A

Project Report

On

File Compression/Decompression Tool

Submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

In

Computer Science and Engineering

By

Surjeet Kumar Verma	**2261562**
Suraj Singh	**2261560**
Chetan Negi	**2261154**
Gaurav Singh Koranga	**2261224**

Under the Guidance of

Mr. Prince Kumar

ASSISTANT / ASSOCIATE PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS

SATTAL ROAD, P.O. BHOWALI,

DISTRICT- NAINITAL-263132

2024-2025

STUDENT'S DECLARATION

We, **Surjeet Kumar Verma, Suraj Singh, Chetan Negi, and Gaurav Singh Koranga** hereby declare the work, which is being presented in the project, entitled '**File Compression/Decompression Tool**' in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of Mr. Prince Kumar.

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

Surjeet Kumar Verma

Suraj Singh

Chetan Negi

Gaurav Singh Koranga

CERTIFICATE

The project report entitled "File Compression/Decompresson Tool" being submitted by Surjeet Kumar Verma S/o Mr. Rakesh Kumar Verma, 2261562, Suraj Singh S/o Mr. Deepak Bisht, 2261560, Chetan Negi S/o Mr.H.S. Negi, 2261154, Gaurav Singh Koranga S/o Mr., 2261224 of B.Tech.(CSE) to Graphic Era Hill University Bhimtal Campus for the award of bonafide work carried out by them. They have worked under my guidance and supervision and fulfilled the requirement for the submission of a report.

Mr. Price Kumar
(Project Guide)

Dr. Ankur Singh Bisht
(Head, CSE)

ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director **'Prof. (Col.) Anil Nair (Retd.)',** GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president **'Prof. (Dr.) Kamal Ghanshala'** for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to **'Dr. Ankur Singh Bisht'** (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide **Mr. Prince Kumar'** (Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus) and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

Surjeet Kumar Verma, 2261562
Suraj Singh, 2261560
Chetan Negi, 2261154
Gaurav Singh Koranga, 2261224

Abstract

This project presents the development of a File Compression/Decompression Tool designed to efficiently reduce file sizes for optimized storage and faster transmission. The tool employs lossless compression algorithms to ensure data integrity, enabling users to compress files without any loss of original information and restore them accurately during decompression. It supports multiple file formats and integrates a user-friendly interface for ease of use. The primary objectives include improving storage efficiency, minimizing bandwidth usage, and ensuring data security during file transfers. Benchmark testing demonstrates the tool's effectiveness in achieving significant compression ratios while maintaining high-speed performance and reliability. This project underscores the practical importance of file compression in data management systems and network communications.
This project involves the design and implementation of a **File Compression/Decompression Tool** aimed at reducing file sizes for efficient storage management and faster data transmission. The tool utilizes **lossless compression algorithms** such as Huffman Coding and Run-Length Encoding (RLE), to ensure that the original data can be perfectly reconstructed after decompression. By preserving the integrity and quality of data, this tool is especially useful for compressing text files, documents, source code, and other data-sensitive file types.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ADK: Android Development Kit, what people use to develop anything for the Android such as APPs and ROM's

ADB: Android Debug Bridge, a command-line debugging application included with the SDK.

ART - successor of Dalvik, a virtual machine (VM) that's part of the Android OS. ART is faster than Dalvik and saves more energy :)

Baseband or Radio: In communications & signal processing, the baseband describes signals & systems whose range of the frequencies measured from close to 0 hertz to a cut-off frequency, a maximum bandwidth or highest signal frequency; it is sometimes used to describe frequencies starting close to zero

Binary - (sometimes called bin's) a group of executable files.

INTRODUCTION

1.1 Prologue

In today's digital world, efficient storage and transmission of data have become increasingly critical. Files of all types—from documents and images to videos and software—demand significant storage space and bandwidth. File compression plays a vital role in reducing this footprint without sacrificing data integrity, enabling faster transfer speeds and optimized storage usage.

This project presents a **File Compression/Decompression Tool** that employs two fundamental compression techniques: **Huffman Coding** and **Run-Length Encoding (RLE)**. Huffman Coding is a widely recognized lossless compression algorithm that assigns shorter codes to more frequent data elements, effectively reducing file size. On the other hand, Run-Length Encoding offers a simple and efficient approach to compress data with repetitive sequences by encoding consecutive duplicates as count-value pairs.

The tool offers a user-friendly graphical interface, allowing users to easily select files, choose the preferred compression method, and manage compressed output. It ensures that original file information is preserved, supporting seamless decompression back to the original form.

Through this project, the underlying principles of data compression are explored and implemented, demonstrating how classic algorithms can be leveraged to address modern storage challenges. It aims to provide both a practical utility and an educational insight into file compression techniques.

1.2 Background and Motivations

With the exponential growth of digital data, efficient storage and transmission have become critical challenges. As users generate and consume vast amounts of information daily, the need to reduce file sizes without losing data integrity has become more important than ever. File compression techniques serve this purpose by minimizing the amount of space required to store data and reducing the time and bandwidth needed for data transfer.

Traditional compression algorithms, such as Huffman Coding and Run-Length Encoding (RLE), have proven effective in various applications. Huffman Coding is a fundamental lossless compression algorithm that generates optimal prefix codes based on the frequency of data elements, making it ideal for general-purpose compression. RLE, on the other hand, is a simple yet effective technique for compressing data with repetitive sequences, commonly used in image and video compression.

The motivation behind this project is to develop an easy-to-use compression/decompression tool that leverages these classical algorithms to help users optimize their file storage and transfer processes. By providing a graphical interface that supports multiple compression techniques, the tool not only facilitates practical use but also offers educational insight into how different algorithms operate.

Additionally, this project aims to demonstrate how foundational compression methods can still be relevant and useful in handling everyday file management tasks, especially in scenarios where lightweight, lossless compression is required without relying on complex external libraries or formats.

1.3 Problem Statement

With the continuous increase in digital data creation and storage, managing file sizes efficiently has become a significant challenge. Large files consume substantial storage space and require longer times for transfer across networks, impacting both system performance and user experience. While many compression tools exist, they often come with complexities, proprietary formats, or lack user-friendly interfaces for basic lossless compression needs.

This project addresses the need for a simple, effective, and accessible file compression and decompression tool that supports lossless algorithms like Huffman Coding and Run-Length Encoding. The challenge lies in implementing these algorithms efficiently within a graphical interface that allows users to compress files to smaller sizes and reliably restore the original data without loss, all while ensuring usability and reasonable processing time for files of varying sizes.

1.4 Objectives and Research Methodology

The main objectives of this project are:

1. **Develop a User-Friendly Interface:**
   Create an intuitive graphical user interface (GUI) that allows users to easily select files and choose between different compression methods.
2. **Implement Lossless Compression Algorithms:**
   Implement Huffman Coding and Run-Length Encoding (RLE) algorithms to enable efficient and reliable lossless compression and decompression of files.
3. **Preserve File Integrity:**
   Ensure that decompressed files exactly match the original files, maintaining data integrity with no loss of information.
4. **Support Multiple File Types:**
   Allow compression and decompression of various file types, preserving their original extensions and metadata where possible.
5. **Optimize Performance:**
   Design the application to handle files of varying sizes efficiently, minimizing processing time and system resource usage.
6. **Provide Feedback and Progress Indicators:**
   Incorporate progress bars and status messages to keep users informed during compression and decompression processes.

Research Methodology

The methodology followed for this project includes the following steps:

1. **Literature Review:**
   Conducted a detailed study of existing lossless compression techniques, focusing on Huffman Coding and Run-Length Encoding. Reviewed related research papers, online resources, and algorithm descriptions to understand their principles, strengths, and limitations.
2. **Algorithm Design and Implementation:**
   Designed and implemented the core algorithms for Huffman and RLE compression and decompression in Python. Special attention was paid to encoding efficiency, memory management, and correctness.
3. **Graphical User Interface Development:**
   Developed a user-friendly GUI using the Tkinter library in Python, integrating file selection dialogs, compression options, and progress indicators to enhance user experience.
4. **Testing and Validation:**
   Performed extensive testing with files of various sizes and formats to validate the correctness and efficiency of the compression and decompression processes. Addressed issues such as processing delays and file integrity errors.

1.5 Project Organization

This project is structured into several key modules and components to ensure clear separation of concerns, ease of development, and maintainability. The organization is as follows:

1. **User Interface (UI) Module:**
    - Developed using Python's Tkinter library to provide a graphical interface for user interaction.
    - Handles file selection, compression method selection, and displays progress and status messages.
    - Contains buttons for compressing and decompressing files, and menu options such as "About."
2. **Compression Algorithms Module:**
    - Implements the core logic for file compression and decompression.
    - Contains Huffman Coding functions including building the frequency tree, generating codes, encoding, and decoding data.
    - Contains Run-Length Encoding (RLE) functions for simple and efficient compression of repetitive data.
3. **File Handling Module:**
    - Manages reading from and writing to files, including saving compressed files with appropriate metadata.
    - Supports storing and retrieving information such as the original file extension to ensure correct decompression.
4. **Progress Management Module:**
    - Updates the progress bar dynamically during compression and decompression operations to provide user feedback.
    - Ensures the application remains responsive, especially during operations on large files.
5. **Error Handling and User Notifications:**
    - Detects and manages errors such as invalid file selections or corrupted files.
    - Provides clear messages to the user through dialogs to guide them and prevent crashes.
6. **Documentation and About Section:**
    - Provides users with information about the tool's purpose and usage.
    - Includes code documentation for maintainers and future development.

HARDWARE AND SOFTWARE REQUIREMENTS

2.1Hardware Requirement

Windows	OS X	Linux
Microsoft Windows 8/7/Vista/2003 (32 or 64 bit)	Mac OS X 10.8.5 or higher, up to 10.9 (Mavericks)	GNOME or KDE or Unity desktop
4 GB RAM minimum, 8 GB RAM recommended	2 GB RAM minimum, 4 GB RAM recommended	2 GB RAM minimum, 4 GB RAM recommended
400 MB hard disk space plus at least 1 GB for Android SDK, emulator system images, and caches	400 MB hard disk space plus at least 1 GB for Android SDK, emulator system images, and caches	400 MB hard disk space plus at least 1 GB for Android SDK, emulator system images, and caches
Python	Python	Python
Optional for the accelerated emulator: Intel processor with support for Intel VT-x, Intel EM64T (Intel 64), and Execute Disable (XD) Bit functionality	Optional for the accelerated emulator: Intel processor with support for Intel VT-x, Intel EM64T (Intel 64), and Execute Disable (XD) Bit functionality	GNU C Library (Glibc) 2.11 or later

2.2 Software Requirement

Android studio.

CODING OF FUNCTIONS

Huffman Coding:

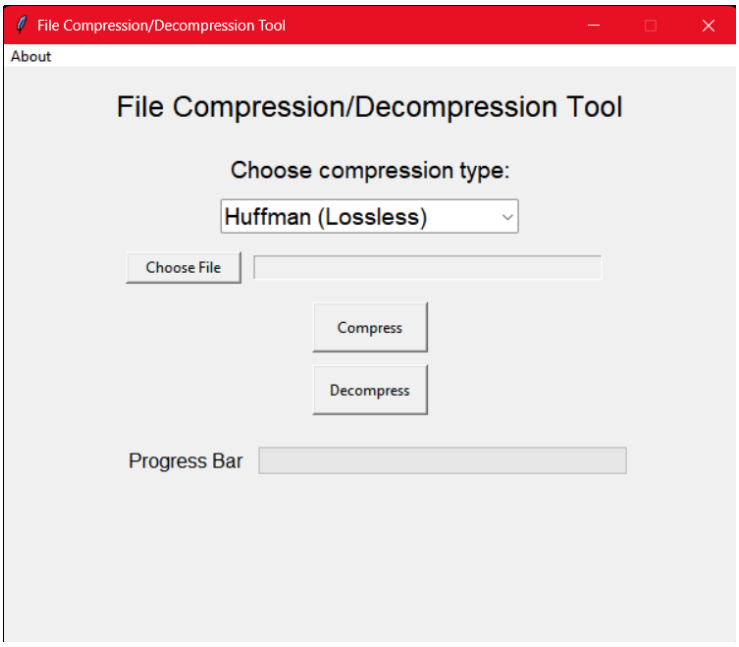
```
import heapq

class HuffmanNode:
    def __init__(self, byte=None, freq=0):
        self.byte = byte
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.freq < other.freq

def build_huffman_tree(data):
    freq = {}
    for b in data:
        freq[b] = freq.get(b, 0) + 1

    heap = [HuffmanNode(b, f) for b, f in freq.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = HuffmanNode(freq=left.freq + right.freq)
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)

    return heap[0]

def build_codes(node, prefix=b"", code_map=None):
    if code_map is None:
        code_map = {}
    if node:
        if node.byte is not None:
            code_map[node.byte] = prefix
        else:
            build_codes(node.left, prefix + b"0", code_map)
            build_codes(node.right, prefix + b"1", code_map)
    return code_map

def encode_data(data, codes):
    bits = ""
    for byte in data:
        bits += codes[byte].decode()  # b"0" or b"1" to "0" or "1"

    # Pad bits to full bytes
    padding = (8 - len(bits) % 8) % 8
    bits += "0" * padding

    # Store padding info in first byte
    padded_info = f"{padding:08b}"
    bits = padded_info + bits

    b_array = bytearray()
    for i in range(0, len(bits), 8):
        byte = bits[i:i+8]
        b_array.append(int(byte, 2))
    return b_array

def decode_data(encoded_bytes, codes):
    bits = "".join(f"{byte:08b}" for byte in encoded_bytes)
    padding = int(bits[:8], 2)
    bits = bits[8:-padding] if padding > 0 else bits[8:]
```

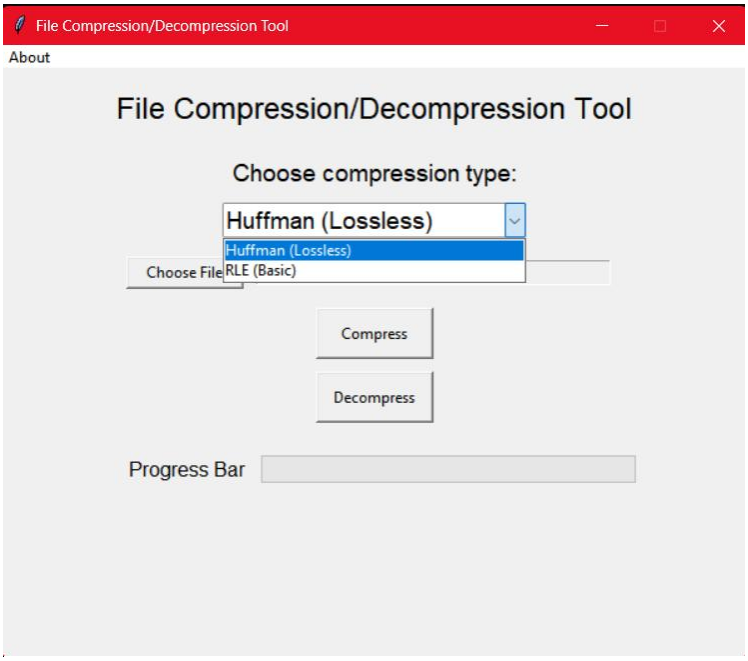
```
    reverse_codes = {v.decode(): k for k, v in codes.items()}
    current_code = ""
    decoded = bytearray()

    for bit in bits:
        current_code += bit
        if current_code in reverse_codes:
            decoded.append(reverse_codes[current_code])
            current_code = ""

    return decoded
```

RLE Encoding:

```
def rle_compress(data):
    compressed = bytearray()
    i = 0
    n = len(data)

    while i < n:
        count = 1
        while i + 1 < n and data[i] == data[i + 1] and count < 255:
            count += 1
            i += 1
        compressed.extend([data[i], count])
        i += 1
    return compressed

def rle_decompress(compressed):
    decompressed = bytearray()
    i = 0
    n = len(compressed)

    while i < n:
        byte = compressed[i]
        count = compressed[i + 1]
        decompressed.extend([byte] * count)
        i += 2
    return decompressed
```

SNAPSHOTS


File Compression/Decompression Tool
About
File Compression/Decompression Tool
Choose compression type:
Huffman (Lossless)
Choose File
Compress
Decompress
Progress Bar


File Compression/Decompression Tool
About
File Compression/Decompression Tool
Choose compression type:
Huffman (Lossless)
Huffman (Lossless)
RLE (Basic)
Choose File
Compress
Decompress
Progress Bar

LIMITATIONS

- **Compression Efficiency:**
    - The compression ratio achieved by Huffman Coding and Run-Length Encoding may not be optimal for all file types, especially for already compressed or highly random data.
    - RLE is particularly effective only for files with long runs of repeated bytes, so it may provide little or no compression benefit on typical binary files.
- **Performance on Large Files:**
    - Compression and decompression of very large files (multi-megabyte or larger) may lead to increased processing time and higher memory consumption, potentially causing the application to become unresponsive without further optimization.
- **Limited Algorithm Choices:**
    - The tool currently supports only Huffman Coding and RLE algorithms, lacking more advanced or specialized compression techniques like LZW, DEFLATE, or arithmetic coding which could yield better compression results.
- **No Support for Encrypted or Proprietary File Formats:**
    - Files that are encrypted or use proprietary compression formats may not be compatible or may not compress effectively.
- **Basic Metadata Handling:**
    - Only the original file extension is preserved for decompression. Other metadata such as timestamps, permissions, or file attributes are not saved or restored.
- **Single-Threaded Processing:**
    - The current implementation runs compression and decompression operations on the main thread, which can lead to UI freezing during lengthy operations on large files.
- **No Partial or Stream Compression:**
    - The tool processes entire files in memory, lacking support for streaming or partial compression/decompression, which limits scalability.

ENHANCEMENTS

1. **Implement Additional Compression Algorithms:**
    - Integrate more advanced lossless compression algorithms such as LZW, DEFLATE, or arithmetic coding to improve compression ratios across diverse file types.
2. **Multithreading or Asynchronous Processing:**
    - Incorporate multithreading or asynchronous task handling to prevent the application's GUI from freezing during compression and decompression of large files, ensuring better responsiveness.
3. **Support for Partial and Streaming Compression:**
    - Enable the tool to process large files in chunks or streams, reducing memory usage and improving scalability for very large files.
4. **Enhanced Metadata Preservation:**
    - Extend metadata handling to save and restore additional file attributes like timestamps, permissions, and ownership during compression and decompression.
5. **Batch Processing:**
    - Allow users to compress or decompress multiple files or entire directories in one operation to improve usability.
6. **File Format Identification and Automatic Selection:**
    - Implement automatic detection of file types and recommend the most effective compression method accordingly.
7. **Improved Compression File Format:**
    - Design a more robust container format for compressed files that includes checksums or error detection to ensure data integrity.
8. **Progress Feedback and Logging:**
    - Provide more detailed progress indicators and logging features to help users monitor operations and diagnose issues.
9. **Cross-Platform Support and Packaging:**
    - Package the application for easy installation on multiple platforms (Windows, macOS, Linux) with a standalone executable.
10. **User Customization Options:**
    - Allow users to customize compression settings, such as compression level or code table export options, to tailor performance and output size.

CONCLUSION

This File Compression/Decompression Tool successfully demonstrates the practical implementation of fundamental lossless compression algorithms, specifically Huffman Coding and Run-Length Encoding (RLE). By integrating these algorithms within a user-friendly graphical interface, the project provides a functional utility for reducing file sizes and restoring original data without loss.

Throughout the development process, challenges related to performance, especially with larger files, and compression efficiency were addressed, resulting in a reliable application suitable for basic compression needs. The tool not only serves as a practical solution for managing storage and transmission but also as an educational resource illustrating core concepts of data compression.

While the current version has limitations in algorithm diversity and handling very large files, it lays a solid foundation for future enhancements. Incorporating more advanced algorithms, optimizing performance, and expanding features will further increase its utility and effectiveness.

Overall, this project reinforces the importance of data compression in digital file management and demonstrates how classical algorithms remain relevant and accessible for everyday use.

REFERENCES

- Huffman, D. A. (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9), 1098–1101.

- Sayood, K. (2017). *Introduction to Data Compression* (5th ed.). Morgan Kaufmann.

- Salomon, D., & Motta, G. (2010). *Handbook of Data Compression* (5th ed.). Springer.

- Nelson, M., & Gailly, J.-L. (1996). The Data Compression Book (2nd ed.). M&T Books.

- Ziv, J., & Lempel, A. (1977). A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23(3), 337–343.

- Wallace, G. K. (1991). The JPEG Still Picture Compression Standard. *Communications of the ACM*, 34(4), 30–44.

- Python Software Foundation. (n.d.). Tkinter — Python interface to Tcl/Tk. Retrieved from https://docs.python.org/3/library/tkinter.html

- Stallings, W. (2013). *Data and Computer Communications* (10th ed.). Pearson.