5.7 HUFFMAN CODING FOR CHARACTER COMPRESSION

Question:

Given a set of characters and their corresponding frequencies, construct the Huffman Tree and generate the Huffman Codes for each character.

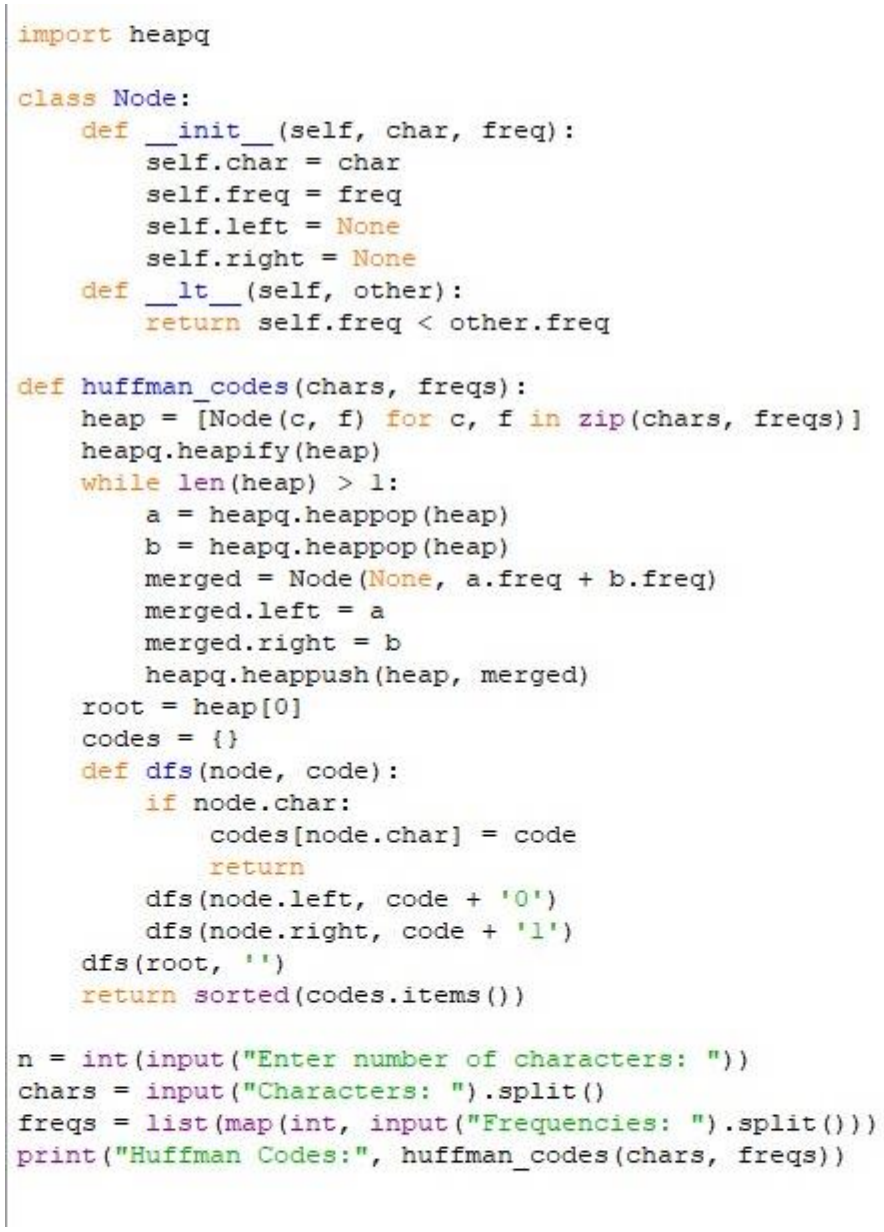
AIM

To implement Huffman Coding using a priority queue and binary tree structure, and generate prefix-free binary codes for each character.

ALGORITHM

1. Create a min-heap (priority queue) of nodes, each containing a character and its frequency.
2. While there is more than one node in the heap:
    - Extract the two nodes with the lowest frequency.
    - Create a new internal node with these two as children and frequency equal to their sum.
    - Insert the new node back into the heap.
3. The remaining node is the root of the Huffman Tree.
4. Traverse the tree to assign binary codes:
    - Left edge → append '0'
    - Right edge → append '1'
5. Store the codes in a dictionary.

PROGRAM



```python
import heapq

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(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)
    root = heap[0]
    codes = {}
    def dfs(node, code):
        if node.char:
            codes[node.char] = code
            return
        dfs(node.left, code + '0')
        dfs(node.right, code + '1')
    dfs(root, '')
    return sorted(codes.items())

n = int(input("Enter number of characters: "))
chars = input("Characters: ").split()
freqs = list(map(int, input("Frequencies: ").split()))
print("Huffman Codes:", huffman_codes(chars, freqs))
```

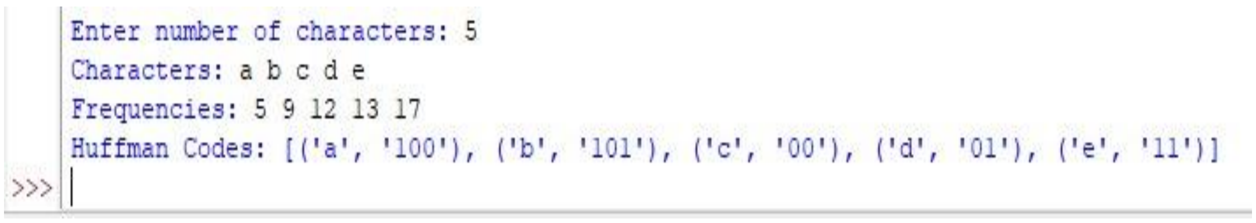
Input:

Enter number of character: 5

Characters: a b c d e

Frequencies: 5 9 12 13 17

Output:



```
Enter number of characters: 5
Characters: a b c d e
Frequencies: 5 9 12 13 17
Huffman Codes: [('a', '100'), ('b', '101'), ('c', '00'), ('d', '01'), ('e', '11')]
>>> |
```

RESULT:

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

PERFORMANCE ANALYSIS:

- Time Complexity: O(n log n), where *n* is the number of unique characters.
- Space Complexity: O(n), for storing the tree and codes.