ANALISIS PERBANDINGAN ALGORITMA LZW DAN HUFFMAN PADA KOMPRESI FILE GAMBAR BMP DAN PNG

Andika Satyapratama, Widjianto, Mahmud Yunus
Program Studi Teknik Informatika, STMIK Pradnya Paramita Malang
E-mail : andy.pratama91@gmail.com

Abstract

File compression is one of important aspect in the development of information technology. The demands of providing information in a short time with large number and size of files makes compression techniques very important. There are a lot of algorithms that have been developed for the compression of files, including: LZW algorithm and Huffman Algorithm. This study discusses the comparative analysis between the two algorithms are against compressed image file format BMP and PNG. Analysis of this study aims to provide knowledge about what is the best algorithm between both of the algorithm to compress the image file format BMP and PNG. This analysis is comparing the ratio and the estimated time of each algorithm for image file format BMP and PNG. The author uses the Java programming language to implement both LZW and Huffman algorithm, and then analyze the process and results of compression on the file format BMP and PNG. After testing and analysis of experiments the compression process, it is concluded that the compression process BMP and PNG file format, Huffman algorithm can compress files faster than LZW algorithm, but the LZW algorithm can produce better compression ratio than Huffman algorithm.

Keywords: *Image file compression, LZW Algorithm, Huffman Algorithm, BMP, PNG*

1. PENDAHULUAN

Saat ini kompresi *file* merupakan salah satu aspek penting dalam dunia teknologi informasi. Seiring dengan perkembangan teknologi, tuntutan untuk menyajikan informasi dengan kualitas terbaik menyebabkan jumlah ukuran suatu *file* menjadi lebih besar. Sebagai contoh pada *file* gambar/citra dimana semakin baik kualitas gambar yang dihasilkan, maka ukuran *pixel* yang dibutuhkan untuk merekam gambar tersebut semakin besar, sehingga berimbas pada ukuran *file* yang harus disimpan pada media penyimpanan.

Format citra yang diulas adalah format BMP dan PNG. Kedua format ini cukup populer dan sering digunakan untuk menyimpan *file* citra. Format BMP merupakan format yang dikembangkan oleh Microsoft yang terdiri dari susunan titik (*pixel*) yang tersimpan di memori komputer. Format BMP merupakan format yang minim kompresi, sehingga ukurannya cukup besar. Sedangkan Format PNG (*Portable Network Graphics*) merupakan format yang menggunakan metode kompresi *lossless* dan sering digunakan pada *website* dikarenakan ukurannya yang relatif lebih kecil sehingga mempercepat akses *website* itu sendiri.

Saat ini terdapat banyak sekali algoritma yang dapat digunakan pada proses kompresi *file* gambar, beberapa diantaranya adalah algoritma LZW (Lempel-Ziv-Welch) dan algorima Huffman. Algoritma LZW melakukan kompresi dengan menggunakan metode *dictionary*, dimana fragmen-fragmen data digantikan dengan indeks yang diperoleh dari sebuah "kamus". Prinsip sejenis juga digunakan dalam kode Braille, dimana kode-kode khusus digunakan untuk merepresentasikan kata-kata yang ada. Sedangkan algoritma Huffman menggunakan prinsip pengkodean yang mirip dengan kode Morse, yaitu tiap karakter (simbol) dikodekan hanya dengan rangkaian beberapa *Bit*, dimana karakter yang sering muncul dikodekan

dengan rangkaian *Bit* yang pendek dan karakter yang jarang muncul dikodekan dengan rangkaian *Bit* yang lebih panjang.

Kedua algoritma tersebut dipilih karena merupakan algoritma yang populer dan merupakan algoritma yang menjadi dasar dari beberapa algortima yang lainnya.

Rumusan masalah penelitian ini adalah membandingkan algoritma LZW dan algoritma Huffman pada hasil proses kompresi *file* gambar berformat BMP dan PNG, dengan batasan-batasan sebagai berikut:

1. Perbandingan yang dilakukan, diantaranya mencakup: estimasi waktu dan rasio perbandingan kompresi *file*.
2. Bahasa pemrograman yang digunakan untuk implementasi kedua algoritma adalah Bahasa Java.

2. KAJIAN LITERATUR

2.1 Kompresi

Kompresi adalah suatu teknik pemampatan data sehingga diperoleh *file* dengan ukuran yang lebih kecil daripada ukuran aslinya (Andre Anggo Siu, 2006: 6). Kompresi bekerja dengan mencari pola-pola perulangan pada data dan menggantinya dengan sebuah penanda tertentu.

Metode pemampatan data atau kompresi data dapat dikelompokan dalam dua kelompok besar, yaitu:

1. Metode *lossless*

Lossless data kompresi adalah kelas dari algoritma data kompresi yang memungkinkan data yang asli dapat disusun kembali dari data kompresi. Kompresi data *lossless* digunakan dalam berbagai aplikasi seperti format ZIP dan GZIP. *Lossless* juga sering digunakan sebagai komponen dalam teknologi kompresi data *lossy*. Kompresi *Lossless* digunakan ketika sesuatu yang penting pada kondisi asli. Beberapa format gambar sperti PNG atau GIF hanya menggunakan kompresi *lossless*, sedangkan yang lainnya sperti TIFF dan MNG dapat menggunakan metode *lossy* atau *lossless*.

Metode *lossless* menghasilkan data yang identik dengan data aslinya, hal ini dibutuhkan untuk banyak tipe data, contohnya: *executablecode*, *word processingfiles*, *tabulated numbers* dan sebagainya. Misalnya pada citra atau gambar dimana metode ini menghasilkan hasil yang tepat sama dengan citra semula, *pixel* per *pixel* sehingga tidak ada informasi yang hilang akibat kompresi. Namun *ratio* kompresi (Rasio kompresi yaitu, ukuran *file* yang dikompresi dibanding yang tak terkompresi dari *file*) dengan metode ini sangat rendah. Metode ini cocok untuk kompresi citra yang mengandung informasi penting yang tidak boleh rusak akibat kompresi, misalnya gambar hasil diagnosa medis. Contoh metode *lossless* adalah metode *run-length*, Huffman, delta dan LZW.

2. Metode *lossy*

Lossy kompresi adalah suatu metode untuk mengkompresi data dan men-dekompresinya, data yang diperoleh mungkin berbeda dari yang aslinya tetapi cukup dekat perbedaaanya. *Lossy* kompresi ini paling sering digunakan untuk mengkompres data multimedia (*Audio* dan gambar statis). Sebaliknya, kompresi *lossless* diperlukan untuk data teks dan *file*, seperti catatan bank, artikel teks dll.

Format kompresi *lossy* mengalami *generation loss*, yaitu jika dilakukan berulang kali kompresi dan dekompresi *file* akan menyebabkan kehilangan kualitas secara progresif. hal ini berbeda dengan kompresi data *lossless*. ketika pengguna yang menerima *file* terkompresi secara *lossy* (misalnya untuk mengurangi waktu *download*) *file* yang diambil dapat sedikit berbeda dari yang asli di level *Bit* ketika tidak dapat dibedakan oleh mata dan telinga manusia untuk tujuan paling praktis.

2.2 Algoritma LZW

LZW merupakan kependekan kata dari Lempel-Ziv-Welch. Abraham Lempel, Jacob Ziv, dan Terry Welch adalah pencipta algoritma kompresi *lossless universal* ini. Kelebihan algoritma ini yaitu cepat dalam implementasi dan kekurangannya kurang optimal karena

hanya melakukan analisis terbatas pada data (Muhammad Maulana Abdullah, 2008: 3).

Algoritma ini melakukan kompresi dengan menggunakan kamus, dimana fragmen-fragmen teks digantikan dengan indeks yang diperoleh dari sebuah "kamus". Pendekatan ini bersifat adaptif dan efektif karena banyak karakter dapat dikodekan dengan mengacu pada *string* yang telah muncul sebelumnya dalam teks.

Berikut Algoritma LZW secara lengkap:

1. *Dictionary* (kamus) diinisialisasi dengan semua karakter dasar yang ada : {'A'..'Z','a'..'z','0'..'9'}.
2. W <- karakter pertama dalam *stream* karakter.
3. K <- karakter berikutnya dalam *stream* karakter.
4. Lakukan pengecekan apakah (W+K) terdapat dalam *Dictionary*
   a. Jika ya, maka W <- W + K (gabungkan W dan K menjadi *string* baru).
   b. Jika tidak, maka :
      - *Output* sebuah kode untuk menggantikan *string* W.
      - Tambahkan *string* (W+ K) ke dalam *dictionary* dan berikan nomor/kode berikutnya yang belum digunakan dalam *dictionary* untuk *string* tersebut.
      - W <- K.
5. Lakukan pengecekan apakah masih ada karakter berikutnya dalam *stream* karakter.
   -Jika ya, maka kembali ke langkah 2.
   -Jika tidak, maka *output* kode yang menggantikan *string* W, lalu terminasi proses (stop).

Proses dekompresi data pada algoritma LZW tidak jauh berbeda dengan proses kompresinya. Pada dekompresi LZW, juga dibuat tabel *dictionary* dari data input kompresi, sehingga tidak diperlukan penyertaan tabel *dictionary* ke dalam data kompresi. Berikut algoritma dekompresi LZW secara lengkap:

1. *Dictionary* diinisialisasi dengan semua karakter dasar yang ada : {'A'..'Z','a'..'z','0'..'9'}.
2. CW kode pertama dari *stream* salah satu karakter dasar).
3. Lihat *dictionary* dan *output string* dari kode tersebut (*string*.CW) ke *stream* karakter.
4. PW <- CW; CW <- kode berikutnya dari *stream* kode.
5. Apakah *string*.CW terdapat dalam *dictionary*?
   a. Jika ada, maka :
      - *Output string*.CW ke *stream* karakter
      - P <- *string*.PW
      - C <- karakter pertama dari *string*.CW
      -Tambahkan *string* (P+C) ke dalam *dictionary*
   b. Jika tidak, maka :
      - P <- *string*.PW
      - C <- karakter pertama dari *string*.PW
      - *Output string* (P+C) ke *stream* tambahkan *string* tersebut ke dalam (sekarang berkorespondensi dengan CW)
6. Apakah terdapat kode lagi di *stream* code?
   a. Jika ya, maka kembali ke langkah 4.
   b. Jika tidak, maka terminasi proses (stop).

2.3 Algoritma Huffman

Algortima Huffman adalah algoritma yang dikembangkan oleh David A. Huffman pada jurnal yang ditulisnya sebagai prasyarat kelulusannya di MIT. Konsep dasar dari metode Huffman adalah dengan membangun sebuah skema atau tabel yang berisikan frekuensi kemunculan masing-masing simbol. Dari tabel tersebut kemudian dibangun suatu kode-kode unik untuk mengidentifikasikan masing-masing simbol.

Kode Huffman salah satu algoritma dasar untuk kompresi data, yang bertujuan untuk mengurangi jumlah *Bit* yang diperlukan untuk merepresentasikan informasi/pesan. Di bawah ini adalah algoritma yang digunakan untuk membuat Kode Huffman (Ata Amrullah, 2010:3) :

procedure Huffman(C:symbols a, *with* frequencies w_i, i = 1,…,n)
F : = forest of n rooted trees, each consisting of single node a_i and assign weight w_i
while F is not a tree
begin
Replace the rooted trees T and T' of least weight from F with w(T) ≥ w(T') with a tree having a new root that has T as its left subtree and T' as its right subtree. Label new edge to T with 0 and the new edge to T' with 1.
Assign w(T) + w(T') as the weight of the new tree.
end
{the Huffman coding for the symbol a, is the concatenation of the labels of the edges in the unique path from the root to the node a_i}

Algoritma di atas adalah algoritma Huffman yang digunakan untuk membuat kode Huffman. Penjelasan mengenai algoritma di atas adalah sebagai berikut :

procedure Huffman(C:symbols aiwith frequencies wi , i = 1, …. , n).

Algoritma ini menunjukkan sebuah prosedur atau fungsi yang menggunakan simbol ai dengan frekuensi wi yang menunjukkan besar probabilitas kemunculan dari simbol tersebut dalam suatu deretan *string* yang berisi informasi/pesan tertentu.

F : = forest of n rooted trees, each consisting of single vertex ai and assign weight w_i.

F didefinisikan sebagai sebuah *forest* yang berisi sekumpulan *node* tunggal ai(*tree*) dan memiliki frekuensi wi seperti yang telah disebutkan sebelumnya.

while F is not a tree

Selama *forest* F masih memiliki lebih dari sebuah *tree* maka proses yang ada di bawahnya akan dijalankan terus.

begin

Replace the rooted trees T and T' of least weight from F with w(T) ≥ w(T') with a tree having a new root that has T as its left subtree and T' as its right subtree. Label new edge to T with 0 and the new edge to T with 1.

Assign w(T) + w(T') as the weight of the new tree.

end

Ganti dua *root* pohon T dan T' yang memiliki frekuensi terendah di dalam F dengan w(T) > w(T') dengan *tree* baru yang memiliki *root* dengan T sebagai *subtree* sebelah kiri dan T' sebagai *subtree* sebelah kanan. Beri label 0 untuk *edge* yang menuju T dan label 1 untuk *edge* yang menuju T'. Jadikan w(T) + w(T') sebagai frekuensi bagi *tree* yang baru dibentuk.

Sampai disini proses selesai tetapi tidak menandakan selesainya pembentukan *tree* secara total. Hal ini disebabkan proses pembentukan *tree* secara total akan selesai ketika hanya tinggal satu bulan *tree* di dalam *forest* F.

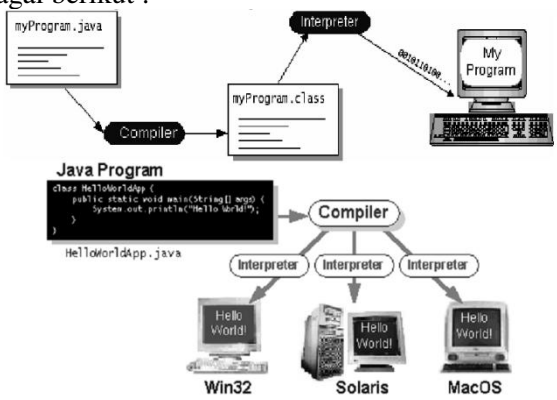
{the Huffman Coding for the symbol ai is the concatenation of the labels of the edges in the unique path from the root to the node ai}

Buat kode dari setiap simbol dengan menggunakan *tree* yang telah dibangun tersebut dengan menggabungkan label dari setiap *edge* dari arah *root* menuju ke *node* ai secara unik.

2.4 Java

Java adalah bahasa pemrograman berorientasi objek murni yang dibuat berdasarkan kemampuan-kemampuan terbaik bahasa pemrograman objek sebelumnya (C++, Ada, Simula). Java diciptakan oleh James Gosling, *developer* dari Sun Microsystems pada tahun 1991.

Cara kerja Java dapat digambarkan sebagai berikut :



Java mempunyai beberapa *platform*, yaitu :

- *Java Virtual Machine* (Java VM)

- *JavaApplication Programming Interface* (Java API)

2.5 Citra BMP

File format BMP bisa disebut juga *bitmap* atau format *file* DIB (untuk perangkat independen *bitmap*), adalah sebuah *file* gambar format yang digunakan untuk menyimpan gambar digital *bitmap*, terutama pada Microsoft Windows dan OS / 2 sistem operasi. Banyak pengguna antarmuka grafis menggunakan *bitmap* dalam membangun subsistem grafis, misalnya Microsoft Windows dan OS / 2 *platform*s' GDI subsistem, dimana format tertentu yang digunakan adalah Windows dan OS / 2 format *file bitmap*, biasanya ekstensi *file* .BMP atau .DIB. Adapun struktur *file* .BMP adalah sebagai berikut :

Offset	***Size***	***Name***	***Description***
0	2	*bfType*	*ASCII "BM"*
2	4	*bfSize*	*Size of file (in bytes)*
6	2	*bfReserved I*	*Zero*
8	2	*bfReserved I*	*Zero*
10	4	*bfOffBits*	*Byte offset in file where image begins*
14	4	*bfSize*	*Size of this header (40 bytes)*
18	4	*biWidth*	*Image width in pixels*
22	4	*biHeight*	*Image height in pixels*
26	2	*biPlanes*	*Number of image planes, must be I*
28	2	*biBitCount*	*Bits per pixel : 1,4,8 or 24*
30	4	*biCompression*	*Compression type*

Format BMP, disebut dengan *bitmap* atau format DIB (*Device Independent Bitmap*) adalah sebuah format citra yang digunakan untuk menyimpan citra *bitmap* digital terutama pada sistem operasi Microsoft Windows atau OS/2. Pada citra berformat BMP (*bitmap*) yang tidak terkompresi, piksel citra disimpan dengan kedalaman warna 1, 4, 8, 16, 24, atau 32 *Bit* per piksel (Lidya, 2013: 14).

Pada umumnya citra *bitmap* terdiri dari 4 blok data yaitu: BMP header, *Bit* Information (DIB header), *Color Palette*, dan *Bitmap* Data. BMP header berisi informasi umum dari citra *bitmap*. Blok ini berada pada bagian awal *file* citra dan digunakan untuk mengidentifikasi citra. Beberapa aplikasi pengolah citra akan membaca blok ini untuk memastikan bahwa citra tersebut berformat *bitmap* dan tidak dalam kondisi rusak. *Bit* information berisi informasi detail dari citra *bitmap*, yang akan digunakan untuk menampilkan citra pada layar. *Color palette* berisi informasi warna yang digunakan untuk indeks warna *bitmap*, dan *bitmap* data berisi data citra yang sebenarnya, piksel per piksel.

2.6 Citra PNG

Format PNG (dibaca ping), merupakan pengembangan dari format gif. Format png menawarkan kelebihan daripada keterbatasan yang ada dalam format gif, di antaranya dukungan warna yang lebih kaya (gif hanya 256 warna) hingga 32 *Bit* (24 *Bit* jika tanpa warna *alpha*) atau 2 pangkat 32 warna atau 4.294.967.296 warna dengan alpha dan 16.777.216 warna tanpa alpha. Dengan adanya alpha *channel* pada png.

Tipe *file* PNG merupakan solusi kompresi yang powerfull dengan warna yang lebih banyak (24 *Bit* RGB + *alfa*). Berbeda dengan JPEG yang menggunakan teknik kompresi yang menghilangkan data, *file* PNG menggunakan kompresi yang tidak menghilangkan data (*lossless compression*).

3. ANALISIS DAN PERANCANGAN

Kompresi data melalui proses encoding berusaha untuk menghilangkan unsur pengulangan ini dengan mengubahnya sedemikian rupa sehingga ukuran data menjadi lebih kecil. Proses pengurangan unsur pengulangan ini dapat dilakukan

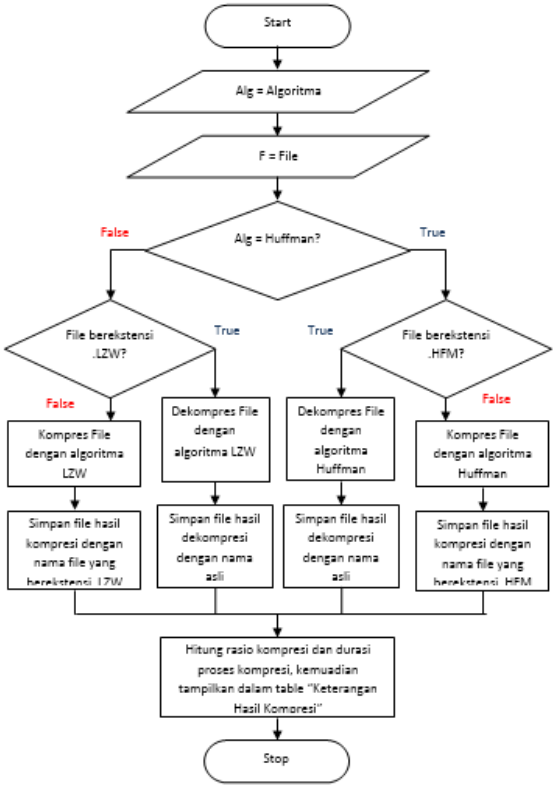
dengan memakai beberapa teknik kompresi. Misalnya jika suatu komponen muncul berulang kali dalam suatu data, maka komponen tersebut tidak harus dikodekan berulang kali pula tetapi dapat dikodekan dengan menulis frekuensi munculnya komponen dan di mana komponen tersebut muncul.

Teknik kompresi data lainnya, berusaha untuk mencari suatu bentuk kode yang lebih pendek untuk suatu komponen yang sering muncul. Keberhasilan pengkompresian data tergantung dari besarnya data itu sendiri dan tipe data yang memungkinkan untuk dikompresi. Biasanya beberapa komponenkomponen di dalam data yang sifatnya lebih umum dari yang lainnya banyak dipakai pada algoritma kompresi data yang memanfaatkan sifat ini. Hal ini dinamakan *redundancy*. Makin besar *redundancy* di dalam data semakin tinggi pula tingkat keberhasilan kompresi data.

3.1 *Flowchart* Alur Kerja Aplikasi

Sebelum aplikasi dibangun, maka langkah yang harus dilakukan adalah membuat *Flowchart* dari aplikasi yang dibuat dengan tujuan untuk mempermudah pembuatan aplikasi tersebut.

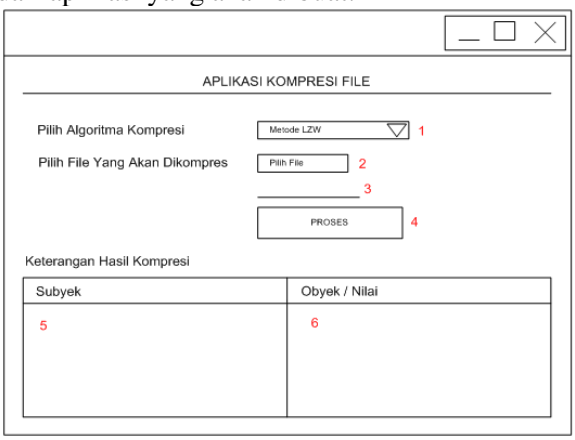
Selain itu, dengan adanya *Flowchart*, dapat mempermudah bagi orang lain untuk mengetahui bagaimana cara kerja dari aplikasi yang dibuat. Di bawah ini adalah gambar *Flowchart* dari aplikasi kompresi yang dibuat.



3.2 Perancangan Aplikasi

3.2.1 Perancangan Tampilan Awal Aplikasi

Berikut adalah tampilan dari rancangan dari aplikasi yang akan dibuat.



Keterangan :

1. *Combobox* untuk memilih algoritma apa yang digunakan untuk proses kompresi, yaitu Algoritma LZW atau Algoritma Huffman

2. Tombol "Pilih *File*" untuk menampilkan jendela yang dapat digunakan untuk memilih *file* yang akan dikompres.
3. Label untuk menampilkan nama *file* beserta alamat (*path*) dari *file* yang telah dipilih.
4. Tombol "Proses" untuk memulai melakukan proses kompresi/dekompresi dari *file* yang telah dipilih berdasarkan algoritma yang telah dipilih.
5. Kolom subyek pengukuran berdasarkan hasil proses kompresi *file*.
6. Kolom obyek / nilai pengukuran berdasarkan hasil proses kompresi *file*.

3.2.2 Perancangan Aplikasi

Pada bagian ini diuraikan beberapa skrip pemrograman java yang diketik untuk membuat aplikasi kompresi. Bagian yang diulas hanya bagian-bagian yang terpenting saja. Berikut adalah pembahasannya.

3.2.2.1 *Class Compression*.java

Merupakan induk / *main class* yang dieksekusi pertama kali ketika aplikasi dijalankan. Berisi sintak yang berfungsi untuk memanggil form utama.

```
package compressionnew;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
/**...4 lines */
public class CompressionNew {
    public static void main(String[] args) throws
            IllegalAccessException,
            ClassNotFoundException, InstantiationExcept
            UnsupportedLookAndFeelException {
        // TODO code application logic here
        UIManager.setLookAndFeel(UIManager.
                getSystemLookAndFeelClassName());
        new MainForm().setVisible(true);
    }
}
```

3.2.2.2 *Class* MainForm.java

Pada *class* ini berisi skrip yang berfungsi untuk meng-generate tampilan form utama dari aplikasi yang dibuat. Pada *class* ini terdapat beberapa skrip yang berfungsi untuk meng-handle action ketika sebuah tombol diklik, beberapa diantaranya adalah tombol "Pilih *File*" dan tombol "Proses".

Berikut adalah skrip yang berfungsi untuk meng-handle action ketika tombol "Pilih *File*" diklik.

```
private void selectFileBtnActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        int returnVal = fc.showOpenDialog(MainForm.this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file   = fc.getSelectedFile();
            //This is where a real application would open the file.
            fileName    = file.getName();
            filePath    = file.getPath();
            fileNameLbl.setText(filePath);
        }
    } catch (Exception err) {
        System.out.println("error disini "+err);
    }
}
```

Pada skrip diatas, setelah tombol "Pilih *File*" diklik, maka akan ditampilkan jendela baru untuk memilih *file* yang akan dikompres atau didekompres. Setelah *file* dipilih, maka alamat/*path* dan nama *file* ditampilkan pada label dibawah tombol "Pilih *File*".

3.2.2.3 *Class* LZW.java

Merupakan *class* yang berisi beberapa *function* untuk mengimplementasikan algoritma LZW, baik proses kompresi maupun dekompresinya. Di bawah ini merupakan penjabaran dari skripnya.

```
package compressionnew;

import java.io.*;
import java.util.*;

public class LZW
{

    private static int dictionary[][]=new int[4000][4000];
    private static int pattern[]=new int[4000];
    private static int element[]=new int[4000];
    private static int chars[]=new int[256];
    private static int chfreq[]=new int[256];
    private static float chprobs[]=new float[256];
    private static int ofreq[]=new int[4000];
    private static int dfree,psize,esize;
    private static int top,tc;
    private static int chnext;

    public static void main(String args[])
    {
        String infile,outfile;
        Scanner cons=new Scanner(System.in);
        int ch=-1;

        while(ch!=3)
        {
            System.out.println("1. Compress a file");
            System.out.println("2. Decompress a file");
            System.out.println("3. Exit\n");
            ch=Integer.parseInt(cons.nextLine());
            System.out.println();

            if(ch==1)
            {
                System.out.print("Input file: ");
                infile=cons.nextLine();
                System.out.print("Output file: ");
                outfile=cons.nextLine();
                ini();
                compress(infile,outfile);
                File in=new File(infile);
                File out=new File(outfile);
```

```
            float cr=(float)out.length()/(float)in.length(
            float entropy=calcEntropy();
            float acl=calcAvCodeLength();
            System.out.println();
            System.out.print("Entropy of the source: "+
                    entropy+"\n");
            System.out.print("Compression ratio: "+cr+"\n"
            System.out.print("Average code length in bytes
                    acl+"\n\n");
        }
        else if(ch==2)
        {
            System.out.print("Input file: ");
            infile=cons.nextLine();
            System.out.print("Output file: ");
            outfile=cons.nextLine();
            ini();
            decompress(infile,outfile);
            System.out.print("\n\n");
        }
    }
}
```

3.2.2.4 *Class* HuffmanCompress.java

Merupakan *class* yang berisi beberapa *function* untuk mengimplementasikan proses kompresi *file* dengan menggunakan algoritma Huffman. Di bawah ini merupakan penjabaran dari skripnya.

```
package huffmancoding;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
public final class HuffmanCompress {
    public static void main(String input, String output)
            throws IOException {
        File inputFile = new File(input);
        File outputFile = new File(output);
        FrequencyTable freq = getFrequencies(inputFile);
        freq.increment(256);  // EOF symbol gets a frequency of 1
    CodeTree code = freq.buildCodeTree();
    CanonicalCode canonCode = new CanonicalCode(code, 257);
    code = canonCode.toCodeTree();
    InputStream in = new BufferedInputStream(
            new FileInputStream(inputFile));
    BitOutputStream out = new BitOutputStream(new
    BufferedOutputStream(new FileOutputStream(outputFile)));
    try {
        writeCode(out, canonCode);
        compress(code, in, out);
    } finally {
        out.close();
        in.close();
    }
}
private static FrequencyTable getFrequencies(File file)
        throws IOException {
    FrequencyTable freq = new FrequencyTable(new int[257]);
    InputStream input = new BufferedInputStream(
            new FileInputStream(file));
    try {
        while (true) {
            int b = input.read();
            if (b == -1) {
                break;
            }
            freq.increment(b);
        }
    } finally {
        input.close();
```

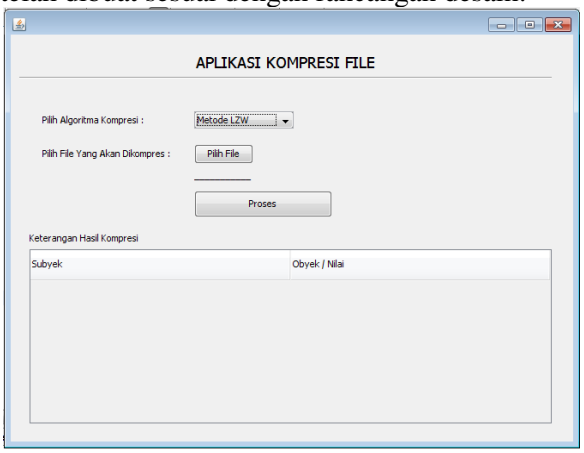
```
    return freq;
}

static void writeCode(BitOutputStream out, CanonicalCode canonCode)
        throws IOException {
    for (int i = 0; i < canonCode.getSymbolLimit(); i++) {
        int val = canonCode.getCodeLength(i);
        if (val >= 256) {
            throw new RuntimeException(
                    "The code for a symbol is too long");
        }
        for (int j = 7; j >= 0; j--) {
            out.write((val >>> j) & 1);
        }
    }
}

static void compress(CodeTree code,
        InputStream in, BitOutputStream out)
        throws IOException {
    HuffmanEncoder enc = new HuffmanEncoder(out);
    enc.codeTree = code;
    while (true) {
        int b = in.read();
        if (b == -1) {
            break;
        }
        enc.write(b);
    }
    enc.write(256);  // EOF
}
}
```

4. PENGUJIAN DAN HASIL

4.1 Uji Coba Desain Tampilan Aplikasi

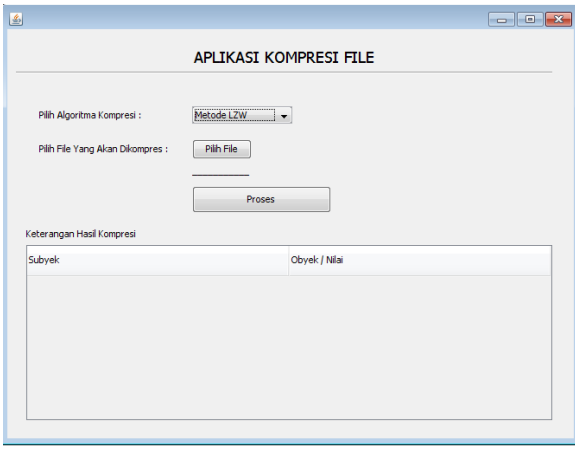
Pada rancangan desain aplikasi di Bab sebelumnya terdapat beberapa komponen, diantaranya: *Combobox* untuk memilih metode algoritma kompresi LZW atau Huffman, tombol untuk memilih *file* yang dikompres dan didekompres, tombol untuk memulai proses kompresi dan dekompresi, serta tabel yang berisi keterangan-keterangan berdasarkan hasil proses kompresi.

Berikut adalah tampilan dari aplikasi yang telah dibuat sesuai dengan rancangan desain:

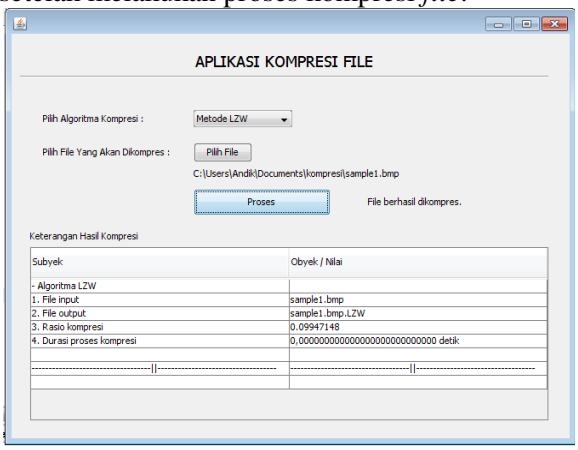


4.2 Hasil Implementasi Rancangan Alur Kerja Aplikasi

Ketika aplikasi pertama kali dijalankan, maka secara default algoritma yang terpilih untuk proses kompresi adalah algoritma LZW.



Berikut adalah tampilan dari aplikasi setelah melakukan proses kompresi *file*.



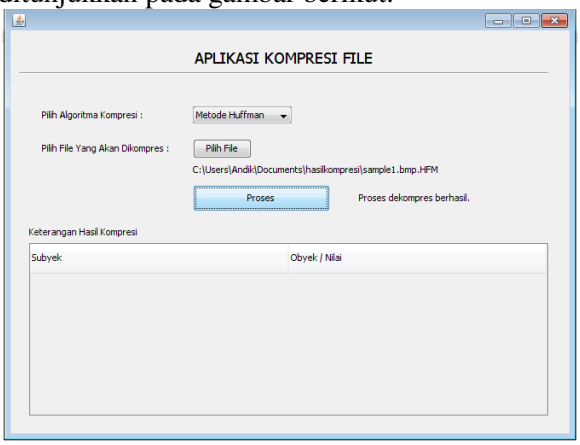
File output dari hasil kompresi yang telah dilakukan dengan algoritma LZW diberi tambahan ekstensi .LZW. Sedangkan apabila menggunakan algoritma Huffman, maka nama *file output* diberi tambahan ekstensi .HFM.

4.3 Uji Coba Proses Dekompresi

Uji coba selanjutnya adalah proses dekompresi *file* yang telah di kompres. Langkah pertama adalah memilih *file* sesuai dengan metode yang digunakan. Apabila metode yang digunakan adalah metode Huffman, maka yang harus dipilih adalah *file* berekstensi .HFM. Begitu juga sebaliknya apabila metode yang digunakan adalah metode LZW, maka *file* yang dipilih harus berekstensi .LZW.

Setelah *file* dipilih, dan terdeteksi bahwa *file* berekstensi .LZW atau .HFM, maka aplikasi secara otomatis melakukan proses dekompresi terhadapa *file* tersebut.

Selanjutnya, proses dekompresi dilaksanakan ketika tombol "Proses" diklik, kemudian ditampilkan pesan "Proses dekompresi berhasil" apabila berhasil, seperti ditunjukkan pada gambar berikut:



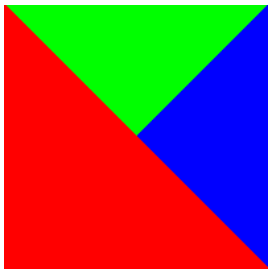
Setelah proses dekompresi selesai, maka nama dari *file output* dikembalikan seperti semula tanpa ekstensi .LZW atau .HFM.

4.4 Implementasi Proses Kompresi *File* Menggunakan Metode LZW.

a. *File* Berformat BMP.

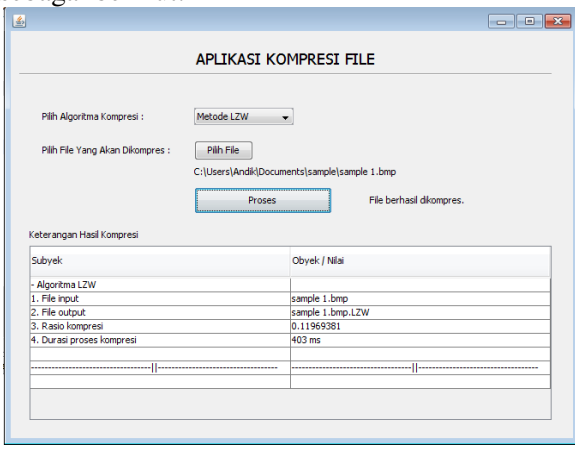
Pada pembahasan selanjutnya dilakukan uji coba / implementasi proses kompresi *file* berformat BMP dengan menggunakan metode LZW.

Berikut adalah *file* gambar bernama *sample* 1.bmp yang dijadikan obyek percobaan.



File tersebut berformat BMP dan mempunyai ukuran *file* sebesar 19,6 KB dengan dimensi 200 *pixel* X 200 *pixel*.

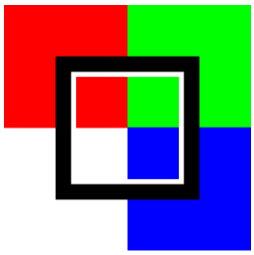
Setelah dilakukan proses kompresi dengan metode LZW, maka didapat hasil *output* sebagai berikut:



b. *File* Berformat PNG.

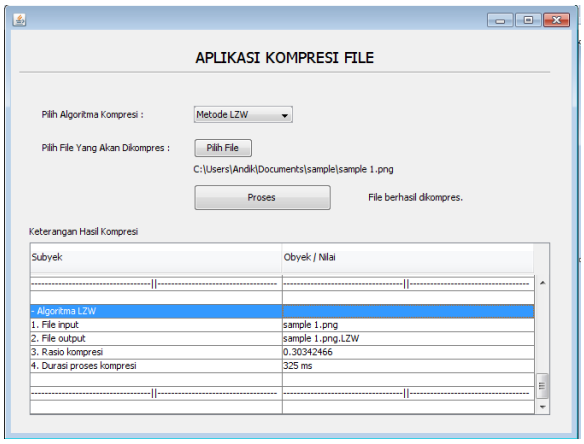
Selanjutnya dilakukan uji coba atau implementasi proses kompresi *file* berformat PNG dengan menggunakan metode LZW.

Berikut adalah *file* gambar bernama *sample* 1.png yang dijadikan obyek percobaan.



File tersebut berformat PNG, memiliki ukuran sebesar 8,55 KB dengan dimensi 1200 *pixel* X 1200 *pixel*.

Setelah dilakukan proses kompresi dengan metode LZW, didapat hasil *output* sebagai berikut:

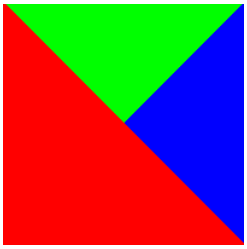


4.5 Implementasi Proses Kompresi *File* Menggunakan Metode Huffman.

a. *File* Berformat BMP.

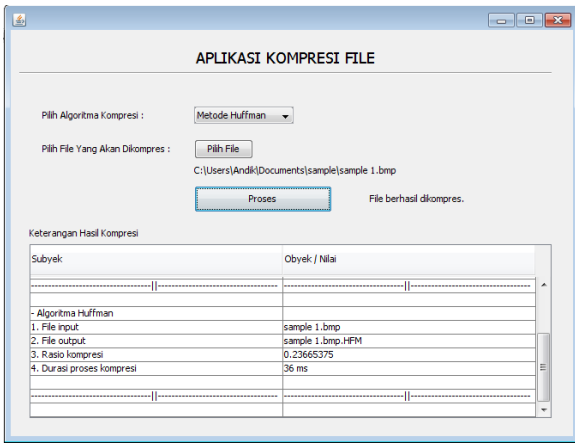
Setelah implementasi dengan menggunakan metode LZW, selanjutnya dilakukan uji coba/implementasi proses kompresi *file* berformat BMP dengan menggunakan metode Huffman.

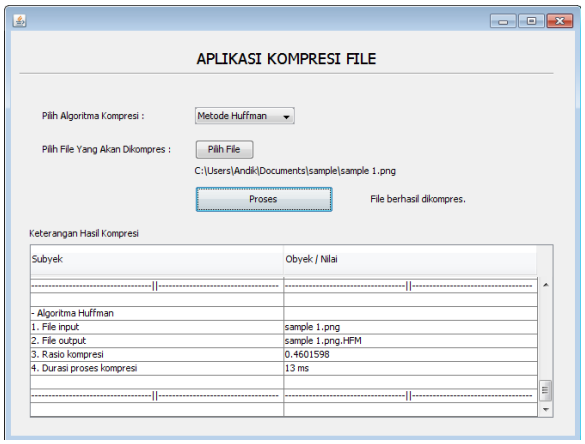
Pada bagian ini digunakan *file* gambar yang sama dengan implementasi Metode LZW sebelumnya, dengan tujuan sebagai perbandingan. Berikut adalah *file* gambar bernama *sample* 1.bmp yang dijadikan obyek percobaan.



File tersebut berformat BMP dan mempunyai ukuran *file* sebesar 19,6 KB dengan dimensi 200 *pixel* X 200 *pixel*.

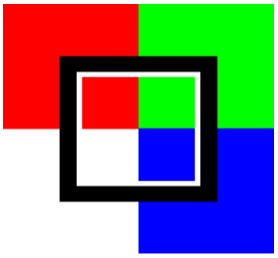
Setelah dilakukan proses kompresi dengan menggunakan metode Huffman, maka didapat hasil *output* sebagai berikut:





b. *File* Berformat PNG.

Selanjutnya dilakukan uji coba/implementasi proses kompresi *file* berformat PNG. Dengan menggunakan *file* yang sama dengan implementasi metode LZW, berikut adalah *file* gambar bernama *sample* 1.png yang dijadikan obyek percobaan.



File tersebut berformat PNG, memiliki ukuran sebesar 8,55 KB dengan dimensi 1200 *pixel* X 1200 *pixel*.

Setelah dilakukan proses kompresi dengan metode Huffman, didapat hasil *output* sebagai berikut:

4.6 Percobaan Kompresi Pada *File* Gambar Berformat BMP

Berikut merupakan hasil dari 10 percobaan proses kompresi *file* gambar berformat BMP dengan metode LZW dan Huffman.

No.	Nama *File*	Ukuran Awal	Metode	Waktu	Ukuran Kompresi	Rasio
1	sample 1.bmp	19,6 KB	LZW	403 ms	2,35 KB	12%
2	sample 1.bmp	19,6 KB	Huffman	36 ms	4,64 KB	24%
3	sample 2.bmp	47,3 KB	LZW	1279 ms	2,74 KB	6%
4	sample 2.bmp	47,3 KB	Huffman	47 ms	10,5 KB	22%
5	sample 3.bmp	17 KB	LZW	343 ms	1, 79 KB	11%
6	sample 3.bmp	17 KB	Huffman	16 ms	4,04 KB	24%
7	sample 4.bmp	21,3 KB	LZW	437 ms	1,97 KB	9%
8	sample 4.bmp	21,3 KB	Huffman	0 ms	4,98 KB	23%
9	sample 5.bmp	13,1 KB	LZW	250 ms	1,65	13%
10	sample 5.bmp	13,1 KB	Huffman	16 ms	3,23 KB	25%

Berdasarkan dari 10 percobaan kompresi yang terlampir pada tabel diatas, maka dapat disimpulkan bahwa pada *file* gambar berformat BMP, metode LZW lebih unggul dalam hal rasio kompresi dan dapat menghasilkan ukuran *file* yang lebih kecil. Namun, dalam hal kecepatan, kompresi dengan metode Huffman lebih unggul dibandingkan dengan metode LZW.

4.7 Percobaan Kompresi Pada *File* Gambar Berformat PNG

Berikut merupakan hasil dari 10 percobaan proses kompresi *file* gambar berformat PNG dengan metode LZW dan Huffman.

No.	Nama *File*	Ukuran Awal	Metode	Waktu	Ukuran Kompresi	Rasio
1	sample 1.png	8,55 KB	LZW	325 ms	2,59 KB	30%
2	sample 1.png	8,55 KB	Huffman	13 ms	3,93 KB	46%
3	sample 2.png	3,98 KB	LZW	347 ms	3,51 KB	88%
4	sample 2.png	3,98 KB	Huffman	32 ms	3,82 KB	96%
5	sample 3.png	4,1 KB	LZW	405 ms	3,64 KB	87%
6	sample 3.png	4,1 KB	Huffman	16 ms	3,94 KB	95%
7	sample 4.png	4,02 KB	LZW	343 ms	3,55 KB	88%
8	sample 4.png	4,02 KB	Huffman	15 ms	3,85 KB	94%
9	sample 5.png	3,99 KB	LZW	343 ms	3,51 KB	88%
10	sample 5.png	3,99 KB	Huffman	16 ms	3,82 KB	95%

Berdasarkan dari 10 percobaan kompresi pada *file* gambar berformat PNG, dapat disimpulkan hasil yang sama dengan kompresi pada format BMP, yaitu metode LZW lebih unggul dalam hal rasio kompresi, namun lebih lambat dibandingkan dengan metode Huffman.

5. KESIMPULAN

5.1 Kesimpulan

Percobaan kompresi *file* yang dilakukan sebanyak total 20 kali, menghasilkan kesimpulan sebagai berikut :

1. Algoritma Huffman dapat mengkompres *file* BMP dan PNG lebih cepat dibandingkan dengan algoritma LZW
2. Pada *file* gambar berformat BMP dan PNG, algoritma LZW dapat menghasilkan rasio kompresi dengan persentase yang lebih kecil dibandingkan dengan algoritma Huffman, sehingga dapat menghasilkan *file* kompresi dengan ukuran *file* yang lebih kecil.

5.2 Saran

Berikut adalah beberapa saran untuk penelitian-penelitian berikutnya:

1. Untuk membuat aplikasi kompresi yang unggul dalam kecepatan, maka sebaiknya menggunakan algoritma Huffman.
2. Sedangkan untuk membuat aplikasi kompresi yang dapat menghasilkan rasio kompresi yang lebih baik, sebaiknya menggunakan algoritma LZW.

6. REFRENSI

Abdullah, Muhammad Maulana. 2003. ***Kompresi String Menggunakan Algoritma LZW dan Huffman***. Jurnal Ilmu Komputer dan Teknologi Informasi ITB Vol III No. 2. 1-5.

Amrullah, Ata. 2010. ***Kompresi dan Enkripsi SMS dengan Metode Huffman Code dan Algoritma Enigma***. Jurnal Teknik Informatika Politeknik Elektonika Negeri Surabaya. 1-7.

Azanuddin. 2013. ***Aplikasi Kompresi Teks SMS Pada Mobile Device dengan Menggunakan Algoritma Huffman Kanonik***. Jurnal Pelita Informatika Budi Darma Vol III. 28-34.

Black, Rex. 2009. ***Managing the Testing Process : Practical Tools and Techniques for Managing Hardware and Software Testing, 3rd Edition.*** Indiana : Wiley Publishing Inc.

Fitriansah, Ahmad. 2008. ***Analisis Perbandingan Kinerja Algoritma Kompresi LZW, Huffman dan Deflate pada Berbagai Jenis File***. (http://digilib.tes.telkomuniversity.ac.id/*index*.php?option=com_content&view=article&id=214:algoritma-lzw-cont&catid=20:informatika&Itemid=14 tanggal 12 Januari 2015 jam 10:00).

Indriati, Merli. 2009. ***Pengenalan Java***. (http://merlindriati.staff.gunadarma.ac.id/*Downloads*/*files*/34028/1_Pengenalan_Java.pdf tanggal 10 Desember 2014 jam 14:00)

Novita, Rahma. 2011. ***Studi dan Implementasi Kompresi File Audio Memanfaatkan Metode Adaptive Arithmatic Coding***. (http://repository.usu.ac.id/handle/123456

789/26098 tanggal 12 Januari 2015 jam 8:00).

Pamungkas, Gatot Mardi. 2009. ***Pemantauan Ruangan Menggunakan Webcam sebagai Pengambil Gambar Berbasis AT89S52***. (http://elib.unikom.ac.id/gdl.php?mod=browse&op=read&id=jbptunikompp-gdl-gatotmardi-15820&newlang=indonesian tanggal 14 Januari 2015 jam 7:30)

Saragih, Echolima. 2010. ***Penerapan Daubechies Wavelet Dan Hamming Code Dalam Watermarking Citra Digital***. (http://repository.usu.ac.id/handle/123456789/19833 tanggal 14 Januari 2015 jam 7:00).

Seftiani, Adinda Reny. 2012. ***Analisis Kualitas Visual Pada Hasil Citra Kompresi Dengan Menggunakan Metode Run Length Encoding***. (http://repository.usu.ac.id/handle/123456789/31325 tanggal 12 Januari 2015 jam 14:00).

Siu, Andre Anggo. 2006. ***Analisis Perbandingan Algoritma Run Length, Huffman dan Half Byte Untuk Pemampatan Data***. Jurnal Ilmu Komputer dan Matematika Universitas Bina Nusantara. 6-48

Subarkah, Aan Fuad. 2009. ***Rancang Bangun Aplikasi Kompresi File Menggunakan Metode LZW Berbasis Java***. (http://lib.uin-malang.ac.id/?mod=th_detail&id=05550055 tanggal 12 Januari 2015 jam 12:00).

Triajiwati, Dessy Febriani. 2014. ***Analisis Perbandingan Zero Compresion Dengan Difference Coding Pada Kompresi File Audio***. (http://repository.usu.ac.id/handle/123456789/42385 tanggal 4 Januari 2015 jam 17:00).