CHAPTER 1

INTRODUCTION TO LZW COMPRESSION

1.1 Brief Description :

Lempel–Ziv–Welch (LZW) is a universal lossless data compression algorithm created by **Abraham Lempel, Jacob Ziv, and Terry Welch**. It was published by Welch in 1984 as an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978. The algorithm is simple to implement, and has the potential for very high throughput in hardware implementations.

The code that the LZW algorithm outputs can be of any arbitrary length, but it must have more bits in it than a single character. The first 256 codes (when using eight bit characters) are by default assigned to the standard character set. The remaining codes are assigned to strings as the algorithm proceeds. The sample program runs as shown with 12 bit codes. This means codes 0-255 refer to individual bytes, while codes 256-4095 refers to substrings.

LZW compression became the first widely used **universal data compression** method on computers. A large English text file can typically be compressed via LZW to about half its original size.

LZW was used in the **public-domain program compress**, which became a more or less standard utility in UNIX systems circa 1986. It has since disappeared from many distributions, both because it infringed the LZW patent and because **gzip** produced better compression ratios using the **LZ77-based DEFLATE algorithm**, but as of 2008 at least FreeBSD includes both compress and uncompress as a part of the distribution. Several other popular compression utilities also used LZW, or closely related methods.

LZW became very widely used when it became part of the **GIF image format in 1987**. It may also (optionally) be used in TIFF and PDF files. Although **LZW** is available in **Adobe Acrobat software**, Acrobat by default uses DEFLATE for most text and color-table-based image data in PDF files.

Abraham Lempel is an Israeli computer scientist and one of the fathers of the LZ family of lossless data compression algorithms. His historically important works start with the presentation of the LZ77 and LZW algorithm in a paper entitled "A Universal Algorithm for Sequential Data Compression" .

Jacob Ziv is an Israeli computer scientist who, along with Abraham Lempel, developed the LZ family of lossless data compression algorithms.

Terry A. Welch is an American computer scientist. Along with Abraham Lempel and Jacob Ziv, he developed the lossless LZW compression algorithm which was published in 1984.

1.2 Problem Definition :

The Problem is to Compress a large English text file can typically be via LZW to about half its original size. The Original Compressed file consists of two parts: code part and Dictionary Part. The Resultant obtained could be decompressed to get the Original file back.

1.3 Objectives :

The project mainly concentrates on the methodology used in compression of a file rather than the size of the compressed file.

The principle objectives of the project are listed below:

- ✓ Better understanding of the LZW compression algorithm.
- ✓ Creating clear view working of compression by automatically displaying the following components to the end user :
    - Compressed file size and Initial file size
    - Dictionary is created during encoding of the file.
    - Final compressed code.
    - The File is assumed to be text document format.

CHAPTER 2

DESIGN AND IMPLEMENTATION

2.1 DESIGN DEVELOPMENT

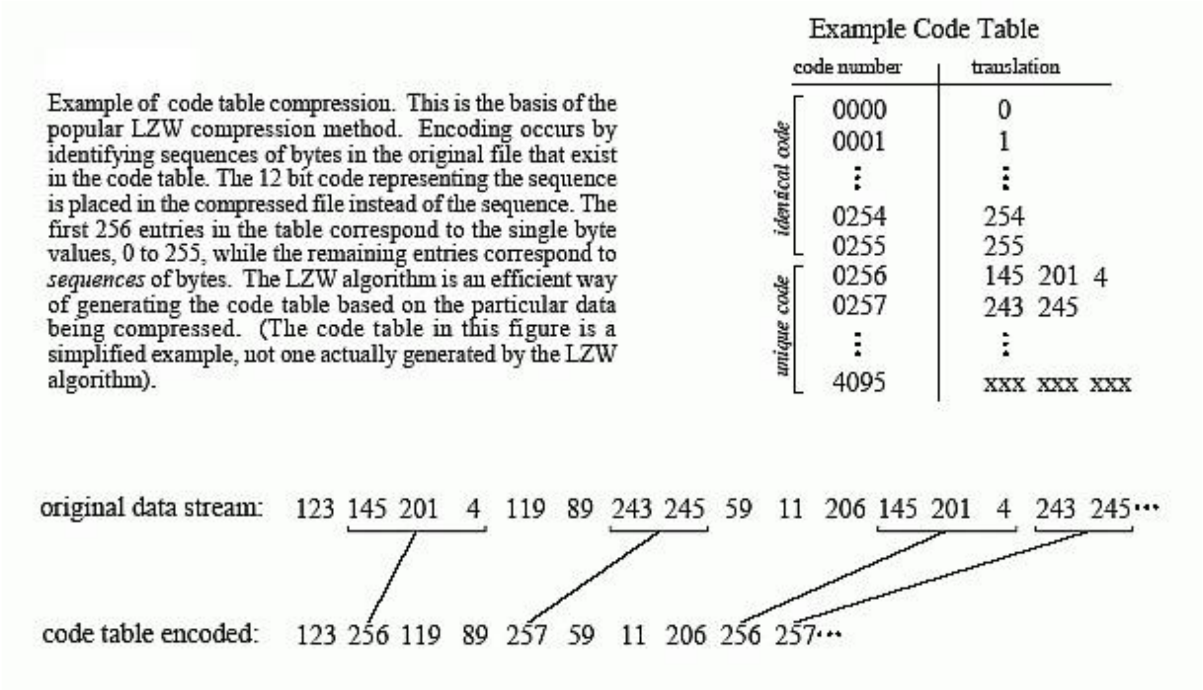
HARDWARE REQUIREMENTS:

- Processor : Pentium i3 Processor, 2.3Ghz
- RAM : 4GB
- Hard Disk : 500GB
- Display: Intel HD graphics(1366X768)
- Input Device: Key Board and Mouse

SOFTWARE REQUIREMENTS

- Operating System: Windows 7 or Ubuntu 10.04
- Other Software's:
    - JDK (Java Development Kit)
    - JRE (Java Runtime Environment)
- User Interface: Net beans IDE

2.2 WORKING:



Example of code table compression. This is the basis of the popular LZW compression method. Encoding occurs by identifying sequences of bytes in the original file that exist in the code table. The 12 bit code representing the sequence is placed in the compressed file instead of the sequence. The first 256 entries in the table correspond to the single byte values, 0 to 255, while the remaining entries correspond to *sequences* of bytes. The LZW algorithm is an efficient way of generating the code table based on the particular data being compressed. (The code table in this figure is a simplified example, not one actually generated by the LZW algorithm).

Example Code Table

	code number	translation
identical code	0000	0
	0001	1
	⋮	⋮
	0254	254
	0255	255
unique code	0256	145 201 4
	0257	243 245
	⋮	⋮
	4095	xxx xxx xxx

original data stream: 123 145 201 4 119 89 243 245 59 11 206 145 201 4 243 245···

code table encoded: 123 256 119 89 257 59 11 206 256 257···

The Original file is given as Input to **lzwcompress.class** file; it is got by successful java compilation of **lzwcompress.java**. This class file in turn creates **dict.txt** which contains unique symbols in the original file and **code.txt** which is compressed form of original file that is transmitted.

Lzwdecompress.class is got by successful java compilation of **lzwdecompress.java**. This **.class** file on execution takes input from **dict.txt** and **code.txt** and gets the original file back.

2.3 FLOW CHART:

The Flowcharts for Encoding and Decoding are shown below:

Encoding:

A high level view of the encoding is shown here:

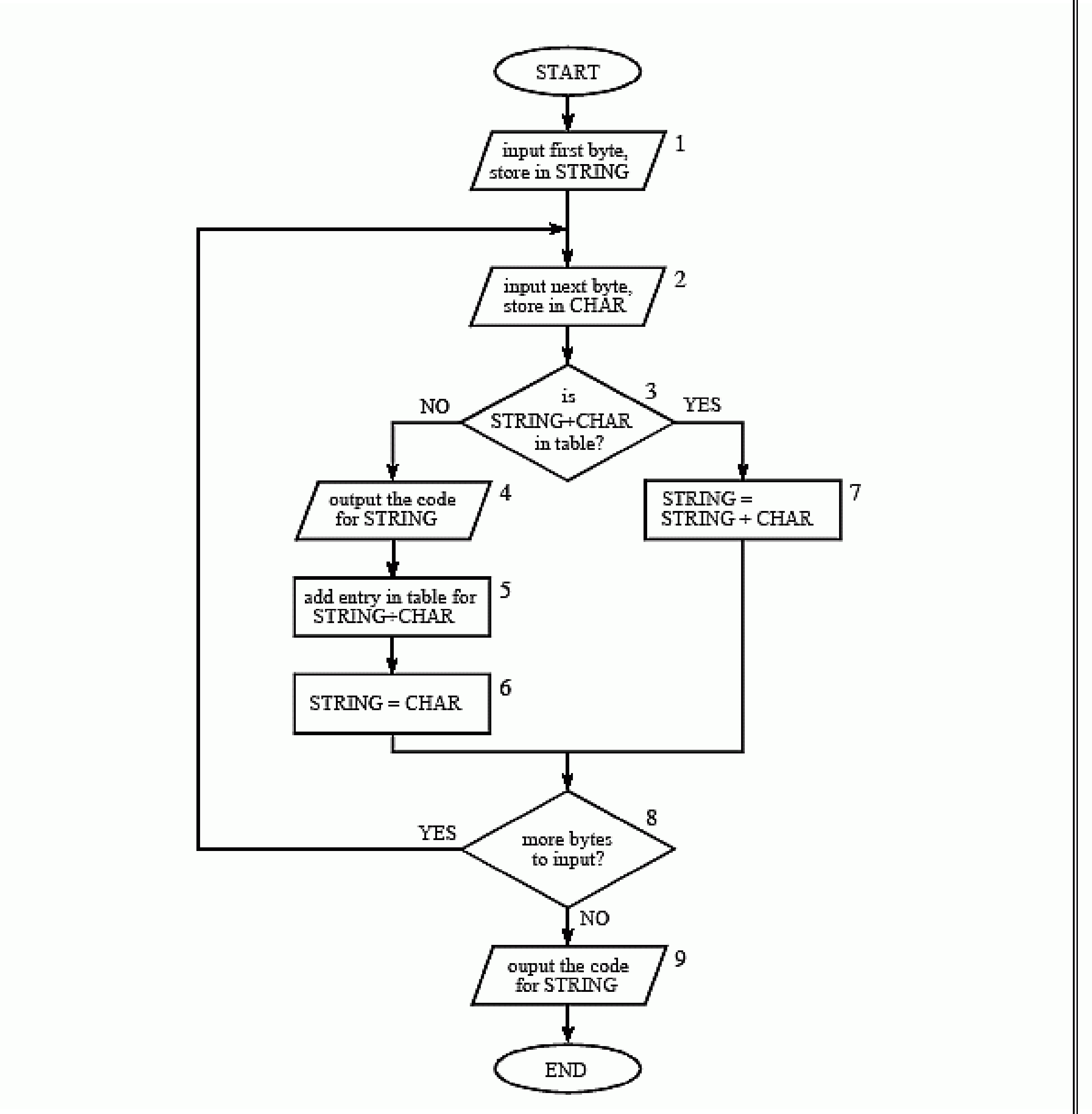
Initialize the dictionary to contain all strings of length one.

A.)Find the longest string W in the dictionary that matches the current input (B).

B.)Emit the dictionary index for W to output and remove W from the input.

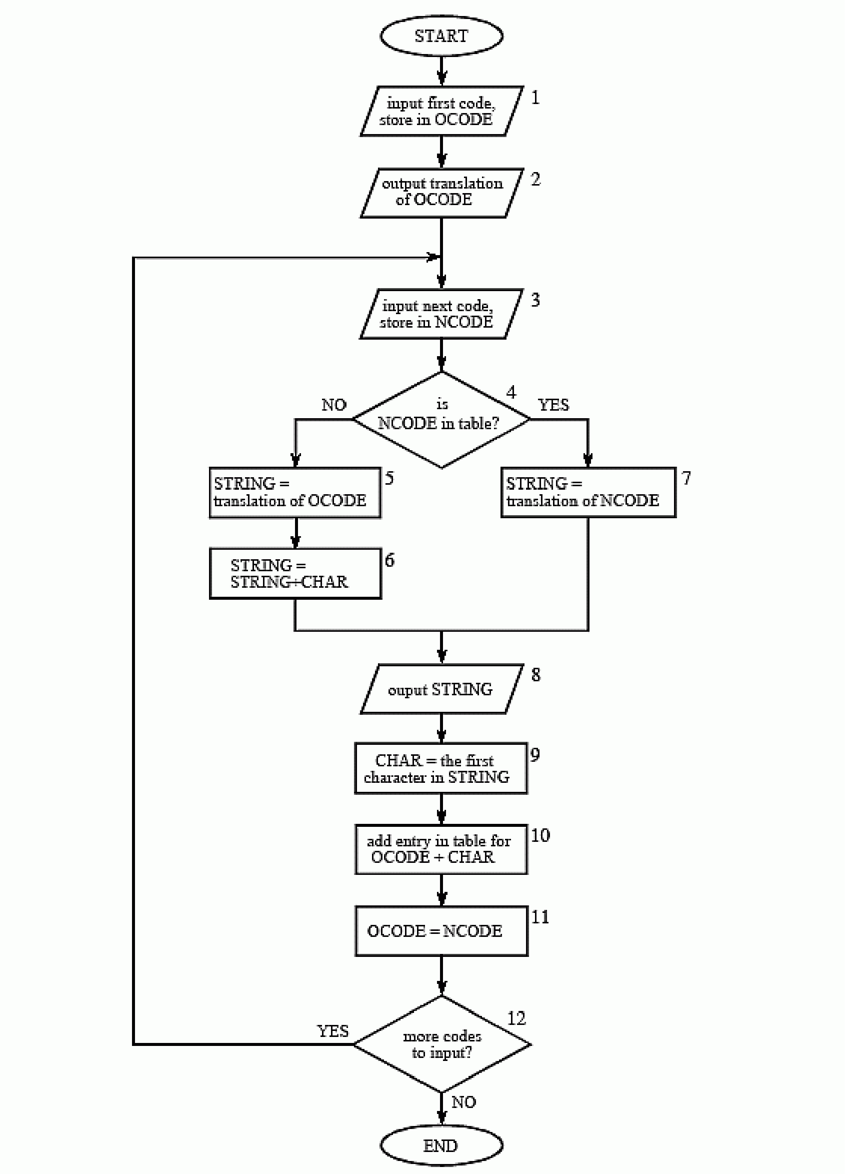
C.)Add W followed by the next symbol in the input to the dictionary.

D.)Go to Step B.


START
input first byte, store in STRING
1
input next byte, store in CHAR
2
is STRING+CHAR in table?
3
NO
YES
output the code for STRING
4
add entry in table for STRING+CHAR
5
STRING = CHAR
6
STRING = STRING + CHAR
7
more bytes to input?
8
YES
NO
ouput the code for STRING
9
END

A dictionary is initialized to contain the single-character strings corresponding to all the possible input characters. The algorithm works by scanning through the input string for successively longer substrings until it finds one that is not in the dictionary. When such a string is found, the index for the string less the last character is retrieved from the dictionary and sent to output, and the new string is added to the dictionary with the next available code. The last input character is then used as the next starting point to scan for substrings. In this way, successively longer strings are registered in the dictionary and made available for subsequent encoding as single output values. The algorithm works best on data with repeated patterns, so the initial parts of a message will see little compression. As the message grows, however, the compression ratio tends asymptotically to the maximum.

Decoding:



The decoding algorithm works by reading a value from the encoded input and outputting the corresponding string from the initialized dictionary. At the same time it obtains the next value from the input, and adds to the dictionary the concatenation of the string just output and the first character of the string obtained by decoding the next input value.

The decoder then proceeds to the next input value and repeats the process until there is no more input, at which point the final input value is decoded without any more additions to the dictionary. In this way the decoder builds up a dictionary which is identical to that used by the encoder, and uses it to decode subsequent input values. Thus the full dictionary does not need be sent with the encoded data; just the initial dictionary containing the single-character strings is sufficient.

2.4 PSEUDO CODE:

COMPRESSION

```
class compress
{
    public static void compr(String content)
    {
        try
        {
            int i=0,hv=0;
            String str;
            char chr;
            Hashtable h=new Hashtable();
            BufferedWriter x=new BufferedWriter(new FileWriter("dict.txt"));
            while(i < content.length())
              {
                str=Character.toString(content.charAt(i));
                if(h.containsKey(str)==false)
                  {
                    if(str.equals("\t") )
                      {
                        if(h.containsKey(str))
                          { }
                         else
                        {
                            h.put(str,Integer.toString(hv));
                            ++hv;
                            x.write(hv-1+" "+"\\t"+"\n");
                        }
                    }
                    else if(str.equals(" ") )
                      {
                        if(h.containsKey(" "))
                    { }
                    else
                    {
                        h.put(" ",Integer.toString(hv));
                        ++hv;
                        x.write(hv-1+" "+"\\s"+"\n");
                    }
                }
                else if(str.equals("\n") )
                {
                    if(h.containsKey(str))
                    {}
                    else
                    {
                        h.put(str,Integer.toString(hv));
                        ++hv;
                        x.write(Integer.toString(hv-1)+"
"+"\\n"+"\n");
                    }
                }
                else
                {
                    h.put(str,Integer.toString(hv));
                    ++hv;
                    x.write(h.get(str)+" "+str+"\n");
                }
            }
            ++i;
        }
        x.close();
        i=0;
        str="";
        BufferedWriter y=new BufferedWriter(new FileWriter("code.txt"));
        while( i < content.length() )
          {
            chr=content.charAt(i);
            if(h.containsKey(str+chr))
            {
                str=str+chr;
            }
            else
            {
                y.write((String)h.get(str)+" ");
                h.put(str+chr,Integer.toString(hv));
                ++hv;
                str=Character.toString(chr);
            }
            ++i;
        }
        //y.write((String)h.get(str));
        y.close();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }

    }

}
```

```
class lzwcompress
{
            public static void xyz(String v[] )
             {
             //NewJframe j1 =new NewJFrame();
             //j1.setvisible(TRUE);
                         try
                         {
                                      BufferedReader reader = new BufferedReader(new
                                      FileReader(v[0]));
                                     String temp,content="";
                                     while( ( temp = reader.readLine() ) != null)
                                     {
                                             content = content + temp + "\n" ;
                                     }
                                     reader.close();
                                     compress.compr(content);

                         }
                         catch(Exception e)
                         {
                                    System.out.println("\n"+e);
                         }
             }
}
```

USER INTERFACE LZW COMPRESSION AND DECOMPRESSION

/*MOUSE CLICK FUNCTION*/

```
private void jTextField1MouseClicked(java.awt.event.MouseEvent evt) {
 JFileChooser j1=new JFileChooser();
    int retval=j1.showOpenDialog(jTabbedPane1);
    if(retval==j1.APPROVE_OPTION)
    {
        f1=j1.getSelectedFile();
       jTextField1.setText(f1.getPath());       // TODO add your handling code here:
    }
```

/*BUTTON CLICK FUNCTION*/

```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
        int i;
```

```
          jProgressBar2.setValue(0);
          for(i=0;i<1000000;i++)
          { jProgressBar2.setValue(i/10000);

          }
          File file3 = new File(jTextField1.getText());
                         long fileSize3 = file3.length();
                         jTextField3.setText("     " + (float)fileSize3/1024);

                        File file1 = new
     File("C:/Users/anirudh/Downloads/LZW/LZW/dict.txt");
                         long fileSize1 = file1.length();

                           File file2 = new
     File("C:/Users/anirudh/Downloads/LZW/LZW/code.txt");
                         long fileSize2 = file2.length();
                         jTextField4.setText("     " + (float)(fileSize1+fileSize2)/1024);

          try
             {

                   BufferedReader reader = new BufferedReader(new FileReader(f1));
                   String temp,content="";
                   while( ( temp = reader.readLine() ) != null)
                     {
                         content = content + temp + "\n" ;
                     }
                   reader.close();
                   compress.compr(content);

             }
           catch(Exception e)
             {
                   System.out.println("\n"+e);
```

```
            }       // TODO add your handling code here:
        try {
Process p=Runtime.getRuntime().exec("write.exe
C:\\Users\\anirudh\\Downloads\\LZW\\LZW\\code.txt" ); //enter file path
Process p1=Runtime.getRuntime().exec("write.exe
C:\\Users\\anirudh\\Downloads\\LZW\\LZW\\dict.txt" );
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

    }
/*Initializing Buttons, Labels for the front end */
File f1;
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JProgressBar jProgressBar2;
    private javax.swing.JTabbedPane jTabbedPane1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextField jTextField4;
```

DE COMPRESSION

```
class decompress
{
```

```
public static void decompr()
{
        try
           {
                BufferedReader reader= new BufferedReader(new FileReader("dict.txt"));
                String temp,content="";
                Hashtable hl=new Hashtable(),h=new Hashtable();
                int hv=0,hav=0;
                while( ( temp = reader.readLine() ) != null)
                   {
                        String content1[ ] = temp.split(" ");

                        if(content1[1].equals("\\s"))
                           hl.put(content1[0]," ");
                        else if(content1[1].equals("\\n"))
                           hl.put(content1[0],"\n");
                        else if(content1[1].equals("\\t"))
                           hl.put(content1[0],"\t");
                        else
                           hl.put(content1[0],content1[1]);
                        ++hv;
                   }
                reader.close();
                reader=new BufferedReader(new FileReader("code.txt"));
                String X="",Y="";

                while( ( temp = reader.readLine() ) != null)
                   {
                        X=X+temp;
                   }
               reader.close();
                String CODE[]=X.split(" ");
                int i=0,j=0;
                String Xn="",ch="",old="",digit="",value="";

                reader=new BufferedReader(new FileReader("hashdumpcompress.txt"));
                temp="";
                X="";
                i=0;j=0;
                while( ( temp = reader.readLine() ) != null)
                   {
                        X=X+temp;

                   }
                char e=13;
                while(i<X.length())
                   {
                        while(Character.isDigit(X.charAt(i)))
                           {
                                digit=digit+X.charAt(i);
                                ++i;
```

```
                            }
                        ++i;
                        while(X.charAt(i)!='$')
                            {
                                if(X.charAt(i)=='\\')
                                {
                                        if(X.charAt(i+1)=='n')
                                        {
                                                value=value+'\n';
                                                ++i;
                                        }
                                        else
                                                value=value+'\\';
                                }
                                else if(X.charAt(i)=='\\')
                                {
                                        if(X.charAt(i+1)=='t')
                                        {
                                                value=value+'\t';
                                                ++i;
                                        }
                                        else
                                                value=value+'\\';
                                }
                                else
                                        value=value+X.charAt(i);
                                ++i;
                            }
                        ++i;
                        h.put(digit,value);
                        digit="";
                        value=""; }i=0;

                BufferedReader name=new BufferedReader(new
FileReader("namefile.txt"));
                String N="decompress"+name.readLine();
                FileOutputStream de=new FileOutputStream(N);
                while(i<CODE.length)
                {
                        Xn=(String)h.get((String)CODE[i]);
                        new PrintStream(de).print(Xn);
                         ++i;
                }
                de.close();
            }
   catch(Exception e)
        {
                 e.printStackTrace();
                System.out.println(e);
        }
    }
```

```
}
public class lzwdecompress
{
          public static void main(String v[])
            {
                    try
              {
                    decompress.decompr();
              }
          catch(Exception e)
              {
                    System.out.println("\n"+e);
              }
}
}
```

CHAPTER 3

TESTING AND RESULTS

3.1 UNIT TESTING

SL_NO	FEATURES	EXPECTED OUTPUT	ACTUAL OUTPUT	REMARKS
1	Reading a File	Reading a File	File Read	SUCCESS
2	Writing to a File	Writing a File	File Writing	SUCCESS
3	Compressing a File	Compressing	File Compressed	SUCCESS
4	Decompressing a File	Decompressing	File Decompressed	SUCCESS
5	Connectivity of Front End to Back End	To be able to Connect	To be able to Connect	SUCCESS
6	File Size Retrieval in the Front End	Correct File Size	Correct File Size	SUCCESS

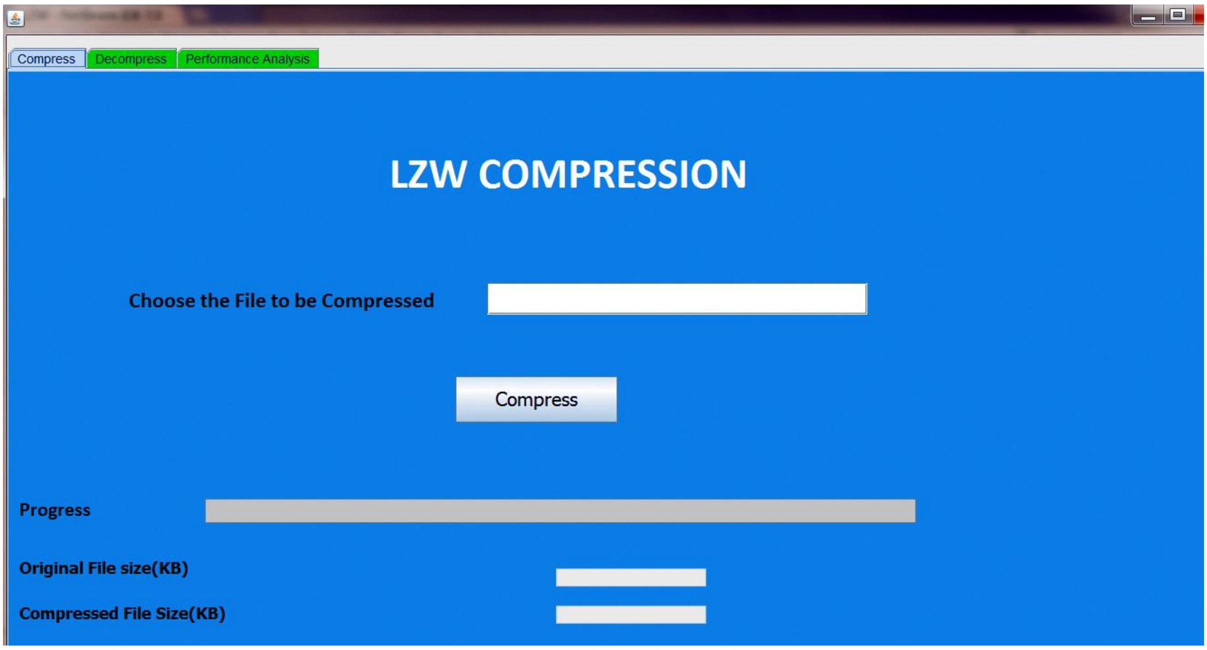
After integrating all the components the system is found to be working fine, thus it has passed Integration (system) testing.

3.2 SCREEN SHOTS

LZW COMPRESSION

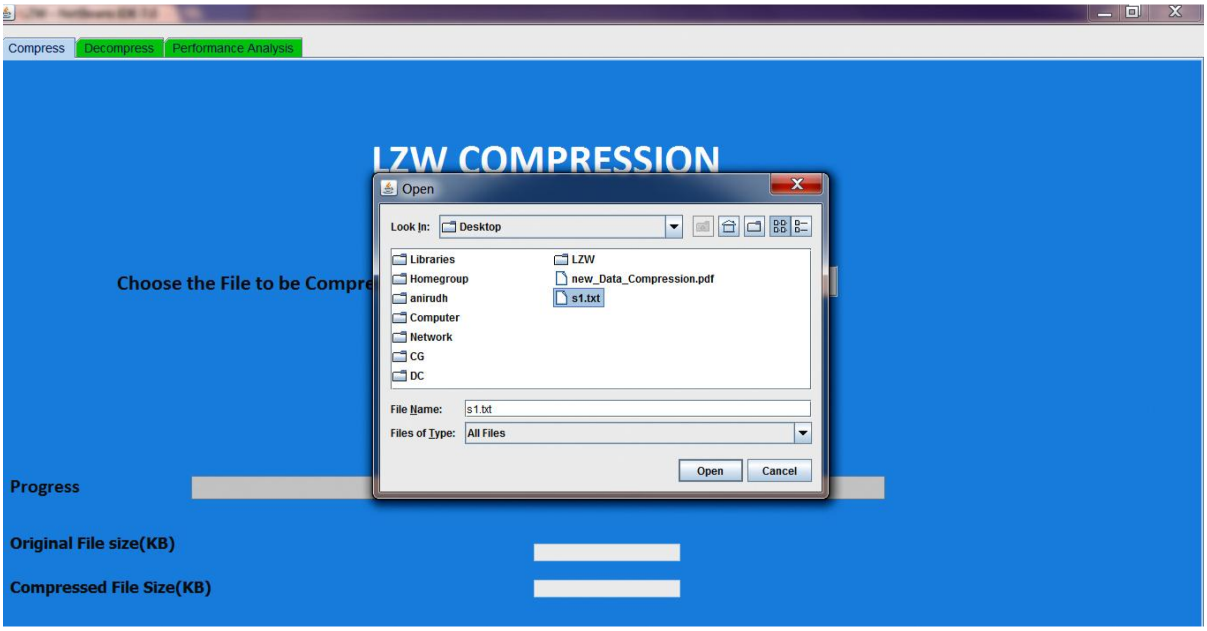
Step 1:

The GUI for LZW Compression is shown above. The user has to choose the file to be compressed. The text field shows the Original file size and compressed size.



Step 2:

S1.txt is a file that has to be compressed is chosen.



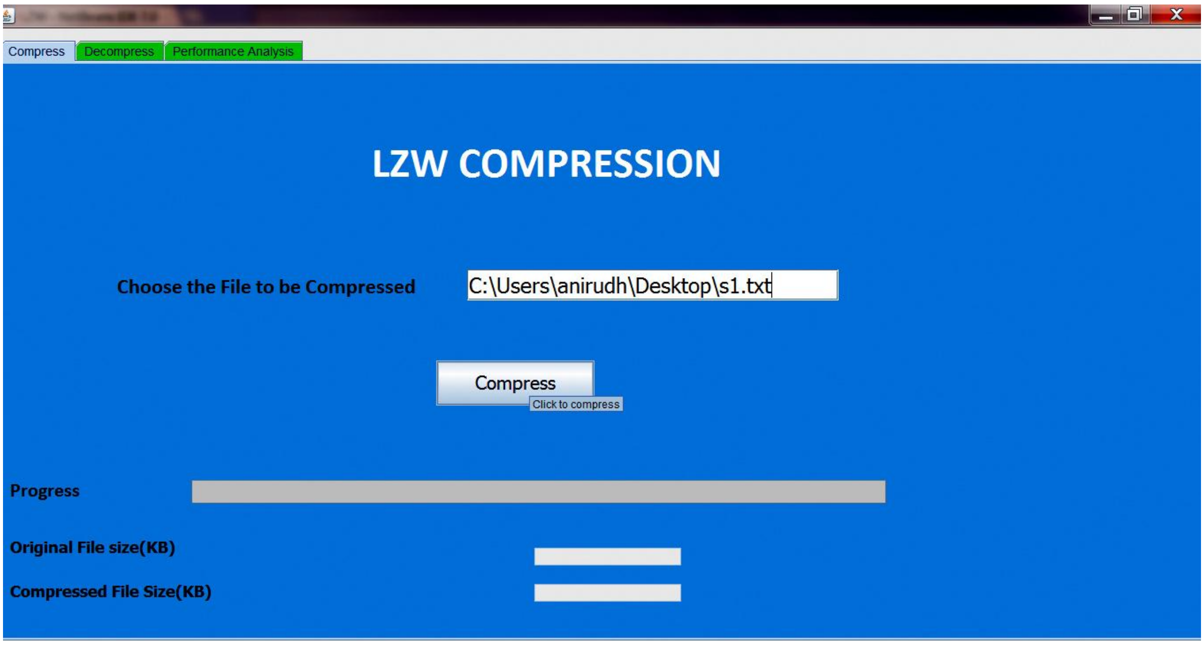
Look In: Desktop

File Name: s1.txt

Files of Type: All Files

Step 3:

After Choosing the File, Compress Button is clicked.



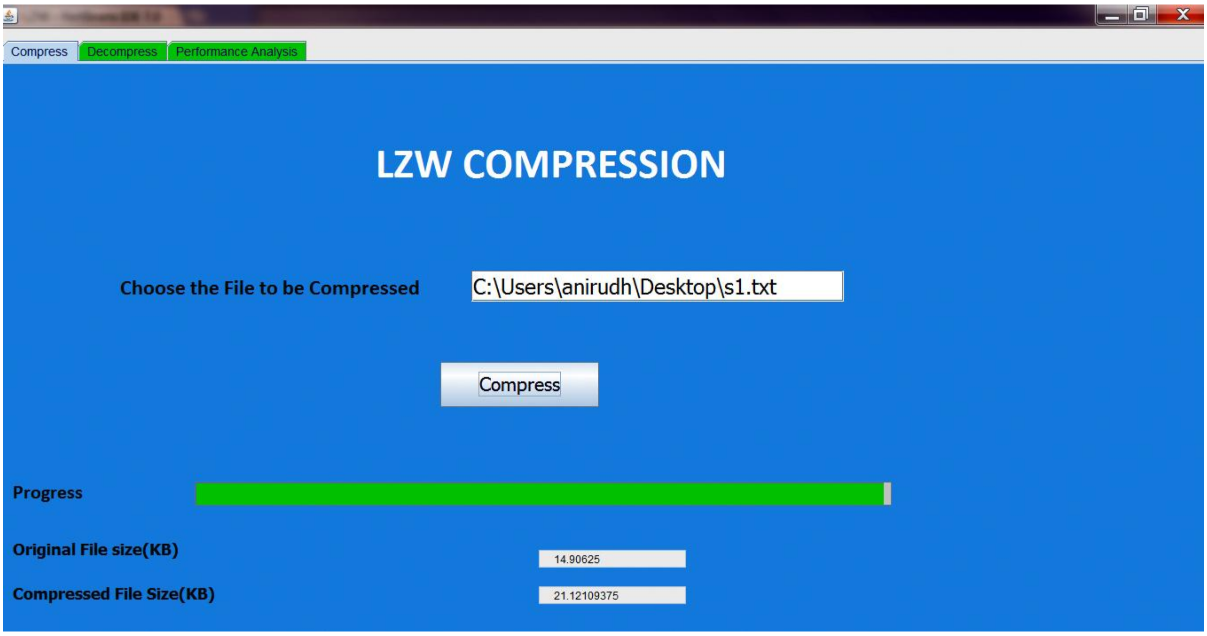
Choose the File to be Compressed C:\Users\anirudh\Desktop\s1.txt

Original File size(KB)

Compressed File Size(KB)

Step 4:

Figure shows the Progressive Bar and the Original File Size and Compressed File Size.



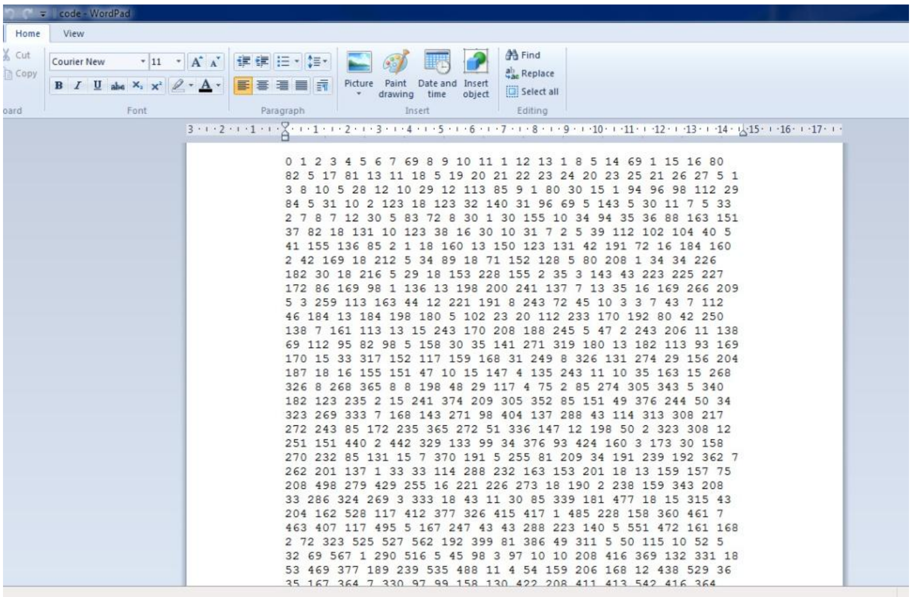
Choose the File to be Compressed C:\Users\anirudh\Desktop\s1.txt

Original File size(KB) 14.90625

Compressed File Size(KB) 21.12109375

Step 5:

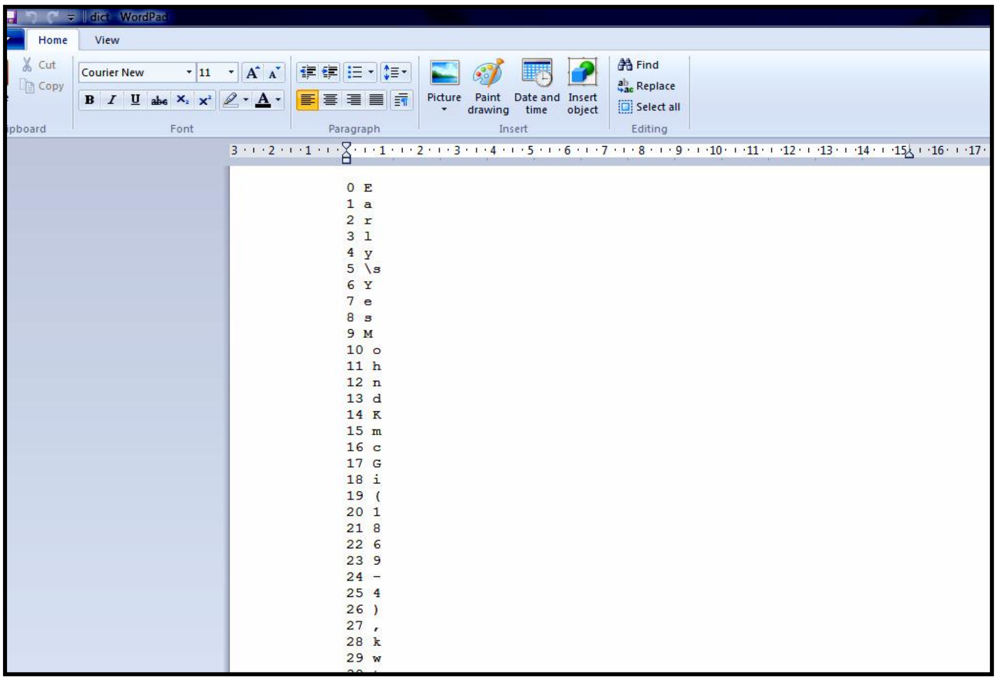
Figure showing code.txt from s1.txt opened form WordPad



```
0 1 2 3 4 5 6 7 69 8 9 10 11 1 12 13 1 8 5 14 69 1 15 16 80
82 5 17 81 13 11 18 5 19 20 21 22 23 24 20 23 25 21 26 27 5 1
3 8 10 5 28 12 10 29 12 113 85 9 1 80 30 15 1 94 96 98 112 29
84 5 31 10 2 123 18 123 32 140 31 96 69 5 143 5 30 11 7 5 33
2 7 8 7 12 30 5 83 72 8 30 1 30 155 10 34 94 35 36 88 163 151
37 82 18 131 10 123 38 16 30 10 31 7 2 5 39 112 102 104 40 5
41 155 136 85 2 1 18 160 13 150 123 131 42 191 72 16 184 160
2 42 169 18 212 5 34 89 18 71 152 128 5 80 208 1 34 34 226
182 30 18 216 5 29 18 153 228 155 2 35 3 143 43 223 225 227
172 86 169 98 1 136 13 198 200 241 137 7 13 35 16 169 266 209
5 3 259 113 163 44 12 221 191 8 243 72 45 10 3 3 7 43 7 112
46 184 13 184 198 180 5 102 23 20 112 233 170 192 80 42 250
138 7 161 113 13 15 243 170 208 188 245 5 47 2 243 206 11 138
69 112 95 82 98 5 158 30 35 141 271 319 180 13 182 113 93 169
170 15 33 317 152 117 159 168 31 249 8 326 131 274 29 156 204
187 18 16 155 151 47 10 15 147 4 135 243 11 10 35 163 15 268
326 8 268 365 8 8 198 48 29 117 4 75 2 85 274 305 343 5 340
182 123 235 2 15 241 374 209 305 352 85 151 49 376 244 50 34
323 269 333 7 168 143 271 98 404 137 288 43 114 313 308 217
272 243 85 172 235 365 272 51 336 147 12 198 50 2 323 308 12
251 151 440 2 442 329 133 99 34 376 93 424 160 3 173 30 158
270 232 85 131 15 7 370 191 5 255 81 209 34 191 239 192 362 7
262 201 137 1 33 33 114 288 232 163 153 201 18 13 159 157 75
208 498 279 429 255 16 221 226 273 18 190 2 238 159 343 208
33 286 324 269 3 333 18 43 11 30 85 339 181 477 18 15 315 43
204 162 528 117 412 377 326 415 417 1 485 228 158 360 461 7
463 407 117 495 5 167 247 43 43 288 223 140 5 551 472 161 168
2 72 323 525 527 562 192 399 81 386 49 311 5 50 115 10 52 5
32 69 567 1 290 516 5 45 98 3 97 10 10 208 416 369 132 331 18
53 469 377 189 239 535 488 11 4 54 159 206 168 12 438 529 36
35 167 364 7 330 97 99 158 130 422 208 411 413 542 416 364
```

Step 6:

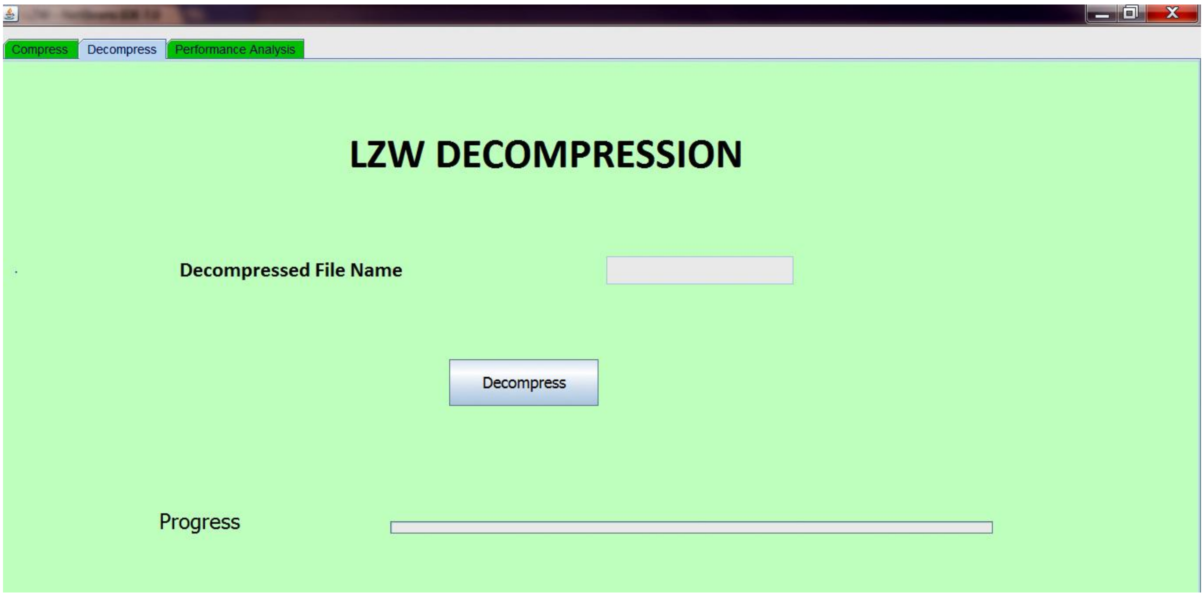
Figure shows the Dictionary of compressed File “s1.txt”



```
0 E
1 a
2 r
3 l
4 y
5 \s
6 Y
7 e
8 s
9 M
10 o
11 h
12 n
13 d
14 K
15 m
16 c
17 G
18 i
19 (
20 1
21 8
22 6
23 9
24 -
25 4
26 )
27 ,
28 k
29 w
```

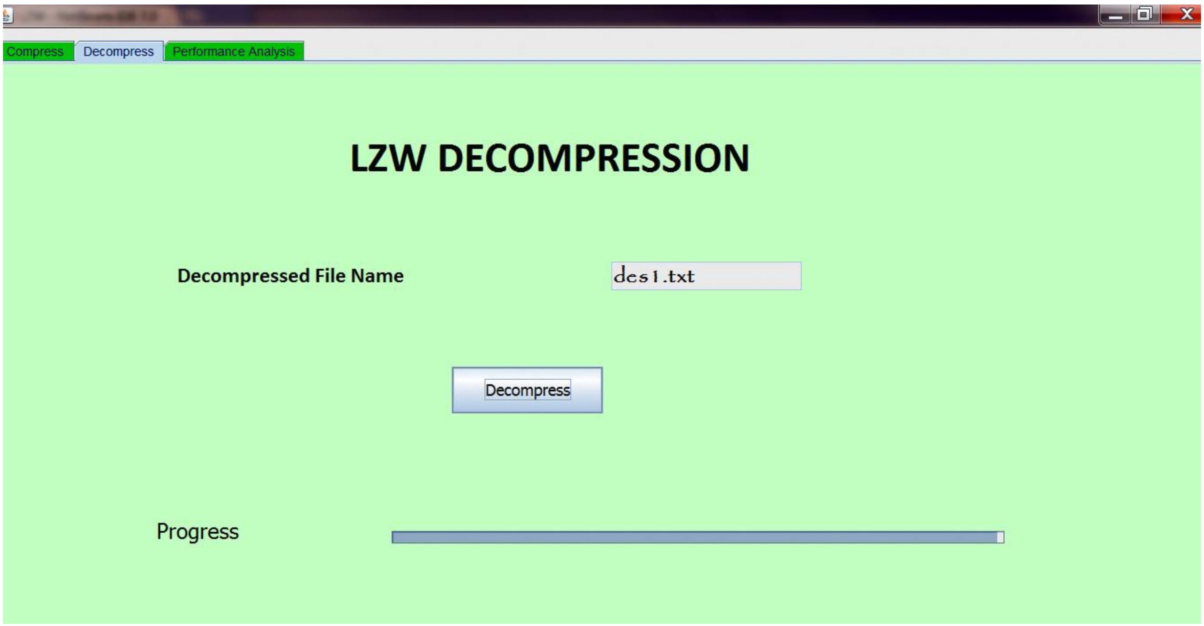
Step 7:

The fig. shows the tab for LZW Decompression



Step 8:

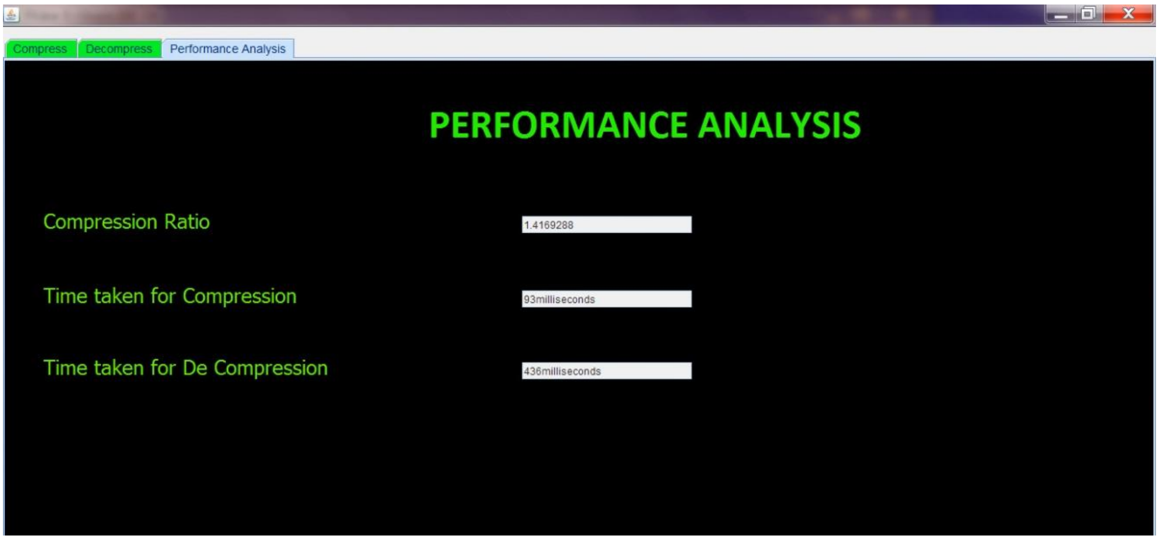
The Decompress button is clicked and the final text file is generated



Decompressed File Name des1.txt

Step 9:

The Performance Analysis Tab shows the Compression Ratio, Time taken for Compression and De Compression



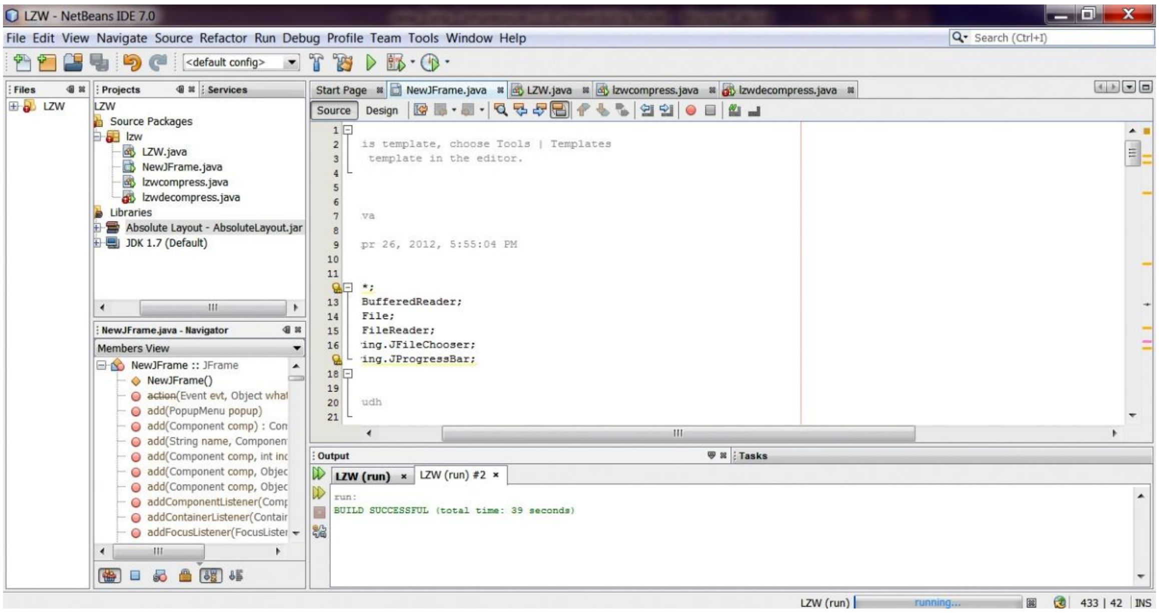
PERFORMANCE ANALYSIS

Compression Ratio: 1.4169288

Time taken for Compression: 93milliseconds

Time taken for De Compression: 436milliseconds

NETBEANS IDE PLATFORM



```
is template, choose Tools | Templates
 template in the editor.



va

pr 26, 2012, 5:55:04 PM


*;
BufferedReader;
File;
FileReader;
ing.JFileChooser;
ing.JProgressBar;


udh
```

```
run:
BUILD SUCCESSFUL (total time: 39 seconds)
```

CHAPTER 4

CONCLUSION AND FUTURE WORK

4.1 Conclusion

The project "LZW Compression" was a great learning Experience in terms of implementing and understanding the algorithm. The results obtained during the execution of the code just mimics the LZW algorithm we studied.

Results included generation of

- Dictionary (text file)
- Code File (Encoded File compressed)

The next part of our project is the User Interface creation using NET BEANS. Java code developed was integrated with interactive user interface and tested.

The compressed file was found to be 50% as less of original file which was one of the best expected outputs. The code built soon after integrating the front end was rebuilt to obtain a ".jar" file, and “.exe” (Executable) file was finally obtained.

Thus a complete project concluded in the output of complete package "LZW COMPRESSOR version 1.0” which remains absolutely free and a windows executable file. The obtained Windows Executable file can be uploaded to any Blogs or websites and users can make use of compression with no cost. A Comparison Graph is drawn by comparing file size of the Original File size and the compressed file size.

4.2 FUTURE WORK

Having this back-end, in future we can implement a front-end of the same by using PHP or any other Front-end designing tool and can be uploaded in any blogs or websites. Efficiency of Text Compression for Rich Text Format, .doc and other types of files can be improved.

CHAPTER 5

REFERENCES

- Introduction to Data Compression 3rd Edition, Sayood
- http://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch
- http://www.cs.cf.ac.uk/Dave/Multimedia/node214.html
- http://netbeans.org/kb/support.html
- http://wiki.netbeans.org/HelpGuidelines
- http://docs.oracle.com/javase/tutorial/uiswing
- http://netbeans.org/kb/docs/java/gui-functionality.html
- http://netbeans.org/kb/docs/java/quickstart-gui.html