In [1]:

```
import heapq
from collections import defaultdict

class HuffmanNode:
    def __init__(self, symbol, freq):
        self.symbol = symbol
        self.freq = freq
        self.left = None
        self.right = None

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

def build_huffman_tree(freq_dict):
    priority_queue = [HuffmanNode(sym, freq) for sym, freq in freq_dict.items()]
    heapq.heapify(priority_queue)

    while len(priority_queue) > 1:
        left_node = heapq.heappop(priority_queue)
        right_node = heapq.heappop(priority_queue)
        merged_node = HuffmanNode(None, left_node.freq + right_node.freq)
        merged_node.left = left_node
        merged_node.right = right_node
        heapq.heappush(priority_queue, merged_node)

    return priority_queue[0]

def build_huffman_codes(node, current_code, huffman_codes):
    if node is None:
        return

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

def huffman_encoding(data):
    freq_dict = defaultdict(int)
    for symbol in data:
        freq_dict[symbol] += 1

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

    encoded_data = "".join(huffman_codes[symbol] for symbol in data)
    return encoded_data, root

def huffman_decoding(encoded_data, root):
    decoded_data = ""
    current_node = root

    for bit in encoded_data:
        if bit == "0":
            current_node = current_node.left
        else:
            current_node = current_node.right
```

```
            if current_node.symbol is not None:
                decoded_data += current_node.symbol
                current_node = root

        return decoded_data

if __name__ == "__main__":
    # Example usage
    data = "this is an example for huffman encoding"

    encoded_data, huffman_tree = huffman_encoding(data)
    print("Encoded data:", encoded_data)

    decoded_data = huffman_decoding(encoded_data, huffman_tree)
    print("Decoded data:", decoded_data)
```

```
Encoded data: 0101001001001001010111001001010111110001011110011101111001110000011011110101
1110111001011001010100110001101110100111111000101111000001111110010101110010000100001
Decoded data: this is an example for huffman encoding
```

In [ ]: