

CHAPTER V

IMPLEMENTATION AND TESTING

5.1 Implementation

In this section, I will show the source code by per step. The first step, of course, we need to pick the file first. Here is the code for the file selector.

```
private void jMenuItemActionPerformed(java.awt.event.ActionEvent evt) { //GEN-FIRST:event_jMenuItemActionPerformed
    // TODO add your handling code here:
    int returnVal = fileChooser.showOpenDialog(this);
    if (returnVal == JFileChooser.APPROVE_OPTION) {
        File file = fileChooser.getSelectedFile();
        try {
            before.read( new FileReader( file.getAbsolutePath() ), null );
            before2.read( new FileReader( file.getAbsolutePath() ), null );
            double bytes = file.length();
                String toBytes = String.valueOf(bytes);
                cFileSize.setText(toBytes);
                dFileSize.setText(toBytes);
            catch (IOException ex) {
                System.out.println("problem accessing file"+file.getAbsolutePath());
            }
        } else {
            System.out.println("File access cancelled by user.");
        }
    } //GEN-LAST:event_jMenuItemActionPerformed
```

Then, the input file is processed in the ReadAFile class, which soon to be compressed by the compressor based on user's choice.

```
import java.util.Scanner;
import java.io.*;

class ReadAFile {
    int choice;
    int choice2;
    public String getTheFileName() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("FILE: ");
        String fileName = scanner.nextLine();
        return fileName;
    }

    public void scanFile(String fileName) throws IOException {
        Scanner scanner = new Scanner(System.in);
        GetFileSize fileSize = new GetFileSize();
        try {
            String tulisan = "";
            File file = new File(fileName);
            if(file.exists()) {
                Scanner inFile = new Scanner(file);
                String line;
                int lineNum = 0;
                while(inFile.hasNext()) {
                    line = inFile.nextLine();
                    //System.out.println(line);
                    tulisan = tulisan.concat(line+"\n");
                }

                String compressionMethod;
                System.out.println("Select compression method (type the number)");
                System.out.println("1. LZSS");
                System.out.println("2. Shannon-Fano");
                System.out.print("Number: ");
                choice = scanner.nextInt();
            }
        }
    }
}
```

```

switch(choice) {
    case 1: compressionMethod = "LZSS";
            break;
    case 2: compressionMethod = "Shannon-Fano";
            break;
}
//System.out.println(tullisan);
if(choice == 1) {
    String behavior;
    System.out.println("behavior pls");
    System.out.println("1. kompres");
    System.out.println("2. dekompres");
    System.out.print("Number: ");
    choice2 = scanner.nextInt();
    switch(choice2) {
        case 1: behavior = "compress";
                break;
        case 2: compressionMethod = "decompress";
                break;
    }
    if(choice2 == 1) {
        LZ77Compress LZSSCom = new LZ77Compress();
        LZSSCom.compress(tullisan);
        System.out.println("Uncompressed file size: "+filesize.FileSize(fileName)+" bytes");
        System.out.println("Compressed file size: "+filesize.FileSize("LZ77Output.txt"+" bytes");
        inFile.close();
    } else if(choice2 == 2) {
        LZSSDecompress LZSSDec = new LZSSDecompress();
        LZSSDec.decompress(tullisan);
        System.out.println("Compressed file size: "+filesize.FileSize(fileName)+" bytes");
        System.out.println("Decompressed file size: "+filesize.FileSize("LZSSOriginal.txt"+" bytes");
    }
} else {
    ShannonFanoCompress ShaFa = new ShannonFanoCompress();
    ShaFa.compress(tullisan);
    System.out.println("Uncompressed file size: "+filesize.FileSize(fileName)+" bytes");
    System.out.println("Compressed file size: "+filesize.FileSize("SFOutput.txt"+" bytes");
    inFile.close();
}
//System.out.println("work yuo fagget");
} else System.out.println("no file yuo fagget");
} catch(Exception er) {
    System.out.println(er);
}
}
}

```

```

public void compress(String file) throws IOException {
    StringBuilder sb = new StringBuilder();
    charCnt = 0;
    String output = "";
    while(charCnt < file.length()){
        if(charCnt - searchFinalLength >= 0) {
            searchInitLength = charCnt - searchFinalLength;
        } else searchInitLength = 0;

        if(charCnt+lookAheadInitLength < file.length()) {
            lookAheadFinalLength = charCnt+lookAheadInitLength;
        } else lookAheadFinalLength = file.length();

        if(charCnt == 0){
            searchWindow = "";
        }else{
            searchWindow = file.substring(searchInitLength,charCnt);
        }
        matchLen = 1;
        String searchTarget = file.substring(charCnt,charCnt + matchLen);
        if(searchWindow.indexOf(searchTarget) != -1){
            matchLen++;
            while(matchLen <= lookAheadInitLength) {
                searchTarget = file.substring(charCnt,charCnt+matchLen);
                matchLoc = searchWindow.indexOf(searchTarget);
                if((matchLoc != -1) && ((charCnt + matchLen) < file.length())) {
                    matchLen++;
                } else break;
            }

            matchLen--;
            matchLoc = searchWindow.indexOf(file.substring(charCnt,charCnt+matchLen));
            charCnt += matchLen;
        }
    }
}

```

And here is where the compression happens. If the user selected LZ77, then the LZ77Compress class is called. Else, if the user selected ShannonFano, then the Shannon Fano Compress class is called.

```

byte [] bytes;
bytes=output.getBytes();
for (int i=0;i<bytes.length;++i) {
    if(bytes[i]==48) bytes[i]=0; // 48 ASCII 0
    if(bytes[i]==49) bytes[i]=1; // 49 ASCII 1
}
BitOutputStream theStream = new BitOutputStream(new BufferedOutputStream(new FileOutputStream(filepath)));
try {
for (int i = 0; i < bytes.length; i++)
{
    theStream.write(bytes[i]);
}
} finally {
    theStream.close();
}
}

```

The method compress accepts the contents of the file as a parameter. Then the search window and look-ahead window is initialized. Search window is always empty at coding position 0, because there is no string yet to be processed. Afterwards, the search window contains strings that has already been traversed, with maximum length as declared. This project uses 64 for its maximum value, therefore the search window can contain up to 64 characters. The look-ahead window then finds a match between its contents with the search window. If there is a match, then calculate the value of offset(how many indexes the decoder

```

if(codingPos < (searchFinalLength + matchLen)) {
    offset = charCnt - matchLoc - matchLen;
} else offset = searchFinalLength - matchLoc;

String nextChar = file.substring(charCnt,charCnt+1);

tokenisasi = new Tuple(offset,matchLen,nextChar);
encodedData.add(tokenisasi);
} else {
    String nextChar = file.substring(charCnt,charCnt+1);
    tokenisasi = new Tuple(0,0,nextChar);
    encodedData.add(tokenisasi);
}
charCnt++;

System.out.print(tokenisasi + " - ");
String tokenString = tokenisasi.toString();
sb.append(tokenString);
output = sb.toString();
if(tokenisasi.stringLen > 0) {
    int start = charCnt - 1 - tokenisasi.stringLen - tokenisasi.offset;
    int end = charCnt - 1 - tokenisasi.offset;
    System.out.println(file.substring(start,end) + tokenisasi.nextChar);
} else {
    System.out.println(tokenisasi.nextChar);
}
}

```

```
System.out.println(output);
```

should step back), length(the length of the match), and nextChar(next unencoded character). If there is no match, then set the value of offset, length, and nextChar as (0,0,(current char)) respectively. Next, those three values are sent to the tuple class to encapsulate them as one object that contains 3 values. Then, populate the ArrayList encodedData with that tuple object. Because this is a loop, the array list encodedData contains an array of tuple object. Finally, that array of tuples is sent to the BitOutputStream class to convert it into binary file form.

After this step, the output is generated with the name LZ77Output.txt. Next is the ShannonFanoCompress class.

```
public void compress(String file) throws IOException {
    int codingPos = 0;
    char highlight = file.charAt(codingPos);
    String output = "";
    HashMap<Character,Integer> map = new HashMap<Character,Integer>();
    HashMap<Character, String> codeMap = new HashMap<Character, String>();
    StringBuilder sb = new StringBuilder();
    for(int i = 0; i < file.length(); i++){
        char c = file.charAt(i);
        Integer value = map.get(c);
        if(value != null) {
            map.put(c, new Integer(value + 1));
        } else {
            map.put(c,1);
        }
    }

    Map<Character,Integer> sortedMap = sortByComparator(map);
    HashMap<Character,Integer> map2 = new HashMap<Character,Integer>(sortedMap);

    int mapsize = sortedMap.size();
    codeMap = sf.compress(map2);

    while(codingPos < file.length() - 1) {
        String test =(String)codeMap.get(highlight);

        sb.append(test);
        output = sb.toString();

    byte [] bytes;
    bytes=output.getBytes();
    System.out.println(bytes);
    for (int i=0;i<bytes.length;++i) {
        if(bytes[i]==48) bytes[i]=0; // 48 ASCII 0
        if(bytes[i]==49) bytes[i]=1; // 49 ASCII 1
    }
    BitOutputStream theStream = new BitOutputStream(new BufferedOutputStream(new FileOutputStream(filepath)));
    try {
    for (int i = 0; i < bytes.length; i++)
    {
        theStream.write(bytes[i]);
    } finally {
        theStream.close();
    }
}
```

What this class does is it accepts the input file's contents as a parameter, then generates a Hashmap that contains the input file's characters and its frequencies. Then, that hashmap is sorted into descending order, with higher frequencies to lower frequencies order. After being sorted, the hashmap is sent to ShannonFano class to generate the code for each letter.

```
public HashMap compress(HashMap freq) {
    HashMap<Character, String> result = new HashMap<Character, String>();
    List<Character> charList = new ArrayList<Character>();

    Iterator entries = freq.entrySet().iterator();
    while( entries.hasNext() ) {
        Map.Entry<Character, Integer> entry = (Map.Entry)entries.next();
        charList.add(entry.getKey());
    }
    addBit(result, charList, true);
    return result;
}

private void addBit(HashMap<Character, String> result, List<Character> charList, boolean up) {
    String bit = "";
    if( !result.isEmpty() ) {
        if(up) {
            bit = "0";
        } else bit = "1";
    }
    for( Character c : charList ) {
        String s = (result.get(c) == null) ? "" : result.get(c);
        result.put(c, s + bit);
    }
    if( charList.size() >= 2 ) {
        int separator = (int)Math.floor((float)charList.size()/2.0);

        List<Character> upList = charList.subList(0, separator);
        addBit(result, upList, true);
        List<Character> downList = charList.subList(separator, charList.size());
        addBit(result, downList, false);
    }
}
```

This class generates a charList that contains the list of characters found in the input map. The splitting into left and right groups and code appending is performed at the addBit method. The left group is given the boolean value "true" to give the code value of 0, likewise the right group is given the boolean value "false" to give the code value of 1. This method is called recursively, and will keep splitting and appending until only 1 character is left in the group. This class returns another hashmap that contains character and its Shannon-Fano code.

The Shannon-Fano compression also uses the BitOutputStream like the LZ77 above.

```

import java.io.IOException;
import java.io.OutputStream;

public final class BitOutputStream {

    private OutputStream output;
    private int currentByte;
    private int numBitsInCurrentByte;

    public BitOutputStream(OutputStream out) {
        if (out == null)
            throw new NullPointerException("Argument is null");
        output = out;
        currentByte = 0;
        numBitsInCurrentByte = 0;
    }

    public void write(int b) throws IOException {

        currentByte = currentByte << 1 | b;
        numBitsInCurrentByte++;
        if (numBitsInCurrentByte == 8) {
            output.write(currentByte);
            numBitsInCurrentByte = 0;
        }
    }

    public void close() throws IOException {
        while (numBitsInCurrentByte != 0)
            write(0);
        output.close();
    }
}

```

This class has an argument `OutputStream out` that contains the output file. This means the binary output will be written in this file. The `write` method accepts the byte sent from the caller's class (LZ77 and Shannon-Fano). A bitwise operation is performed here. The binary form of method `currentByte` is shifted by 1 to the left. For example, if the `currentByte` is 0 (binary value also 0), shifted by 1 bit to the left, that also produces 0, then another bitwise operator `OR`. This bitwise operator compares the `currentByte` with `b` (the byte sent from the compression class), and returns 1 if either of compared variables contains 1. In this case, 0 is compared to 65 (ASCII value of letter 'a'), so `currentByte` becomes 65. The next step is to check whether the number of bits of the current byte is 8, if yes, then write the said bit into the output file.

5.2 Testing

This part will attempt to test the performance of the two algorithms with a given test text file named test1.txt, test2.txt, and test3.txt respectively. Both has difference in file length and occurrence of symbols.

The string for test1.txt is as follows:

abracadabra

test2.txt:

I am sam. Sam. That Sam-I-am! I do not like lontong mas otong. Do you like green eggs and ham? I do not like them, Sam-I-am.

test3.txt:

As with just about every Mega Man game, you can select which order you want to tackle the Mavericks. The best recommended order is listed below, but take note that there are several variations on this. For instance, Optic Sunflower is a good choice, but if you're interested in a "speed run", your best bet is to eliminate Earthrock Trilobyte first (more on this later in the guide). As you complete stages, you'll gather "Metals", which are like P-Chips, Zenny, or whatever other monetary units you might consider them as. You use R&D Lab to buy items. Most items require a large "rare Metal" before you can forge them; you'll find these in the various stages.

The result of the test shall be represented in this table below. Note that the compression ratio here is gotten by dividing the compressed file size with the original file size, then multiplying it by 100 to get the percentage.

File name	File size	Compression output		Compressed file size & compression ratio	
		LZ77	Shannon Fano	LZ77	Shannon Fano
Test1.txt	12 bytes	0,0,a0,0,b0,0,r3,1,c5,1,d7,4,	000100110010001100001001100	4 bytes, 33,33%	4 bytes, 33,33%
Test2.txt	125 bytes	0,0,i0,0,0,0,a0,0,m3,1,s4,2,.8,1,S5,4,T0,0,h15,1,t10,4,-24,1,-24,2,!28,1,l30,1,d0,0,o33,1,n3,1	00010001110100101000011110110100101001011100111001001001010010111001110011001110010011100001110010010010010	40 bytes, 32%	74 bytes, 59,2%

	,t37,1,i0,0,i0,0,k0,0,e5,2,o 11,1,t3,2,g50,1,m51,1,s54 ,1,o10,4,.61,1,D32,2,y38, 1,u37,6,g0,0,r49,1,e60,1, 57,1,g53,1,s78,1,a76,1,d8 6,1,h66,1,m0,0,? 94,1,l95,1,d69,2,n83,2, 75,5,t67,1,e109,1,,114,1, S78,2,-80,1,-88,2,.0,0,	1010001010101001010 0010100011100010001110 10111100001111011011001 1100001111001001111100 0011000011110010110010 1101110011001011001101 0011110100010011011001 1111001110011001011001 1011011100111000011000 01111111011001110100111 1001001111100001100001 1101101110100110001100 1011000111011000110101 1011101100111010010110 0101100111011100100101 0011111001110001000111 0101111000011110110110 01110000111100100111110 000110000111110001110 0110010100100110011100 100100101001010100010 1010101001010010111			
Test3.tx t	662 bytes	0,0,A0,0,s0,0, 0,0,w0,0,i0,0,t0,0,h5,1,j0, 0,u9,1,t10,1,a0,0,b0,0,o7, 1,t16,1,e0,0,v2,1,r0,0,y22, 1,M7,1,g15,1, 5,1,a0,0,n31,1,g22,1,m18, 1,,37,1,y26,2, 0,0,c14,3,s30,1,l32,1,c42, 2,w50,1,i14,1,h58,1,o40,1 ,d44,2, 31,4,w50,2,t75,1,t76,1, 78,1,a55,1,k53,2, 56,1,h29,2,M91,1,v59,2,i4 8,2,s0,0,.98,1,T54,3,b100, 1,s100,2,r108,2,o0,0,m53, 1,e98,1,d111,1,d119,7,i97, 1, 116,1,i96,2,e89,2,b131,1,l 106,1,w0,0,,121,2,u123,2, t146,1,k138,2,n134,1,t148 ,2,t155,1,a114,3,h144,2,e 166,1,a112,3,s171,1,v166 ,2,a166,1, 125,1,a182,1,i148,2,i174,	00111111101000001011111 0101101110111010110000 10101110111100111010111 0110000101000101000111 1010111100111011000010 10011111110110011111100 11111100001001011110011 110101010001000001001 0111100010110011000010 1010101000101100101001 1100101000001011111111 0101111000000101001010 0010110011000010111010 1001111100010011110010 111011000010111101010 1110110100101010110000 1011010111001001101001 1111100000010111111101 011110000001011110100 0101100111110110000101 1101111010000010111011 1000101001010111111000 1001110000101110111010 1110011100001001011110	237 bytes, 35,8%	476 bytes, 71,9%

1,n190,2,o171,1,
170,2,i161,1,.195,1,F202,
1,r203,1,i161,2,t207,1,n0,
0,c211,1,.214,1,O0,0,p18
6,2,c221,1,S0,0,u195,1,f2
09,1,o0,0,w229,2, 203,2,
228,1,
0,0,g224,1,o0,0,d238,1,c2
26,1,o208,2,e219,2,b216,
1,t238,2,f254,1,y250,1,u0,
0,'255,1,e258,3,t242,2,e2
69,2,e243,2,i265,1,
258,2,"266,1,p274,1,e267
,2,r262,1,n237,1,.265,4,r2
82,2,e267,2,
245,2,t296,2,s306,1,t301,
1,
301,1,l313,1,m307,2,a311
,2,
0,0,E307,1,r323,1,h325,1,
o0,0,c0,0,k327,1,T318,1,i
295,1,o315,1,y297,3,f318,
1,r329,3,
(321,1,o345,1,e353,1,o33
1,1,
327,2,i351,2,l346,3,r367,2
,n322,3,e375,1,g0,0,u378,
1,d376,1,)0,0,.381,1,A346
,2,y384,1,u385,1,c392,1,
m0,0,p391,1,e391,3,s399,
1,a367,1,e401,1,.365,4,'3
98,1,l390,2,a399,3,r417,1,
"0,0,M389,2,a396,1,s373,
1,,431,1,w430,1,i416,1,h4
36,1,a395,1,e436,1,l389,1
,k429,2,P0,0,-
0,0,C404,2,p438,3,Z455,1
,n397,1,y453,2,o445,2,w4
53,1,a458,1,e0,0,v462,3,o
472,5,m438,1,n481,3,r45
3,1,
0,0,u461,1,i454,1,s490,1,
y468,1,u455,2,i0,0,g493,1
,t501,1,c469,2,s506,1,d48
9,3,t492,2,m513,1,a485,1,
.521,1,Y490,3,u501,1,e52

0010111101100111110010
110100101011111101000
1100000100111101010111
0011100001010001110011
1111010111011000010111
0010011110010110101100
1011001010011111001110
0110100111100110000010
1101011100100110100111
1110000001010110111010
0000101100010110111010
1110111001111001100000
1010001110011111000110
1011111000101000001010
00111111001110110000101
11011100010101111100111
0000101100111101011101
1100111000010111011101
0111000101110110000101
11011101011100111111001
0011100001010001011100
1001110000101110101001
11111101100111111001000
101100000001011101100
0101110010110100010111
01110110110101100111110
1000001011010110011000
01011101110101110110111
0100011000001001010110
1011100000010101101100
11111010111011100010110
0111001010011100101000
0010011001101101110111
0110100100000100111011
1100110011101001100011
01011111010011111100000
0101011011101000001010
0010000010101010110101
101010011000001010010
1010111101010110100101
0011100101000001010001
1111100111011000010101
1010100000010111111110
1011110000011011100100
1110000101011011001111
10111001111110010011111

5,1,R0,0,&0,0,D533,1,L52	1010111011100111100110
5,1,b501,2,o541,1,b535,1,	0000101011011001100001
y540,1,i548,1,e537,1,s51	0100010000010000011111
9,2,M550,1,s547,2,i509,4,	0101101101001111001111
554,1,e0,0,q547,1,i512,2,	0011000001011100111100
563,1,	1100110000110010100000
0,0,i569,1,r0,0,g565,2,"53	1011111110101111001110
6,1,a535,3,M583,1,t580,1,	0000010100011100111111
i540,1,	0101110110000101000111
587,1,e0,0,f590,1,r564,2,y	0011111101100001010110
600,1,u603,1,c576,1,n607	1110100000101110111101
.1,f560,2,g592,2,t0,0,h610	0000010100111110001011
,2,;577,4,'612,1,i583,2,i59	0110010101101100111000
0,1,d587,4,s632,2,i610,2,t	1011101110011100001001
607,2,	0011100010111001110111
0,0,v641,1,r606,1,o636,1,	0101111100110101001010
s650,1,s632,1,a639,2,s0,	1111000010011110111001
0,.0,0,	0110110001101010001111
	1111110111001110000101
	0100101101110011101011
	1011000010000111110010
	1101011100100111000010
	1101011001100001011101
	1101011101101110100000
	1011000100010111011100
	1111110000001010110110
	0110000101110111010111
	0011100001010101011110
	0101101001101001110010
	0001100000100011111110
	1000001011111110101111
	0000001010010110101100
	1011011011000100111111
	0111001110000101110101
	1101110001010101010011
	1111010001010000010111
	11111010111100000110110
	001100000001010101010
	00101110111010111001111
	1100000010000011010111
	1001111110111000101100
	0111010000011001010000
	0101111101010111011010
	0101010110000101000101
	1100100111000010110001
	0110101111100111000010

0110100010110100010101
1101101101101110100010
1000001010000100111110
0111100111111100101000
0010110101110000001011
1110101011100010111011
1001111110110011111100
0000101101011101110101
1100111111000000101100
10110101100111001111110
11100010111001111110000
10111100110011101101110
1111101000001011111110
1011110000001011001010
1101010101010111110110
0001010010110101100111
1101010110100110100111
1110000001011101110101
1100111110010000010100
0101110100011000001001
11111101011110000001011
1100111010100111000010
0110110001001001000001
0010110100010100011000
0101110111101000001010
00111111001111110000101
01101110111001111100101
1101000110000010010111
1101011101011101100001
01011011101110011111001
0111010000010111001001
1111011111100101101110
0100111000010100010000
0101100010001011100101
0101001110000100000111
1100100010111001001110
00010010111100111111011
1000101100000001100001
0100011100111101001101
01110010011100001011111
1110101111000000101001
0100010110011000010101
0011010111001010101001
1100001011101110101110
0111110010001110000010
1111111010111100000110

			1100011000000010101001 0110110011100110000010 11101110101110011111101 0100111000010101101100 1100001011101110101110 0111000010111101100010 11100101101101011110011 1010000010111010111011 1000101010101001111110 1000110		
--	--	--	---	--	--

Table 5.2 Test result table

5.3 Main Interface Window

The main interface of the compression program. First, the program prompts the user to select an input file (more precisely, a .txt file). Then, user must click the compress button, there are LZ77 Compress or Shannon Fano compress buttons. After the button is pressed, the compression result is displayed on the right text area.

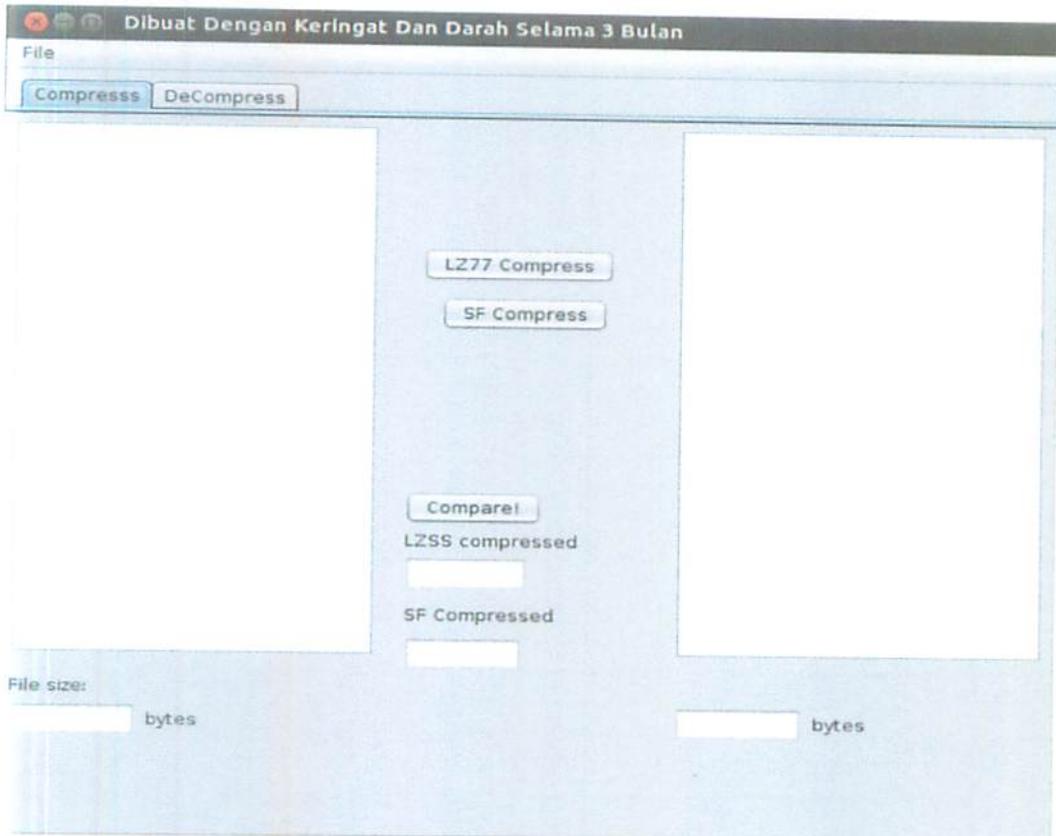


Figure 5.3.1 The main interface window

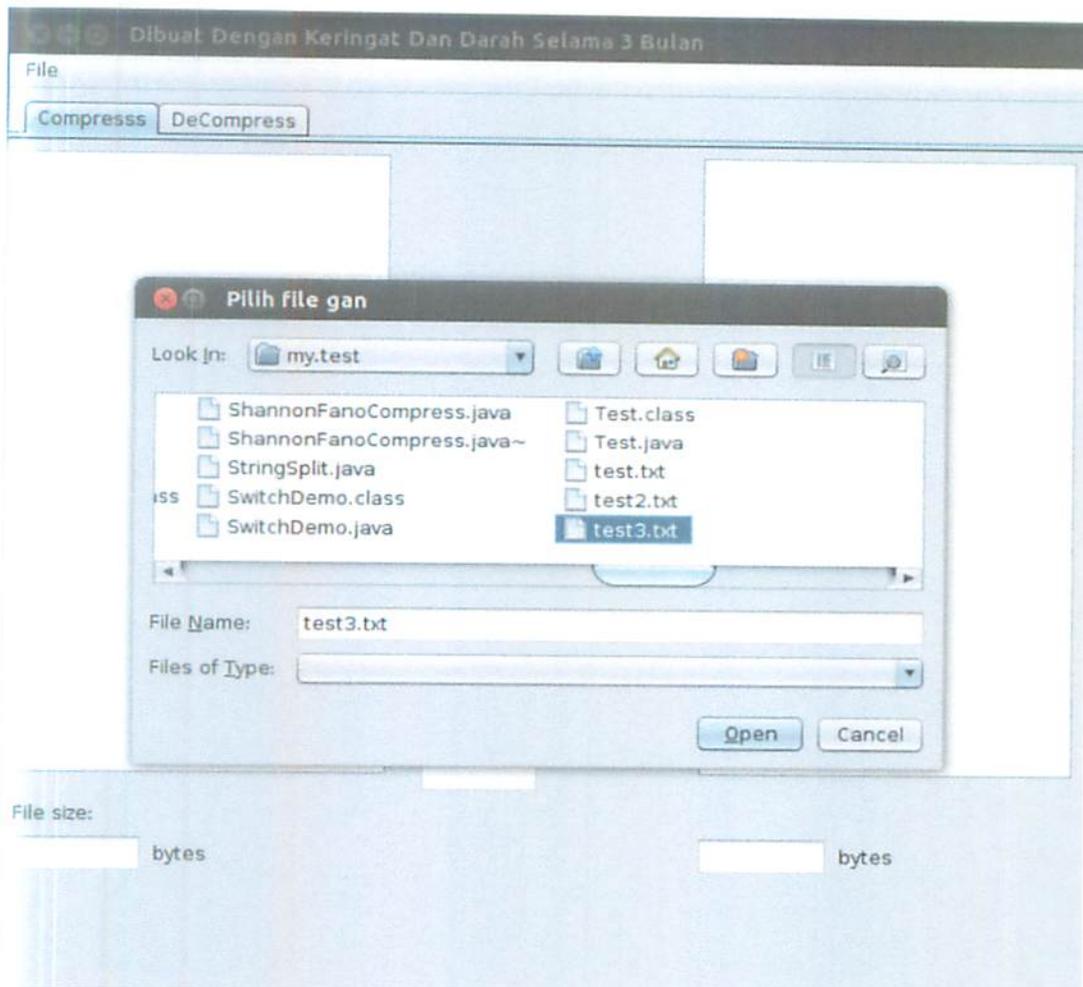


Figure 5.3.2 The file selector

At the demonstration here, we are going to use the test3.txt file, as it is the longest text test file available. Below is the contents of the test3.txt shown by the program:

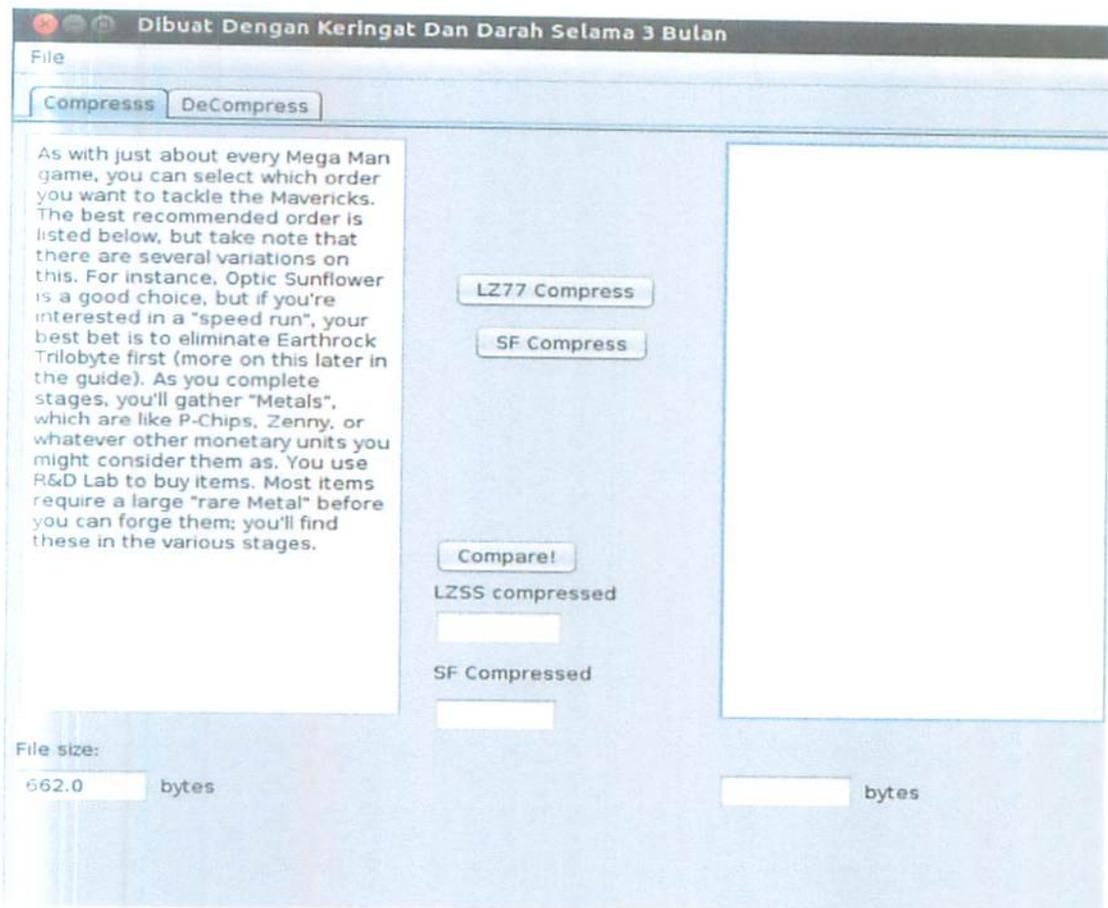


Figure 5.3.3 Showing the contents of test3.txt

Next, we should choose one of two compression algorithms available, the LZ77 or the Shannon – Fano. If the LZ77 Compress button is pressed, the program will process the given text file with LZ77 algorithm, then the text field on the right will show the compressed form of the input file. Else if the SF Compress button is pressed, the program will process the text file with Shannon – Fano algorithm, and the text box's contents is changed into the Shannon – Fano compressed form.

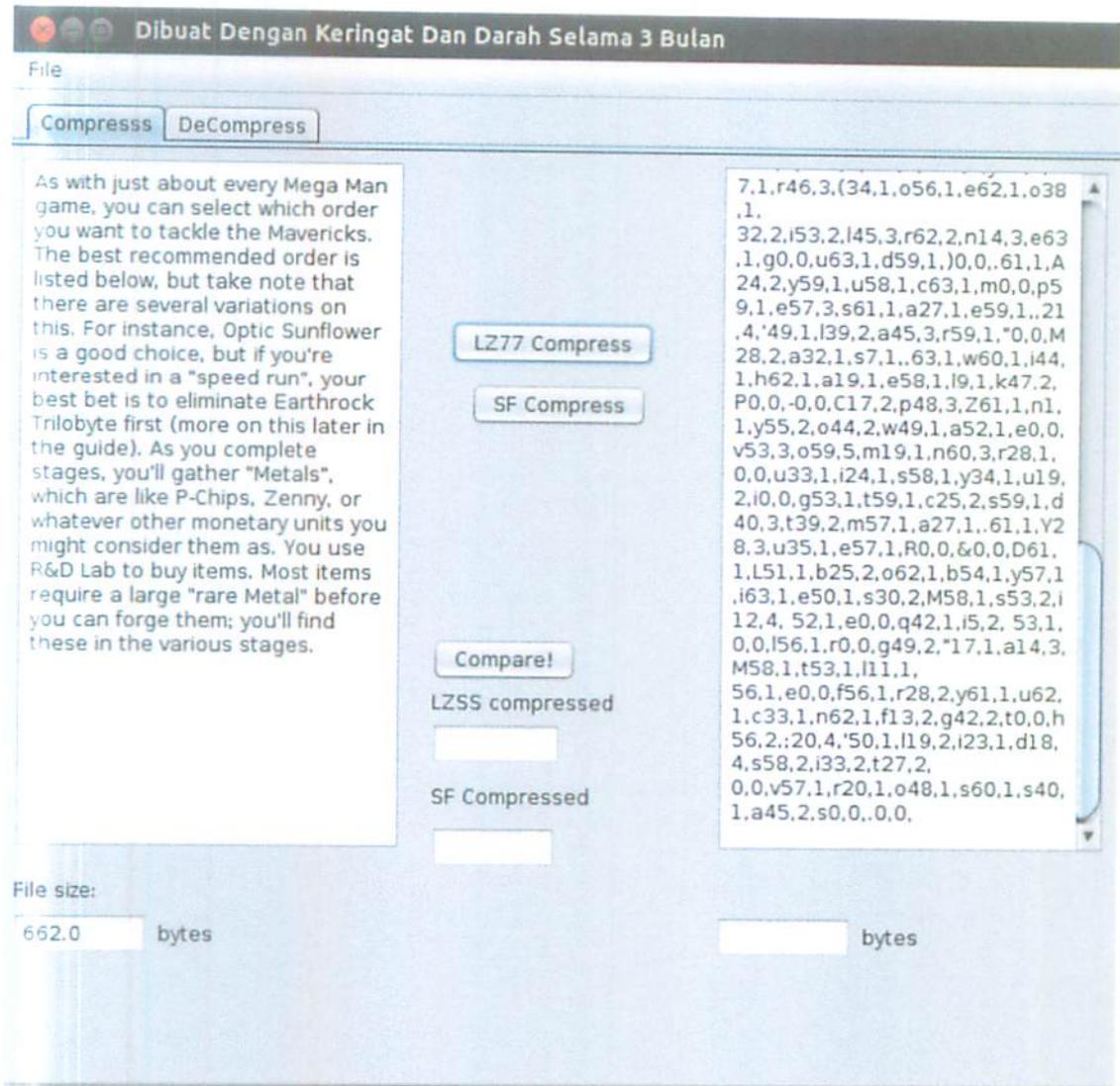


Figure 5.3.4 LZ77 compressed form

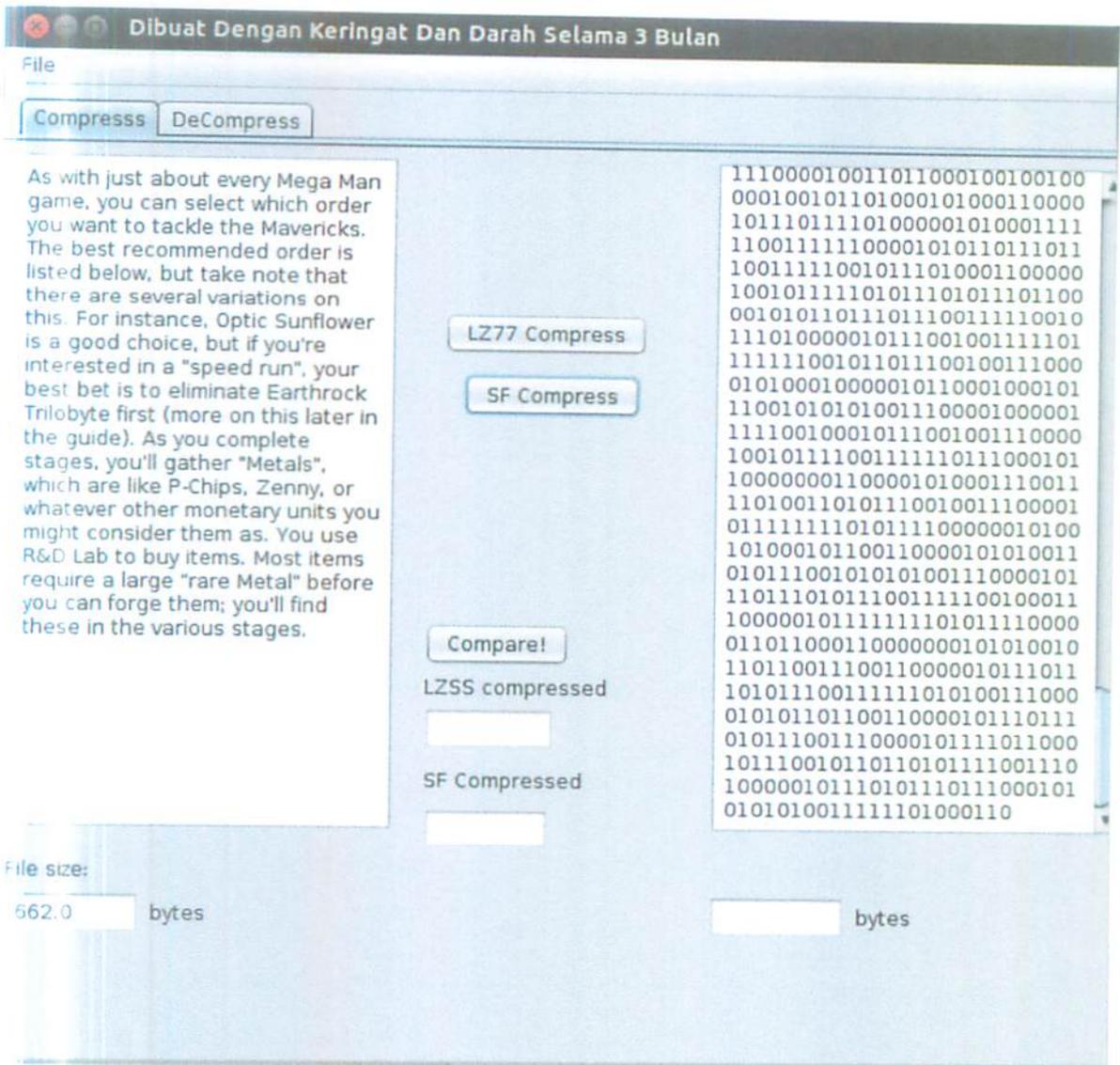


Figure 5.3.5 Shannon – Fano compressed form

This program has a problem in this section. The program is supposed to be able to write an output file and get its file size, but that does not seem to work here. This program does not want to produce a new output binary compressed file in GUI mode. This is because in the action button coding, an IOException cannot be applied, while the output writing code needs a thrown IOException. So the file size of the compressed file of the same file test3.txt will be shown in text mode:

```

rootz@rootz-faptop: ~/Dokumen/finalproject/my.test
0,m53,1,e98,1,d111,1,d119,7,i97,1, 116,1,i96,2,e89,2,b131,1,l106,1,w0,0,,121,2,u
123,2,t146,1,k138,2,n134,1,t148,2,t155,1,a114,3,h144,2,e166,1,a112,3,s171,1,v166
,2,a166,1, 125,1,a182,1,i148,2,i174,1,n190,2,o171,1, 170,2,i161,1,.195,1,F202,1,
r203,1,i161,2,t207,1,n0,0,c211,1,,214,1,00,0,p186,2,c221,1,S0,0,u195,1,f209,1,o0
,0,w229,2, 203,2, 228,1, 0,0,g224,1,o0,0,d238,1,c226,1,o208,2,e219,2,b216,1,t238
,2,f254,1,y250,1,u0,0,'255,1,e258,3,t242,2,e269,2,e243,2,i265,1, 258,2,"266,1,p2
74,1,e267,2,r262,1,n237,1,,265,4,r282,2,e267,2, 245,2,t296,2,s306,1,t301,1, 301,
1,l313,1,m307,2,a311,2, 0,0,E307,1,r323,1,h325,1,o0,0,c0,0,k327,1,T318,1,i295,1,
o315,1,y297,3,f318,1,r329,3,(321,1,o345,1,e353,1,o331,1, 327,2,i351,2,l346,3,r36
7,2,n322,3,e375,1,g0,0,u378,1,d376,1,)0,0,.381,1,A346,2,y384,1,u385,1,c392,1,m0,
0,p391,1,e391,3,s399,1,a367,1,e401,1,,365,4,'398,1,l390,2,a399,3,r417,1,"0,0,M38
9,2,a396,1,s373,1,,431,1,w430,1,i416,1,h436,1,a395,1,e436,1,l389,1,k429,2,P0,0,-
0,0,C404,2,p438,3,Z455,1,n397,1,y453,2,o445,2,w453,1,a458,1,e0,0,v462,3,o472,5,m
438,1,n481,3,r453,1, 0,0,u461,1,i454,1,s490,1,y468,1,u455,2,i0,0,g493,1,t501,1,c
469,2,s506,1,d489,3,t492,2,m513,1,a485,1,.521,1,Y490,3,u501,1,e525,1,R0,0,&0,D
533,1,L525,1,b501,2,o541,1,b535,1,y540,1,l548,1,e537,1,s519,2,M550,1,s547,2,i509
,4, 554,1,e0,0,q547,1,i512,2, 563,1, 0,0,l569,1,r0,0,g565,2,"536,1,a535,3,M583,1
,t580,1,l540,1, 587,1,e0,0,f590,1,r564,2,y600,1,u603,1,c576,1,n607,1,f560,2,g592
,2,t0,0,h610,2,;577,4,'612,1,l583,2,i590,1,d587,4,s632,2,i610,2,t607,2, 0,0,v641
,1,r606,1,o636,1,s650,1,s632,1,a639,2,s0,0,.0,0,
Uncompressed file size: 662.0 bytes
Compressed file size: 237.0 bytes
rootz@rootz-faptop:~/Dokumen/finalproject/my.test$

```

Figure 5.3.6 LZ77 compression in text mode, with proper output file generation and its file size

```

rootz@rootz-faptop: ~/Dokumen/finalproject/my.test
10101011110010110100110100111001000011000001000111111101000001011111101011100
000010100101101011001011011000100111110111001110000101110101101110001010101
01001111101000101000001011111110101110000011011000110000000101010101000101110
11101011100111110000001000001101011100111110111000101100011101000001100101000
00101111101010111011010010101010000101000101100100111000010110001011010111100
11100001001101000101101000101011011011011011010001010000010100001001111001111
0011111110010100000101101011000000101111010101100010111011100111111101100111
1110000001011010111011010110011111100000101100101101011001110011111011100010
1110011111100001011100110011011011101101000001011111101011110000001011001
010110101010101111011000010100101101011001111101010110011010011111100000010
111011101011100111100100000101000101110100011000001001111110101111000000101111
0011101010011100001001101100010010010000010010110100010100011000010111011101000
0010100011111001111100001010110111001111001011101000110000010010111110101
1101011101100001010110111001111001011101000001011100100111101111110010110
1110010011100001010001000001011000100010110010101010011100001000001111100100010
1110010011100001001011110011111011100010110000000110000101000111001111010011010
11100100111000010111111101011110000001010010100010110011000010100110101110010
1010100111000010111011101011100111100100011100000101111110101111000001011000
11000000010101001011001110011000001011101110101110011111101010011100001010110
11001100001011101110101110011000001011101110101110011111101010011100001010110
011101011101100010101010011111101000110
Uncompressed file size: 662.0 bytes
Compressed file size: 476.0 bytes
rootz@rootz-faptop:~/Dokumen/finalproject/my.test$

```

Figure 5.3.7 Shannon - Fano compression in text mode, with proper output file generation and its file size

There is also another problem with the LZ77 compression. If the input file contains only 1 character, like this testx.txt that contains the character 'a', the file size does not change, it stays on 2 bytes like shown in this picture.

```
ferdianb@ferdianb-BTA: ~/Documents/finalproject
1. LZSS
2. Shannon-Fano
Number: 1
behavior pls
1. kompres
2. dekompres
Number: 1
0,0,a - a
0,0,
-
0
44
0
44
97
0
44
0
44
10
Uncompressed file size: 2.0 bytes
Compressed file size: 2.0 bytes
ferdianb@ferdianb-BTA:~/Documents/finalproject$
```

Figure 5.3.8 The unchanged compressed file size

If the text file is changed into something longer, like 2/3/4 characters with no matching characters, the compressed file's size is valued at original file size – 1 as shown in the set of pictures below.

```
ferdianb@ferdianb-BTA: ~/Documents/finalproject
Number: 1
0,0,a - a
0,0,b - b
0,0,
-

0
44
0
44
97
0
44
0
44
98
0
44
0
44
10
Uncompressed file size: 3.0 bytes
Compressed file size: 2.0 bytes
ferdianb@ferdianb-BTA:~/Documents/finalproject$
```

Figure 5.3.9 Testing with 2-characters length

```
ferdianb@ferdianb-BTA: ~/Documents/finalproject

0
44
0
44
97
0
44
0
44
98
0
44
0
44
99
0
44
0
44
10
Uncompressed file size: 4.0 bytes
Compressed file size: 3.0 bytes
ferdianb@ferdianb-BTA:~/Documents/finalproject$
```

Figure 5.3.10 Testing with 3-characters length

```
ferdianb@ferdianb-BTA: ~/Documents/finalproject
97
0
44
0
44
98
0
44
0
44
99
0
44
0
44
100
0
44
0
44
10
Uncompressed file size: 5.0 bytes
Compressed file size: 4.0 bytes
ferdianb@ferdianb-BTA:~/Documents/finalproject$
```

Figure 5.3.11 Testing with 4-characters length

Using any longer text file will no longer produce such result. Instead, it has no clear pattern afterwards, the file size depends on the number of matches found. The more match found, the smaller the file size will be. Another test will be performed using three text files with the same size, but different contents. Text 1 will be abcdefghijk, text 2 will be aaabbcaaabb, and text 3 will be aaaaaaaaaa. The result will be shown in the picture below.

```
ferdianb@ferdianb-BTA: ~/Documents/finalproject
104
0
44
0
44
105
0
44
0
44
106
0
44
0
44
107
0
44
0
44
10
Uncompressed file size: 12.0 bytes
Compressed file size: 8.0 bytes
ferdianb@ferdianb-BTA:~/Documents/finalproject$
```

Figure 5.3.12 Testing with a file without a possible match (abcdefghijkl), generates 8 bytes from 12 bytes file

```
ferdianb@ferdianb-BTA: ~/Documents/finalproject
97
1
44
1
44
97
0
44
0
44
98
1
44
1
44
99
54
44
53
44
10
Uncompressed file size: 12.0 bytes
Compressed file size: 4.0 bytes
ferdianb@ferdianb-BTA:~/Documents/finalproject$
```

Figure 5.3.13 Testing with a file with a few matches (aaabbcaaabb), generates 4 bytes from 12 bytes file.

```
ferdianb@ferdianb-BTA: ~/Documents/finalproject
0
44
0
44
97
1
44
1
44
97
51
44
51
44
97
55
44
52
44
10
Uncompressed file size: 12.0 bytes
Compressed file size: 3.0 bytes
ferdianb@ferdianb-BTA:~/Documents/finalproject$
```

Figure 5.3.14 Testing with a file with a definite match (aaaaaaaaaaa), generates 3 bytes from 12 bytes.

The explanation would be a file with more matches, more characters will also be converted into one single tuple object, in this case aaaaaaaaaaaa becomes only (0,0,a) (1,1,a)(3,3,a),(7,4,a), while the string abcdefghijk becomes a much longer encoded form (0,0,a)(0,0,b)(0,0,c)(0,0,d)(0,0,e)(0,0,f) (0,0,g)(0,0,h)(0,0,i)(0,0,j)(0,0,k) because there is no possible match found, and this of course leads into more bits/bytes, sans a larger compressed file.