**1. Project Title**

File Compressor

**2. Abstract**

File compression is a crucial process in computing, aiming to reduce the size of files while preserving their essential data. It plays a pivotal role in space optimization, bandwidth conservation, cost savings, and data backup. Common compression techniques include lossless methods like Huffman Coding and lossy methods like JPEG for multimedia.

Compressed files find applications in archiving, data transfer, software distribution, and backup. Archive formats like ZIP and RAR bundle files, enabling efficient storage and distribution. Despite its benefits, compression poses challenges, such as resource-intensive processing, compatibility issues between formats, and the trade-offs of lossy compression, where some data quality is sacrificed.

In conclusion, file compression is indispensable for efficient data management and transmission. By selecting appropriate compression methods, users can strike a balance between data size reduction and data integrity preservation.

**3. Introduction**

File compression is a widely used technique in computer science and information technology to reduce the size of files while preserving their essential data. One of the most effective methods for file compression is Huffman coding, which is a variable-length prefix coding algorithm. Huffman coding is widely employed in various applications, including data compression, image compression, and multimedia compression.

Huffman coding is a lossless data compression algorithm that assigns variable-length binary codes to different characters or symbols in a data stream. Unlike fixed-length coding methods, where each character is represented by a fixed number of bits, Huffman coding assigns shorter codes to more frequent characters and longer codes to less frequent characters. This means that commonly occurring characters are represented by shorter bit sequences, leading to more efficient compression.

**How Huffman Coding Works:**

1. **Frequency Analysis**: To apply Huffman coding, the first step is to perform a frequency analysis of the input data. This involves determining the frequency of each character or symbol in the data stream. The more frequent a character is, the shorter its Huffman code will be.
2. **Building the Huffman Tree**: Once the frequencies are determined, a Huffman tree is constructed. This tree is a binary tree in which each leaf node represents a character and is associated with its frequency. The tree is built in a way that ensures that characters with higher frequencies are closer to the root of the tree.
3. **Assigning Huffman Codes**: Starting from the root of the tree, Huffman codes are assigned to each character by traversing the tree. Moving left in the tree corresponds to adding a '0' to the code, and moving right corresponds to adding a '1'. The codes are read by following the path from the root to the leaf associated with a character.
4. **Compression**: With the Huffman codes assigned to each character, the input data is then encoded using these codes. This results in a compressed representation of the data, which can be stored or transmitted more efficiently.
5. **Decompression**: To retrieve the original data, the Huffman tree and encoded data are used. Starting at the root of the tree, the encoded data is traversed, and as the path is followed, characters are decoded until the original data is reconstructed.

**Advantages of Huffman Coding:**

1. **Efficient Compression**: Huffman coding is highly efficient for compressing data, especially when there are significant variations in character frequencies.
2. **Lossless Compression**: It is a lossless compression method, meaning that the original data can be perfectly reconstructed from the compressed data.
3. **Variable-Length Codes**: Huffman coding allows for variable-length codes, ensuring that frequently occurring characters have shorter codes, optimizing compression.
4. **Widely Used**: Huffman coding is widely used in various applications, including file compression, image compression (e.g., JPEG), and text compression (e.g., ZIP files).

**4. Literature Review**

Huffman coding, developed by David A. Huffman in 1952, is a variable-length prefix coding technique that assigns shorter codes to more frequent symbols, optimizing compression ratios. This pioneering work laid the foundation for lossless data compression and has since seen significant advancements.

**Adaptive Huffman Coding**: Adaptive Huffman coding algorithms, such as the Vitter algorithm, offer the advantage of dynamically updating the codebook during compression and decompression.

**5. Problem Statement**

File compression is a fundamental aspect of modern data management, with significant implications for storage, transmission, and efficiency in various domains. Despite the existence of numerous compression algorithms and tools, several challenges and issues remain unresolved:

**Balancing Compression Ratios and Processing Speed**: Achieving high compression ratios often comes at the cost of longer compression and decompression times. Striking the right balance between compression efficiency and processing speed remains a challenge, particularly in real-time data streaming and large-scale data analysis applications.

**Lossy Compression Optimization**: Lossy compression is essential for multimedia data like images, audio, and video. However, optimizing lossy compression algorithms to minimize perceptual data loss while maximizing compression ratios remains a complex problem. Striving for higher fidelity without significantly increasing file sizes is a constant challenge.

**Adaptation to Emerging Technologies**: The landscape of computing is continually evolving, with the emergence of new technologies such as quantum computing and edge computing. Existing compression methods may not be optimized for these paradigms, necessitating the development of new algorithms and techniques to address their unique requirements.

**Security Concerns**: Compression is often used to obscure file contents, which can be exploited for malicious purposes. Research is needed to develop compression techniques that not only maintain data integrity but also enhance security, protecting against data breaches and unauthorized access.

**Scalability**: With the exponential growth of data, scalable compression solutions are vital. Compression algorithms and tools must be capable of handling vast datasets efficiently, without compromising performance or requiring extensive computational resources.

**Energy Efficiency**: In energy-constrained environments, such as mobile devices and IoT sensors, compression algorithms must be optimized to minimize energy consumption during compression and decompression processes.

**Standardization and Compatibility**: The proliferation of various compression formats can lead to interoperability issues. There's a need for standardization efforts to ensure compatibility between different compression tools and formats, reducing data fragmentation and ensuring seamless data exchange.

Addressing these challenges requires interdisciplinary research that combines computer science, information theory, mathematics, and engineering. Innovative approaches, including machine learning and artificial intelligence, may hold the key to optimizing compression algorithms and tools for the complex and dynamic data environments of today and the future. Solving these issues will contribute to more efficient data management, reduced storage costs, faster data transmission, and enhanced data security.

**6. Objectives**

* Security-Enhanced Compression: Develop compression techniques that not only maintain data integrity but also enhance security. Implement encryption and authentication mechanisms within compression algorithms to protect against unauthorized access and data breaches.
* Energy-Efficient Compression: Optimize compression algorithms for energy efficiency, particularly in resource-constrained environments like mobile devices and IoT sensors. Reduce energy consumption during compression and decompression processes.
* Efficiency Enhancement: Improve the efficiency of compression algorithms to achieve higher compression ratios while minimizing processing overhead. Develop algorithms that strike a better balance between compression efficiency and speed.
* User Interface(UI):The user interface (UI) of the File Compressor plays a crucial role in enhancing the user experience and facilitating efficient interaction with the compression tool. A well-designed UI ensures that users can easily navigate through the compression process, providing necessary inputs and receiving feedback on the progress.

**7. Methodology**

Creating file compressor using huffman coding include several steps:

Data Analysis: read the input of the file to analyse frequency of the data.

Frequency Table: Build a frequency table that maps each symbol/character. This table is important for constructing the huffman tree

Huffman Tree:Use the frequency table to build a Huffman tree. The goal is to create a tree where frequently occurring symbols have shorter codes and less frequent symbols have longer codes.

Generate Huffman Code:Traverse the Huffman tree to generate unique binary codes for each symbol. Codes are assigned based on the path from the root to each leaf node, with left branches typically representing "0" and right branches representing "1".

Encode the Data:

Replace each symbol in the original data with its corresponding Huffman code. This results in a compressed representation of the data.

File Header (Metadata):

Create a file header or metadata that includes information necessary for decompression. This header should contain the frequency table or any other information required to reconstruct the Huffman tree during decompression. Append this header to the compressed data.

Write Compressed Data to Output File:

Write the compressed data (including the header) to an output file. This file is the compressed version of the input file.

Decompression:

To decompress a Huffman-compressed file, read the file header to reconstruct the Huffman tree. Then, decode the compressed data by traversing the tree to retrieve the original symbols.

User Interface(UI)

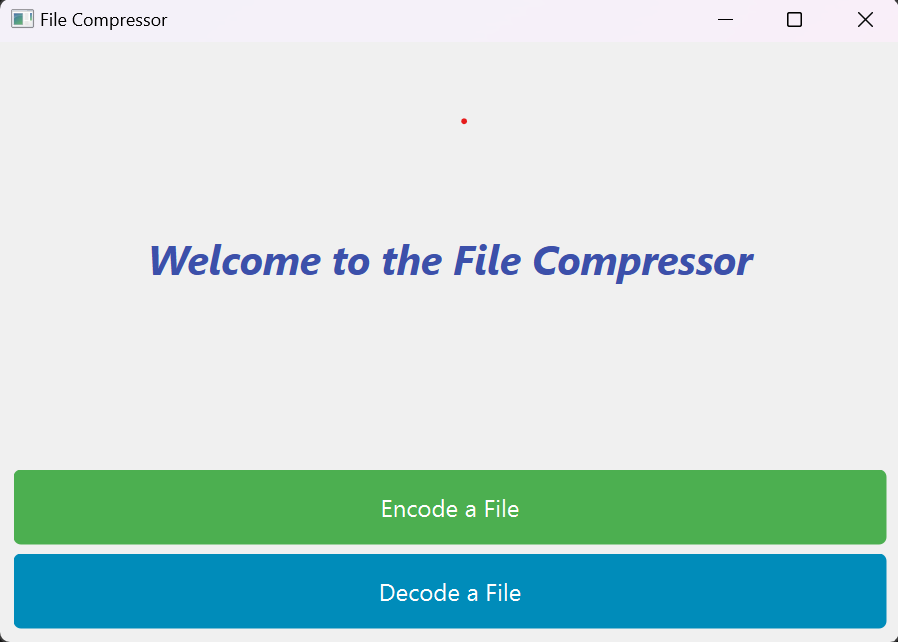
We have developed the User Interface using the QT framework, a powerful C++ Library widely use for GUI Application

Key Component of the UI include:

#### 7.1 Buttons for Encode and Decode:

Two prominent buttons have been implemented for distinct functionalities:

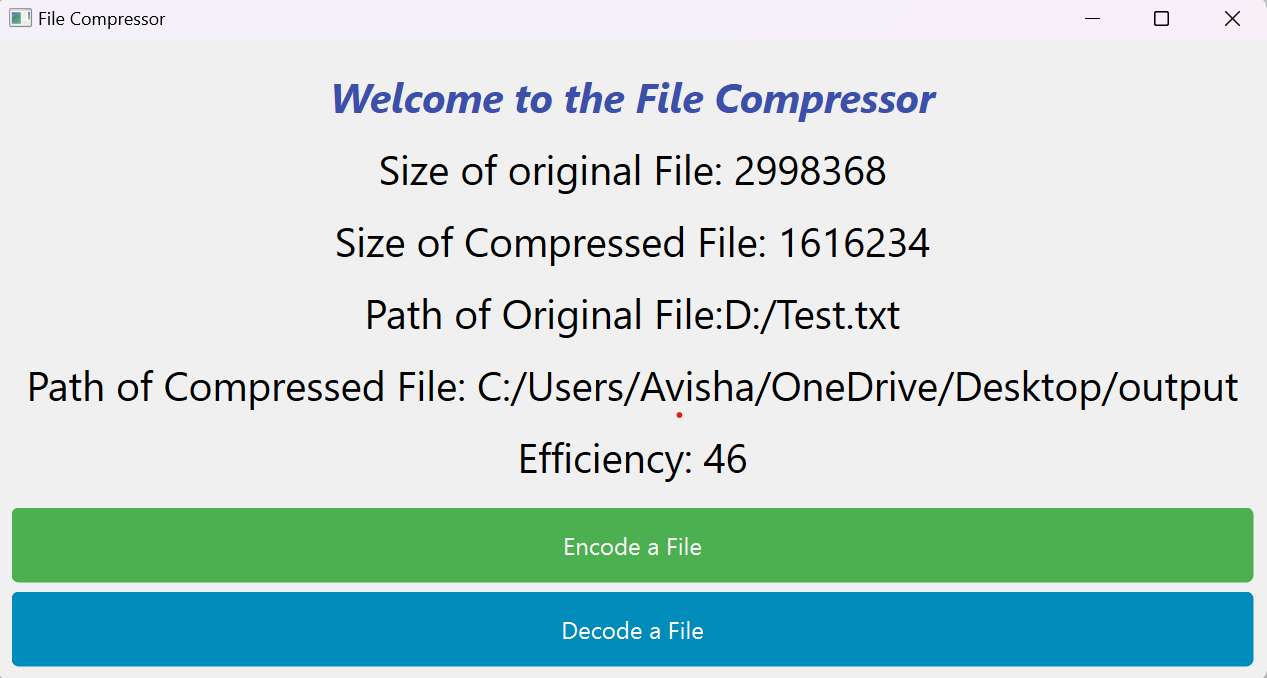
* **Encode Button:** Initiates the file compression process using Huffman coding. Users can select the file they wish to compress, and upon clicking the "Encode" button, the tool performs the compression and displays relevant information.
* **Decode Button:** Enables users to decompress a previously compressed file. After selecting the compressed file, clicking the "Decode" button triggers the decompression process, restoring the original file.



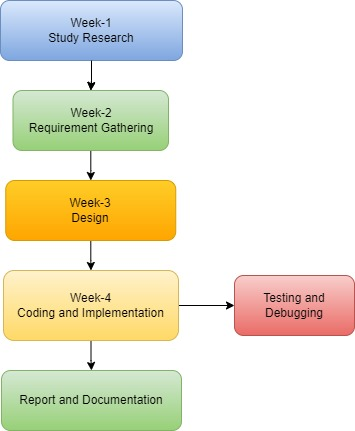
#### 7.2 Display of Details:

Upon completion of encoding or decoding, the UI provides users with crucial information:

* **Efficiency Metrics:** The UI displays metrics related to the compression efficiency, such as the achieved compression ratio. This information helps users assess the effectiveness of the compression process.
* **Location of Original and Compressed Files:** The UI dynamically shows the file paths of both the original file and the compressed file. This transparency allows users to verify the locations and ensures a clear understanding of the compression process.



**8.Pert Chart**



**9. References**

**[1]“Data Structures And Algorithms Made Easy: Data Structure And Algorithmic Puzzles”-Narasimha Karumanchi.**

**[2]**[**https://www.javatpoint.com/huffman-coding**](https://www.javatpoint.com/huffman-coding)

**[3]**[**https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/**](https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/)

**[4]**[**https://youtu.be/EkjaiDsiM-Q?si=3Bf4pkHzuMn5VBtK**](https://youtu.be/EkjaiDsiM-Q?si=3Bf4pkHzuMn5VBtK)**(QT tutorial)**

**[5]**[**https://www.geeksforgeeks.org/data-structures/linked-list/doubly-linked-list/**](https://www.geeksforgeeks.org/data-structures/linked-list/doubly-linked-list/)

**[6]**[**https://www.geeksforgeeks.org/binary-tree-data-structure/**](https://www.geeksforgeeks.org/binary-tree-data-structure/)