PROJECT NAME: TEXT COMPRESSION-ENCODING AND DECODING

Revanth Reddy Ambati-202151

Branch: Electrical and Electronics Engineering

INTRODUCTION:

Huffman Coding:-

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.

Working:-

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

Each character occupies 8 bits. There are a total of 15 characters in the above string.

Thus, a total of **8 x 15 = 120** bits are required to send this string.

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

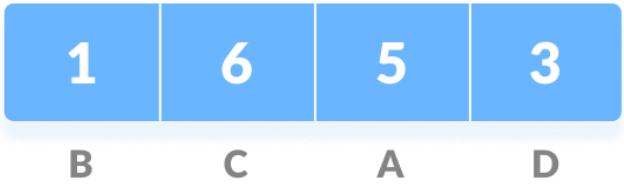
Huffman coding first creates a tree using the frequencies of the character and then generates code for each character.

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 i.e., a code associated with a character should not be present in the prefix of any other code. The tree created helps in maintaining the property.

Steps:

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

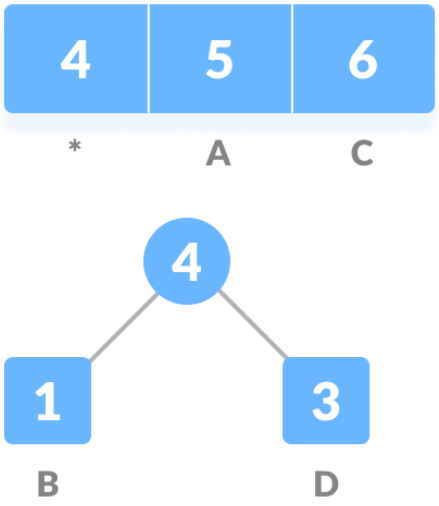


Frequency of string

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

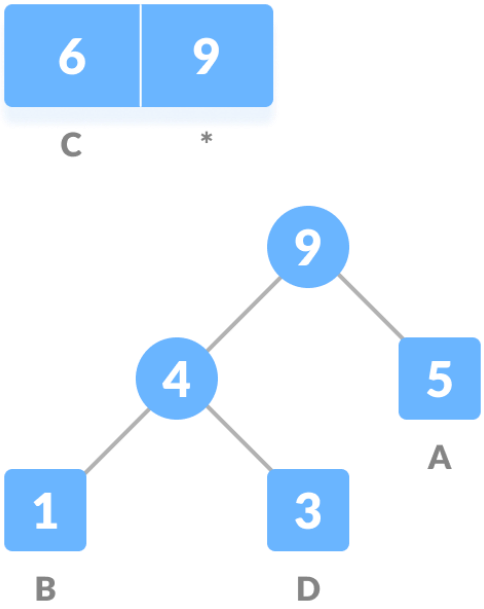
Characters sorted according to the frequency

3. Making each unique character as a leaf node.
4. Creating an empty node ‘Z’. Assigning the minimum frequency to the left child of ‘Z’ and assigning 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.

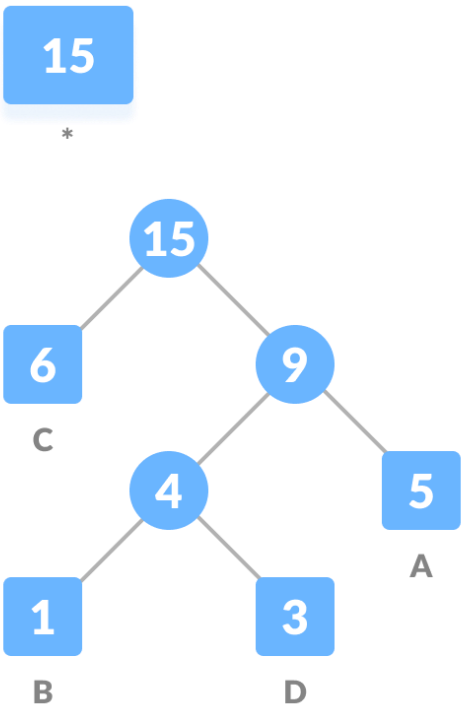


Getting the sum of the least numbers

5. Removing 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 priority Queue.
7. Repeat steps 3 to 5 for all the characters.

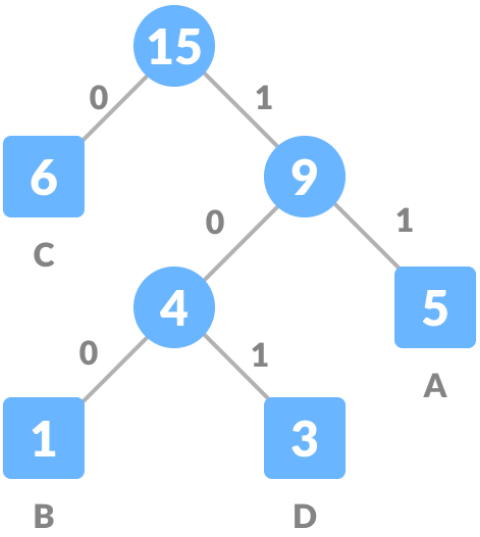


Repeat steps 3 to 5 for all the characters.



Repeat steps 3 to 5 for all the characters.

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



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

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

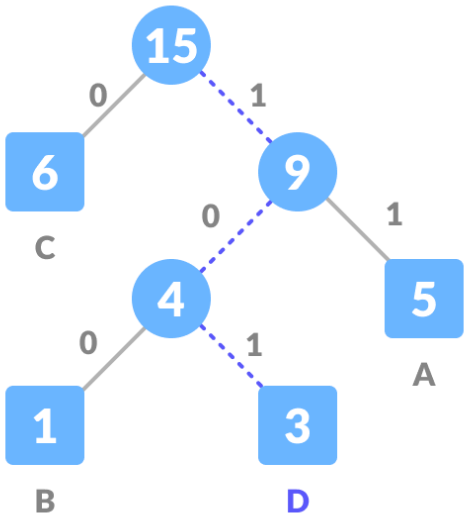
Without encoding, the total size of the string was 120 bits.

After encoding the size is reduced to 32 + 15 + 28 = 75.

Decoding the Code:-

For decoding the code, we can take the code and traverse through the tree to find the character.

Let 101 is to be decoded, we can traverse from the root as in the figure below.



Decoding

To decode the encoded data, we require the Huffman tree. We iterate through the binary encoded data. To find character corresponding to current bits, we use following simple steps.

We start from root and do following until a leaf is found.

If current bit is 0, we move to left node of the tree.

If the bit is 1, we move to right node of the tree.

If during traversal, we encounter a leaf node, we print character of that particular leaf node and then again continue the iteration of the encoded data starting from step 1.

The below code takes a string as input, it encodes it and save in a variable encodedString. Then it decodes it and print the original string. The below code performs full Huffman Encoding and Decoding of a given input data.

CODE:

```
#include<bits/stdc++.h>
using namespace std;
class TextCompression
{
        public:
          // creating the structure HuffmanNode
           struct HuffmanNode
           {
              char data;
              int freq;
              HuffmanNode* LeftNode;
              HuffmanNode* RightNode;
           };
          // typedef (keyword used to assign alternative names to the existing datatypes)
           typedef HuffmanNode* Huffman;
          // function to get Huffman code for each character
           void    Huffman_encode(Huffman    root,unordered_map<char,string> &mp1,string store="")
            {
              if(root->LeftNode==NULL && root->RightNode==NULL){
                 mp1[root->data]=store;
                 return ;
             }
              // traversing LeftNode then we Add '0' to the string
              Huffman_encode(root->LeftNode,mp1,store+'0');
```

```
            // traversing RightNode then we Add '1' to the string
            Huffman_encode(root->RightNode,mp1,store+'1');
        }
        string Huffman_decode(Huffman root,string s)
        {
            string ans="";
            int n=s.size();
            Huffman temp=root;
            for(int i=0;i<n;i++)
              {
                if(s[i]==' ') ans+=' ';
                else
                {
                    if(s[i]=='0'){
                        temp=temp->LeftNode;
                        }
                        else{
                           temp=temp->RightNode;
                        }
                        // checking for leaf Node
                if(temp->LeftNode==NULL&&temp>RightNode==NULL ){
                           ans+=temp->data;
                           temp=root;
                        }
                    }
                }
```

```
        return ans;
          }
    // function to create HuffmanNode.
    Huffman make_node(char data,int freq)
    {
        Huffman Temp=new HuffmanNode;
        Temp->data=data;
        Temp->freq=freq;
        Temp->LeftNode=NULL;
        Temp->RightNode=NULL;
        return Temp;
    }
    /*
    compartor for MIN HEAP
     As we want heap should be sorted based on the frequency of charecter occured
    */
    struct cmp
    {
        bool operator()(Huffman Node1, Huffman Node2)
        {
            if(Node1->freq > Node2->freq){
                return true;
            }
            else{
                return false;
            }
```

```
        }
    };
    // function to print the huffmanCodes for each character

    void PrintCodes(unordered_map<char,string> Temp){
        cout<<"\nThe Encoded charecters in the string are \n"<<endl;
        for(auto it:Temp){
            cout<<"The Huffman Code for "<<it.first<<" is "<<it.second<<endl;
        }
    }
    // function to encode the given string

    String Encode_string(string s,unordered_map<char,string> HuffmanMap){
        string encode;
        for(auto it:s){
            if(it==' ') encode+=' ';
            else encode+=HuffmanMap[it];
        }
        return encode;
    }
    // function to create the Huffman Tree
        void huffmanCodes(string s,unordered_map<char,int> data)
        {
            // using priority_queue to get the minimum frequency characters

    priority_queue<Huffman,vector<Huffman>,cmp> HuffmanQueue;
```

```
        // inserting the character and frequency into priority_queue

        for(auto it:data)
        HuffmanQueue.push(make_node(it.first,it.second));

        // creating Huffman Tree

        while(HuffmanQueue.size()!=1)
        {
            Huffman left,right,combined;

            left=HuffmanQueue.top();
            HuffmanQueue.pop();

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

            combined=make_node('$',left->freq+right->freq);

            combined->LeftNode=left;
            combined->RightNode=right;

            // inserting the combined node into priority_queue

            HuffmanQueue.push(combined);
```

```
        }
        // when all the nodes are combined the final node will be the root of Huffman Tree
        Huffman root=HuffmanQueue.top();
        //using unordered_map to store the huffmanCode of each character

        unordered_map<char,string> ans;
        Huffman_encode(root,ans);

        // printing the huffmanCodes for each character

        PrintCodes(ans);

        cout<<"\nThe original String is --> "<<s<<endl;

        // encoding the given string

        string encode=Encode_string(s,ans);

        cout<<"\nThe Encoded String is --> "<<encode<<endl;

        // DECODING THE STRING
        string decoded=Huffman_decode(root,encode);

        cout<<"\nThe Decoded string is --> "<<decoded<<endl;
                }
};
```

```
int main(){
        string S;
        cout<<"Enter the string to be encoded \n"<<endl;
        getline(cin,S);
        /*
        using unordered_map to store the frequency of each character
        occurred in the string and neglecting spaces.
        */
        unordered_map<char,int> HuffmanQueue;

        for(auto it: S){
          if(it!=' ')   HuffmanQueue[it]++;
        }
        cout<<"\nThe frequency of each character in the string are \n"<<endl
        for(auto it: HuffmanQueue){
     cout<<"The frequency of "<<it.first<<" is "<<it.second<<endl;
        }
        TextCompression encode;
        encode.huffmanCodes(S,HuffmanQueue);
        return 0;
}
```

Code for Huffman encoding and decoding:-

https://ideone.com/p9oE2w

Huffman Coding Algorithm:-

- create a priority queue Q consisting of each unique character.
- sort then in ascending order of their frequencies.
- for all the unique characters:
  - ✓ create a newNode
  - ✓ extract minimum value from Q and assign it to leftChild of newNode
  - ✓ extract minimum value from Q and assign it to rightChild of newNode
  - ✓ calculate the sum of these two minimum values and assign it to the value of newNode insert this newNode into the tree
- return rootNode

Huffman Coding Complexity:-

The time complexity for encoding each unique character based on its frequency is 0(nlogn).

Extracting minimum frequency from the priority queue takes place 2*(n-1) times and its complexity is 0(nlogn). Thus, the overall complexity is 0(nlogn).

Huffman Coding Applications:-

- Huffman coding is used in conventional compression formats like GZIP, BZIP2, PKZIP, etc.
- For text and fax transmissions.
- Multimedia formats like JPEG, PNG, and MP3 use Huffman encoding.

REFERENCES:-

https://www.programiz.com/

THANK YOU