Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Spring, Year: 2023), B.Sc. in CSE (Day)

File Compression Tool (Project Report)

Course Title: ***CSE 206 Algorithms Lab***
Course Code: ***CSE - 206***
Section: ***PC - 213 DA***

Students Details

Name	ID
Md. Shahidul Islam Prodhan	**213902017**
-	-

Submission Date: 21st June, 2023 (Wednesday)

Course Teacher's Name: ***Md. Sultanul Islam Ovi***

[For teachers use only: Don't write anything inside this box]

Lab Project Status

Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

The project aims to develop a file compression tool using data structure algorithms and specifically, the Huffman and Lampel-Ziv-Welch (also known as LZW) encoding algorithm, implemented in Java. The tool will provide efficient compression and decompression capabilities for text file types, reducing storage requirements and facilitating faster file transfers.

1.2 Motivation

In today's data-driven world, the need for efficient file compression tools is increasing. The File Compression Tool aims to address this need by leveraging data structure algorithms and the Huffman encoding/algorithm in a Java implementation as well as LZW algorithm. By compressing files, it will help reduce storage costs, enable quicker data transfers, and enhance overall system performance.

1.3 Problem Definition

1.3.1 Problem Statement / Problem Domain

The project aims to solve the problem of efficient file compression and decompression using Java. The challenges include preserving file integrity, minimizing data loss, and maintaining efficient compression and decompression processes. Existing tools may have limitations in terms of compression ratios and speed, which the proposed project seeks to overcome.

1.3.2 Complex Engineering Problem

By addressing these following question domains, we can identify and understand the complexities involved in developing the File Compression Tool using data structure algorithms, the Huffman encoding and LZW algorithm. Recognizing these complexities will guide the development process and help anticipate challenges and potential solutions throughout the project.

1.4 Design Goals / Objectives

The project has the following design goals and objectives in the Java implementation:

- Implement the Huffman encoding/algorithm in Java as the core compression mechanism, ensuring high compression ratios while maintaining file integrity.
- Design an efficient data structure in Java for storing and manipulating compressed files to optimize compression and decompression processes.
- Develop a user-friendly Java GUI (Graphical User Interface) that allows users to easily compress and decompress files of various types and sizes.
- Optimize the compression and decompression algorithms in Java for speed and efficiency, minimizing processing time.
- Provide support for encryption of compressed files in Java to enhance security.

1.5 Application

The File Compression Tool implemented in Java has various applications in different domains, including:

- Compressing large multimedia files (mainly Text based files on this project; But I will try image format to compress if possible) for storage or sharing purposes, enabling efficient usage of storage resources.
- Facilitating faster backups and transfers of files over networks or cloud storage platforms, saving time and bandwidth.
- Supporting the compression of data in resource-constrained environments, such as embedded systems or mobile devices, to optimize storage utilization.
- Enhancing data management processes in enterprise-level systems by compressing data and reducing storage requirements.

Table 1.1: Summary of the attributes touched by the mentioned projects

Name of the P Attributes	Explain how to address
P1: Depth of knowledge required	It involves in-depth knowledge of data structures, algorithm design, file systems, compression algorithms, and Java programming.
P2: Range of conflicting requirements	This tool faces conflicting requirements that need to by achieving high compression ratios to minimize file size and storage requirements and it must ensure minimal loss of data and maintain file integrity during compression and decompression.
P3: Depth of analysis required	It requires a significant depth of analysis such as - analyzing the characteristics of different file types and sizes to optimize compression strategies. Analyzing the performance of the Huffman encoding/algorithm and fine-tuning its parameters to achieve optimal compression ratios and processing speed are essential.
P4: Familiarity of issues	This tool may pose familiarity challenges due to the complexity of the chosen algorithms and the requirement for expertise in Java programming. Familiarity with data structures, compression algorithms, and the Huffman encoding/algorithm, Java GUI development is necessary for implementing the user-friendly interface.
P5: Extent of applicable codes	The development of the File Compression Tool will involve utilizing existing code libraries and frameworks to support various functionalities.
P6: Extent of stakeholder involvement and conflicting requirements	—-
P7: Interdependence	The development of the tool involves multiple inter-dependencies between different components, for example - The compression and decompression algorithms heavily rely on the data structures designed to efficiently store and manipulate compressed files. Additionally, the user interface interacts with the underlying compression and decompression functionalities.

1.6 Methodology

To achieve the project objectives in the Java implementation, the following steps will be taken:

- Research and understand the principles and implementation of the Huffman encoding/algorithm in the context of Java programming.
- Design and implement efficient Java data structures for storage and manipulation of compressed files.
- Develop the compression and decompression algorithms in Java using the Huffman encoding/algorithm, ensuring data integrity and high compression ratios.
- Create a user-friendly Java GUI for users to interact with the tool, allowing them to select files for compression or decompression.
- Test the Java implementation of the tool using various file types and sizes to ensure accurate compression and decompression operations.
- Optimize the Java code by refining the algorithms and data structures, addressing edge cases, and enhancing error handling.
- Document the project, providing clear instructions on tool usage, Java dependencies, and any additional features.

1.7 Expected Outcome

The expected outcome of the project is a fully functional File Compression Tool implemented in Java that successfully compresses and decompresses files using data structure algorithms and the Huffman encoding/algorithm. The Java implementation will demonstrate efficient compression ratios, fast processing times, and a user-friendly GUI. It will support various file types and sizes and offer the option to encrypt compressed files for added security.

By following this project proposal and implementing the File Compression Tool in Java, we aim to create an effective and efficient tool that addresses the need for file compression using data structure algorithms and the Huffman encoding/algorithm. The Java implementation will allow for broad applications and contribute to improved storage utilization, faster data transfers, and enhanced system performance.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

This chapter focuses on the design, development, and implementation of the file compression project, which utilizes the Huffman encoding and Lempel-Ziv-Welch (LZW) algorithms to compress and decompress text-based files, reducing their file size. The project aims to provide an efficient solution for file compression, optimizing storage utilization and facilitating faster file transfers.

2.2 Project Details

The file compression project is implemented in Java and utilizes two powerful compression algorithms: Huffman encoding and LZW. These algorithms are widely recognized for their effectiveness in achieving high compression ratios while maintaining file integrity. By leveraging these algorithms, the project aims to address the challenges of reducing storage requirements and enhancing system performance.

2.2.1 Design and Architecture

The project begins with the design and architecture phase, where the overall structure of the compression tool is planned. This includes designing the data structures, choosing efficient and correct algorithms, file handling mechanisms, and user interface components. The goal is to create a robust and scale-able architecture that can efficiently handle file compression and decompression tasks.

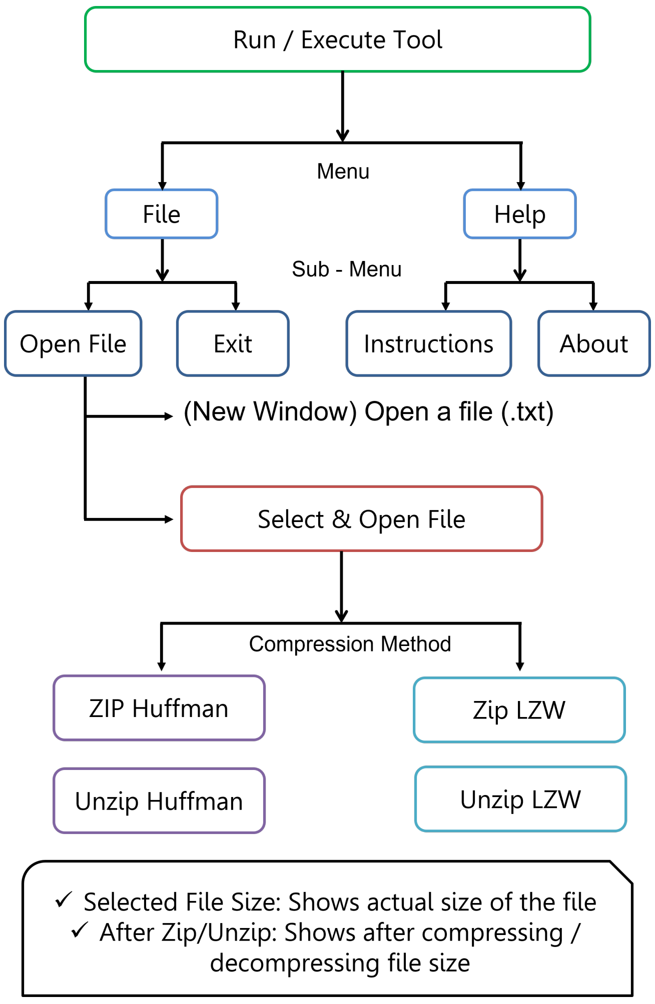

Run / Execute Tool
Menu
File
Help
Sub - Menu
Open File
Exit
Instructions
About
(New Window) Open a file (.txt)
Select & Open File
Compression Method
ZIP Huffman
Zip LZW
Unzip Huffman
Unzip LZW
✓ Selected File Size: Shows actual size of the file
✓ After Zip/Unzip: Shows after compressing / decompressing file size

Figure 2.1: Working Design of this Project

2.3 Implementation

2.3.1 Huffman Encoding Implementation

The first algorithm implemented is Huffman encoding. This algorithm uses a variable-length prefix coding scheme to represent characters based on their frequency of occurrence. The implementation involves building a Huffman tree and generating a Huffman codebook. The codebook is then used to encode the input file, replacing characters with shorter codes for more frequent characters and longer codes for less frequent characters.

2.3.2 Lempel-Ziv-Welch (LZW) Implementation

The second algorithm implemented is Lempel-Ziv-Welch (LZW). LZW is a dictionary-based compression algorithm that dynamically builds a dictionary of frequently occurring phrases or patterns in the input file. It replaces these patterns with shorter codes, thereby reducing the file size. The LZW implementation involves dictionary management, pattern matching, and code generation.

2.3.3 Compression and Decompression Processes

Once the Huffman encoding and LZW algorithms are implemented, the compression and decompression processes are developed. The compression process takes an input text-based file and applies the Huffman encoding followed by the LZW compression. The result is a compressed file with a reduced size. On the other hand, the decompression process reverses the compression steps by performing LZW decompression followed by Huffman decoding, resulting in the original file being restored.

2.3.4 User Interface Development (with screenshots)

To provide a user-friendly experience, a graphical user interface (GUI) is developed. The GUI allows users to select files for compression or decompression and provides feedback on the compression ratio achieved. It also includes options for selecting compression algorithms, specifying output locations, and handling errors or exceptions.

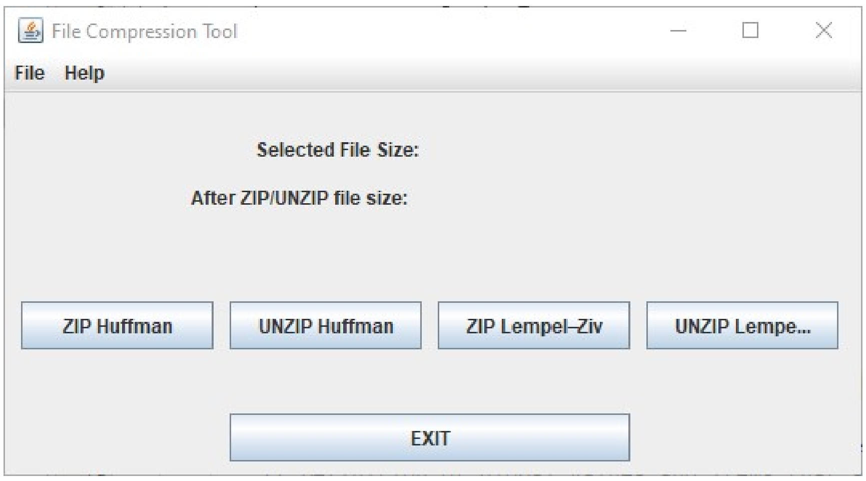


Figure 2.2: Main User Interface of the Tool

2.4 Algorithms/Pseudocode

The algorithms and the programming codes in detail should be included .
Pseudo-codes are also encouraged very much to be included in this chapter for your project.

Huffman Encoding

Zip: The following algorithms and data structures were used - Huffman Coding, Priority Queue, Depth-First Search (DFS)

1. The code is for a file compression tool.

2. It defines a class called Hzipping in the package com.mycompany.filecompressiontool.
3. The code includes imports for various input/output classes and a priority queue.
4. It declares and initializes several static variables and arrays.
5. The TREE class is defined as a nested class within Hzipping.
6. The code contains methods for calculating the frequency of bytes in a file, converting bytes to binary, freeing memory, creating codes, creating tree nodes, encrypting, faking zip, and creating a real zip.
7. The beginHzipping method is the main entry point of the code.

Unzip:

1. Implements the Huffman coding algorithm for file decompression. Uses a priority queue (pq1) to store tree nodes based on their frequencies.
2. The TREE class represents a node in the Huffman tree.
3. Generates binary codes for each byte using the dfs1() function.
4. Reads frequency information from a file and updates the freq1 array.
5. Converts a binary string to an eight-digit binary string using the makeeight() function.
6. Performs decompression by reading a compressed file, decoding it using Huffman codes, and writing the decompressed data to a new file.
7. The to() function converts a Byte value to an int.
8. The initHunzipping() function initializes variables and data structures. The beginHunzipping() function is the entry point for decompression.

Lempel-Ziv-Welch (LZW)

Zip: Lempel-Ziv-Welch (LZW) compression algorithm. HashMap (dictionary) for storing codes and their corresponding values

1. The code implements file compression using the LZW algorithm.
2. It includes functions to convert Byte values to int (btoi) and convert integers to binary strings (fil)
3. Another function, strtobt, converts binary strings to Byte values.
4. The precalc function calculates the dictionary length based on the input file.
5. The Lamzip function performs the compression process:

   (a) It initializes a dictionary with the initial 256 ASCII characters.
   (b) It reads the input file byte by byte and generates compressed codes.
   (c) If a sequence is not present in the dictionary, a new code is created.
   (d) The compressed data is written to the output file.

6. The beginLzipping function is the entry point for compression. It calls the necessary functions to calculate the dictionary length and perform compression.

Unzip: The code primarily implements Lempel-Ziv-Welch (LZW) compression algorithm for file decompression. It utilizes a dictionary to store mappings and performs string decompression based on the dictionary entries.

1. The code is a part of a file compression tool. It includes a class named Lunzipping. The class has several methods and variables.
2. The pre() method initializes an array bttost with binary to string conversions for 8 bits.
3. The btoi() method converts a byte to an integer.
4. The stoi() method converts a binary string to an integer.
5. The Lunzip() method performs the decompression process.
6. It uses a dictionary to store mappings of integers to strings.
7. The method reads a compressed input file, decompresses it, and writes the decompressed data to an output file.
8. The beginLunzipping() method initializes variables and calls the Lunzip() method to start the decompression process.
9. There is a commented-out main() method that can be used to test the decompression process.

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

The simulation environment was developed using Java programming language, specifically utilizing the Java Swing library for creating the graphical user interface (GUI). The environment provides a user-friendly interface for file compression using two algorithms: Huffman encoding and Lempel-Ziv-Welch encoding.

The development environment used for this project includes:

- Java Development Kit (JDK) version 8 or higher
- Integrated Development Environment (IDE) such as Eclipse or IntelliJ IDEA. In this case, I have used NetBeans

The simulation environment supports the compression of text-based file formats, such as .txt files. It allows users to select a file to be compressed, visualize its size before and after compression, and choose between the Huffman and Lempel-Ziv-Welch algorithms for compression and decompression operations.

3.2 Main Interface/Basic Working Process

The simulation environment provides a straightforward and intuitive user experience, allowing users to compress and decompress text-based files. The GUI interface facilitates easy file selection, algorithm selection, and visualization of file sizes before and after compression.

1. Launching the Simulation
2. Opening a File
3. Compression Process
4. Decompression Process
5. Exiting the Simulation

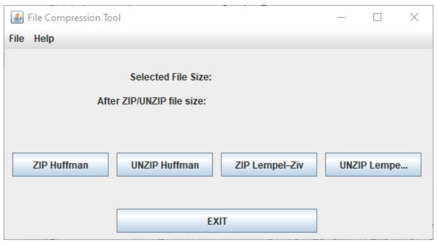
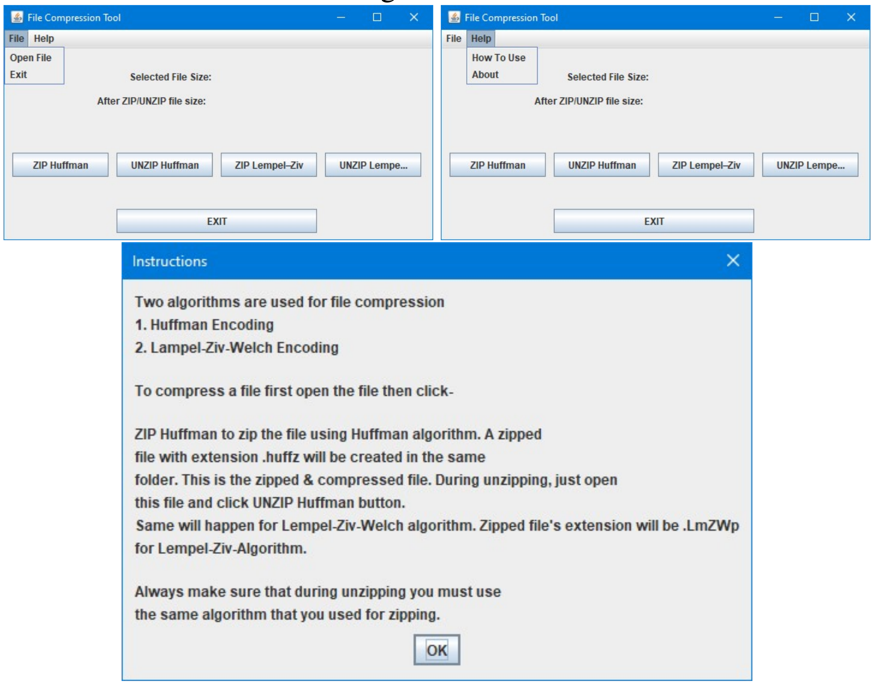


Figure 3.1: 1



Two algorithms are used for file compression
1. Huffman Encoding
2. Lampel-Ziv-Welch Encoding

To compress a file first open the file then click-

ZIP Huffman to zip the file using Huffman algorithm. A zipped
file with extension .huffz will be created in the same
folder. This is the zipped & compressed file. During unzipping, just open
this file and click UNZIP Huffman button.
Same will happen for Lempel-Ziv-Welch algorithm. Zipped file's extension will be .LmZWp
for Lempel-Ziv-Algorithm.

Always make sure that during unzipping you must use
the same algorithm that you used for zipping.

Figure 3.2: Instructions to use this tool

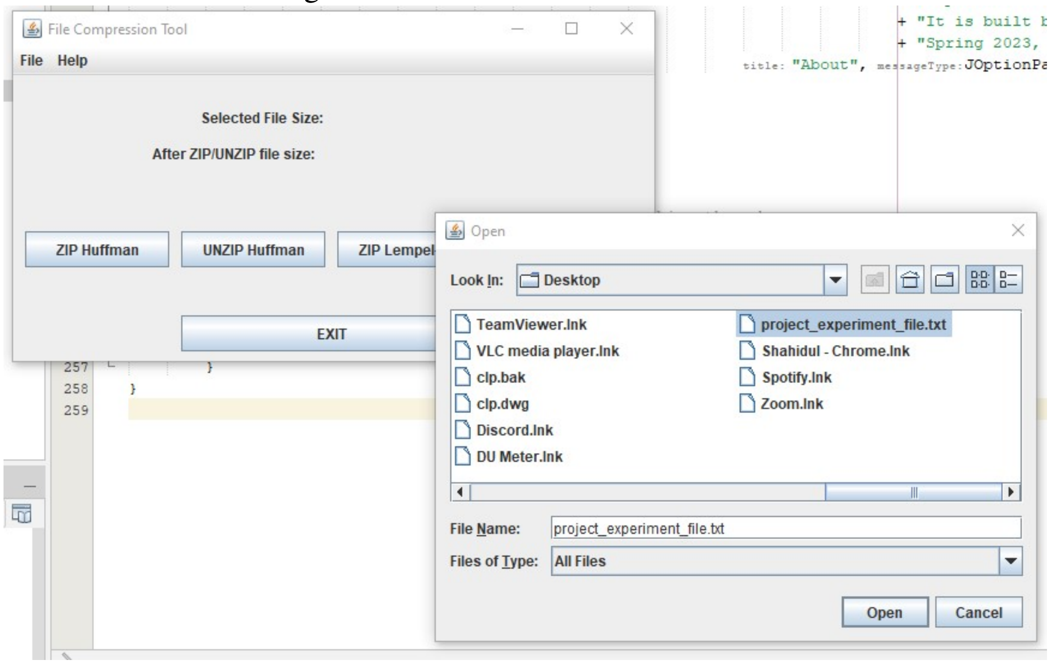


Figure 3.3: 2

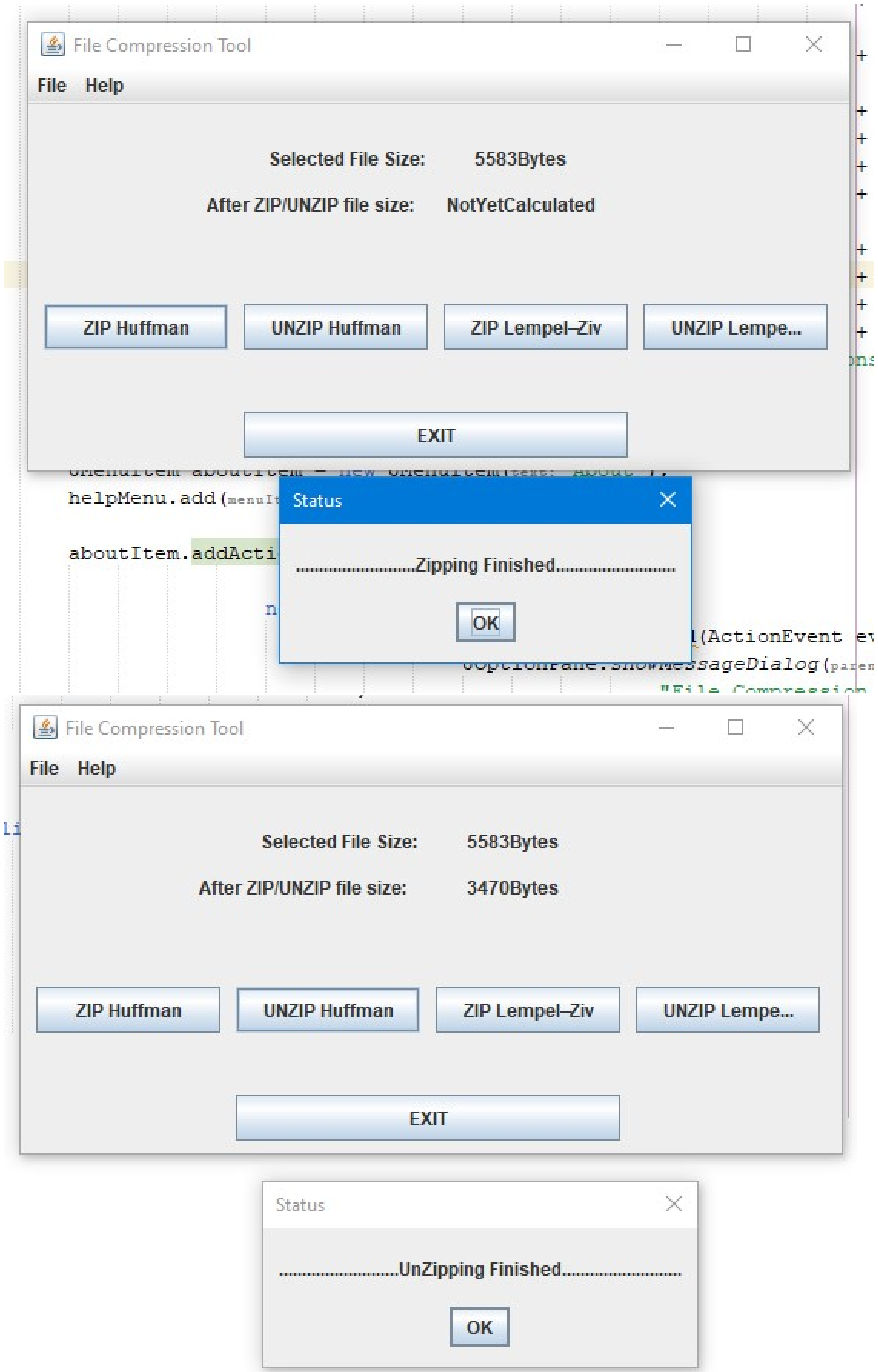
Figure 3.4: Main User Interface of the Tool

Chapter 4

Result Analysis/Testing

4.1 Huffman and Lempel-Ziv-Welch

Here is the working procedure of it:



Selected File Size: 5583Bytes

After ZIP/UNZIP file size: NotYetCalculated

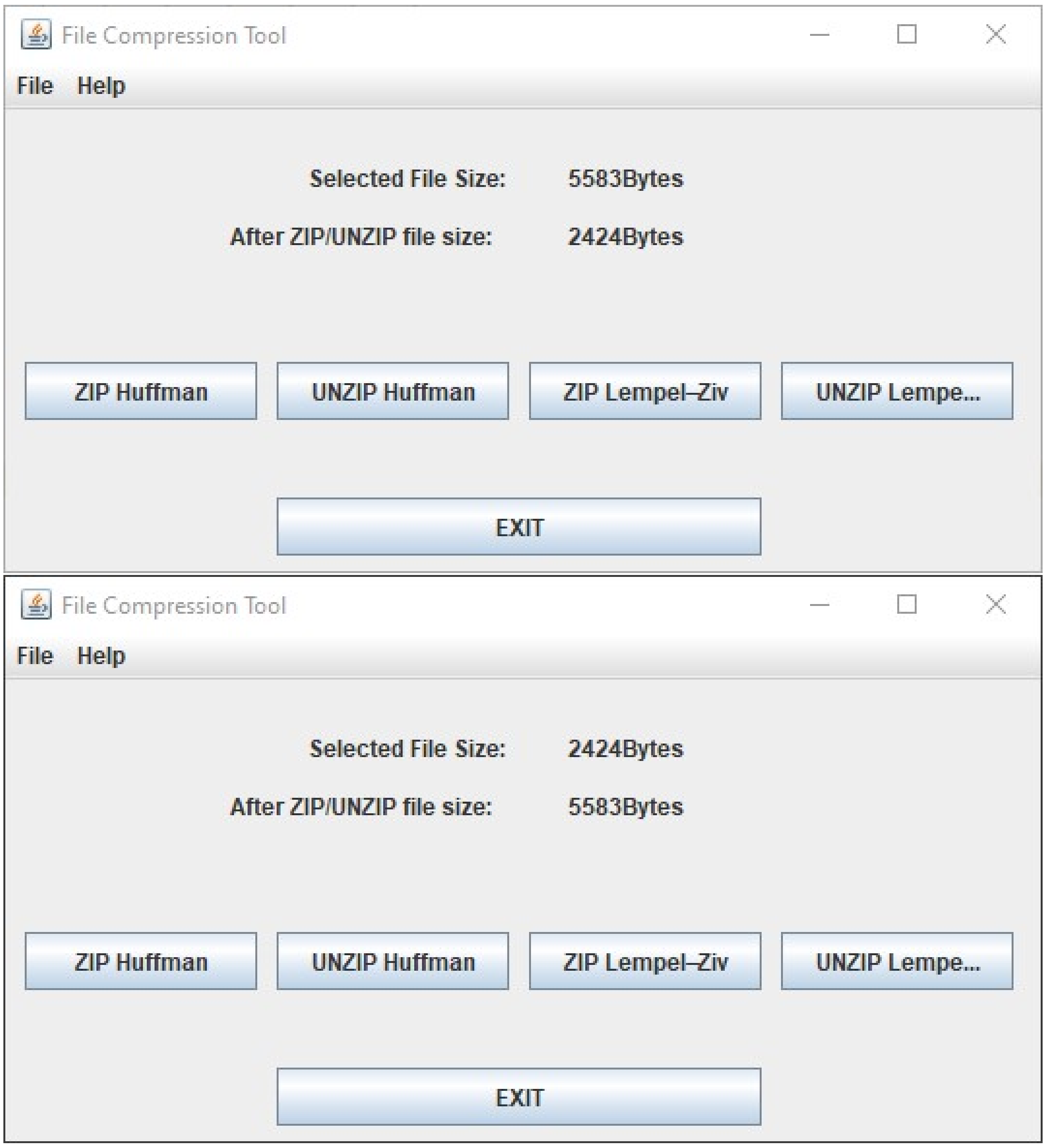
.........................Zipping Finished.........................

Selected File Size: 5583Bytes

After ZIP/UNZIP file size: 3470Bytes

.........................UnZipping Finished.........................

Figure 4.1: Zipping and Unzipping of the file using Huffman



Selected File Size: 5583Bytes

After ZIP/UNZIP file size: 2424Bytes

Selected File Size: 2424Bytes

After ZIP/UNZIP file size: 5583Bytes

Figure 4.2: Zipping and Unzipping of the file using LZW

Chapter 5

Conclusion

5.1 Discussion

The file compression tool developed using Java and incorporating Huffman encoding and Lempel-Ziv-Welch encoding algorithms provides an effective solution for reducing the size of text-based files. The project successfully implemented a user-friendly GUI that allows users to compress and decompress files effortlessly. By applying the chosen algorithm, the tool achieves a significant reduction in file size, enhancing storage efficiency and facilitating faster file transfers.

5.2 Limitations

Despite its effectiveness in compressing text-based files, the tool has certain limitations. It currently supports compression for only text-based files such as .txt files. Other file formats like images, PNG, audio files, or pre-compressed ZIP files are not supported. This limitation arises from the specific implementation of the Huffman and Lempel-Ziv-Welch algorithms, which are optimized for text data. Future versions of the tool could address this limitation by incorporating additional algorithms or techniques to support a wider range of file formats.

5.3 Scope of Future Work

Moving forward, there are several opportunities for improvement and future work on this file compression tool:

1. **Expansion to support various file formats:** Enhancing the tool to handle diverse file formats, such as images, audio, and compressed files, would significantly increase its utility and applicability.

2. **Integration of advanced compression algorithms:** Exploring and integrating more advanced compression algorithms, such as Burrows-Wheeler Transform or

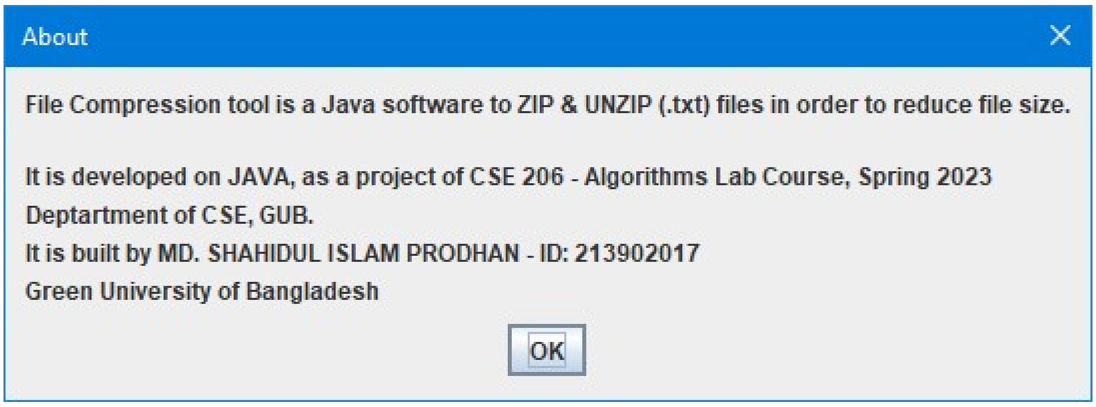
Arithmetic Coding, can potentially yield even better compression ratios and performance.

3. **Multi-threading and performance optimization:** Implementing multi-threading techniques to parallelize the compression and decompression processes could enhance the tool's efficiency, enabling faster processing of large files.

4. **Error handling and robustness:** Strengthening the tool's error handling mechanisms and incorporating appropriate error messages and safeguards would improve the overall robustness and user experience.

5. **Enhanced user interface:** Further enhancing the user interface with additional features, such as progress bars, file preview options, and customization settings, would make the tool more intuitive and user-friendly.

5.4 Project Declaration and Authorship



File Compression tool is a Java software to ZIP & UNZIP (.txt) files in order to reduce file size.

It is developed on JAVA, as a project of CSE 206 - Algorithms Lab Course, Spring 2023
Deptartment of CSE, GUB.
It is built by MD. SHAHIDUL ISLAM PRODHAN - ID: 213902017
Green University of Bangladesh

Figure 5.1: Author of the File Compression Tool

References