PENGKOMPRESIAN FILE
DENGAN MENERAPKAN ALGORITMA HUFFMAN
BERBASIS MODEL BIGRAM

SKRIPSI

oleh:
ULVA WULANDARI
0410960060-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2009

UNIVERSITAS BRAWIJAYA

PENGKOMPRESIAN FILE
DENGAN MENERAPKAN ALGORITMA HUFFMAN
BERBASIS MODEL BIGRAM

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar
Sarjana Komputer dalam bidang Ilmu Komputer

oleh:
ULVA WULANDARI
0410960060-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2009


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

PENGKOMPRESIAN FILE
DENGAN MENERAPKAN ALGORITMA HUFFMAN
BERBASIS MODEL BIGRAM

Oleh:
ULVA WULANDARI
0410960060-96

Setelah dipertahankan di depan Majelis Penguji
Pada tanggal 2 Februari 2009
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana dalam bidang Ilmu Komputer

Pembimbing I	Pembimbing II
Drs. Muh. Arif Rahman, M.Kom	**Drs. Marji, MT**
NIP. 131 971 481	**NIP. 131 993 386**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Agus Suryanto, MSc.
NIP. 132 126 049

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**: Ulva Wulandari**
NIM	**: 0410960060**
Jurusan	**: Matematika**
Penulis Skripsi berjudul	**: Pengkompresian file dengan menerapkan algoritma Huffman berbasis Model Bigram**

Dengan ini menyatakan bahwa :

1. **Isi dari Skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam Skripsi ini.**
2. **Apabila dikemudian hari ternyata Skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 18 Desember 2008
Yang menyatakan,

(Ulva Wulandari)
NIM. 0410960060


UNIVERSITAS BRAWIJAYA

PENGKOMPRESIAN FILE DENGAN MENERAPKAN ALGORITMA HUFFMAN BERBASIS MODEL BIGRAM

ABSTRAK

Kompresi bertujuan untuk memampatkan sebuah data agar diperoleh data dengan ukuran yang lebih kecil daripada ukuran aslinya, sehingga lebih efisien dalam menyimpannya serta mempersingkat waktu pertukaran data tersebut. Algoritma Huffman merupakan salah satu algoritma kompresi yang bersifat *Lossless,* yang menjamin bahwa berkas yang dikompresi dapat selalu dikembalikan ke bentuk aslinya.

Algoritma Huffman menggunakan prinsip pengkodean dimana karakter yang sering dipakai dikodekan dengan rangkaian bit yang pendek, dan karakter yang jarang dipakai dikodekan dengan rangkaian bit yang lebih panjang. Penggabungan algoritma Huffman dengan Bigram dilakukan dengan mengkodekan bigram yaitu tiap pasangan karakter berdekatan dalam sebuah file. Kode Huffman yang dibentuk akan bersifat dinamis dengan bergantung pada komposisi data yang bersangkutan. Pembentukan *header* pada file terkompresi dilakukan agar dapat dijadikan sebagai pedoman dalam proses dekompresi. Sehingga sebuah file terkompresi akan dapat dikembalikan menjadi file aslinya.

Uji coba sistem dilakukan terhadap file yang berekstensi **.doc*, **.txt, *.html* , dan **.bmp.* Dalam proses kompresi, evaluasi dilakukan terhadap ukuran file hasil kompresi. Sedangkan dalam proses dekompresi, evaluasi dilakukan terhadap ketersesuaian file hasil dekompresi dengan file aslinya. Evaluasi juga dilakukan terhadap waktu yang dibutuhkan selama proses berlangsung serta rasio kompresi yang terbentuk dari sistem. Berdasarkan uji coba yang telah dilakukan, diperoleh rasio kompresi rata-rata sebesar 2,07 (52%) untuk file **.doc*; 1,33 (25%) untuk file **.txt*; 1,19 (16%) untuk file **.html*; dan untuk file **.bmp* sebesar 3,01 (67%). Hasil perbandingan antara sistem dengan aplikasi WinZip dan WinRar adalah 1:2, dimana hasil file kompresi dari sistem ini memiliki ukuran dua kali lebih besar dibandingkan dengan aplikasi WinZip maupunWinRar.

FILE COMPRESSION BY APPLYING HUFFMAN ALGORITHM BASE ON BIGRAM MODEL

ABSTRACT

Compression has a purpose to compress data to get data smaller than its original size so the data wol be more eficient because it wol be easier to save it and the time needed to exchange teh data can be shortened. Huffman algorithm is one of compression algorithms that always returned to original form.

Huffman algorithm uses coding principle, that the most used character is coded with short series bit, and the rarely used character is coded with longer series bit. The combination of Huffman method and bigram is done by coded bigram that is every nearby character couple in a file. Code of Huffman formed will have the character of dynamic by base on pertinent data composition. Forming of header at compress file done so that can be made as guidance in course of decompression. So that a compress file will be able to be returned to become its original file.

System testing has been applied to file with extension of **.doc*, **.txt*, **.html*, and **.bmp*. During the process of the compression, evaluation was applied to the file size after the compression. While during the process of decompression, evaluation was applied to the file compatibility with its original form. BesideseEvaluation was given to the time needed for the process and the ratio of compression created from the system. Based on the test that has been done, the average ratio of tehe compression is 2,07 (52%) for **.doc* file; 1,33 (25%) for **.txt* file; 1,19 (16%) for **.html;* and 3,01 (67%) for **.bmp* file. The comparison result of system with result of compression from application of WinZip and WinRar is 1:2, where the result of compress file of this system have size measure twice as big as result of application of WinZip and also of WinRar.

KATA PENGANTAR

Alhamdulillahi rabbil 'alamin. Puji syukur penulis panjatkan kehadirat Allah SWT, karena atas segala rahmat dan limpahan hidayahNya, skripsi yang berjudul **"Pengkompresian File dengan Menerapkan Algoritma Huffman Berbasis Model Bigram"** ini dapat diselesaikan. Skripsi ini disusun dan diajukan sebagai syarat untuk memperoleh gelar sarjana pada program studi Ilmu Komputer, jurusan Matematika, fakultas MIPA, universitas Brawijaya.

Semoga Allah melimpahkan rahmat atas Nabi Muhammad SAW, makhluk paling mulia yang senantiasa memberikan cahaya petunjuk, dan atas keluarganya dan sahabat-sahabatnya..

Dalam penyelesaian skripsi ini, penulis telah mendapat begitu banyak bantuan baik moral maupun materiil dari banyak pihak. Atas bantuan yang telah diberikan, penulis ingin menyampaikan penghargaan dan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Drs. Muh. Arif Rahman, M.Kom dan Drs. Marji, MT , terima kasih atas semua waktu,saran, bantuan dan bimbingan yang telah diberikan.
2. Wayan Firdaus Mahmudy, SSi, MT selaku Ketua Program Studi Ilmu Komputer Unibraw Malang.
3. Candra Dewi, S.Kom selaku Penasihat Akademik.
4. Dr. Agus Suryanto, Msc selaku Ketua Jurusan Matematika Fakultas MIPA Unibraw Malang
5. Segenap bapak dan ibu dosen yang telah mendidik dan mengamalkan ilmunya kepada penulis.
6. Segenap staf dan karyawan di Jurusan Matematika FMIPA Universitas Brawijaya
7. Ayah, Ibu, Kakak dan Adik. Terima kasih atas cinta, kasih sayang, doa, dukungan dan semangat yang tiada henti.
8. Sahabat-sahabat Loelie serta teman-teman ilkomers `04. Terima kasih atas senyuman, semangat, dukungan, do'a dan hari-hari kita.
9. Pihak lain yang telah membantu terselesaikannnya Skripsi ini yang tidak bisa penulis sebutkan satu-persatu.

Semoga penulisan laporan tugas akhir ini bermanfaat bagi pembaca sekalian. Dengan tidak lupa atas kodratnya sebagai manusia, penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan, dan mengandung banyak kekurangan, sehingga dengan segala kerendahan hati penulis mengharapkan kritik dan saran yang membangun dari pembaca.

Malang, Desember 2008

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN


UNIVERSITAS BRAWIJAYA

BAB I
PENDAHULUAN

1.1 Latar Belakang

Dalam kehidupan sehari-hari, terutama dalam bidang teknologi informasi, komunikasi data sangat sering dilakukan. Komunikasi data ini berhubungan erat dengan pengiriman data menggunakan sistem transmisi elektronik dari satu terminal komputer ke terminal komputer yang lain. Besarnya ukuran data terkadang menjadi kendala dalam proses pengiriman data ini. Data dengan ukuran besar akan memakan waktu *transfer* yang lebih lama dibandingkan dengan data yang memiliki ukuran lebih kecil, terkadang ada resiko tidak dapat tertampung pada media penyimpanan dan tidak tersampaikannya data dalam proses *transfer* data. Suatu data yang berukuran besar akan juga membutuhkan banyak tempat dalam penyimpanannya, sehingga akan mengurangi kapasitas kosong dalam memori media peyimpanan. Oleh karena itu, manusia selalu berusaha untuk menemukan suatu cara alternatif untuk menangani permasalahan tersebut, salah satunya dengan cara kompresi (Winanti, 2006).

Kompresi data berarti suatu teknik untuk memampatkan data agar diperoleh data dengan ukuran yang lebih kecil daripada ukuran aslinya sehingga lebih efisien dalam menyimpannya serta mempersingkat waktu pertukaran data tersebut (Inti, 2006). Dengan kata lain, kompresi data sebenarnya adalah proses meminimalkan ukuran data atau berkas dengan mengurangi data yang berulang, karena umumnya pada sebuah data sering terjadi pengulangan. Data yang telah dikompres agar bisa digunakan kembali harus dikembalikan lagi seperti semula. Proses pengembalian sebuah data yang terkompres menjadi seperti data aslinya disebut dengan dekompresi (Bytastudios, 2007).

Pada kompresi data, terdapat dua tipe macam kompresi, yaitu *lossless compression* dan *lossy compression*. Pada *lossless compresion*, semua informasi yang ada pada data akan kembali menjadi seperti aslinya dan tidak ada informasi yang hilang. Teknik ini biasanya digunakan untuk dokumen-dokumen, file *executable*, dan lainnya. Karena, kehilangan sebuah informasi merupakan hal yang fatal bagi file-file tersebut. Sedangkan pada *lossy compression*, tidak semua informasi yang ada akan kembali seperti semula. Hanya

informasi-informasi inti yang dikembalikan. Hal ini terjadi, karena pada *lossy compression* informasi-informasi yang tidak berguna akan dihilangkan. Walaupun ada informasi yang hilang, namun hal ini tidak terlalu disadari oleh pengguna. Teknik ini biasanya digunakan pada file video, gambar, suara yang mana file-file tersebut biasanya berukuran besar (Bytastudios, 2007).

Dengan adanya kompresi diharapkan dapat menghemat biaya serta waktu yang dikeluarkan guna menambah fasilitas media penyimpanan data pada komputer serta mempercepat proses transfer data. Banyak teknik algoritma yang dapat digunakan untuk melakukan pengkompresian sebuah file, sebagai contoh adalah algoritma *Huffman*, algoritma LZ (*Lempel-Ziv*), algoritma DMC (*Dinamyc Markov Compression*), *Block-Shorting Lossless, Run-Lenght*, *Shannon-Fano*, *Arithmetic*, PPM (*Prediction by Partial Matching*), *Burrows-Wheeler Block Sorting*, dan *Half Byte*, dll (Inti, 2006).

Beberapa *software* kompresi yang banyak digunakan para pengguna komputer dewasa ini diantaranya adalah WinZip (menghasilkan format .zip) dan WinRAR (menghasilkan format .rar). Selain itu, telah banyak penelitian yang dilakukan mengenai pengkompresian sebuah file dengan berbagai macam algoritma., misalnya Aplikasi Pohon Dalam Teknik Kompresi Data Dengan Algoritma Huffman dan Algoritma Huffman Kanonik (Atmavidya, Arif Nanda, 2007), Kompresi Teks dengan menggunakan Algoritma Huffman (Wardoyo, 2005), Penggunaan *Hidden Markov Model* untuk Kompresi Kalimat (Wibisono, 2008).

Algoritma Huffman adalah salah satu algoritma kompresi yang paling terkenal untuk pengkompresian file teks (Wardoyo, 2005). Terdapat tiga fase dalam menggunakan algoritma Huffman, pertama adalah fase pembentukan pohon Huffman, kedua fase *encoding* dan ketiga fase *decoding*. Prinsip yang digunakan oleh algoritma Huffman adalah karakter yang sering muncul di -*encoding* dengan rangkaian bit yang pendek dan karakter yang jarang muncul di-*encoding* dengan rangkaian bit yang lebih panjang (Atmavidya, 2007).

Dalam penelitian ini, algortima Huffman diterapkan dengan berbasis pada model Bigram yang merupakan sebuah bentuk yang terdiri dari dua karakter (Hatem, 2007), kode bit biner dari tiap simpul yang terbentuk dari pohon Huffman digunakan untuk

merepresentasikan satu pasangan karakter, dimana tiap pasangan karakter terdiri dari dua karakter yang berdekatan pada sebuah file. Berbeda dengan algoritma Huffman pada umumnya, dimana kode bit biner yang terbentuk digunakan untuk menggantikan tiap satu karakter (simbol) dalam data (Wardoyo, 2005).

Maka berdasarkan latar belakang yang telah dikemukakan di atas, judul yang diambil dalam Skripsi ini adalah **"Pengkompresian File dengan Menerapkan Algoritma Huffman Berbasis Model Bigram".**

1.2 Rumusan Masalah

Berdasarkan uraian pada latar belakang masalah, maka rumusan masalah dalam skripsi ini adalah:

1. Bagaimanakah bentuk Kode Huffman dengan berbasis Bigram untuk pengkompresian sebuah file?
2. Bagaimanakah bentuk struktur data dari Model Huffman dengan berbasis Bigram untuk pengkompresian sebuah file?
3. Bagaimanakah kinerja dari sistem pengkompresian sebuah file dengan menerapkan algoritma Huffman berbasis model Bigram?

1.3 Tujuan Penelitian

Dari rumusan masalah yang telah dipaparkan di atas maka tujuan penelitian ini adalah:

1. Membuat bentuk kode Huffman berbasis Bigram untuk pengkompresian sebuah file.
2. Membuat struktur data dari model Huffman yang berbasis Bigram untuk pengkompresian sebuah file.
3. Mengukur kinerja dari sistem pengkompresian sebuah file dengan menerapkan algoritma Huffman berbasis model Bigram.

1.4 Batasan Masalah

Untuk mencegah tidak meluasnya permasalahan dan tercapainya Skripsi ini, berikut ini diberikan beberapa batasan masalah:

1. Pengkompresian hanya dilakukan pada sebuah file bukan sebuah folder yang berisi lebih dari satu file.

2. Data yang digunakan dalam penelitian ini adalah file yang berekstensi **.doc*, **.txt, *.html* , dan **.bmp*.

1.5 Manfaat Penelitian

Menyediakan aplikasi yang dapat digunakan untuk pengkompresian sebuah file sehingga dapat memberikan keuntungan pada proses penyimpanan dan komunikasi data, yang kemudian hasil data terkompres tersebut dapat dikembalikan seperti file aslinya dengan proses dekompresi sehingga dapat digunakan sesuai dengan kebutuhan.

1.6 Metodologi Pemecahan Masalah

Untuk mencapai tujuan yang dirumuskan sebelumnya, maka metodologi yang digunakan dalam penulisan skripsi ini adalah:

1. Studi Literatur
   Mempelajari teori-teori yang berhubungan dengan konsep kompresi file. Disamping itu juga mempelajari *software* kompresi yang sudah ada, sebagai acuan dalam perencanaan dan pembuatan Skripsi.
2. Pendefinisian dan analisis masalah
   Mendefinisikan dan menganalisis masalah untuk mencari solusi yang tepat.
3. Perancangan dan implementasi sistem
   Membuat perancangan aplikasi dengan analisis terstruktur dan mengimplementasikan hasil rancangan tersebut yaitu membuat aplikasi kompresi file.
4. Uji coba dan analisa hasil implementasi
   Menguji aplikasi, dan menganalisa hasil dari implementasi tersebut apakah sudah sesuai dengan tujuan yang dirumuskan sebelumnya, untuk kemudian dievaluasi dan disempurnakan.

1.7 Sistematika Penulisan

Skripsi ini disusun berdasarkan sistematika penulisan sebagai berikut:

1. BAB I PENDAHULUAN

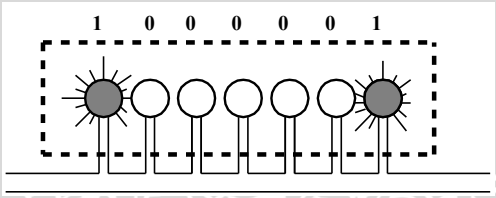
Berisi latar belakang masalah, perumusan masalah, tujuan penelitian, batasan masalah, manfaat penelitian, metodologi pemecahan masalah, dan sistematika penulisan.

2. BAB II TINJAUAN PUSTAKA
   Menguraikan teori-teori yang berhubungan dengan kompresi file.
3. BAB III METODOLOGI DAN PERANCANGAN
   Pada bab ini akan dijelaskan mengenai metode-metode yang digunakan dalam pembuatan aplikasi kompresi file.
4. BAB IV IMPLEMENTASI DAN PEMBAHASAN
   Pada bab ini akan dilakukan implementasi sistem, pengujian dan analisa sistem aplikasi yang dibangun, yaitu apakah menghasilkan aplikasi kompresi file yang efektif dan apakah proses dekompresi dapat mengembalikan data yang sama seperti file aslinya.
5. BAB V KESIMPULAN DAN SARAN
   Berisi kesimpulan dari seluruh rangkaian penelitian serta saran kemungkinan pengembangannya.

BAB II
TINJAUAN PUSTAKA

2.1 Data

Kata "data" diadopsi dari bahasa Inggris dan berasal dari kata Yunani "*datum*" yang berarti "fakta". Data di komputer memiliki ukuran dalam penyebutannya. Data terkecil di komputer disebut dengan *bit*, yaitu sinyal elektronik yang melewati suatu rangkaian digital (prosesor) komputer. *Bit-bit* tersebut selanjutnya dirangkai, dan rangkaian tersebut diberi kode lagi yang disebut dengan *character* (Wahyudi, 2008).



Gambar 2.1 Penggambaran *Bit-bit* yang Membentuk Karakter (Wahyudi, 2008)

2.1.1 Representasi Data

Elemen-elemen data secara umum adalah terdiri dari (Handayani, 2001):

1. Karakter

Karakter merupakan elemen data yang paling kecil. Karakter merupakan lambang-lambang yang terdiri dari huruf, angka, serta lambang-lambang lainnya, yang dibentuk dari susunan bit. Kode yang dihasilkan dari ASCII dapat diolah oleh komputer menjadi informasi yang disebut dengan karaker tadi, sehingga manusia pun dapat membacanya. Setiap karakter yang dibentuk melalui kode komputer disebut juga dengan satu byte. Satu byte adalah merupakan sebuah karakter yang dibangun dari tujuh atau delapan bit. Satuan yang digunakan untuk menunjukkan kapasitas dalam dunia digital, termasuk komputer, besar file, serta ukuran lain, digunakan dalam satuan byte ini. Informasi yang akan dan yang diolah disimpan di dalam suatu memori.

2. Field

Level data lebih tinggi adalah field yang berisi sekumpulan karakter. Field kadang-kadang juga disebut sebagai suatu item. Field dari sebuah data menggambarkan atribut dari beberapa entitas.

3. Record

Hubungan field-field data yang digabungkan menjadi bentuk satu record. Suatu record menggambarkan kumpulan atribut yang menggambarkan entitas.

4. File

Sekumpulan Record yang saling berhubungan dikenal sebagai file data. File adalah arsip yang disimpan dalam suatu media, yang terdiri dari kumpulan karakter, dan didokumentasikan dalam bentuk data digital oleh komputer. Atau dengan kata lain, file adalah entitas dari data yang disimpan di dalam sistem berkas yang dapat diakses dan diatur oleh pengguna. Sebuah berkas memiliki nama yang unik dalam direktori di mana file berada.

2.1.2 Jenis File

File data dapat digolongkan menurut jenisnya, diantaranya adalah (Handayani, 2001) :

1. File Induk
   Suatu file yang diperlukan untuk memperlancar operasi sistem dan diperbaharui secara teratur.
2. File Transaksi
   Digunakan untuk memperbaharui file induk dengan informasi yang baru.
3. File Penunjang
   Merupakan kutipan dari file untuk menjaga kemungkinan adanya file asli yang rusak.
4. File Riwayat Hidup
   Informasi yang dikumpulkan selama periode waktu tertentu dan biasanya digunakan untuk menyusun laporan statistik.
5. File Data Transaksi
   Merupakan sebuah rekaman pada pita atau piringan untuk setiap transaksi yang melengkapi file induk.

6. File Kesalahan
   Rekaman tentang kesalahan yang disimpan pada file untuk pembetulan di kemudian hari.
7. File Laporan
   Merupakan turunan laporan tercetak yang ditahan pada piringan atau pita menunggu printer siap mencetak.
8. File Sementara
   File yang disiapkan untuk pemrosesan peralihan.
9. File Pustaka (*Library*)
   File yang digunakan untuk menyimpan program aplikasi, program utiliti, dan program lain,.
10. File Kerja (*Work*)
    File ini berisi record-record yang disusun sedemikian rupa sehingga dapat dibuat sebuah program dan dipakai oleh program lain sebagai *input.*
11. File Program
    File ini berisi perintah-perintah untuk memproses data. Perintah dapat ditulis dalam bahasa pemrograman, bahasa rakitan atau bahasa mesin.

2.1.3 Ekstensi File

Setiap file dalam media penyimpanan memiliki tanda pengenal atau ciri-ciri yang menyatakan jenis file tersebut. Umumnya pengenal tipe file tertera pada nama file tersebut, yaitu tiga huruf paling kanan setelah titik. Fungsinya adalah untuk mengetahui atau membedakan jenis file. Beberapa jenis ekstensi file ditunjukkan pada tabel 2.1 berikut (Prayogo, 2008).

Tabel 2.1 Jenis-jenis ekstensi file

Ekstensi	Jenis	Aplikasi yang digunakan
asm	Source code pemrograman Assembly	Sembarang teks editor, seperti MS Word, Notepad
bat	Teks	MS Word, Notepad, Wordpad, Edit (pada DOS Prompt)

Tabel 2.1 Jenis-jenis ekstensi file (lanjutan)

Ekstensi	Jenis	Aplikasi yang digunakan
bmp	image	Sembarang editor, seperti PhotoShop, Paint, dll.
doc	Document MS Word	MS Word
exe	Aplikasi	Merupakan file aplikasi
fon	File font	Font viewer
Htm, html, shtml	Internet document	Netscape, navigator, MS Internet Explorer, dll.
gif	Image, animasi	Sembarang image editor, sedangkan untuk membuat animasinya digunakan Ulead Gif Animator, dll.
ico	File icon	microangelo
Jpg/jpeg	Image	Sembarang image editor, seperti PhotoShop, Paint
Mp3	Audio	WinAmp
pas	Source code bahasa pemrograman Pascal/Delphi	Sembarang teks editor
pdf		Adobe Acrobat Reader
psd	Image	Adobe Photoshop
txt	Teks	Sembarang teks editor
zip	File kompresi	WinZip, WinRar

2.2 Kode ASCII

Kode Standar Amerika untuk Pertukaran Informasi atau ASCII (*American Standard Code for Information Interchange*) merupakan suatu standar internasional dalam kode huruf dan simbol yang bersifat universal yang selalu digunakan oleh komputer dan alat komunikasi lain untuk menunjukkan teks. Kode ASCII sebenarnya memiliki komposisi bilangan biner sebanyak 8 bit. Dimulai dari 00000000 hingga 11111111. Total kombinasi yang dihasilkan sebanyak 256, dimulai dari kode 0 hingga 255 dalam sistem bilangan Desimal. Daftar beberapa karakter dalam ASCII terdapat pada lampiran 1 (Tabel ASCII, 2008).

2.3 Pohon Biner *(Binary Tree)*

2.3.1 Definisi Pohon

Definisi formal dari pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit (Winanti, 2006). Struktur pohon adalah struktur data yang penting di bidang informatika dan pemrograman. Struktur pohon memungkinkan untuk mengorganisasi informasi berdasarkan suatu struktur lojik dan memungkinkan bebagai cara akses untuk suatu elemen.

2.3.2 Sifat Pohon

Misalkan $G = (V, E)$ adalah graf tak-berarah sederhana dan jumlah simpulnya n. Maka, semua pernyataan di bawah ini adalah ekivalen:

1. G adalah pohon
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
3. G terhubung dan memiliki $m = n - 1$ buah sisi.
4. G tidak mengandung sirkuit dan memiliki $m = n - 1$ buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
6. G terhubung dan semua sisinya adalah jembatan.

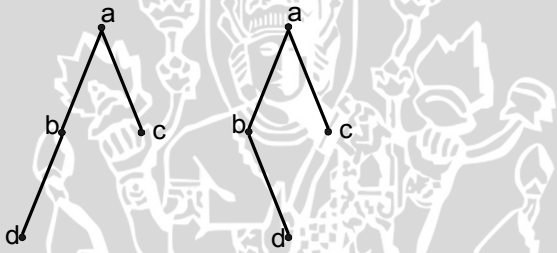
Beberapa istilah penting dalam pohon adalah :

- Hutan : kumpulan pohon yang saling lepas
- Anak (*child*) : subpohon dari sebuah akar
- Orangtua (*parent*) : akar dari sebuah subpohon
- Upapohon (*subtree*) : upagraf sebuah pohon sedemikian hingga upagraf tersebut mengandung x dan semua keturunannya
- Derajat (*degree*) : jumlah upapohon (jumlah anak) pada suatu simpul
- Ketinggian (*level*) : panjangnya jalan dari akar sampai dengan simpul yang bersangkutan
- Daun (*leaf*) : simpul terminal dari pohon
- Simpul (*node*) : elemen dari pohon yang memungkinkan akses pada subpohon dimana simpul tersebut berfungsi sebagai akar

- Kedalaman (*depth*) : panjang maksimum jalan dari akar menuju ke sebuah daun

2.3.3 Pohon biner

Pohon biner adalah pohon yang setiap simpulnya memiliki paling banyak dua buah anak yaitu kiri (*left*) dan kanan (*right*). Alih-alih menyebutnya anak pertma dan anak kedua dari suatu simpul dalam, dapat disebut anak kiri (*left child*) dan anak kanan (*right child*). Pohon yang akarnya adalah anak kiri disebut upapohon kiri (*left subtree*), sedangkan pohon yang akarnya adalah anak kanan disebut upapohon kanan (*right subtree*). Karena adanya perbedaan anak/upapohon kiri dan anak/upapohon kanan, maka pohon biner adalah pohon terurut.

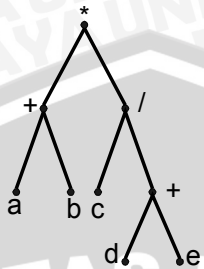


Gambar 2.2 Contoh dua pohon biner yang berbeda
(Winanti, 2006)

2.3.4 Terapan Pohon Biner

1. Pohon Ekspresi

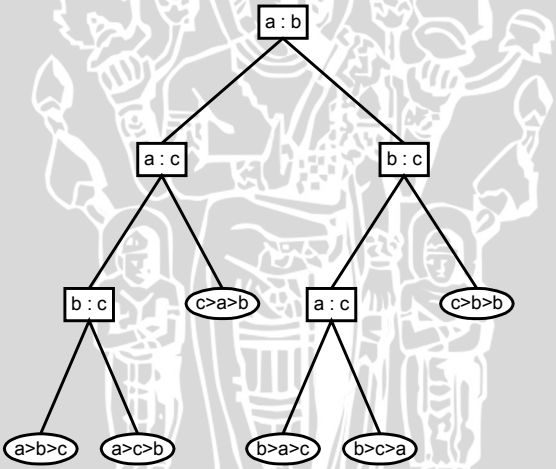
Pohon ekspresi ialah pohon biner dengan daun berupa *operand* dan simpul dalam (termasuk akar) berupa operator.



Gambar 2.3 Contoh pohon ekspresi dari *(a + b)*(c/(d + e))*
(Winanti, 2006)

2. Pohon Keputusan

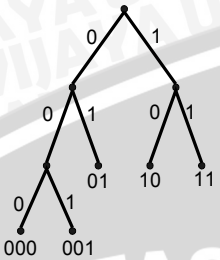
Pohon keputusan digunakan untuk memodelkan persoalan yang terdiri dari serangkaian keputusan yang mengarah ke solusi. Tiap simpul dalam menyatakan keputusan, sedangkan daun menyatakan solusi.



Gambar 2.4 Pohon keputusan untuk mengurutkan 3 buah elemen
(Winanti, 2006)

3. Kode Awalan

Kode awalan (*prefix code*) adalah himpunan kode, misalnya kode biner, sedemikian sehingga tidak ada anggota kumpulan yang merupakan awalan dari anggota yang lain.



Gambar 2.5 Pohon biner dari kode prefiks { 000, 001, 01, 10, 11} (Winanti, 2006)

4. Kode Huffman

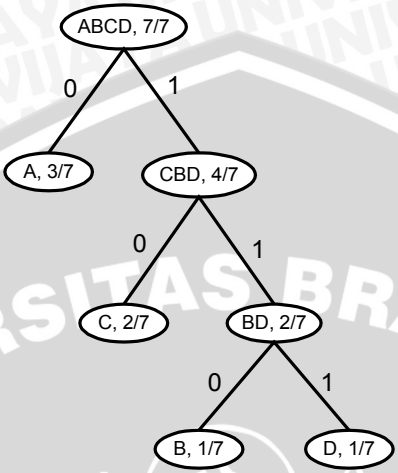
Kode Huffman merupakan suatu teknik yang dapat digunakan untuk mengkonstruksi kode prefiks. Contoh rangkaian bit untuk string '*ABACCDA'* berdasarkan kode ASCII adalah:

01000001010000010010000010100000110100000110100010001000001

Tabel 2.2 Tabel kekerapan dan kode Huffman untuk *string 'ABACCDA'*

simbol	kekerapan	peluang	Kode Huffman
A	3	3/7	0
B	1	1/7	110
C	2	2/7	10
D	1	1/7	111

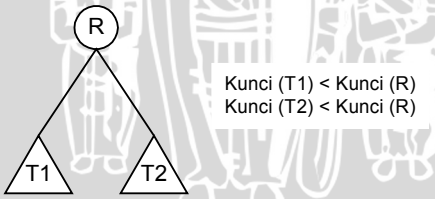
Jadi, rangkaian bit kode Huffman untuk *'ABACCDA'* adalah 0110010101110



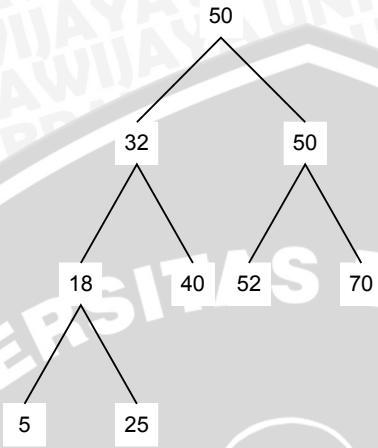
Gambar 2.6 Pohon Huffman untuk pesan *'ABACCDA'*
(Winanti, 2006)

5. Pohon Pencarian Biner

Pohon pencarian biner (*binary search tree* – BST) mungkin adalah pohon biner yang paling penting, khususnya pada persoalan yang banyak melakukan operasi pencarian, penyisipan, dan penghapusan elemen.



Gambar 2.7 Skema Pohon Pencarian
(Winanti, 2006)



Gambar 2.8 Contoh Pohon Pencarian
(Winanti, 2006)

2.4 Kompresi Data

2.4.1 Definisi

Kompresi data berarti sebuah proses mengkodekan informasi menggunakan bit atau *information-bearing unit* yang lain yang lebih rendah daripada representasi data yang tidak terkodekan dengan suatu sistem encoding tertentu (Anton, 2005).

Kompresi data juga diartikan sebagai suatu teknik untuk memampatkan data agar diperoleh data dengan ukuran yang lebih kecil daripada ukuran aslinya sehingga lebih efisien dalam menyimpannya atau mempersingkat waktu pertukaran data tersebut (Inti, 2006). Seperti diketahui bahwa sebuah data terkadang memiliki informasi yang berulang-ulang yang membuat ukuran sebuah data menjadi besar. Dengan menggunakan algoritma dan teknik-teknik tertentu, informasi yang sama dan berulang-ulang tersebut dikodekan sedemikian rupa sehingga data tersebut menjadi berukuran lebih kecil. File atau data yang sudah dikompres agar bisa digunakan kembali harus dikembalikan lagi seperi semula. Proses pengembalian sebuah file yang terkompres menjadi seperti file aslinya disebut proses dekompresi (Byta, 2007).

Pengiriman data hasil kompresi dapat dilakukan jika pihak pengirim/yang melakukan kompresi dan pihak penerima memiliki

aturan yang sama dalam hal kompresi data. Pihak pengirim harus menggunakan algoritma kompresi data yang sudah baku dan pihak penerima juga menggunakan teknik dekompresi data yang sama dengan pengirim sehingga data yang diterima dapat dibaca/di-*dekode* kembali dengan benar (Anton, 2005).

Misalnya terdapat kata "Hari ini adalah hari Jum'at. Hari Jum'at adalah hari yang menyenangkan". Jika kita telaah lagi, kalimat tersebut memiliki pengulangan karakter seperti karakter pembentuk kata hari, hari Jum'at, dan adalah. Dalam teknik sederhana kompresi pada perangkat lunak, kalimat di atas dapat diubah menjadi pola sebagai berikut ”# ini $ %. % $ # ya@ menyena@kan”. Dimana dalam kalimat diatas, karakter pembentuk hari diubah menjadi karakter #, hari Jum'at menjadi %, adalah menjadi $, ng menjadi @. Saat berkas ini akan dibaca kembali, maka perangkat lunak akan mengembalikan karakter tersebut menjadi karakter awal dalam kalimat (Jando, 2005).

2.4.2 Jenis Teknik Kompresi

Teknik kompresi secara umum dapat diklasifikasikan menjadi tiga (Restyandito, 2008), yaitu:

1. *Entropy coding* adalah teknik kompresi yang menggunakan proses lossless. Tekniknya tidak berdasarkan pada media dengan spesifikasi dan karakteristik tertentu namun berdasarkan urutan data serta tidak memperhatikan semantik data.
2. *Source coding* adalah teknik kompresi dengan menggunakan proses lossy. Teknik ini berkaitan dengan data semantik (arti data) dan media.
3. *Hybrid coding* adalah teknik kompresi dengan menggunakan kombinasi atau gabungan dari *entropy coding* dan *source coding*.

Berdasarkan outputnya, teknik kompresi dapat dibedakan menjadi dua:

1. Teknik kompresi *Lossy*

Kompresi menggunakan *lossy*, beberapa bagian data asli hilang ketika berkas di *decoded*. Keuntungan dari algortima ini adalah bahwa rasio kompresi cukup tinggi. Hal ini dikarenakan cara algoritma *lossy* yang mengeliminasikan beberapa data dari suatu berkas. Namun data yang dieliminasikan biasanya adalah data yang

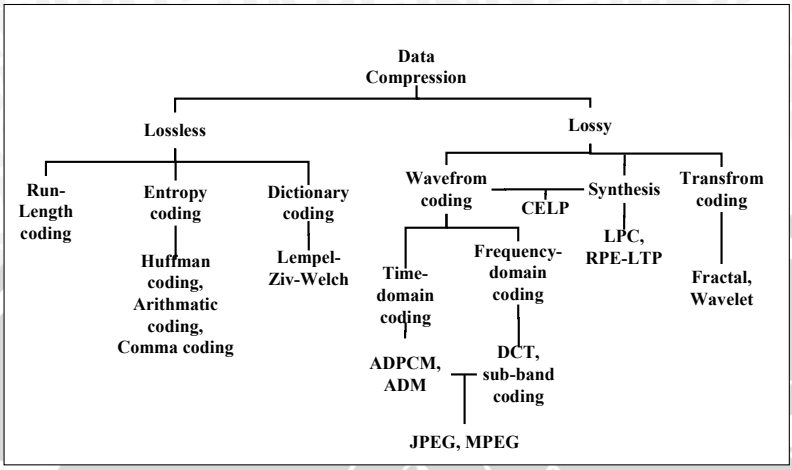
kurang diperhatikan atau diluar jangkauan manusia, sehingga pengeliminasian data tersebut kemungkinan besar tidak akan mempengaruhi manusia yang berinteraksi dengan berkas tersebut.

2. Teknik kompresi *Lossless*

Kompresi menggunakan *lossless* menjamin bahwa berkas yang dikompresi dapat selalu dikembalikan ke bentuk aslinya. Algoritma *lossless* digunakan untuk kompresi berkas *text*, seperti program komputer (berkas zip, rar, dll), karena kita ingin mengembalikan berkas yang telah dikompres ke status aslinya. Kadangkala ada data-data yang setelah dikompresi dengan teknik ini ukurannya menjadi lebih besar atau sama

Berdasarkan teknik pengkodean/pengubahan simbol yang digunakan, metode kompresi dapat dibagi ke dalam tiga kategori (Linawati dan Henry Panggabean, 2004), yaitu :

1. Metode *symbolwise* : menghitung peluang kemunculan dari tiap simbol dalam file input, lalu mengkodekan satu simbol dalam satu waktu, dimana simbol yang lebih sering muncul diberi kode lebih pendek dibandingkan simbol yang lebih jarang muncul. Contoh: algoritma *Huffman.*
2. Metode *dictionary* : menggantikan karakter/fragmen dalam file input dengan indeks lokasi dari karakter/fragmen tersebut dalam sebuah kamus (*dictionary*). Contoh: algoritma LZW.
3. Metode *predictive* : menggunakan model *finite-context* atau *finite-state* untuk memprediksi distribusi probabilitas dari simbol-simbol selanjutnya. Contoh: algoritma DMC.



Gambar 2.9 Klasifikasi dari beberapa teknik kompresi (Fauzi, 2003)

Jenis kompresi data berdasarkan mode penerimaan data oleh manusia (Anton, 2005), adalah:

1. *Dialoque Mode*: yaitu proses penerimaan data dimana pengirim dan penerima seakan berdialog (real time), seperti pada contoh *video conference.* Dimana kompresi data harus berada dalam batas penglihatan dan pendengaran manusia. Waktu tunda (delay) tidak boleh lebih dari 150 ms, dimana 50 ms untuk proses kompresi dan dekompresi, 100 ms mentransmisikan data dalam jaringan.
2. *Retrieval Mode*: yaitu proses penerimaan data tidak dilakukan secara real time. Dapat dilakukan *fast forward* dan *fast rewind* di client. Dapat dilakukan random access terhadap data dan dapat bersifat interaktif

Terdapat beberapa faktor yang sering menjadi pertimbangan dalam memilih suatu metode kompresi yang tepat karena tidak ada suatu metode kompresi yang paling efektif untuk semua jenis file. Faktor-faktor tersebut adalah:

1. Kualitas data hasil enkoding: ukuran lebih kecil, data tidak rusak untuk kompresi *lossy*.

2. Ketepatan proses dekompresi data: data hasil dekompresi tetap sama dengan data sebelum dikompres (kompresi *loseless*).
3. Kecepatan, ratio, dan efisiensi proses kompresi dan dekompresi.

2.4.3 Rasio Kompresi

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

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

$$\text{Rasio} = \frac{\text{ukuran file asli}}{\text{ukuran file terkompresi}} \qquad (2.1)$$

Jika dinyatakan dalam presentase maka dituliskan dalam persamaan berikut:

$$P = \left(1 - \frac{\text{ukuran file terkompresi}}{\text{ukuran file asli}}\right) * 100\ \% \qquad (2.2)$$

Yang berarti ukuran *file* berkurang sebesar *P* (dalam persentase) dari ukuran semula. Semakin tinggi rasio tingkat suatu teknik kompresi data maka semakin efektif teknik kompresi tersebut. Pada saat dikompresi, rasio kompresi akan berselang-seling berdasarkan pengaruh data terhadap algortima yang digunakan. Dengan demikian maka yang perlu diperhatikan adalah rasio kompresi rata-rata. Bukan pada rasio yang dicapai pada suatu waktu tertentu. Pada umumnya algoritma yang baik dapat mencapai presentase rasio kompresi rata-rata 1,5 atau jika dalam presentase sebesar 33%, bahkan saat ini sudah banyak algoritma yang menghasilkan rasio kompresi rata-rata lebih besar dari dua (Madenda, 1996).

2.5 Model Bigram

Bigram merupakan salah satu bagian dari n-gram yang mempunyai ukuran sebanyak dua. Sedangkan n-gram adalah sebuah bagian dari rangkaian sebanyak n materi, yang diberikan pada sebuah rangkaian tertentu. Materi yang dimaksud dalam n-gram dapat berupa karakter, kata, maupun keduanya tergantung dari aplikasi. N-gram biasa digunakan dalam perhitungan statistik language proceessing, sedangkan untuk ouputnya dapat digunakan untuk Statistik mesin penerjemah dan pengecekkan ejaan (Hatem, 2007).

Contoh model Bigram adalah sebagai berikut:

Kalimat : MAMA MASAK NASI
Bigram : MA, MA, _M, AS, AK, _N, AS, I

2.6 Algoritma Huffman

Algoritma Huffman dikembangkan oleh David Huffman, seorang mahasiswa MIT. Jika ditinjau dari segi teknik pengkodean karakter yang digunakan, algoritma Huffman ini termasuk dalam algoritma yang menggunakan metode *symbolwise*. Yang dimaksud dengan metode *symbolwise* adalah suatu metode yang menghitung probabilitas kemunculan suatu karakter dalam suatu waktu.

Kode Huffman merupakan *prefix code* yang berisi himpunan bit biner dari suatu karaker yang tidak mungkin menjadi awalan *(prefix)* dari himbunan bit biner karakter lainnya. Untuk karakter yang sering muncul akan dikodekan dengan rangkaian bit yang pendek, sedangkan untuk karakter yang jarang muncul akan dikodekan dengan bit yang lebih panjang. Kode *prefix* biasanya direpresentasikan dalam sebuah pohon biner, dimana setiap cabangnya memiliki suatu label nilai tertentu (Atmavidya, 2007).

Algoritma Huffman secara lengkap adalah sebagai berikut:

1. Pass pertama
   Baca (*scan*) *file* input dari awal hingga akhir untuk menghitung frekuensi kemunculan tiap karakter dalam file. *n* <- jumlah semua karakter dalam *file* input. *T* <- daftar semua karakter dan nilai peluang kemunculannya dalam file input. Tiap karakter menjadi node daun pada pohon Huffman.

2. Pass kedua
   Ulangi sebanyak (*n -1*) kali :

i. Item *m1 dan m2* <- dua subset dalam *T* dengan nilai peluang yang terkecil.

ii. ii. Gantikan *m1* dan *m2* dengan sebuah item {*m1,m2*} dalam *T*, dimana nilai peluang dari item yang baru ini adalah penjumlahan dari nilai peluang *m1* dan *m2*.

iii. Buat node baru {*m1, m2*} sebagai *father node* dari node *m1* dan *m2* dalam pohon Huffman.

3. *T* sekarang tinggal berisi satu item, dan item ini sekaligus menjadi node akar pohon Huffman. Panjang kode untuk suatu simbol adalah jumlah berapa kali simbol tersebut bergabung dengan item lain dalam *T*.

2.6.1 Algoritma Huffman Dinamis

Algoritma Huffman merupakan salah satu algoritma pemampatan yang banyak digunakan. Namun, pengkodean Huffman yang biasa (kadang disebut algoritma Huffman statis) memiliki beberapa kekurangan. Pertama, pembacaan terhadap berkas masukan dilakukan sebanyak dua kali, yaitu untuk mengetahui frekuensi setiap karakter yang muncul dan melakukan pengkodean untuk setiap karakter tersebut. Kedua, pohon yang dibuat untuk pengkodean harus turut disertakan pada berkas hasil pemampatan, sehingga hal ini memeperbesar ukuran berkas hasil pemampatan dan menurunkan rasio pemampatan.

Pengkodean Huffman Dinamis (*Dynamic Huffman Coding*) merupakan algoritma pengkodean yang dikembangkan dari pengkodean Huffman statis dengan memperbaiki kekurangan-kekurangan di atas. Perbaikan tersebut dilakukan dengan membuat pohon Huffman yang digunakan untuk pengkodean secara dinamis, tergantung karakter yang dibaca dari berkas masukan. Bobot tiap-tiap simpul dalam pohon diperbaharui pada setiap pembacaan satu karakter. Dapat dikatakan, pohon Huffman yang dibuat beradaptasi dengan karakter yang dibaca, sehingga algortima ini disebut juga Algoritma Huffman Adaptif (*Adaptive Huffman Coding*). Algoritma Huffman dinamis tertentu dapat menghasilkan kode bit yang lebih pendek daripada hasil yang didapat dari pengkodean Huffman statis atau dengan kata lain pengunaan kode Huffman yang bersifat dinamis akan lebih efektif bila dibandingkan dengan kode Huffman statis (Satrio Utomo, 2008).

2.7 Perangkat Lunak untuk Kompresi Data

Dalam dunia komputer, kompresi sudah bukan menjadi hal yang baru. Hal ini terbukti dengan telah banyaknya aplikasi-aplikasi kompresi yang dijual bebas di pasaran. Banyak perusahaan-perusahaan modem yang mengembangkan aplikasi kompresi menerapkan berbagai macam algoritma agar hasil kompresi dapat semaksimal mungkin. Aplikasi kompresi yang telah dikenal di pasaran diantaranya adalah WinZip dan WinRar.

2.7.1 WinZip

Ditemukan oleh Phil Katz untuk program PKZIP kemudian dikembangkan untuk WinZip, WinRAR, 7-Zip. Hasil kompresi dengan aplikasi ini menghasilkan *file* berekstensi *.zip dan MIME application/zip. WinZip Dapat menggabungkan dan mengkompresi beberapa *file* sekaligus menggunakan bermacam-macam algoritma, namun paling umum menggunakan Katz's Deflate Algorithm (Anton, 2005). Beberapa metode yang diterapkan dalam aplikasi WinZip adalah:

1. *Shrinking* : merupakan metode variasi dari LZW
2. *Reducing* : merupakan metode yang mengkombinasikan metode same byte sequence based dan probability based encoding.
3. *Imploding* : menggunakan metode byte sequence based dan Shannon-Fano encoding.
4. *Deflate* : menggunakan LZW
5. *Bzip2*, dan lain-lain

Dalam mengompres *file binary*, program zip dengan kompresi cepat dapat memepetkannya sehingga menjadi 1047175 byte (dari 83886080) atau menjadi sekitar 1.248330% dari ukuran aslinya. Kompresi dapat dilakukan dalam waktu 2 detik. Dalam mengompres *file binary*, program zip dengan daya kompresi paling tinggi dapat memepetkannya sehingga menjadi 662414 byte (dari 83886080) atau menjadi sekitar 0,789659% dari ukuran aslinya. Kompresi dilakukan dalam waktu 8 detik. Unzip untuk *file* binary dapat dilakukan dalam waktu sekitar 2 detik.

Dalam mengompres *file* teks, program zip dengan kompresi cepat dapat memepetkannya sehingga menjadi 1353968 *byte* (dari

310362112) atau menjadi sekitar 0,436254% dari ukuran aslinya. Kompresi dilakukan dalam 13 detik. Dalam mengompres *file* teks, program zip dengan daya kompresi paling tinggi dapat memepetkannya menjadi 301379 *byte* (dari 310362112) atau menjadi sekitar 0,097105% dari ukuran aslinya. Kompresi juga dilakukan dalam waktu 13 detik lebih sedikit. Unzip untuk *file* teks dilakukan dalam 13 detik (Noprianto, 2004).

2.7.2 WinRar

WinRar ditemukan oleh Eugene Roshal, sehingga RAR merupakan singkatan dari **R**oshal **Ar**chive pada 10 Maret 1972 di Rusia. Kompresi dengan WinRar menghasilkan *file* berekstensi .rar dan MIME *application/x-rar-compressed*. Proses kompresi WinRar lebih lambat jika dibandingkan dengan kompresi WinZip, akan tetapi ukuran *file* hasil kompresi lebih kecil. Selain menangani ekstensi *.rar, aplikasi WinRar juga mampu menangani *.zip dan mendukung *volume split*, enkripsi AES (Anton, 2005).

Dalam mengompres *file binary*, program rar dengan kompresi cepat dapat memepetkannya sehingga menjadi 372357 *byte* (dari 83886080) atau menjadi sekitar 0,443884% dari ukuran aslinya. Kompresi dilakukan dalam waktu 13 detik. Dalam mengompres file binary, program rar dengan daya kompresi paling tinggi dapat memepetkannya sehingga menjadi 317406 *byte* (dari 83886080) atau menjadi sekitar 0,378377% dari ukuran aslinya. Kompresi dilakukan dalam waktu 18 detik. Unrar untuk *file binary* dapat dilakukan dalam waktu sekitar 2 detik.

Dalam mengompres *file* teks, program rar dengan kompresi cepat dapat memepetkannya sehingga menjadi 151626 *byte* (dari 310362112) atau menjadi sekitar 0,048854% dari ukuran aslinya. Kompresi di lakukan dalam waktu 27 detik. Dalam mengompres file teks, program rar dengan daya kompresi paling tinggi dapat memepetkannya menjadi 151621 *byte* (dari 310362112) atau menjadi sekitar 0,048852% dari ukuran aslinya. Kompresi dapat dilakukan dalam waktu 37 detik. Unrar untuk *file* teks dilakukan dalam waktu 14 detik (Noprianto, 2004).

BAB III
METODOLOGI DAN PERANCANGAN

Pada bab ini akan dibahas mengenai metode dan tahap-tahap yang digunakan dalam pembuatan aplikasi untuk pengkompresian sebuah file dengan menggunakan algoritma *Huffman* berbasis model *Bigram.*

3.1. Data yang Digunakan

Pada penelitian ini data yang diambil sebagai obyek penelitian adalah data faktual dengan berbagai ukuran dari yang kecil sampai ukuran besar. Adapun data yang digunakan sebagai obyek penelitian adalah 10 buah file berekstensi **.doc*, 10 buah file berekstensi **.txt*, 10 buah file berekstensi **.html,* dan 10 buah file berekstensi **bmp.*

3.2. Rancangan Proses

Dalam proses yang dijalankan, masukan dari *user* berupa file akan akan melalui beberapa tahap. Terdapat dua proses utama yang terdapat dalam sistem ini, yaitu proses kompresi dan proses dekompresi. Berikut ini akan dijelaskan lebih terperinci masing-masing proses yang akan dijalankan oleh sistem ini.

3.2.1. Proses Kompresi

Seperti dijelaskan pada Subbab 2.4.1 bahwa kompresi data berarti sebuah proses mengkodekan informasi menggunakan bit atau *information-bearing unit* yang lain yang lebih rendah daripada representasi data yang tidak terkodekan dengan suatu sistem encoding tertentu. *Information-bearing unit* yang akan digunakan dalam penelitian ini adalah berupa kode Huffman yaitu sebuah kode yang terdiri dari kumpulan bilangan biner yang didapatkan dari pembentukan sebuah pohon Huffman. Langkah-langkah yang dilakukan pada proses kompresi adalah:

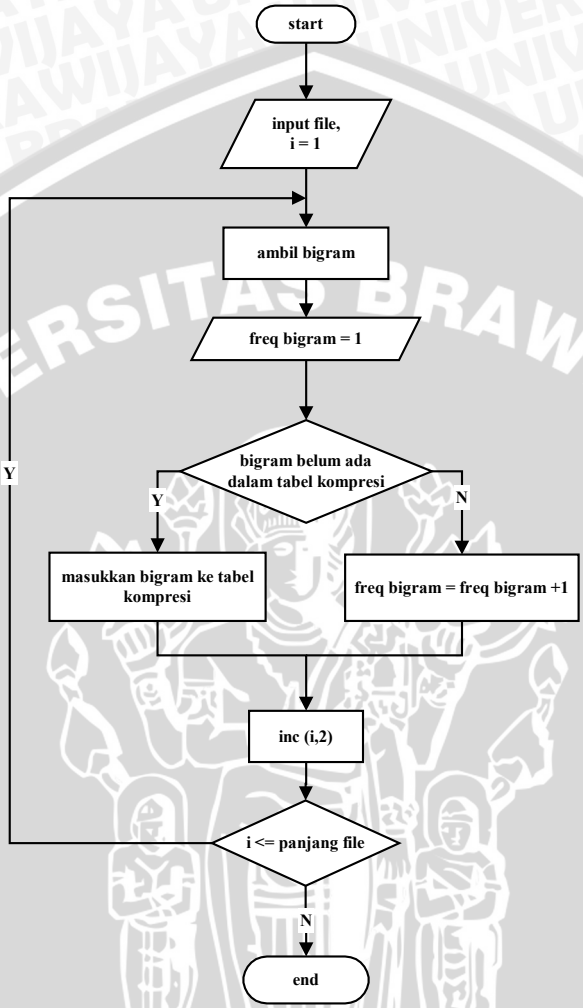
1. Pembentukan tabel kompresi

Tabel kompresi dibentuk guna memungkinkan encoding *prefiks* data yang terjadi berdasarkan peluang kemunculan karakter dalam data. Pada awalnya tabel kompresi ini berisi kumpulan pasangan karakter penyusun file beserta frekuensi kemunculannya. Sehingga tabel ini digunakan sebagai dasar pada pembentukan pohon

Huffman. Setelah didapatkan kode Huffman untuk masing-masing pasangan karakter, maka kode Huffman tersebut juga akan dimasukkan ke dalam tabel kompresi. Sehingga tabel kompresi berisi kumpulan pasangan karakter penyusun beserta kode Huffman-nya yang memungkinkan untuk proses *encoding*.

Tabel kompresi dibentuk dengan memanfaatkan struktur data berupa *pointer* yang menunjuk ke suatu *record*. Dalam pembentukan tabel kompresi ini dibutuhkan sebuah variabel yang bertipe *record*, dimana variabel ini berisi record nama (pasangan karakter), jumlah (frekuensi kemunculan) dan nilai (kode huffman untuk karakter yang bersangkutan).

Tabel kompresi yang dibentuk ini bersifat dinamis, artinya bahwa kapan saja komposisi karakter dalam data berubah maka akan diikuti dengan perubahan tabel kompresi. Seperti yang telah dijelaskan pada Subbab 2.6.1 bahwa kode Huffman yang bersifat dinamis akan menghasilkan kode yang lebih efisien. *Flowchart* dari proses pembentukan tabel kompresi ditunjukkan pada Gambar 3.1 berikut:

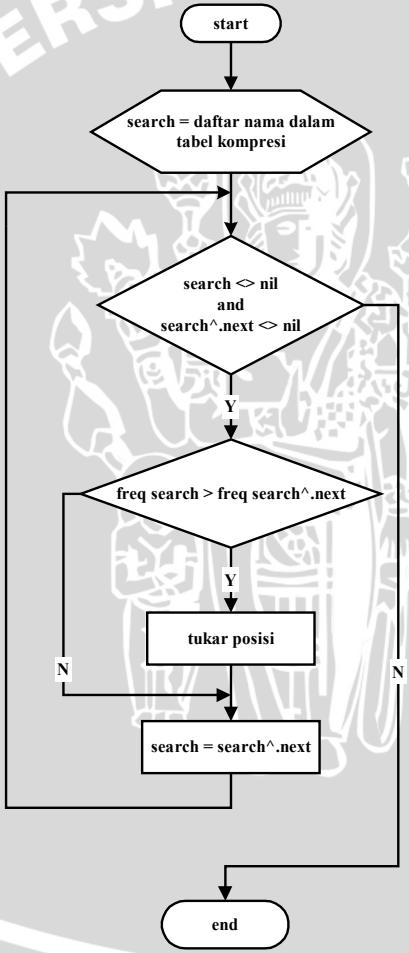


Gambar 3.1 *Flowchart* proses Pembetukan tabel kompresi

2. Pengurutan pasangan karakter

Sistem akan meminta masukan berupa file. Semua pasangan karakter yang ada dalam file akan dibaca untuk dihitung frekuensi kemunculannya. Setiap pasangan karakter penyusun data dinyatakan sebagai pohon bersimpul tunggal. Dan setiap simpul ini di-*assign* dengan peluang kemunculan karakter tersebut.

Sebelum dilakukan pembentukkan sebuah pohon Huffman dari string karakter data, terlebih dahulu dilakukan pengurutan terhadap karakter sesuai dengan kenaikan peluang kemunculannya. Pengurutan secara *ascending* dilakukan sesuai algoritma *Selection Sort* yaitu dengan membandingkan elemen yang sekarang dengan elemen berikutnya sampai ke elemen yang terakhir. Jika ditemukan elemen lain yang lebih kecil dari elemen yang sekarang maka dicatat posisinya dan langsung ditukar. *Flowchart* dari proses pengurutan pasangan karakter ditunjukkan pada Gambar 3.2.



Gambar 3.2 *Flowchart* proses *Sorting*

3. Pertimbangan dalam pembentukan Kode Huffman

Pertimbangan dalam pembentukan kode Huffman dilakukan agar terbentuk sebuah kode Huffman yang optimal. Aturan diterapakan jika ditemukan sebuah karakter tunggal atau tidak memiliki pasangan. Hal ini akan terdapat pada sebuah file yang frekuensi karakter-karakter penyusunnya berjumlah ganjil, sehingga ada satu karakter tunggal di akhir data. Karakter tunggal tersebut akan selalu memiliki frekuensi kemunculan sebesar satu.

Karakter tunggal yang terbentuk ini akan dianggap sebuah simpul tunggal. Dalam proses *sorting* karakter, simpul tunggal ini akan dianggap sebagai pasangan karakter dengan frekuensi terkecil dalam data. Sehingga dalam pembentukan pohon Huffman simpul ini akan terdapat dalam urutan paling kiri.

4. Pembentukan pohon Huffman

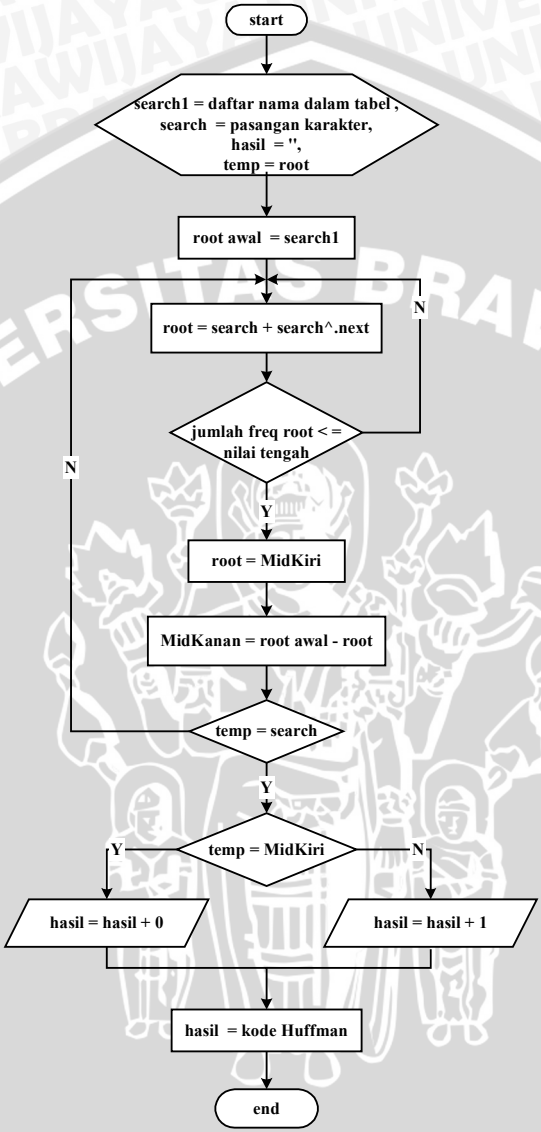
Berdasarkan pada Model Bigram, setiap kode Huffman akan digunakan untuk merepresentasikan setiap pasangan karakter penyusun data dimana setiap pasangan terdiri dari dua karakter yang berdekatan dalam data.

Pembentukan pohon Huffman dilakukan dengan memanfaatkan struktur data berupa *record,* dimana didalamnya terdapat *record* nama (pasangan karakter yang merupakan sebuah simpul tunggal), simpul kiri (berisi pasangan karakter yang terdapat pada simpul kiri) dan simpul kanan (berisi pasangan karakter yang terdapat pada simpul kiri). Aturan yang diterapkan adalah jika sebuah bigram terletak pada simpul kiri maka diinisialisasi dengan nilai 0, sedangkan jika sebuah bigram terdapat pada simpul kanan maka akan diinisialisasi nilai 1. Rangkaian bit yang terbentuk pada setiap lintasan dari akar (*root* awal) ke daun (sebuah bigram) merupakan kode Huffman untuk pasangan karakter yang berpadanan. Kode Huffman ini kemudian akan disimpan dalam tabel kompresi.

Simpul tunggal pada pohon Huffman yang terbentuk merepresentasikan sebuah bigram. Simpul-simpul tunggal pembentuk pohon Huffman telah disusun secara *ascending* berdasarkan pada masing-masing frekuensi kemunculannya. Pada awalnya setiap simpul dianggap sebagai simpul bebas. Akar dari pohon Huffman merupakan gabungan dari keseluruhan pasangan karakter penyusun. Nilai tengah dari jumlah frekuensi keseluruhan pasangan karakter didefinisikan sebagai :

$$Nilai\ tengah = \frac{\sum frekuensi\ bigram}{2} \qquad (3.1)$$

Tiap satu simpul tunggal akan digabungkan dengan simpul berikutnya. Jika jumlah frekuensi dari gabungan simpul tersebut kurang atau sama dengan nilai tengah, maka penggabungan simpul dihentikan, dan gabungan simpul tersebut akan menjadi simpul kiri dari akarnya. Sehingga simpul kanan merupakan gabungan simpul yang terdapat dalam akar yang bukan merupakan simpul kiri dari akar tersebut. Demikian seterusnya hingga daun dari pohon Huffman yang terbentuk adalah sebuah simpul tunggal yang menginisialisasi sebuah bigram. Flowchart dari pembentukan pohon Huffman ditunjukkan pada Gambar 3.3.

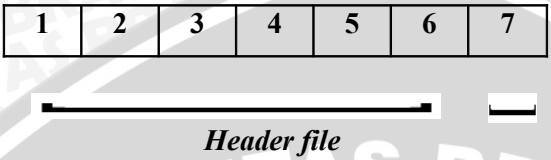


Gambar 3.3 *Flowchart* proses Pembentukan pohon Huffman

5. Pembentukan header file

Header file merupakan bagian dari data yang akan disimpan ke dalam file terkompresi. Header file berisi bagian-bagian yang

berfungsi sebagai pedoman atau kamus dalam proses dekompresi nantinya. Adapun header file ditunjukkan pada gambar 3.4 berikut:



Gambar 3.4 Mekanisme file terkompresi

Adapun keterangan untuk bagian-bagian pada gambar 3.4 adalah:

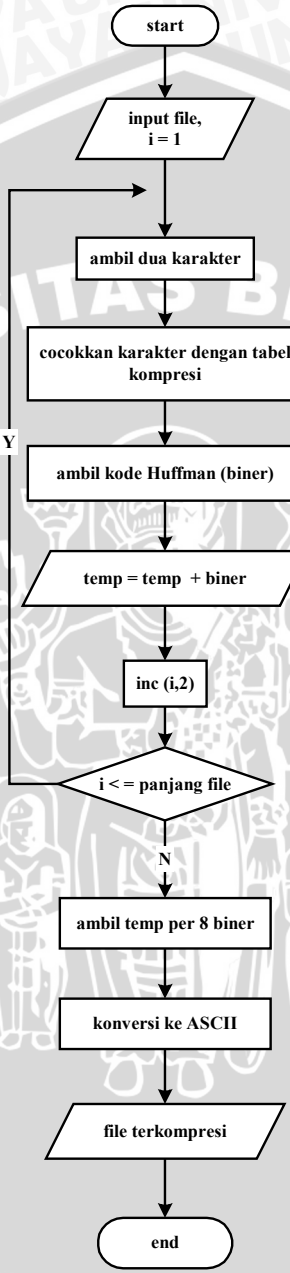
1. bagian satu berisi ekstensi **.Bhuf* yang merupakan ekstensi dari file terkompresi.
2. bagian dua berisi ekstensi dari file yang akan dikompresi.
3. bagian tiga berisi jumlah karakter pada header1.
4. bagian empat berisi header1 yang merupakan kumpulan pasangan karakter yang terdapat pada tabel kompresi.
5. bagian lima berisi header2 yang merupakan frekuensi secara urut dari masing-masing kumpulan karakter pada header1
6. bagian enam jumlah tambahan bilangan biner nol yang diperlukan untuk pengkonversian string biner ke dalam karakter ASCII.

Keterangan:

1. setiap bagian dari isi file tersebut diberi pembatas berupa karakter titik (.)
2. setiap frekuensi untuk masing-masing pasangan karakter pada header2 diberi pembatas berupa karakter strip (-), hal ini dilakukan agar tidak terjadi ambiguitas dalam proses pembacaan file untuk proses selanjutnya.

6. Proses *encoding*

Flowchart dari proses *encoding* ditunjukkan pada gambar 3.5 berikut:



Gambar 3.5 *Flowcahrt* proses *encoding*

Proses *encoding* bertujuan untuk menyusun *string* biner dari kumpulan karakter yang ada pada file. Kode untuk string biner diambil dari tabel kompresi yang telah dibentuk sebelumnya. Setiap pasangan karakter dalam string input akan dibandingkan dengan kode *prefiks* dalam tabel. Bila terjadi kecocokan maka kode Huffman diambil dari tabel kompresi tersebut dan digunakan untuk menggantikan karakter pada string data yang asli.

7. Penyimpanan file terkompresi

Dalam proses kompresi, hasil dari proses *encoding* adalah berupa *string* biner. Sebelum disimpan ke dalam file terkompresi, per delapan bit dari *string* biner ini akan terlebih dahulu dikonversi ke dalam karakter ASCII . Karena jika string biner hasil *encoding* tersebut langsung disimpan ke dalam file terkompresi, maka akan semakin menambah besarnya memori dari file tersebut. File hasil kompresi dari aplikasi ini akan disimpan dengan ekstensi **.Bhuf*.

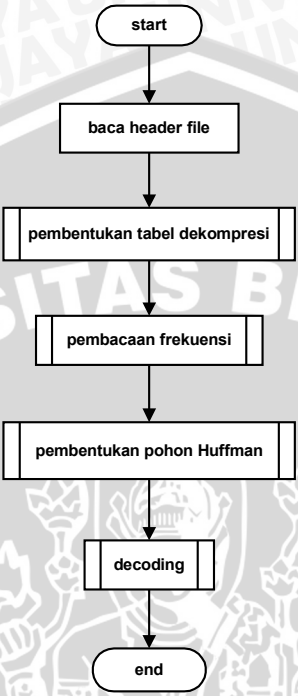
File **.Bhuf* tersusun atas header file dan data. Seperti telah dijelaskan sebelumnya bahwa header file merupakan kamus kompresi yang dibentuk yang akan digunakan sebagai kamus untuk menerjemahkan string biner menjadi karakter kembali dalam proses dekompresi. Sedangkan data dari file **.Bhuf* berisi karakter ASCII yang merupakan hasil konversi dari per delapan bit dari string biner hasil proses *encoding*. Jika panjang bilangan dari string biner ini tidak habis dibagi delapan, maka akan ditambahkan bilangan biner nol (0) di belakangnya. Sehingga string biner ini akan tepat habis jika dikonversi per delapan bit-nya ke dalam karakter ASCII.

3.2.2 Proses Dekompresi

Proses dekompresi adalah proses pengembalian sebuah file yang terkompres menjadi kembali seperti file aslinya. Adapun langkah-langkah yang dilakukan dalam proses dekompresi akan dijelaskan sebagai berikut:

1. Pembangkitan kode Huffman

Proses ini diawali dengan pembacaan file *input* yaitu berupa file terkompresi. Aplikasi yang akan dibuat dalam penelitian ini hanya bisa digunakan untuk melakukan proses dekompresi pada file **.Bhuf*. *Flowchart* dari proses pembangkitan kode Huffman ditunjukkan pada Gambar 3.6 berikut:



Gambar 3.6 *Flowchart* proses Pembangkitan kode Huffman

Proses pembangkitan kode Huffman ini akan melalui beberapa langkah, yaitu:

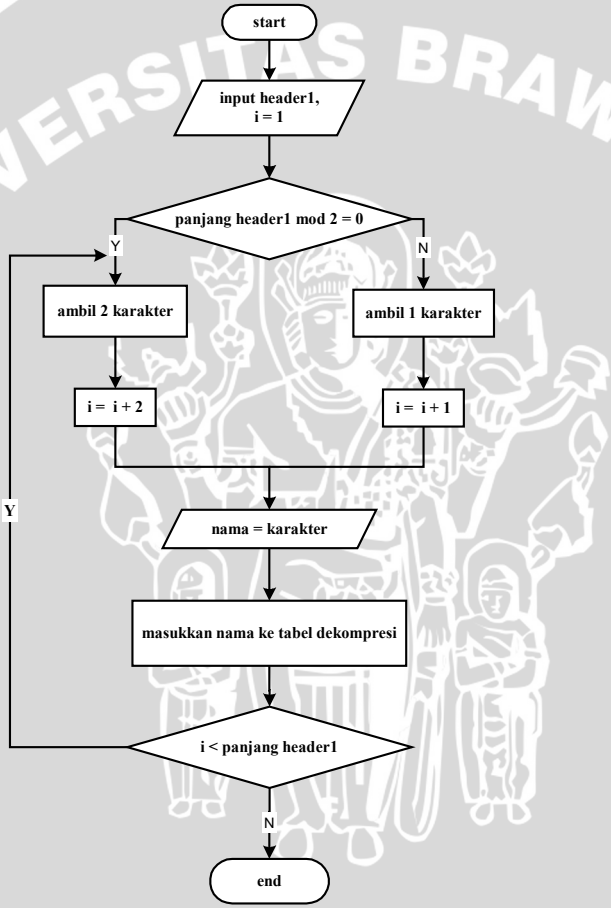
1.1 **Penerjemahan *header file***

Pembacaan awal dari *header file* adalah pada bagian dua, sehingga didapatkan ekstensi asli dari file terkompresi tersebut. Kemudian akan dilakukan pembacaan pada bagian tiga yaitu banyaknya karakter yang akan diambil sebagai header1 pada bagian selanjutnya. Dalam hal ini header1 berisi kumpulan pasangan karakter penyusun file.

1.2 Pembentukan tabel dekompresi

Tabel dekompresi ini pada awalnya berisi kumpulan pasangan karakter penyusun dari file yang didapatkan dari pembacaan pada header1. Jumlah karakter yang diambil adalah sesuai dengan angka yang terisi pada bagian tiga pada header file terkompresi. Jika jumlah karakter pada bagian tiga beriisi angka

ganjil maka satu karakter paling depan adalah merupakan pasangan karakter tunggal. Tetapi jika bagian tiga berisi angka genap, berarti kumpulan karakter pada header1 keseluruhannya adalah berupa pasangan dua karakter. Tabel dekompresi dibentuk guna memungkinkan proses *decoding*. *Flowchart* pembentukan tabel dekompresi ditunjukkan pada gambar 3.7.

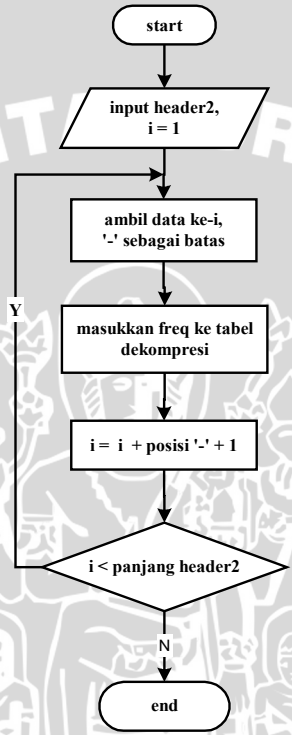


Gambar 3.7 *Flowchart* proses Pembentukan table dekompresi

1.3 Pembacaan frekuensi

Frekuensi dari masing-masing pasangan karakter didapatkan dari pembacaan header2 yaitu bagian lima pada *header file.*

Frekuensi kemunculan ini kemudian akan dimasukkan dalam tabel dekompresi guna pembentukan pohon Huffman. *Flowchart* proses pembacaan frekuensi ditunjukkan pada Gambar 3.8 berikut:



Gambar 3.8 *Flowchart* proses Pembacaan frekuensi

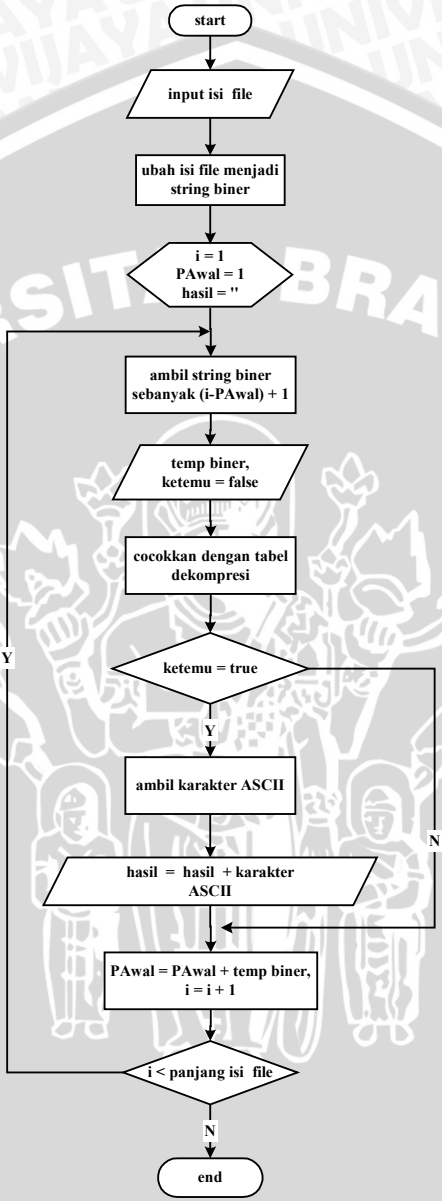
1.4 Pembentukan pohon Huffman

Aturan yang diterapkan pada pembentukan pohon Huffman dalam proses dekompresi ini adalah sama dengan aturan yang diterapkan pada pembentukan pohon Huffman pada proses kompresi. Dari pohon Huffman ini akan terbentuk kode Huffman untuk masing-masing pasangan karakter yang kemudian disimpan dalam tabel dekompresi. Kode Huffman untuk proses dekompresi adalah sama dengan kode Huffman pada proses kompresi. Aturan ini diterapkan agar proses dekompresi dapat mengembalikan file terkompresi tepat sesuai dengan file aslinya.

1.5 **Proses *decoding***

Decoding merupakan kebalikan dari *encoding*. Proses *decoding* bertujuan untuk menyusun kembali data dari *string* biner menjadi sebuah karakter kembali. Proses *decoding* dilakukan dengan membaca data pada bagian tujuh yang berupa kumpulan karakter ASCII. Kemudian kumpulan karakter ASCII ini akan dikonversi kembali ke dalam string biner. Setelah itu string biner hasil konversi tersebut akan dipotong dari belakang sebanyak angka yang dibaca dari bagian enam. Bagian yang dipotong ini merupakan bilangan biner '0' yang ditambahkan dalam proses pengkonversian ke dalam karakter ASCII. Sehingga bilangan biner ini bukan merupakan bagian dari isi file.

Proses *decoding* selanjutnya adalah membandingkan sisa string biner dari hasil pemotongan di atas dengan tabel dekompresi. Pembacaan dilakukan per bilangan biner, jika tidak terdapat kecocokan dalam tabel dekompresi maka dibaca satu bilangan biner berikutnya. Proses berlanjut hingga ditemukan kecocokkan antara string biner yang dibaca dengan yang terdapat pada tabel dekompresi. Jika terdapat kecocokan dalam kode Huffman maka pasangan karakter yang bersesuaian akan diambil dari tabel untuk menggantikan kode Huffman tersebut. *Flowchart* proses *decoding* ditunjukkan pada gambar 3.9 berikut:



Gambar 3.9 *Flowchart* proses *decoding*

2. Penyimpanan file terdekompresi

Setelah semua string biner telah diterjemahkan kembali kedalam karakter asli, maka akan dilakukan penyimpanan ke dalam sebuah file baru. File yang akan disimpan ini akan berekstensi sesuai dengan hasil pembacaan dari bagian dua.

3.2.3 Perhitungan waktu proses

Waktu yang dibutuhkan selama proses berlangsung merupakan hasil pengurangan dari waktu ketika proses dimulai dengan waktu ketika proses berakhir.

3.2.4 Perhitungan Rasio Kompresi

Perhitungan rasio kompresi digunakan untuk menguji tingkat keefektifan suatu teknik kompresi. Rasio kompresi akan dihitung berdasarkan pada persamaan 2.1. Rasio kompresi merupakan sebuah parameter pengukur dalam sebuah teknik kompresi. Semakin tinggi tingkat rasio suatu teknik kompresi, maka semakin efektif pula teknik kompresi tersebut.

3.2.5 Contoh Perhitungan Manual

Sebagai contoh, dalam kode ASCII kalimat ”MAMA MASAK NASI” membutuhkan representasi 10 × 8 bit = 80 bit (10 byte), dengan rincian sebagai berikut:

01001101	01000101	01010100	01000101	01001101	01000001
M	A	T	E	M	A

01010100	01001001	01001011	01000001
T	I	K	A

Untuk mengurangi jumlah bit yang dibutuhkan, panjang kode untuk tiap karakter dapat dipersingkat, terutama untuk karakter yang peluang kemunculannya besar.

Akan dilakukan pengkompresian terhadap kalimat ”MAMA MASAK NASI”. Pertama, akan dilakukan pengkompresian dengan menggunakan teknik Huffman biasa. Langkah awal adalah menghitung frekuensi kemunculan dari setiap karakter penyusun. Hasil perhitungan frekuensi diperlihatkan pada Tabel 3.1.

Langkah-langkah pembentukan pohon Huffman untuk kalimat ”MAMA MASAK NASI” ditunjukkan pada Gambar 3.10. Dari

pohon Huffman pada Gambar 3.10 tersebut maka kode Huffman yang terbentuk ditunjukkan pada tabel 3.2.

Tabel 3.1 Contoh Tabel Peluang Kemunculan Karakter Penyusun

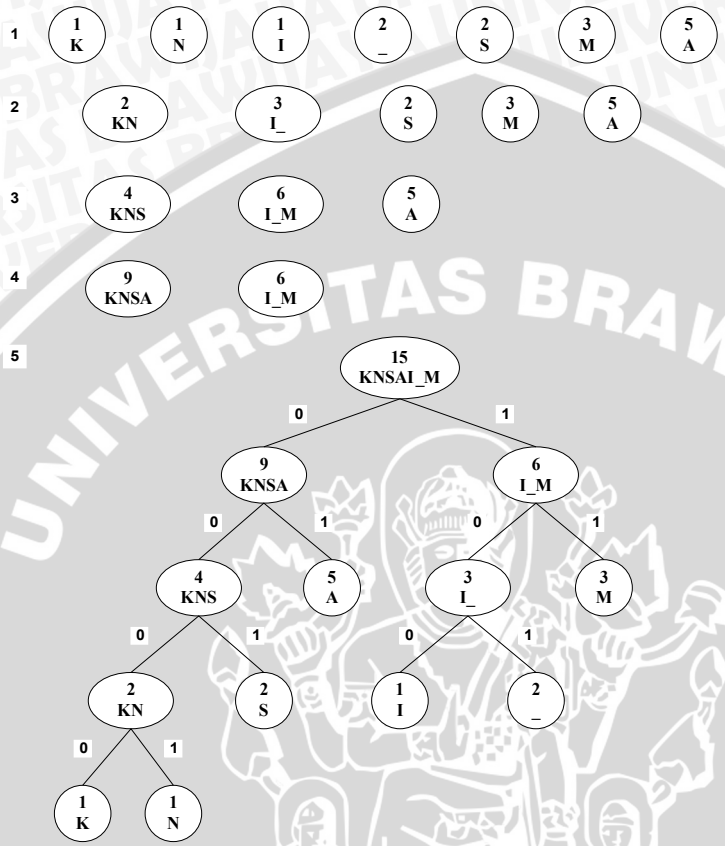
Karakter	Frekuensi
K	1
N	1
I	1
_ (SPASI)	2
S	2
M	3
A	5

Table 3.2 Kode Huffman yang terbentuk dari teknik Huffman biasa

Karakter	Kode Huffman
K	0000
N	0001
I	100
_ (SPASI)	101
S	001
M	11
A	01

Dengan menggunakan kode Huffman pada tabel 3.2 di atas, kalimat ”MAMA MASAK NASI” direpresentasikan menjadi rangkaian bit : **1101110110111010010100001010001010011100**.

Dengan menggunakan teknik Huffman biasa ini dibutuhkan 39 bit untuk menyimpan hasil pengkompresian kalimat ”MAMA MASAK NASI”.



Gambar 3.10 Pohon Huffman dengan teknik Huffman biasa

Kedua, akan dilakukan pengkompresian untuk kaliamat yang sama, akan tetapi dengan menggunakan teknik Huffman berbasis model Bigram yang diterapkan dalam penelitian ini. Langkah awal yang dilakukan sama dengan teknik pertama, yaitu melakukan penghitungan frekuensi, yang membedakan dengan algoritma Huffman biasa yaitu perhitungan frekuensi dilakukan terhadap masing-masing pasangan karakter penyusun file. Hasil perhitungan tersebut diperlihatkan pada tabel 3.3 berikut.

Tabel 3.3 Contoh Tabel Peluang Kemunculan Karakter Penyusun

Karakter	Frekuensi
I	1
_M	1
AK	1
_N	1
MA	2
AS	2

Langkah-langkah pembentukan pohon Huffman dari kalimat ”MAMA MASAK NASI” dengan menggunakan teknik Huffman berbasis Bigram ditunjukkan pada gambar 3.11. Dari pohon Huffman pada gambar 3.11 tersebut, maka dapat disimpulkan bahwa kode Huffman yang terbentuk adalah pada tabel 3.4 berikut:

Tabel 3.4 Kode Huffman yang terbentuk dengan teknik Huffman berbasis Bigram

Karakter	Kode Huffman
I	000
_M	001
AK	010
_N	011
MA	10
AS	11

Berdasarkan tabel 3.4 di atas maka kalimat ”MAMA MASAK NASI” direpresentasikan menjadi kumpulan bit biner yaitu **101000111010011111000**. Sehingga dengan teknik Huffman berbasis Bigram ini hanya dibutuhkan 20 bit untuk menyimpan kumpulan bit dari kalimat ”MAMA MASAK NASI”.

Dibandingkan dengan teknik Huffman biasa yang membutuhkan 39 bit, terbukti teknik Huffman berbasis Bigram ini membutuhkan jumlah bit yang lebih kecil yaitu 20 bit.

Gambar 3.11 Pohon Huffman dengan teknik Huffman berbasis Bigram

Contoh mekanisme penyimpanan file terkompresi dari kalimat ”MAMA MASAK NASI” adalah

BHuf.txt.11.I MAK NMAAS.1-1-1-1-2-2.4. £§▽

Keterangan:

1. bagian 1 berisi eksktensi ***.Bhuf***
2. bagian 2 berisi **txt**, hal ini menunjukkan bahwa kalimat ”MAMA MASAK NASI” adalah file yang berekstensi **.txt.*
3. bagian 3 berisi angka **11** (sebelas), hal ini berarti jumlah karakter pada header1 adalah sebanyak sebelas yaitu I MAK NMAAS.
4. bagian 4 adalah header1 yang berisi **I MAK NMAAS**

5. bagian 5 adalah header2 berisi frekuensi kemunculan secara terurut dari pasangan karakter yang terdapat pada header1
6. bagian 6 berisi angka empat, hal ini berarti dibutuhkan tambahan bilangan biner nol sebanyak empat agar string biner pada data dapat terkonversi per delapat bit-nya ke dalam karakter ASCII
7. bagian 7 berisi data hasil kompresi dari kalimat ”MAMA MASAK NASI” yaitu **£§▽,** ini merupakan hasil konversi per 8 bit dari string biner **10100011101001111000000**0 ke dalam karakter ASCII, dengan rincian sebagai berikut:

10100011	**10100111**	**10000000**
£	**§**	▽

Karena string biner hasil dari proses *encoding* terdiri dari 20 bilangan, maka dibutuhkan empat bilangan biner ’0’ sebagai tambahan agar string biner tersebut tepat habis dibagi delapan.

Proses dekompresi dilakukan agar file yang telah terkompresi dapat dikembalikan lagi seperti file aslinya agar dapat digunakan sesuai kebutuhan. Proses dekompresi diawalai dengan pembacaan header pada file **.Bhuf*.

Pembacaan awal adalah pada bagian tiga dan empat, yaitu mengambil sebanyak sebelas karakter pada header1. Sehingga didapatkan I_MAK_NMAAS. Karena bagian tiga berisi angka ganjil maka karakter terdepan pada header1 tersebut adalah merupakan pasangan karakter tunggal, dalam hal ini adalah karakter **I**. Sehingga didapatkan kumpulan pasangan karakternya adalah I, _M, AK, _N, MA, AS.

Setelah itu dilakukan pembacaan frekuensi kemunculan pada bagian lima. Sehingga didapatkan data berupa:

I = 1 _M = 1 AK = 1 _N = 1 MA = 2 AS = 2

Setelah didapatkan data diatas, proses selanjutnya adalah pembentukan pohon Huffman. Dari pohon Huffman tersebut didapatkan kode Huffman untuk masing-masing pasangan karakter. Karena aturan yang diterapkan dalam pembentukan pohon Huffman ini sama dengan pembentukan pada proses kompresi maka akan didapatkan pohon Huffman yang sama, yaitu ditunjukkan pada

Gambar 3.11. Dengan demikan kode Huffman untuk proses dekompresi adalah sama seperti yang tertera pada Tabel 3.4.

Proses selanjutnya adalah membaca data dari file terkompresi pada bagian tujuh. Sehingga didapatkan kumpulan karakter ASCII berupa ²¶ ▽. Kemudian kumpulan karakter ASCII ini dikonversi kembali kedalam bilangan biner, sehingga didapatkan **101000111010011110000000**. Sebelum proses *decoding* dilakukan, dilakukan pembacaan pada bagian enam, sehingga didapatkan angka empat. Dengan demikian string biner **101000111010011110000000** akan dipotong sebanyak empat bilangan dari belakang sehingga didapatkan string biner berupa **10100011101001111000**.

Selanjutnya proses decoding dapat dilakukan dengan membaca string biner **10100011101001111000**. Karena tiap kode Huffman yang dihasilkan unik, maka proses dekompresi dapat dilakukan dengan mudah. Contoh: saat dibaca kode bit pertama dalam rangkaian bilangan biner "**1**0100011101001111000", yaitu bilangan biner "1", ini akan dibandingkan dengan tabel dekompresi seperti yang tertera pada Tabel 3.4. Tidak ada kode Huffman "1", lalu dibaca bilangan biner selanjutnya, sehingga menjadi "10", kemudian akan dibandingkan kembali dengan tabel dekompresi. Rangkaian kode bit "10" adalah pemetaan dari pasangan karakter MA, sehingga didapatkan pasanga karakter pertaman penyusun file yaitu 'MA'.

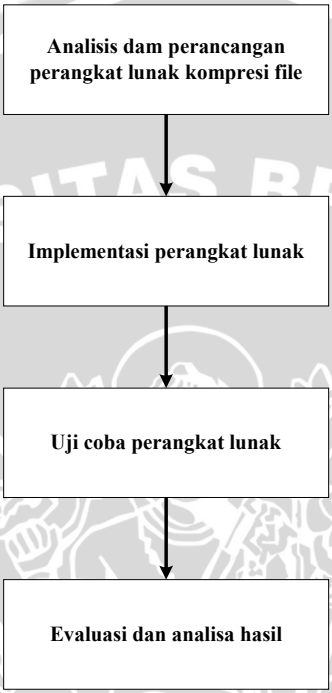
Kemudian dibaca kode bit selanjutnya yaitu bilangan biner "1", dalam tabel dekompresi tidak terdapat kode "1", maka dilakukan pembacaan kembali pada string biner, sehingga didapatakan bilangan biner "10". Kode biner "10" ini direpresentasikan untuk pasangan karakter MA kembali. Demikian seterusnya hingga semua string biner terbaca keseluruhan. Sehingga hasil dekompresi yang didapatkan adalah **MAMA MASAK NASI**.

3.3 Mekanisme Sistem

Secara garis besar tahapan pembuatan sistem yang akan dekembangkan adalah sebagai berikut:

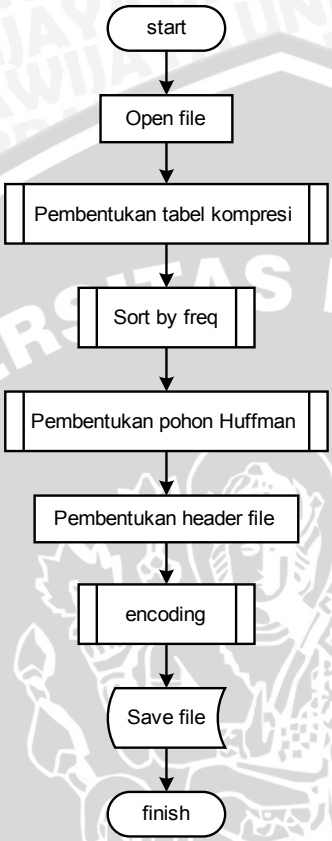
1. Melakukan studi literatur mengenai algoritma Huffman dan Model Bigram.
2. Menganalisis dan merancang perangkat lunak kompresi file.
3. Implementasi perangkat lunak berdasarkan analisis dan perancangan yang dilakukan.
4. Melakukan uji coba terhadap perangkat lunak.

5. Melakukan evaluasi hasil yang diperoleh dari uji coba perangkat lunak.



Gambar 3.12 Diagram alir pembuatan perangkat lunak

Sistem yang akan dikembangkan pada penelitian ini terdiri dari dua proses utama yaitu kompresi dan dekompresi data. *Flowchart* dari proses kompresi dan dekompresi ditunjukkan pada Gambar 3.13 dan Gambar 3.14 berikut.



Gambar 3.13 *Flowchart* proses kompresi

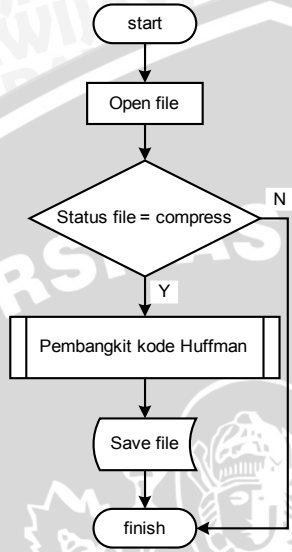
Langkah-langkah yang dilakukan oleh sistem ini ketika *user* melakukan proses kompresi adalah sebagai berikut:

1. Sistem akan dimulai dengan diinputkannya file *uncompress* (yaitu file yang bukan merupakan file terkompresi) oleh *user.*
2. Sistem akan membaca (me-*scan*) semua pasangan-pasangan karakter penyusun file untuk dimasukkan dalam tabel kompresi, dan disertai dengan penghitungan frekuensi dari pasangan karakter tersebut.
3. Agar terbentuk kode Huffman yang optimal maka pembentukan pohon Huffman diawali dengan pengurutan secara *ascending* terhadap pasangan-pasangan karakter penyusun berdasarkan frekuensi kemunculannya.

4. Rangkaian bit yang terbentuk pada setiap lintasan dari akar ke daun dari pohon Huffman merupakan kode Huffman untuk pasangan karakter yang berpadanan.
5. Kode Huffman yang dibentuk akan disimpan ke dalam tabel kompresi. Tabel kompresi yang dibentuk ini bersifat dinamis sesuai file yang di-*input*-kan.
6. Proses *encoding* dilakukan berdasarkan pada tabel kompresi yang terbentuk. Dengan membandingkan antara karakter asli dengan kode Huffman pada tabel dan jika terjadi kecocokkan maka kode tersebut diambil untuk menggantikan karakter yang berpadanan.
7. Hasil *output* dari proses kompresi adalah file dengan ekstensi **.Bhuf* , dan akan disimpan dengan nama dan direktori sesuai dengan yang telah ditentukan oleh *user*.

Langkah-langkah yang dilakukan oleh sistem ini ketika *user* melakukan proses dekompresi adalah sebagai berikut:

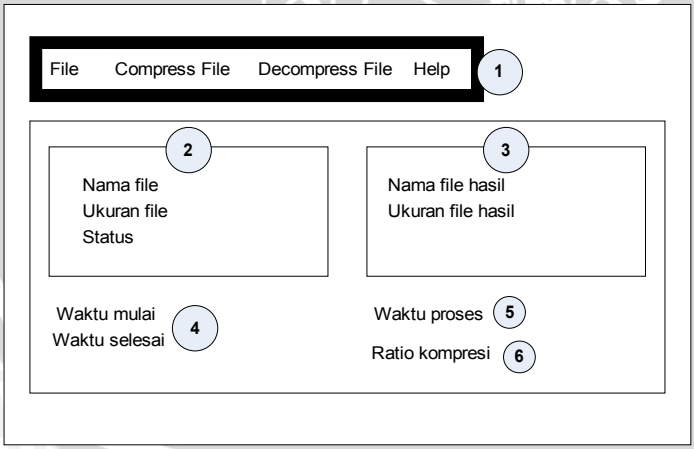
1. Sistem akan dimulai dengan diinputkannya file *compress* yang berektensi **.Bhuf* oleh *user*.
2. Sistem akan membaca (me-*scan*) header dari file dan akan menerjemahkannya sesuai aturan, kemudian akan dibentuk tabel dekompresi yang berisi semua pasangan-pasangan karakter penyusun file beserta frekuensi kemunculannya.
3. Dalam proses dekompresi akan dilakukan pembentukan kembali pohon Huffman guna mendapatkan kode Huffman. Pohon Huffman yang terbentuk berdasarkan tabel dekompresi adalah sama dengan pohon Huffman yang terbentuk dari proses kompresi untuk file yang bersangkutan, sehingga kode Huffman yang dihasilkan adalah sama. Hal ini memungkinkan bahwa proses dekompresi akan menghasilkan file yang sama dengan file aslinya.
4. Proses *encoding* dilakukan berdasarkan pada tabel dekompresi yang terbentuk. Dengan membandingkan antara string biner dengan daftar kode Huffman untuk pasangan karakter pada tabel dan jika terjadi kecocokkan maka pasangan karakter tersebut diambil untuk menggantikan stirng biner yang berpadanan.
5. Hasil *output* dari proses dekompresi adalah file dengan ekstensi sesuai dengan file aslinya, dan akan disimpan dengan nama dan direktori sesuai dengan yang telah ditentukan oleh *user*.



Gambar 3.14 *Flowchart* proses dekompresi

3.4 Rancangan Antarmuka

Antarmuka yang akan dibangun ditunjukkan pada gambar 3.15 berikut:



Gambar 3.15 Rancangan antarmuka aplikasi Pengkompresian file dengan menerapkan algoritma Huffman berbasis Model Bigram

Adapun keterangan bagian-bagian yang ada dalam Gambar 3.15 adalah:

1. Main menu yang berisi menu *File*, *Compress File*, *Decompress File* dan *Help*. Menu *File* digunakan untuk membuka file yang akan diproses. Menu *Compress File* adalah untuk melakukan proses kompresi dan menu *Decompress File* untuk melakukan proses dekompresi. sedangkan menu *Help* berisi petunjuk atau keterangan tentang aplikasi ini.
2. Frame 2 berisi keterangan tentang file *input* yang akan diproses baik kompresi maupun dekompresi, yaitu berupa nama file yang disertai direktori penyimpanannya, ukuran file dan status file (*compress* atau *uncompress*).
3. Frame 3 berisi keterangan tentang file *output* hasil dari proses kompresi maupun dekompresi, yaitu berupa nama file dan direktori penyimpanannya serta ukuran file.
4. Label yang menampilkan waktu ketika proses mulai dijalankan serta waktu ketika proses berakhir.
5. Label yang menampilakan waktu yang dibutuhkan selama proses berlangsung baik proses kompresi maupun dekompresi.
6. Label yang menampilkan rasio kompresi yang terbentuk.

3.5 Rancangan Uji Coba dan Evaluasi Hasil

Uji coba sistem kompresi terhadap file akan digunakan untuk melakukan evaluasi terhadap hasil kompresi dan dekompresi yang dihasilkan oleh sistem. Tujuannya yaitu untuk mengetahui apakah penerapan algoritma *Huffman* berbasis model *Bigram* ini dapat mengkompresi suatu file dengan efektif.

3.5.1 Rancangan Evaluasi

Pengujian yang akan dilakukan terhadap sistem adalah dengan menginputkan file yang berekstensi **.doc*, **.txt, *.html* , dan **.bmp.* Uji coba dilakukan terhadap file dengan berbagai ukuran. Hal ini bertujuan untuk mengetahui bagaimanakah pengaruh ukuran file terhadap hasil kompresi yang dilakukan.

3.5.2 Analisa dan evaluasi hasil

Output yang dihasilkan dari proses kompresi akan dievaluasi dengan memperhatikan ukuran file hasil kompresi, waktu yang

dibutuhkan untuk proses yang dijalankan serta rasio kompresi yang dicapai. Sedangkan *output* yang dihasilkan dari proses dekompresi akan dievaluasi dengan memperhatikan ukuran file hasil dekompresi, waktu yang dibutuhkan untuk proses yang dijalankan serta ketersesuaian antara file hasil dekompresi dengan file asli sebelum dilakukan proses kompresi.

Rancangan tabel untuk hasil dari uji coba yang dilakukan pada penelitian ini akan ditampilkan pada tabel 3.5 dan tabel 3.6.

Tabel 3.5 Rancangan tabel hasil uji coba untuk proses kompresi

Ukuran file asli	Ukuran file terkompresi	Waktu yang dibutuhkan	Rasio kompresi

Tabel 3.6 Rancangan tabel hasil uji coba untuk proses dekompresi

Ukuran file terkompresi	Ukuran file hasil dekompresi	Waktu yang dibutuhkan

3.5.3 Perbandingan Hasil Uji Coba dengan Hasil Kompresi WinZip dan WinRar

Hasil yang diperoleh dari uji coba yang dilakukan terhadap sistem akan dibandingkan dengan hasil dari proses kompresi yang dilakukan dengan WinZip dan WinRar. Perbandingan dapat dijabarkan dalam tabel 3.7, tabel 3.8, tabel 3.9, tabel 3.10 serta tabel 3.11. Perbandingan yang dilakukan adalah terhadap:

1. ukuran file hasil proses kompresi
2. ukuran file hasil proses dekompresi
3. waktu yang dibutuhkan selama proses berlangsung
4. rasio kompresi yang dicapai

Tabel 3.7 Rancangan tabel perbandingan ukuran hasil proses kompresi

No.	Nama file	Ekstensi file	Ukuran file asli	Ukuran file terkompresi		
				Huffman-Bigram	WinZip	WinRar

Tabel 3.8 Rancangan tabel perbandingan ukuran hasil proses dekompresi

No.	Nama file	Ekstensi file	Ukuran file terkompresi	Ukuran file hasil dekompresi		
				Huffman-Bigram	WinZip	WinRar

Tabel 3.9 Rancangan tabel perbandingan waktu proses kompresi

No.	Nama file	Ekstensi file	Waktu proses		
			Huffman - Bigram	WinZip	WinRar

Tabel 3.10 Rancangan tabel perbandingan waktu proses dekompresi

No.	Nama file	Ekstensi file	Waktu proses		
			Huffman - Bigram	WinZip	WinRar

Tabel 3.11 Rancangan tabel perbandingan rasio kompresi

No.	Nama file	Ekstensi file	Rasio kompresi		
			Huffman - Bigram	WinZip	WinRar

Dari hasil perbandingan yang diperoleh akan disimpulkan apakah penerapan algoritma Huffman berbasis Bigram ini memberikan hasil yang efektif dalam proses pengkompresian file, dan bagaimanakah perbandingan antara hasil dari sistem dengan hasil dari WinZip dan WinRar dalam hal hasil kompresi yang dicapai, waktu yang dibutuhkan serta rasio kompresi yang terbentuk.

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

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

Uji coba dilakukan terhadap sistem pengkompresian sebuah file dengan menerapkan algoritma Huffman berbasis model Bigram. Hasil dari uji coba ini akan digunakan untuk mengevaluasi tingkat keefektifan yang dilakukan oleh sistem yang telah dibuat.

Perangkat lunak yang digunakan dalam pengembangan sistem pengkompresian ini adalah:

1. Sistem operasi yang digunakan adalah Microsoft Windows XP Professional.
2. *Software* yang digunakan untuk mengembangkan sistem adalah Borland Delphi 7

Perangkat keras yang digunakan untuk mengembangkan sistem pengkompresian sebuah file dengan menerapkan algoritma Huffman berbasis model Bigram, adalah:

1. Prosesor Intel P4 1.80 Ghz
2. Memori 376 MB
3. Harddisk dengan kapasitas 40 GB
4. Monitor 15”
5. Keyboard
6. Mouse

4.1 Implementasi Program

Berdasarkan analisa dan rancangan sistem pada Subbab 3.2, maka akan dilakukan implementasi proses-proses tersebut.

4.1.1 Implementasi proses kompresi

Proses kompresi bertujuan untuk memampatkan data dengan sistem *encoding* tertentu agar diperoleh data yang memiliki ukuran lebih kecil daripada ukuran file aslinya. Dalam proses kompresi akan dilakukan beberapa langkah guna mendapatkan kode Huffman untuk menggantikan pasangan karakter penyusun file. Implementasi

flowchart proses kompresi pada Gambar 3.13, ditunjukkkan pada Gambar 4.1 sebagai berikut:

```
procedure TForm1.CompressFile1Click(Sender: TObject);
var
  ToF: file;
  NumRead, NumWritten: integer;
  Buf:array [1..8] of Char;
  i,j:int64;
  temp,Hasil:string;
  times1,times2:string;
  time1,time2:real;
  IsiFile:string;
  FileCari:TSearchRec;
begin
    Savedialog1.Filter:='BHuf (*.BHuf)|*.BHuf';
    if SaveDialog1.Execute then      { Display Save dialog box}
    begin
        PB1.Visible:=True;
        LNamaFileHasil.Caption:=SaveDialog1.FileName + '.BHuf';
        timer1.Enabled:=True;
        times1:=Trim(formatdatetime('hh:nn:ss',time)+':'+
                 floattostr(MilliSecondOf(time)));
        Insert_Kamus_Huffman(Isi_File);
        SortbyFreq;
        Create_HuffmanTree;
        Create_HeaderFile;
        IsiFile:=Create_IsiFile(Isi_File);
        Hasil:=Header_File +'.'+ kurangbit +'.'+isifile;
        PanjangFilecompressi:=length(hasil);
        AssignFile(ToF, SaveDialog1.FileName + '.BHuf');
        Rewrite(ToF, 1);      { Record size = 1 }
        i:=1;
        while i <= length(Hasil) do
        begin
            numread:=1;
            temp:=midstr(hasil,i,1);
            j:=1;
            while j <= 1 do
            begin
                buf[j]:=temp[j];
                j:=j+1;
            end;
            repeat
                BlockWrite(ToF,Buf, NumRead, NumWritten);
                numread:=0;
            until (NumRead <> 1) or (NumWritten <> NumRead);
            inc(i,1);
        end;
       CloseFile(ToF);
       timer1.Enabled:=false;
```

Gambar 4.1 Implementasi proses kompresi

```
        SaveDialog1.FileName:='';
        time1:=(strtofloat(midstr(times1,1,2))*3600 +
                strtofloat(midstr(times1,4,2))* 60 +
                strtofloat(midstr(times1,7,2)))* 1000 +
                strtoint(rightstr(times1,length(times1)-9));
        times2:=Trim(formatdatetime('hh:nn:ss',time)+':'+
                floattostr(MilliSecondOf(time)));
        time2:=(strtofloat(midstr(times2,1,2))*3600 +
                strtofloat(midstr(times2,4,2))* 60 +
                strtofloat(midstr(times2,7,2)))* 1000;
        time2:=time2 + trtofloat(rightstr(times2,
                length(times2)-9));
        LWktMulai.Caption:=times1;
        LWktSelesai.Caption:=times2;
        Ratio:=(PanjangFileAsli/PanjangFileCompressi);
        LRatio.caption:=floattostr(Ratio);
        PB1.Position:=100;
        LWaktuProses.Caption:= floattostr(time2-time1);
        FindFirst(LNamaFileHasil.Caption, $37, FileCari);
        LUFileHasil.Caption:= inttostr(FileCari.Size);
    end;
end;
```

Gambar 4.1 Implementasi proses kompresi (lanjutan)

Sebelum proses kompresi dijalankan, terlebih dahulu sistem akan memanggil komponen *savedialog*, dan meminta *inputan* berupa nama serta direktori file *output* hasil dari proses yang akan dijalankan. Setelah itu proses kompresi akan dijalankan dengan melalui beberapa langakah. Berikut ini merupakan implementasi dari rancangan proses-proses yang telah dijelaskan sebelumnya:

1. Pembentukan tabel kompresi

```
procedure TForm1.Insert_Kamus_Huffman(IsiFile: String);
var
    i:Int64;
    Temp,Kata:string;
begin
    i:=1;
    While i <= length(IsiFile) do
    begin
        Temp:=copy(IsiFile,i,2);
        Kata:= Temp;
        Insert_Huruf(awal,kata);
        inc(i,2);
        application.ProcessMessages;
    end;
end;
```

Gambar 4.2 Prosedur *Insert_Kamus_Huffman*

```
procedure TForm1.Insert_Huruf(Var Start:HurufPtr;Nama:
String);
Var
    Search,Temp :HurufPtr;
    Cocok:Boolean;
begin
    if start=nil then
    begin
        new(temp);
        Temp^.Nama:=nama;
        Temp^.Juml:=1;
        Temp^.Next:=nil;
        Start:=Temp;
    end

    else
    begin
        Search:=start;
        cocok:=false;
        while (search <> nil) and not cocok do
        begin
            if search^.Nama = nama then
            begin
                cocok:=True;
                search^.Juml:=search^.Juml+1;
            end;
            Search:=Search^.Next;
        end;
        if not cocok then
            Insert_Huruf(Start^.next,Nama);
    end;
end;
```

Gambar 4.3 Prosedur *Insert_Huruf*

Prosedur **Insert_Kamus_Huffman** yang ditunjukkan pada Gambar 4.2 merupakan implementasi dari proses pembentukan tabel kompresi. Dengan memanggil prosedur **Insert_Huruf** yang terdapat pada Gambar 4.3, tabel kompresi dibentuk dengan memanfaatkan *record* dengan variabel **ListHurufRec**. *Record* dibentuk untuk menyimpan pasangan-pasangan karakter penyusun file beserta frekuensi kemunculannya. Ketika sebuah pasangan karakter dibaca dari file maka pasangan tersebut akan dimasukkan ke dalam *record* dengan variabel **Nama** yang merupakan *record* dari **ListHurufRec**. Kemudian sistem akan membaca pasangan karakter berikutnya, jika pasangan tersebut belum tercantum di dalam tabel kompresi maka pasangan tersebut akan menjadi data baru untuk *record*. Akan tetapi jika pasangan tersebut sudah

tercantum di dalam tabel kompresi, maka akan dilakukan penambahan pada frekuensi kemunculannya yang diimplementasikan dengan variabel **Juml** yang merupakan record dari **ListHurufRec.** Implementasi dari pembentukan struktur data berupa pointer yang menunjuk ke sebuah *record* dalam pembentukan tabel kompresi ditunujukkan pada Gambar 4.4 berikut:

```
HurufPtr = ^ListHurufRec;
 ListHurufRec=Record
              Nama : String;
              Juml :Int64;
              Nilai:String;
              Next : HurufPtr;
         end;
```

Gambar 4.4 *Record* tabel kompresi

2. Pengurutan pasangan karakter

Gambar 4.5 merupakan implementasi dari proses pengurutan pasangan karakter. Proses ini dilakukan terhadap data yang terdapat pada tabel kompresi yang berupa daftar pasangan karakter. Pengurutan yang dilakukan bersifat *ascending* berdasarkan pada frekuensi kemunculannya. Sehingga untuk proses selanjutnya yaitu proses pembentukan pohon Huffman akan dapat menghasilkan kode Huffman yang efektif, dimana pasangan karakter dengan frekuensi besar akan memiliki kode Huffman yang lebih pendek dibandingkan dengan pasangan karakter dengan frekuensi yang kecil.

Aturan yang diterapkan pada prosedur **SortByFreq** adalah sebagai berikut:

1. Jika sebuah pasangan huruf dalam daftar *record* memiliki frekuensi lebih besar daripada pasangan huruf berikutnya, maka akan dilakukan pertukaran posisi. Demikian seterusnya hingga data dalam *record* tersusun secara *ascending.*
2. Jika dua buah pasangan karakter memiliki frekuensi yang sama maka aturan yang diterapakan adalah dengan memperhatikan panjang dari pasangan tersebut. Pasangan yang terdiri dari satu karakter akan dianggap lebih kecil dibandingkan dengan pasangan yang terdiri dari dua karakter. Hal ini diterapkan karena ada kemungkinan sebuah pasangan huruf terdiri dari satu karakter saja.

```
procedure TForm1.SortByFreq;
Var
    Search,Search1 :HurufPtr;
    Temp:Int64;
    TempNama:String;
begin
  Search1:=awal;
  While Search1 <> nil do
  begin
    Search:=awal;
    While Search <> nil do
    begin
      if search^.Next <> nil then
      begin
        if Search^.Juml > Search^.Next^.Juml then
        begin
            Temp:= Search^.Juml;
            Search^.Juml:=Search^.next^.Juml;
            Search^.next^.Juml:=Temp;
            TempNama:= Search^.Nama;
            Search^.Nama:=Search^.next^.Nama;
            Search^.Next^.Nama:=TempNama;
        end
        else if  Search^.Juml = Search^.Next^.Juml then
        if length(Search^.Nama) > Length(Search^.next^.Nama) then
        begin
            Temp:= Search^.Juml;
            Search^.Juml:=Search^.next^.Juml;
            Search^.next^.Juml:=Temp;
            TempNama:= Search^.Nama;
            Search^.Nama:=Search^.next^.Nama;
            Search^.Next^.Nama:=TempNama;
        end;
      end;
      Search:=Search^.Next;
      application.ProcessMessages;
    end;
    Search1:=Search1^.Next;
    application.ProcessMessages;
  end;
end;
```

Gambar 4.5 Prosedur *SortByFreq*

3. Pembentukan pohon Huffman

Proses pembentukan pohon Huffman akan menghasilkan kode Huffman yang akan digunakan dalam proses *encoding*. Sebelum dibentuk sebuah pohon Huffman terlebih dahulu dilakukan pembentukan sebuah *record* seperti yang ditunjukkan pada Gambar 4.6 berikut:

```
    RHuruf=Record
              Nama    :String;
              midKiri :String;
              midKanan:String;
              NilaiMidKiri:Int64;
           end;
```

Gambar 4.6 *Record* untuk pembentukan pohon Huffman

Implementasi dari proses pembentukan pohon Huffman ditunjukkan pada Gambar 4.7.

```
procedure TForm1.Create_HuffmanTree;
var
  Temp:String;
  Search:HurufPtr;
  Hasil:string;
  Ketemu:Boolean;
  MidKiri,MidKanan:String;
  TempAwal:String;
begin
    TempAwal:='';
    Search:=awal;
    While Search <> nil do
    begin
      TempAwal:=TempAwal+Search^.Nama;
      search:=search^.next;
      application.ProcessMessages;
    end;
    Search:=awal;
    While Search <> nil do
    begin
      hasil:='';
      Ketemu:=False;
      Temp:=TempAwal;
```

Gambar 4.7 Prosedur *Create_HuffmanTree*

```
      While (not ketemu) do
      begin
        MidKiri:=Get_MidKiri(temp,Get_NilaiTemp(Temp));
        MidKanan:=Get_midKanan(Temp,Midkiri);
        if (get_nilaiHuruf2(temp,search^.nama) <=
            Huruf.NilaiMidKiri)  then
        begin
          if (length(temp)=3) and (midkiri=Search^.nama) then
          begin
            temp:=midkiri;
            hasil:=hasil+'0';
          end
          else if (length(temp)=3) and (midkanan=Search^.nama)
              then
          begin
             temp:=midkanan;
             hasil:=hasil+'1';
          end
          else if (length(temp) > 3) then
          begin
             temp:=midkiri;
             hasil:=hasil+'0';
          end;
        end

        else
            begin
                temp:=midkanan;
                hasil:=hasil+'1';
            end;
        if temp=search^.Nama then
            begin
                ketemu:=true;
                Search^.Nilai:=hasil;
            end;
      end;
      search:=search^.Next;
      application.ProcessMessages;
   end;
end;
```

Gambar 4.7 Prosedur *Create_HuffmanTree* (lanjutan)

Prosedur **Create_HuffmanTree** akan memanggil fungsi **Get_MidKiri** dan **Get_MidKanan**. Fungsi **Get_MidKiri** diimplementasikan untuk menentukan simpul kiri dari pohon Huffman yang akan dibentuk. Terdapat dua aturan yang diterapkan dalam fungsi **Get_MidKiri**.

Aturan 1 diterapkan jika jumlah pasangan karakter pada *root* berjumlah lebih dari dua. Maka simpul kiri didapatkan dengan

menggabungkan simpul-simpul hingga jumlah frekuensi total dari gabungan tersebut bernilai sama dengan atau lebih kecil dari nilai tengah, dimana nilai tengah dihitung berdasarkan pada persamaan 3.1 yang dijelaskan dalam Subbab 3.2.1.

Aturan 2 diterapkan jika jumlah pasangan karakter pada *root* berjumlah dua. Jika panjang karakter pada *root* adalah genap, maka simpul kiri adalah dua karakter paling depan. Sementara jika panjang karakter pada *root* adalah ganjil, maka simpul kiri adalah satu karakter paling depan dari *root.* Implementasi dari kedua aturan tersebut ditunjukkan pada Gambar 4.9.

Jika simpul kiri telah didapatkan, maka simpul kanan juga dapat ditentukan. Implementasi proses untuk menentukan simpul kanan dari sebuah *root* ditunjukkan pada Gambar 4.8 berikut ini.

```
function TForm1.Get_MidKanan(Kata, MidKiri: String): string;
begin
  Get_midkanan:=mids tr(kata,length(MidKiri)+1,Length(kata)-
                Length(midKiri));
end;
```

Gambar 4.8 Fungsi *Get_MidKanan*

Hasil dari proses *Create_HuffmanTree* yang berupa susunan bit kode Huffman, akan dimasukkan ke dalam tabel kompresi yang diimplementasikan dengan variabel **`nilai`** yang merupakan *record* dari **`ListHurufRec`**.

```
//aturan 1

if JPasangKata > 2 then
begin
    i:=1;
    TempJPasang:=0;
    While (TempNilai < (nilai div 2)) and (TempJPasang <
            PasangKata) do
    begin
        if (length(kata) mod 2 > 0) and (i=1) then
        begin
            TempHuruf:= copy(kata,i,1);
            inc(i);
        end
        else
        begin
            TempHuruf:= copy(kata,i,2);
            inc(i,2);
        end;
        Temp:=Temp + TempHuruf;
        inc(TempJPasang);             TempNilai:=TempNilai
        +Get_NilaiHuruf(TempHuruf);
        if Tempnilai <= (nilai div 2) then
        begin
            Huruf1:=temp;
            NilaiHuruf:=TempNilai;
        end;
        application.ProcessMessages;
    end;
end

//aturan 2

else if JPasangKata=2 then
begin
    if length(kata)mod 2=0 then
        Temp:=leftstr(kata,2)
    else
        Temp:=leftstr(kata,1);
    NilaiHuruf:=TempNilai+Get_NilaiHuruf(Temp);
    Huruf1:=Temp;
end;
Get_MidKiri:=Huruf1;
Huruf.NilaiMidKiri:= NilaiHuruf;
```

Gambar 4.9 Potongan fungsi *Get_MidKiri*

4. Pembentukan header file

Implementasi dari proses pembentukan header file ditunjukkan pada Gambar 4.10 sebagai berikut:

```
Procedure TForm1.Create_HeaderFile;
var
  Search:hurufptr;
begin
    Header1:='';
    Header2:='';
    Search:=awal;
    While (Search <> nil) do
    begin
        Header1:=Header1 + Search^.Nama;
        if search <> awal then Header2:=Header2+'-';
        Header2:=Header2 + inttostr(Search^.Juml);
        Search:=Search^.next;
    end;
    JCharKamus:=Inttostr(Length(Header1));
    Header_File:= 'BHuf.'+ eks +'.'+ JCharKamus+'.'+
                  Header1+'.'+ Header2;
end;
```

Gambar 4.10 Prosedur *Create_HeaderFile*

Pembentukan *header* file ini dimaksudkan agar dapat bermanfaat dalam proses dekompresi untuk file yang bersangkutan. Seperti rancangan yang telah dijelaskan sebelumnya, bahwa *header* file yang terbentuk akan berisi enam bagian. Prosedur di atas terlebih dahulu akan membentuk lima bagian pertama dari header file, yaitu ekstensi **.Bhuf;* ekstensi dari file *input;* **`JcharKamus`** yang merupakan jumlah karakter pada header1; header1 dan header2.

Header1 dan header2 dibentuk dengan memanfaatkan *pointer* yang memanggil *record* yang menunjuk ke alamat tabel kompresi. Header1 terdiri dari kumpulan pasangan karakter yang terdapat pada tabel kompresi, sedangkan header2 terdiri dari frekuensi dari pasangan karakter pada header1 secara terurut. Antara frekuensi satu dengan yang lain akan diberi penanda '-' (strip) agar tidak terjadi kerancuan dalam pembacaan file untuk proses selanjutnya.

Sedangkan satu bagian terakhir dari *header* file yaitu **`kurangbit`** yang merupakan jumlah tambahan bilangan biner nol yang diperlukan untuk pengkonversian string biner ke dalam karakter ASCII, akan dihitung ketika proses *encoding* telah dilakukan oleh sistem.

5. *Encoding*

Implementasi dari proses *encoding* terdapat pada fungsi **`Create_IsiFile`** seperti yang ditunjukkan pada Gambar 4.11 berikut ini:

```
i:=1;
While i<=length(IsiFile) do
begin
    Kata:=midstr(IsiFile,i,2);
    ketemu:=False;
    Search:=awal;
    While (Search <> nil) and (not ketemu) do
    begin
        if kata=search^.Nama then
        begin
           Ketemu:=True;
           Temp:=Temp+search^.Nilai;
           application.ProcessMessages;
        end
        else
           Search:=Search^.next;
        application.ProcessMessages;
    end;
    inc(i,2);
end;
```

Gambar 4.11 Prosedur proses *Encoding*

Langkah awal yang dilakukan dari proses *encoding* adalah dengan pembacaan per dua karakter dari isi file. Kemudian pasangan karakter tersebut akan dicocokkan dengan kode Huffman pada tabel kompresi. Jika pasangan ditemukan di dalam tabel kompresi maka kode Huffman yang terdapat pada record **`Nilai`** pada **`ListHurufRec`** akan diambil untuk menggantikan pasangan tersebut. Demikian seterusnya hingga semua isi file telah ter-*encoding* oleh sistem.

Kemudian proses selanjutnya adalah pengitungan bilangan biner '0' yang diperlukan sebagai tambahan dalam proses pengkonversian hasil *encoding* agar dapat tepat dikonversi per delapan bit. Penghitungan **`kurangbit`** ini merupakan bagian dari fungsi **`Create_IsiFile`** yang diimplementasikan pada Gambar 4.12 sebagai berikut:

```
Temp1:='';
Kurangbit:=inttostr(8-(length(temp)mod 8));
i:=1;
While i <= strtoint(kurangbit) do
begin
   temp:=temp+'0';
   i:=i+1;
end;
i:=1;
```

Gambar 4.12 Fungsi perhitungan kurangbit

Dengan demikian hasil dari proses *encoding* serta tambahan bilangan biner '0' yang diperlukan dalam proses pengkonversian direpresentasikan dengan variabel **Temp** adalah berupa string biner.

6. Penyimpanan file terkompresi

Hasil dari prose *encoding* yang berupa string biner akan terlebih dahulu dikonversi ke dalam karakter ASCII. String biner akan dibaca per delapan bit. Pengkonversian terhadap **`Temp`** dilakukan dengan menerapkan aturan yaitu:

1. Jika nilai desimal delapan bit dari **`Temp`** ditambah dengan lima belas menghasilkan nilai yang lebih besar dari 254, maka penkonversiannya adalah :

   `KarakterASCII(14) + karakterASCII(temp, 8)`

2. Jika nilai desimal delapan bit dari **`Temp`** ditambah dengan lima belas menghasilkan nilai yang lebih kecil dari 254, maka penkonversiannya adalah :

   `KarakterASCII((temp,8)+15)`

Implementasi dari aturan tersebut merupakan bagian dari prosedur **`Create_IsiFile`**, seperti ditunjukkan pada gambar 4.13 berikut ini:

```
While i<=length(Temp) do
begin
  if (bintoint(copy(Temp,i,8)) + 15) > 254 then
     Temp1:= Temp1+chr(14)+chr(bintoint(copy(Temp,i,8)))
  else
     Temp1:= Temp1 +  chr(bintoint(copy(Temp,i,8)) +15);
  inc(i,8);
end;
Create_isiFile:=Temp1;
```

Gambar 4.13 Aturan pengkonversian

Hasil dari proses pengkonversian tersebut direpresentasikan dengan **`Temp1`** yang berisi string karakter ASCII. Imlementasi penyimpanan file terkompresi termasuk dalam prosedur kompresi, ditunjukkan pada gambar 4.14 sebagai berikut:

```
IsiFile:=Create_IsiFile(Isi_File);
Hasil:=Header_File +'.'+ kurangbit +'.'+isifile;
PanjangFilecompressi:=length(hasil);
AssignFile(ToF, SaveDialog1.FileName + '.BHuf'); { Open
    output file }
Rewrite(ToF, 1);       { Record size = 1 }
i:=1;
while i <= length(Hasil) do
begin
    numread:=1;
    temp:=midstr(hasil,i,1);
    j:=1;
    while j <= 1 do
    begin
        buf[j]:=temp[j];
        j:=j+1;
    end;
    repeat
        BlockWrite(ToF,Buf, NumRead, NumWritten);
        numread:=0;
    until (NumRead <> 1) or (NumWritten <> NumRead);
    inc(i,1);
end;
CloseFile(ToF);
```

Gambar 4.14 Penyimpanan file terkompresi

4.1.2 Implementasi proses dekompresi

Proses dekompresi bertujuan untuk mengembalikan sebuah file terkompresi menjadi seperti file aslinya, agar dapat digunakan sesuai

kebutuhan. Implementasi dari *flowchart* proses dekompresi pada gambar 3.14, ditunjukkan pada gambar 4.15 berikut ini:

```
procedure TForm1.DecompressFile1Click(Sender: TObject);
var
  ToF: file;
  NumRead, NumWritten: integer;
  Buf:array [1..8] of Char;
  i:int64;
  j:integer;
  temp,Hasil:string;
  times1,times2:string;
  time1,time2:Word;
  FileCari:TSearchRec;
begin
  Savedialog1.Filter:='All File(*.*)|*.*';
  if SaveDialog1.Execute then   {Display Save dialog box}
  begin
    PB1.Visible:=True;
    times1:=Trim(formatdatetime('hh:nn:ss',time)+':'+
    floattostr(MilliSecondOf(time)));
    timer1.Enabled:=true;
    Pembangkit_Huffman(Isi_File);
    LNamaFileHasil.Caption:= SaveDialog1.FileName + '.' + eks;
    AssignFile(ToF, SaveDialog1.FileName + '.' + eks);
    Rewrite(ToF, 1);       { Record size = 1 }
    i:=1;
    Hasil:=isi_file;
    while i <= length(Hasil) do
    begin
       numread:=1;
       temp:=midstr(hasil,i,1);
       for j:=1 to 1 do
           buf[j]:=temp[j];
       repeat
           BlockWrite(ToF,Buf, NumRead, umWritten);
           numread:=0;
       until (NumRead <> 1) or (NumWritten <> NumRead);
       inc(i,1);
    end;
    CloseFile(ToF);
    SaveDialog1.FileName:='';
    time1:=(strtoint(midstr(times1,1,2))*3600 +
        strtoint(midstr(times1,4,2))* 60 +
        Strtoint(midstr(times1,7,2)))* 1000 +
        strtoint(rightstr(times1,length(times1)-9));
    times2:=Trim(formatdatetime('hh:nn:ss',time)+':'+
        floattostr(MilliSecondOf(time)));
    time2:=(strtoint(midstr(times2,1,2))*3600 + strtoint
        (midstr(times2,4,2))* 60 +
        strtoint(midstr(times2,7,2)))* 1000;
    time2:= time2 + strtoint(rightstr(times2,length(times2)-9));
```

Gambar 4.15 Implementasi proses dekompresi

```
      LWktMulai.Caption:=times1;
      LWktSelesai.Caption:=times2;
      timer1.Enabled:=false;
      PB1.Position:=100;
      LWaktuProses.Caption:= floattostr(time2-time1);
      FindFirst(LNamaFileHasil.Caption, $37, FileCari);
      LUFileHasil.Caption:= inttostr(FileCari.Size);
   end;
end;
```

Gambar 4.15 Implementasi proses dekompresi (lanjutan)

Ketika proses dekompresi dijalankan, sistem akan memanggil komponen *Savedialog* dan meminta *input* berupa nama file serta direktori untuk file *output* yang akan dihasilkan nantinya. Proses dekompresi melibatkan beberapa sub proses, berikut ini merupakan penjelasan implementasi dari rancangan proses dekompresi yang telah dijelaskan pada subbab 3.2.2.

1. Pembangkitan kode Huffman

Proses pembangkitan kode Huffman merupakan proses inti dari proses dekompresi. Proses ini melibatkan beberapa sub proses yang saling berhubungan. Proses dekompresi pada sistem ini hanya dapat dilakukan pada file terkompresi dengan ekstensi **.Bhuff*. Gambar 4.16 berikut ini merupakan implementasi dari proses pembangkitan kode Huffman:

```
procedure TForm1.Pembangkit_Huffman(IsiFile: string);
Var
  PrevChar:int64;
  Temp,Temp1:string;
  i:Int64;
  TempHasil:Int64;
  Search:Hurufptr;
  PAwal:Int64;
  Hasil:string;
Begin
  zeromemory(@awal,sizeof(awal));
  if Pos('BHuf',IsiFile)>0 then
  begin
      EksHuff:=LeftStr(IsiFile,Posisi_Titik(IsiFile)-5);
      isiFile:=rightstr(isiFile,length(isiFile)-
      Posisi_Titik(IsiFile));
      Eks:=LeftStr(IsiFile,Posisi_Titik(IsiFile)-1);
      isiFile:=rightstr(isiFile,length(isiFile)-
      Posisi_Titik(IsiFile));
      JCharKamus:=LeftStr(IsiFile,Posisi_Titik(IsiFile)-1);
      isiFile:=rightstr(isiFile,length(isiFile)-
               Posisi_Titik(IsiFile));
      Header1:=leftstr(IsiFile,strtoint(JCharKamus));
      Insert_Kamus_Huffman1(Header1);
      IsiFile:=rightstr(isiFile,length(isiFile) -
              (length(Header1)+1));
      Header2:=LeftStr(IsiFile,Posisi_Titik(IsiFile)-1);
      i:=1;
      TempHasil:=0;
      Temp1:=Header2;
      Search:=awal;
      While (i<Length(Header2)) and (search<> nil) do
      begin
            temp:=LeftStr(Temp1,Posisi_strip(Temp1)-1);
            if temp <> '' then
            begin
                Temp1:=rightstr(Temp1,length(Temp1)-
                       (length(temp)+1));
                TempHasil:=TempHasil+strtoint(temp);
                search^.Juml:=strtoint(temp);
            end
            else
            begin
                 TempHasil:=TempHasil+strtoint(Temp1);
                 search^.Juml:=strtoint(Temp1);
                 i:=i+length(temp1)+1;
            end;
            i:=i+Posisi_strip(Temp1);
            Search:=Search^.Next;
            application.ProcessMessages;
      end;
```

Gambar 4.16 Prosedur *Pembangkit_Huffman*

```
        isiFile:=rightstr(isiFile,length(isiFile)-
                 Posisi_Titik(IsiFile));
        Kurangbit:= LeftStr(IsiFile,Posisi_Titik(IsiFile)-1);
        IsiFile:=RightStr(IsiFile,length(IsiFile)-
                 length(Kurangbit)-1);
        Temp:='';
        create_HuffmanTree;
        PrevChar:=0;
        i:=1;
        Temp:='';
        while i <=length(isiFile) do
        begin
           if (ord(isiFile[i]) > 14) then
           begin
              if (PrevChar <> 14) then
                  Temp:=Temp+inttobin(ord(isiFile[i])-15)
              else
              begin
                  Temp:=Temp+inttobin(ord(isiFile[i]));
                  PrevChar:=0;
              end;
           end
           else
              PrevChar:=14;
           i:=i+1;
           application.ProcessMessages;
        end;
        IsiFile:=LeftStr(Temp,Length(Temp)-strtoint64(kurangbit));
        Hasil:='';
        PAwal:=1;
        i:=1;
        while i <= length(isiFile) do
        begin
             Temp:=midstr(isifile,Pawal,(i-Pawal)+1);
             Temp1:=Find_huruf(Temp);
             If Temp1<>'' then
             begin
                 Hasil:=Hasil+temp1;
                 Pawal:=Pawal+length(temp);
             end;
             i:=i+1;
             application.ProcessMessages;
        end;
        Isi_File:=Hasil;
    end
    else
        messagedlg('Data Tidak Ter-compress',mtinformation,
            [mbok],0);
end;
```

Gambar 4.16 Prosedur *Pembangkit_Huffman* (lanjutan)

1.1 Penerjemahan *header* file

Proses penerjemahan *header* file termasuk dalam prosedur **`Pembangkit_Huffman`**. Proses yang dilakukan adalah dengan memotong-motong *header* file sesuai dengan bagian-bagian yang telah dijelaskan sebelumnya. Bagian pertama yang dipotong adalah **`EksHuff`** yang merupakan ekstensi *default* file terkompresi dari sistem ini. Bagian kedua yang dipotong adalah **`Eks`** yaitu ekstensi dari file asli. Bagian ketiga yang dipotong adalah **`JcharKamus`** yang merupakan panjang dari karakter yang terdapat pada tabel dekompresi. Bagian keempat adalah **`Header1`** yang merupakan daftar karakter yang terdapat pada tabel dekompresi. Bagian kelima adalah **`Header2`** yang merupakan daftar frekuensi dari masing-masing karakter pada header1. dan bagian keenam adalah **`Kurangbit`** yang merupakan tambahan bilangan biner '0' yang dibutuhkan dalam pengkonversian isi file.

1.2 Pembentukan tabel dekompresi

Implementasi pembentukan tabel dekompresi ditunjukkan dalam gambar 4.17. Dalam proses ini, jika panjang tabel dekompresi adalah bilangan genap, maka pembacaan header1 untuk dimasukkan ke dalam tabel dekompresi dilakukan per dua karakter. Akan tetapi jika panjang tabel dekompresi adalah bilangan ganjil, maka pembacaan pertama dilakukan pada satu karakter terdepan, kemudian diikuti dengan pembacaan per dua karakter.

Jika telah didapatkan pasangan karakter maka akan dimasukkan ke dalam tabel dekompresi. Tabel dekompresi dibentuk dengan memanfaatkan *record.* Implementasi dari proses pemasukkan data ke dalam tabel dekompresi ditunjukkan pada gambar 4.18.

```
procedure TForm1.Insert_Kamus_Huffman1(IsiFile: String);
var
    i:Int64;
    Temp:string;
    Status_Awal:Boolean;
begin
    i:=1;
    Status_Awal:=False;
    While i < length(IsiFile) do
    begin
        if (length(isiFile)mod 2 > 0) and (not status_awal) then
        begin
            Status_Awal:=True;
            Temp:=leftstr(isifile,1);
            i:=i+1;
        end
        else
        begin
            Temp:=midstr(IsiFile,i,2);
            i:=i+2;
        end;
        Insert_Huruf1(awal,Temp);

    end;
end;
```

Gambar 4.17 Prosedur *Insert_Kamus_Huffman1*

```
procedure TForm1.Insert_Huruf1(var Start: hurufPtr; Nama:
String);
Var
    Temp :HurufPtr;
begin
    if start=nil then
    begin
        new(temp);
        Temp^.Nama:=nama;
        Temp^.Next:=nil;
        Start:=Temp;
    end
    else
        Insert_Huruf1(Start^.next,Nama);
end;
```

Gambar 4.18 Prosedur *Insert_Huruf1*

1.3 Pembacaan frekuensi

Implementasi proses pembacaan frekuensi ditunjukkan pada gambar 4.19 berikut:

```
While (i<Length(Header2)) and (search<> nil) do
begin
   temp:=LeftStr(Temp1,Posisi_strip(Temp1)-1);
   if temp <> '' then
   begin
      Temp1:=rightstr(Temp1,length(Temp1)-(length(temp)+1));
      TempHasil:=TempHasil+strtoint(temp);
      search^.Juml:=strtoint(temp);
   end
   else
   begin
      TempHasil:=TempHasil+strtoint(Temp1);
      search^.Juml:=strtoint(Temp1);
      i:=i+length(temp1)+1;
   end;
   i:=i+Posisi_strip(Temp1);
   Search:=Search^.Next;
   application.ProcessMessages;
end;
```

Gambar 4.19 Proses pembacaan frekuensi

Proses pembacaan frekuensi dilakukan untuk melengkapi data yang terdapat pada tabel dekompresi. Pembacaan frekuensi dilakukan pada *header* file pada bagian header2. Proses ini termasuk dalam prosedur **`Pembangkit_Huffman`**.

Karena terdapat penanda '-' (strip) pada setiap frekuensi pada header2, maka dibentuk sebuah fungsi **`Posisi_Strip`** yang digunakan untuk memotong penanda pada header2 tersebut.

1.4 Pembentukan pohon Huffman

Kode Huffman untuk proses dekompresi harus sama dengan yang digunakan pada proses kompresi. Oleh karena itu, proses pembentukan pohon Huffman pada proses dekompresi ini akan memanggil kembali prosedur **`Create_Huffman_Tree`** yang telah diimplementasikan pada proses kompresi yang ditunjukkan pada gambar 4.7.

1.5 Decoding

Decoding dilakukan untuk mengembalikan sebuah file terkompresi menjadi seperti file aslinya. Langkah awal yang dilakukan adalah mengkonversi kembali karakter ASCII ke dalam string biner. Aturan konversi yang diterapkan pada proses ini sama dengan aturan yang diterapakan pada pengkonversian pada proses kompresi. Sehingga didapatkan susunan string biner yang direpresentasikan dengan **Temp**. Langkah selanjutnya adalah memotong **Temp** dari belakang sebanyak **Kurangbit**. Implementasi dari proses pengkonversian tersebut ditunjukkan pada gambar 4.20.

```
while i <=length(isiFile) do
begin
   if (ord(isiFile[i]) > 14) then
   begin
      if (PrevChar <> 14) then
            Temp:=Temp+inttobin(ord(isiFile[i])-15)
      else
      begin
            Temp:=Temp+inttobin(ord(isiFile[i]));
            PrevChar:=0;
      end;
   end
   else
      PrevChar:=14;
   i:=i+1;
   application.ProcessMessages;
end;
IsiFile:= LeftStr(Temp,Length(Temp)-strtoint64(kurangbit));
```

Gambar 4.20 Proses pengkonversian

Setelah didapatkan **IsiFile** yang berupa kumpulan bilangan biner, proses selanjutnya adalah melakukan *decoding* terhadap **IsiFile**. String biner akan dibandingkan dengan tabel dekompresi, jika terjadi kecocokkan maka karakter akan diambil untuk menggantikan string biner tersebut. Implementasi dari proses *decoding* ditunjukkan pada Gambar 4.21. Proses *decoding* ini akan memanggil fungsi *Find_Huruf,* fungsi ini bertujuan untuk mecocokkan string biner dengan kode Huffman yang terdapat pada tabel dekompresi. Implementasi dari fungsi *Find_Huruf* ditunjukkan pada Gambar 4.22.

```
while i <= length(isiFile) do
begin
   Temp:=midstr(isifile,Pawal,(i-Pawal)+1);
   Temp1:=Find_huruf(Temp);
   If Temp1<>'' then
   begin
        Hasil:=Hasil+temp1;
        Pawal:=Pawal+length(temp);
   end;
   i:=i+1;
   application.ProcessMessages;
end;
Isi_File:=Hasil;
```

Gambar 4.21 Prosedur proses *decoding*

```
function TForm1.Find_Huruf(Nilai: string): string;
var
    Search:Hurufptr;
    ketemu:boolean;
begin
    Find_Huruf:='';
    search:=awal;
    ketemu:=false;
    While (search <> nil)and (not ketemu) do
    begin
        if search^.Nilai=nilai then
        begin
            Find_Huruf:=Search^.Nama;
            Ketemu:=true;
        end;
        Search:=Search^.Next;
    end;
end;
```

Gambar 4.22 Prosedur *Find_Huruf*

2. Penyimpanan file terdekompresi

Hasil dari fungsi **`Find_Huruf`** adalah berupa susunan karakter. Hasil ini direpresentasikan oleh **`Temp1`**. Hasil akan disimpan ke dalam sebuah file baru dengan ekstensi sesuai dengan yang tertera pada *header* file.

```
AssignFile(ToF, SaveDialog1.FileName + '.' + eks);
Rewrite(ToF, 1);       { Record size = 1 }
i:=1;
Hasil:=isi_file;
while i <= length(Hasil) do
begin
     numread:=1;
     temp:=midstr(hasil,i,1);
     for j:=1 to 1 do
          buf[j]:=temp[j];
     repeat
          BlockWrite(ToF,Buf, NumRead, NumWritten);
          numread:=0;
     until (NumRead <> 1) or (NumWritten <> NumRead);
     inc(i,1);
end;
CloseFile(ToF);
SaveDialog1.FileName:='';
```

Gambar 4.23 Prosedur penyimpanan file terdekompresi

4.1.3 Implementasi perhitungan waktu proses

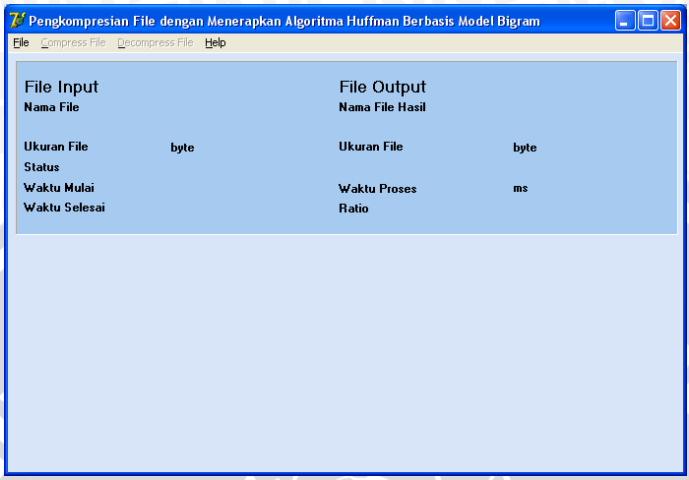
Waktu proses merupakan waktu yang diperlukan selama proses berlangsung, baik proses kompresi maupun proses dekompresi. Waktu proses mulai dihitung ketika proses mulai dilakukan. Dan waktu akhir proses adalah waktu ketika *output* dari proses telah selesai disimpan. Waktu proses pada kompresi diimplementasikan pada prosedur `CompressFile1Click` yang ditunjukkan pada Gambar 4.1. Sedangkan waktu proses pada dekompresi diimplementasikan pada prosedur `DecompressFile1Click` yang ditunjukkan pada Gambar 4.15.

4.1.4 Implementasi perhitungan rasio kompresi

Rasio kompresi merupakan perbandingan antara ukuran file asli dengan ukuran file terkompresinya. Rasio kompresi menunjukkan tingkat kefektifan sebuah sistem kompresi. Semakin tinggi rasio yang dicapai oleh sebuah sistem, maka akan semakin efektif pula proses kompresi yang dilakukan terhadap file tersebut. Implementasi dari proses perhitungan rasio kompresi ini termasuk dalam prosedur `CompressFile1Click` yang ditunjukkan pada Gambar 4.1.

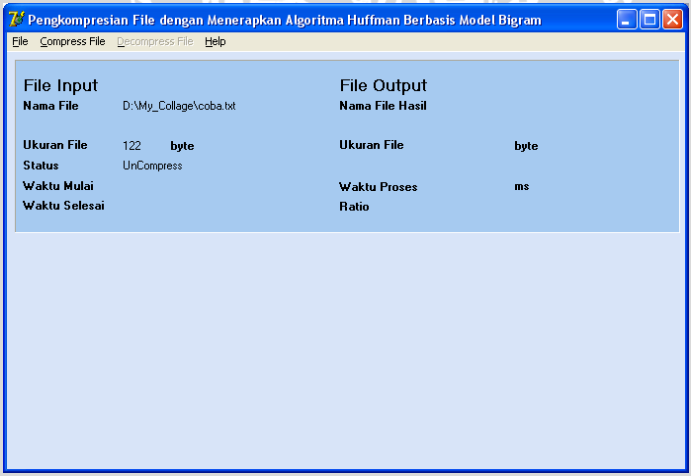
4.2 Implementasi Antarmuka

Berdasarkan pada rancangan antarmuka maka dihasilkan tampilan awal dari sistem yang ditunjukkan pada gambar 4.24.



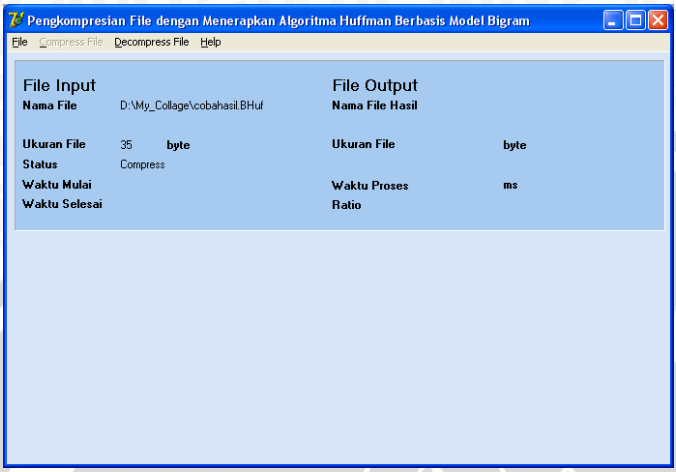
Gambar 4.24 Antarmuka Aplikasi Pengkompresian File dengan Menggunakan Algoritma Huffman Berbasis Model Bigram

Jika inputan berupa file yang belum terkompresi maka status dari file tersebut adalah *uncompress,* seperti ditunjukkan pada Gambar 4.25. Nama, ukuran serta direktori file input juga akan ditampilkan.

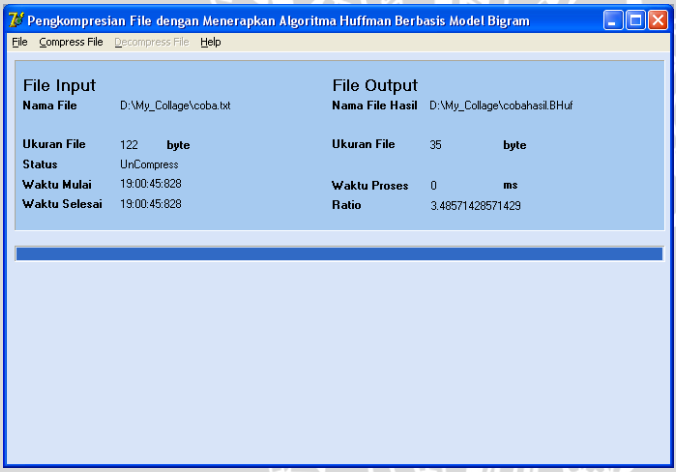


Gambar 4.25 Tampilan setelah file diinputkan

Akan tetapi, jika inputan berupa file yang terkompresi maka status dari file tersebut adalah *compress,* seperti yang ditunjukkan pada Gambar 4.26.



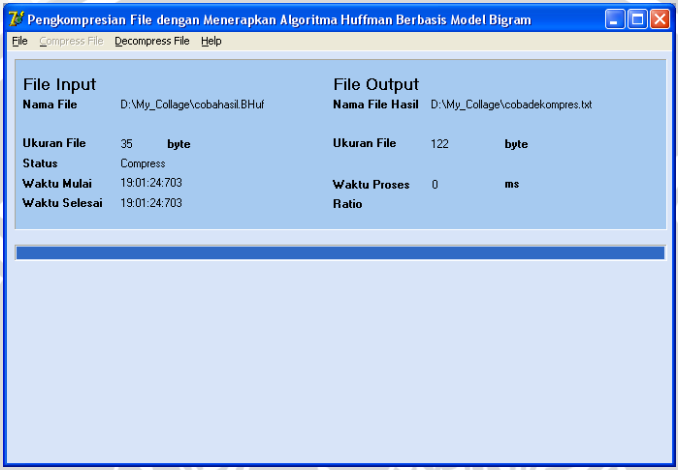
Gambar 4.26 Tampilan setelah file terkompresi diinputkan



Gambar 4.27 Antarmuka untuk proses kompresi

Menu *Compress File* akan aktif jika file yang dibuka adalah file dengan status *uncompress.* Nama disertai direktori dan ukuran dari file *output* akan ditampilkan pada *form*. Selain itu rasio serta waktu proses juga akan ditampilkan apabila proses telah berakhir. Jika sebuah proses telah berakhir maka akan ditandai dengan berhentinya *ProgessBar.* Tampilan antarmuka untuk proses kompresi ditunjukkan pada gambar 4.27.

Menu *Decompress File* akan aktif apabila file yang dibuka adalah file dengan status *compress*. Hasil *output* yang ditampilkan pada form jika proses telah berakhir adalah nama disertai direktori file, ukuran file serta waktu proses yang dibutuhkan. Tampilan antarmuka untuk proses dekompresi ditunjukkan pada gambar 4.28.



Gambar 4.28 Antarmuka untuk proses dekompresi

Gambat 4.29 Tampilan menu *Help*

Menu *Help* merupakan petunjuk bagi *user* jika akan menjalankan sistem ini. Tampilan menu *Help* ditunjukkan pada Gambar 4.29.

4.3 Implementasi Uji Coba dan Evaluasi Hasil

4.3.1 Rancangan Evaluasi

Pengujian dilakukan terhadap 40 file yaitu masing-masing sepuluh file dengan ekstensi **.doc*, **.txt*, **.html* , dan **.bmp*. Uji coba dilakukan terhadap berbagai ukuran file. Data *input* yang digunakan dalam uji coba ditunjukkan pada tabel 4.1 berikut.

Tabel 4.1 Tabel data *input* untuk uji coba

No.	Nama file	Ekstensi	Ukuran file (bytes)
1.	Kata	**.doc*	19.968
2.	Pengantar	**.doc*	20.992
3.	Datmin	**.doc*	28.160
4.	Data	**.doc*	32.768
5.	Exercises	**.doc*	34.816
6.	Abstraksi	**.doc*	40.960
7.	Bab	**.doc*	45.056
8.	Abstrak	**.doc*	48.128
9.	Pendahuluan	**.doc*	60.928
10.	AI	**.doc*	76.288
11.	Findex	**.html*	1.271
12.	Huffman	**.html*	4.089
13.	Teko	**.html*	14.201
14.	Lang	**.html*	17.498
15.	Linkedlist	**.html*	20.143
16.	Index	**.html*	28.022
17.	Dalmassi	**.html*	31.611
18.	Web	**.html*	36.336
19.	ASCII	**.html*	43.435
20.	ZIP	**.html*	59.239
21.	Mama	**.txt*	44
22.	Data	**.txt*	879
23.	Dokumen	**.txt*	3.099
24.	Filosofi	**.txt*	5.221
25.	Flopy	**.txt*	7.182
26.	Komputer	**.txt*	9.084

Tabel 4.1 Tabel data *input* untuk uji coba (lanjutan)

No.	Nama file	Ekstensi	Ukuran file (bytes)
27.	Pohon	*.*txt*	14.266
28.	Implementasi	*.*txt*	22.399
29.	Manajemen	*.*txt*	34.296
30.	Doslinux	*.*txt*	43.909
31.	Btooth	*.*bmp*	502
32.	Winter	*.*bmp*	1.942
33.	Any48	*.*bmp*	2.358
34.	Any78	*.*bmp*	5.494
35.	Bon	*.*bmp*	7.286
36.	Sory3	*.*bmp*	8.182
37.	Any27	*.*bmp*	10.422
38.	Val	*.*bmp*	25.150
39.	Kay	*.*bmp*	30.526
40.	Kom	*.*bmp*	39.486

4.3.2 Analisa dan Evaluasi Hasil

Hasil dari uji coba terhadap sistem untuk proses kompresi file yang berekstensi *.*doc* ditunjukkan pada tabel 4.2. Uji coba proses kompresi untuk sebuah file telah menghasilkan sebuah file terkompresi dengan ekstensi *.*Bhuf*. Dari tabel 4.2 terlihat bahwa sistem dapat menghasilkan sebuah file dengan ukuran yang lebih kecil dibandingkan dengan ukuran file aslinya. Rasio kompresi rata-rata yang dicapai oleh sistem untuk file dengan ekstensi *.*doc* adalah sebesar 2,07.

Tabel 4.2 Tabel hasil kompresi file ***.doc***

No.	Nama file	Ukuran file asli (bytes)	Ukuran file terkompresi (bytes)	Waktu proses (ms)	Rasio kompresi
1.	Kata	19.968	5.195	15.921	3,85
2.	Pengantar	20.992	8.002	64.218	2,62
3.	Datmin	28.160	13.101	224.828	2,15
4.	Data	32.768	17.385	379.407	1,88
5.	Exercises	34.816	19.175	681.094	1,82
6.	Abstraksi	40.960	22.733	355.547	1,80
7.	Bab	45.056	30.439	1.836.141	1,48
8.	Abstrak	48.128	25.214	940.250	1,90
9.	Pendahuluan	60.928	37.310	1.989.063	1,63
10.	AI	76.288	47.910	3.256.703	1,59

Hasil dari uji coba terhadap sistem untuk proses kompresi file yang berekstensi **.html* ditunjukkan pada tabel 4.3. Rasio rata-rata yang dicapai oleh sistem untuk file berekstenai **.html* adalah sebesar 1,19. Tingkat rasio kompresi untuk sebuah file sangat dipengaruhi oleh komposisi data yang bersangkutan.

Hasil dari uji coba terhadap sistem untuk proses kompresi file yang berekstensi **.txt* ditunjukkan pada tabel 4.4. Rasio kompresi rata-rata untuk file berekstensi **.txt* yang dicapai oleh sistem adalah sebesar 1,33. Proses kompresi terhadap sebuah data yang berukuran sangat kecil akan menghasilkan sebuah file dengan ukuran yang lebih besar dari aslinya. Hal ini disebabkan karena adanya tambahan *header* untuk sebuah file terkompresi. Akan tetapi hal ini juga dipengaruhi oleh komposisi dari data. Seperti terlihat pada uji coba yang dilakukan untuk file 'Mama.*txt*'. Meskipun file ini memiliki ukuran yang sangat kecil, akan tetapi proses kompresi untuk file ini dapat menghasilkan sebuah file dengan ukuran yang lebih kecil dari aslinya. Hal ini disebabkan karena file ini memiliki komposisi data yang berulang-ulang. Sehingga file dapat terkompresi secara maksimal.

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

No.	Nama file	Ukuran file asli (bytes)	Ukuran file terkompresi (bytes)	Waktu proses (ms)	Rasio kompresi
1.	Findex	1.271	1.629	1.906	0,78
2.	Huffman	4.089	3.605	6547	1,14
3.	Teko	14.201	12.370	101.937	1,15
4.	Lang	17.498	13.997	123.719	1,25
5.	Linkedlist	20.143	16.945	212.672	1,19
6.	Index	28.022	22.004	240.937	1,27
7.	Dalmassi	31.611	22.294	138.157	1,09
8.	Web	36.336	25.977	309.734	1,39
9.	ASCII	43.435	32.296	743.094	1,34
10.	ZIP	59.239	43.259	757.281	1,37

Tabel 4.4 Tabel hasil kompresi file ***.txt***

No.	Nama file	Ukuran file asli (bytes)	Ukuran file terkompresi (bytes)	Waktu proses (ms)	Rasio kompresi
1.	Mama	44	27	16	1,63
2.	Data	879	1.142	812	0,77
3.	Dokumen	3.099	2.860	4125	1,08
4.	Filosofi	5.221	4.532	10.516	1,15
5.	Flopy	7.182	5730	15625	1,25
6.	Komputer	9.084	6.416	10.844	1,42
7.	Pohon	14.266	9.458	17.984	1,51
8.	Implementasi	22.399	15.835	711	1,41
9.	Manajemen	34.296	21.567	598.797	1,59
10.	Doslinux	43.909	28.881	258.234	1,52

Tabel 4.5 Tabel hasil kompresi file ***.bmp***

No.	Nama file	Ukuran file asli (bytes)	Ukuran file terkompresi (bytes)	Waktu proses (ms)	Rasio kompresi
1.	Btooth	502	284	47	1,77
2.	Winter	1.942	860	109	2,26
3.	Any48	2.358	1.113	734	2,119
4.	Any78	5.494	1.639	1.437	3,35
5.	Bon	7.286	2.117	671	3,44
6.	Sory3	8.182	2.451	3.735	3,34
7.	Any27	10.422	3.350	3.094	3,11
8.	Val	25.150	10.223	161.672	2,46
9.	Kay	30.526	6.437	10.657	4,72
10.	Kom	39.486	11.527	77.609	3,51

Hasil dari uji coba terhadap sistem untuk proses kompresi file yang berekstensi **.bmp* ditunjukkan pada tabel 4.5. Uji coba yang dilakukan terhadap sistem untuk file dengan ekstensi **.bmp* mengasilkan rasio kompresi rata-rata sebesar 3,01. Rasio rata-rata untuk file **.bmp* merupakan rasio rata-rata terbesar yang dicapai dari uji coba yang dilakukan terhadap sistem.

Seperti yang dijelaskan pada Subbab 2.4.3, bahwa sebuah algoritma yang baik dapat menghasilkan rasio kompresi rata-rata sebesar 1,5. Sementara rasio kompresi rata-rata dari sistem Pengkompresian File dengan Menerapkan Algoritma Huffman Berbasis Model Bigram yang bernilai lebih besar dari 1,5 adalah untuk file dengan ekstensi **.doc* dan **.bmp*, sehingga dapat disimpulkan bahwa algoritma Huffman berbasis model Bigram lebih efektif jika digunakan untuk mengkompresi sebuah file dengan ekstensi **.doc* dan **.bmp.* Akan tetapi hal ini juga sangat dipengaruhi oleh komposisi dari data yang akan dikompresi.

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

Tabel 4.6 Tabel hasil dekompresi file ***.doc***

No.	Nama file	Ukuran file terkompresi (bytes)	Ukuran file hasil dekompresi	Waktu proses (ms)
1.	Kata	5.195	19.968	21.796
2.	Pengantar	8.002	20.992	15.964
3.	Datmin	13.101	28.160	53.926
4.	Data	17.385	32.768	31.175
5.	Exercises	19.175	34.816	53.082
6.	Abstraksi	22.733	40.960	25.217
7.	Bab	30.439	45.056	13.248
8.	Abstrak	25.214	48.128	8.664
9.	Pendahuluan	37.310	60.928	15.336
10.	AI	47.910	76.288	20.551

Tabel 4.7 Tabel hasil dekompresi file ***.html***

No.	Nama file	Ukuran file terkompresi (bytes)	Ukuran file hasil dekompresi	Waktu proses (ms)
1.	Findex	1.629	1.271	1.953
2.	Huffman	3.605	4.089	18.636
3.	Teko	12.370	14.201	42.120
4.	Lang	13.997	17.498	7.712
5.	Linkedlist	16.945	20.143	19.425
6.	Index	22.004	28.022	327.937
7.	Dalmassi	22.294	31.611	19.436
8.	Web	25.977	36.336	438.093
9.	ASCII	32.296	43.435	33.514
10.	ZIP	43.259	59.239	1.884.718

Tabel 4.8 Tabel hasil dekompresi file ***.txt***

No.	Nama file	Ukuran file terkompresi (bytes)	Ukuran file hasil dekompresi	Waktu proses (ms)
1.	Mama	27	44	16
2.	Data	1.142	879	891
3.	Dokumen	2.860	3.099	4.703
4.	Filosofi	4.532	5.221	12.235
5.	Flopy	5730	7.182	20.344
6.	Komputer	6.416	9.084	16.781
7.	Pohon	9.458	14.266	32.031
8.	Implementasi	15.835	22.399	15.069
9.	Manajemen	21.567	34.296	21.002
10.	Doslinux	28.881	43.909	15.288

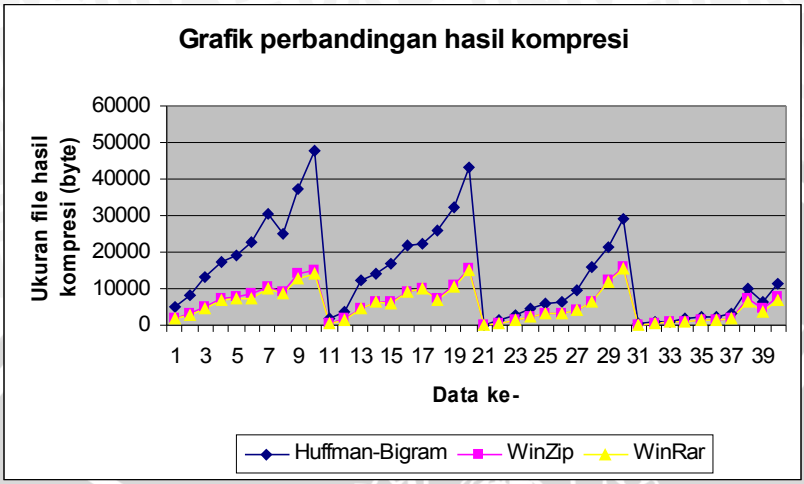
Tabel 4.9 Tabel hasil dekompresi file ***.bmp***

No.	Nama file	Ukuran file terkompresi (bytes)	Ukuran file hasil dekompresi	Waktu yang dibutuhkan (ms)
1.	Btooth	284	502	31
2.	Winter	860	1.942	171
3.	Any48	1.113	2.358	813
4.	Any78	1.639	5.494	1500
5.	Bon	2.117	7.286	6.657
6.	Sory3	2.451	8.182	3.907
7.	Any27	3.350	10.422	23.297
8.	Val	10.223	25.150	28.943
9.	Kay	6.437	30.526	16.719
10.	Kom	11.527	39.486	13.433

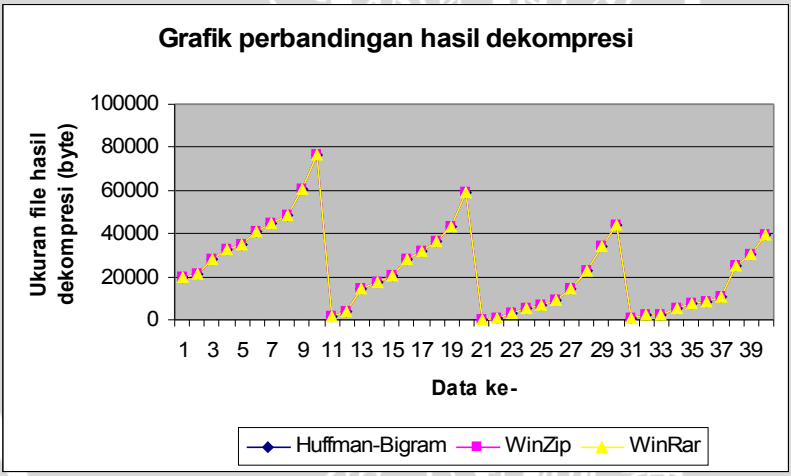
4.3.3 Perbandingan Hasil Uji Coba dengan Hasil Kompresi WinZip dan WinRar

Evalusi terhadap sistem dilakukan dengan membandingkan hasil dari sistem dengan hasil dari *software* yang telah beredar di pasaran yaitu WinZip dan WinRar. Hal ini dilakukan untuk mengetahui seberapa efektif penerapan algoritma Huffman berbasis model Bigram dalam pengkompresian sebuah file jika dibandingkan dengan aplikasi yang telah ada.

Hasil uji coba proses kompresi secara lengkap dapat dilihat pada Lampiran 2. Grafik yang ditunjukkan pada gambar 4.30 menunjukkan bahwa hasil kompresi dari sistem memiliki nilai hampir dua lebih besar jika dibandingkan dengan hasil kompresi dengan WinZip maupun WinRar. Akan tetapi dalam proses dekompresi, sistem dapat mengembalikan sebuah file terkompresi secara sempurna, sama halnya dengan WinZip dan WinRar. Hal ini terbukti dengan hasil proses dekompresi dari sistem memiliki nilai yang sama dengan hasil dekompresi dari WinZip dan WinRar. Hasil uji coba dari proses dekompresi secara lengkap dapat dilihat pada Lampiran 3. Grafik perbandingan proses dekompresi ditunjukkan pada gambar 4.31.



Gambar 4.30 Grafik perbandingan hasil kompresi

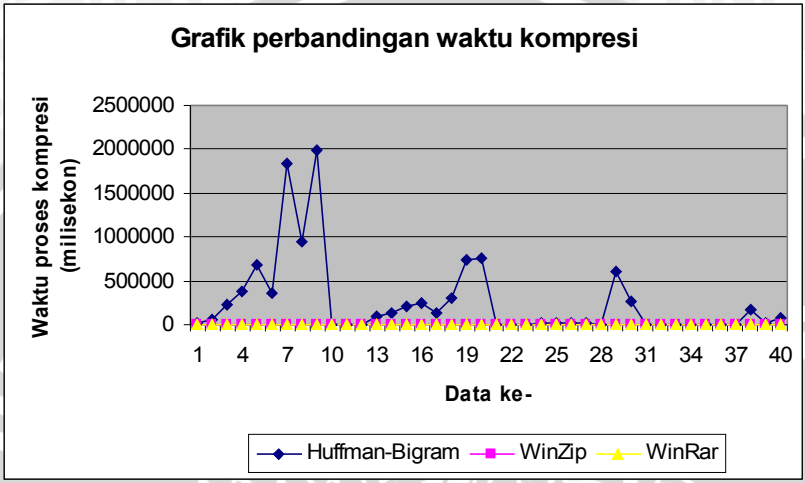


Gambar 4.31 Grafik perbandingan hasil dekompresi

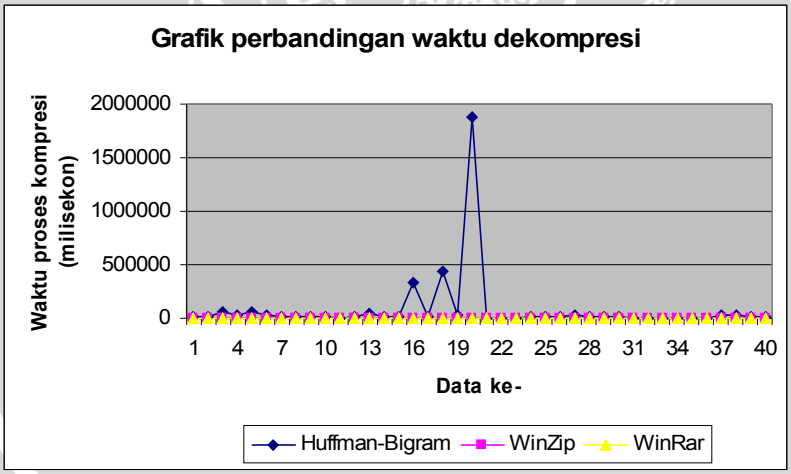
Hasil perbandingan untuk waktu kompresi secara lengkap dapat dilihat pada Lampiran 4, sedangkan hasil perbandingan untuk waktu dekompresi secara lengkap dapat dilihat pada Lampiran 5.

Dari grafik pada gambar 4.32 dan gambar 4.33 didapatkan bahwa waktu proses yang dibutuhkan oleh sistem, baik waktu

kompresi maupun waktu dekompresi, lebih besar dibandingkan dengan waktu proses yang dibutuhkan oleh WinZip dan WinRar.



Gambar 4.32 Grafik perbandingan waktu kompresi

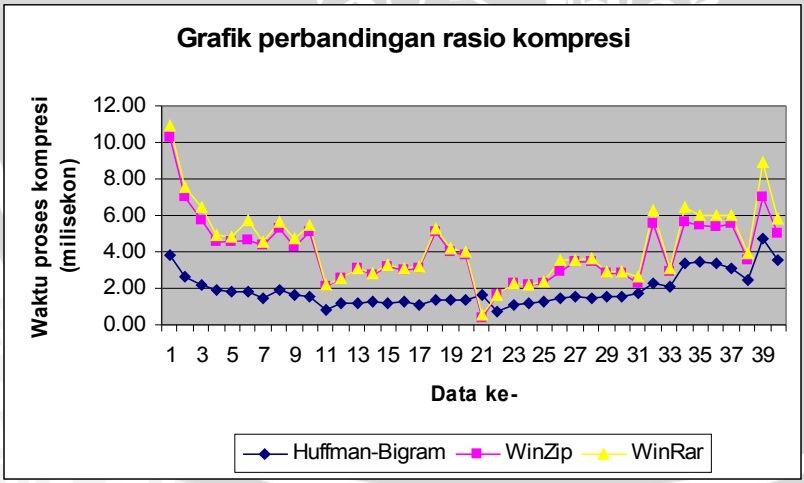


Gambar 4.33 Grafik perbandingan waktu dekompresi

Dalam sistem ini, pembacaan terhadap file akan dilakukan sebanyak dua kali yaitu pembacaan untuk pembentukan kode Huffman dan pembacaan untuk proses *encoding*. Karena kode Huffman yang digunakan dalam proses kompresi bersifat dinamis berdasarkan data *inputan,* maka setiap proses kompresi terhadap sebuah file akan membutuhkan waktu yang cukup guna pembentukan pohon Huffman.

Aplikasi WinZip menerapakan metode *deflate* yang merupakan gabungan antara algoritma Lempel Ziv 77 (LZ77) dan algoritma Huffman. Sedangkan WinRar menerapkan algoritma Lempel Ziv Markov Chain (LZMA). Seperti yang telah dijelaskan pada Subbab 2.7 bahwa aplikasi WinZip dan WinRar merupakan aplikasi yang dibentuk dengan menggabungkan beberapa metode, sehingga setiap proses dari sistem dapat dijalankan secara maksimal dengan waktu yang lebih cepat.

Grafik perbandingan rasio kompresi dari sistem dengan aplikasi WinZip dan WinRar ditunjukkan pada Gambar 4.34. Dari hasil perbandingan yang telah dilakukan,didapatkan nilai rasio kompresi rata-rata dari sistem ini yang jauh lebih kecil dibandingkan dengan nilai rasio kompresi rata-rata dari WinZip dan WinRar. Perbedaan rasio kompresi rata-rata tersebut hampir mencapai dua kali lipat. Perbandingan rasio kompresi dari uji coba yang dilakukan dapat dilihat pada Lampiran 6.



Gambar 4.34 Grafik perbandingan rasio kompresi

Sehingga dari hasil uji coba yang didapatkan tersebut dapat disimpulkan bahwa algoritma Huffman dengan berbasis Model Bigram ini dapat diterapkan untuk pengkompresian sebuah file dalam sistem ini. Akan tetapi hasil dari sistem ini tidak dapat memberikan hasil yang lebih efektif bila dibandingkan dengan aplikasi kompresi yang telah ada yaitu WinZip dan WinRar yang menggabungkan beberapa metode dalam proses pengkompresian sebuah file.


UNIVERSITAS BRAWIJAYA

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

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

1. Kode Huffman yang digunakan dalam proses *encoding* adalah bersifat dinamis sehingga kode Huffman akan berubah-ubah sesuai komposisi data yang diinputkan. Setiap kode Huffman merepresentasikan sebuah pasangan karakter, dimana setiap satu pasangan karakter terdiri dari dua karakter yang berdekatan dalam sebuah file.
2. Tabel kompresi yang dibentuk dalam proses kompresi memanfaatkan struktur data berupa *record.* Tabel kompresi berisi daftar pasangan karakter penyusun beserta frekuensi kemunculannya dan kode Huffman. Tabel kompresi ini akan dimanfaatkan dalam proses kompresi.
3. Sebuah file hasil kompresi akan terdiri dari *Header file* dan isi file. *Header file* berisi informasi dari tabel kompresi yang terbentuk. *Header file* dimanfaatkan untuk pembentukan tabel dekompresi dalam proses dekompresi. Sedangkan isi file berisi hasil kompresi dari keselurahan isi data dari file *input.*
4. Sistem Pengkompresian sebuah file dengan menerapkan algoritma Huffman berbasis Bigram ini menghasilkan ratio kompresi dengan rinciannya adalah untuk file **.doc* ratio kompresi rata-rata yang dicapai adalah sebesar 2,07 atau sekitar 52%; untuk file **.txt* ratio kompresi rata-rata yang dicapai adalah 1,33 atau sekitar 25%; untuk file **.html* ratio kompresi rata-rata yang dicapai adalah 1,19 atau sekitar 16%; sedangkan untuk file **.bmp* ratio kompresi rata-rata yang dicapai adalah sebesar 3,01 atau sekitar 67%.
5. Proses kompresi dari sebuah file sangat dipengaruhi oleh komposisi data yang di-*inputkan.* Hasil *output* dari proses kompresi tersimpan dengan nama dan direktori sesuai yang telah ditentukan oleh *user* dengan berekstensi **.Bhuf.* Proses dekompresi dari sistem yang dibuat telah dapat mengembalikan sebuah file terkompresi menjadi seperti file aslinya.

5.2 Saran

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

1. Perbaikan dalam proses pembentukan pohon Huffman Sehingga dengan demikian proses kompresi dapat berlangsung lebih cepat.
2. Pembentukan struktur *header file* yang lebih efisien sehingga memungkinkan terbentuk sebuah file terkompresi dengan ukuran yang lebih kecil

DAFTAR PUSTAKA

Anton. 2005. *Kompresi dan Teks*. Fakultas Teknik Informatika. Universitas Kristen Duta Wacana.
http://lecturer.ukdw.ac.id/anton/download/multimedia6.pdf
Tanggal akses: 12 Maret 2008

Atmavidya, Arif Nanda. 2007. *Aplikasi Pohon Dalam Teknik Kompresi Data Dengan Algoritma Huffman dan Algoritma Huffman Kanonik*. Jurusan Teknik Informatika. Institut Teknologi Bandung.

Byta. 2007. Apakah kompresi itu?.
http://www.bytastudios.com/blog/2007/10/27apakah-kompresi-itu

Fauzi, Rahmad. 2003. *Analisa Beberapa Teknik Coding*. Fakultas Teknik. Universitas Sumatra Utara.
http://library.usu.ac.id/modules.php?
op=modload&name=Downloads&file=index&req=getit&lid=556
Tanggal akses: 15 Maret 2008

Handayani, Dewi. 2001. *Sistem Berkas*. Yogyakarta: J&J Learning.

Hatem, M. 2007. "N-gram and Fast Pattern Extraction Algorithm".
http://www.codeproject.com/KB/recipes/Patterns .aspx.
Tanggal akses : 21 April 2008.

Inti.2006. *Pengantar Teknologi Informasi*.STMIK
http://www.inti.ac.id/default/stmik/Materi_Kuliah_SMT_I/PENGANTAR%20TEKNOLOGI%20INFORMASI/PTI%20-%20Pertemuan%206.pdf
Tanggal akses: 15 Maret 2008

Jando, Emanuel. 2005. *Pengantar Arsitektur dan Sistim Operasi Komputer*. Fakultas Ilmu Komputer. Universitas Indonesia.

Linawati dan Henry Panggabean. 2004. *Perbandingan Algoritma Kompresi pada Berbagai Tipe File*. FMIPA. Universitas Katolik Parahyangan Bandung.

Madenda, Sarifuddin, Hayet L. dan I. Bayu. 1996. Kompresi Citra Berwarna Menggunakan Metode Pohon Biner Huffman. Universitas Gunadarma.

Noprianto. 2004. *Adu Kemampuan Software Kompresi*. http://ilkom.unud.ac.id/infolinux/Tahun%202004/PDF%20LINUX%200804/24_Adu%20Software_08.pdf
Tanggal akses: 15 Maret 2008

Prayogo. 2008. Daftar ekstensi file. http://prayogo.wordpress.com/2008/08/29/daftar-ekstensi-file/
Tanggal akses: 15 Maret 2008

Restyandito. *Metode Statistik Kompresi Data*. http://www2.ukdw.ac.id/kuliah/info/TP4113/HO03-MetodeStatistik.pdf
Tanggal akses: 25 Maret 2008

Satrio Utomo, M, dkk. 2008. Pemampatan Data dengan Pengkodean Huffman Dinamis. Laboratorium Ilmu danRekayasa Komputasi. Departemen Teknik Informatika Institut Teknologi Bandung.

Tabel ASCII.2008.
www.LookupTables.com.
Tanggal akses: 20 November 2008

Wahyudi, Bambang. 2008. *Catatan Manajemen Basis Data (Bag. 1)*. Fakultas Ekonomi. Universitas Gunadarma.

Wardoyo, Irwan.dkk. 2005. *Kompresi Teks dengan Menggunakan Algortima Huffman.* Jurusan Teknik Informatika. Sekolah Tinggi Teknologi Telkom. Bandung.

Wibisono, Yudi. 2008. *Penggunaan Hidden Markov Model untuk Kompresi Kalimat.* Program Studi Informatika. Institut Teknologi Bandung.

Winanti, Winda. 2006. *Aplikasi Pohon Biner*. Teknik Informatika Institut Teknologi Bandung.

LAMPIRAN

Lampiran 1

Tabel ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space
1	1	001	SOH (start of heading)	33	21	041	&#33;	!
2	2	002	STX (start of text)	34	22	042	&#34;	"
3	3	003	ETX (end of text)	35	23	043	&#35;	#
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&
7	7	007	BEL (bell)	39	27	047	&#39;	'
8	8	010	BS (backspace)	40	28	050	&#40;	(
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,
13	D	015	CR (carriage return)	45	2D	055	&#45;	-
14	E	016	SO (shift out)	46	2E	056	&#46;	.
15	F	017	SI (shift in)	47	2F	057	&#47;	/
16	10	020	DLE (data link escape)	48	30	060	&#48;	0
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7
24	18	030	CAN (cancel)	56	38	070	&#56;	8
25	19	031	EM (end of medium)	57	39	071	&#57;	9
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:
27	1B	033	ESC (escape)	59	3B	073	&#59;	;
28	1C	034	FS (file separator)	60	3C	074	&#60;	<
29	1D	035	GS (group separator)	61	3D	075	&#61;	=
30	1E	036	RS (record separator)	62	3E	076	&#62;	>
31	1F	037	US (unit separator)	63	3F	077	&#63;	?

Tabel ASCII (lanjutan)

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
64	40	100	@	@	96	60	140	`	`
65	41	101	A	A	97	61	141	a	a
66	42	102	B	B	98	62	142	b	b
67	43	103	C	C	99	63	143	c	c
68	44	104	D	D	100	64	144	d	d
69	45	105	E	E	101	65	145	e	e
70	46	106	F	F	102	66	146	f	f
71	47	107	G	G	103	67	147	g	g
72	48	110	H	H	104	68	150	h	h
73	49	111	I	I	105	69	151	i	i
74	4A	112	J	J	106	6A	152	j	j
75	4B	113	K	K	107	6B	153	k	k
76	4C	114	L	L	108	6C	154	l	l
77	4D	115	M	M	109	6D	155	m	m
78	4E	116	N	N	110	6E	156	n	n
79	4F	117	O	O	111	6F	157	o	o
80	50	120	P	P	112	70	160	p	p
81	51	121	Q	Q	113	71	161	q	q
82	52	122	R	R	114	72	162	r	r
83	53	123	S	S	115	73	163	s	s
84	54	124	T	T	116	74	164	t	t
85	55	125	U	U	117	75	165	u	u
86	56	126	V	V	118	76	166	v	v
87	57	127	W	W	119	77	167	w	w
88	58	130	X	X	120	78	170	x	x
89	59	131	Y	Y	121	79	171	y	y
90	5A	132	Z	Z	122	7A	172	z	z
91	5B	133	[	[	123	7B	173	{	{
92	5C	134	\	\	124	7C	174			|
93	5D	135	]	]	125	7D	175	}	}
94	5E	136	^	^	126	7E	176	~	~
95	5F	137	_	_	127	7F	177	&#127;	DEL

Tabel ASCII (lanjutan)

128	Ç	161	í	193	┴	225	ß
129	ü	162	ó	194	┬	226	Γ
130	é	163	ú	195	├	227	π
131	â	164	ñ	196	─	228	Σ
132	ä	165	Ñ	197	┼	229	σ
133	à	166	ª	198	╞	230	µ
134	å	167	º	199	╟	231	τ
135	ç	168	¿	200	╚	232	Φ
136	ê	169	⌐	201	╔	233	Θ
137	ë	170	¬	202	╩	234	Ω
138	è	171	½	203	╦	235	δ
139	ï	172	¼	204	╠	236	∞
140	î	173	¡	205	═	237	ϕ
141	ì	174	«	206	╬	238	ε
142	Ä	175	»	207	╧	239	∩
143	Å	176	░	208	╨	240	≡
144	É	177	▒	209	╤	241	±
145	æ	178	▓	210	╥	242	≥
146	Æ	179	│	211	╙	243	≤
147	ô	180	┤	212	╘	244	⌠
148	ö	181	╡	213	╒	245	⌡
149	ò	182	╢	214	╓	246	÷
150	û	183	╖	215	╫	247	≈
151	ù	184	╕	216	╪	248	°
152	ÿ	185	╣	217	┘	249	∙
153	Ö	186	║	218	┌	250	·
154	Ü	187	╗	219	▉	251	√
156	£	188	╝	220	▄	252	ⁿ
157	¥	189	╜	221	▌	253	²
158	₧	190	╛	222	▐	254	■
159	ƒ	191	┐	223	▀	255	
160	á	192	└	224	α		

Lampiran 2

Tabel hasil perbandingan ukuran hasil proses kompresi

No	Nama file	Eks. file	Ukuran file asli (byte)	Ukuran file terkompresi (byte)		
				Huffman - Bigram	WinZip	WinRar
1.	Kata	*.doc	19.968	5.195	1.949	1.827
2.	Pengantar	*.doc	20.992	8.002	3.005	2.796
3.	Datmin	*.doc	28.160	13.101	4.883	4.390
4.	Data	*.doc	32.768	17.385	7.151	6.671
5.	Exercises	*.doc	34.816	19.175	7.592	7.280
6.	Abstraksi	*.doc	40.960	22.733	8.807	7.148
7.	Bab	*.doc	45.056	30.439	10.407	9.853
8.	Abstrak	*.doc	48.128	25.214	9.146	8.566
9.	Pendahuluan	*.doc	60.928	37.310	14.128	12.803
10.	AI	*.doc	76.288	47.910	14.978	13.910
11.	Findex	*.html	1.271	1.629	607	583
12.	Huffman	*.html	4.089	3.605	1.628	1.589
13.	Teko	*.html	14.201	12.370	4.658	4.589
14.	Lang	*.html	17.498	13.997	6.356	6.231
15.	Linkedlist	*.html	20.143	16.945	6.303	6.132
16.	Index	*.html	28.022	22.004	9.207	9.026
17.	Dalmassi	*.html	31.611	22.294	10.216	9.932
18.	Web	*.html	36.336	25.977	7.124	6.876
19.	ASCII	*.html	43.435	32.296	10.939	10.425
20.	ZIP	*.html	59.239	43.259	15.465	14.902
21.	Mama	*.txt	44	27	123	85
22.	Data	*.txt	879	1.142	549	529
23.	Dokumen	*.txt	3.099	2.860	1.374	1.347
24.	Filosofi	*.txt	5.221	4.532	2.414	2.402
25.	Flopy	*.txt	7.182	5.730	3.106	3.059
26.	Komputer	*.txt	9.084	6.416	3.083	3.029
27.	Pohon	*.txt	14.266	9.458	4.088	3.971
28.	Implementasi	*.txt	22.399	15.835	6.396	6.189

Tabel hasil perbandingan ukuran hasil proses kompresi (lanjutan)

No	Nama file	Eks. file	Ukuran file asli (byte)	Ukuran file terkompresi (byte)		
				Huffman - Bigram	WinZip	WinRar
29.	Manajemen	*.*txt*	34.296	21.567	12.181	11.788
30.	Doslinux	*.*txt*	43.909	28.881	15.799	15.277
31.	Btooth	*.*bmp*	502	284	224	188
32.	Winter	*.*bmp*	1.942	860	348	308
33.	Any48	*.*bmp*	2.358	1.113	809	760
34.	Any78	*.*bmp*	5.494	1.639	969	848
35.	Bon	*.*bmp*	7.286	2.117	1.344	1.216
36.	Sory3	*.*bmp*	8.182	2.451	1.514	1.358
37.	Any27	*.*bmp*	10.422	3.350	1.890	1.731
38.	Val	*.*bmp*	25.150	10.223	7.110	6.389
39.	Kay	*.*bmp*	30.526	6.437	4.325	3.411
40.	Kom	*.*bmp*	39.486	11.527	7.932	6.739

Lampiran 3

Tabel hasil perbandingan ukuran hasil proses dekompresi

No	Nama file	Eks. file	Ukuran terkompresi (byte)	Ukuran file hasil dekompresi		
				Huffman - Bigram	WinZip	WinRar
1.	Kata	*.*doc*	5.195	19.968	19.968	19.968
2.	Pengantar	*.*doc*	8.002	20.992	20.992	20.992
3.	Datmin	*.*doc*	13.101	28.160	28.160	28.160
4.	Data	*.*doc*	17.385	32.768	32.768	32.768
5.	Exercises	*.*doc*	19.175	34.816	34.816	34.816
6.	Abstraksi	*.*doc*	22.733	40.960	40.960	40.960
7.	Bab	*.*doc*	30.439	45.056	45.056	45.056
8.	Abstrak	*.*doc*	25.214	48.128	48.128	48.128
9.	Pendahuluan	*.*doc*	37.310	60.928	60.928	60.928
10.	AI	*.*doc*	47.910	76.288	76.288	76.288

Tabel hasil perbandingan ukuran hasil proses dekompresi (lanjutan)

No	Nama file	Eks. file	Ukuran terkompresi (byte)	Ukuran file hasil dekompresi		
				Huffman - Bigram	WinZip	WinRar
11.	Findex	*.*html*	1.629	1.271	1.271	1.271
12.	Huffman	*.*html*	3.605	4.089	4.089	4.089
13.	Teko	*.*html*	12.370	14.201	14.201	14.201
14.	Lang	*.*html*	13.997	17.498	17.498	17.498
15.	Linkedlist	*.*html*	16.945	20.143	20.143	20.143
16.	Index	*.*html*	22.004	28.022	28.022	28.022
17.	Dalmassi	*.*html*	22.294	31.611	31.611	31.611
18.	Web	*.*html*	25.977	36.336	36.336	36.336
19.	ASCII	*.*html*	32.296	43.435	43.435	43.435
20.	ZIP	*.*html*	43.259	59.239	59.239	59.239
21.	Mama	*.*txt*	27	44	44	44
22.	Data	*.*txt*	1.142	879	879	879
23.	Dokumen	*.*txt*	2.860	3.099	3.099	3.099
24.	Filosofi	*.*txt*	4.532	5.221	5.221	5.221
25.	Flopy	*.*txt*	5730	7.182	7.182	7.182
26.	Komputer	*.*txt*	6.416	9.084	9.084	9.084
27.	Pohon	*.*txt*	9.458	14.266	14.266	14.266
28.	Implementasi	*.*txt*	15.835	22.399	22.399	22.399
29.	Manajemen	*.*txt*	21.567	34.296	34.296	34.296
30.	Doslinux	*.*txt*	28.881	43.909	43.909	43.909
31.	Btooth	*.*bmp*	284	502	502	502
32.	Winter	*.*bmp*	860	1.942	1.942	1.942
33.	Any48	*.*bmp*	1.113	2.358	2.358	2.358
34.	Any78	*.*bmp*	1.639	5.494	5.494	5.494
35.	Bon	*.*bmp*	2.117	7.286	7.286	7.286
36.	Sory3	*.*bmp*	2.451	8.182	8.182	8.182
37.	Any27	*.*bmp*	3.350	10.422	10.422	10.422
38.	Val	*.*bmp*	10.223	25.150	25.150	25.150
39.	Kay	*.*bmp*	6.437	30.526	30.526	30.526
40.	Kom	*.*bmp*	11.527	39.486	39.486	39.486

Lampiran 4

Tabel hasil perbandingan waktu proses kompresi

No.	Nama file	Ekstensi file	Waktu proses (ms)		
			Huffman - Bigram	WinZip	WinRar
1.	Kata	*.doc	15.921	1.490	1.500
2.	Pengantar	*.doc	64.218	1.500	1.510
3.	Datmin	*.doc	224.828	1.500	1.530
4.	Data	*.doc	379.407	1.510	1.550
5.	Exercises	*.doc	681.094	1.500	1.500
6.	Abstraksi	*.doc	355.547	1.480	1.540
7.	Bab	*.doc	1.836.141	1.510	1.540
8.	Abstrak	*.doc	940.250	1.490	1.500
9.	Pendahuluan	*.doc	1.989.063	1.550	1.550
10.	AI	*.doc	3.256.703	1.560	1.560
11.	Findex	*.html	1.906	900	750
12.	Huffman	*.html	6.547	1.500	1.540
13.	Teko	*.html	101.937	1.400	1.430
14.	Lang	*.html	123.719	1.500	1.600
15.	Linkedlist	*.html	212.672	1.480	1.480
16.	Index	*.html	240.937	1.500	1.500
17.	Dalmassi	*.html	138.157	1.450	1.500
18.	Web	*.html	309.734	1.490	1.550
19.	ASCII	*.html	743.094	1.510	1.500
20.	ZIP	*.html	757.281	1.500	1.500
21.	Mama	*.txt	16	1.500	1.550
22.	Data	*.txt	812	1.510	1.500
23.	Dokumen	*.txt	4125	1.530	1.520
24.	Filosofi	*.txt	10.516	1.510	1.370
25.	Flopy	*.txt	15625	1.500	1.480
26.	Komputer	*.txt	10.844	1.540	1.550
27.	Pohon	*.txt	17.984	1.540	1.560
28.	Implementasi	*.txt	711	1.500	1.500
29.	Manajemen	*.txt	598.797	1.540	1.560

Tabel hasil perbandingan waktu proses kompresi (lanjutan)

No.	Nama file	Ekstensi file	Waktu proses (ms)		
			Huffman - Bigram	WinZip	WinRar
30.	Doslinux	*.txt	258.234	1.520	1.540
31.	Btooth	*.bmp	47	1.400	1.460
32.	Winter	*.bmp	109	1.600	1.840
33.	Any48	*.bmp	734	1.500	1.510
34.	Any78	*.bmp	1.437	1.300	1.340
35.	Bon	*.bmp	671	1.510	1.500
36.	Sory3	*.bmp	3.735	1.490	1.490
37.	Any27	*.bmp	3.094	1.400	1.390
38.	Val	*.bmp	161.672	1.400	1.370
39.	Kay	*.bmp	10.657	1.410	1.440
40.	Kom	*.bmp	77.609	1.410	1.420

Lampiran 5

Tabel hasil perbandingan waktu proses dekompresi

No.	Nama file	Ekstensi file	Waktu proses (ms)		
			Huffman - Bigram	WinZip	WinRar
1.	Kata	*.doc	21.796	2.010	1.940
2.	Pengantar	*.doc	15.964	2.000	1.930
3.	Datmin	*.doc	53.926	2.010	1.860
4.	Data	*.doc	31.175	1.900	2.020
5.	Exercises	*.doc	53.082	1.900	1.890
6.	Abstraksi	*.doc	25.217	2.010	2.000
7.	Bab	*.doc	13.248	1.900	1.920
8.	Abstrak	*.doc	8.664	2.000	2.070
9.	Pendahuluan	*.doc	15.336	2.010	2.020
10.	AI	*.doc	20.551	1.900	1.970
11.	Findex	*.html	1.953	1.900	1.900
12.	Huffman	*.html	18.636	1.930	1.930

Tabel hasil perbandingan waktu proses dekompresi (lanjutan)

No.	Nama file	Ekstensi file	Waktu proses (ms)		
			Huffman - Bigram	WinZip	WinRar
13.	Teko	*.*html*	42.120	1.970	1.970
14.	Lang	*.*html*	7.712	1.900	1.920
15.	Linkedlist	*.*html*	19.425	1.900	1.930
16.	Index	*.*html*	327.937	1.910	1.970
17.	Dalmassi	*.*html*	19.436	1.920	1.960
18.	Web	*.*html*	438.093	1.930	1.940
19.	ASCII	*.*html*	33.514	1.920	1.940
20.	ZIP	*.*html*	1.884.718	1.910	1.960
21.	Mama	*.*txt*	16	1.900	1.930
22.	Data	*.*txt*	891	1.920	1.960
23.	Dokumen	*.*txt*	4.703	1.900	1.920
24.	Filosofi	*.*txt*	12.235	1.920	1.920
25.	Flopy	*.*txt*	20.344	1.950	1.970
26.	Komputer	*.*txt*	16.781	1.960	1.970
27.	Pohon	*.*txt*	32.031	1.900	1.930
28.	Implementasi	*.*txt*	15.069	1.900	1.880
29.	Manajemen	*.*txt*	21.002	1.970	1.960
30.	Doslinux	*.*txt*	15.288	1.950	1.990
31.	Btooth	*.*bmp*	31	2.000	2.020
32.	Winter	*.*bmp*	171	1.990	2.030
33.	Any48	*.*bmp*	813	1.960	1.970
34.	Any78	*.*bmp*	1500	1.980	2.010
35.	Bon	*.*bmp*	6.657	1.950	1.960
36.	Sory3	*.*bmp*	3.907	1.930	1.940
37.	Any27	*.*bmp*	23.297	1.950	1.960
38.	Val	*.*bmp*	28.943	1.890	1.870
39.	Kay	*.*bmp*	16.719	1.900	1.920
40.	Kom	*.*bmp*	13.433	1.950	1.970

Lampiran 6

Tabel hasil perbandingan rasio kompresi

No.	Nama file	Ekstensi file	Rasio kompresi		
			Huffman - Bigram	WinZip	WinRar
1.	Kata	*.doc	3,85	10,25	10,91
2.	Pengantar	*.doc	2,62	6,99	7,51
3.	Datmin	*.doc	2,15	5,77	6,41
4.	Data	*.doc	1,88	4,58	4,91
5.	Exercises	*.doc	1,82	4,59	4,78
6.	Abstraksi	*.doc	1,80	4,65	5,73
7.	Bab	*.doc	1,48	4,33	4,57
8.	Abstrak	*.doc	1,90	5,26	5,62
9.	Pendahuluan	*.doc	1,63	4,31	4,76
10.	AI	*.doc	1,59	5,09	5,48
11.	Findex	*.html	0,78	2,09	2,18
12.	Huffman	*.html	1,14	2,51	2,57
13.	Teko	*.html	1,15	3,05	3,09
14.	Lang	*.html	1,25	2,75	2,81
15.	Linkedlist	*.html	1,19	3,19	3,28
16.	Index	*.html	1,27	3,04	3,10
17.	Dalmassi	*.html	1,09	3,09	3,18
18.	Web	*.html	1,39	5,10	5,28
19.	ASCII	*.html	1,34	3,97	4,17
20.	ZIP	*.html	1,37	3,83	3,98
21.	Mama	*.txt	1,63	0,36	0,52
22.	Data	*.txt	0,77	1,60	1,66
23.	Dokumen	*.txt	1,08	2,26	2,30
24.	Filosofi	*.txt	1,15	2,16	2,17
25.	Flopy	*.txt	1,25	2,31	2,35
26.	Komputer	*.txt	1,42	2,95	3,59
27.	Pohon	*.txt	1,51	3,49	3,59
28.	Implementasi	*.txt	1,41	3,50	3,62
29.	Manajemen	*.txt	1,59	2,81	2,91
30.	Doslinux	*.txt	1,52	2,78	2,87

Tabel hasil perbandingan rasio kompresi (lanjutan)

No.	Nama file	Ekstensi file	Rasio kompresi		
			Huffman - Bigram	WinZip	WinRar
31.	Btooth	*.*bmp*	1,77	2,24	2,67
32.	Winter	*.*bmp*	2,26	5,58	6,31
33.	Any48	*.*bmp*	2,12	2,91	3,10
34.	Any78	*.*bmp*	3,35	5,67	6,48
35.	Bon	*.*bmp*	3,44	5,42	5,99
36.	Sory3	*.*bmp*	3,34	5,40	6,03
37.	Any27	*.*bmp*	3,11	5,51	6,02
38.	Val	*.*bmp*	2,46	3,54	3,94
39.	Kay	*.*bmp*	4,72	7,01	8,95
40.	Kom	*.*bmp*	3,51	4,98	5,86