Project Title :

Text Compression Using Huffman Coding Algorithm for Efficient Data Storage and Transmission

Course Code : 22ALT33

Course name : Design and Analysis of Algorithm

Project Members

- **Sanjay N** (23ALR087)
- **Rohan Prabakar V** (23ALR079)
- **Santhosh K** (23ALR089)

Problem Statement :

With the exponential growth of digital data, efficient storage and transmission have become critical. Text data, often filled with repetitive characters, provides an opportunity for compression. This project addresses the challenge by implementing the Huffman Coding algorithm to reduce text file sizes. It assigns shorter codes to frequent characters, ensuring efficient and lossless compression. The project includes both encoding and decoding mechanisms to verify accuracy and demonstrate enhanced storage and transmission efficiency.

Objective :

The primary aim is to implement the Huffman Coding algorithm to:

1. **Compress Text Data**: Assign optimal binary codes to characters based on their frequency.
2. **Reduce Storage Space**: Minimize the file size while retaining original information.
3. **Enhance Transmission Efficiency**: Facilitate faster and more efficient data transfer.

Through this project, we demonstrate the algorithm's real-world applications, such as:

- File storage optimization.
- Network data transmission.

- Multimedia compression.

Algorithm :

Input: Text
Steps:

1. Create a frequency dictionary for each character in the text.
2. Insert each character with its frequency into a min-heap.
3. While more than one node exists in the heap:
    - Remove the two nodes with the lowest frequencies.
    - Create a new node combining their frequencies and set it as their parent.
    - Insert the new node back into the heap.
4. Use a recursive function generate_code(node, current_code) to generate binary codes:
    - If at a leaf node, store the code.
    - Else, traverse left (add "0") or right (add "1").
5. Replace each character in the text with its Huffman code.
6. For decoding, traverse the Huffman tree to retrieve the original characters.

Output: Huffman codes, encoded text, and decoded text (to verify correctness).

Pseudocode :

1. **Input**: Text.
2. **Build Frequency Table**.
3. **Initialize Min-Heap**.
4. **Combine Nodes**:
    - Extract two minimum-frequency nodes.
    - Merge and reinsert into the heap.
5. **Generate Huffman Codes**:
    - Recursive traversal assigns binary codes.

6. **Encode**: Replace characters with Huffman codes.
7. **Decode**: Traverse the tree for original text reconstruction.

Implementation :

The project was implemented using a structured programming approach to ensure clarity and functionality. The source code is available on GitHub.

reference:

https://github.com/sanjayn23alr087/HUFFMAN-CODING-ALGORITHM-DAA-

---

Conclusion :

This project successfully demonstrates how Huffman Coding efficiently compresses text data. By assigning shorter binary codes to frequently occurring characters, the method achieves significant reductions in file size, ensuring faster data transmission. With its support for both encoding and decoding, the algorithm guarantees accurate data reconstruction. The practical applications of Huffman Coding in file compression and data transfer underscore its importance in optimizing digital data management.

Implementation :