CS--218 DATA STRUCTURES & ALGORITHM

# PROJECT REPORT

# FILE COMPRESSION TOOL

**Group Members**:
Osama Hassan -- CS-24120
Muhammad Ali -- CS-24134
Basil Hassan -- CS-24133

## 1. Problem Statement

The aim of this project is to develop a file compression tool that reduces the size of text files efficiently while maintaining data integrity. The project includes a Graphical User Interface (GUI) developed using Streamlit, allowing users to compress and decompress files easily without requiring command-line interaction.

The tool demonstrates complex problem-solving through algorithm selection, implementation, and optimization

## 2. Compression Algorithm

For this project, the Huffman Coding Algorithm was selected due to its efficiency in lossless text compression.

**Algorithm Overview:**

*1. Frequency Calculation*: Count the occurrence of each character in the input file.
*2. Huffman Tree Construction:* Build a binary tree where the least frequent characters have the longest codes.
*3. Code Generation:* Assign variable-length binary codes to each character based on the tree structure.
*4. Encoding:* Replace each character in the file with its corresponding Huffman code.
*5. Decoding:* Traverse the Huffman tree to retrieve the original text from compressed codes.

Reason for Selection: Huffman coding guarantees optimal prefix codes, ensuring minimal average code length and adequate compression for text data.

## 3. Implementation (Code Overview)

The project was implemented in Python with Streamlit for GUI interaction.

**Key Features:**

Select input text files for compression.
Display compressed file size and compression ratio.
Save compressed output to disk.
Decompress files back to original content.
Visual feedback via Streamlit interface.

## 4. Future Enhancements

*Future improvements may include:*

Support for multiple file types (images, videos)
Improved compression using hybrid algorithms (Huffman + LZW)
Adding encryption for secure compressed files
Performance dashboard in GUI
Drag-and-drop support in Streamlit

## 5. Time and Space Complexity

**Time Complexity:**

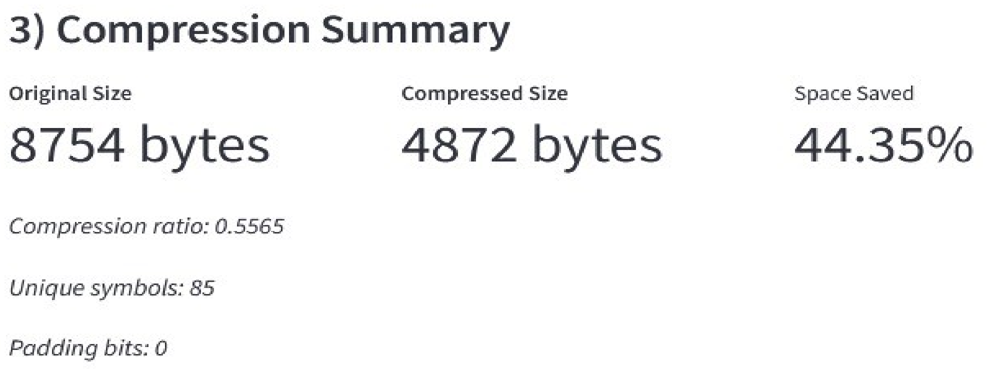
Frequency calculation: *O(n)*
Heap construction: *O(k) (k = number of unique characters)*
Tree construction: *O(k log k)*
Encoding file: *O(n)*
Overall: *O(n + k log k), where n = file size in characters*

**Space Complexity:**

Frequency dictionary: *O(k)*
Huffman tree: *O(k)*
Encoded output: *O(n) bits*
Overall: *O(n + k), efficient for typical text files*

## 6. Sample Input and Output

### i- Compressing (PY File):



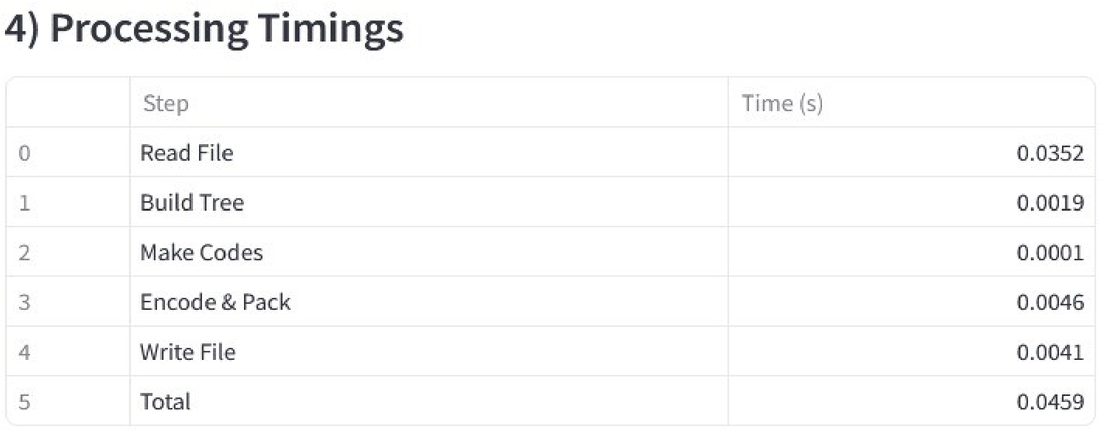
3) Compression Summary

| Original Size | Compressed Size | Space Saved |
|---|---|---|
| 8754 bytes | 4872 bytes | 44.35% |

*Compression ratio: 0.5565*

*Unique symbols: 85*

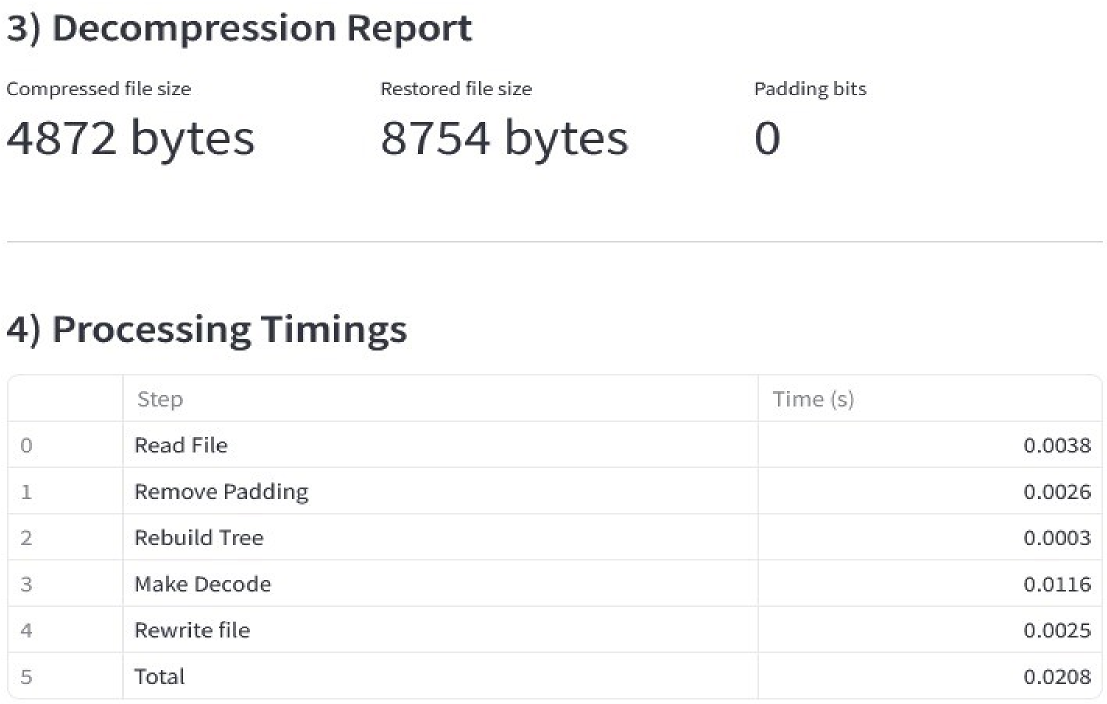
*Padding bits: 0*



4) Processing Timings

| | Step | Time (s) |
|---|---|---|
| 0 | Read File | 0.0352 |
| 1 | Build Tree | 0.0019 |
| 2 | Make Codes | 0.0001 |
| 3 | Encode & Pack | 0.0046 |
| 4 | Write File | 0.0041 |
| 5 | Total | 0.0459 |

### i-Decompressing (PY File):



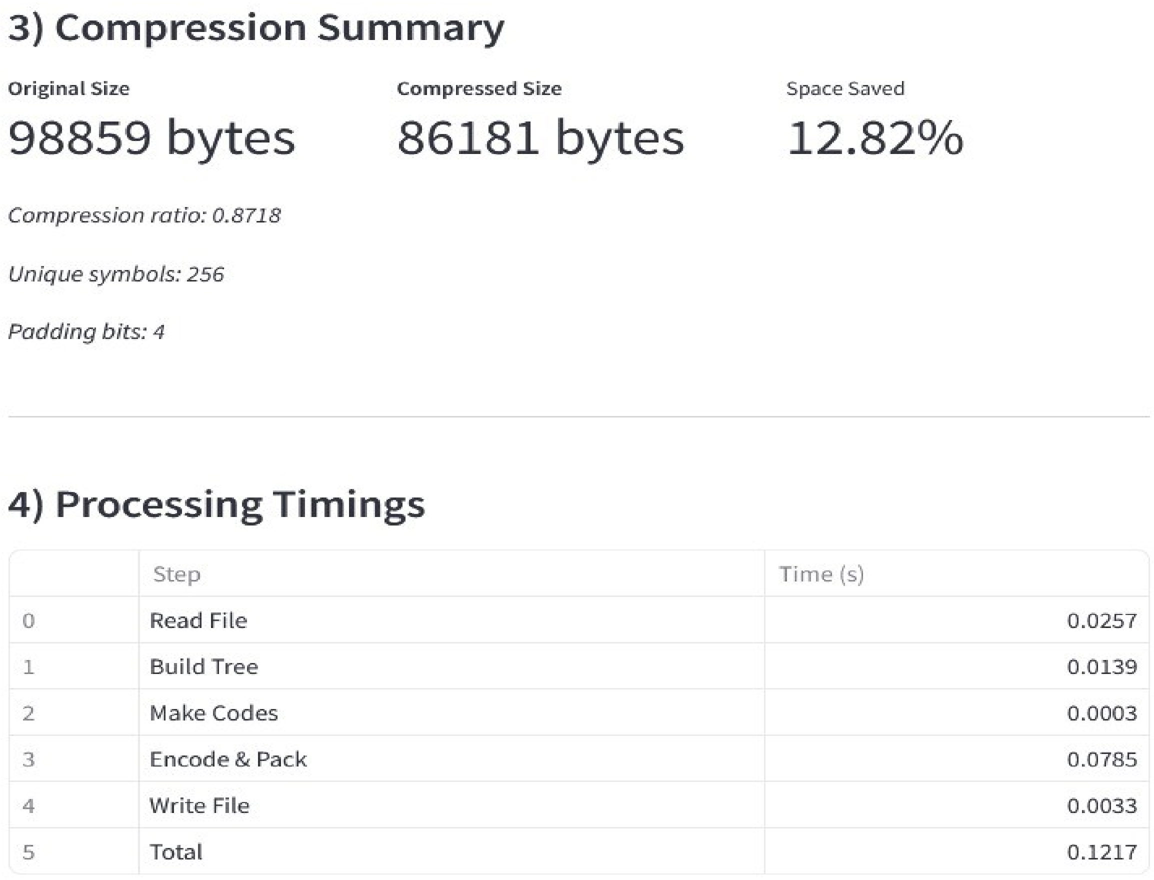
## 3) Decompression Report

| Compressed file size | Restored file size | Padding bits |
|---|---|---|
| 4872 bytes | 8754 bytes | 0 |

## 4) Processing Timings

| | Step | Time (s) |
|---|---|---|
| 0 | Read File | 0.0038 |
| 1 | Remove Padding | 0.0026 |
| 2 | Rebuild Tree | 0.0003 |
| 3 | Make Decode | 0.0116 |
| 4 | Rewrite file | 0.0025 |
| 5 | Total | 0.0208 |

**ii- Compressing (PDF File):**



## 3) Compression Summary

| Original Size | Compressed Size | Space Saved |
|---|---|---|
| 98859 bytes | 86181 bytes | 12.82% |

*Compression ratio: 0.8718*

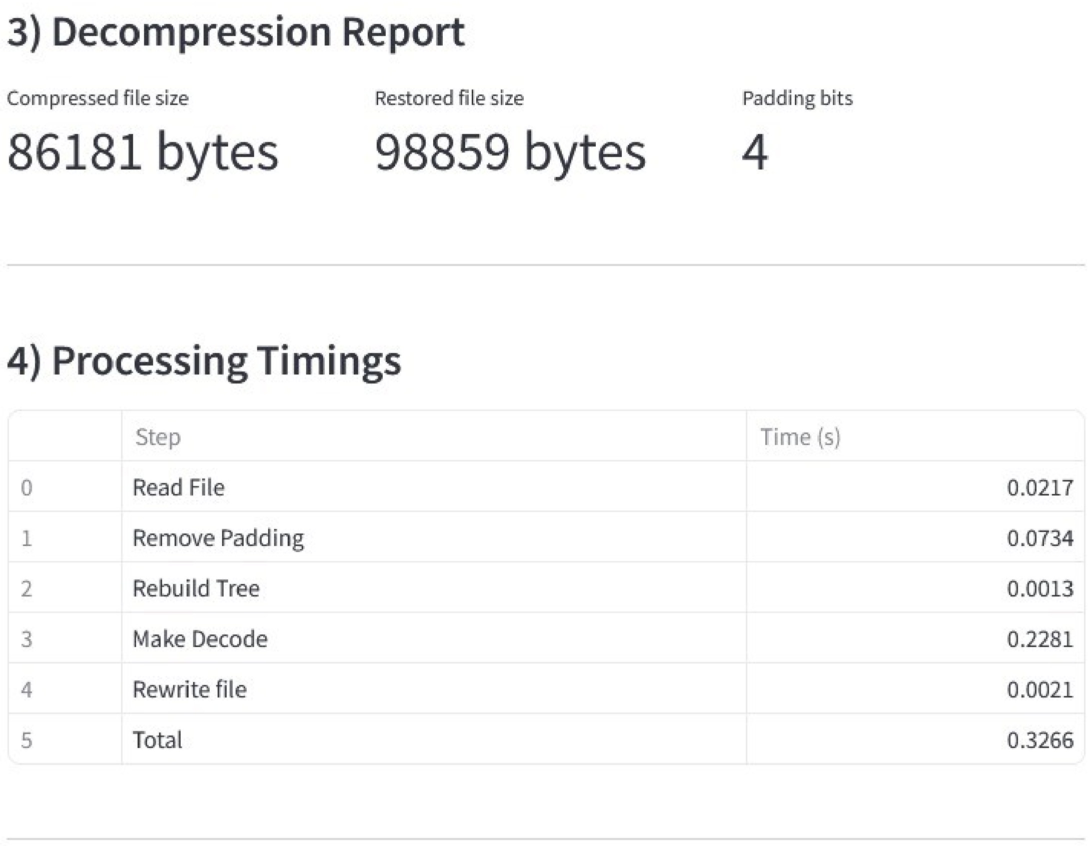
*Unique symbols: 256*

*Padding bits: 4*

## 4) Processing Timings

| | Step | Time (s) |
|---|---|---|
| 0 | Read File | 0.0257 |
| 1 | Build Tree | 0.0139 |
| 2 | Make Codes | 0.0003 |
| 3 | Encode & Pack | 0.0785 |
| 4 | Write File | 0.0033 |
| 5 | Total | 0.1217 |

**ii- Decompression (PDF file):**



### 3) Decompression Report

Compressed file size
86181 bytes

Restored file size
98859 bytes

Padding bits
4

### 4) Processing Timings

| | Step | Time (s) |
|---|---|---|
| 0 | Read File | 0.0217 |
| 1 | Remove Padding | 0.0734 |
| 2 | Rebuild Tree | 0.0013 |
| 3 | Make Decode | 0.2281 |
| 4 | Rewrite file | 0.0021 |
| 5 | Total | 0.3266 |

## 7. Conclusion

*The project successfully demonstrates:*

Application of Huffman coding for efficient text compression.
Streamlit-based GUI for user-friendly interaction.
Effective compression with significant size reduction.
Clear understanding of algorithmic complexity and practical implementation.

The tool is ready to be presented with a GUI for demonstration.