Name: Ankush Singh
Group: 5C13
Roll No.: 040

Date: 03-10-2023

EXPERIMENT- 3

Aim:

To implement Huffman Coding and analyse its time complexity.

Code:

```
#include <bits/stdc++.h>

using namespace std;

struct HuffmanNode {
    char data;
    int frequency;
    HuffmanNode* left;
    HuffmanNode* right;

    HuffmanNode(char data, int frequency) {
        this->data = data;
        this->frequency = frequency;
        left = right = nullptr;
    }
};

struct CompareNodes {
    bool operator()(HuffmanNode* a, HuffmanNode* b) {
        return a->frequency > b->frequency;
    }
};

HuffmanNode* buildHuffmanTree(map<char, int>& frequencies) {
    priority_queue<HuffmanNode*, vector<HuffmanNode*>, CompareNodes> minHeap;

    for (auto pair : frequencies) {
        minHeap.push(new HuffmanNode(pair.first, pair.second));
    }

    while (minHeap.size() > 1) {
        HuffmanNode* left = minHeap.top();
        minHeap.pop();
        HuffmanNode* right = minHeap.top();
        minHeap.pop();

        HuffmanNode* mergedNode = new HuffmanNode('\0', left->frequency + right-
>frequency);
        mergedNode->left = left;
        mergedNode->right = right;

        minHeap.push(mergedNode);
    }

    return minHeap.top();
}

void generateHuffmanCodes(HuffmanNode* root, string code, map<char, string>&
huffmanCodes) {
    if (!root)
        return;

    if (root->data != '\0') {
        huffmanCodes[root->data] = code;
    }

    generateHuffmanCodes(root->left, code + "0", huffmanCodes);
    generateHuffmanCodes(root->right, code + "1", huffmanCodes);
}

int main() {
    map<char, int> frequencies;
    int num_characters;

    cout << "Enter the number of characters: ";
    cin >> num_characters;

    for (int i = 0; i < num_characters; i++) {
        char character;
        int frequency;

        cout << "Enter character " << i + 1 << ": ";
        cin >> character;

        cout << "Enter frequency for character " << character << ": ";
        cin >> frequency;

        frequencies[character] = frequency;
    }
    clock_t start_time = clock();

    HuffmanNode* root = buildHuffmanTree(frequencies);

    map<char, string> huffmanCodes;
    generateHuffmanCodes(root, "", huffmanCodes);

    cout << "Huffman Codes:" << endl;
    for (auto pair : huffmanCodes) {
        cout << pair.first << ": " << pair.second << endl;
    }

    clock_t end_time = clock();
    double execution_time = (double)(end_time - start_time) * 1000.0 /
CLOCKS_PER_SEC;
    cout << "\nExecution time: " << execution_time << " milliseconds" << endl;

    return 0;
}
```

Output:

```
Enter the number of characters: 7
Enter character 1: a
Enter frequency for character a: 10
Enter character 2: e
Enter frequency for character e: 15
Enter character 3: i
Enter frequency for character i: 12
Enter character 4: o
Enter frequency for character o: 3
Enter character 5: u
Enter frequency for character u: 4
Enter character 6: s
Enter frequency for character s: 13
Enter character 7: t
Enter frequency for character t: 1
Huffman Codes:
a: 111
e: 10
i: 00
o: 11011
s: 01
t: 11010
u: 1100

Execution time: 0.073 milliseconds
```