Volume IV 2009 ISSN: 1907-3526

SRITI
Seminar Nasional Riset Teknologi Informasi

Proceeding

Seminar Nasional Riset Teknologi Informasi 2009

"Ubiquitos Computing"

Yogyakarta, 08 Agustus 2009

Komputasi

Kecerdasan Buatan

Teknologi Basis Data

(Data Meaning, Data Warehouse)

Pemodelan dan Aplikasi Sistem Informasi

Komunikasi Data dan Jaringan Komputer

Signal Processing

Sistem Kendali Robotika

Pengolahan Citra

Multimedia dan Grafika

Games

Diselenggarakan Oleh :

YAYASAN PENDIDIKAN WIDYA BAKTI
STMIK
AKAKOM
YOGYAKARTA
Yang Pertama dan Utama

Perbandingan Metode Regresi dan Jaringan Saraf Ttiruan dalam Melakukan Prediksi
Sri Redjeki 105

Sistem Pendukung Keputusan Pengendalian Persediaan Menggunakan Model EOQ Studi Kasus Pada Perusahaan Flooring "NMS"
Emy Susanti 111

Aplikasi Hall Effect Sensor Pada Perhitungan Tingkat Ketebalan Cangkang Telur Itik Menggunakan Logika Fuzzy
Darmanto, Dwi Taufik Hidayat, Indra Budi Tresno 119

Implementasi Fuzzy Controller Dengan Pemrograman BASCOM
Zakarias Situmorang 125

Pengembangan Sistem Ekstraksi Informasi untuk Dokumen Legal Indonesia: Studi Kasus Dokumen Undang-Undang Republik Indonesia
Susy Violina dan Indra Budi 135

C. Teknologi Basis Data

Alat Bantu Penentuan Harga Pokok Produksi dengan Metode *Job Order Costing*
Al. Agus Subagyo 143

Algoritma Blowfish Untuk Pengamanan Data
Indra Yatini B. 151

Analisis Data dari Pembangunan Datawarehouse Perusahaan Percetakan
LN Harnaningrum 157

Aplikasi Buku Telepon Untuk Operator Telepon Di STMIK AKAKOM
Sigit Anggoro,S.T.,M.T. 163

Klasifikasi Artikel Berita Berbahasa Indonesia secara Otomatis dengan Menggunakan Metode Naive Bayes Classifier
Arni Darliani Asy'arie, Adi Wahyu Pribadi 173

Organisasi Berkas Dengan Menggunakan 3 Varian Metode Hash (*Coalesced Hashing, Prograssive Overflow, Buckets*)
Pulut Suryati 179

Pemanfaatan Layanan SMS untuk Pengiriman Data Pengukuran Secara Paket
Djohar Syamsi , Oka Mahendra 187

Peringkas Otomatis Artikel Berita Berbahasa Indonesia dengan Menggunakan Metode TF-IDF
Dzakiah Nur Fadhilah, Adi Wahyu Pribadi 195

Sistem Pencatatan Konsumsi Listrik atau Air di Pelanggan Dengan Jaminan Konsistensi Data
Sigit Anggoro; Lucia Nugraheni Harnaningrum 203

Visualisasi Pengkodean *Huffman* dengan Pohon Biner
Febri Nova Lenti 211

D. Pemodelan dan Aplikasi SI

Analisis Sistem Informasi Strategis PT Intan Pariwara Klaten
Nurcahyani Dewi Retnowati 221

Analisis Tren Penelitian Tugas Akhir Mahasiswa Jenjang S1 STMIK AKAKOM
Totok Suprawoto 229

Aplikasi Penyimpanan Data Sementara pada Perangkat Mobile untuk Aplikasi Pengelola Keuangan di Komputer *Desktop*
Ardiansyah, Wahyu Pujiyono, Mazin Ma'dan 241

Aplikasi Presensi Sidik Jari Menggunakan Database Server
Badiyanto 251

Visualisasi Pengkodean *Huffman* dengan Pohon Biner

Febri Nova Lenti

STMIK AKAKOM Yogyakarta
ln. Raya Janti 143 Karangjambe yogyakarta
E-mail : *febri@akakom.ac.id*

Abstrak

Kompresi data adalah hal yang sangat penting dalam penyimpanan arsip maupun dalam pengiriman data. Salah satu metode untuk kompresi data adalah Pengkodean Huffman yang bekerja dengan cara meminimumkan jumlah bit yang dibutuhkan, sehingga panjang kode untuk setiap karakter sedapat mungkin diperpendek , terutama untuk karakter yang frekuensi kemunculannya besar. Cara untuk membentuk kode Huffman adalah dengan membentuk pohon biner, sehingga penelitian ini akan membahas pengembangan perangkat lunak untuk visualisasi pengkodean Huffman dengan pohon biner. Pada penelitian ini pengembangan perangkat lunak menggunakan model proses *waterfall* dan hasil / keluarannya berupa visualisasi pohon Huffman dengan kapasitas maksimum 511 simpul dan kedalaman sampai 32 level, data histogram untuk serialisasi kode Hufman, rasio kompresi dan data terkompresi.

Kata Kunci : Data Histogram , HCG, Kode Huffman, Kompresi, Pohon Biner

PENDAHULUAN

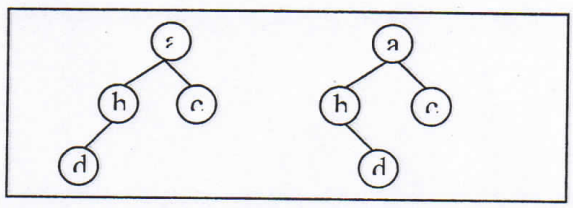
Dalam komunikasi data, pesan yang dikirim seringkali ukurannya sangat besar sehingga waktu pengirimannya lama. Begitu juga dalam penyimpanan data, arsip (*file*) yang berukuran besar memakan memori yang besar. Kedua masalah ini dapat diatasi dengan mengkodekan pesan sesingkat mungkin , sehingga waktu pengiriman pesan juga relative cepat dan ruang penyimpanan yang dibutuhkan juga sedikit. Cara pengkodean seperti ini disebut pemampatan (*compression*) data.

Salah satu cara untuk melakukan kompresi data adalah dengan menggunakan kode Huffman yang meminimumkan jumlah bit yang dibutuhkan, sehingga panjang kode untuk setiap karakter sedapat mungkin diperpendek , terutama untuk karakter yang frekuensi kemunculannya besar.

Cara untuk membentuk kode Huffman adalah dengan membentuk pohon biner. Untuk itu pada penelitian ini akan dibangun sebuah aplikasi yang akan memvisualisasikan pembentukan kode Huffman dengan pohon biner sehingga *tracing* dari pembentukan kode Huffman dapat dilihat.

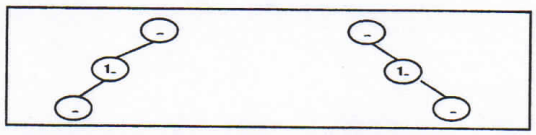
Pohon Biner

Munir (2003) dalam bukunya *Matematika Diskrit,* pohon biner adalah pohon yang setiap simpul cabangnya mempunyai paling banyak dua buah anak, yang disebut anak kiri (*left child*) dan anak kanan (*right child*). Gambar 1 Adalah dua pohon biner yang berbeda.

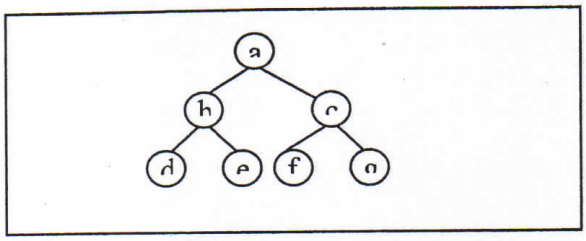


Gambar 1. dua buah pohon biner yang berbeda

Pohon yang semua simpulnya terletak dibagian kiri saja atau dibagaian kanan saja disebut pohon condong(*skewed tree*). Pohon yang condong ke kiri disebut pohon condong kiri dan pohon yang condong ke kanan disebut pohon condong kanan seperti yang ditunjukkan gambar 2. Pohon Biner penuh adalah pohon biner yang setiap simpulnya mempunyai tepat dua buah anak, kiri dan kanan, kecuali simpul pada level bawah. Seperti ditunjukkan gambar 3.



Gambar 2. Pohon Condong kiri dan Pohon condong kanan



Gambar 3. Pohon biner penuh

Kode Huffman

Munir (2003) untuk meminimumkan jumlah bit yang dibutuhkan, panjang kode untuk setiap karakter sedapat mungkin diperpendek, terutama untuk karakter yang frekuensi kemunculannya besar. Pemikiran seperti inilah yang mendasari munculnya kode Huffman. Misal kode ASCII untuk A = 01000001, B = 01000010, C = 01000011, D= 01000100 maka string 'ABACCDA' direpresentasikan menjadi serangkaian bit :
01000001010000100100000101000011010000110100001000001

Berdasarkan metode pengkodean ASCII , representasi 7 huruf diatas membutuhkan 7 x 8 = 56 bit (7 byte). Dengan kode Huffman pertama akan dicari frekuensi kemunculan huruf huruf diatas, yaitu seperti yang ditunjukkan tabel 1.

Dengan menggunakan kode Huffman , string 'ABACCDA' ini direpresentasikan menjadi rangkaian bit :

0110010101110

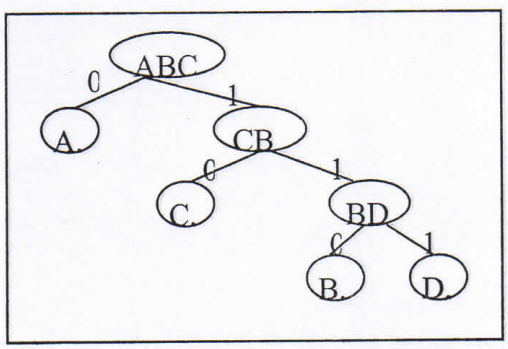
Tabel 1. Tabel frekwensi dan kode huffman untuk string 'ABACCDA'

Simbol	Frekuensi	Peluang	Kode Huffman
A	3	3/7	0
B	1	1/7	110
C	2	2/7	10
D	1	1/7	111

Jadi, dengan menggunakan kode Huffman ini, jumlah bit yang dibutuhkan untuk string ABACCDA hanya 13 bit. Simbol symbol yang sering muncul direpresentasikan dengan kode yang lebih pendek daripada kode untuk symbol lain. Kode untuk setiap symbol tidak boleh merupakan awalan dari kode yang lain sebab akan menimbulkan keraguan (*ambigu*) dalam proses *decoding*.

Untuk mendapatkan kode Huffman, mula mula dihitung dulu frekwensi kemunculan tiap simbol di dalam teks. Cara pembentukan kode Huffman adalah dengan membentuk pohon biner sebagai berikut :

a. Pilih dua simbol dengan peluang (probability) paling kecil (pada string diatas adalah simbol B dan D). Kedua simbol tadi dikombinasikan sebagai simpul orangtua dari simbol B dan D sehingga menjadi simbol BD dengan peluang 1/7 +1/7 = 2/7, yaitu jumlah peluang kedua anaknya. Simbol baru ini diperlakukan sebagai simpul baru dan diperhitungkan dalam mencari simbol selanjutnya yang memiliki peluang paling kecil.
b. Selanjutnya pilih dua simbol berikutnya, termasuk simbol baru yang mempunyai peluang terkecil. Pada contoh ini, dua simbol tersebut adalah C (peluang = 2/7) dan BD (peluang = 2/7). Lakukan hal yang sama seperti langkah sebelumnya sehingga dihasilkan simbol baru CBD dengan frekwensi 2/7 + 2/7 = 4/7.
c. Prosedur yang sama dilakukan pada dua simbol berikutnya yang mempunyai peluang terkecil ,yaitu A (peluang = 3/7) dan CBD (peluang = 4/7) sehiingga menghasilkan simpul ABCD, yang merupakan akar pohon biner dengan peluang 3/7 + 4/7 = 7/7.



Gambar 4. Pohon Huffman untuk pesan 'ABACCDA'

Daun pada pohon biner menyatakan simbol yang digunakan dalam teks atau pesan. Kode setiap simbol dengan memberi label 0 pada setiap cabang kiri dan label 1 untuk setiap cabang kanan. Pohon Huffman untuk string 'ABACCDA' ditunjukkan pada gambar 4.

Dengan membuat lintasan dari akar ke daun, akan dihasilkan kode untuk setiap simbol. Dari gambar 4. diperoleh kode untuk masing-masing simbol asal sebagai berikut:

A = 0 B= 110 C=10 D=111

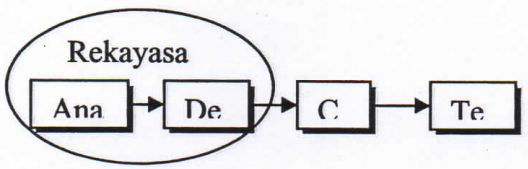
Simbol yang paling sering muncul memiliki kode dengan jumlah bit paling sedikit. Perhatikan bahwa tidak ada simbol yang kodenya merupakan kode awalan untuk simbol yang lain. Dengan cara ini, kode yang diawali 0 dapat dipastikan adalah A,

sedangkan kode yang diawali 1 mungkin B, C atau D yang harus diperiksa lagi dari bit bit selanjutnya.

Kode Huffman tidak bersifat unik, artinya kode untuk setiap karakter berbeda-beda pada setiap pesan bergantung pada frekwensi kemunculan karakter tersebut di dalam pesan. Selain itu, keputusan apakah suatu simpul pada pohon Huffman diletakkan di kiri atau kanan juga menentukan kode yang dihasilkan (tetapi tidak mempengaruhi panjang kodenya)

PEMBAHASAN

Pengembangan perangkat lunak visualisasi Pembangkitan kode Huffman (HCG) ini dilakukan dengan menggunakan model *waterfall* yang tahapan tahapannya seperti ditunjukkan gambar 5.



Gambar 5. Model proses Waterfall

Analisis

Perangkat Lunak Pembentukan Kode Huffman menggunakan Pohon Biner (*Huffman Code Generator*, HCG) memiliki spesifikasi sebagai berikut:

1. Memiliki fungsi aplikasi editor standar (buka file yang sudah ada [open], buat file baru [new], modifikasi [edit] , simpan [save dan save as], copy text ke clipboard, paste text dari clipboard, cut text, print preview, print, setup printer, ganti font, text justifying, text wrapping dll).
2. Menampilkan data histogram kode huffman.
3. Menampilkan pohon huffman.
4. Menampilkan data terkompresi
5. Hanya bisa menangani jumlah simpul maksimum sebesar 511 simpul.
6. Hanya bisa menangani tingkat kedalaman pohon sebesar 32 level (dibatasi ukuran 32 bits)

Perangkat lunak yang dibangun mempunyai 3 utilitas yaitu:

1. Menampilkan Kode Huffman
2. Menampilkan Gambar Pohon Huffman
3. Menampilkan Data Terkompresi

Rancangan

Ada beberapa struktur data utama pada program yang dibutuhkan dalam proses algoritma huffman:

1. Streambit data output / input (menggunakan struktur data record yang salah satu *member*-nya berupa pointer)
2. Data histogram untuk menyimpan serialisasi data input (struktur data record)
3. Data Simpul untuk menyimpan setiap simpul dalam pohon (struktur data pohon biner).

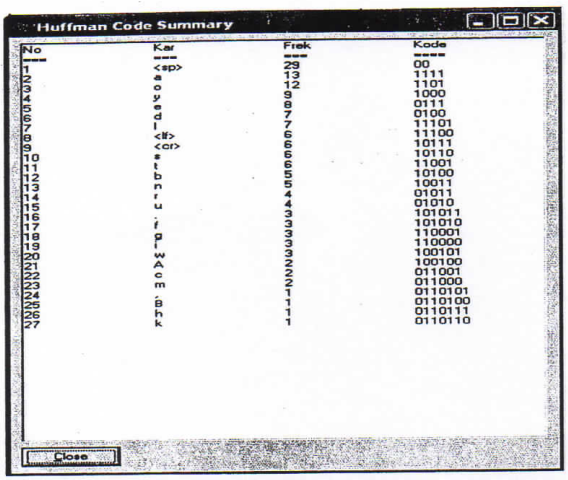
Kode Huffman dibangun melalui fungsi Huffman_Hist dan Huffman_Make_Tree. Data yang akan ditampilkan tidak hanya sekedar kode huffman tapi juga kekerapan / frekuensi dari simbol / karakter yang ada pada data input (*uncompressed data*). Yang akan ditampilkan adalah data dari struktur data *record* histogram yang dideklarasikan sebagai berikut:

```
typedef struct {
    int Symbol;
    unsigned int Count;
    unsigned int Code;
    unsigned int Bits;
} huff_sym_t;
```

Untuk menampilkan kode huffman dari suatu input data adalah menggunakan algoritma berikut:

1. Inisialisasi stream data output.
2. Membuat data histogram dari serialisasi data input (update variabel Symbol dan Count).
3. Membuat pohon huffman (update variabel Code dan Bits).
4. Sort data histogram berdasarkan frekuensi munculnya karakter mulai dari yang terkerap (*descending*).
5. Tampilkan tabel kode huffman dari data histogram dengan kolom sebagai berikut:
   a. Nomor Urut Karakter (No)
   b. Symbol / Karakter (Kar)
   c. Kekerapan / Frekuensi (Frek)
   d. Kode huffman karakter tersebut (Kode).

Gambar 6. memperlihatkan contoh tampilan tabel data histogram kode huffman.



Gambar 6 Tabel data histogram kode huffman

Semua fungsi yang menjelajahi (*traverse*) variabel data pohon menggunakan jenis fungsi rekursif. Urutan algoritma dalam menampilkan pohon huffman adalah sebagai berikut:

1. Inisialisasi stream data output.
2. Membuat data histogram dari serialisasi data input
3. Membuat pohon huffman yang akan menginisialisasi variabel simpul (*nodes*) dan mengembalikan nilai fungsi ke variabel pointer yang menunjuk ke akar simpul (variabel *root*).
4. Menampilkan pohon huffman ke layar dengan menjelajahi (*traverse*) variabel pohon (*nodes*) mulai dari akar simpul (yang ditunjuk oleh variabel pointer).

Algoritma untuk menampilkan pohon huffman (nomor 4) akan dibahas lebih detail dengan urutan algoritma sebagai berikut:

a. Menentukan tingkat kedalaman pohon (disimpan dalam variabel *level*). Fungsi yang dibuat akan menjelajahi pohon hingga ke akar yang terdalam. Nama fungsinya adalah `get_level_tree`. Kedalaman dihitung mulai dari akar (level 1) sampai dengan simpul terdalam (level n).

b. Menentukan lebar pohon dengan mengasumsikan pohon berjenis *full binary tree* berdasarkan tingkat kedalaman (variabel *level*) dengan rumus:

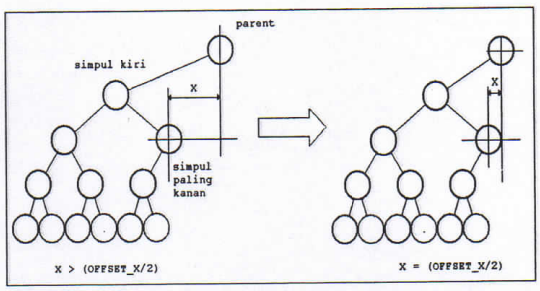
```
width=(2^(level-1) * jarak_antar_simpul_anak
```

variabel `jarak_antar_simpul_anak` pada program diberi nama `OFFSET_X`

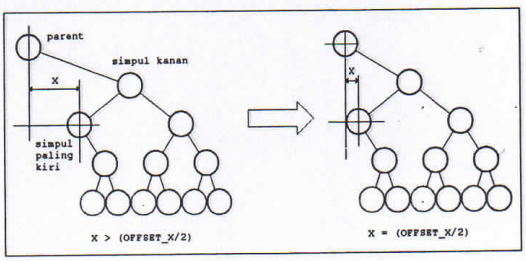
c. Menentukan titik tengah setiap simpul. Algoritma ini sudah mengimplementasikan perampingan pohon yang semula diasumsikan *full binary tree* dengan cara:

- simpul kiri (beserta simpul-simpul dibawahnya) akan bergeser ke kanan jika jarak simpul anak paling kanan (dijelajahi sampai terdalam) dengan simpul *parent* 1 level di atas simpul kiri tersebut lebih dari *jarak_antar_simpul_anak* dibagi 2 (OFFSET_X/2) [lihat gambar 7] .
- simpul kanan (beserta simpul-simpul dibawahnya) akan bergeser ke kiri jika jarak simpul anak paling kiri (dijelajahi sampai terdalam) dengan simpul *parent* 1 level di atas simpul kanan tersebut lebih dari *jarak_antar_simpul_anak* dibagi 2 (OFFSET_X/2) [lihat gambar 8].

Kedua algoritma diatas dijalankan secara rekursif sehingga pohon menjadi ramping.



Gambar 7. Perampingan pohon ke kanan



Gambar 8. Perampingan pohon ke kiri

d. Menggeser koordinat pohon yang sudah ramping ke tepi kiri layar.
Dengan asumsi awal jenis pohon adalah *full binary tree*, maka titik koordinat simpul akar masih ditengah dari lebar layar yang diasumsikan. Maka untuk itu pohon kemudian digeser ke tepi kiri layar. Sebelum digeser dicari titik koordinat simpul paling kiri dan paling kanan (margin kiri dan kanan), lebar pohon terbaru (margin kanan dikurangi margin kiri), tinggi pohon (berdasarkan variabel *level* dan konstanta OFFSET_Y [jarak antar simpul anak dengan simpul parent])

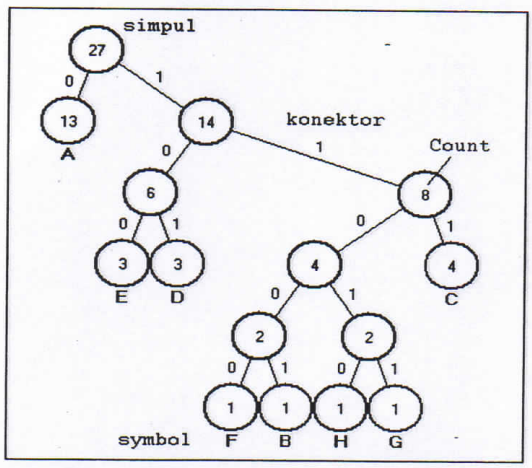
e. Menset variabel Count pada setiap simpul *parent* sehingga variabel Count pada *parent*

merupakan penjumlahan dari frekuensi simpul anak dibawahnya.

f. Menentukan lebar dan tinggi object Image berdasarkan variabel lebar dan tinggi pohon yang didapat sebelumnya.

g. Menampilkan / menggambar pohon pada object Image. Ada 4 jenis object yang akan digambar yaitu, simpul (berbentuk lingkaran), frekuensi simpul (variabel Count pada struktur data pohon), karakter jika simpul tersebut simpul daun (variabel Symbol) dan terakhir adalah konektor antar simpul beserta keterangan konektor (0 jika konektor ke simpul kiri dan 1 jika konektor ke simpul kanan).

Gambar 9. merupakan contoh tampilan pohon hasil dari algoritma di atas dengan data input sebagai berikut:
AAAAAAAAAAAAABCCCCDDDEEEFGH.



Gambar 9. Contoh Gambar Pohon Huffman

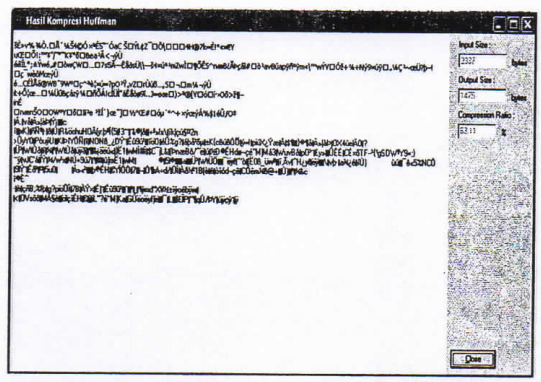
Menampilkan data terkompresi adalah sama dengan melakukan fungsi coder secara penuh dan kemudian menampilkan output data terkompresi ke layar (tentu saja hasilnya tidak akan terbaca). Gambar 10. memperlihatkan contoh tampilan data terkompresi beserta rasio pengkompresiannya.

IMPLEMENTASI

Implementasi aplikasi perangkat lunak HCG (*Huffman Code Generator*) menggunakan kompiler Borland C++ Builder 6.0. Kompiler ini dipilih karena pustaka yang tersedia menggunakan bahasa C dan mendukung pemograman visual sehingga mudah untuk mengimplementasikan fungsi akhir dari perangkat lunak yaitu menampilkan hasilnya pada layar atau *Graphical User Interface* (GUI)

Adapun tahap-tahap pembuatan perangkat lunak HCG adalah sebagai berikut:

1. Membuat aplikasi editor teks.
2. Modifikasi pustaka *Basic Compiler Library* (BCL) sesuai kebutuhan aplikasi.
3. Membuat 3 form untuk menampilkan 3 fungsi aplikasi yaitu:
   a. Menampilkan Kode Huffman
   b. Menampilkan Pohon Huffman
   c. Menampilkan Data Terkompresi
4. Membuat menu dan submenu sebagai interface untuk menjalankan 3 fungsi aplikasi tersebut.
5. Menulis kode untuk menggambar pohon huffman



Gambar 10. Tampilan data terkompresi dan rasio kompresi

Kode sumber (*source code*) untuk implementasi kompresi dan dekompresi data kode *Huffman* pada penelitian ini mengambil dari kode yang telah dibuat oleh Marcus Geelnard dengan beberapa modifikasi sesuai dengan kegunaan program yang akan dibangun (perubahan dapat dilihat secara detil pada bab IV). Marcus Geelnard pada rentang waktu tahun 2003-2006 mengembangkan program pustaka untuk kompresi dan dekompresi data yang dinamakan *Basic Compression Library* (BCL). Pustaka ini terbuka dan bebas untuk digunakan, dimodifkasi dan dikembangkan lebih lanjut (*open source*). Kode sumber tersebut ditulis menggunakan bahasa standar ANSI C. Versi BCL yang dipakai adalah versi 1.2.

Pustaka BCL Huffman mempunyai 2 file yaitu: huffman.h (file header) dan huffman.c (file implementasi). Seperti pustaka umumnya, BCL ada 2 bagian yaitu struktur data dan implementasi algoritma (fungsi).

1. Struktur Data
   Kode Huffman diimplementasikan menggunakan stuktur data pohon biner (*binary tree*). Struktur ini menggunakan struktur dasar *double linked list*. Selain itu dalam implementasinya pustaka ini juga terdapat struktur data *record* untuk menampung histogram dari serialisasi data input dan struktur

data *record* yang berfungsi sebagai stream data output (pada proses kompresi) dan sebagai stream data input (pada proses dekompresi). Berikut implementasi struktur data pada BCL Huffman:

```
/* struktur data record yang digunakan
sebagai :
   1. stream data output pada proses
kompresi dan
   2. stream data input pada proses
dekompresi */

typedef struct {
    unsigned char *BytePtr;
    unsigned int  BitPos;
} huff_bitstream_t;

/* struktur data record untuk menampung
   histogram dari serialisasi data input
*/

typedef struct {
    int Symbol;
    unsigned int Count;
    unsigned int Code;
    unsigned int Bits;
} huff_sym_t;

/* struktur data pohon biner untuk
implementasi kode Huffman
   pada proses kompresi */

typedef struct huff_encodenode_struct
huff_encodenode_t;

struct huff_encodenode_struct {
    huff_encodenode_t *ChildA, *ChildB;
    int Count;
    int Symbol;
};

/* struktur data pohon biner untuk
implementasi kode Huffman
   pada proses dekompresi *?

typedef struct huff_decodenode_struct
huff_decodenode_t;

struct huff_decodenode_struct {
    huff_decodenode_t *ChildA, *ChildB;
    int Symbol;
};
```

2. Fungsi

Fungsi-fungsi yang ada di Huffman BCL adalah sebagai berikut:

a. _Huffman_InitBitstream (insialisasi stream data)
b. _Huffman_ReadBit (membaca 1 bit dari bitstream)
c. _Huffman_Read8Bits (membaca 1 byte (8 bit) dari bitstream)
d. _Huffman_WriteBits (menulis 1 atau beberapa bits ke bitstream)
e. _Huffman_Hist (membuat histogram dari serialisasi data input)
f. _Huffman_StoreTree (menyimpan pohon huffman ke stream output dan ke dalam *look-up table* / array simbol).
g. _Huffman_MakeTree (membangun /meng-generate pohon huffman).
h. _Huffman_RecoverTree (mengambil / me-recover pohon huffman dari bitstream).
i. Huffman_Compress (fungsi coder untuk mengkompres data menggunakan kode Huffman).
j. Huffman_Uncompress (fungsi decoder untuk mendekompres data yang dikompres menggunakan kode Huffman).

Fungsi-fungsi [a]-[h] adalah fungsi *private* (ditandai dengan karakter underscore pada awal penamaan fungsi) artinya hanya bisa diakses oleh fungsi coder / decoder dan tidak bisa secara langsung dari program yang menggunakan pustaka tersebut. Sedangkan fungsi [i] dan [j] adalah fungsi *public* (fungsi yang bisa langsung diakses oleh program yang menggunakan pustaka tersebut).

Fungsi Coder Huffman (Huffman_Compress)

Fungsi ini mengkompres data input, menghasilkan data output hasil kompresi. Fungsi ini mengembalikan nilai ukuran / size data output (*return value*).

Sintaks:

```
outsize = Huffman_Compress(in, out,
insize)
```

```
ket: outsize = size of output buffer
after compression
     in       = pointer to the input
buffer (uncompressed data)
     out      = pointer to the output
buffer (compressed data)
     insize   = size of input buffer
```

Fungsi ini secara berurutan melakukan hal sebagai berikut:

1. Inisialisasi stream data output (_Huffman_InitBitstream).
2. Membuat histogram dari serialisasi data input (_Huffman_Hist)
3. Membangkitkan pohon Huffman (_Huffman_MakeTree)

   Fungsi ini memanggil fungsi _Huffman_StoreTree (didalam fungsi ini ada pemanggilan fungsi _Huffman_WriteBits untuk menulis data pohon ke streambit data output).

4. Meng-encode stream data input ke streambit data output berdasar kode huffman menggunakan fungsi _Huffman_WriteBits.

Fungsi Decoder Huffman (Huffman_Uncompress)

Fungsi ini mendekompres data input (compressed data) menjadi data output yang tidak terkompres (uncompressed data).
Sintaks:

```
Huffman_Uncompress(in, out, insize,
outsize)
```

```
ket: in       = pointer to the input
buffer (compressed data)
     out      = pointer to the output
buffer (uncompressed data)
     insize   = size of input buffer
     outsize  = size of output buffer
after decompression
```

Fungsi ini secara berurutan melakukan hal sebagai berikut:

1. Inisialisasi stream data input (_Huffman_InitBitstream).
2. Me-recover / mengambil pohon huffman dari bitstream input (_Huffman_RecoverTree). Fungsi ini mengembalikan ke variabel dengan struktur data pohon (*return value*). Fungsi ini menggunakan fungsi _Huffman_ReadBit dan _Huffman_Read8Bits dalam menjalankan fungsinya.
3. Meng-decode stream data input ke stream data output berdasarkan kode huffman menggunakan fungsi _Huffman_ReadBit.

Batasan-batasan (*constraint*) BCL versi 1.2. adalah sebagai berikut:

1. Jumlah maksimum simpul pohon adalah (2^(8+1)-1) = 511
   Yang didefinisikan dalam baris program sebagai berikut:

   ```
   #define MAX_TREE_NODES 511
   ```

   Dan dideklarasikan variabel array simpul sebagai berikut:

   ```
   huff_encodenode_t
   nodes[MAX_TREE_NODES];
   ```

2. Kedalaman pohon maksimum adalah 32 level (dibatasi oleh ukuran 32 bits).

Pada penelitian ini sesuai dengan tujuannya yaitu membuat perangkat lunak untuk membentuk kode Huffman dengan pohon biner dan kemudian divisualisasikan dalam bentuk gambar (image) pohon Huffman maka terdapat beberapa perubahan / modifikasi dari *source code* BCL sebagai berikut :

- Modifikasi pada struktur data, meliputi struktur data pohon ditambah 2 variabel
- Modifikasi pada fungsi Huffman_MakeTree (membangun /meng-generate pohon huffman), Huffman_InitBitstream (insialisasi stream data), Huffman_WriteBits (menulis 1 atau beberapa bits ke bitstream), Huffman_Hist (membuat histogram dari serialisasi data input), Huffman_StoreTree (menyimpan pohon huffman ke stream output dan ke dalam *look-up table* / array simbol).

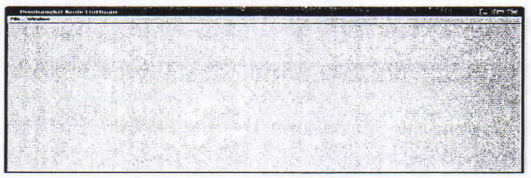
Kemudian beberapa *source code* baru yang dibuat pada penelitian ini adalah sebagai berikut :

1. Menampilkan secara grafis / gambar pohon Huffman
2. Menampilkan data terkompressi beserta rasio kompressi

PENGUJIAN

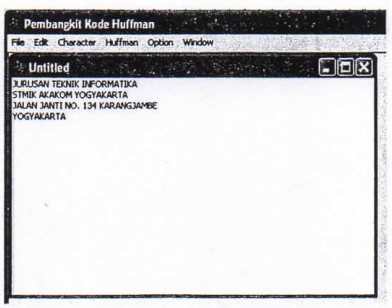
Langkah-langkah pengujian adalah sebagai berikut:

a. Menjalankan aplikasi
   Double click pada file "Huffman.exe" maka aplikasi akan dijalankan dan akan muncul jendela seperti pada gambar 11.



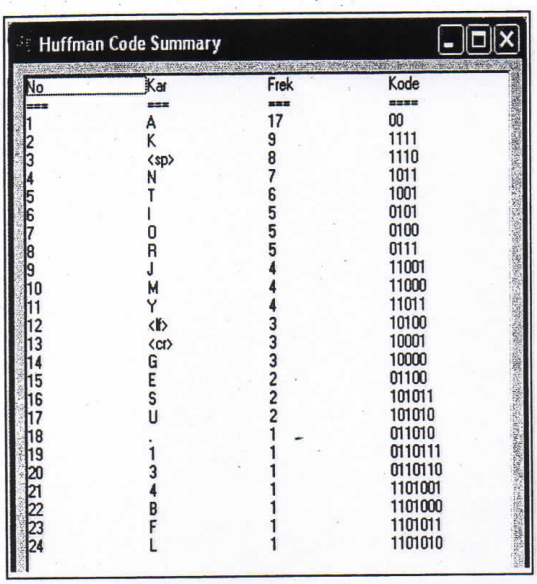
Gambar 11. Tampilan awal aplikasi HCG

Klik menu File→New dan kemudian ketik teks berikut:
JURUSAN TEKNIK INFORMATIKA
STMIK AKAKOM YOGYAKARTA
JALAN JANTI NO. 134 KARANGJAMBE
YOGYAKARTA
Tampilannya dapat dilihat pada gambar 12



Gambar 12. Tampilan aplikasi HCG setelah diberi teks

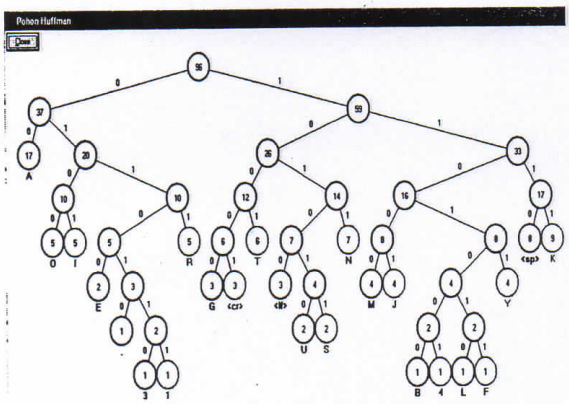
b. Tampilkan Tabel Kode Huffman untuk teks tersebut (menu Huffman→Show Huffman Code). Tampilannya sepert pada gambar 13.



No	Kar	Frek	Kode
1	A	17	00
2	K	9	1111
3	<sp>	8	1110
4	N	7	1011
5	T	6	1001
6	I	5	0101
7	O	5	0100
8	R	5	0111
9	J	4	11001
10	M	4	11000
11	Y	4	11011
12	<lf>	3	10100
13	<cr>	3	10001
14	G	3	10000
15	E	2	01100
16	S	2	101011
17	U	2	101010
18	.	1	011010
19	1	1	0110111
20	3	1	0110110
21	4	1	1101001
22	B	1	1101000
23	F	1	1101011
24	L	1	1101010

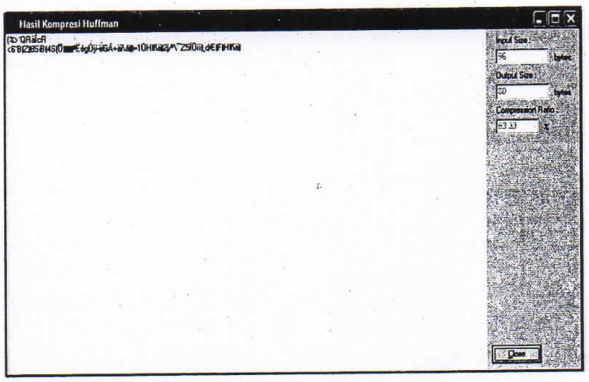
Gambar 13. Tabel Kode Huffman

c. Tampilkan Pohon Huffman (menu Huffman→Show Huffman Tree). Tampilannya seperti gambar 14.



Gambar 14. Pohon Huffman

d. Tampilkan Teks Terkompresi (menu Huffman→Show Compression Teks). Tampilannya seperti gambar 15,



Gambar 15. Teks Terkompresi

PENUTUP

Visualisasi Pengkodean Huffman dengan Menggunakan Pohon Biner ini dapat digunakan sebagai bahan peraga dalam pengajaran. Jumlah maksimum simpul pohon yang dapat ditampilkan adalah (2 ^(8+1)-1) = 511 dan kedalamannya maksimum sampai level 32.

DAFTAR PUSTAKA

[1]. Dulimarta, H., *Catatan Kuliah Teori Graph*, Teknik Informatika ITB, 1994

[2]. Deo, N., *Graph Theory with Aplications To Engineering and Computer Science*,Prentice – Hall International, 1974

[3]. Geelnard, M. ,Basic Compression Library, http://bcl.comli.eu, 2008

[4]. Munir, R., *Matematika Diskrit,* edisi kedua, Penerbit Informatika Bandung 2003.

[5]. Pressman R.S., Software Engineering A Practitioner's Approach, Mc Graw-Hill 1997.