5.8 DECODING A HUFFMAN ENCODED STRING USING A HUFFMAN TREE

Question:

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

AIM

To traverse the Huffman Tree using the bits of the encoded string and reconstruct the original message.

ALGORITHM

1. Start at the **root** of the Huffman Tree.
2. For each bit in the encoded string:
    - If the bit is '0', move to the left child.
    - If the bit is '1', move to the right child.
3. When a leaf node is reached (i.e., a node with a character), append the character to the result and reset to the root.
4. Continue until all bits are processed.

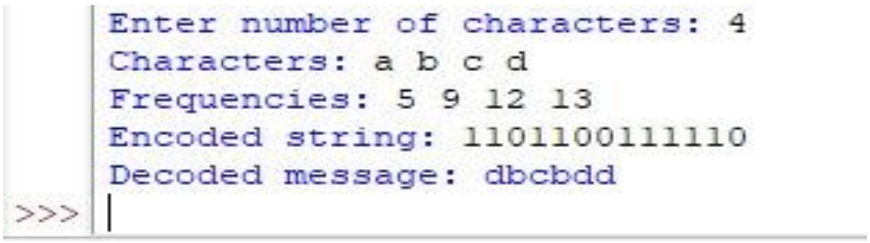
PROGRAM

```
import heapq
class Node:
    def __init__(self, char=None, freq=0):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None
    def __lt__(self, other):
        return self.freq < other.freq

def build_huffman_tree(chars, freqs):
    heap = [Node(c, f) for c, f in zip(chars, freqs)]
    heapq.heapify(heap)
    while len(heap) > 1:
        a = heapq.heappop(heap)
        b = heapq.heappop(heap)
        merged = Node(None, a.freq + b.freq)
        merged.left = a
        merged.right = b
        heapq.heappush(heap, merged)
    return heap[0]

def decode(root, encoded):
    result = ''
    node = root
    for bit in encoded:
        if bit == '0':
            node = node.left
        else:
            node = node.right
        if node.char is not None:
            result += node.char
            node = root
    return result

n = int(input("Enter number of characters: "))
chars = input("Characters: ").split()
freqs = list(map(int, input("Frequencies: ").split()))
encoded = input("Encoded string: ")
root = build_huffman_tree(chars, freqs)
print("Decoded message:", decode(root, encoded))
```

Input:

Enter number of characters : 4
Characters: a b c d
Frequencies: 5 9 12 13
Encoded string: 1101100111110
Decoded message: dbcbdd

Output:



RESULT:

Thus the program is successfully executed and the output is verified.

PERFORMANCE ANALYSIS:

· Time Complexity: O(n)

· Space Complexity: O(1)