EXPERIMENT 16

Experiment No: 16 **Date:** 29/06/2021

Aim: Implementation of Huffman Coding Technique of Data Compression

Theory:

Huffman Coding

- Huffman coding is a lossless data compression algorithm.
- 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
    - Another one to traverse the tree to find codes.
- For an example:
    - Consider some strings "YYYZXXYYX", the frequency of character Y is larger than X and the character Z has the least frequency.
    - So, the length of the code for Y is smaller than X, and code for X will be smaller than Z.

EXPERIMENT 16

Advantages of Huffman Encoding

- This encoding scheme results in saving lot of storage space, since the binary codes generated are variable in length
- It generates shorter binary codes for encoding symbols/characters that appear more frequently in the input string
- The binary codes generated are prefix-free (sometimes called "prefix-free codes", that is, the bit string representing some particular symbol is never a prefix of the bit string representing any other symbol). This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream

Disadvantages of Huffman Encoding

- Lossless data encoding schemes, like Huffman encoding, achieve a lower compression ratio compared to lossy encoding techniques. Thus, lossless techniques like Huffman encoding are suitable only for encoding text and program files and are unsuitable for encoding digital images.
- Huffman encoding is a relatively slower process since it uses two passes- one for building the statistical model and another for encoding. Thus, the lossless techniques that use Huffman encoding are considerably slower than others.

EXPERIMENT 16

- Since length of all the binary codes is different, it becomes difficult for the decoding software to detect whether the encoded data is corrupt. This can result in an incorrect decoding and subsequently, a wrong output.

Real-life applications of Huffman Encoding

- Huffman encoding is widely used in compression formats like GZIP, PKZIP (winzip) and BZIP2.
- Multimedia codecs like JPEG, PNG and MP3 uses Huffman encoding (to be more precise the prefix codes).
- Huffman encoding still dominates the compression industry since newer arithmetic and range coding schemes are avoided due to their patent issues.

Algorithm

Algorithm Huffman(X):

```
Input: String X of length n with d distinct characters Output:
Coding tree for X
Compute the frequency f(c) of each character c of X a priority
queue Q for
 each character c in X do
Create a single-node binary tree T storing c. Insert T into Q with
key f(c).
while Q.size() > 1 do
fi->Q.minKey()
T1->Q.removeMin()
f2->Q.minKey()
T2-> Q.removeMin()
```

EXPERIMENT 16

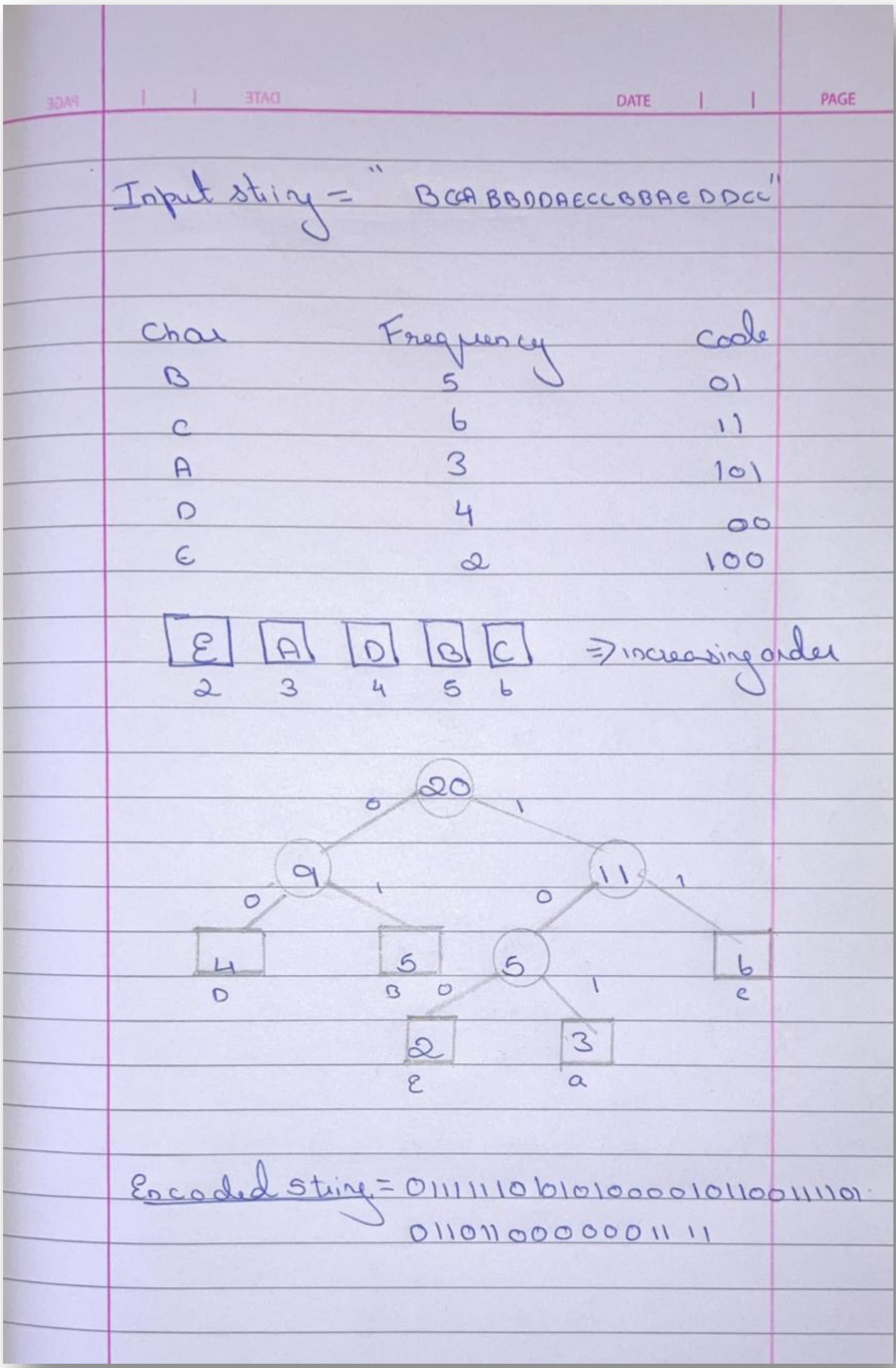
```
Create a new binary tree T with left subtree T1 and right subtree
T2 Insert
T into Q
with key f1 + f2.
return tree Q.removeMin()
```

Algorithm Writting

- The Huffman coding algorithm begins with each of the d distinct characters of the string X to encode being the root node of a single-node binary tree.
- The algorithm proceeds in a series of rounds.
- In each round, the algorithm takes the two binary trees with the smallest frequencies and merges them into a single binary tree.
- It repeats this process until only one tree is left.

EXPERIMENT 16

<u>Tracing with Example</u>


DATE PAGE

Input string = "BCABBDDAECCCBBAEDDCC"

Char	Frequency	Code
B	5	01
C	6	11
A	3	101
D	4	00
E	2	100

$\boxed{E}$ $\boxed{A}$ $\boxed{D}$ $\boxed{B}$ $\boxed{C}$ ⇒ increasing order
2 3 4 5 6

20
0
1
9
11
0
1
0
1
4
D
5
B
5
0
1
6
c
2
E
3
a

Encoded string = 01111110101010000101100111101
011011000000011 11

EXPERIMENT 16

Program

```
#include <bits/stdc++.h>
#define MAX_TREE_HT 256
using namespace std;
map<char, string> codes;
map<char, int> freq;
struct MinHeapNode
{
  char data;
  int freq;
  MinHeapNode *left, *right;

  MinHeapNode(char data, int 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 storeCodes(struct MinHeapNode* root, string str)
{
    if (root==NULL)
        return;
    if (root->data != '$')
        codes[root->data]=str;
    storeCodes(root->left, str + "0");
```

EXPERIMENT 16

```
    storeCodes(root->right, str + "1");
}
priority_queue<MinHeapNode*, vector<MinHeapNode*>, compare> minHeap;


void HuffmanCodes(int size)
{
    struct MinHeapNode *left, *right, *top;
    for (map<char, int>::iterator v=freq.begin(); v!=freq.end(); v++)
        minHeap.push(new MinHeapNode(v->first, v->second));
    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);
    }
    storeCodes(minHeap.top(), "");
}
void calcFreq(string str, int n)
```

EXPERIMENT 16

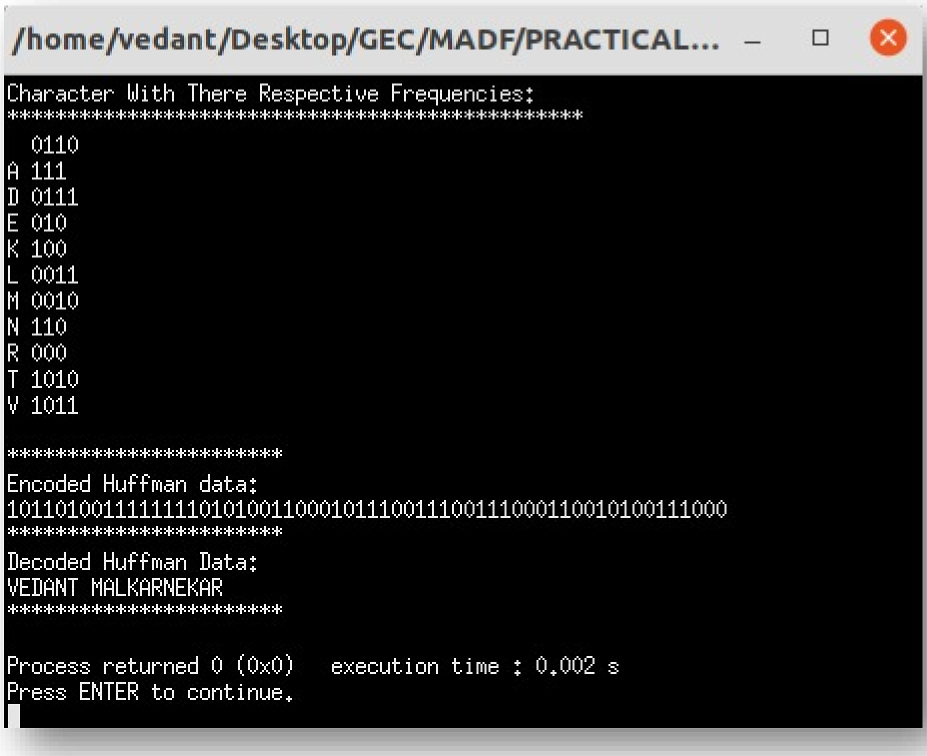
```
{
    for (int i=0; i<str.size(); i++)
        freq[str[i]]++;
}
string decode_file(struct MinHeapNode* root, string s)
{
    string ans = "";
    struct MinHeapNode* curr = root;
    for (int i=0;i<s.size();i++)
    {
        if (s[i] == '0')
           curr = curr->left;
        else
           curr = curr->right;

        if (curr->left==NULL and curr->right==NULL)
        {
            ans += curr->data;
            curr = root;
        }
    }
    return ans+'\0';
}
```

EXPERIMENT 16

```
int main()

{

   string str = "VEDANT MALKARNEKAR";

   string encodedString, decodedString;

   calcFreq(str, str.length());

   HuffmanCodes(str.length());

   cout << "Character With There Respective Frequencies:\n";

    cout<<"************************************************"<<endl;

   for (auto v=codes.begin(); v!=codes.end(); v++)

      cout << v->first <<' ' << v->second << endl;

   for (auto i: str)

      encodedString+=codes[i];

   cout<<"\n***********************";

   cout << "\nEncoded Huffman data:\n" << encodedString << endl;

   decodedString = decode_file(minHeap.top(), encodedString);

   cout<<"***********************";

   cout << "\nDecoded Huffman Data:\n" << decodedString << endl;

   cout<<"***********************"<<endl;

   return 0;

}
```

EXPERIMENT 16

Output



```
Character With There Respective Frequencies:
************************************************
  0110
A 111
D 0111
E 010
K 100
L 0011
M 0010
N 110
R 000
T 1010
V 1011

***********************
Encoded Huffman data:
101101001111111101010011000101110011100111000110010100111000
***********************
Decoded Huffman Data:
VEDANT MALKARNEKAR
***********************

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Conclusion

- Detailed concept of Huffman Coding Technique of Data Compression was studied successfully.
- Program using Huffman Coding Algorithm was executed successfully.