DAA Assignment 5

Ishaan Shaikh
231070063
SY CE

Q. Download books from the website in html, text, doc, and pdf format. Compress these books using Huffman coding technique. Find the compression ratio

Algorithm:

```
~~HuffmanCoding (file)~~ GenerateHeap (file):
// Input : Text file is set of characters with frequency
// Output : Compression ratio

    n = |file|
    Q = file
    for i = 1 to n-1
        allocate new node z
        z.left = x = extract_min(Q)
        z.right = y = extract_min(Q)
        z.freq = x.freq + y.freq
        Insert(Q, z)
return extract_min(Q)
```

```
generateCode (node, prefix = '', codebook = {})
//Input: The current node of heap, prefix of current
        node, list of codes
//Output: List of codes assigned in encoding
    if (node):
        if (node.char):
            codebook[node.char] = prefix
        generateCode (node.left, prefix + '0', codebook)
        generateCode (node.right, prefix + '1', codebook)
    return codebook
```

```
findRatio (original, encoded):
//Input: Original and encoded data
//Output: Compression ratio
        n1 = original.len * 8 // 8 bits
        n2 = encoded.len
        return n1/n2
```

```
Encoding (data):
// Input: data to be encoded
// Output: Encoded data and huffman codes
    freq = { }
    for char in data:
        freq [char] += 1

    huffman_tree = GenerateHeap (freq)
    huffman_codes = generate Code (huffman_tree)
    encoded_data = ''.join (huffman_codes [char] for char
                                                in data)
    return encoded_data, huffman_codes
```

Test cases:

Test Cases:

1) Input: "The quick brown fox jumps over the lazy dog"
Output: Compression ratio: 1.773195

2) Input: "I have a dream that one day this nation will rise up and live out the true meaning of its creed: "We hold these truths to be self-evident, that all men are created equal."
Output: compression ratio: 1.921787

3) Input: Roses are red,
Violets are blue,
Sugar is sweet,
And so are you.
Output: Compression ratio: 1.947565

4) Input: (Empty file)
Output: File is empty

5) Input: (Files with only line breaks)
Output: Compression ratio: None

6) Input: Invalid file path
Output: No such file or directory

Time Complexity:

Time Complexity:

Input size : n (file size)

Basic operation : Build a Huffman Tree and assign Huffman codes to each character

1) Counting frequency of each character

Let $C_{worst}(n)$ be the T.C

$$C_{worst}(n) = \sum_{i=1}^{n} 1 = n-1+1 = n$$

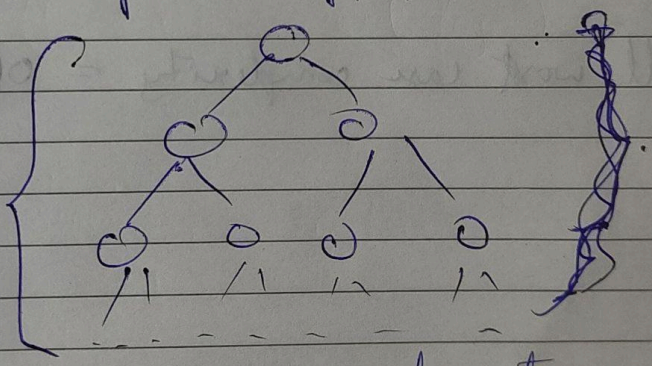
$\therefore$ T.C = $O(n)$

2) Building Huffman Tree

Let m be the no. of unique characters in the file of size n ($m \leq n$)

It involves push and pop operations in the heap

log m



There are m unique characters

$\therefore$ There will be m nodes in the Huffman Tree

~~$\therefore$ Tree will~~

$\therefore$ m elements will be pushed and popped from the tree

$\therefore$ T.C is $O(m \log m)$

3) Encoding each character
Each node is visited and there are m nodes
$\therefore O(m)$ is the T.C.

4) Encoding the given file
There are n characters in the file. Each one is replaced
$\therefore O(n)$ is the T.C

$\therefore$ Total T.C $= O(m) + O(n) + O(n\log n) + O(n)$

The worst case will occur when $m = n$
(no. of unique characters = no. of total characters)

$\therefore$ Total T.C $= O(n) + O(n) + O(n\log n) + O(n)$
$= O(n\log n) + 3\,O(n)$
~~$= O(n\log n)$~~

$\therefore$ Overall worst case complexity $= O(n\log n)$

<u>Program:</u> PEP 08 Coding style for python is used

```
import heapq
from collections import defaultdict
```

```python
class Node:
    """A node in the Huffman tree."""

    def __init__(self, char, freq):
        """Initialize a node with a character and frequency."""
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        """Less-than comparison for heapq."""
        return self.freq < other.freq

def build_huffman_tree(frequencies):
    """Builds the Huffman tree and returns the root node."""
    heap = [Node(char, freq) for char, freq in frequencies.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = Node(None, left.freq + right.freq)
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)

    return heap[0]  # The root of the Huffman tree

def generate_codes(node, prefix='', codebook=None):
    """Generates Huffman codes for the characters in the tree."""
    if codebook is None:
        codebook = {}
    if node is not None:
        if node.char is not None:
            codebook[node.char] = prefix
        generate_codes(node.left, prefix + '0', codebook)
        generate_codes(node.right, prefix + '1', codebook)
    return codebook
```

```python
def huffman_encoding(data):
    """Encodes the input string using Huffman coding."""
    if not isinstance(data, str):
        print("Input must be a string.")
        return None, None

    frequencies = defaultdict(int)
    for char in data:
        frequencies[char] += 1

    huffman_tree = build_huffman_tree(frequencies)
    huffman_codes = generate_codes(huffman_tree)

    encoded_data = ''.join(huffman_codes[char] for char in data)
    return encoded_data, huffman_codes

def calculate_compression_ratio(original, encoded):
    """Calculates the compression ratio."""
    original_size = len(original) * 8  # Assuming 8 bits per character
    compressed_size = len(encoded)  # Length of the encoded string
    if compressed_size == 0:
        print('Input file has no text')
        return None  # Handle division by zero
    return original_size / compressed_size

def run_huffman_test(file_path):
    """Runs Huffman encoding on the given file and prints results."""
    with open(file_path, 'r') as file:
        data = file.read()

    if len(data) == 0:
        print("File is empty")
        return

    encoded_data, huffman_codes = huffman_encoding(data)
    compression_ratio = calculate_compression_ratio(data, encoded_data)

    print(f"Test for file: {file_path}")
    print("Encoded Data:", encoded_data)
```

```
    print("Huffman Codes:", huffman_codes)
    print("Compression Ratio:", compression_ratio)
    print("-" * 40)

# Run tests on specified files
if __name__ == "__main__":
    run_huffman_test('test1.txt')
    run_huffman_test('test2.txt')
    run_huffman_test('test3.txt')
    run_huffman_test('test4.txt')
    run_huffman_test('test5.txt')
    run_huffman_test('test6.txt')
```

Output:

```
Test for file: test1.txt
Compression Ratio: 1.7731958762886597
Test for file: test2.txt
Compression Ratio: 1.9217877094972067
Test for file: test3.txt
Compression Ratio: 1.9475655430711611
Test for file: test4.txt
File is empty
Test for file: test5.txt
Input file has no text
Compression Ratio: None
Traceback (most recent call last):
  File "c:\Users\Ishaan\Desktop\PD lab\main.py", line 91, in <module>
    run_huffman_test('test6.txt')
  File "c:\Users\Ishaan\Desktop\PD lab\main.py", line 71, in run_huffman_test
    with open(file_path, 'r') as file:
         ^^^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'test6.txt'
```

Conclusion: Hence, we have studied the algorithm of Huffman coding. We have implemented the program greedy technique. Greedy technique gives less number of bits to the more frequent characters so that the encoded file is compressed to minimum size.