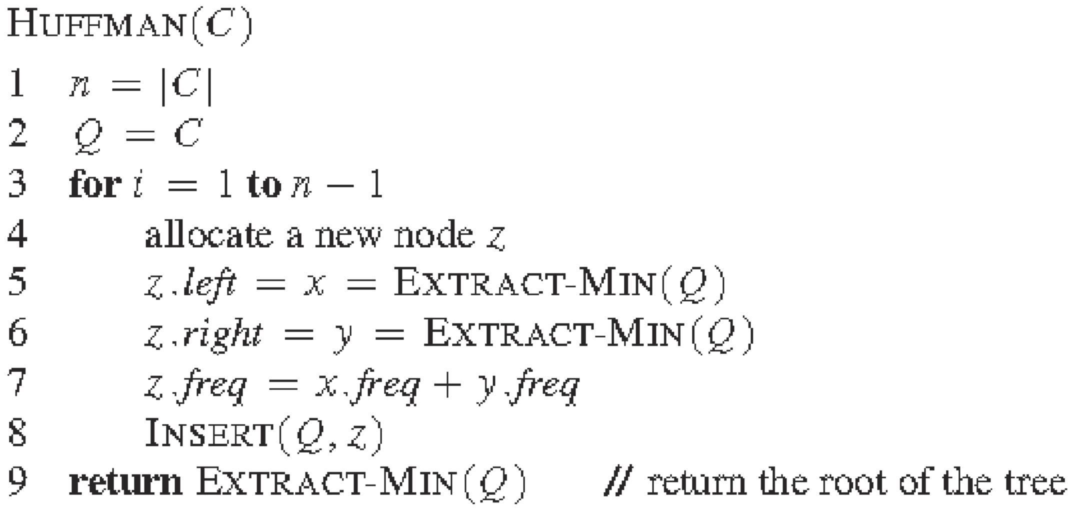
Experiment No. : 4

Title: Implement Huffman Algorithm using Greedy approach

Batch:A2 **Roll No.: 16010421063** **Experiment No.: 4**

Aim: To Implement Huffman Algorithm using Greedy approach and analyse its time Complexity.

Algorithm of Huffman Algorithm: Refer Coreman for Explaination



```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    z.left = x = EXTRACT-MIN(Q)
    z.right = y = EXTRACT-MIN(Q)
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)    // return the root of the tree
```

Explanation and Working of Variable Length Huffman Algorithm:

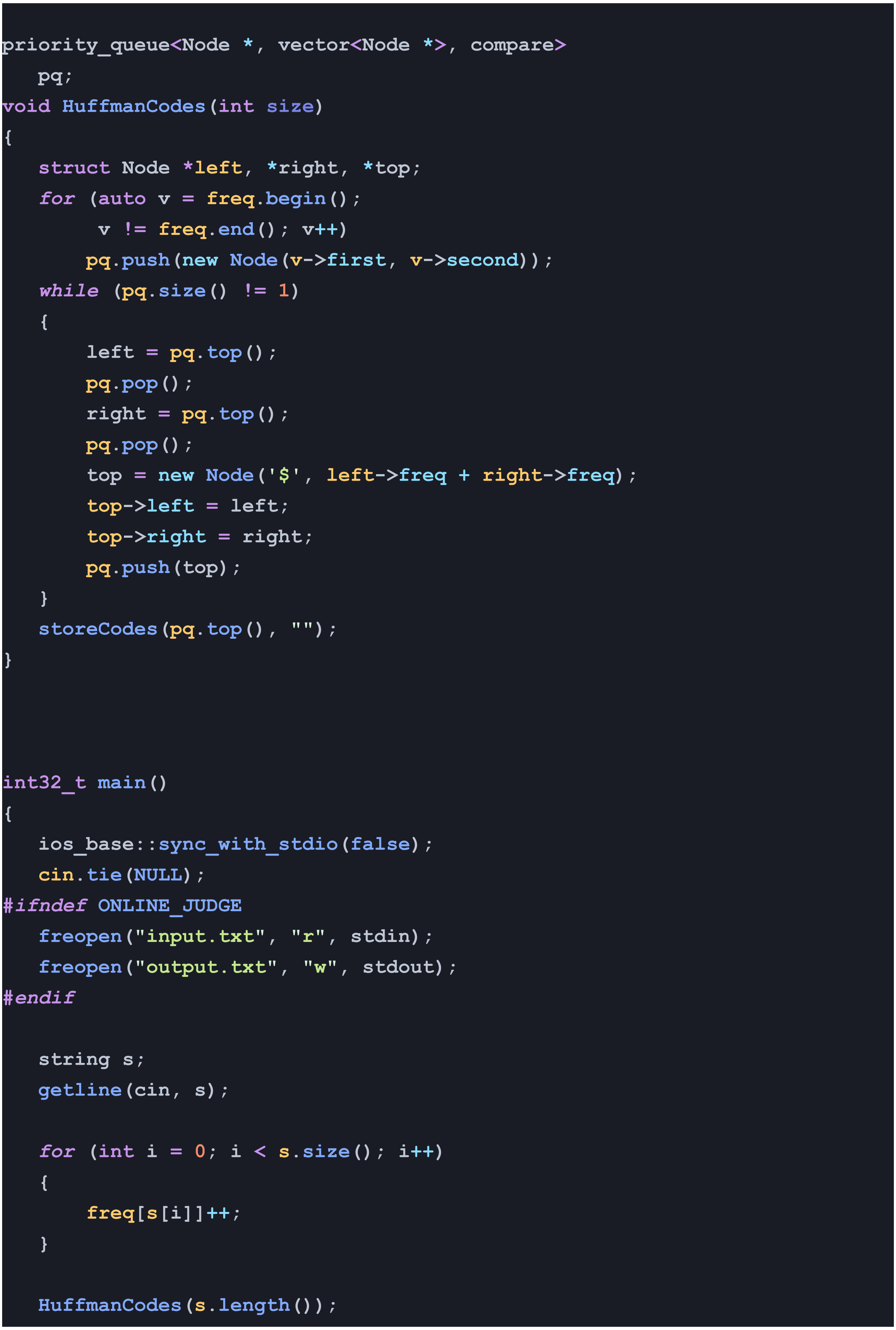
Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters; lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code. 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 character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bit stream.

Derivation of Huffman Algorithm:

Time complexity Analysis
The time complexity of the Huffman algorithm is O(nlogn). Using a heap to store the weight of each tree, each iteration requires O(logn) time to determine the cheapest weight and insert the new weight. There are O(n) iterations, one for each item.

Program(s) of Huffman Algorithm:

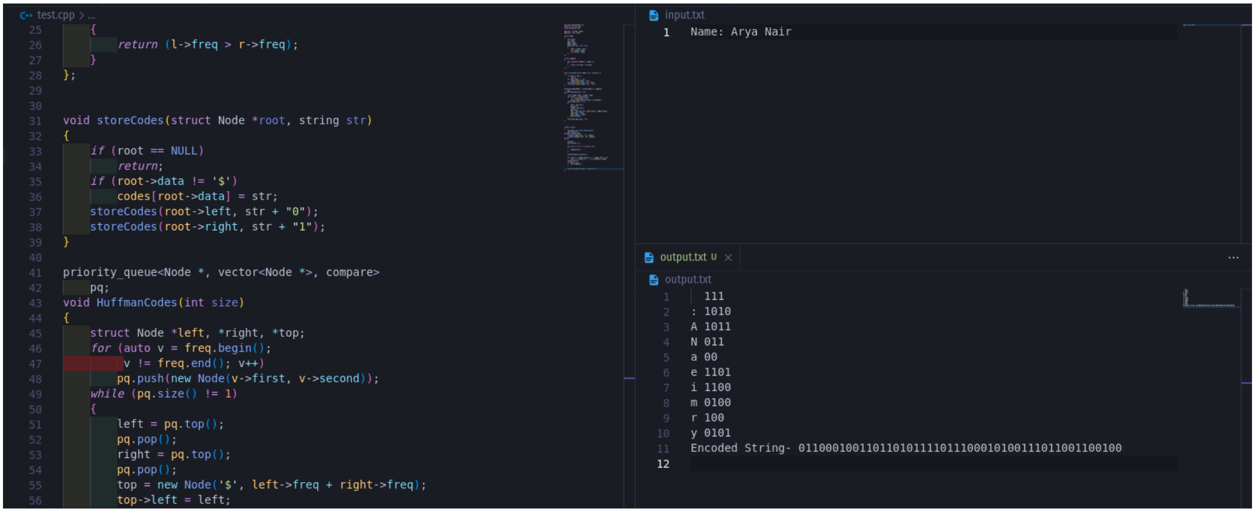
```
#include <bits/stdc++.h>
#define int long long
using namespace std;

map<char, string> codes;
map<char, int> freq;

struct Node
{
   char data;
   int freq;
   Node *left;
   Node *right;
   Node(char data, int freq)
   {
       left = right = NULL;
       this->data = data;
       this->freq = freq;
   }
};

struct compare
{
   bool operator()(Node *l, Node *r)
   {
       return (l->freq > r->freq);
   }
};


void storeCodes(struct Node *root, string str)
{
   if (root == NULL)
       return;
   if (root->data != '$')
       codes[root->data] = str;
   storeCodes(root->left, str + "0");
   storeCodes(root->right, str + "1");
}
```



```
priority_queue<Node *, vector<Node *>, compare>
   pq;
void HuffmanCodes(int size)
{
   struct Node *left, *right, *top;
   for (auto v = freq.begin();
        v != freq.end(); v++)
       pq.push(new Node(v->first, v->second));
   while (pq.size() != 1)
   {
       left = pq.top();
       pq.pop();
       right = pq.top();
       pq.pop();
       top = new Node('$', left->freq + right->freq);
       top->left = left;
       top->right = right;
       pq.push(top);
   }
   storeCodes(pq.top(), "");
}



int32_t main()
{
   ios_base::sync_with_stdio(false);
   cin.tie(NULL);
#ifndef ONLINE_JUDGE
   freopen("input.txt", "r", stdin);
   freopen("output.txt", "w", stdout);
#endif

   string s;
   getline(cin, s);

   for (int i = 0; i < s.size(); i++)
   {
       freq[s[i]]++;
   }

   HuffmanCodes(s.length());
```

```
    for (auto v = codes.begin(); v != codes.end(); v++)
        cout << v->first << ' ' << v->second << endl;
    string ans="";
    for(auto i:s){
        ans+=codes[i];
    }

    cout<<"Encoded String- "<<ans<<"\n";
}
```

Output(o) of Huffman Algorithm:



test.cpp > ...

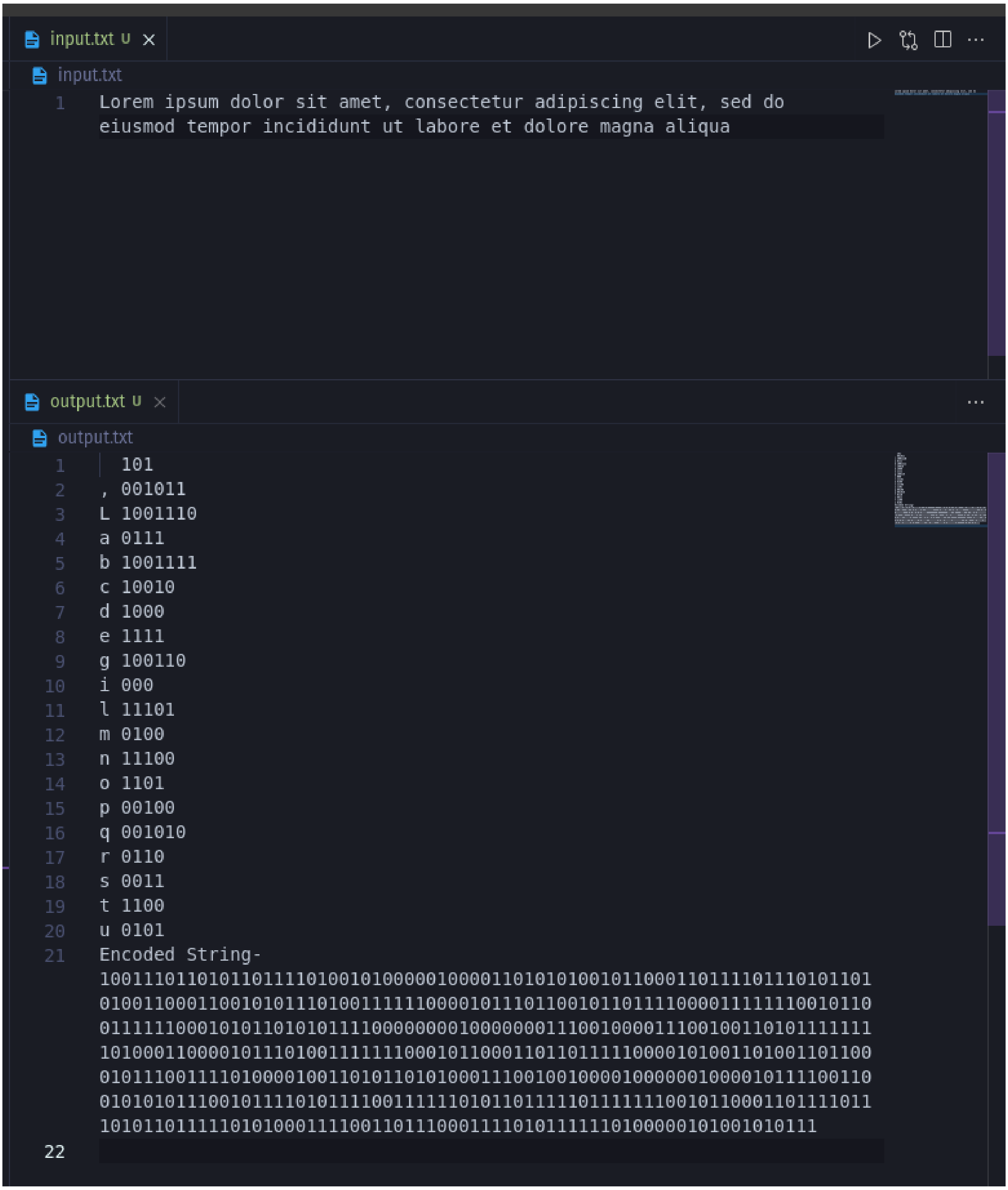
```
        {
            return (l->freq > r->freq);
        }
};


void storeCodes(struct Node *root, string str)
{
    if (root == NULL)
        return;
    if (root->data != '$')
        codes[root->data] = str;
    storeCodes(root->left, str + "0");
    storeCodes(root->right, str + "1");
}

priority_queue<Node *, vector<Node *>, compare>
    pq;
void HuffmanCodes(int size)
{
    struct Node *left, *right, *top;
    for (auto v = freq.begin();
         v != freq.end(); v++)
        pq.push(new Node(v->first, v->second));
    while (pq.size() != 1)
    {
        left = pq.top();
        pq.pop();
        right = pq.top();
        pq.pop();
        top = new Node('$', left->freq + right->freq);
        top->left = left;
```

input.txt

```
Name: Arya Nair
```

output.txt

```
  111
: 1010
A 1011
N 011
a 00
e 1101
i 1100
m 0100
r 100
y 0101
Encoded String- 011000100110110101111011100010100111011001100100
```



```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua
```

```
  101
, 001011
L 1001110
a 0111
b 1001111
c 10010
d 1000
e 1111
g 100110
i 000
l 11101
m 0100
n 11100
o 1101
p 00100
q 001010
r 0110
s 0011
t 1100
u 0101
Encoded String-
10011101101011011110100101000001000011010101001011000110111101110101101
01001100011001010111010011111100001011101100101101111000011111110010110
01111110001010110101011110000000010000000111001000011100100110101111111
10100011000010111010011111110001011000110110111110000101001101001101100
01011100111101000010011010110101000111001001000010000001000010111100110
01010101110010111101011110011111101011011111011111110010110001101111011
101011011111010100011110011011100011110101111110100000101001010111
```

Post Lab Questions:- Differentiate between Fixed length and Variable length Coding with suitable example.

Conclusion: (Based on the observations):

We conclude that we were able to Implement Huffman Algorithm using Greedy approach and analyse its time Complexity.

Outcome:

CO 2. Implement Greedy and Dynamic Programming algorithms

References:

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.