Case Study 9: Greedy Algorithms - Huffman Coding

6. Write a Python program to construct a Huffman tree and generate codes.

```
import heapq
from collections import defaultdict, Counter


class Node:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

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


def huffman_codes(text):
    # Count the frequency of each character in the text
    frequency = Counter(text)
    # Create a priority queue (min-heap) of nodes
    heap = [Node(char, freq) for char, freq in frequency.items()]
    heapq.heapify(heap)

    # Build the Huffman tree
    while len(heap) > 1:
```

```
        left = heapq.heappop(heap)   # Node with the smallest frequency
        right = heapq.heappop(heap)  # Node with the second smallest frequency
        merged = Node(None, left.freq + right.freq)  # Merge two nodes
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)

    root = heap[0]  # Root of the Huffman tree

    # Generate Huffman codes
    huffman_code = {}

    def generate_code(node, current_code):
        if node:
            if node.char is not None:  # Leaf node
                huffman_code[node.char] = current_code
            generate_code(node.left, current_code + "0")  # Traverse left
            generate_code(node.right, current_code + "1")  # Traverse right

    generate_code(root, "")
    return huffman_code

def encode_text(text, huffman_code):
    # Encode the text using the generated Huffman codes
    return ''.join(huffman_code[char] for char in text)
```

```
# Input the text to encode
text = input("Enter the text to encode: ")

# Generate Huffman codes
huffman_code = huffman_codes(text)

# Print Huffman codes
print("Huffman Codes:")
for char, code in huffman_code.items():
    print(f"'{char}': {code}")

# Encode and print the encoded text
encoded_text = encode_text(text, huffman_code)
print("\nEncoded Text:")
print(encoded_text)
```