"Data Compression Decompression and Image Steganography"

A Minor Project Report Submitted to
Rajiv Gandhi Proudyogiki Vishwavidyalaya

Towards Partial Fulfillment for the Award of
Bachelor of Engineering in Computer Science Engineering

Submitted by:
Aadesh Garg(0827CS201001)
Amisha Linjhara (0827CS201029)
Amisha Prajapati (0827CS201030)
Ananya Shrivastava
(0827CS201032)

Guided by:
Prof. Priyanka Jangde
Professor, CSE

Acropolis Institute of Technology & Research, Indore
Jan - June 2023

EXAMINER APPROVAL

The Minor Project entitled ***"*Data Compression Decompression and Image Steganography*"*** submitted by **Aadesh Garg(0827CS201001) Amisha Linjhara (0827CS201029) Amisha Prajapati (0827CS201030) Ananya Shrivastava (0827CS201032)** has been examined and is hereby approved towards partial fulfillment for the award of ***Bachelor of Technology degree in Computer Science Engineering*** discipline, for which it has been submitted. It understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed, or conclusion drawn therein, but approve the project only for the purpose for which it has been submitted.

(Internal Examiner)	**(External Examiner)**
Date:	**Date:**

RECOMMENDATION

This is to certify that the work embodied in this minor project entitled **“Data Compression Decompression and Image Steganography”** submitted by **Aadesh Garg(0827CS201001) Amisha Linjhara (0827CS201029) Amisha Prajapati (0827CS201030) Ananya Shrivastava (0827CS201032)** a satisfactory account of the bonafide work done under the supervision of ***Dr. Kamal Kumar Sethi***, is recommended towards partial fulfillment for the award of the Bachelor of Technology (Computer Science Engineering) degree by Rajiv Gandhi Proudyogiki Vishwavidhyalaya, Bhopal.

(Project Guide)

(Project Coordinator)

(Dean Academics)

STUDENTS UNDERTAKING

This is to certify that the minor project entitled **"Data Compression Decompression and Image Steganography"** has developed by us under the supervision of ***Dr. Kamal Kumar Sethi***. The whole responsibility of the work done in this project is ours. The sole intension of this work is only for practical learning and research.

We further declare that to the best of our knowledge; this report does not contain any part of any work which has been submitted for the award of any degree either in this University or in any other University / Deemed University without proper citation and if the same work found then we are liable for explanation to this.

Aadesh Garg(0827CS201001)

Amisha Linjhara (0827CS201029)

Amisha Prajapati (0827CS201030)

Ananya Shrivastava (0827CS201032)

Acknowledgement

We thank the almighty Lord for giving me the strength and courage to sail out through the tough and reach on shore safely.

There are number of people without whom this project would not have been feasible. Their high academic standards and personal integrity provided me with continuous guidance and support.

We owe a debt of sincere gratitude, deep sense of reverence and respect to our guide and mentor **Dr. Kamal Kumar Sethi,** Professor, AITR, Indore for his motivation, sagacious guidance, constant encouragement, vigilant supervision, and valuable critical appreciation throughout this project work, which helped us to successfully complete the project on time.

We express profound gratitude and heartfelt thanks to **Dr Kamal Kumar Sethi**, Professor & Head CSE, AITR Indore for his support, suggestion, and inspiration for carrying out this project. I am very much thankful to other faculty and staff members of the department for providing me all support, help and advice during the project. We would be failing in our duty if do not acknowledge the support and guidance received from **Dr S C Sharma**, Director, AITR, Indore whenever needed. We take opportunity to convey my regards to the management of Acropolis Institute, Indore for extending academic and administrative support and providing me all necessary facilities for project to achieve our objectives.

We are grateful to **our parent** and **family members** who have always loved and supported us unconditionally. To all of them, we want to say "Thank you", for being the best family that one could ever have and without whom none of this would have been possible.

Aadesh Garg(0827CS201001) ,Amisha Linjhara (0827CS201029) ,Amisha Prajapati (0827CS201030) ,Ananya Shrivastava (0827CS201032)

Executive Summary

This project **“Data Compression Decompression and Image Steganography”** is submitted to Rajiv Gandhi Proudyogiki Vishwavidhyalaya, Bhopal (MP), India for partial fulfillment of Bachelor of Engineering in Information Technology branch under the sagacious guidance and vigilant supervision of ***Dr. Kamal Kumar Sethi***.

The project “Data Compression Decompression and image steganography” System is totally built on Java technology and provides an interface to the users so that they can easily store and transfer large files. This java project also provides encoding techniques in compression so as to ensure the security of the data. Using this project, the users can compress the data according to the requirement at any point and for any number of times.

The layout of the project is built using the swing and AWT packages of Java using their predefined classes, interfaces, and methods. The io(Input Output) and util packages of Java provide the predefined classes to compress and decompress the files directly by passing the filename and location.

"Where the vision is one year, cultivate flowers;

Where the vision is ten years, cultivate trees;

Where the vision is eternity, cultivate people."

- Oriental Saying

List of Figures

Table of Contents

Chapter 1.Introduction

Introduction

With the development of the Internet, information processing technologies and the rapid development of communication, it is necessary to share information resources, and the network has becoming the main means of communication. Nevertheless, the Internet is an open environment so; information security has becoming increasingly important. Today, information security technology has two main branches are cryptography and information hiding. Steganography is the art of hiding information in such a way that prevents the detection of hidden messages. The message is the data that the sender wants to remain confidential. It can be in the form of text, image, audio, video, or any other data that can be represented by a stream of bits. The cover or host is the medium in which the message is embedded. It serves to hide the presence of the message. We can use gray images, videos, sound files, and other computer files that contain perceptually irrelevant or redundant information as a cover image. It is important to note that the hidden data is not detectable in the stego-image.

1.1 Overview

The goals of this study were to develop a system intended for securing files through the technique of image steganography integrated with cryptography by utilizing ZLIB Algorithm for compressing and decompressing secret files, DES Algorithm for encryption and decryption, and Least Significant Bit Algorithm for file embedding and extraction to avoid compromise on highly confidential files from exploits of unauthorized persons. The system will be excellently effective based on Functionality, Reliability, Usability, Efficiency, Maintainability and Portability. The system can be a useful tool for both government agencies and private institutions for it could keep not only the message secret but also the existence of that particular message or file secret maintaining the privacy of highly confidential and sensitive files from unauthorized access.

1.2 Background and Motivation

The word Steganography is derived from two Greek words- ‘stegos’ meaning ‘to cover’ and ‘grayfia’, meaning ‘writing’, thus translating to ‘covered writing’, or ‘hidden writing’. Steganography is a method of hiding secret data, by embedding it into an audio, video, image or text file. It is one of the methods employed to protect secret or sensitive data from malicious attacks. Image Steganography – As the name suggests, Image Steganography refers to the process of hiding data within an image file. The image selected for this purpose is called the cover-image and the image obtained after steganography is called the stego-image As the holy grail of computer vision research is to tell a story from a single image or a sequence of images, object detection and recognition has been studied for more than four decades. Significant efforts have been paid to develop representation schemes and algorithms aiming at recognizing generic objects in images taken under different imaging conditions (e.g., viewpoint, illumination, and occlusion).

1.3 Problem Statement and Objectives

Generally, the study aimed to develop a system Image Steganography of Multiple File Types with Encryption and Compression Algorithms. Specifically, this study aimed to: accept secret file to be hidden; compress secret file using an Algorithm; encrypt and decrypt secret file using Data Encryption Standard Algorithm; embed and extract data file in stego-image using LSB Algorithm; and evaluate the system using aspecific standards: functionality, usability, reliability, portability, efficiency

1.4 Scope of the Project

In the present system, there are so many problems while transferring or sharing a file of large size. These files cannot be sent over a network if they are above certain size limits. Even if the size of the file is small but if the internet connection speed is slow, then too it is difficult to send them. These limitations create a lot of problems when we need to share an urgent file and we just can't due to the size limit of the file. Therefore, there is a need to build a system that can compress and decompress the files easily so they can be shared efficiently without any constraint.

1.5 Team Organization

Aadesh garg
Along with doing preliminary investigation and understanding the limitations of current system, I studied about the topic and its scope and surveyed various research papers related to the object detection and the technology that is to be used.

Amisha Linjhara
I also worked on the implementation of tensorflow framework and the working of counting of objects in the project also helped in coding part.

Amisha Prajapati

Worked on creating database for storing results in database. Documentation is also a part of the work done by me in this project.

I investigated and found the right technology and studied in deep about it. For the implementation of the project

Ananya Shrivastava
I collected the object data and trained the model for it. Implementation logic for the project objective and coding of internal functionalities is also done by me.

Also, worked on Back end design for storing results in database for maintaining logs.

1.6 Report Structure

The project ***Real-time Object detection and Recognition*** is primarily concerned with the **Image processing in real-time** and whole project report is categorized into five chapters.
Chapter 1: Introduction- introduces the background of the problem followed by rationale for the project undertaken. The chapter describes the objectives, scope and applications of the project. Further, the chapter gives the details of team

members and their contribution in development of project which is then subsequently ended with report outline.

Chapter 2: Review of Literature- explores the work done in the area of Project undertaken and discusses the limitations of existing system and highlights the issues and challenges of project area. The chapter finally ends up with the requirement identification for present project work based on findings drawn from reviewed literature and end user interactions.

Chapter 3: Proposed System - starts with the project proposal based on requirement identified, followed by benefits of the project. The chapter also illustrate software engineering paradigm used along with different design representation. The chapter also includes block diagram and details of major modules of the project. Chapter also gives insights of different type of feasibility study carried out for the project undertaken. Later it gives details of the different deployment requirements for the developed project.

Chapter 4: Implementation - includes the details of different Technology/ Techniques/ Tools/ Programming Languages used in developing the Project. The chapter also includes the different user interface designed in project along with their functionality. Further it discuss the experiment results along with testing of the project. The chapter ends with evaluation of project on different parameters like accuracy and efficiency.

Chapter 5: Conclusion - Concludes with objective wise analysis of results and limitation of present work which is then followed by suggestions and recommendations for further improvement.

Chapter 2 .Review of Literature

Review of Literature

Compression is the process of encoding information using fewer bits than the original representation of the data. Compression of data is useful as it reduces the resources required to store the data and transmit it over the network. When we talk about data transmission, it is called source encoding. Before transmitting the data, encoding is done for security purposes. The reverse process of compression is called decompression which is used to get the original data from the compressed data.

2.1 Preliminary Investigation

2.1.1 Current System

In the present system, there are so many problems while transferring or sharing a file of large size. These files cannot be sent over a network if they are above certain size limits. Even if the size of the file is small but if the internet connection speed is slow, then too it is difficult to send them. These limitations create a lot of problems when we need to share an urgent file and we just can't due to the size limit of the file. Therefore, there is a need to build a system that can compress and decompress the files easily so they can be shared efficiently without any constraint.

2.2 Limitations of Current System

- The intention of file compression is to significantly reduce the space to store a file and transmit it
- File compression also increases data transfer speed. It is obvious that the longer file would take more time to be transferred and there is a maximum possibility of interruption between the file transfer process

- There is also a chance of the file being corrupted and the user gets a file which is of no use
- Compression also reduces the disk space required on the internet servers along with the time taken by the Internet servers to find the file stored on the hard drive
- File Compression also provides file security by hiding information. This is useful when we do not want information to be available to the public. This can be done by compressing a file that can't be decompressed by a commonly used computer software.

2.3 Requirement Identification and Analysis for Project

Significant work has been done in the field of DATA COMPRESSION DECOMPRESSION ; however, it is not easy to achieve desired results. The review of literature leads to draw certain major findings which are as under :
The project "Data Compression and Decompression" System is totally built on Java technology and provides an interface to the users so that they can easily store and transfer large files. This java project also provides encoding techniques in compression so as to ensure the security of the data. Using this project, the users can compress the data according to the requirement at any point and for any number of times.

The layout of the project is built using the swing and AWT packages of Java using their predefined classes, interfaces, and methods. The io(Input Output) and util packages of Java provide the predefined classes to compress and decompress the files directly by passing the filename and location.

Technology and IDE used to run the project:

- Java 1.8 or above
- Eclipse or Netbeans or IntelliJ

Algorithms used to compress and decompress

We have used 4 algorithms for compression and decompression in this project. They are:

- LZW

- GZIP
- HUFFMAN
- RUNLENGTH

1. LZW Algorithm:

The LZW algorithm stands for Lempel-Ziv-Welch. These three were the inventors of these algorithms. It is a compression algorithm that compresses a file into a smaller one using a table-based lookup. This algorithm is mainly used to compress GIF files and optionally to compress and PDF and TIFF files. The files compressed using this algorithm are saved with .lzw extension.

2. GZIP Algorithm:

The GZIP stands for the GNU Zip algorithm and is used to compress the files into a gzip file format. This file format must not be confused with the ZIP archive file format as the gzip works on a single file. The GZIP is based on the Deflate algorithm.

3. HUFFMAN Algorithm

Huffman coding algorithm is a data compression algorithm that works by creating a binary tree of nodes. All nodes contain the character itself and priority queue is applied to build a binary tree.

4. RUNLENGTH Algorithm

Run length compression or Run Length Encoding(RLE) works by reducing the physical size of a repeating string of characters. This repeating string is called ‘run’ and the size is called ‘length’. This type of compression is mainly used for file formats such as TIFF, BMP, and PCX.

2.3.1 Conclusion

This chapter reviews the literature surveys that have been done during the research work. The related work that has been proposed by many researchers has been discussed. The research papers related to object detection and recognition of objects from 1985 to 2015 have been shown which discussed about different methods and algorithm to identify objects.

Chapter 3 .Proposed System

Proposed System

3.1 The Proposal

With the rapid development of mobile communication technology, the demand for signal transmission bandwidth is increasing. Technologies such as carrier aggregation will also increase the amount of data transmitted between the baseband processing unit and the remote radio unit. This puts forward higher requirements for optical fiber transmission capabilities. In the current communication system based on optical fiber transmission, the application of optical modules with a transmission rate of 10Gb/s has become popular. In 5G applications, 40G or 100G optical modules are even required to meet the ever-increasing application requirements. In order to reduce and control the consumption of optical fiber resources, it is necessary to develop a data compression method to reduce data transmission pressure and reduce costs. This paper proposes an IQ data compression and decompression algorithm based onFPGA hardware and according to the characteristics of programmable logic devices. As a result, the antenna system improves the efficiency of radio frequency power amplifiers, and reduces operation and maintenance costs.

3.2 Benefits of the Proposed System

Data compression minimizes the space that files occupy on a hard drive and reduces the time needed to transfer or download them. This reduction of space and time can result in significant cost savings. For example, organizations that store large amounts of data, such as corporations and healthcare providers, can save on data storage expenses, as compression allows them to store more files with less capacity. Also, since compressed files take less time to transmit across the internet, such organizations have less need for investing in costly bandwidth upgrades.

For certain other organizations, compression allows them to provide optimal service at the highest convenience. For instance, telecommunications providers handle enormous amounts of audio and video data. Compression allows them to provide service to a large number of customers with minimal compromise in auditory or visual quality.

3.3 Block Diagram

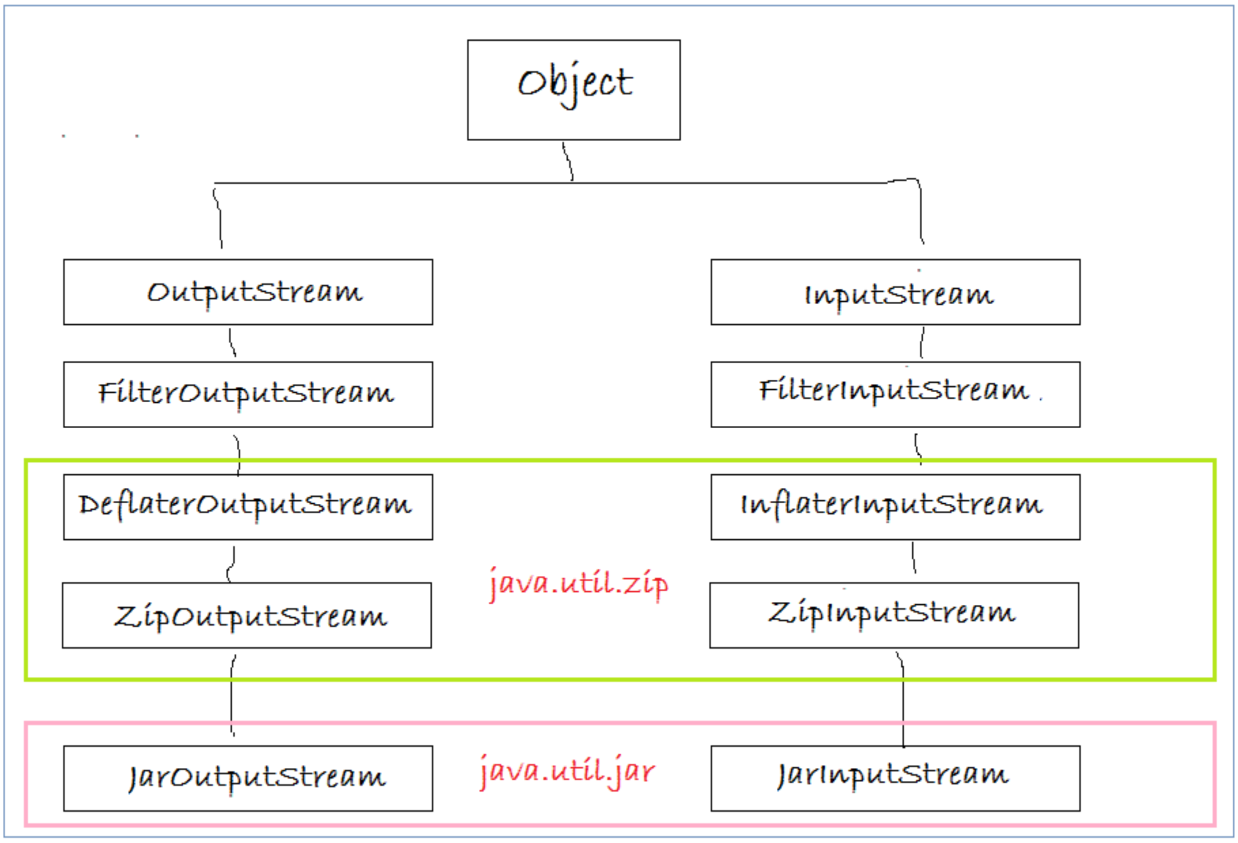


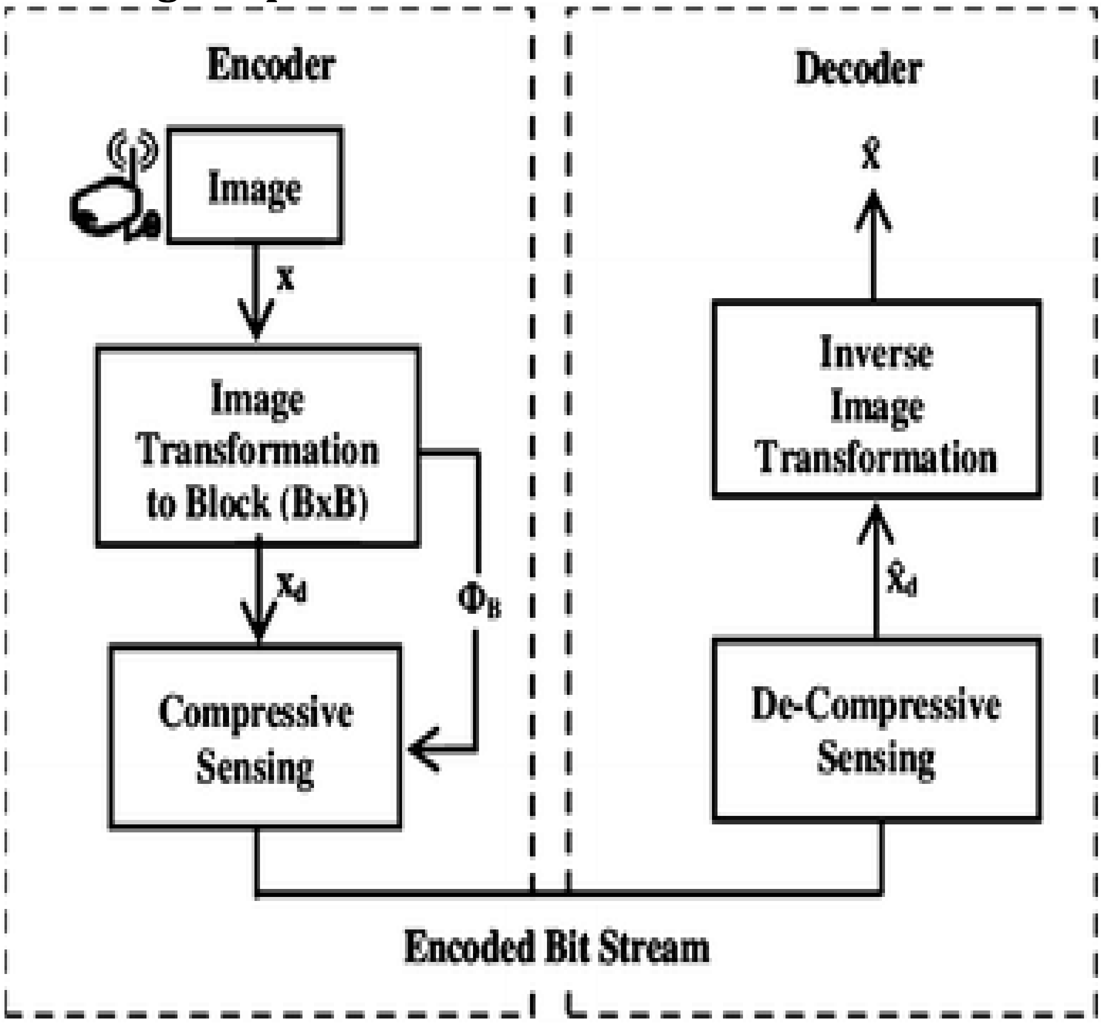
Figure 3-1 : Block Diagram

3.4 Feasibility Study

A feasibility study is an analysis of how successfully a system can be implemented, accounting for factors that affect it such as economic, technical and operational factors to determine its potential positive and negative outcomes before investing a considerable amount of time and money into it.
For any real-time system, there is a need to process images from the video. For this, the kind of framework used must be the one that is capable of extracting those objects from the images easily and accurately in real-time. It is a framework designed by Google for efficiently dealing with deep learning and concepts like neural networks , making the system technically feasible.

For making the system technically feasible, there is a requirement of GPU built system with high processor for better performance.

3.5 Design Representation



3.6 Deployment Requirements

We have used Swing and AWT to create the layout of the project.

In **CGZipCompressor** folder, there are two java classes:

CGZipEncoder.java

In this class, we have imported the java.io package for using the file operations and util.zip.GZIPOutputStream class . This class implements a stream filter for writing compressed data in the GZIP file format. We also define a loadFile() method overloaded with different number of parameters. The boolean encodeFile() method returns true on the successful encoding of the file.

CGZipDecoder.java

In this class, we have imported the java.io package for using the file operations and util.zip.GZIPInputStream class . This class implements a stream filter for reading compressed data in the GZIP file format. We also define a loadFile() method overloaded with different number of parameters. The boolean decodeFile() method returns true on the successful decoding of the file.

In CHuffmanCompressor folder, there are four java classes and an interface

CHuffmanEncoder.java

The class CHuffmanEncoder implements the HuffmanSignature interface which is defined in the HuffmanSignature.java in another file. It has three constructors that call the loadFile() method with different arguments. The encodeFie() method returns true if the file is encoded successfully. There are methods to build and get the Huffman codes.

CHuffmanDecoder.java

This class also implements the HuffmanSignature interface. It is used to decode the Hiffman codes to get the original file. There are three constructors in which we have called the loadFile() method with different arguments. The decodeFile() method is of boolean type and returns true on successful decoding of the file. There is an int findCodeword() method that finds the codeword for the Huffman code.

HuffmanNode.java

This class stores all the nodes of the Huffman tree.

CPriorityQueue.java

The CPriorityQueue class is used to find the Huffman codes for every character of the file. There are several functions like enqueue(), dequeue(), isFull(), totalNodes() in this class.
Interface:

HuffmanSignature.java

This is an interface that declares final variables with a value.

In CLZWCompressor, there are two classes and one interface

CLZWEncoder.java

This class implements the LZWInterface and creates instances using different constructors. There is a boolean method called encodeFile() that returns true on successful encoding of the file.

CLZWDecoder.java

This class implements the LZWInterface and creates new instances by defining constructors with different parameters. We also use hashtable as this algorithm is table-based and stores values in a table. There is a boolean method called decodeFile() that returns true on the successful decoding of the file.

LZWInterface.java

This interface defines the final variables.

In **CRLECompressor**, there are two classes and one interface
CRLEEncoder

This class implements the RLEInterface. The method encodeFile() creates two instances, one of input files and another of output file, and then copies the single byte from input source file to output source file. In the output, it adds the extension provided by the user at the runtime(I.e., rle extension).

CRLEDecoder

This class also implements RLEInterface. In this also, there are two file sources, one is of RLE signature and destination file. Then we read file character-wise until we get an escape character. Finally, we get the original file at a particular location with the file size.

RLEInterface

In this interface, we have defined some final variables with some initial values.

In the **FileBitIO folder**, there are 3 classes and one Form file
CFileBitReader.java

This class imports the java.io package to use the classes and methods related to input File. This class mainly deals with reading from the input file. There is a loadFile() method that is used to load the file. There is a getBit() method that converts the file into the form of bits. The getByte() method returns the file in the form of bytes or 8-bit sequence. There is eof method that checks whether the file has reached to end or not. The closeFile() method closes the file object. We have used these methods in other classes directly to perform the file operations.

CFileBitWriter.java

This class also uses the java.io package and deals with writing to the file. There are methods like putBit(), putBits(), and putString() to write into the file in different formats-either bitwise or bytewise. It also has closeFile() method to close the file after writing.

GfhWorkingDlg.form

This is an xml file that defines all the components and dimensions of the components.

In wingph folder, there are four java files and one form file:

Main.java

In the Main class, we call the invokeLater() method of java.awt.EventQueue class and inside the run() method we create the object of Gph class and call its setVisible() method and pass true in it.

GfhWorkingDlg.java

This class extends the predefined class javax.swing.JDialog and implements the interface GphGuiConstants.There are two parameterized constructors. We create a thread to compress or decompress. This class basically performs the actions after clicking the button and shows the progress bar of processing.

GphGuiConstants.java

This interface has two String arrays – one for algorithms names and other for extensions. We have also defined some final variables with an initial value.

Gph.form

This is an xml file that defines all the properties of the components with their dimensions.

Gph.java

This class imports the java.awt.Color class and creates the layout of the application interface. We call the initComponents() methods in which we have created the whole frame using different components of javax.swing package.

Chapter 4 .Implementation

Implementation

For the problem of counting the number of students and vehicles entering the college campus manually, the system is designed in such a way so as to automate the process by placing a camera at the entrance gate so that students, bikes and cars getting inside the college campus can be identified and counted.

4.1 Technique Used

Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is intended to let application developers write once, and run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java was first released in 1995 and is widely used for developing applications for desktop, web, and mobile devices. Java is known for its simplicity, robustness, and security features, making it a popular choice for enterprise-level applications.

The word Steganography is derived from two Greek words- ‘stegos’ meaning ‘to cover’ and ‘grayfia’, meaning ‘writing’, thus translating to ‘covered writing’, or ‘hidden writing’. Steganography is a method of hiding secret data, by embedding it into an audio, video, image or text file. It is one of the methods employed to protect secret or sensitive data from malicious attacks. Image Steganography – As the name suggests, Image Steganography refers to the process of hiding data within an image file. The image selected for this purpose is called the cover-image and the image obtained after steganography is called the stego-image. Dependencies: The major requirement of the resources for designing and developing the proposed smart map is as follows.

- HTML
- CSS
- JavaScript
- Java

HTML: HTML stands for Hyper Text Markup Language. It is the standard markup language for creating web pages. It describes the structure of a web page. HTML consists of a series of elements. HTML elements tell the browser how to display the content. CSS: stands for Cascading Style Sheets. It describes how HTML elements are to be displayed on the screen, paper, or in other media. It can control the layout of multiple web pages all at once and saves a lot of work. External stylesheets are stored as CSS files. JavaScript: is a scripting language, primarily used on the Web. It is used to enhance HTML pages and is commonly found embedded in HTML code. JavaScript is an interpreted language. Thus, it doesn't need to be compiled. JavaScript renders web pages in an interactive and dynamic fashion. Java: is a one of the most popular programming languages for many years. Java is Object Oriented. However, it is not considered as pure object-oriented as it provides support for primitive data type

4.3 Language Used

Java language is used in the system due to the following Characterstics :

Simple :

Java is a simple and minimalistic language. Reading a good java program feels almost like reading English (but very strict English!). This pseudo-code nature of java is one of its greatest strengths. It allows you to concentrate on the solution to the problem rather than the syntax i.e. the language itself.

Free and Open Source :

Java is an example of a FLOSS (Free/Libre and Open Source Software). In simple terms, you can freely distribute copies of this software, read the software's source code, make changes to it, use pieces of it in new free programs, and that you know you can do these things. FLOSS is based on the concept of a community which shares knowledge. This is one of the reasons why java is so good - it has been created and improved by a community who just want to see a better java

Object Oriented :

Java supports procedure-oriented programming as well as object-oriented programming. In procedure-oriented languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In object-oriented languages, the program is built around objects which combine data and functionality. Python has a very powerful but simple way of doing object-oriented programming, especially, when compared to languages like C++ or Java.

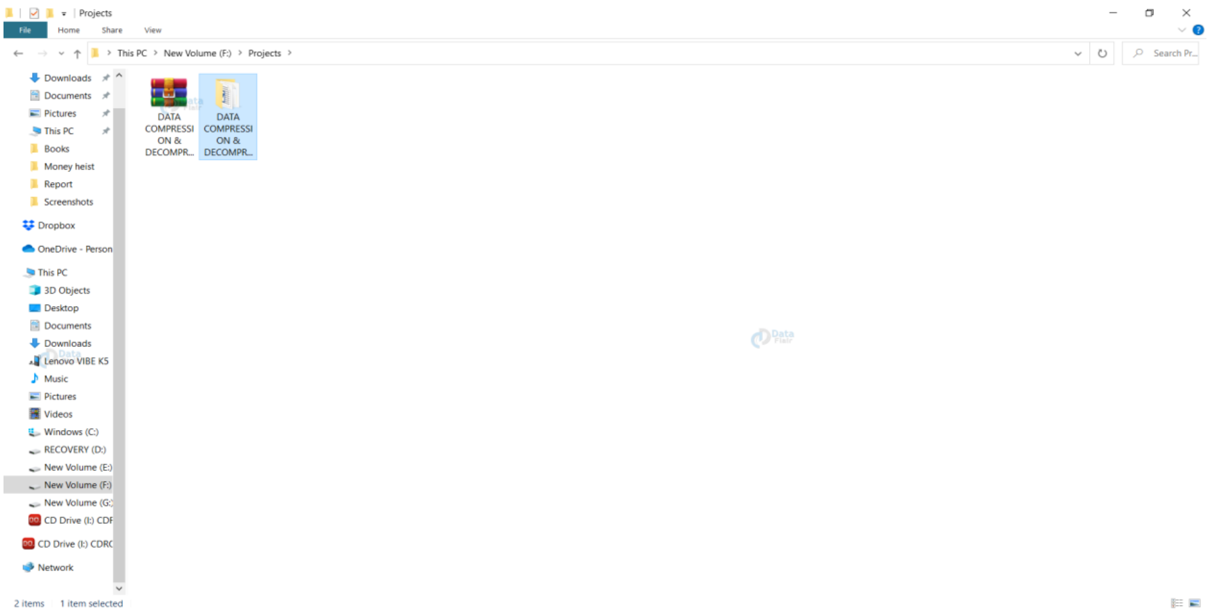
Extensive Libraries :

The Standard Library is huge indeed. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, ftp, email, XML.

XML-RPC, HTML, WAV files, cryptography, GUI(graphical user interfaces) using Tk, and also other system-dependent stuff. Remember, all this is always available wherever Python is installed. This is called the "batteries included" philosophy of Python.

4.5 Execution of a project

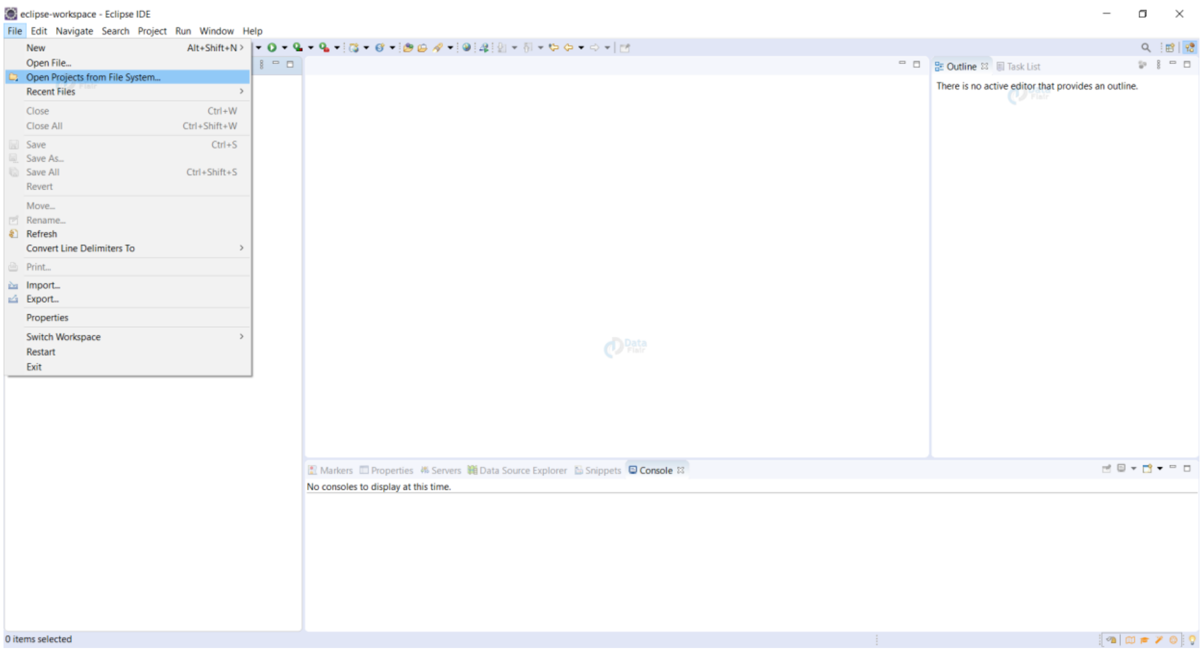
Now, we have a .rar file. Please save this folder into a project folder. Now right click on this file and select “Extract here” option:



Now, we need to import this project in our Eclipse IDE. Let’s start:

1. Open Eclipse

2. Click the file menu button and choose open project from file system option:

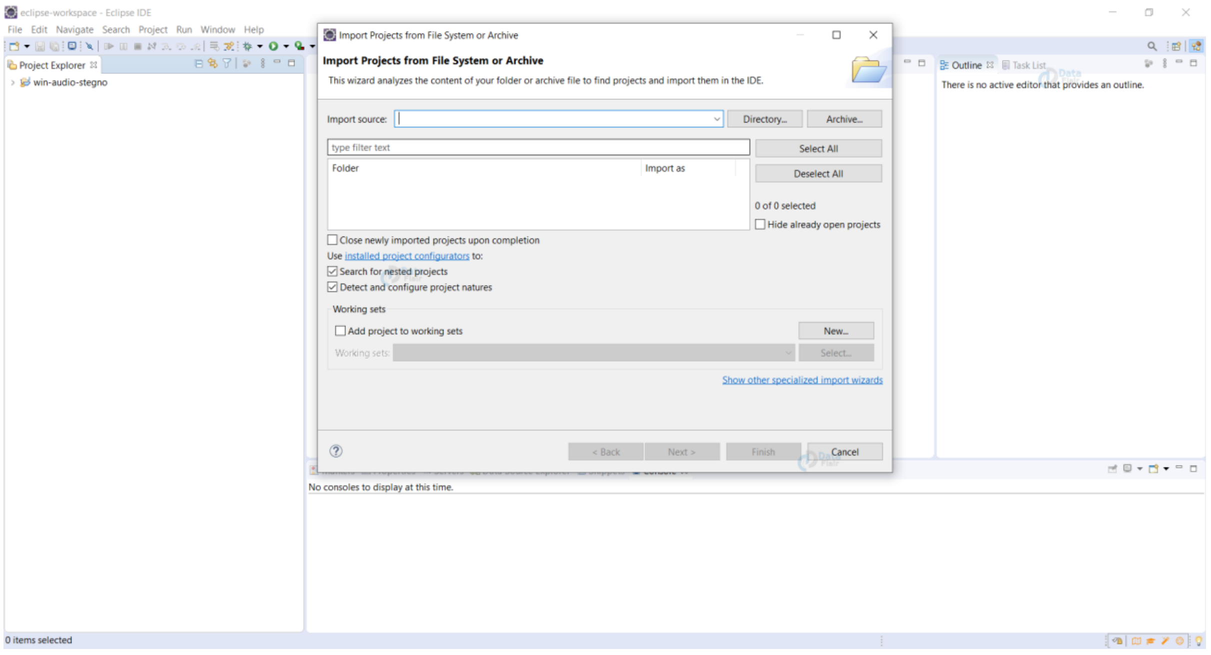


There is no active editor that provides an outline.

No consoles to display at this time.

0 items selected

3. Click the Directory Button:



Import Projects from File System or Archive

This wizard analyzes the content of your folder or archive file to find projects and import them in the IDE.

0 of 0 selected

☐ Hide already open projects

☐ Close newly imported projects upon completion

Use installed project configurators to:

☑ Search for nested projects

☑ Detect and configure project natures

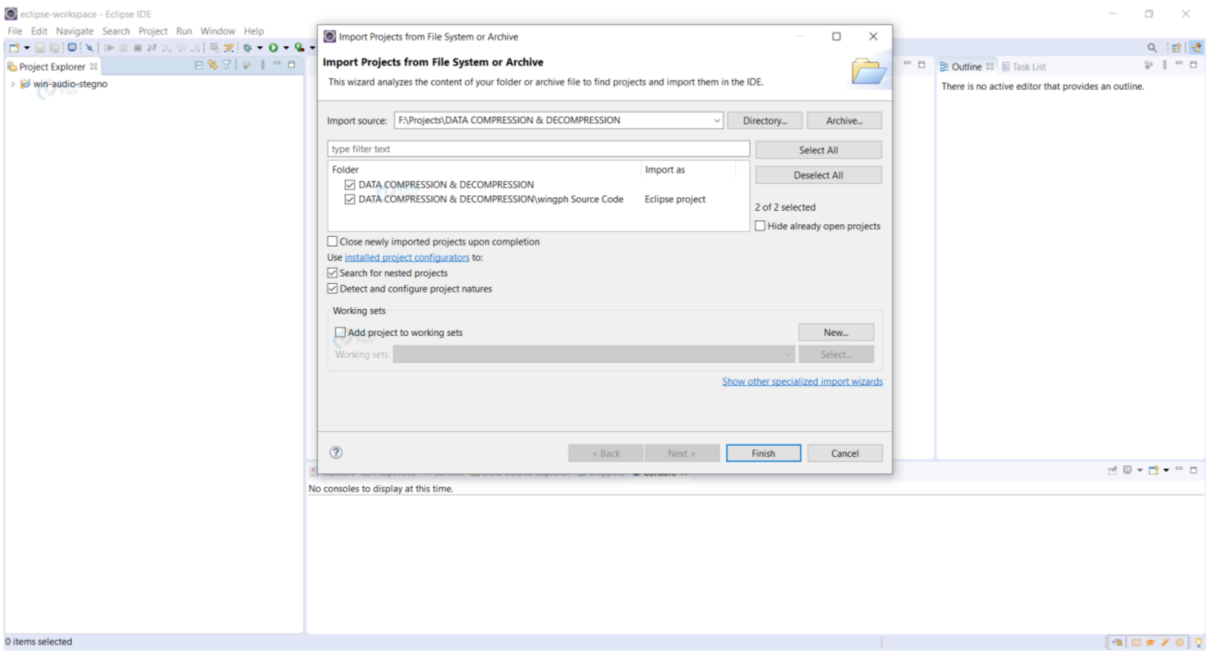
☐ Add project to working sets

There is no active editor that provides an outline.

No consoles to display at this time.

0 items selected

4. Now, select the folder that we extracted previously.


Import Projects from File System or Archive

This wizard analyzes the content of your folder or archive file to find projects and import them in the IDE.

Import source: F:\Projects\DATA COMPRESSION & DECOMPRESSION

Folder	Import as
☑ DATA COMPRESSION & DECOMPRESSION	
☑ DATA COMPRESSION & DECOMPRESSION\wingph Source Code	Eclipse project

2 of 2 selected

☐ Hide already open projects

☐ Close newly imported projects upon completion

Use installed project configurators to:

☑ Search for nested projects

☑ Detect and configure project natures

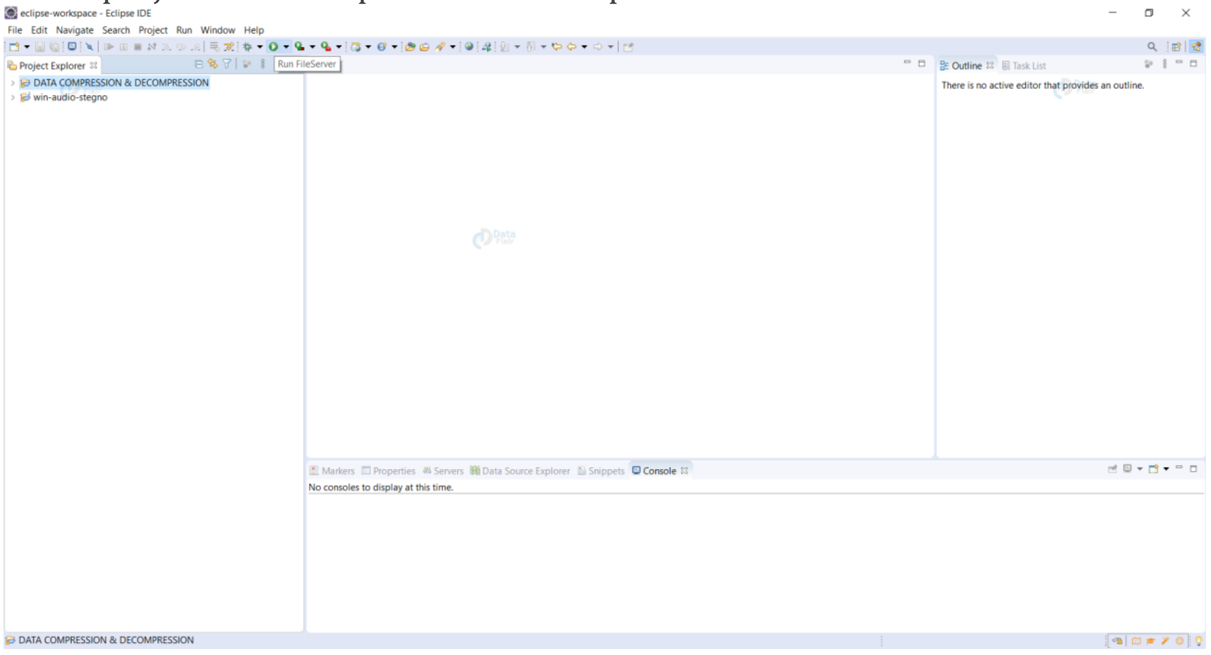
Working sets

☐ Add project to working sets

No consoles to display at this time.

0 items selected

5. The project is now imported in the eclipse:


Project Explorer: DATA COMPRESSION & DECOMPRESSION, win-audio-stegno

There is no active editor that provides an outline.

No consoles to display at this time.

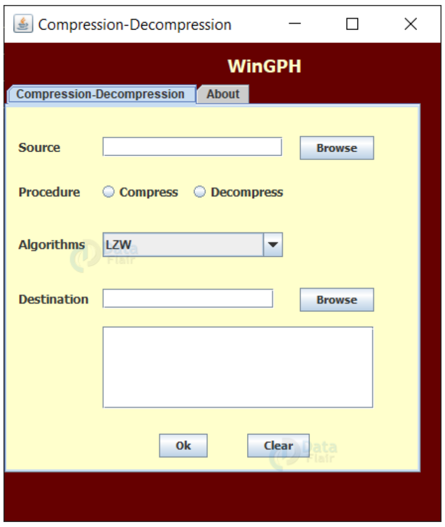
DATA COMPRESSION & DECOMPRESSION

Executing the project

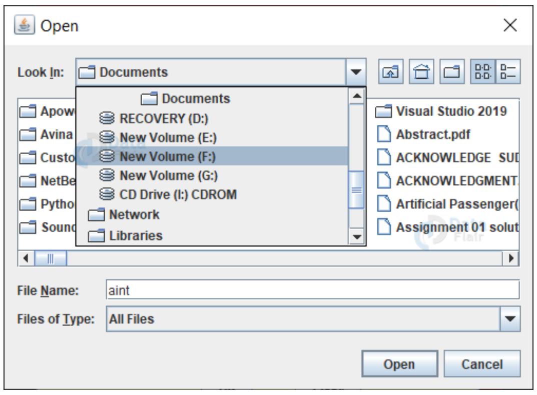
To run this project in your system you need to import the wingph Source Code folder, then run the Main.java file, then the output window will appear.

Steps to run the project

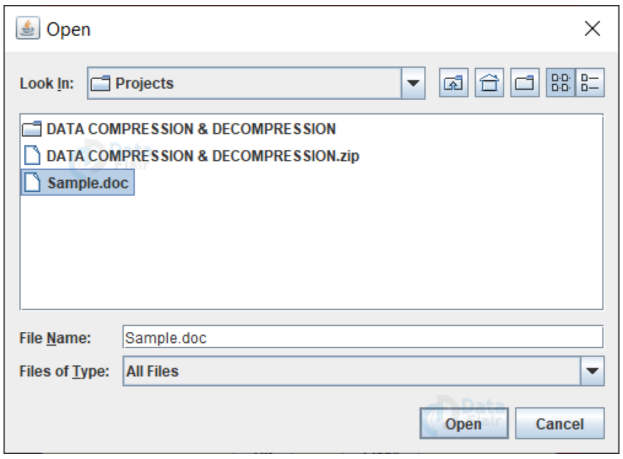
1. When you run the Main.java file, you will see following window:



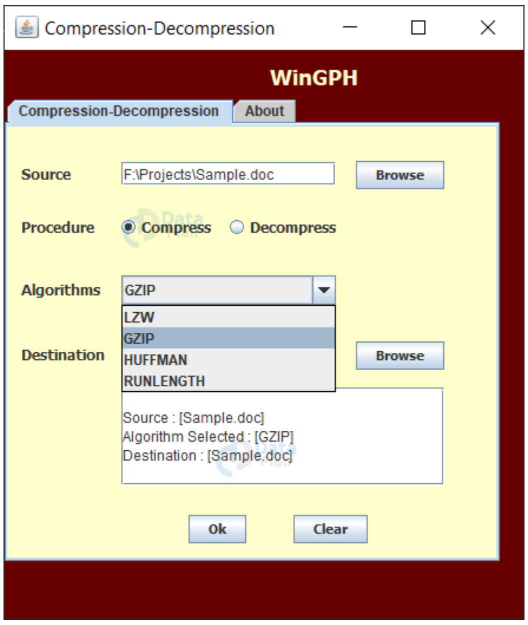
2. Click the Browse button:



3.Select the file you want to compress



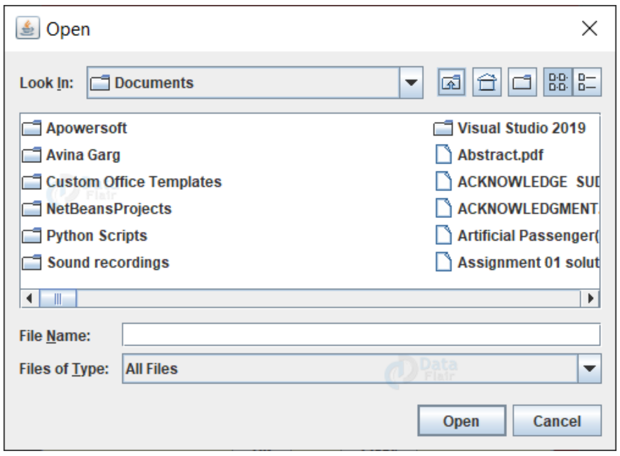
4. Now, select the compression algorithm:



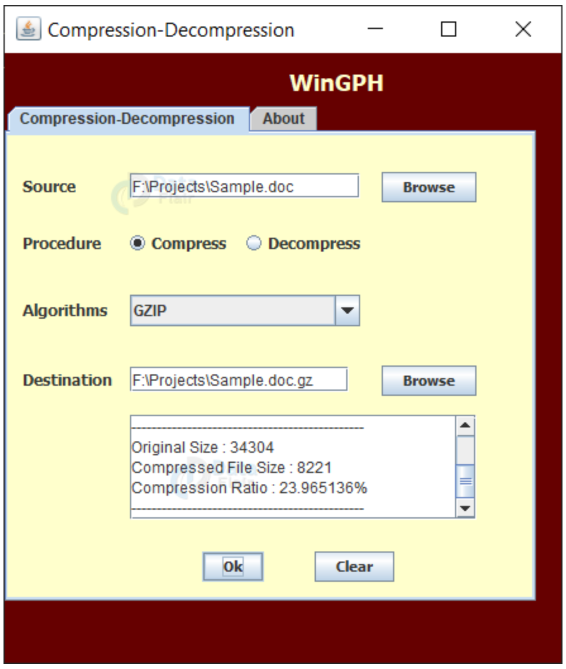
Source: F:\Projects\Sample.doc
Procedure: (●) Compress ( ) Decompress
Algorithms: GZIP
LZW
GZIP
HUFFMAN
RUNLENGTH
Destination

```
Source : [Sample.doc]
Algorithm Selected : [GZIP]
Destination : [Sample.doc]
```

5. Choose the destination folder:



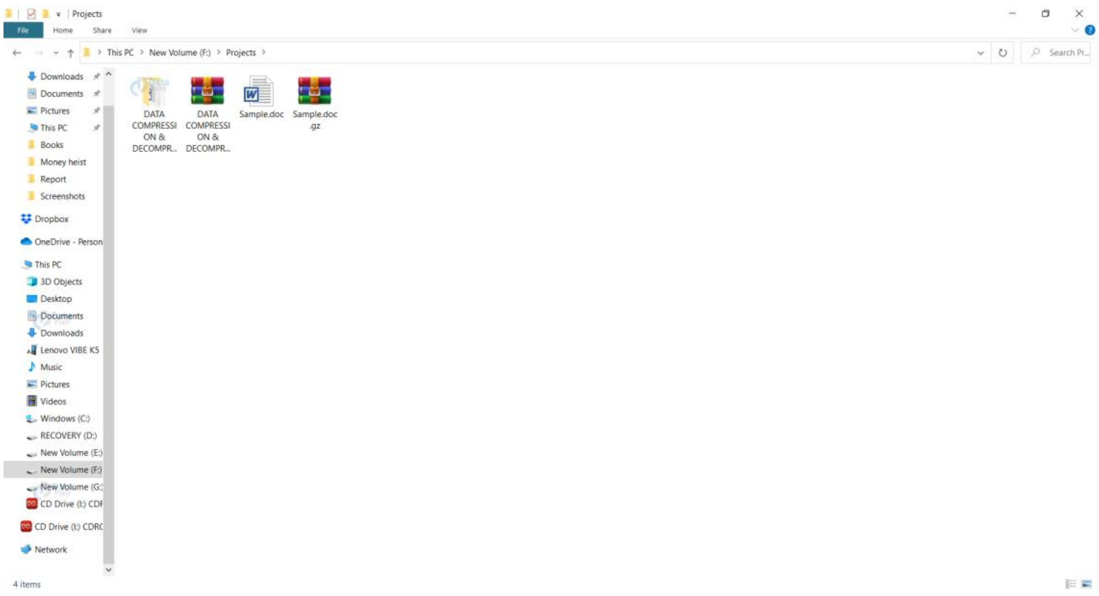
6. Click the Ok button:



Source: F:\Projects\Sample.doc
Procedure: (●) Compress ( ) Decompress
Algorithms: GZIP
Destination: F:\Projects\Sample.doc.gz

```
--------------------------------------------------
Original Size : 34304
Compressed File Size : 8221
Compression Ratio : 23.965136%
--------------------------------------------------
```

Now, we can see that in our folder, the zip file has been saved:



Chapter 5.Conclusion

Conclusion

5.1 Conclusion

From this project, you can easily reduce the large size file into a file of smaller size using compression and send them easily over the network. We can also decompress the compressed file to get the original file and the receiver's end. There are four algorithms used in this process. And various interfaces and classes of Swing and AWT packages are used to create the layout.
The work done manually can now be completely replaced by this automated system and it can reduce all the extra efforts of maintain the records.

5.2 Limitations of the Work

- The working of this project would be a little slow because framework like need high-processing hardware and GPU(graphical processing unit) systems but we are using CPU only.
- The models that we are using for identifying the objects are pre-trained models. So, if we want to train our own model, it takes a lot of time and processing.
- In the system, scanning of each frame is one per second but still it needs improvement. If the objects move too fast, it may not detect them.

5.3 Suggestion and Recommendations for Future Work

- The Model would be trained for detecting more number of objects.
- SNS service will be integrated in this project for alert notification when an unwanted object is detected.
- Currently, the bounding box technique is used which is bounding the targeted object within a rectangle. In future , Segmentation will be used.

Bibliography

[1] H. Fujiyoshi, A. J. Lipton and R. S. Patil, "Moving Target Classification and Tracking from Real-time Video," in *Fourth IEEE workshop*, 1998.

[2] D. Comaniciu, R. Vishwanathan and P. Meer, "Real-Time Tracking of Non-Rigid Objects using Mean Shift," in *IEEE*, 2000.

[3] K. Levi and Y. Weiss, "Learning Object Detection from a Small Number of Examples: the Importance of Good Features," in *IEEE Computer Society Conference*, 2004.

[4] V. Lepetit, P. Lagger and P. Fua, "Randomized Trees for Real-Time Keypoint Recognition," in *IEEE Computer Society Conference*, 2005.

[5] T. Yang, S. Z. Li and J. Li, "Real-time Multiple Objects Tracking with Occlusion Handling in Dynamic Scenes," in *IEEE Computer Society Conference*, 2005.

[6] J. Zhou and J. Hoang, "Real Time Robust Human Detection and Tracking System," in *IEEE Computer Society Conference*, 2005.

[7] C. P. Papageorgiou, M. Oren and T. Poggio, "A General Framework for Object Detection," in *Center for Biological and Computational Learning Artificial Intelligence Laboratory*, Cambridge, 2005.

[8] R. D. Charette and F. Nashashibi, "Real TimeVisual Traffic Lights Recognition Based on Spot Light Detection andAdaptive Traffic Lights Template," in *IEEE*, 2009.

[9] S. K. Nayar, S. A. Nene and M. Hiroshi, "Real-Time 100 Object Recognition System," in *IEEE International Conference*, 1996.

[10] A. Adam, E. Rivlin, S. Ilan and D. Reinitz, "Robust Real-Time Unusual Event Detection Using Multiple Fixed-Location Monitors," in *IEEE*, 2008.

[11] C. Bahlmann, Y. Zhu, R. Vishwanathan, M. Pelkoffer and T. Koehler, "A System for Traffic Sign Detection, Tracking, and Recognition Using Color, Shape, and Motion Information," in *IEEE*, 2005.

[12] M. Betke, E. Haritaoglu and L. S. Davis, "Real-time multiple vehicle detection and tracking from a moving vehicle," *Machine Vision and Applications,* p. 12, 2000.

[13] Q. Chen, N. D. Georganas and E. M. Petriu, "Real-time Vision-based Hand Gesture Recognition Using Haar-like Features," in *IEEE Conference*, 2007.

[14] R. Cutler and L. S. Davis, "Robust Real-Time Periodic Motion Detection, Analysis, and Applications," *IEEE transactions on Pattern Analysis and Machine Intelligence,* 2000.

[15] J. Heikkila and O. Silven, "A real-time system for monitoring of cyclists and pedestrians," *Image and Vision Computing,* 2004.

[16] D. Maturana and S. Scherer, "VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition," in *IEEE International Conference*, 2015.

[17] C. Papageorgiou and T. Poggio, "A Trainable System for Object Detection," *Internation Journal of Computer Vision,* 2000.

[18] J. Redmon, S. Divvala, R. Girshik and A. Farhadi, "You Only Look Once: Unified" in *IEEE conference on Computer Vision and Pattern Rocgnition*, 2016.

[19] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman and A. Blake, "Real-Time Human Pose Recognition in Parts

from Single Depth Images," *Computer Vision and Pattern Recognition,* 2011.

[20] K. P. A. Menon, "Management of Agriculture and Rural Development Programme: Problems and Prospects, India," *J Extension Education,* vol. 21, no. 1, pp. 94-98, 1985.

[21] S. Hinterstoisse, V. Lepetit, S. Ilic, P. Fua and N. Nassir, "Dominant Orientation Templates for compression decompression of Texture-Less Object," in *IEEE Conference*, 2010.

Source Code

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

          public class SIM {

              public static void main(String[] args) throws IOException {
                  if(args.length == 0){
                      System.out.println("Please Enter correct number of Arguments!!");
                  }else {
                      String selector = args[0];
                      if(selector.equals("1"))
                          Compressor();
                      else if(selector.equals("2"))
                          Decompressor();
                      else
                          System.out.println("Please provide the correct Input!!\n" +
                                  "(1 for Compression OR 2 for  Decompression)");
                  }
              }

              static final String compressInFile = "original.txt";
              static final String compressOutFile = "cout.txt";
              static final String decompressInFile = "compressed.txt";
              static final String decompressOutFile = "dout.txt";
              static  StringBuilder compressedCode = new StringBuilder();
              static StringBuilder decompressedCode = new StringBuilder();

          //---------------------------------------------COMPRESSION ALGORITHM----------
          -----------------------------------------

              // Method for the compression algorithm
              private static void Compressor() throws IOException {
                  LinkedHashMap<String, Integer> tempDictMap = new LinkedHashMap<>();
                  LinkedList<String> inputList = new LinkedList<>();
                  LinkedHashMap<String, String> dictionary;

                  //Function to read the original uncompressed input file for
          compression
                  ReadOriginalFile(inputList,tempDictMap);
                  //Get the final Dictionary
                  dictionary = DictionaryFunction(tempDictMap);
                  //Implementation of Compression Strategies
                  ImplementCompressionStrategies(inputList, dictionary);
                  //Function to write the compressed file into the output file
```

```
        WriteCompressedOutputToFile(dictionary);
    }

    // Function to read the original uncompressed input file for compression
    private static void ReadOriginalFile(LinkedList<String> inputList,
LinkedHashMap<String, Integer> tempDictMap) throws IOException {

        String lineReader;
        BufferedReader bufferReader = new BufferedReader(new
FileReader(compressInFile));

        // Take the input from the input file and Store it in the map along
with its frequency of occurrence
        while ((lineReader = bufferReader.readLine()) != null) {
            int count;
            inputList.add(lineReader);
            if(tempDictMap.containsKey(lineReader)){
                count = tempDictMap.get(lineReader);
                tempDictMap.replace(lineReader, count, count+1);
            }
            else {
                count = 1;
                tempDictMap.put(lineReader, count);
            }
        }

        //Close the buffered reader
        bufferReader.close();
    }

    }

    //Function to create a Map for Dictionary with its values mapped to the
respective dictionary index
    private static LinkedHashMap<String, String>
DictionaryFunction(LinkedHashMap<String, Integer> tempDictMap) {
        LinkedHashMap<String, String> dictionary = new LinkedHashMap<>();
        String dictKey, dictValue;
        for(int dictIndex = 0; dictIndex < 8; dictIndex++){
            dictKey = DictionaryMaker(tempDictMap);
            //Convert the dictionary index to a 3-bit binary number
            dictValue = String.format("%3s",
Integer.toBinaryString(dictIndex)).replaceAll(" ", "0");
            dictionary.put(dictKey, dictValue);
            tempDictMap.remove(dictKey);
        }
        return dictionary;
    }

    // Function to implement all the compression strategies
    private static void ImplementCompressionStrategies(LinkedList<String>
inputList, LinkedHashMap<String, String> dictionary) {

        int rleCounter = -1;
        String previousCode = "";
        String rleString;

        for (String currentCode : inputList) {
            //Run Length Encoding Strategy
            Mismatch oneMismatchOneBit = findMismatchLocations(dictionary,
currentCode, 1);
            Mismatch twoMismatchOneBit = findMismatchLocations(dictionary,
currentCode, 2);
            String bitmaskDetailsFinder = BitmaskEncoder(dictionary,
currentCode);
            List<Integer> mismatchLocations;
            String dictionaryIndex;
            if (currentCode.equals(previousCode))
                rleCounter += 1;
            else {
                if (rleCounter > -1) {
                    rleString = String.format("%2s",
Integer.toBinaryString(rleCounter)).replaceAll(" ", "0");
                    compressedCode.append("000").append(rleString);
                    rleCounter = -1;
                }
                // If Code has Direct Mapping With the Dictionary Strategy
                if (dictionary.containsKey(currentCode))

compressedCode.append("101").append(dictionary.get(currentCode));
                    // If Code contains Mismatch of 1-bit Strategy
                else if (oneMismatchOneBit != null) {
                    int mismatchLoc;
                    String binaryLocation;
                    mismatchLocations = oneMismatchOneBit.mismatchLocations;
                    mismatchLoc = mismatchLocations.get(0);
                    binaryLocation = String.format("%5s",
Integer.toBinaryString(mismatchLoc)).replaceAll(" ", "0");
                    dictionaryIndex = oneMismatchOneBit.dictionaryIndex;

compressedCode.append("010").append(binaryLocation).append(dictionaryIndex);
                }
                // If Code contains mismatch of two consecutive bits
                else if (twoMismatchOneBit != null &&
(twoMismatchOneBit.mismatchLocations.get(1) -
twoMismatchOneBit.mismatchLocations.get(0)) == 1) {
                    int mismatchLoc;
                    String binaryLocation;
                    mismatchLocations = twoMismatchOneBit.mismatchLocations;
                    mismatchLoc = mismatchLocations.get(0);
                    binaryLocation = String.format("%5s",
Integer.toBinaryString(mismatchLoc)).replaceAll(" ", "0");
```

```
                    dictionaryIndex = twoMismatchOneBit.dictionaryIndex;

compressedCode.append("011").append(binaryLocation).append(dictionaryIndex);
                }
                // If code can be compressed using bitmask compression
                else if (bitmaskDetailsFinder.length() > 7) {
                    String bitmask = bitmaskDetailsFinder.substring(0, 4);
                    int bitmaskLocation;
                    String dictIndex, binBitmaskLoc;
                    if (bitmaskDetailsFinder.length() < 9) {
                        bitmaskLocation =
Integer.parseInt(String.valueOf(bitmaskDetailsFinder.charAt(4)));
                        dictIndex = bitmaskDetailsFinder.substring(5);
                    } else {
                        bitmaskLocation =
Integer.parseInt(bitmaskDetailsFinder.substring(4, 6));
                        dictIndex = bitmaskDetailsFinder.substring(6);
                    }
                    binBitmaskLoc = String.format("%5s",
Integer.toBinaryString(bitmaskLocation)).replaceAll(" ", "0");

compressedCode.append("001").append(binBitmaskLoc).append(bitmask).append(dict
Index);
                }
                // If code contains two mismatches of one bits
                else if (twoMismatchOneBit != null) {
                    int mismatchLoc1, mismatchLoc2;
                    String binaryLocation1, binaryLocation2;
                    mismatchLocations = twoMismatchOneBit.mismatchLocations;
                    mismatchLoc1 = mismatchLocations.get(0);
                    mismatchLoc2 = mismatchLocations.get(1);
                    binaryLocation1 = String.format("%5s",
Integer.toBinaryString(mismatchLoc1)).replaceAll(" ", "0");
                    binaryLocation2 = String.format("%5s",
Integer.toBinaryString(mismatchLoc2)).replaceAll(" ", "0");
                    dictionaryIndex = twoMismatchOneBit.dictionaryIndex;

compressedCode.append("100").append(binaryLocation1).append(binaryLocation2).a
ppend(dictionaryIndex);
                }
                // If Code cannot be compressed
                else
                    compressedCode.append("110").append(currentCode);
            }
            previousCode = currentCode;
        }
    }

    //Separate class for finding all the mismatches
    static class Mismatch{
        List<Integer> mismatchLocations;
        int numOfMismatch;
        String dictionaryIndex;

        Mismatch(List<Integer> misLoc, int numBitMismatch, String dictIndex){
            mismatchLocations = misLoc;
            numOfMismatch = numBitMismatch;
            dictionaryIndex = dictIndex;
        }
    }

    //Function to find the number of mismatches, location of the mismatches
and the bit that is mismatched
    //Along with the concerned Dictionary Index
    private static Mismatch findMismatchLocations(LinkedHashMap<String,
String> dictionary, String binaryCode, int NumBitMismatches) {
        List<Integer> mismatchLocations = new ArrayList<>();
        String dictionaryIndex = "";
        int numOfMismatches = 0;

        //Find the mismatch locations with respect to each dictionary entry
        for (Map.Entry<String, String> dictionaryEntry :
dictionary.entrySet()) {
            for(int index = 0; index < binaryCode.length(); index++){
                if(dictionaryEntry.getKey().charAt(index) !=
binaryCode.charAt(index)){
                    mismatchLocations.add(index);
                }
            }
            numOfMismatches = mismatchLocations.size();
            dictionaryIndex = dictionaryEntry.getValue();

            //If the number of mismatches equals the required mismatches,
break the loop
            if(numOfMismatches == NumBitMismatches)
                break;
            mismatchLocations = new ArrayList<>();
            numOfMismatches = 0;
        }

        //If the required number of mismatches were found with respect to any
of the dictionary entry
        //Return the mismatch locations, their values and the dictionary index
        if(numOfMismatches == NumBitMismatches)
            return new Mismatch(mismatchLocations, numOfMismatches,
dictionaryIndex);
        else
            return null;
    }

    //Function to create the BitMask for bitmask encoding
    private static String BitmaskEncoder(LinkedHashMap<String, String>
dictionary, String currentCode) {
        StringBuilder stringBuilder = new StringBuilder();

        // If the code contains two bit mismatches that are not consecutive
        if(twoBitMismatch != null){
            if (twoBitMismatch.mismatchLocations.get(1) -
twoBitMismatch.mismatchLocations.get(0) < 4 &&
                    twoBitMismatch.mismatchLocations.get(1) -
twoBitMismatch.mismatchLocations.get(0) > 1) {

                // If the mismatch pattern  is "MNMN" where M = mismatch and N
= No mismatch
                if (twoBitMismatch.mismatchLocations.get(1) -
twoBitMismatch.mismatchLocations.get(0) == 2) {
                    stringBuilder.append("1010");

stringBuilder.append(twoBitMismatch.mismatchLocations.get(0));
                    stringBuilder.append(twoBitMismatch.dictionaryIndex);
                }
                // If the mismatch pattern  is "MNNM" where M = mismatch and N
= No mismatch
                else {
                    stringBuilder.append("1001");

stringBuilder.append(twoBitMismatch.mismatchLocations.get(0));
                    stringBuilder.append(twoBitMismatch.dictionaryIndex);
                }
            }
        }
        //If the code contains three bit mismatches
        else if(threeBitMismatch != null){
            if (threeBitMismatch.mismatchLocations.get(2) -
threeBitMismatch.mismatchLocations.get(0) < 4) {
                // If the mismatch pattern  is "MMMN" where M = mismatch and N
= No mismatch
                if(threeBitMismatch.mismatchLocations.get(2) -
threeBitMismatch.mismatchLocations.get(0) == 2){
                    stringBuilder.append("1110");

stringBuilder.append(threeBitMismatch.mismatchLocations.get(0));
                    stringBuilder.append(threeBitMismatch.dictionaryIndex);
                }
                // If the mismatch pattern  is "MMNM" where M = mismatch and N
= No mismatch
                else if(threeBitMismatch.mismatchLocations.get(2) -
threeBitMismatch.mismatchLocations.get(0) == 3) {
                    stringBuilder.append("1101");
        }
        // If the code contains four bit mismatches
        else if(fourBitMismatch != null){
            if (fourBitMismatch.mismatchLocations.get(3) -
fourBitMismatch.mismatchLocations.get(0) < 4) {
                stringBuilder.append("1111");

stringBuilder.append(fourBitMismatch.mismatchLocations.get(0));
                stringBuilder.append(fourBitMismatch.dictionaryIndex);
            }
        }
        //Return the string that contains the bitmask, mismatch location,
dictionary index
        return stringBuilder.toString();
    }

    //Code to write the output of the compressed code to cout.txt file
    private static void WriteCompressedOutputToFile(LinkedHashMap<String,
String> dictionary) throws IOException {
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(compressOutFile));
        int offset = 0;
        int length = compressedCode.length();
        //Write the compressed code in batches of 32 characters per line
        while(length >= 32) {
            bufferedWriter.write(compressedCode.toString(), offset, 32);
            bufferedWriter.write("\n");
            offset += 32;
            length = compressedCode.substring(offset).length();
        }


//--------------------------------------------DECOMPRESSION ALGORITHM---------
-----------------------------------------

    //Method for the decompression algorithm
    private static void Decompressor() throws IOException {
        String lineReader, dictValue;
        int dictIndex = 0, offset = 0;

        BufferedReader bufferReader = new BufferedReader(new
FileReader(decompressInFile));
        StringBuilder compressedCode = new StringBuilder();
        LinkedHashMap<String,String> Dictionary = new LinkedHashMap<>();

        //Read the entries of the dictionary and store it in a dictionary that
is a linked hashMap
        while ((lineReader = bufferReader.readLine()) != null){
            dictValue = String.format("%3s",
Integer.toBinaryString(dictIndex)).replaceAll(" ", "0");
            Dictionary.put(dictValue,lineReader);
            dictIndex++;
        }

        //Decode the compressed code by breaking it into bits and pieces
        while(offset != compressedCode.length())
            offset = DecodingStrategySelector(compressedCode, Dictionary,
offset);

        //Write the decompressed Output to File
        WriteDecompressedOutputToFile();
    }

    private static int DecodingStrategySelector(StringBuilder compressedCode,
LinkedHashMap<String, String> Dictionary, int offset) {
        switch (compressedCode.substring(offset, offset + 3)) {
            //The case when the code is compressed using RLE
            case "000":
                RLEDecoding(compressedCode, offset);
                offset += 5;
                break;
            //The case when the code is compressed using Bitmask compression
            case "001":
                BitMaskDecoding(compressedCode, offset, Dictionary);
                offset += 15;
                break;
            //The case when the code has one bit mismatch with a dictionary
entry
            case "010":
                OneBitMismatchDecoding(compressedCode, offset, Dictionary);
                offset += 11;
                break;
            //The case when the code has two bits mismatch with a dictionary
entry
            case "011":
                TwoBitMismatchDecoding(compressedCode, offset, Dictionary);
                offset += 11;
                break;
            //The case when the code has two one bit mismatches with a
dictionary entry
            case "100":
                Two_OneBitMismatchDecoding(compressedCode, offset,
Dictionary);
                offset += 16;
                break;
            //The case when the code has a direct mapping with the dictionary
            case "101":
                DirectDictionaryMappingDecoding(compressedCode, offset,
Dictionary);
                offset += 6;
                break;
            //The case when the code is uncompressed
            case "110":
                OriginalBinaryDecoding(compressedCode, offset);
                offset += 35;
                break;
            //WHen the last bits are padded with 1's
            case "111":
                offset = compressedCode.length();
                break;
        }
        return offset;
    }

    // Decoding strategy when the code was compressed using RLE
    private static void RLEDecoding(StringBuilder compressedCode, int offset)
{
        String compCode = compressedCode.substring(offset+3, offset+5);
        int rleCount = Integer.parseInt(compCode,2) + 1;

        //fetch the previous code
        String previousCode =
decompressedCode.substring(decompressedCode.length() - 33);
        //Add as many lines of previous code as the RLE count

decompressedCode.append(String.valueOf(previousCode).repeat(Math.max(0,
rleCount)));
    }

    // Decoding strategy when the code was compressed using RLE
    private static void BitMaskDecoding(StringBuilder compressedCode, int
offset, LinkedHashMap<String, String> dictionary) {
        String bitmaskLocation, bitMask, dictIndex, dictionaryEntry;

        //fetch the details of the bitmask
        bitmaskLocation = compressedCode.substring(offset + 3, offset + 8);
        int bitmaskLoc = Integer.parseInt(bitmaskLocation, 2);
        bitMask = compressedCode.substring(offset + 8, offset + 12);

        dictIndex = compressedCode.substring(offset + 12, offset + 15);
        dictionaryEntry = dictionary.get(dictIndex);

        //Correct  the mismatch according to the bitmask
        for(int bitPosition = 0; bitPosition < 4; bitPosition ++){
            if(bitMask.charAt(bitPosition) == '1')
                dictionaryEntry = mismatchCorrector(dictionaryEntry,
bitmaskLoc + bitPosition);
        }

        decompressedCode.append(dictionaryEntry).append("\n");
    }

    //Decoding strategy when there is only one bit mismatch with a dictionary
entry
    private static void OneBitMismatchDecoding(StringBuilder compressedCode,
int offset, LinkedHashMap<String, String> dictionary) {
        String dictionaryEntry, mismatchLocation, dictionaryIndex;
        dictionaryEntry = mismatchCorrector(dictionaryEntry, mismatchLoc);

        decompressedCode.append(dictionaryEntry).append("\n");
    }


  ////Decoding strategy when there are two 1-bit mismatches with a dictionary
entry


        //Correct the two consecutive bits of mismatches
        dictionaryEntry = mismatchCorrector(dictionaryEntry, mismatchLoc1);
        dictionaryEntry = mismatchCorrector(dictionaryEntry, mismatchLoc2);

        decompressedCode.append(dictionaryEntry).append("\n");
    }

    //Function to correct the mismatch in the code with the dictionary entry
at the given location
    private static String mismatchCorrector(String dictionaryEntry, int
mismatchLoc) {
        if(dictionaryEntry.charAt(mismatchLoc) == '0')
            dictionaryEntry = dictionaryEntry.substring(0, mismatchLoc) + '1'
                    + dictionaryEntry.substring(mismatchLoc + 1);
        else
            dictionaryEntry = dictionaryEntry.substring(0, mismatchLoc) + '0'
                    + dictionaryEntry.substring(mismatchLoc + 1);

        return dictionaryEntry;
    }

    //Decoding strategy when there no mismatch with a dictionary entry
    private static void DirectDictionaryMappingDecoding(StringBuilder
compressedCode, int offset, LinkedHashMap<String, String> dictionary) {
        String dictionaryIndex, dictionaryEntry;

        dictionaryIndex = compressedCode.substring(offset + 3, offset + 6);
        dictionaryEntry = dictionary.get(dictionaryIndex);

        decompressedCode.append(dictionaryEntry).append("\n");
    }

    //Decoding strategy when the code was not compressed
    private static void OriginalBinaryDecoding(StringBuilder compressedCode,
int offset) {
        String originalBinary;
        originalBinary = compressedCode.substring(offset + 3, offset + 35);

        decompressedCode.append(originalBinary).append("\n");
    }
    //Function to write the decompressed output to the dout file
    private static void WriteDecompressedOutputToFile() throws IOException {
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(decompressOutFile));
        bufferedWriter.write(decompressedCode.substring(0,
decompressedCode.length() - 1));
        bufferedWriter.close();
    }
}
```