran *file*, akan tetapi jumlah karakter semakin banyak.

6. KESIMPULAN DAN SARAN

Pada akhir dari penelitian ini, dapat ditarik beberapa kesimpulan dan saran yang dapat dilakukan dalam rangka mengembangkan hasil penelitian ini di kelak kemudian hari.

6.1. Kesimpulan

Dari hasil ujicoba yang sudah dilakukan, maka dapat diambil kesimpulan sebagai berikut:

1. Dari hasil pengujian yang dilakukan, algoritma huffman dapat melakukan kompresi teks sekitar 70% jika dibandingkan dengan menggunakan kode ASCII dan sekitar 25% jika dibandingkan dengan menggunakan 3 bit kode.
2. Kompresi file dengan algoritma Huffman tidak berhasil jika terdapat 256 jenis dan masing-masing jenis mempunyai frekuensi kemunculan yang hampir sama banyak.
3. Kompresi file juga kurang berhasil jika isi file terlalu sedikit sehingga ukuran file asli bisa jadi lebih kecil dari file hasil kompresi karena file kompresi masih harus menyimpan *huffman tree*-nya.

6.2. Saran

Untuk dapat lebih melihat dan membuktikan keefektifan, kelebihan dan kelemahan dari algoritma Huffman, perlu diadakannya sebuah penelitian yang bertujuan untuk membandingkan seluruh algoritma kompresi dalam berbgai jenis data maupun file.

7. DAFTAR PUSTAKA

Jonathan, Tommy, *Pemampatan Data Dengan Algoritma Huffman*, Surakarta, FMIPA-UNS, 1996

Lewis R, Harry & Deneberg, Larry,*Data Structure and Their Algorithm*, Harper Publisher, New York, 1991.

Madenda, Sarifudin, *Kompresi Citra Gray-Level dengan Metode Block Coding*, Proceeding Workshop on ECI, ITB, 1999

Yuwono, Rudi, *Penerapan Teknik Companding File Teks dengan Menggunakan Algoritma Huffman*, Proceeding Third Workshop on ECI, ITB, 1999

http://www.delphi-id.org, *Algoritma Huffman dan LZW* (diakses pada: 13 Mei 2009, pukul 15.00 WIB)

http://www.informatika.org, *Berbagai Variasi Algoritma Huffman* (diakses pada: 13 Mei 2009, pukul 17.30 WIB)

http://www.math.abdn.ac.uk *Huffman Codes* (diakses pada: 20 Mei 2009, pukul 13.00 WIB)