Experiment No. : 4

Title: Implement Huffman Algorithm using Greedy approach

Batch: A2 **Roll No.:16010421073** **Experiment No.: 4**

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

Algorithm of Huffman Algorithm: Refer Coreman for Explaination

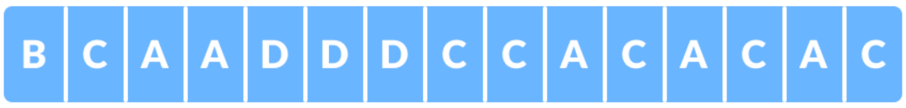
```
HUFFMAN($C$)
1  $n = |C|$
2  $Q = C$
3  for $i = 1$ to $n - 1$
4      allocate a new node $z$
5      $z.left = x =$ EXTRACT-MIN($Q$)
6      $z.right = y =$ EXTRACT-MIN($Q$)
7      $z.freq = x.freq + y.freq$
8      INSERT($Q, z$)
9  return EXTRACT-MIN($Q$)      // return the root of the tree
```

Explanation and Working of Variable Length Huffman Algorithm:

Huffman Coding is a technique of compressing data to reduce its size without losing any of the details. It was first developed by David Huffman.

Huffman Coding is generally useful to compress the data in which there are frequently occurring characters.

Suppose the string below is to be sent over a network.



Initial string

Each character occupies 8 bits. There are a total of 15 characters in the above string. Thus, a total of `8 * 15 = 120` bits are required to send this string.

Using the Huffman Coding technique, we can compress the string to a smaller size.

Huffman coding first creates a tree using the frequencies of the character and then generates code for each character.

Once the data is encoded, it has to be decoded. Decoding is done using the same tree.

Huffman Coding prevents any ambiguity in the decoding process using the concept of **prefix code** ie. a code associated with a character should not be present in the prefix of any other code. The tree created above helps in maintaining the property.

Huffman coding is done with the help of the following steps.

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

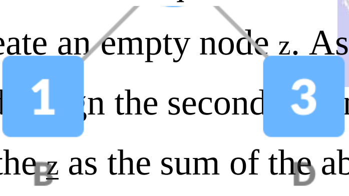
Frequency of string

2.

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

Characters sorted according to the frequency

3. Make each unique character as a leaf node.
4. Create an empty node z. Assign the minimum frequency to the left child of z and assign the second minimum frequency to the right child of z. Set the value of the z as the sum of the above two minimum frequencies.

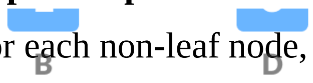


Getting the sum of the least numbers

5. Remove these two minimum frequencies from Q and add the sum into the list of frequencies (* denote the internal nodes in the figure above).
6. Insert node z into the tree.
7. Repeat steps 3 to 5 for all the characters.

Repeat steps 3 to 5 for all the characters.

Repeat steps 3 to 5 for all the characters.



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

Assign 0 to the left edge and 1 to the right edge

For sending the above string over a network, we have to send the tree as well as the above compressed-code. The total size is given by the table below.

Character	Frequency	Code	Size
A	5	11	5*2 = 10
B	1	100	1*3 = 3
C	6	0	6*1 = 6
D	3	101	3*3 = 9
4 * 8 = 32 bits	15 bits		28 bits

Without encoding, the total size of the string was 120 bits. After encoding the size is reduced to `32 + 15 + 28 = 75`.

Derivation of Huffman Algorithm:

Time complexity Analysis

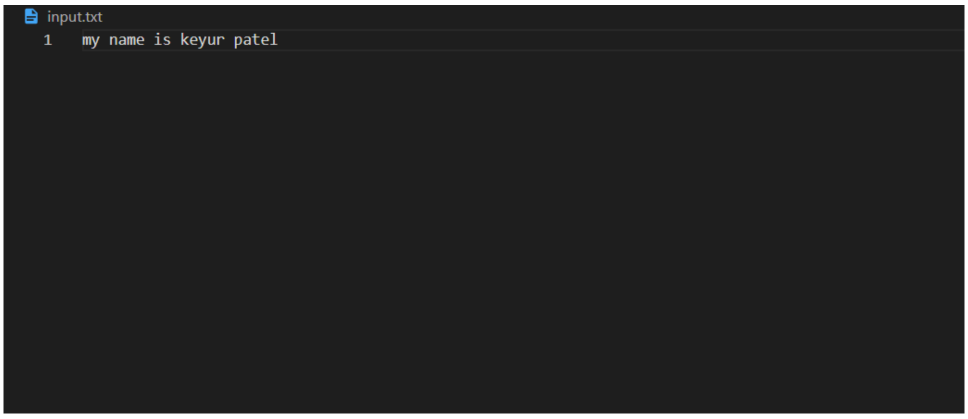
The time complexity of the Huffman algorithm is O(nlogn). Using a heap to store the weight of each tree, each iteration requires O(logn) time to determine the cheapest weight and insert the new weight. There are O(n) iterations, one for each item.

Program(s) of Huffman Algorithm:

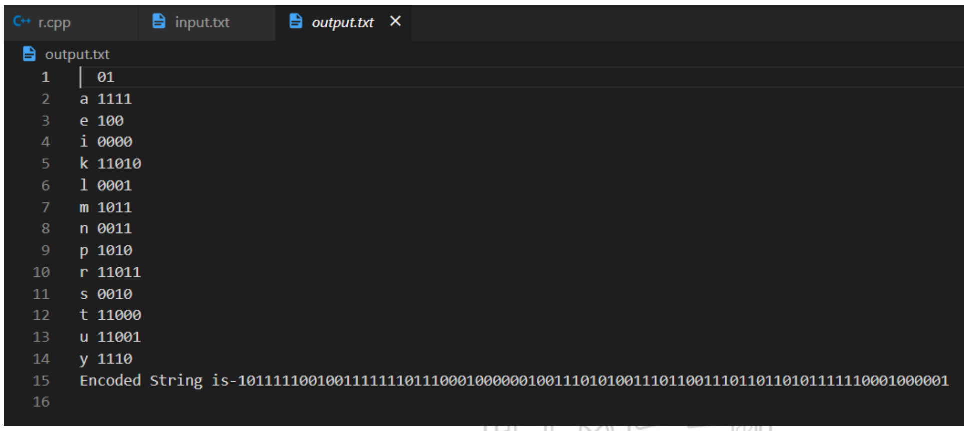
```
#include <bits/stdc++.h>

#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 is-"<<ans<<"\n";
}
```

Output(o) of Huffman Algorithm:



```
my name is keyur patel
```



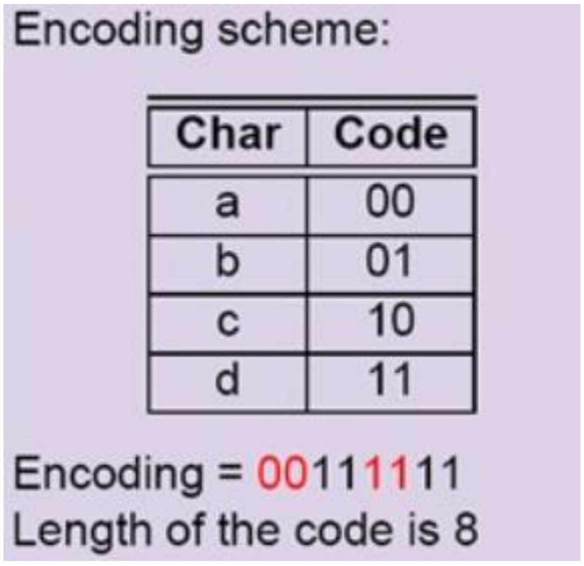
```
 01
a 1111
e 100
i 0000
k 11010
l 0001
m 1011
n 0011
p 1010
r 11011
s 0010
t 11000
u 11001
y 1110
Encoded String is-101111100100111111101110001000000100111010100111011001110110110101111110001000001
```

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

A fixed length variable is one whose length never varies, whereas a variable one is whose length varies with changes in inputs.

- The fixed length ensures the fixed length is never variable.
- A variable-length is one in which the length of each record can vary, depending on what data is stored in the field.
- While the fixed length, leads to wastage of memory, the variable length is a memory efficient variable**.**

Example:

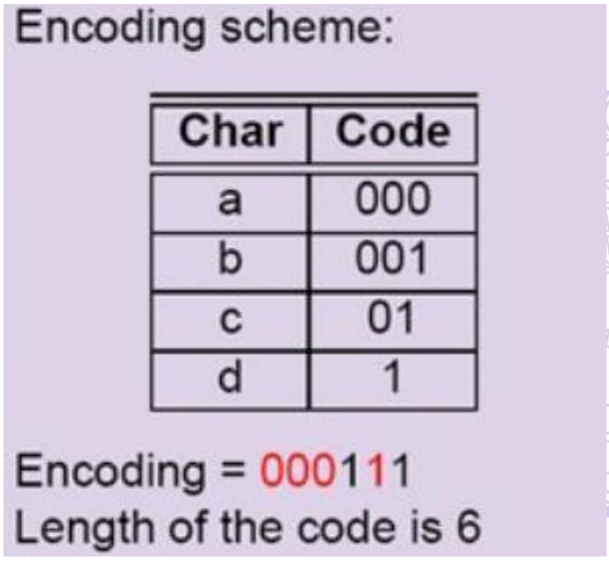


Encoding scheme:

Char	Code
a	00
b	01
c	10
d	11

Encoding = 00111111
Length of the code is 8

Fixed Length Coding



Encoding scheme:

Char	Code
a	000
b	001
c	01
d	1

Encoding = 000111
Length of the code is 6

Variable length Coding

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.