Huffman Compression Tool Report

Data compression is an essential technique in computer science, allowing for efficient storage and transmission of information. One widely used lossless compression algorithm is Huffman coding, which assigns variable-length binary codes to characters based on their frequencies. This project implements Huffman encoding and decoding to compress text files, store the compressed data in binary format, and compare file sizes before and after compression to evaluate the efficiency of the algorithm.

The primary objective of this project is to implement Huffman coding as a method for lossless textual data compression. Huffman coding is a widely used algorithm in data compression, ensuring that data can be reduced in size while maintaining its original integrity upon decompression. This project aims to build a system that performs both encoding and decoding operations, efficiently stores compressed data, and evaluates the compression's effectiveness. One of the key goals of this project is to convert plain text into a compressed binary format using Huffman coding. The encoding process involves analyzing character frequencies in the input text, building a Huffman Tree, which assigns shorter binary codes to more frequently occurring characters, and replacing characters in the text with their corresponding Huffman codes to form the compressed representation. After converting the text into Huffman codes, the next step is to save the compressed data in a binary file so it can be used later. Normally, text files store characters using standard formats like ASCII, where each character takes up the same amount of space. However, Huffman coding replaces characters with shorter binary codes, making the file smaller. By doing this, the project shows how Huffman compression can help store files more efficiently while keeping all the original information intact.

Since Huffman coding is a lossless compression method, the original text must be fully recovered after decompression. The compressed data is decoded using the same Huffman tree that was created during encoding. The binary sequences in the compressed file are translated back into their original characters. To understand how effective Huffman coding is, the project compares the sizes of the original and compressed files. This comparison helps determine how much storage space is saved after compression. Huffman coding is most effective for files with predictable patterns, such as text documents, logs, and structured data.

The implementation of this project is divided into four key components, each playing a specific role in the encoding, compression, and decoding processes. The huffman_algorithm.py file contains the core logic for Huffman encoding and decoding. It is responsible for transforming text into a compressed binary representation and reconstructing it when needed. The Huffman encoding process begins with calculating the frequency of each character in the input text. The program counts how often each character appears, which helps determine the structure of the Huffman Tree. Next, a priority queue (min-heap) is used to build the Huffman Tree, where characters that occur more frequently receive shorter codes, while less frequent characters get longer ones. Once the tree is constructed, the program generates unique binary codes for each character based on their position in the tree. Finally, the text is encoded by replacing each character with its corresponding Huffman code, creating a compressed binary sequence that reduces file size while maintaining all original information. The Huffman decoding process reverses the encoding to reconstruct the original text. It begins by reading the Huffman codes

and rebuilding the Huffman Tree. Then, the program traverses the binary sequences using the tree, mapping each sequence back to its corresponding character. This step-by-step traversal ensures that the compressed data is accurately converted back into its original form. The decoding process guarantees lossless decompression, proving that the method is both efficient and reversible, making Huffman coding a reliable technique for data compression.

The file_processor.py module is essential for managing file operations in the Huffman compression process, ensuring data is properly read, stored, and retrieved before and after compression. It handles reading the input text from a file (input.txt), writing the compressed data into a binary file (compressed.bin), and efficiently storing encoded data to minimize storage overhead. Instead of storing text as fixed-size characters, Huffman encoding saves space by storing variable-length binary sequences. Additionally, the module reads and decompresses the stored binary file, reconstructing the original text accurately. By performing these tasks, file_processor.py ensures a seamless compression and decompression process while optimizing storage and preserving data integrity. The file size comparison function in file_processor.py plays a crucial role in evaluating the effectiveness of Huffman compression. It calculates and compares the sizes of the original text file and the compressed binary file, providing insight into how much space is saved. Understanding file size reduction also lays the groundwork for potential optimizations and improvements in future implementations.

The main.py file is the main program that runs the entire Huffman compression process. It starts by reading the text from input.txt, then applies Huffman encoding to create binary codes for each character. The compressed data is saved in a file called compressed.bin. To make sure the process works correctly, the program then decodes the compressed data back into the original text. Finally, it compares the sizes of the original and compressed files to check how much space was saved. This file brings everything together, making sure encoding, decoding, and file handling work smoothly.

The huffman_gui.py file adds a Graphical User Interface (GUI) to make the Huffman compression tool easier to use. Instead of typing commands in a terminal, users can interact with the program through buttons and input fields. The GUI allows users to select a text file for compression, view the generated Huffman codes in a clear format, and click buttons to compress or decompress files. It also provides a visual comparison of file sizes before and after compression. This makes the tool more user-friendly, especially for those who are not familiar with coding, while also improving the overall experience by showing real-time results in an interactive way.

This project successfully implements Huffman coding for lossless text compression, showing how it can reduce file sizes while keeping all information intact. By using a structured approach with separate modules for encoding, decoding, and file handling, the project demonstrates the efficiency and usefulness of Huffman compression. The addition of a GUI makes it easier for users to operate without command-line knowledge. By comparing file sizes before and after compression, the project highlights the advantages of Huffman coding in compression techniques and real-world applications.