NAME: Atharva Kangralkar

ROLL NO: 54

CLASS: TY CS(AIML)

PRN NO: 12311493

BATCH: 2

LAB ASSIGNMENT 5

Aim: Huffman coding

Step 1: Pseudo code

```
CLASS Node:
    freq
    char
    left, right

    CONSTRUCTOR(frequency, character):
        freq = frequency
        char = character
        left = NULL
        right = NULL

CLASS Compare:
    FUNCTION operator(a, b):
        RETURN (a.freq > b.freq)

FUNCTION preorder(root, temp, ans):
    IF root IS NULL:
        RETURN
    IF root.left IS NULL AND root.right IS NULL:
```

```
        ans.ADD(temp)
        RETURN
    temp.APPEND('0')
    preorder(root.left, temp, ans)
    temp.REMOVE_LAST()
    temp.APPEND('1')
    preorder(root.right, temp, ans)
    temp.REMOVE_LAST()

FUNCTION huffmanCodes(S, freq):
    N = LENGTH(S)
    CREATE minHeap
    FOR i FROM 0 TO N-1:
        minHeap.PUSH(new Node(freq[i], S[i]))
    WHILE size(minHeap) > 1:
        first  = minHeap.POP()
        second = minHeap.POP()
        root = new Node(first.freq + second.freq, '$')
        root.left = first
        root.right = second
        minHeap.PUSH(root)
    root = minHeap.POP()
    ans = EMPTY LIST
    temp = EMPTY STRING
    preorder(root, temp, ans)
    RETURN ans

MAIN:
    PRINT "Enter characters (no spaces):"
```

```
READ S
N = LENGTH(S)
freq = ARRAY[N]
PRINT "Enter frequency for each character:"
FOR i FROM 0 TO N-1:
    PRINT S[i] + ": "
    READ freq[i]
codes = huffmanCodes(S, freq)
PRINT "Huffman Codes:"
FOR i FROM 0 TO N-1:
    PRINT S[i] + ": " + codes[i]
```

Step2: Code

```
#include <iostream>
#include <vector>
#include <queue>
#include <string>
using namespace std;

class Solution {
public:
    class Node {
    public:
        int freq;
        char c;
        Node *left, *right;
        Node(int frequency, char name) {
            freq = frequency;
            c = name;
```

```
        left = right = NULL;
    }
};

class comp {
public:
    bool operator()(Node *a, Node *b) {
        return a->freq > b->freq;
    }
};

void preorder(Node *root, string &temp, vector<string> &ans) {
    if (!root) return;
    if (!root->left && !root->right) {
        ans.push_back(temp);
        return;
    }
    temp.push_back('0');
    preorder(root->left, temp, ans);
    temp.pop_back();
    temp.push_back('1');
    preorder(root->right, temp, ans);
    temp.pop_back();
}

vector<string> huffmanCodes(string S, vector<int> f) {
    priority_queue<Node*, vector<Node*>, comp> pq;
    int N = S.size();
    for (int i = 0; i < N; i++)
```

```
            pq.push(new Node(f[i], S[i]));
        while (pq.size() > 1) {
            Node *first = pq.top(); pq.pop();
            Node *second = pq.top(); pq.pop();
            Node *root = new Node(first->freq + second->freq, '$');
            root->left = first;
            root->right = second;
            pq.push(root);
        }
        Node *root = pq.top(); pq.pop();
        vector<string> ans;
        string temp;
        preorder(root, temp, ans);
        return ans;
    }
};

int main() {
    string S;
    cout << "Enter characters (no spaces): ";
    cin >> S;

    int N = S.size();
    vector<int> f(N);
    cout << "Enter frequency for each character:\n";
    for (int i = 0; i < N; i++) {
        cout << S[i] << ": ";
        cin >> f[i];
    }
```

```
    Solution sol;
    vector<string> codes = sol.huffmanCodes(S, f);
    cout << "\nHuffman Codes:\n";
    for (int i = 0; i < N; i++) {
        cout << S[i] << ": " << codes[i] << "\n";
    }
    return 0;
}
```

Step3: Output

```
PS C:\TY CS(AIML)\DAA\Lab Codes> cd "c:\TY CS(AIML)\DAA\Lab Codes\" ; if ($?) { g++ huffmann_coding.cpp -o huffmann_coding } ;
 if ($?) { .\huffmann_coding }
Enter characters (no spaces): abc
Enter frequency for each character:
a: 4
b: 2
c: 10

Huffman Codes:
a: 00
b: 01
c: 1
```

```
PS C:\TY CS(AIML)\DAA\Lab Codes> cd "c:\TY CS(AIML)\DAA\Lab Codes\" ; if ($?) { g++ huffmann_coding.cpp -o huffmann_coding } ;
 if ($?) { .\huffmann_coding }
Enter characters (no spaces): abcdef
Enter frequency for each character:
a: 16
b: 4
c: 20
d: 90
e: 10
f: 6

Huffman Codes:
a: 000
b: 0010
c: 0011
d: 010
e: 011
f: 1
```

Step4: Time complexity Analysis

1)Insert characters into priority queue

- You push N nodes into a min-heap.
- Each push is O(log N).
- tc = O(N log N).

2)Build Huffman Tree

- While heap size > 1:
    - Pop 2 nodes (O(log N) each).
    - Create new node (O(1)).
    - Push new node back (O(log N)).
- This runs (N - 1) times.
- tc = O(N log N).

3)Preorder Traversal to Generate Codes

- Visit each node exactly once.
- tc = O(N).

TC = O(N log N) + O(N log N) + O(N)

Therefore, TC = **O(N log N)**