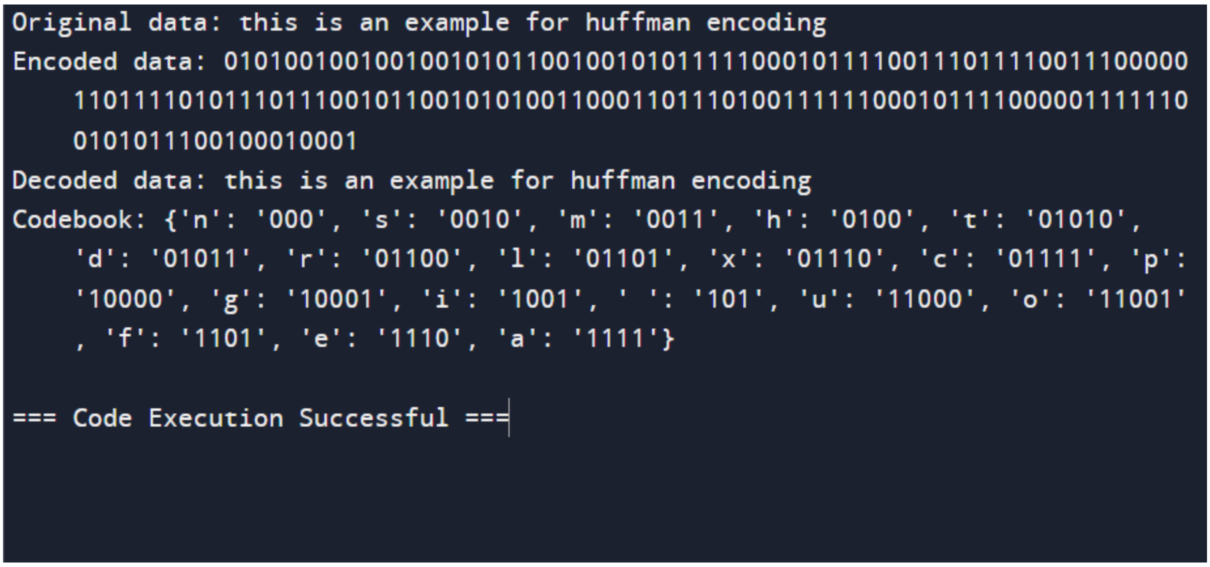
92. Optimal Tree Problem: Huffman Trees and Codes

PROGRAM:-

```
import heapq
from collections import Counter

class Node:
   def __init__(self, char=None, freq=0):
      self.char = char
      self.freq = freq
      self.left = None
      self.right = None

   # Define comparison operators for priority queue
   def __lt__(self, other):
      return self.freq < other.freq

def build_huffman_tree(frequencies):
   # Create a priority queue (min-heap) from the frequency dictionary
   heap = [Node(char, freq) for char, freq in frequencies.items()]
   heapq.heapify(heap)

   # Merge nodes until only one tree remains
   while len(heap) > 1:
      left = heapq.heappop(heap)
      right = heapq.heappop(heap)
      merged = Node(freq=left.freq + right.freq)
      merged.left = left
      merged.right = right
      heapq.heappush(heap, merged)

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

def build_codes(node, prefix="", codebook={}):
   if node.char is not None:
      # It's a leaf node, add it to the codebook
      codebook[node.char] = prefix
   else:
      # Traverse the left and right children
      if node.left:
         build_codes(node.left, prefix + "0", codebook)
      if node.right:
         build_codes(node.right, prefix + "1", codebook)
   return codebook

def huffman_encoding(data):
   # Count the frequency of each character in the data
   frequencies = Counter(data)

   # Build the Huffman tree
   huffman_tree = build_huffman_tree(frequencies)
```

```
    # Build the codes from the Huffman tree
    codebook = build_codes(huffman_tree)

    # Encode the data
    encoded_data = ''.join(codebook[char] for char in data)
    return encoded_data, codebook

def huffman_decoding(encoded_data, codebook):
    # Build the inverse codebook
    inverse_codebook = {v: k for k, v in codebook.items()}

    # Decode the data
    decoded_data = []
    current_code = ""
    for bit in encoded_data:
        current_code += bit
        if current_code in inverse_codebook:
            decoded_data.append(inverse_codebook[current_code])
            current_code = ""
    return ''.join(decoded_data)

# Example usage:
data = "this is an example for huffman encoding"
encoded_data, codebook = huffman_encoding(data)
decoded_data = huffman_decoding(encoded_data, codebook)

print("Original data:", data)
print("Encoded data:", encoded_data)
print("Decoded data:", decoded_data)
print("Codebook:", codebook)
```

OUTPUT:-



```
Original data: this is an example for huffman encoding
Encoded data: 0101001001001001010110010010101111100010111100111011110011100000
    11011110101110111001011001010100110001101110100111111000101111000001111110
    0101011100100010001
Decoded data: this is an example for huffman encoding
Codebook: {'n': '000', 's': '0010', 'm': '0011', 'h': '0100', 't': '01010',
    'd': '01011', 'r': '01100', 'l': '01101', 'x': '01110', 'c': '01111', 'p':
    '10000', 'g': '10001', 'i': '1001', ' ': '101', 'u': '11000', 'o': '11001'
    , 'f': '1101', 'e': '1110', 'a': '1111'}

=== Code Execution Successful ===
```

TIME COMPLEXITY:-O(n logn)