Visvesvaraya Technological University
Belagavi-590 018, Karnataka

A MINI PROJECT REPORT ON

"Text File Compression using LZW Algorithm"

Mini-Project Report submitted in Partial fulfillment of the requirement for the 6th Semester File Structures Laboratory with Mini-Project [17ISL68]

Bachelor of Engineering
In
Information Science and Engineering

Submitted by

SRIKANTH S [1JT17IS043]

Under the Guidance of
Mr.Vadiraja A
Asst.Professor,Department of ISE

Department of Information Science and Engineering
Jyothy Institute of Technology,
Tataguni, Bengaluru-56008

Jyothy Institute of Technology,

Department of Information Science and Engineering
Tataguni, Bengaluru-560082

CERTIFICATE

Certified that the mini project work entitled **"Text File Compression"** carried out by **SRIKANTH S [1JT17IS043]** bonafide student of Jyothy Institute of Technology, in partial fulfillment for the award of **Bachelor of Engineering** in **Information Science and Engineering** Department of the **Visvesvaraya Technological University, Belagavi** during the year **2020-2021**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

Mr. Vadiraja A
Guide,Asst, Professor
Dept. Of ISE

Dr. Harshvardhan Tiwari
Associate Professor and HoD
Dept. OF ISE

External Viva Examiner

Signature with Date :

-
-

ACKNOWLEDGMENT

The satisfaction that accompanies the successful completion of my project Would be incomplete without mentioning the people who made it possible, I am thankful to **Dr. Gopalakrishna K,** Principal, JIT, Bangalore for Being kind enough to provide us an opportunity to work on a project in this Institution and I am thankful to **Dr.Harshvardhan Tiwari,**HoD,Dept of ISE for his co-operation and encouragement at all moments of our approach and Whose constant guidance and encouragement crowns all the efforts with Success. I would greatly mention enthusiastic influence provided by **Mr. Vadiraja A,** Asst. Professor, Dept. of ISE for his ideas and co-operation Showed on us during our venture and making this project a great success.Finally,it's pleasure and happiness to the friendly co-operation showed by all The staff members of Information Science Department, JIT.

SRIKANTH S
1JT17IS043

ABSTRACT

The Text File Compression is the data Compression Method in which the logical size of the file is reduced to Save disk Space For easier and Faster Transmission Over a Network. It Enables the Creation of a one or more files the Same data at a Size Substantially Smaller than Orginal file.

File Compression Method Using LZW(Lempel-Ziv-Welch) Algorithm.
This Algorithm is Typically used In GIF and Optionally in POF and TIFF. The Algorithm is Simple to Implement and has the Potential for very high Throughput in Hardware Implementation.
LZW Compression Works by Reading a Sequence of Symbols into Strings and Converting Strings into Codes.Because the Codes take up Less Space than the Strings they Replace We Get Compression.

The Text-File Compression Typically Works by Finding Similar Strings within a Text File and Replacing Those Strings with a Temporary Binary Representation to make overall Filesize Smaller.
It Can be Implemented Through the idea of the Compression Algorithms as the input data is being processed,a Dictionary keeps a Correspondence between the longest encountred words and Lists of Code Values.

This project has been created using Eclipse, with the platform WINDOWS. The project title-“Text File Compression” talks about how we Compress the Given File and data accepted from the USER Interface(UI) and also use it to retrieve Information.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

INTRODUCTION

1.1 Introduction to File Structures

In simple terms, a file is a collection of data stored on mass storage (e.g., disk or tape).But there is one important distinction that must be made at the outset when discussing file structures. And that is the difference between the logical and physical organization of the data.

On the whole a file structure will specify the logical structure of the data, that is the relationships that will exist between data items independently of the way in which these relationships may actually be realized within any computer. It is this logical aspect that we will concentrate on. The physical organization is much more concerned with optimizing the use of the storage medium when a particular logical structure is stored on, or in it. Typically for every unit of physical store there will be a number of units of the logical structure (probably records) to be stored in it.

For example, if we were to store a tree structure on a magnetic disk, the physical organization would be concerned with the best way of packing the nodes of the tree on the disk given the access characteristics of the disk.

Like all subjects in computer science the terminology of file structures has evolved higgledy-piggledy without much concern for consistency, ambiguity, or whether it was possible to make the kind of distinctions that were important.

It was only much later that the need for a well-defined, unambiguous language to describe file structures became apparent. In particular, there arose a need to communicate ideas about file structures without getting bogged down by hardware considerations.

1.2 Introduction to File System

In computing, a file system or file system controls how data is stored and retrieved. Without a file system, information placed in a storage medium would be one large body of data with noway to tell where one piece of information stops and the next begins. By separating the data into pieces and giving each piece a name, the information is easily isolated and identified.
Taking its name from the way paper-based information systems are named, each groups of data is called a “file”. The structure and logic rules used to manage the groups of information and their names is called a “file system”.

There are many different kinds of file systems. Each one has different structure and logic, properties of speed, flexibility, security, size and more. Some file systems have been designed to be used for specific applications. For example, the ISO 9660 file system is designed specifically for optical discs.
File systems can be used on numerous different types of storage devices that use different kinds of media. The most common storage device in use today is a hard disk drive. Other kinds of media that are used include flash memory, magnetic tapes, and optical discs. In some cases, such as with tmpfs, the computer's main memory (random-access memory, RAM) is used to create a temporary file system for short-term use.

Some file systems are used on local data storage devices; others provide file access via a network protocol(for example, NFS, SMB, or 9P clients). Some file systems are “virtual”. Meaning that the supplied “files” (called virtual files) are computed on request (e.g. procfs) or are merely a mapping into a different file system used as a blacking store. The file system manages access to both the content of files and the metadata about those files. It is responsible for arranging storage space; reliability, efficiency, and tuning with regard to the physical storage medium are important design considerations.

1.3 Data Compression

With the advancement in technology, multimedia content of digital information is increasing day by day, which mainly comprises of TextFile. Hence storage and transmission of these files require a large memory space and bandwidth. The solution is to reduce the storage space required for these files while maintaining acceptable File quality.

How compression works

Compression is performed by a program that uses a formula or algorithm to determine how to shrink the size of the data. For instance, an algorithm may represent a string of bits -- or 0s and 1s -- with a smaller string of 0s and 1s by using a dictionary for the conversion between them, or the formula may insert a reference or pointer to a string of 0s and 1s.

Text compression can be as simple as removing all unneeded characters, inserting a single repeat character to indicate a string of repeated characters and substituting a smaller bit string for a frequently occurring bit string. Data compression can reduce a text file to 50% or a significantly higher percentage of its original size.

For data transmission, compression can be performed on the data content or on the entire transmission unit, including header data. When information is sent or received via the internet, larger files, either singly or with others as part of an archive file, may be transmitted in a ZIP, GZIP or other compressed Format

LZW (Lempel–Ziv–Welch) Compression technique

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. It is the algorithm of the widely used Unix file compression utility compress and is used in the GIF image format.

- LZW compression works best for files containing lots of repetitive data. This is often the case with text and monochrome images. Files that are compressed but that do not contain any repetitive information at all can even grow bigger!
- LZW compression is fast.
- LZW is a fairly old compression technique. All recent computer systems have the horsepower to use more efficient algorithms.
- Royalties have to be paid to use LZW compression algorithms within applications (see below).

LZW is a dictionary coder. It starts with a dictionary that has entries for all one-byte sequences, associating them with codes. The dictionary is updated with multiple byte sequences, as the data file DF is read. The encoded file EF will consist entirely of codes.

LZW compression works by reading a sequence of symbols, grouping the symbols into strings, and converting the strings into codes. Because the codes take up less space than the strings they replace, we get compression.Characteristic features of LZW includes,

- LZW compression uses a code table, with 4096 as a common choice for the number of table entries. Codes 0-255 in the code table are always assigned to represent single bytes from the input file.
- When encoding begins the code table contains only the first 256 entries, with the remainder of the table being blanks. Compression is achieved by using codes 256 through 4095 to represent sequences of bytes.
- As the encoding continues, LZW identifies repeated sequences in the data, and adds them to the code table.
- Decoding is achieved by taking each code from the compressed file and translating it through the code table to find what character or characters it represents.

The compressor algorithm builds a string translation table from the text being compressed. The string translation table maps fixed-length codes (usually 12-bit) to strings. The string table is initialized with all single-character strings (256 entries in the case of 8-bit characters). As the compressor character-serially examines the text, it stores every unique two-character string into the table as a code/character concatenation, with the code mapping to the corresponding first character. As each two-character string is stored, the first character is outputted. Whenever a previously-encountered string is read from the input, the longest such previously-encountered string is determined, and then the code for this string concatenated with the extension character (the next character in the input) is stored in the table. The code for this longest previously-encountered string is outputted and the extension character is used as the beginning of the next string.

LZW compression works by reading a sequence of symbols, grouping the symbols into strings, and converting the strings into codes. Because the codes take up less space than the strings they replace, we get compression.Characteristic features of LZW includes,

- LZW compression uses a code table, with 4096 as a common choice for the number of table entries. Codes 0-255 in the code table are always assigned to represent single bytes from the input file.
- When encoding begins the code table contains only the first 256 entries, with the remainder of the table being blanks. Compression is achieved by using codes 256 through 4095 to represent sequences of bytes.
- As the encoding continues, LZW identifies repeated sequences in the data, and adds them to the code table.
- Decoding is achieved by taking each code from the compressed file and translating it through the code table to find what character or characters it represents.

LZW (Lempel–Ziv–Welch) DeCompression technique

The LZW decompressor creates the same string table during decompression. It starts with the first 256 table entries initialized to single characters. The string table is updated for each character in the input stream, except the first one.Decoding achieved by reading codes and translating them through the code table being built.

The decompressor algorithm only requires the compressed text as an input, since it can build an identical string table from the compressed text as it is recreating the original text. However, an abnormal case shows up whenever the sequence character/string/character/string/character (with the same character for each character and string for each string) is encountered in the input and character/string is already stored in the string table. When the decompressor reads the code for character/string/character in the input, it cannot resolve it because it has not yet stored this code in its table. This special case can be dealt with because the decompressor knows that the extension character is the previously-encountered character.

Pseudo-Code

```
 1: begin
 2:    tmp = ""
 3:    initialization of the dictionary
 4:    while ((code = read code) != EOF) do
 5:     begin
 6:      if ( is phrase with code (code) in dictionary ) then
 7:       begin
 8:         phrase := dictionary[code]
 9:         output(phrase)
10:         add to dictionary (tmp + first character of phrase)
11:       end
12:      else
13:       begin
14:         if ( code > maximum index of dictionary + 1 ) then
15:            exit
16:         phrase := tmp + first char of tmp
17:         output(phrase)
18:         add to dictionary (phrase)
19:       end
20:      tmp := phrase
21:    end
22: end
```

CHAPTER 2

REQUIREMENT ANALYSIS AND DESIGN

REQUIREMENT ANALYSIS AND DESIGN

2.1 Domain Understanding

The Main Objective of this Project is to Compress The Given Text File.The outcome is it gives a Compressed File in an Efficient,Friendly and In understandable GUI.

2.2 Classification of Requirements

System Requirements

- OS:Windows
- Java Installed

Software and Hardware Requirements

Programming Languages: Front End – JAVA Swings

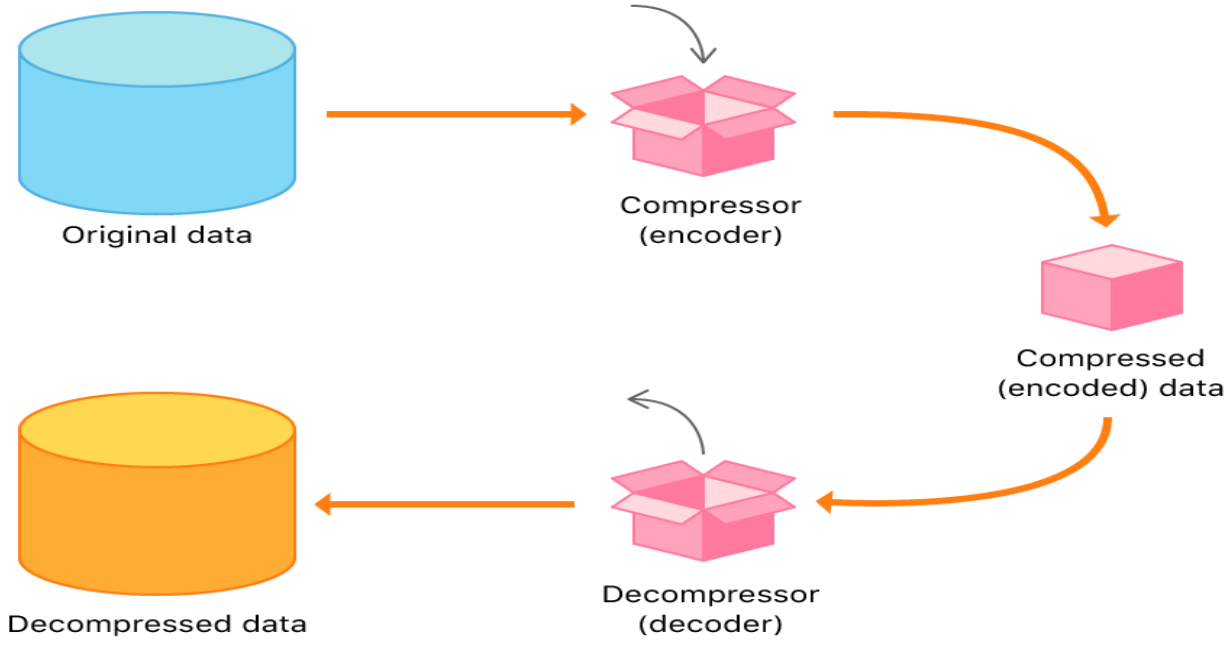
Back End –JAVA

2.1 System Analysis

When the program is executed, it loads a simple Gui which primarily asks the user to press the Compression and Decompression whatever they want to access. If the user chooses the Compression button, then it asks to input the TextFile which they want to compress that. After the TextFile is obtained, the entire File is viewed for verification of Size of the file with name and location.

When user press Compression button it starts to compress the TextFile. After the completion of compressing, the user is given information about compression such as time taken, Compressed Textfile Size, and The New File Path.

When user press Decompression button it starts to Decompress the TextFile.and it takes only LZW File.After the completion of decompressing, the user is given information about decompression such as time taken, Decompressed Textfile Size, and The New File Path.


Original data
Compressor
(encoder)
Compressed
(encoded) data
Decompressed data
Decompressor
(decoder)

CHAPTER 3
IMPLEMENTATION

IMPLEMENTATION

The Text File Compression Using LZW Algorithm is Implemented Through Two Types of Tehnique

- LZW COMPRESSION
- LZW DECOMPRESSION

- **LZW COMPRESSION**

Algorithm for LZW Compression:

Step-1: Initialize table with single character strings
Step-2 : first input character
Step-3: WHILE not end of input stream
Step-4: C = next input character
Step-5: IF P + C is in the string table
Step-6: P = P + C
ELSE
Step-7: output the code for P
Step-8: add P + C to the string table
Step-9: P = C
END WHILE
Step-10: output code for P

The Above Algorithm Shows How The LZW Algorithm Compresses The Given File.

LZW DECOMPRESSION

Algorithm for LZW Decompression

```
Step-1:   Initialize table with single character strings
Step-2:    OLD = first input code
Step-3:    output translation of OLD
Step-4:    WHILE not end of input stream
Step-5:        NEW = next input code
Step-6:        IF NEW is not in the string table
Step-7:                S = translation of OLD
Step-8:                S = S + C
           ELSE
Step-9:               S = translation of NEW
Step-10:       output S
Step-11:       C = first character of S
Step-12:       OLD + C to the string table
Step-13:       OLD = NEW
  END WHILE
```

The Above Algorithm Shows How The LZW Algorithm DeCompresses The Given File.

COMPRESSION

```
public void actionPerformed(ActionEvent evt)
{
    // if the user presses the save button show the save dialog
    String com = evt.getActionCommand();

    if (com.equals("COMPRESSION")) {
        // create an object of JFileChooser class
        JFileChooser j = new JFileChooser(FileSystemView.getFileSystemView().getHomeDirectory());
        j.setAcceptAllFileFilterUsed(false);

        // set a title for the dialog
        j.setDialogTitle("Select a .pdf file");

        // only allow files of .txt extension
        FileNameExtensionFilter restrict = new FileNameExtensionFilter("Only .txt files", "txt");
        j.addChoosableFileFilter(restrict);

        // invoke the showsSaveDialog function to show the save dialog
        int r = j.showOpenDialog(null);

        // if the user selects a file
        if (r == JFileChooser.APPROVE_OPTION)

        {
            // set the label to the path of the selected file
            l.setText(j.getSelectedFile().getAbsolutePath());
        }
        // if the user cancelled the operation
        else
            l.setText("the user cancelled the operation");
        try
        {
        compression lzw = new compression();

         File file = new File(j.getSelectedFile().getAbsolutePath());

         Scanner fileScanner = new Scanner(file);

         String line = "";
```

Figure 1

Fig 1:The Below Function is used To Compress the Given Text File Using LZW Algorithm.It Takes only .txt File It Reads the File and then it Converts into LZW Format(.lzw).

DECOMPRESSION

```
// create an object of JFileChooser class
JFileChooser j = new JFileChooser(FileSystemView.getFileSystemView().getHomeDirectory());
j.setAcceptAllFileFilterUsed(false);

// set a title for the dialog
j.setDialogTitle("Select a .lzw file");

// only allow files of .txt extension
FileNameExtensionFilter restrict = new FileNameExtensionFilter("Only .lzw files", "lzw");
j.addChoosableFileFilter(restrict);


// invoke the showsOpenDialog function to show the save dialog
int r = j.showOpenDialog(null);

// if the user selects a file
if (r == JFileChooser.APPROVE_OPTION)

{
    // set the label to the path of the selected file
    l.setText(j.getSelectedFile().getAbsolutePath());
}
// if the user cancelled the operation
else
    l.setText("the user cancelled the operation");
try {
    decompression lzw = new decompression();


    File file = new File(j.getSelectedFile().getAbsolutePath());

    Scanner fileScanner = new Scanner(file);

    String line = "";

    while (fileScanner.hasNext()) {
        line = fileScanner.nextLine();
        System.out.println("Contents of your file being decompressed:\n"+ line);
    }
```

Figure 2

Fig 2: The Above function Shows How the Compressed File is again Decompress to its Orginal File.It takes the Decompressed File like .lzw Files and it Converts its .txt File And Back to its Orginal Size.

USER INTERFACE

```
// a default constructor
UI()
{
    getContentPane().setLayout(null);
}

public static void main(String args[])
{
    //FS2 a=new FS2();
    //a.setVisible(true);
    // frame to contains GUI elements
    JFrame f = new JFrame("file chooser");



    // set the size of the frame
    f.setSize(1000,500);
    f.setBackground(Color.black);

    // set the frame's visibility
    f.setVisible(true);

    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    l = new JLabel("no file selected");
    l = new JLabel("LZW Compression and Decompression",JLabel.CENTER);
    l.setVerticalAlignment(JLabel.TOP);
    l.setFont(new Font("Verdana", Font.BOLD, 15));


    // button to open save dialog
    JButton button1 = new JButton("COMPRESSION");
    button1.setBounds(52, 105, 147, 55);
    button1.setFont(new Font("Tahoma", Font.BOLD, 15));
    button1.setForeground(Color.BLACK);
    button1.setBackground(Color.WHITE);
    button1.setAlignmentX(Component.RIGHT_ALIGNMENT);


    // button to open open dialog
    JButton button2 = new JButton("DECOMPRESSION");
   button2.setForeground(Color.BLACK);
    button2.setBackground(Color.WHITE);
    button2.setFont(new Font("Tahoma", Font.BOLD, 15));
    button2.setBounds(263, 105, 147, 55);
```

Figure 3

Fig 3:The Above Function Is for the Representation of the User Interface.

Figure 4

Fig 4:The Fig 4 is the User Interface Overview Page In Which It Contains Two Buttons Compression and Decompression.

CHAPTER 4

RESULT AND ANALYSIS

RESULT AND ANALYSIS

Figure 5:-HOME SCREEN

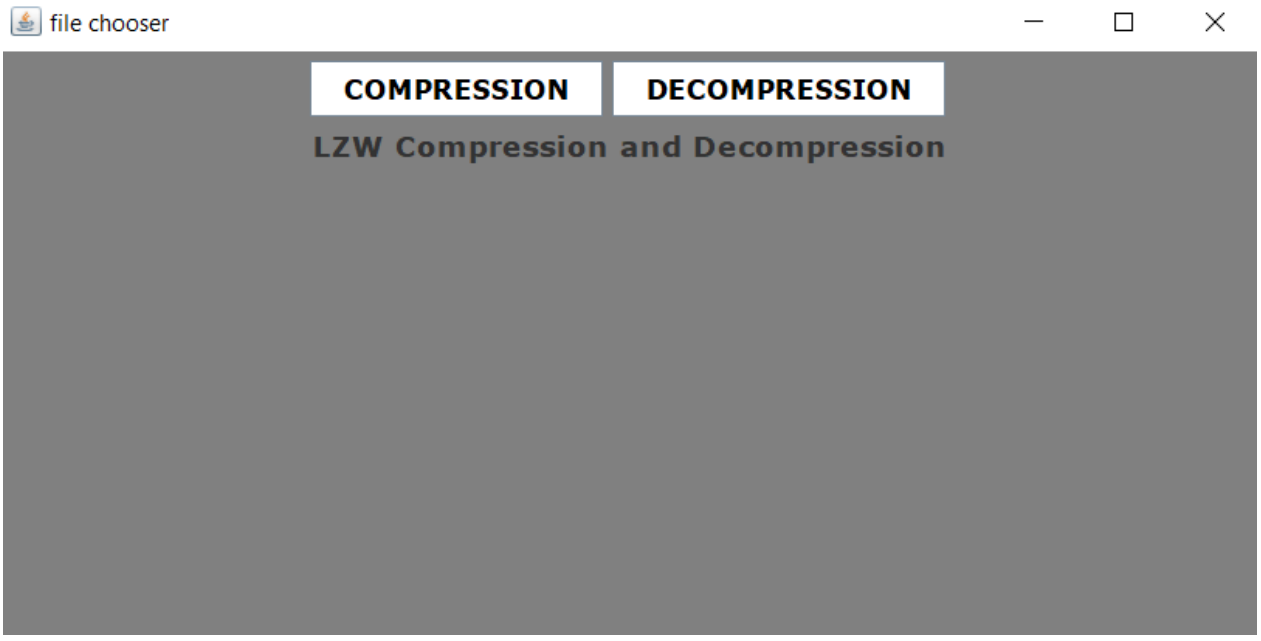
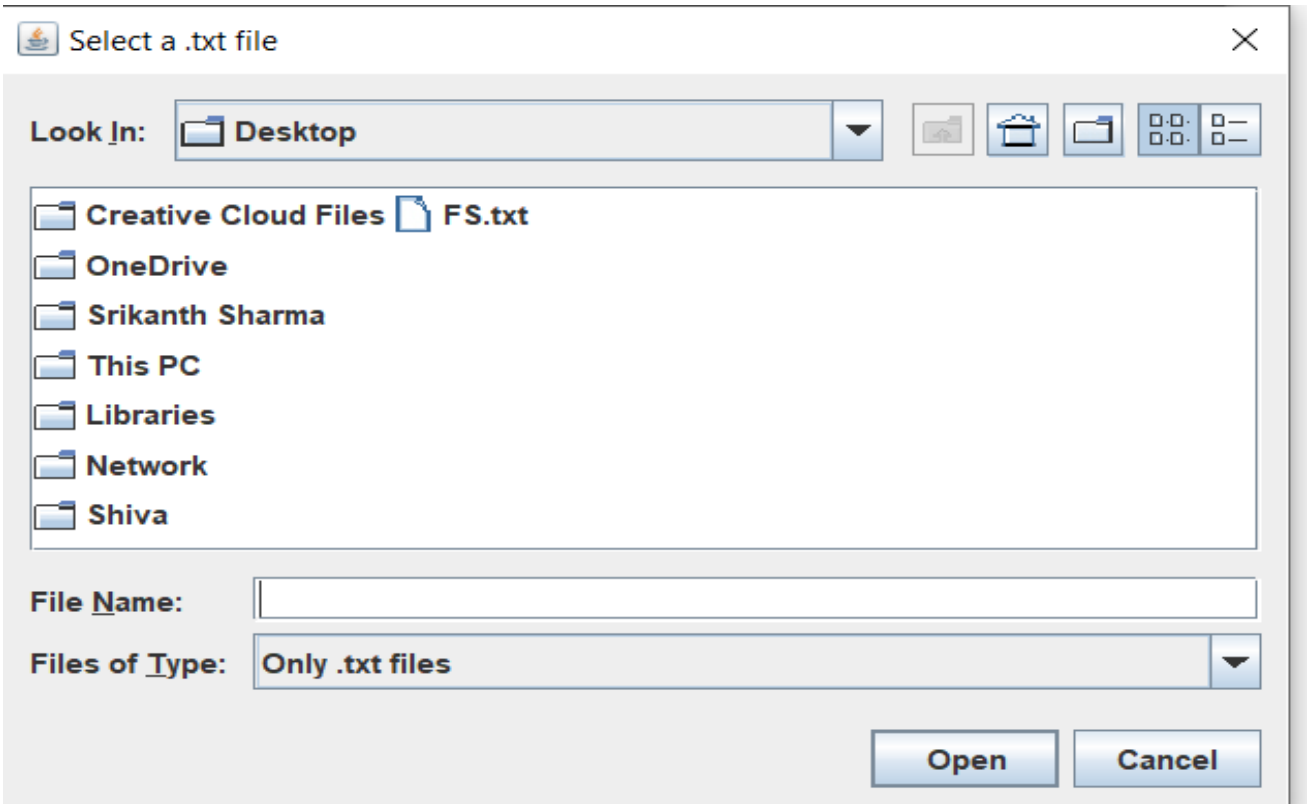


Figure 5

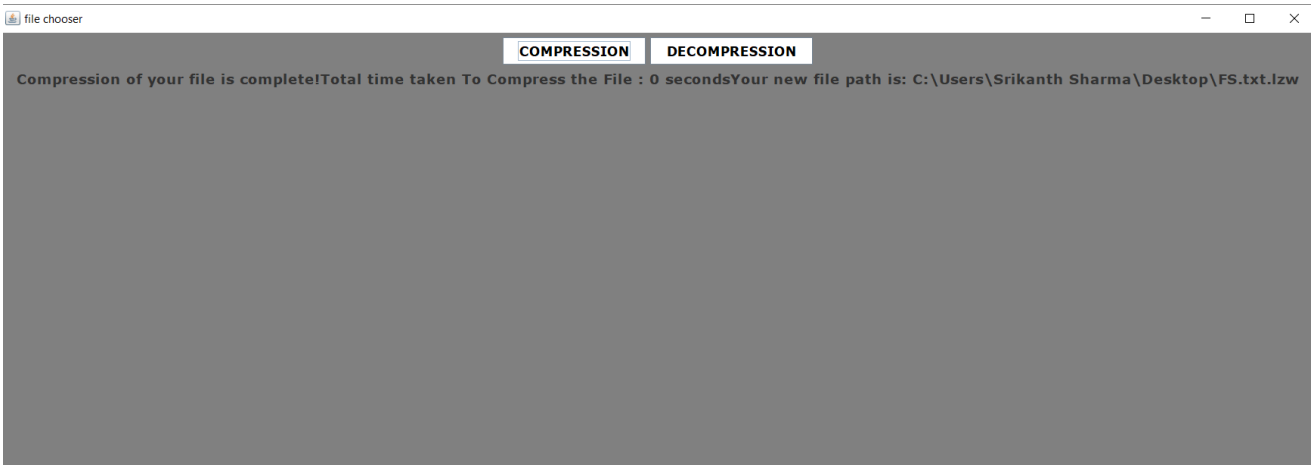
The Above Figure Is the User Interface Home Screen In which it Contains Two Buttons Compression and Decompression and The Title LZW Compression and Decompression.

Figure 6:-Compression



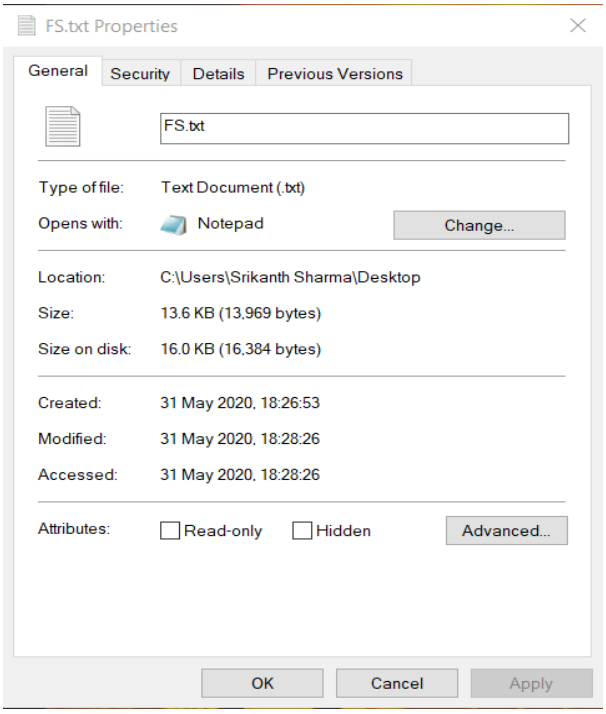
The Compression Field In which User has to Choose File in Which They want to Compress and The Condition is Here it Accepts only .txt Files only.

Figure 7:-Compression Result



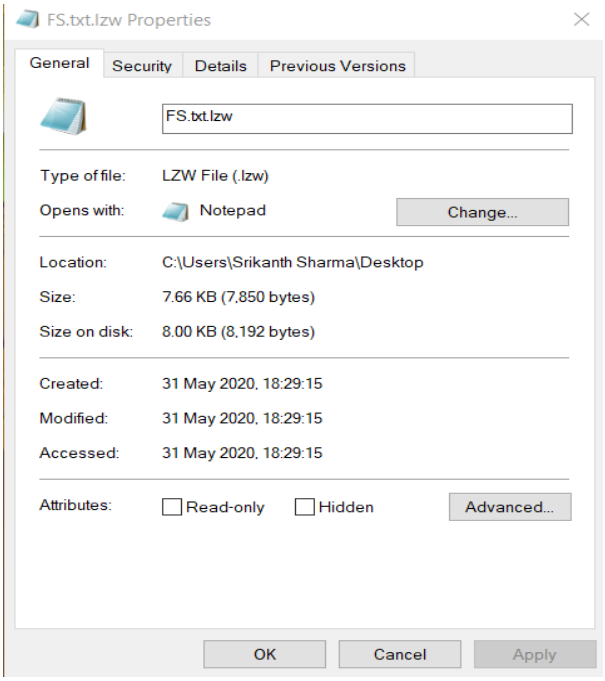
After the Compression The in The Home Screen it Shows the Result To the User in which It Contains Result and Time Taken To compress the File And Compressed File Path.

Figure 8:-Compression Analysis



Orginal File

Figure 9



In the above two Figures Fig 8 and 9 Orginal File Size was 13 KB and it is Compressed to 8KB.

Result and Analysis

Figure 10:-Compression Graph(Time V/s SIZE)

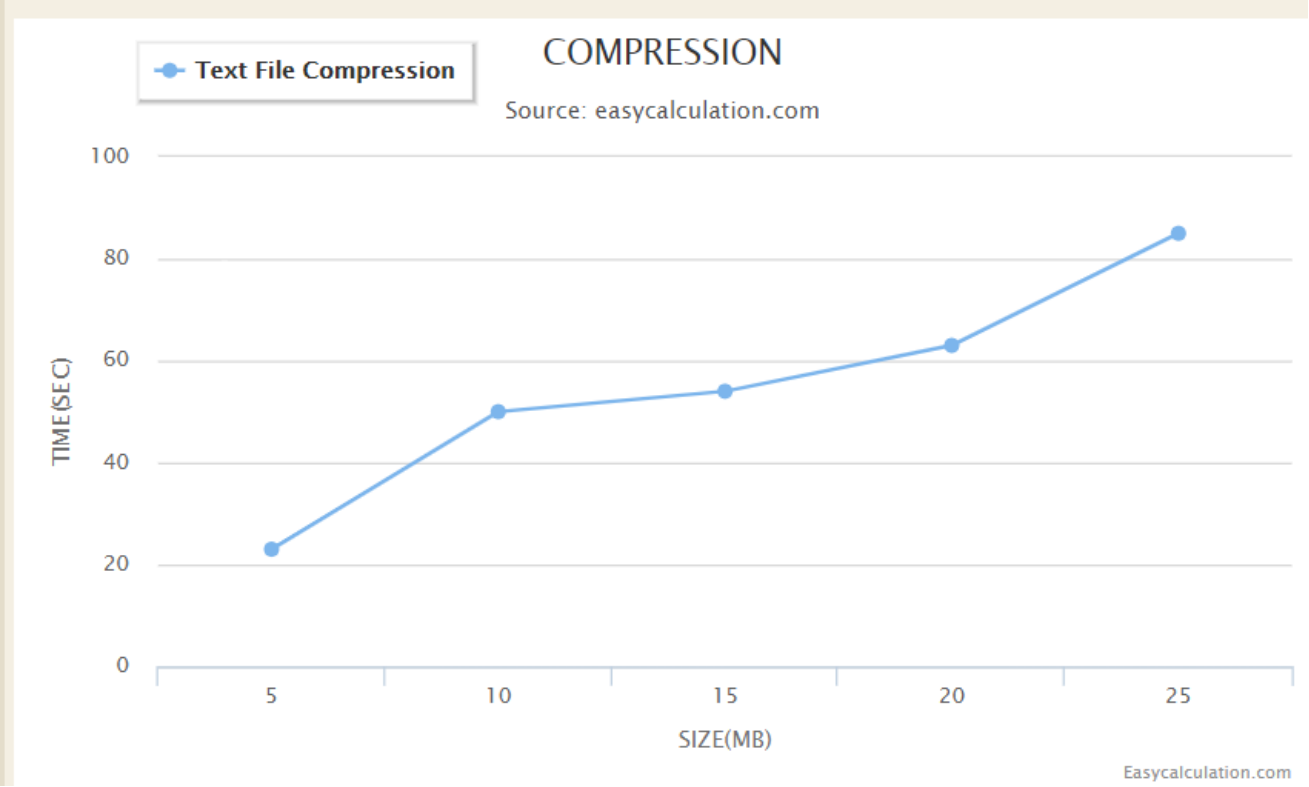
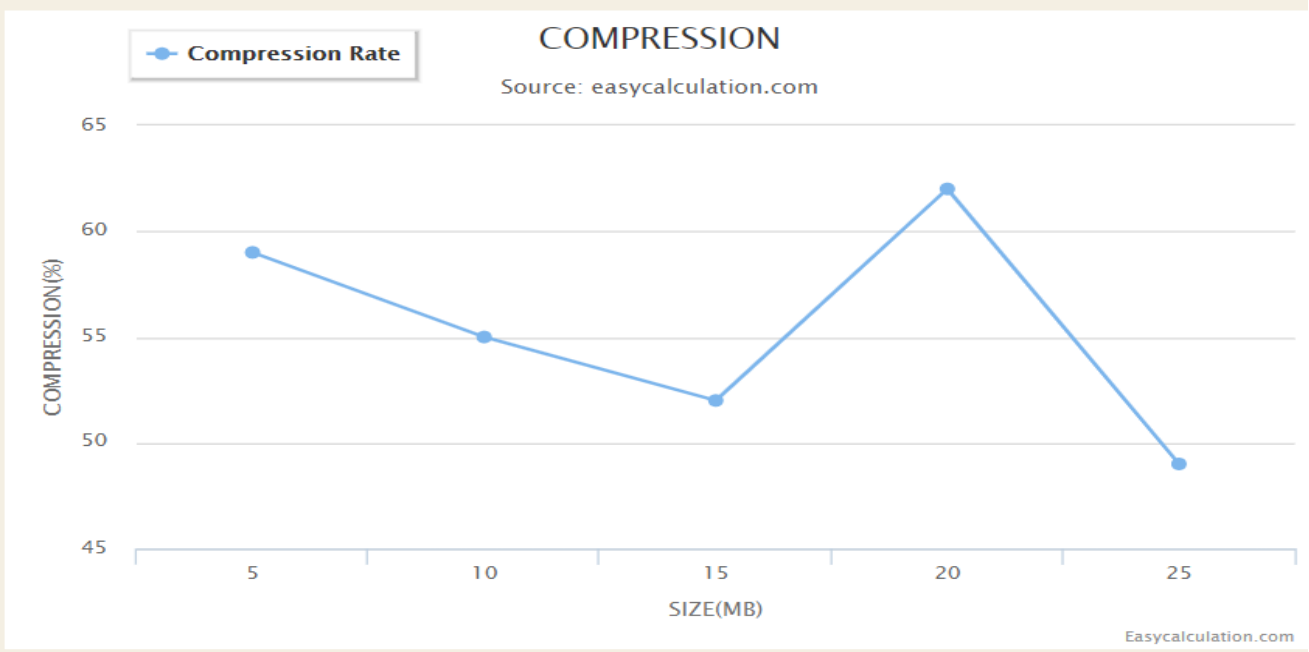


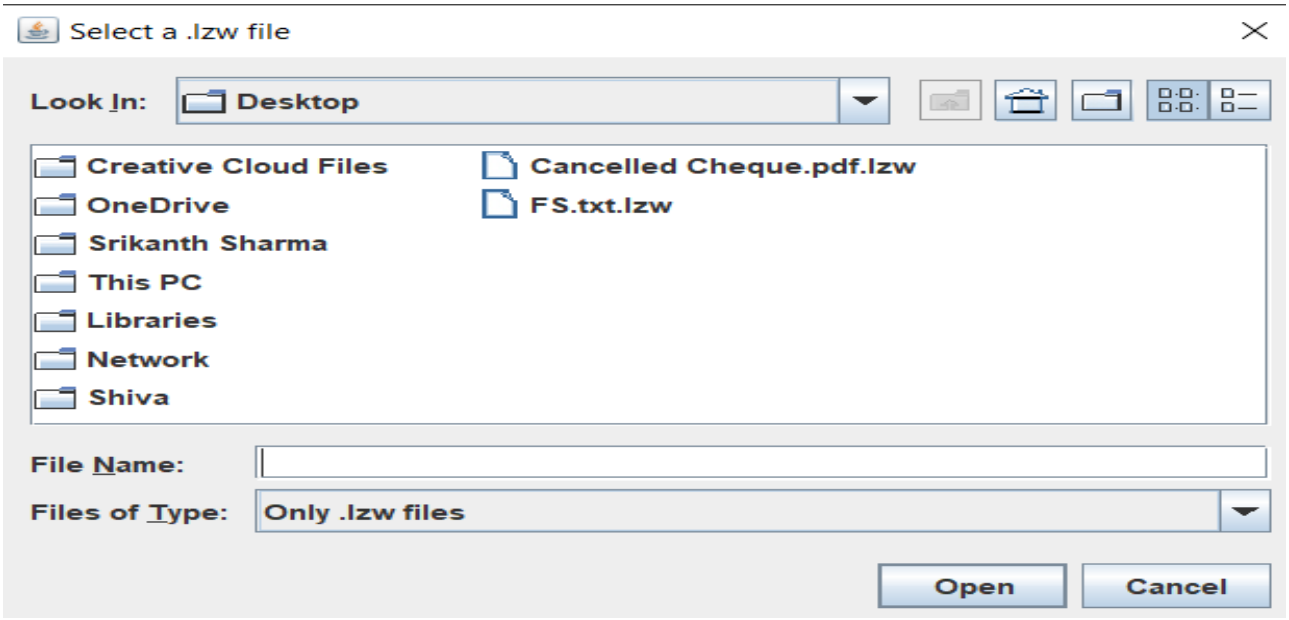
Figure 11:-Compression Graph(Time V/s Compression rate)



Result and Analysis

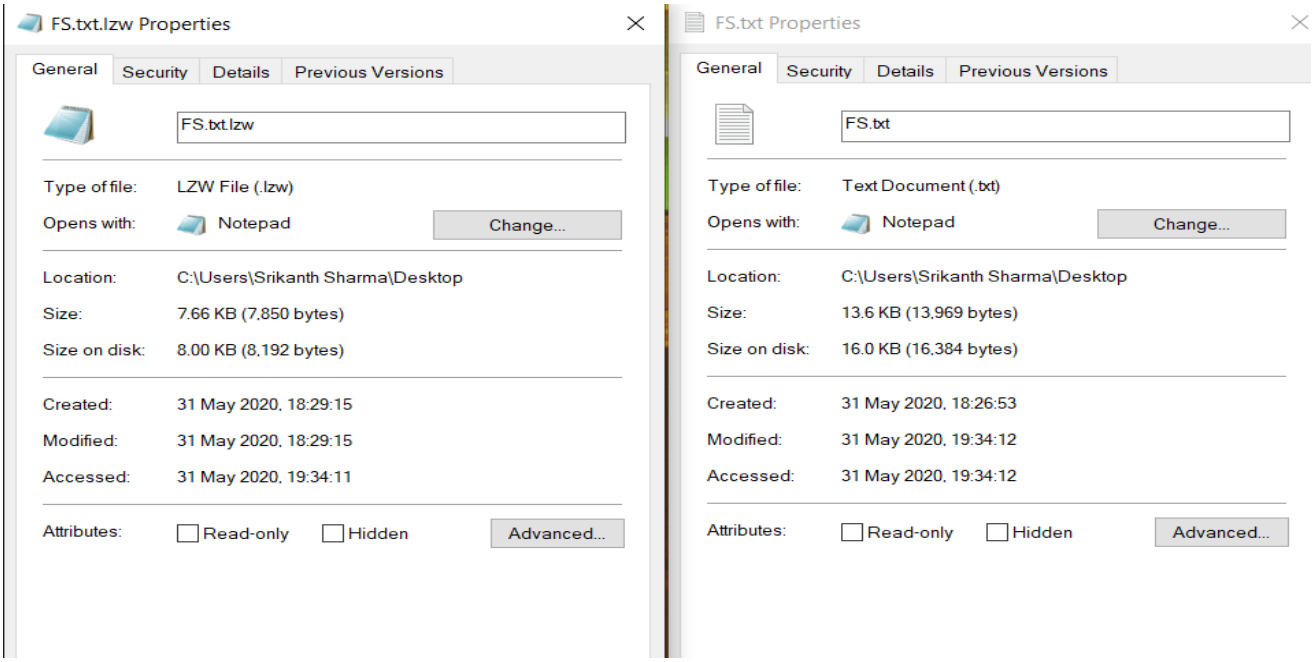
Figure 12:-DeCompression

Page| 18



The DeCompression Field In which User has to Choose File in Which They want to DeCompress and The Condition is Here it Accepts only .lzw Files only.

Figure 13:-DeCompression Analysis



Result and Analysis

Figure 13:-Decompression Graph(Time vs Size)

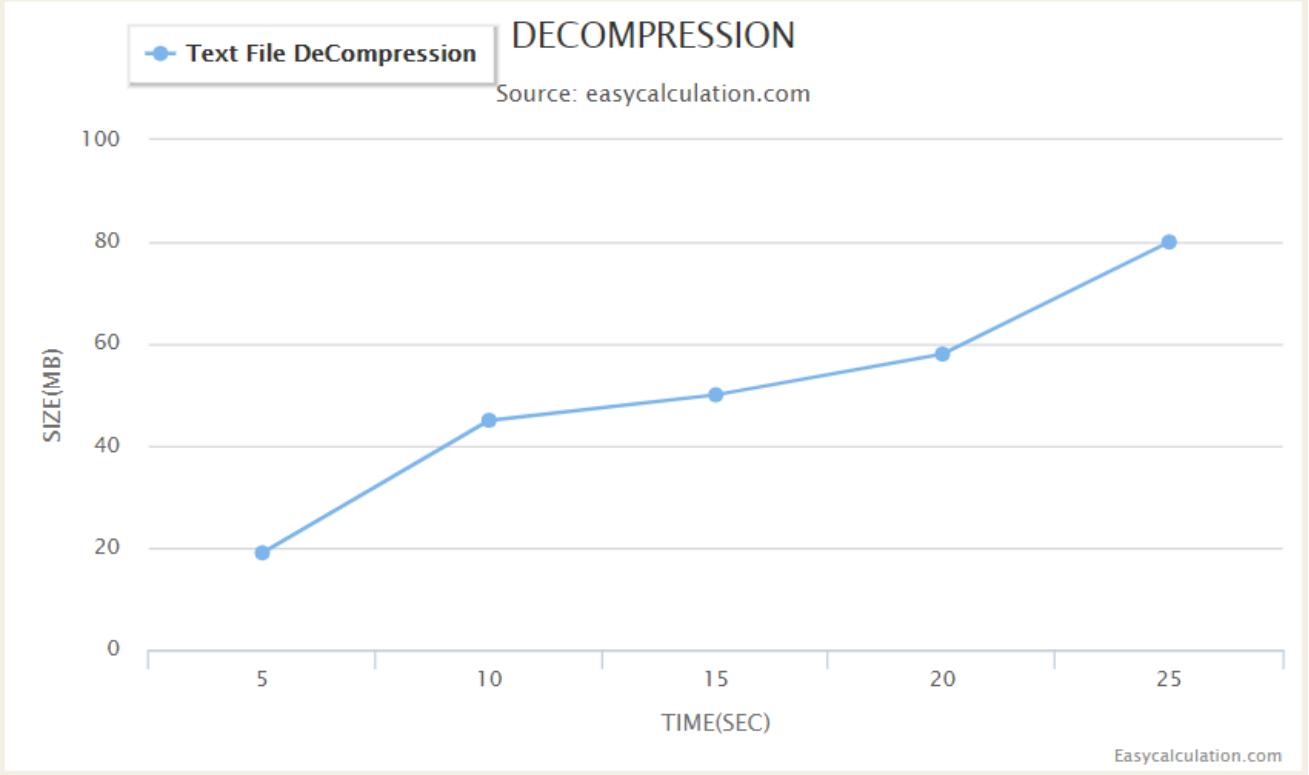
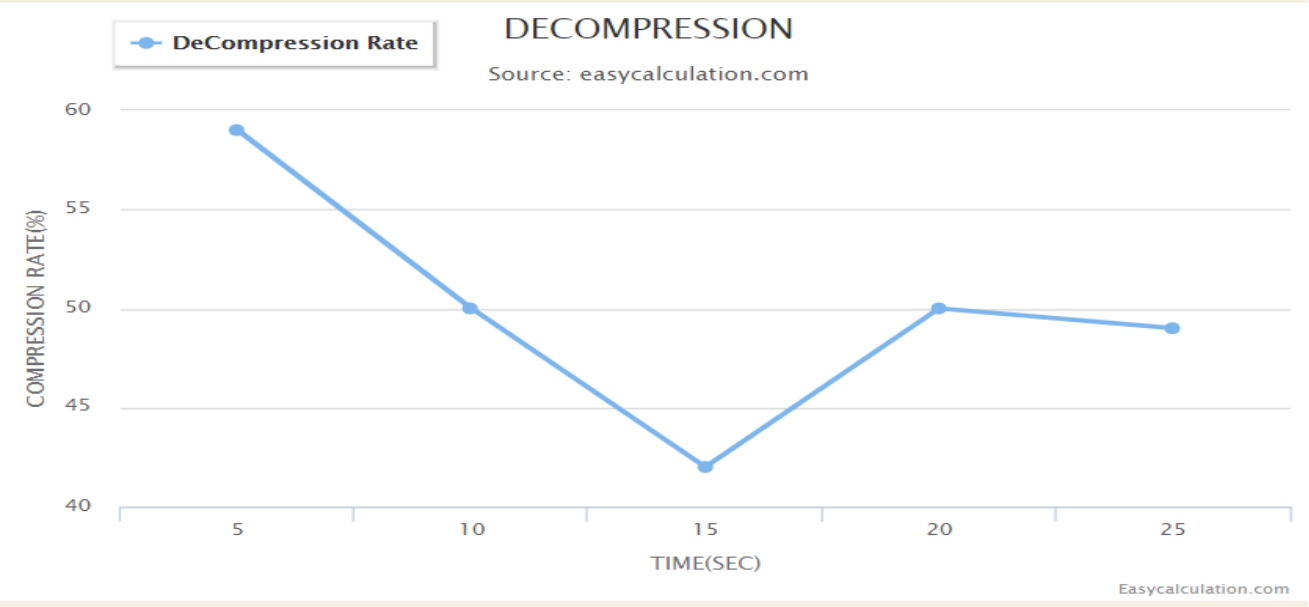


Figure 14:-DeCompression Graph(Time V/s Compression rate)



CONCLUSION:

enabling you to record a more clear, clean sound. Technically speaking, if instruments

become too loud or too soft during a recording, the sound levels can pitch too high or too low within the range your equipment can capture.

In This Mini-Project I learnt How the data is Compressed and How Data is Decompressed Using Different Forms of Algorithms.

Result and Analysis

REFERNCES:

[1]File Compression Journals:
https://academic.oup.com/comjnl/article-abstract/48/6/677/358346?redirectedFrom=fulltext

[2] LZW Compression Journal by American Journal of Engineering Research (AJER)
http://www.ajer.org/papers/v3(2)/C0322226.pdf

3) https://stackoverflow.com/questions/62086402/lzw-decompression

4) https://tutorialspoint.dev/computer-science/computer-network-tutorials/lzw-lempel-ziv-welch-compression-technique

5)File Organization by Binnur Kurt.

6) https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/

7) https://dspguide.com/ch27/5.htm