LAB REPORT

Submitted by

KASHISH KEDIA [RA2011003010355]
SAKTHI MAHALAKSHMI S [RA2011003010363]
KETSI DEBORAH P [RA2011003010372]

Under the Guidance of

Dr. N. ARUNACHALAM

Assistant Professor - Department of Networking and Communications

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

with specialization in Cloud Computing

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

JUNE 2022

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this mini project titled **“File Compressor”** is the bonafide work done by **KASHISH KEDIA [RA2011003010355], SAKTHI MAHALAKSHMI S [RA2011003010363], KETSI DEBORAH P [RA2011003010372]** who carried out the lab exercises under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE
Dr. N ARUNACHALAM
DAA – Course Faculty
Assistant Professor
Department of Networking and Communications

TABLE OF CONTENTS

Contribution Table

REG NUMBER	NAME	CONTRIBUTION
RA2011003010355	KASHISH KEDIA	PROBLEM DEFINITION, EXPLANATION, DESIGN TECHNIQUE
RA2011003010363	SAKTHI MAHALAKSHMI S	ALGORITHM, ANALYSIS, EXPLANATION
RA2011003010372	KETSI DEBORAH P	IMPLEMENTATION, OUTPUT, CONCLUSION

Problem Definition

Everyday, a humongous amount of data is being transmitted around the world. Most of this data are files which are transferred from one user/organization to another. Therefore, to make sure that the files are transferred efficiently without losing any data, they are first compressed/zipped into smaller files and sent. Next, they are received by the other user, who extracts/decompresses the zipped/compressed file to view the original file.

FILE COMPRESSOR

Problem Explanation

We take input of any file like .pdf, .word, .jpeg, .png, etc in the system. Then, we run the program. We use Huffman Coding Algorithm to encode the file and compress it into a smaller file.
Next, we use a decoder which uses Huffman Decoding to decode the file and decompress/ extract the original file from it. In this way, files are compressed and decompressed successfully.

Input:
Enter the type of file: pdf
Enter the name of the file: Project

If file is not present,

Output:
Oops!! The file could not be found. Try Again !

If the file is present,

Output:
File Compressed Successfully !!!

For Decompressing,
Input:
Enter the type of file: pdf
Enter the name of the file: Project_compressed

Output:
File Decompressed Successfully !!!

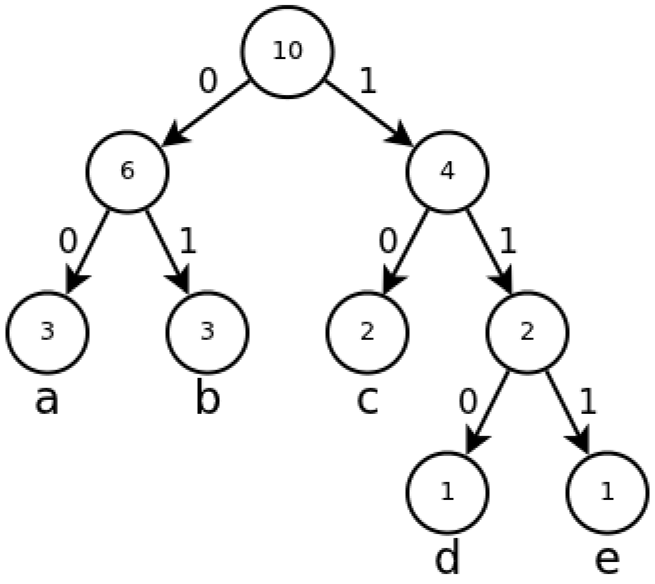
Design Technique Used

Huffman Coding Algorithm - Greedy Algorithm

Huffman Coding is a lossless data compression algorithm. The idea is to assign variable length to input characters; lengths of the assigned codes are based on the frequencies of the 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 the character assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated main bitstream.

There are mainly two major parts in Huffman Coding,

1) Build a Huffman tree from input characters
2) Traverse the Huffman tree and assign codes to characters.



Algorithm

Step-1 : Calculate the frequency of each character in the string.
Step-2 : Sort the characters in increasing order of the frequency. These are stored in a priority queue.
Step-3: Make each unique character as a leaf node.
Step-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.
Step-5: Remove these two minimum frequencies from Q and add the sum into the list of frequencies.
Step-6: Insert node z into the tree.
Step-7: Repeat steps 3 to 5 for all the characters.

Explanation with example

Example:

Input:
arr[]={‘a’,’b’,’c’,’d’,’e’,’f’}
freq[]={5,9,12,13,16,45}

Output:
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111

Explanation:

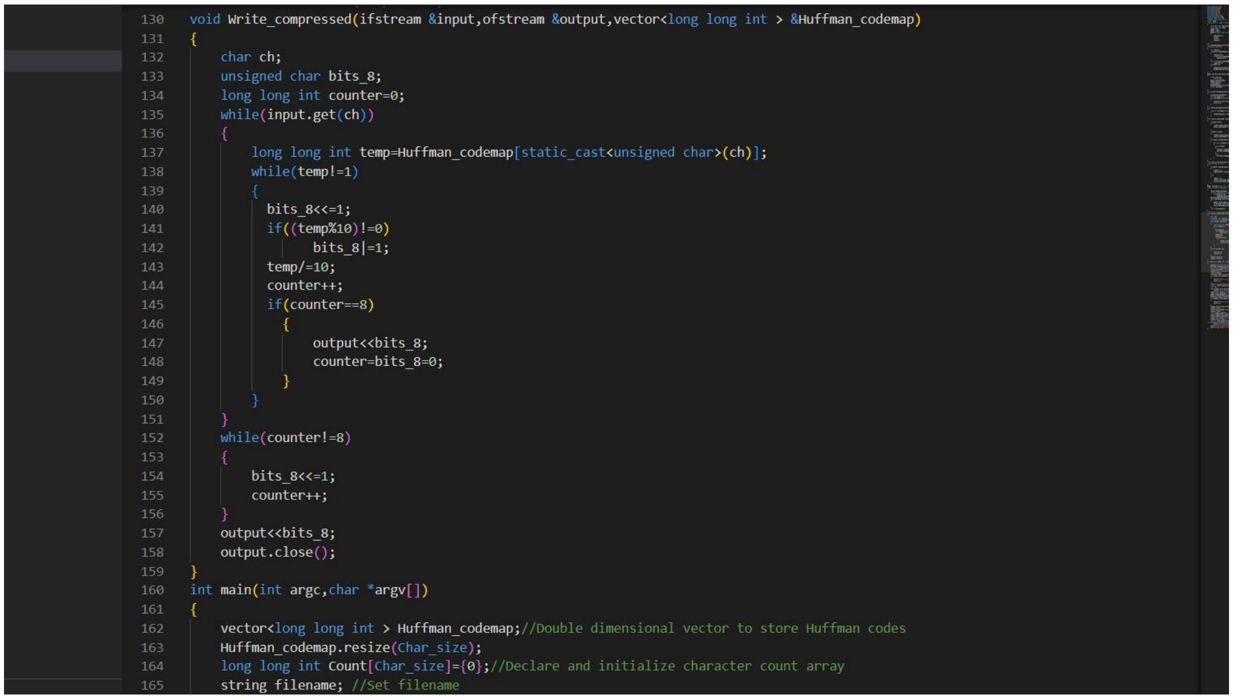
Since, the character f is the most frequent, it is given the smallest code.
f -> 45 -> 0

Since, the character a is the least frequent, it is given the largest code.
a -> 5 -> 1100

Thus, it ensures efficient compression of the file’s contents as it assigns the smallest bit code to the most frequent character, and the largest bit code to the least frequent character, so that the file takes up least bits and is compressed efficiently without losing any data.

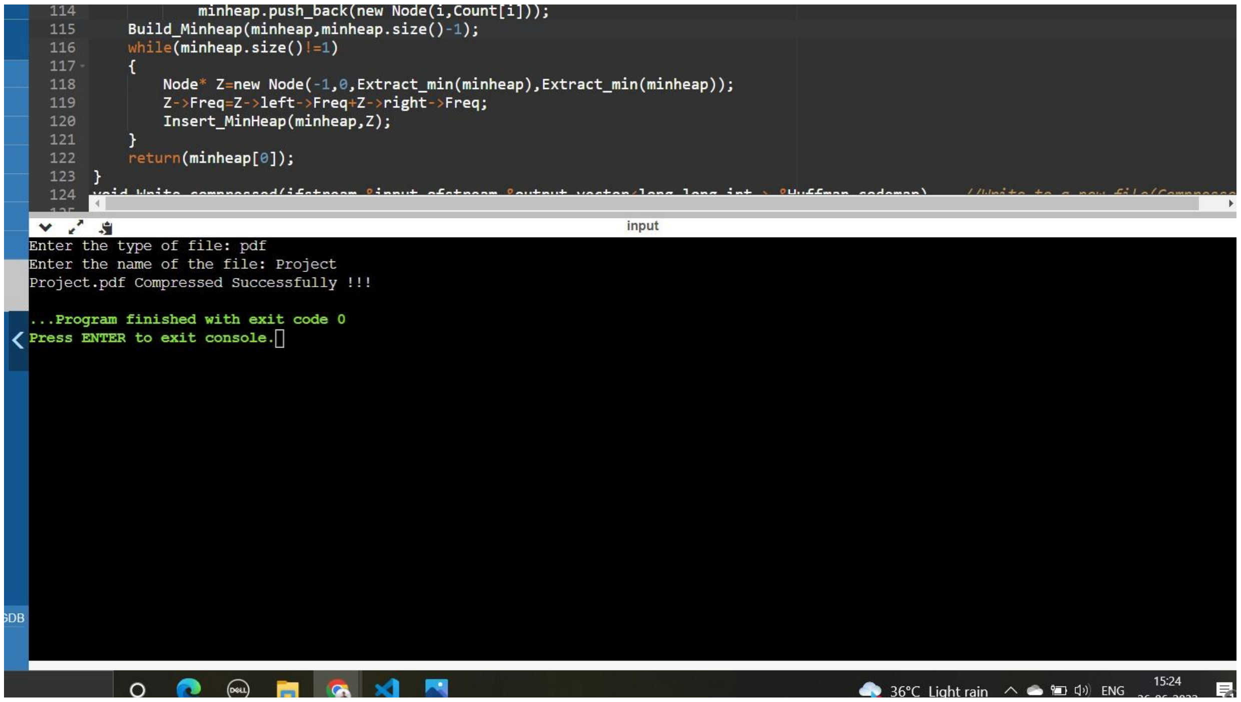
Implementation

Encoding Code :



```cpp
void Write_compressed(ifstream &input,ofstream &output,vector<long long int > &Huffman_codemap)
{
    char ch;
    unsigned char bits_8;
    long long int counter=0;
    while(input.get(ch))
    {
        long long int temp=Huffman_codemap[static_cast<unsigned char>(ch)];
        while(temp!=1)
        {
          bits_8<<=1;
          if((temp%10)!=0)
                bits_8|=1;
          temp/=10;
          counter++;
          if(counter==8)
            {
                output<<bits_8;
                counter=bits_8=0;
            }
        }
    }
    while(counter!=8)
    {
        bits_8<<=1;
        counter++;
    }
    output<<bits_8;
    output.close();
}
int main(int argc,char *argv[])
{
    vector<long long int > Huffman_codemap;//Double dimensional vector to store Huffman codes
    Huffman_codemap.resize(Char_size);
    long long int Count[Char_size]={0};//Declare and initialize character count array
    string filename; //Set filename
```

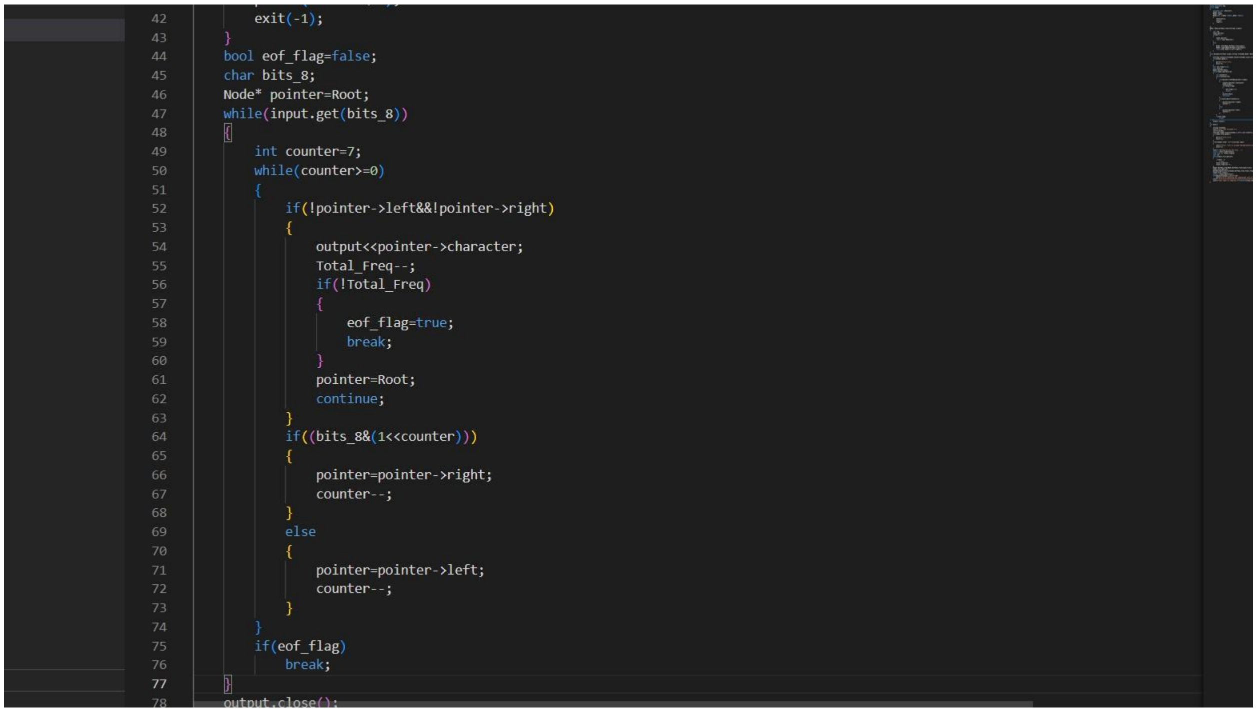
Output:



```
            minheap.push_back(new Node(i,Count[i]));
    Build_Minheap(minheap,minheap.size()-1);
    while(minheap.size()!=1)
    {
        Node* Z=new Node(-1,0,Extract_min(minheap),Extract_min(minheap));
        Z->Freq=Z->left->Freq+Z->right->Freq;
        Insert_MinHeap(minheap,Z);
    }
    return(minheap[0]);
}
```

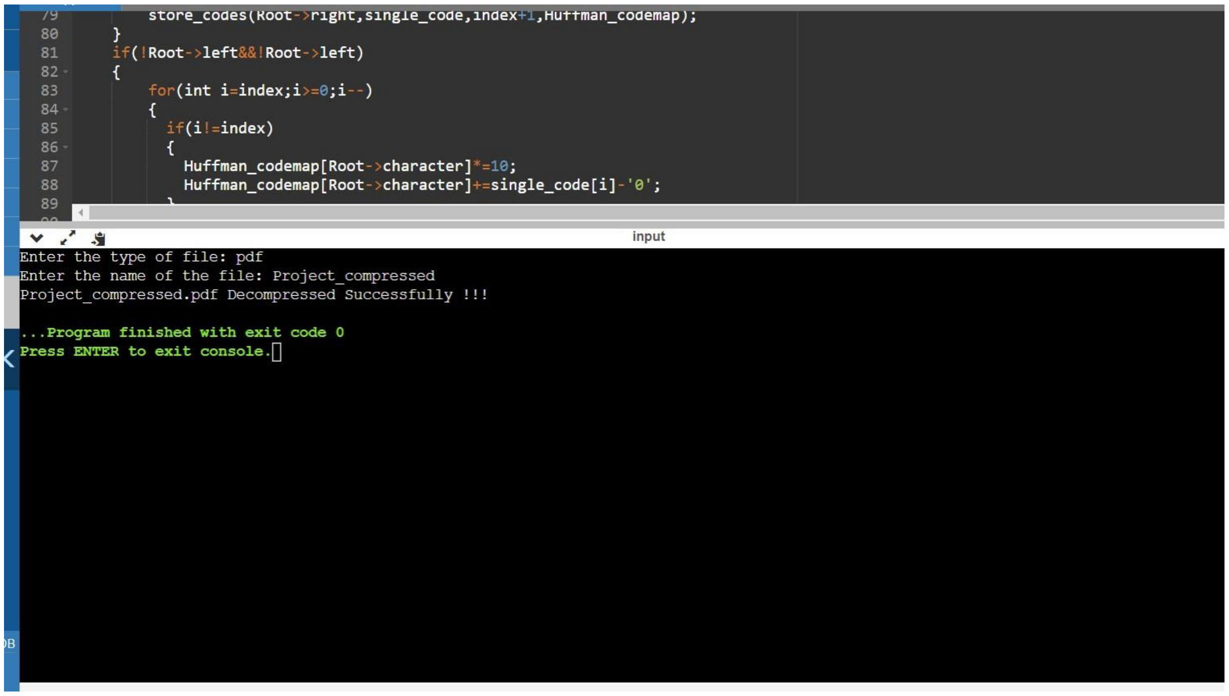
```
Enter the type of file: pdf
Enter the name of the file: Project
Project.pdf Compressed Successfully !!!

...Program finished with exit code 0
Press ENTER to exit console.
```

Decoding Code:



```
        exit(-1);
    }
    bool eof_flag=false;
    char bits_8;
    Node* pointer=Root;
    while(input.get(bits_8))
    {
        int counter=7;
        while(counter>=0)
        {
            if(!pointer->left&&!pointer->right)
            {
                output<<pointer->character;
                Total_Freq--;
                if(!Total_Freq)
                {
                    eof_flag=true;
                    break;
                }
                pointer=Root;
                continue;
            }
            if((bits_8&(1<<counter)))
            {
                pointer=pointer->right;
                counter--;
            }
            else
            {
                pointer=pointer->left;
                counter--;
            }
        }
        if(eof_flag)
            break;
    }
    output.close();
```

Output:



```
        store_codes(Root->right,single_code,index+1,Huffman_codemap);
    }
    if(!Root->left&&!Root->left)
    {
        for(int i=index;i>=0;i--)
        {
          if(i!=index)
          {
            Huffman_codemap[Root->character]*=10;
            Huffman_codemap[Root->character]+=single_code[i]-'0';
```

```
Enter the type of file: pdf
Enter the name of the file: Project_compressed
Project_compressed.pdf Decompressed Successfully !!!

...Program finished with exit code 0
Press ENTER to exit console.
```

Complexity Analysis

1) **Time Complexity: O(nlogn)**

2) **Space Complexity: O(n)**

Conclusion

Huffman Coding is a lossless data compression algorithm. It is a Greedy Algorithm. It is used for encoding and decoding in various day-to-day life applications.
Thus, we successfully implemented the Huffman Coding Algorithm to build a file compressor program which compresses the files efficiently using Huffman Encoding and decompresses it using Huffman Decoding which ensures data integrity without any ambiguity.

References

[1] GeeksforGeeks : https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/

[2] Programmiz : https://www.programiz.com/dsa/huffman-coding

[3] Wikipedia : https://en.wikipedia.org/wiki/Huffman_coding

[4] Stack Overflow: https://stackoverflow.com/

[5] Javatpoint : https://www.javatpoint.com/huffman-coding