

SEPTEMBER 29,2020

DESIGN AND ANALYSIS OF ALGORITHMS

(QUESTION: 5.2, HUFFMAN CODE)
EXERCISE 5

SUBMITTED BY-

NAME - RAJ KRISHNA

ROLL NO - 1805233

BATCH - CSE-G1

GROUP - B

1. The objective of the Experiment

The objective of the experiment is to **compress the size of the strings** to be sent over a network **without losing any data** by **Huffman coding technique**.

2. Solution Code

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

struct MinHeapNode {
    char data;
    unsigned int 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 != 'N')
        cout << root->data << ": " << str << " ";
```

```

        printCodes(root->left, str + "0");
        printCodes(root->right, str + "1");
    }
void HuffmanCodes(char data[], int freq[], int size)
{
    struct MinHeapNode *l, *r, *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) {
        l = minHeap.top();
        minHeap.pop();
        r = minHeap.top();
        minHeap.pop();
        top = new MinHeapNode('N', l->freq + r->freq);
        top->left = l;
        top->right = r;
        minHeap.push(top);
    }
    printCodes(minHeap.top(), "");
}
int main()
{
    char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
    int freq[] = { 45, 13, 12, 16, 9, 5 };
    int size = sizeof(arr) / sizeof(arr[0]);
    cout<<"Huffman Codes- "<<endl;
    HuffmanCodes(arr, freq, size);
    cout<<endl;
}

```

```
    return 0;  
}
```

3. Summary of the program

Huffman Coding is a technique of **compressing data** to reduce its size without losing any of the details. In this algorithm, a variable-length code is assigned to input different characters. The code length is related to how frequently characters are used. Most frequent characters have the smallest codes and longer codes for least frequent characters.

There are mainly **two parts**. First one to create a **Huffman tree**, and another one to **traverse the tree to find codes**.

Steps to build Huffman Tree

Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of the frequency field is used to compare two nodes in a min heap.)
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps 2 and 3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

Steps to print codes from Huffman Tree

Traverse the tree formed starting from the root. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.

Time Complexity-

The time complexity is $O(n \log n)$.

Extracting minimum frequency from the priority queue takes place $2*(n-1)$ times and its complexity is $O(\log n)$. Thus the overall complexity is $O(n \log n)$

4. Sample Output

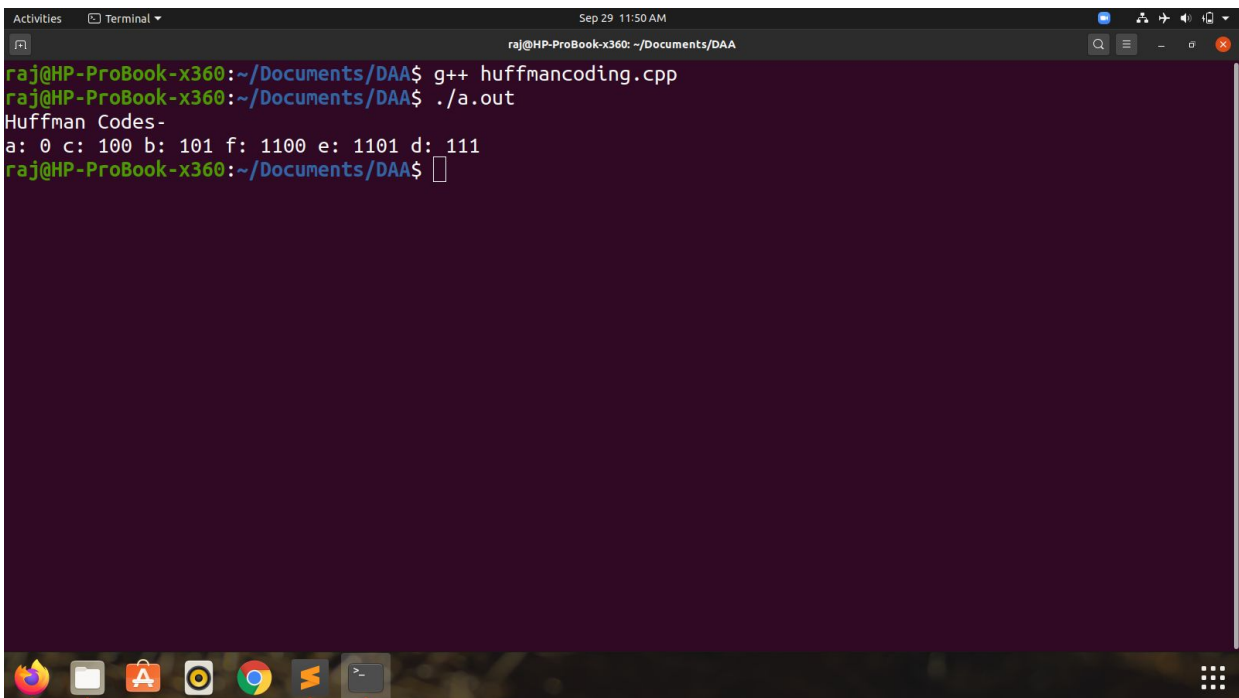
Input-

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

Output-

Huffman Codes-

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



```
raj@HP-ProBook-x360: ~/Documents/DAA$ g++ huffmancoding.cpp  
raj@HP-ProBook-x360: ~/Documents/DAA$ ./a.out  
Huffman Codes -  
a: 0 c: 100 b: 101 f: 1100 e: 1101 d: 111  
raj@HP-ProBook-x360: ~/Documents/DAA$
```