

## 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 :

```
main.py +
1 import heapq
2 from collections import defaultdict, Counter
3
4 # Node class for the Huffman tree
5 class Node:
6     def __init__(self, char, freq):
7         self.char = char # character
8         self.freq = freq # frequency of the character
9         self.left = None # left child
10        self.right = None # right child
11
12    # Define the comparison operators for heapq
13    def __lt__(self, other):
14        return self.freq < other.freq
15
16    # Build the Huffman Tree
17    def build_huffman_tree(text):
18        # Count frequency of each character in the text
19        freq = Counter(text)
20
21        # Create a priority queue to hold the nodes
22        heap = [Node(char, freq) for char, freq in freq.items()]
23        heapq.heapify(heap)
24
25        # Build the Huffman Tree
26        while len(heap) > 1:
27            left = heapq.heappop(heap)
28            right = heapq.heappop(heap)
29            merged = Node(None, left.freq + right.freq)
30            merged.left = left
31            merged.right = right
32            heapq.heappush(heap, merged)
33
34        return heap[0]
35
36    # Generate Huffman codes from the tree
37    def generate_codes(root, current_code="", codes={}):
38        if root is None:
39            return
40        if root.char is not None:
41            codes[root.char] = current_code
42            generate_codes(root.left, current_code + "0", codes)
43            generate_codes(root.right, current_code + "1", codes)
44        return codes
45
46    # Encode the text
47    def huffman_encode(text, codes):
48        return ''.join([codes[char] for char in text])
49
50    # Decode the encoded text
51    def huffman_decode(encoded_text, root):
52        decoded_text = []
53        current = root
54        for bit in encoded_text:
55            current = current.left if bit == '0' else current.right
56            if current.char:
57                decoded_text.append(current.char)
58                current = root
59        return ''.join(decoded_text)
60
61    # Main function to perform Huffman coding
62    def huffman_coding(text):
63        # Step 1: Build Huffman Tree
64        root = build_huffman_tree(text)
65
66        # Step 2: Generate Huffman codes
67        codes = generate_codes(root)
68
69        # Step 3: Encode the text
70        encoded_text = huffman_encode(text, codes)
71
72        # Step 4: Decode the encoded text to verify correctness
73        decoded_text = huffman_decode(encoded_text, root)
74
75        return codes, encoded_text, decoded_text
76
77    # Example usage
78    text = input("Enter the String: ")
79    codes, encoded_text, decoded_text = huffman_coding(text)
80
81    print("Huffman Codes:", codes)
82    print("Encoded Text:", encoded_text)
83    print("Decoded Text:", decoded_text)
```