DAA – Assignment no: 02

Write a Python program to implement Huffman Encoding using a greedy strategy.

Input:

Assignment2.py

```python
class HuffmanNode:
    def __init__(self, symbol, frequency):
        self.symbol = symbol
        self.frequency = frequency
        self.left = None
        self.right = None

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


def build_huffman_tree(symbols, frequencies):

    nodes = []
    for symbol, frequency in zip(symbols, frequencies):
        nodes.append(HuffmanNode(symbol, frequency))

    while len(nodes) > 1:
        nodes.sort()
        left = nodes.pop(0)
        right = nodes.pop(0)
        new_node = HuffmanNode(None, left.frequency + right.frequency)
        new_node.left = left
        new_node.right = right
        nodes.append(new_node)

    return nodes[0]


```

Assignment2.py

Assignment2.py > ...

```python
def generate_huffman_codes(node, code=""):

    if node.symbol is not None:
        return {node.symbol: code}

    huffman_codes = {}
    huffman_codes.update(generate_huffman_codes(node.left, code + "0"))
    huffman_codes.update(generate_huffman_codes(node.right, code + "1"))
    return huffman_codes


def encode_huffman(message, huffman_codes):

    encoded_message = ""
    for symbol in message:
        encoded_message += huffman_codes[symbol]

    return encoded_message


def decode_huffman(encoded_message, huffman_tree):

    decoded_message = ""
    node = huffman_tree
    for bit in encoded_message:
        if bit == "0":
            node = node.left
        elif bit == "1":
            node = node.right

        if node.symbol is not None:
            decoded_message += node.symbol
            node = huffman_tree
```

Assignment2.py

```
    return decoded_message


def main():
    # Get the message to encode.
    message = input("Enter the message to encode: ")

    # Calculate the frequency of each symbol in the message.
    frequencies = {}
    for symbol in message:
        if symbol in frequencies:
            frequencies[symbol] += 1
        else:
            frequencies[symbol] = 1

    # Build the Huffman tree.
    huffman_tree = build_huffman_tree(list(frequencies.keys()), list(frequencies.values()))

    # Generate Huffman codes for the symbols in the Huffman tree.
    huffman_codes = generate_huffman_codes(huffman_tree)

    # Encode the message using Huffman encoding.
    encoded_message = encode_huffman(message, huffman_codes)

    # Print the encoded message.
    print("Encoded message:", encoded_message)

```

Assignment2.py

```
        else:
            frequencies[symbol] = 1

    # Build the Huffman tree.
    huffman_tree = build_huffman_tree(list(frequencies.keys()), list(frequencies.values()))

    # Generate Huffman codes for the symbols in the Huffman tree.
    huffman_codes = generate_huffman_codes(huffman_tree)

    # Encode the message using Huffman encoding.
    encoded_message = encode_huffman(message, huffman_codes)

    # Print the encoded message.
    print("Encoded message:", encoded_message)

    # Decode the encoded message using Huffman decoding.
    decoded_message = decode_huffman(encoded_message, huffman_tree)

    # Print the decoded message.
    print("Decoded message:", decoded_message)


if __name__ == "__main__":
    main()

```

Output:

```
PS D:\Tanmay Mohadikar\Sem 7 Practicals\DAA> & D:/Python/python.exe "d:/Tanmay Mohadikar/Sem 7 Practicals/DAA/Assignment2.py"
Enter the message to encode: 'BCAADDDCCACACAC'
Encoded message: 011010111010000000111110110111011011
Decoded message: 'BCAADDDCCACACAC'
PS D:\Tanmay Mohadikar\Sem 7 Practicals\DAA>
```