TIPSEB1326
72017912C
ARYAN DEEPAK PAWAR

GROUP A : ASSIGNMENT No. 2

Page No.
Date

Title: Huffman Encoding

Objective: To implement Huffman Encoding using a greedy strategy.

Problem Statement:
Write a program to implement Huffman Encoding using a greedy strategy.

Software & Hardware Requirement:
1. Any Operating System
2. Desktop / Laptop
3. Compiler / Interpreter
4. IDE (any)

Theory:

Huffman Coding:
1) Huffman coding is a lossless data compression algorithm
2) The idea is to design / assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters.
3) The most frequent character gets the smallest code & the least frequent character gets the largest code.
4) The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that

the code assigned to one characters is not the prefix of code assigned to any other character.

5) This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.

Greedy Algorithm:

1) Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious & immediate benifit.

2) Problems where choosing locally optimal also leads to global solution are the best fit for greedy.

3) A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment.

4) It doesn't worry whether the current best will result will bring the overall optimal result.

5) The algorithm never reverses the earlier decision even if the choice is wrong.

6) It works in top-down approach.

7) This algorithm may not produce the best result for all the problems. It's because it always goes for the local best choice to produce the global best result.

Page No.
Date

Algorithm:

Step 1: Calculate the frequency of each character in the string.

Step 2: Sort the characters in increasing order of the frequency. These are stored in a priority queue Q.

Step 3: Make each unique character as a leaf node.

Step 4: Create an empty node z. Assign the min. freq. to the left child of z & assign the second min. freq. to the right child of z. Set the value of z as the sum of the above two min. freq.

Step 5: Remove these two min. freq. from Q & add the sum into the list of freq.

Step 6: Insert node z into the tree.

Step 7: Repeat step 3 to 5 for all the characters.

Step 8: For each non-leaf node, assign 0 to the left edge & 1 to the right edge.

Mathematical Model:

Time Complexity:

$O(n \log n)$ where n is the no. of unique characters. ~~If there are n nodes,~~

If the input array is sorted, there exists a linear time algorithm.

Page No.
Date

Test Cases:

No.	Input	Expected Output	Actual Output	Result
1.	{'a', 'b', 'c', 'd', 'e', 'f'} {5, 9, 12, 13, 16, 45}	f = 0 c = 100 d = 101 a = 1100 b = 1101 e = 111	f = 0 c = 100 d = 101 a = 1100 b = 1101 e = 111	Pass
2.	{'a', 'b', 'c', 'd'} {5, 1, 6, 3}	a = 11 b = 100 c = 0 d = 101	a = 11 b = 100 c = 0 d = 101	Pass

Conclusion: Successfully implemented Huffman Encoding using a greedy strategy.

28/9/22

Code:

```
#include <bits/stdc++.h>
using namespace std;

struct MinHeapNode {

    char data;

    unsigned freq;

    MinHeapNode *left, *right;

    MinHeapNode(char data, unsigned freq)
    {
        left = right = NULL;
        this->data = data;
        this->freq = freq;
    }
};

struct compare {

    bool operator()(MinHeapNode* l, MinHeapNode* r)

    {
        return (l->freq > r->freq);
    }
};

void printCodes(struct MinHeapNode* root, string str)
{

    if (!root)
        return;

    if (root->data != '$')
        cout << root->data << ": " << str << "\n";

    printCodes(root->left, str + "0");
    printCodes(root->right, str + "1");
}

void HuffmanCodes(char data[], int freq[], int size)
{
    struct MinHeapNode *left, *right, *top;

    priority_queue<MinHeapNode*, vector<MinHeapNode*>, compare> minHeap;

    for (int i = 0; i < size; ++i)
        minHeap.push(new MinHeapNode(data[i], freq[i]));

    while (minHeap.size() != 1) {
```

```
        left = minHeap.top();
        minHeap.pop();

        right = minHeap.top();
        minHeap.pop();

        top = new MinHeapNode('$', left->freq + right->freq);

        top->left = left;
        top->right = right;

        minHeap.push(top);
    }

    printCodes(minHeap.top(), "");
}

int main()
{

    char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
    int freq[] = { 5, 9, 12, 13, 16, 45 };

    int size = sizeof(arr) / sizeof(arr[0]);

    HuffmanCodes(arr, freq, size);

    return 0;
}
```

Output:

```
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
```