Sherwyn Fernandes
SE COMP-B
191105063

Experiment No: 16

AIM: Implementation of Huffman coding technique of data compression

THEORY:

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 bitstream.

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 frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)

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.

Sherwyn Fernandes
SE COMP-B
191105063

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 of X
Initialise 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**
f1 ⇐ Q.minKey()
T1⇐ Q.removeMin()
f2⇐Q.minKey()
T2⇐Q.removeMin()
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()
}

Time Complexity

- The time complexity of this algorithm is O (n + dlogd), where n is the size of string and d is the number of distinct characters

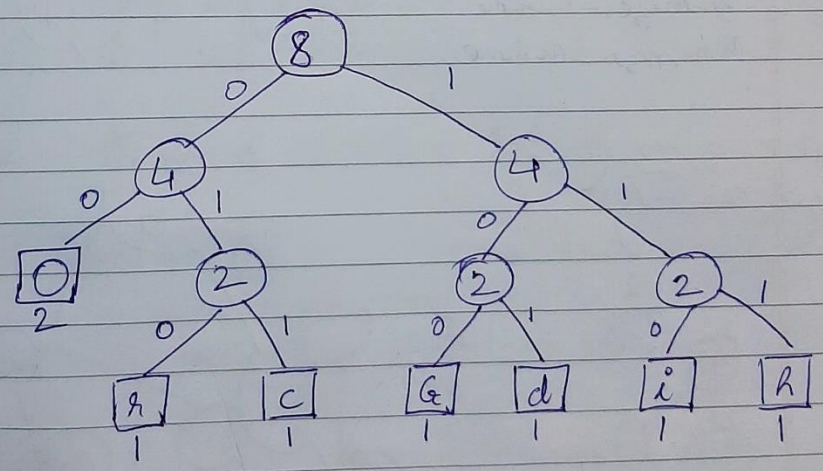
Sherwyn Fernandes
SE COMP-B
191105063

Problem Tracing

Problem Tracing of Huffman coding Algorithm.

Let string = Goodrich.

chars	Frequency	code
G	1	100
o	2	00
d	1	101
r	1	010
i	1	110
c	1	011
h	1	111.



Encoded string = 100000101010110011111

Sherwyn Fernandes
SE COMP-B
191105063

PROGRAM IMPLEMENTATION:

```
#include<iostream>

#include<queue>

#include<map>

using namespace std;


map<char,string>encode;                    //map for encoding


//struct to create a huffman tree node

struct hnode

{

        char c;

        int f;

        struct hnode *left,*right;


        hnode(char data, int freq)

        {

                left=right=NULL;

                this->c = data;

                this->f = freq;

        }


}; typedef struct hnode node;
```

Sherwyn Fernandes
SE COMP-B
191105063

```
void EncodeChar(char *data)
{
	map<char,string> :: iterator itr;

	int size = strlen(data);
	for(int i=0;i<size;i++)
		cout<<encode[data[i]];

	cout<<endl;
}


void printCodes(node *root, string str)		//prints the code of each char
{
	if(!root)
		return;

	if(root->c != '*')
		{
			encode[root->c] = str;
			cout<<root->c<<" : "<<str<<endl;
		}

	printCodes(root->left,str + "0");
```

Sherwyn Fernandes
SE COMP-B
191105063

```
        printCodes(root->right,str + "1");


}


struct compare                  //used by the STL to create a minheap
{
         bool operator()(node* l, node* r)


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



void GenHuff(char *data, int *freq, int n)
{


        node *left,*right,*top;             //temp pointers


        priority_queue<node *, vector<node*>, compare>minHeap;//STL priority queue
creates minHeap


        for(int i=0;i<n;i++)
                minHeap.push(new node(data[i],freq[i]));
```

Sherwyn Fernandes
SE COMP-B
191105063

```
	while(minHeap.size()>1)
	{
		left = minHeap.top();
		minHeap.pop();

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

		top = new node('*', left->f + right->f);//'*' indicates new node having left and right child
		top->left = left;
		top->right = right;

		minHeap.push(top);
	}

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



}



int main()
{
```

Sherwyn Fernandes
SE COMP-B
191105063

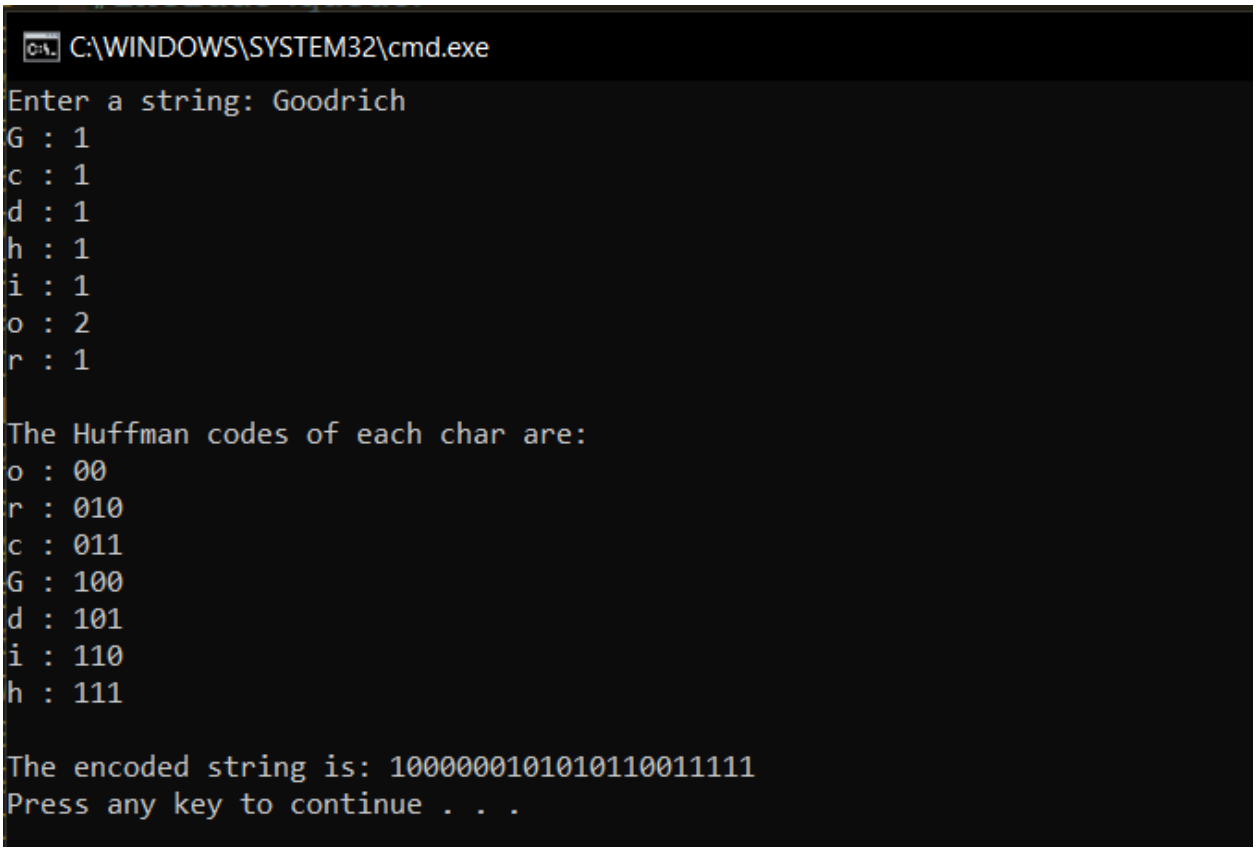
```
//int n; //number of unique characters
int count,freq[20];
map<char,int>obj;
char data[100],chars[20];

cout<<"Enter a string: ";
cin.getline(data,100);

int n = strlen(data);
for(int i=0;i<n;i++)
	{
		if(obj.count(data[i]) != 1)
			obj[data[i]] = 1;

		else
		{
			count = obj[data[i]];
			obj[data[i]] = ++count;
		}
	}

map<char,int> :: iterator it;

int i=0;
for(it=obj.begin(); it!=obj.end(); it++)
```

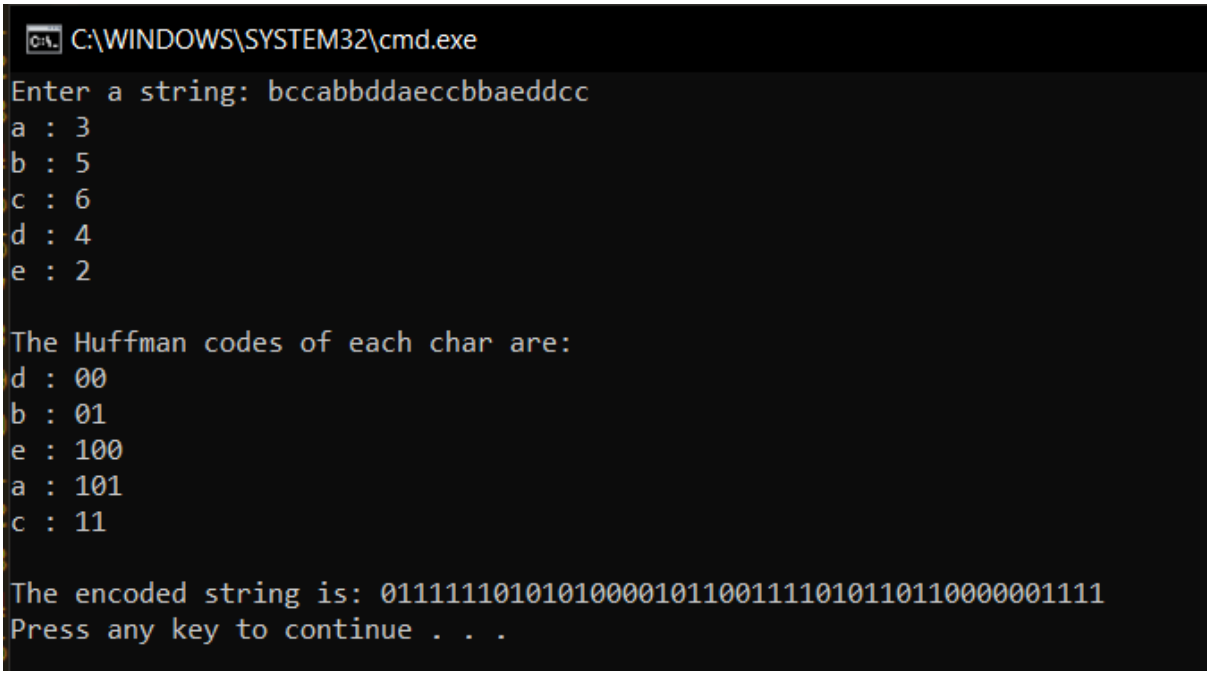
Sherwyn Fernandes
SE COMP-B
191105063

```
		{
			chars[i] = it->first;
			freq[i] = it->second;
			i++;
		}
	n = i;
	int j = strlen(chars);
	for(int i=0;i<j;i++)
		cout<<chars[i]<<" : "<<freq[i]<<endl;



	cout<<"\nThe Huffman codes of each char are: \n";
	GenHuff(chars,freq,n);

	cout<<"\nThe encoded string is: ";
	EncodeChar(data);

	return 0;
}
```

Sherwyn Fernandes
SE COMP-B
191105063

OUTPUTS:





Conclusion:

- Time complexity of the algorithm is of the order of O(n+dlogd), where n is the size of the string and d is the number of unique characters in the string.