

201: Given a Huffman Tree and a Huffman encoded string, decode the string to get the original message.

Program:

```
from heapq import heappush, heappop, heapify
```

```
class Node:
```

```
    def __init__(self, char, freq):
```

```
        self.char = char
```

```
        self.freq = freq
```

```
        self.left = None
```

```
        self.right = None
```

```
def build_huffman_tree(characters, frequencies):
```

```
    nodes = []
```

```
    for i in range(len(characters)):
```

```
        nodes.append(Node(characters[i], frequencies[i]))
```

```
    while len(nodes) > 1:
```

```
        nodes = sorted(nodes, key=lambda x: x.freq)
```

```
        left = nodes.pop(0)
```

```
        right = nodes.pop(0)
```

```
        parent = Node(None, left.freq + right.freq)
```

```
        parent.left = left
```

```
        parent.right = right
```

```
        nodes.append(parent)
```

```
    return nodes[0]
```

```
def huffman_decode(root, encoded_string):
```

```
    decoded_string = ""
```

```
    current = root
```

```
    for bit in encoded_string:
```

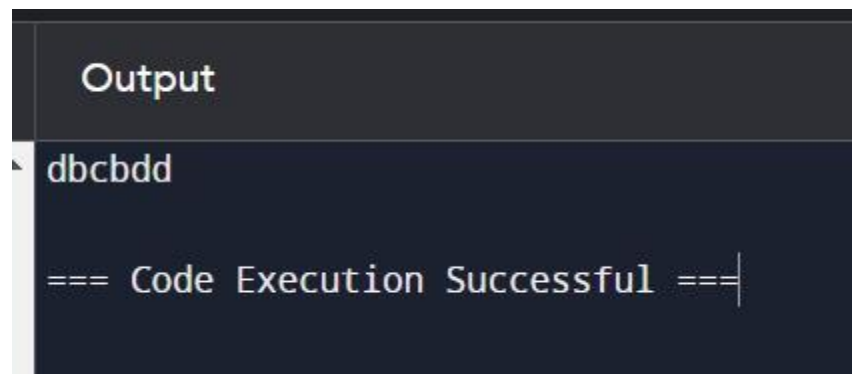
```
        if bit == '0':
```

```
        current = current.left
    else:
        current = current.right
    if current.char:
        decoded_string += current.char
        current = root
    return decoded_string

# Test Case 1

characters1 = ['a', 'b', 'c', 'd']
frequencies1 = [5, 9, 12, 13]
encoded_string1 = '1101100111110'
root1 = build_huffman_tree(characters1, frequencies1)
decoded_message1 = huffman_decode(root1, encoded_string1)
print(decoded_message1)
```

Output:

A screenshot of a code execution environment. It features a dark-themed window with a title bar that says "Output". Inside the window, the text "dbcbdd" is displayed on the first line. On the second line, the text "=== Code Execution Successful ===" is shown, with a vertical cursor at the end of the line.

```
Output
dbcbdd
=== Code Execution Successful ===
```

Time complexity: $O(m)$