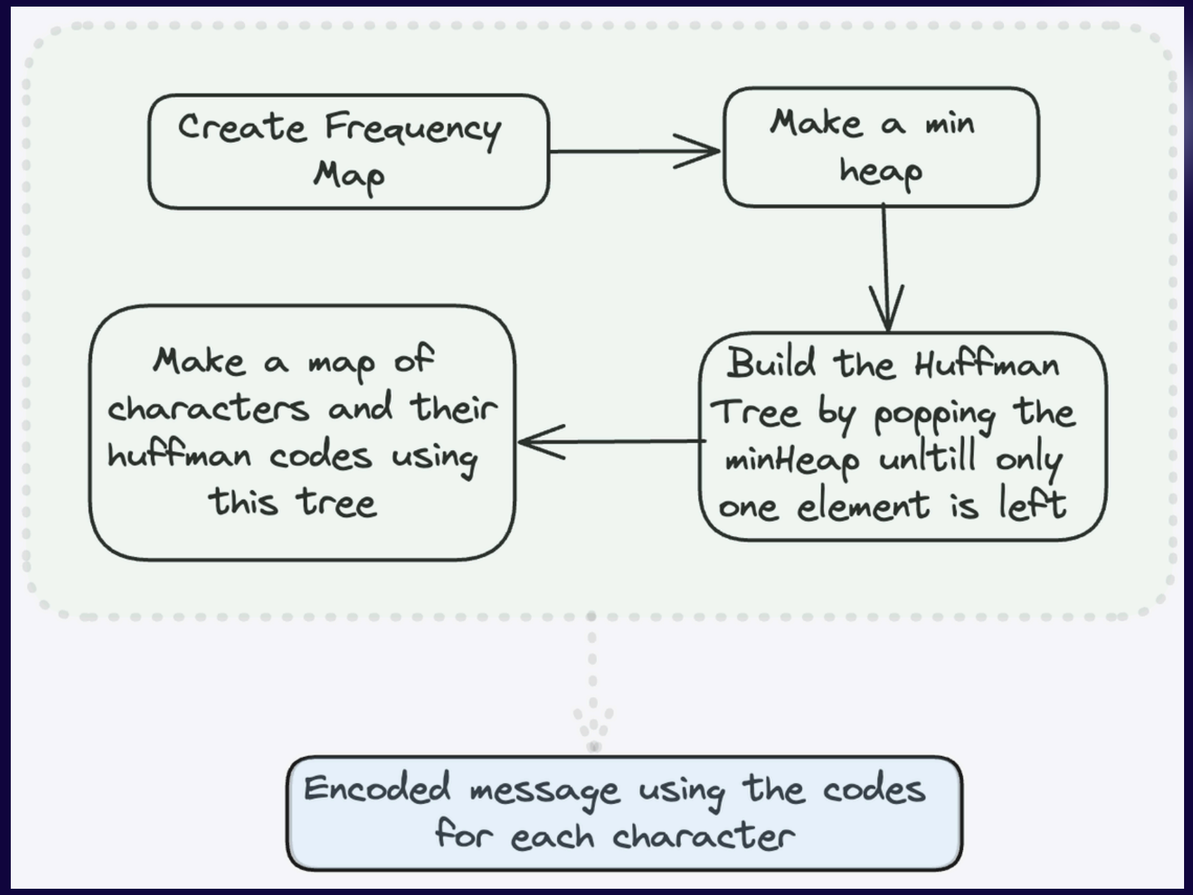
THE HUFFMAN HIGHWAYS

MIDTERM PROJECT

Topic - Text Compression and Decompression using Huffman Coding

YATHARTH DANGI
231197



ABOUT 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 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.
Let us understand prefix codes with a counter exam

DECOMPRESSING

To decompress the compressed file , we we need the characters and their frequencies as necessary information alongwith the compressed file . Using these we can create the Huffman frequencymap using which we can build out HuffmanTree.

This tree has its nodes set to either the newline character (‘\0‘) or the characters which have to be decoded .

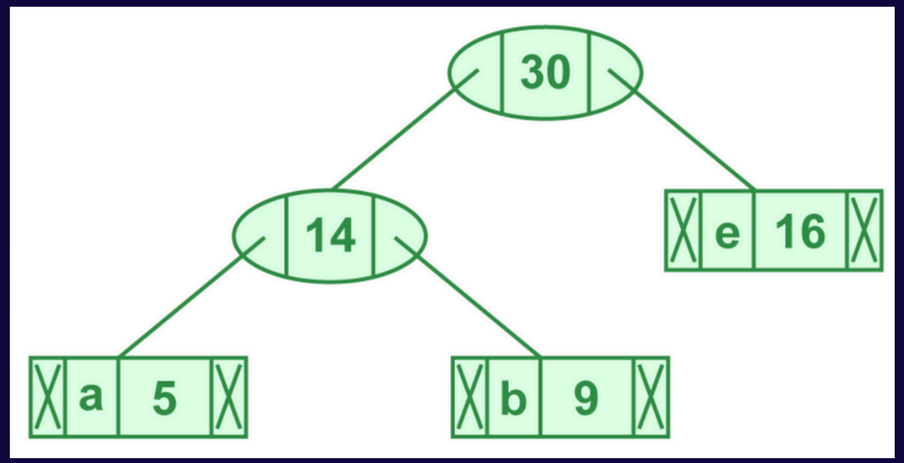
```
class HuffmanNode{
    public :
        int freq ;
        char data ;
        HuffmanNode * left ;
        HuffmanNode * right ;
    HuffmanNode( int f , char ch ) : freq(f) , data(ch) ,left(nullptr),right(nullptr) {}
};
```

The structure of nodes of the huffman tree

HUFFMAN TREE

```
void buildHuffmanTree(){
    priority_queue<HuffmanNode* , vector<HuffmanNode*> , Cmp > minHeap ;

    for(int itr = 0 ; itr < map_size ; itr++ ){
        minHeap.push( new HuffmanNode(frequencies[itr] , characters[itr] )) ;
    }

    while(minHeap.size() != 1){
        HuffmanNode * left = minHeap.top() ; minHeap.pop() ;
        HuffmanNode * right = minHeap.top() ; minHeap.pop() ;

        int sum = left->freq + right->freq ;
        HuffmanNode * sum_node = new HuffmanNode(sum , '\0') ;

        sum_node->left = left ;
        sum_node->right = right ;

        minHeap.push(sum_node) ;
    }

    root = minHeap.top() ;
    cout<<root->freq<<endl ;
}
```

- MinHeap is created which consists of the characters and their frequencies stored inside a huffman node in increasing order of the frequencies .
- We pop first two nodes having characters with lowest current frequencies , and push a sum node having sum of these 2 frequencies and the null character .
  This creates a tree in which all the leaf nodes represent the individual characters.



This is as an example tree , where frequency map is as follows :
freq[a] = 5 , freq[b] = 9 , freq[e]=16

DECODING THE COMPRESSED TEXT

This code snippet reads the compressed.txt file containing the compressed data and decodes it .

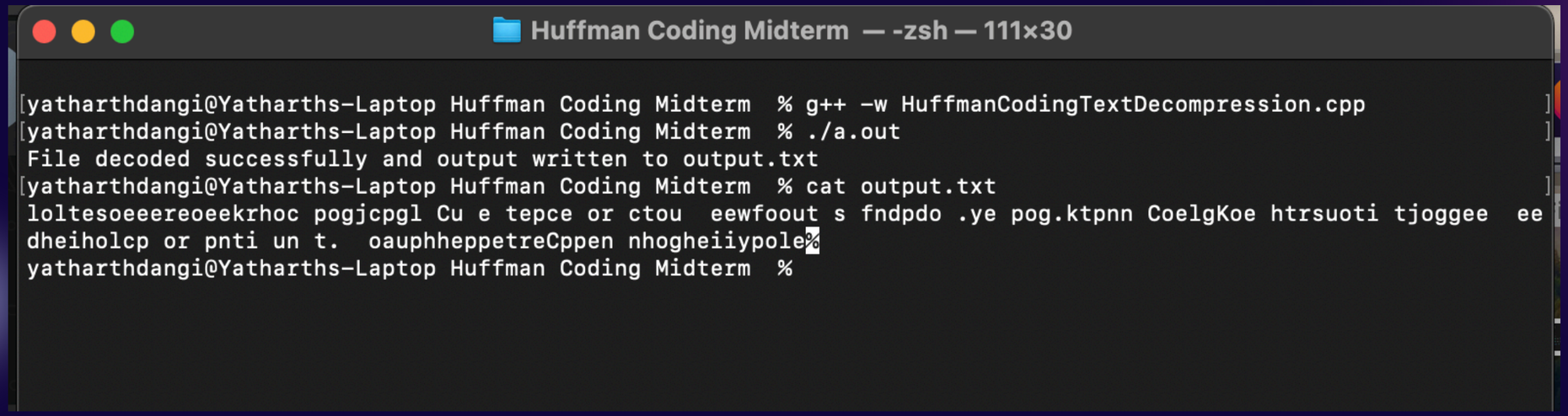
A simple scheme is used to traverse in the tree and decode the data :

- A 0 means moving to the left node and 1 means moving to the right node , we take the bit from encoded file and do this traversal until we reach a leaf node .
- As we saw , each leaf node has a unique character , so as we reach a leaf node .
- we concatenate this node character at the end of out current decoded message .
- Set the curr equal to root again , journey to find the next character !

Since each traversal path is unique , it is ensured that each character has a unique huffman code .

```
class HuffmanCoding{
        string decode(){
            buildHuffmanTree() ;

            string InputFIleName = "compressed.txt" ;

            ifstream inputFile(InputFIleName , ios::in) ;

            if(!inputFile.is_open()){
                cerr<<"Error opening the input file!"<<endl ;
                return "";
            }

            string encoded_data((istreambuf_iterator<char>(inputFile)), istreambuf_iterator<char>());
            string output ;

            HuffmanNode * curr = root ;
            //Traversal inside tree ;
            for( char ch : encoded_data){
                if(ch == '0') curr = curr->left ;
                if(ch == '1') curr = curr->right ;


                if(curr->right == nullptr && curr->left == nullptr){
                    output += curr->data ;
                    curr = root ;
                    continue ;
                }
            }

            return output ;
        }
```

OUTPUT



Finally , we call the writeOutput() function , which writes the decoded message in the output.txt file .

Above is the execution of my code along with the output .

THANK YOU