CMPSC 413 - Lab 5

Huffman Coding

Task 1: Design, analyze and implement the algorithm of computing Huffman code. Input: text containing any of 26 English characters. Example: ABAAABCBCCCDEFFFEE Output: Huffman codeword of each character.

```
Input_Text AAABCBCCCDEFFFEE
Encoded Text: 00000001110011101010010111110110110111111
Decoded Text: AAABCBCCCDEFFFEE
```

- Creating the frequency table: O(n), where n is the length of the input text.
- Building the priority queue: O(n log n) using a binary heap.
- Constructing the Huffman tree: O(n log n) using the priority queue.
- Generating codewords: O(n) since each character is processed only once.
- Total time complexity for Task 1: O(n log n).

Task 2: First encoding, and then, decoding a text file using the Huffman codeword (the output of the Task1). Input: a text file consists of the characters in Task 1 Output: Encoded the text file and decoded it back.

Encoding and Decoding both have a time complexity of O(k), where k is the length of the text or the encoded text.

`input.txt` :

```
Hi, my name is Dharmik Patel
```

OUTPUT

```
Encoded Text:
10101011101001110011100011100011010001101111101111010111100011000001010011001011100101110001001011001101100001101
Decoded Text: Hi, my name is Dharmik Patel
```

`encoded.txt` :

```
101010111010011100111000111000110100011011111011110101111000110000010100110010111001011100010010110011011000011011
```

`decoded.txt` :

```
Hi, my name is Dharmik Patel
```

Code:

```
import heapq
from collections import defaultdict

def build_frequency_table(text):
    frequency_table = defaultdict(int)
    for char in text:
        frequency_table[char] += 1
    return frequency_table

class HuffmanNode:
    def __init__(self, char, frequency):
        self.char = char
        self.frequency = frequency
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.frequency < other.frequency

def build_huffman_tree(frequency_table):
    priority_queue = [HuffmanNode(char, freq) for char, freq in frequency_table.items()]
    heapq.heapify(priority_queue)
    while len(priority_queue) > 1:
        left_node = heapq.heappop(priority_queue)
        right_node = heapq.heappop(priority_queue)

        parent_node = HuffmanNode(None, left_node.frequency + right_node.frequency)
        parent_node.left = left_node
        parent_node.right = right_node

        heapq.heappush(priority_queue, parent_node)

    return priority_queue[0]

def build_huffman_codes(node, current_code, huffman_codes):
    if node.char is not None:
        huffman_codes[node.char] = current_code
    if node.left is not None:
        build_huffman_codes(node.left, current_code + "0", huffman_codes)
    if node.right is not None:
        build_huffman_codes(node.right, current_code + "1", huffman_codes)

def huffman_encode(text):
    frequency_table = build_frequency_table(text)
    root = build_huffman_tree(frequency_table)

    huffman_codes = {}
    build_huffman_codes(root, "", huffman_codes)

    encoded_text = "".join(huffman_codes[char] for char in text)

    return encoded_text, huffman_codes

def huffman_decode(encoded_text, huffman_tree):
    decoded_text = ""
    current_node = huffman_tree

    for bit in encoded_text:
        if bit == "0":
            current_node = current_node.left
        elif bit == "1":
            current_node = current_node.right

        if current_node.char is not None:
            decoded_text += current_node.char
            current_node = huffman_tree

    return decoded_text

if __name__ == "__main__":
    input_text = "AAABCBCCCDEFFFEE"

    encoded_text, huffman_codes = huffman_encode(input_text)
    print("Encoded Text:", encoded_text)

    decoded_text = huffman_decode(encoded_text, huffman_codes)
    print("Decoded Text:", decoded_text)
    # Read input text from file
    with open("input.txt", "r") as file:
        input_text = file.read()

    # Encode the input text
    encoded_text, huffman_codes = huffman_encode(input_text)

    # Write the encoded text to an output file
    with open("encoded.txt", "w") as file:
        print("Encoded Text:", encoded_text)
        file.write(encoded_text)
```

```
# Decode the encoded text
frequency_table = build_frequency_table(input_text)
huffman_tree = build_huffman_tree(frequency_table)
decoded_text = huffman_decode(encoded_text, huffman_tree)

# Write the decoded text to an output file
with open("decoded.txt", "w") as file:
    print("Decoded Text:", decoded_text)
    file.write(decoded_text)
```