Computer Algorithms Project Report

**University**: UTAA

**Course Code:** SENG 450  
**Project Topic:** File Compression and Indexing System Using Huffman Coding and Tree Structures **Preparer:** İlayda KILINÇ  
**Tutor:** Dr.Öğr.Üyesi Şadi ŞEHAB

**List of Contents**

[**1) Introduction**](#page3)**…………………...……………………………………………………...**[**3**](#page3)

**[2) 2.Compression Module – Huffman Coding……………………………...…………3](#page3)**

**2.1 Overview………………………………………………………………………….3**

**2.2 Implementation Steps……………………………………………………………3**

**2.3 Edge Case Handling……………………………………………………………..4**

**3) 3. Storage Module – B+ Tree and Red-Black Tree………………………………..4**

**3.1 B+ Tree (Used for File Directory Indexing)……………………………………4**

**3.2 Red-Black Tree (Used for File Name Search)………………………………….4**

**4) 4. Integration – Command Line Interface (CLI)…………………………………..4**

**4.1 Features…………………………………………………………………………...4**

**4.2 Persistence………………………………………………………………………...5**

**5) 5. Testing and Example Scenarios…………………………………………………..6**

**5.1 Example Commands……………………………………………………………6**

**5.2 Edge Case Tests…………………………………………………………………6**

**6) 6. Design Justification………………………………………………………………..6**

**7) 7. Conclusion………………………………………………………………………….7**

**8) 8.Work Distribution………………………………………………………………….7**

**9) 9.Resources/References………………………………………………………………8**

**1. Introduction**

The goal of this project is to design and implement a simplified file management system that supports efficient **file compression** and **directory indexing** using well-known data structures. This system addresses three major functionalities:

1. **File Compression/Decompression** using **Huffman Coding**.
2. **Efficient File Indexing** using a **B+ Tree**.
3. **Efficient File Search** using a **Red-Black Tree**.
4. A user-friendly **Command Line Interface (CLI)** to interact with all features.

This report outlines the project architecture, algorithms, design choices, and usage.

**2. Compression Module – Huffman Coding**

**2.1 Overview**

Huffman coding is a lossless data compression algorithm that assigns variable-length binary codes to characters based on their frequencies. More frequent characters receive shorter codes.

**2.2 Implementation Steps**

* **Frequency Analysis**: The input file is read, and the frequency of each character is counted.
* **Tree Construction**: A priority queue (min-heap) is used to build the Huffman Tree by repeatedly combining the lowest frequency nodes.
* **Encoding**: Using the tree, each character is assigned a binary code. The input text is then converted into a binary string.
* **Compression**: The binary data is packed into bytes and saved along with metadata (code map) in a .compressed file.
* **Decompression**: The reverse process reconstructs the Huffman tree from the saved codes and decodes the binary data.

**2.3 Edge Case Handling**

* Single-character files are supported using a dummy node.
* Encoding errors (unknown characters) are handled gracefully.
* Both UTF-8 and Latin-1 encodings are supported for input files.

**3. Storage Module – B+ Tree and Red-Black Tree**

**3.1 B+ Tree (Used for File Directory Indexing)**

* B+ Trees support efficient insertion, deletion, and range queries.
* Leaf nodes contain actual data (file paths), and internal nodes store keys.
* A next pointer connects leaf nodes for efficient traversal and listing.

**Design Parameters:**

* Tree order (fanout) is set to 4.
* Insert and delete operations maintain tree balance and handle node splits/merges.

**3.2 Red-Black Tree (Used for File Name Search)**

* A self-balancing binary search tree where each node is red or black.
* Ensures logarithmic time complexity for insert, delete, and search operations.
* Used to enable fast exact file name lookups.

**4. Integration – Command Line Interface (CLI)**

**4.1 Features**

The CLI provides a simple way for users to:

* **Compress** a file:

metin, yazı tipi, ekran görüntüsü, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

* **Decompress** a file:

metin, yazı tipi, ekran görüntüsü, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

* **Insert** a file into the index:

metin, yazı tipi, beyaz içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

* **Search** for a file by name:

metin, yazı tipi, beyaz, ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

* **Delete** a file from the index:

metin, yazı tipi, beyaz, ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

* **List** all indexed files:

metin, yazı tipi, ekran görüntüsü, beyaz içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**4.2 Persistence**

* The index is saved in two JSON files (rbt\_index.json and bplus\_index.json) for Red-Black Tree and B+ Tree respectively.
* When the system starts, these indexes are reloaded to maintain state.

**5. Testing and Example Scenarios**

**5.1 Example Commands**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**5.2 Edge Case Tests**

* Compression of empty or single-character files.
* Inserting duplicate files.
* Searching for non-existent files.
* Handling corrupted or incompatible compressed files.

**6. Design Justification**

* **Python** was chosen due to its readability, strong standard libraries, and ease of file handling.
* **Huffman Coding** ensures high compression ratios for text files.
* **B+ Tree** is ideal for ordered data traversal (listing directory contents).
* **Red-Black Tree** provides efficient exact-match search.
* The **CLI** allows for easy integration and testing of all modules.

**7. Conclusion**

This project successfully demonstrates the integration of classic data structures and compression algorithms to build a functional and efficient file management system. It combines theoretical concepts from algorithms with practical implementation, resulting in a versatile tool that is modular, testable, and extendable.

**Appendices**

**Appendix A – File Structure**

metin, makbuz, yazı tipi, ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**Appendix B – Libraries Used**

* argparse – Command-line parsing
* heapq – Priority queue for Huffman tree
* json – Index persistence
* os, shutil – File operations

**8.Work Distribution**İlayda KILINÇ – Backend Developer, Requirement Engineer, Project Reporter **9.Resources/References**  
[1] https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/  
[2] <https://www.youtube.com/watch?v=iEm1NRyEe5c>  
[3] <https://asecuritysite.com/calculators/huff>  
[4] <https://cmps-people.ok.ubc.ca/ylucet/DS/Huffman.html>  
[5] <https://www.cs.usfca.edu/~galles/visualization/RedBlack.html>  
[6] <https://www.eecs.umich.edu/courses/eecs380/ALG/red_black.html>  
[7] <https://www.researchgate.net/publication/342869509_A_revisited_representation_of_the_Red-black_tree>  
[8] <https://www.wscubetech.com/resources/dsa/red-black-tree>  
[9] <https://www.cs.usfca.edu/~galles/visualization/BTree.html>  
[10] <https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html>  
[11] <https://www.youtube.com/watch?v=K1a2Bk8NrYQ>  
[12] <https://www.youtube.com/watch?v=K1a2Bk8NrYQ>  
[13] <https://github.com/KhaledAshrafH/B-Tree>  
[14] <https://github.com/gyaikhom/huffman>  
[15] <https://github.com/bhrigu123/huffman-coding>  
[16] <https://github.com/DanielScocco/Simple-Huffman-Coding/blob/master/huffman.c>  
[17] <https://github.com/stanislavkozlovski/Red-Black-Tree>  
[18] https://github.com/zarif98sjs/RedBlackTree-An-Intuitive-Approach  
[19] https://github.com/anandarao/Red-Black-Tree  
[20] https://github.com/emilydolson/python-red-black-trees  
[21] https://github.com/Aayushjn/B-Tree  
[22] https://github.com/FriendlyCSharp/B-tree  
[23]  
https://owl.purdue.edu/owl/research\_and\_citation/ieee\_style/ieee\_general\_format.html  
[24] https://www.ieee.org/conferences/publishing/templates