# Abstract of the project

Our project basically creates a text compressor using Huffman coding algorithm which mainly focuses on data compression whether it be text or image compression. The idea is to assign variable-length codes to input characters; the 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.

# Problem Statement

We usually face storage issues because of the large files on our devices. This project can help us to compress the text files with different characters present in the string. This is done with the help of the Huffman coding algorithm.

## Topics used -

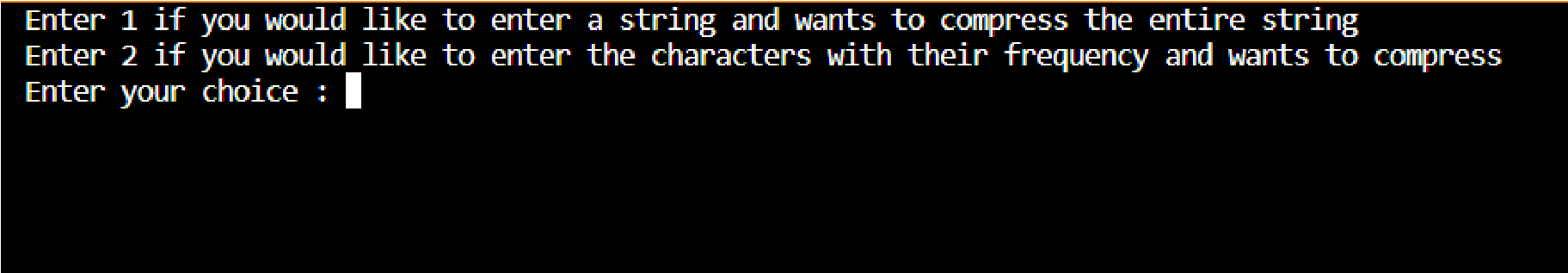
1. Trees
2. Priority queue
3. Class
4. Structure
5. Map

# Design of the project -

# Welcome page -

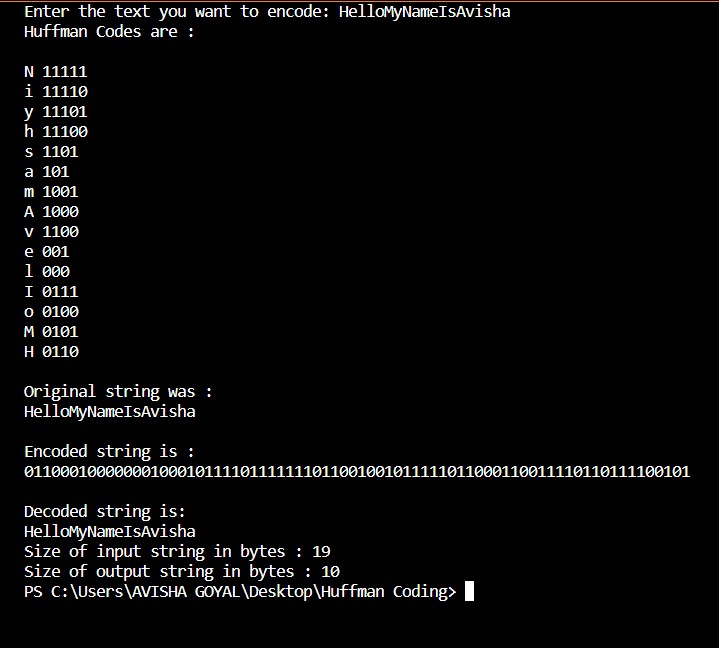


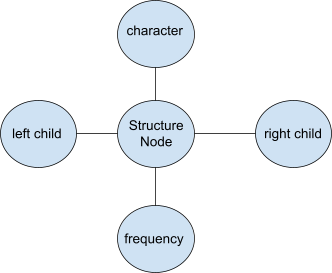
**Overview -**

Our code basically uses two techniques of text compression-

In **first case** basically the entire string given by the user is encoded i.e. the input string is assigned a variable length code which is then decoded back to the original string.

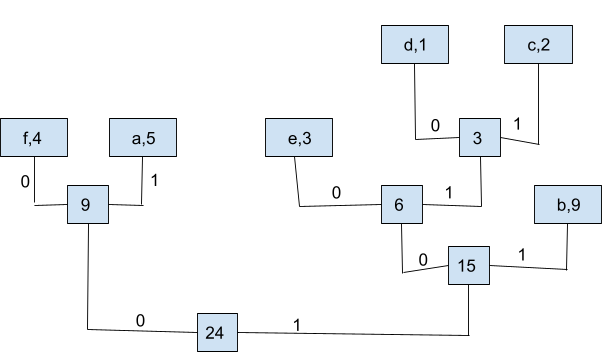
Here a structure is created for the huffman node which basically stores the character, its frequency and its left and right child, it also uses priority queue which stores the characters of string based on their frequency where lowest frequency character gets first priority.





Corresponding Huffman Tree for the input string let say s=” aaaaabbbbbbbbbccdeeeffff " is

Clearly frequencies are as - a = 5, b=9, c=2, d=1, e=3, f=4

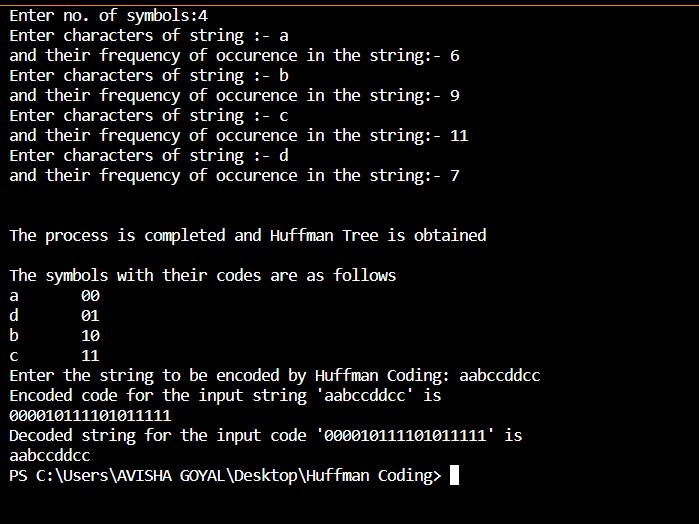


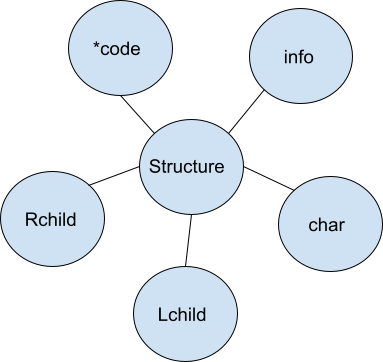
Clearly code for each character is - a = 01 , b = 11 , c = 1011 , d = 1010 , e = 100 , f = 00

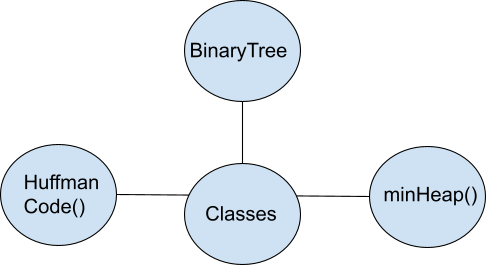
Therefore, encoded string is -

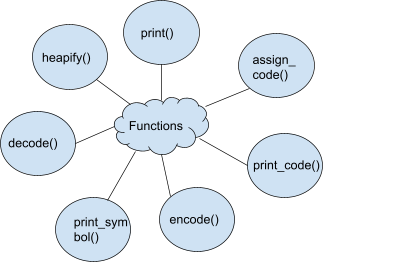
010101010111111111111111111110111011101010010010000000000

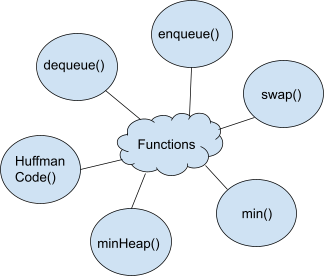
In **second** case the user enters characters with their frequency, the code encodes these characters then it asks the user to enter the string using these characters and then it encodes that string too. Then it also prints that decoded string.











# Conclusion -

Huffman coding comes as a very handy algorithm. It is a method of lossless compression. Lossless compression is valuable because it can reduce the amount of information (or in your computer, memory) needed to communicate the *exact same* message. That means that the process is perfectly invertible. Lossy compression on the other hand will lose information. The message that is reconstructed will be slightly different. Huffman coding creates a binary tree that is [guaranteed](http://www.cs.utoronto.ca/~brudno/csc373w09/huffman.pdf) to generate the most efficient way to compress your message. Thus, it helps in efficient transfer of message.

# Acknowledgment -

We have learned a lot and really enjoyed working on this project. We did not had much-advanced knowledge of C++ Language before starting this project but across this project, we learned how to implement our theoretical knowledge and make a successful project. We literally want to thank all who have helped up throughout this project. The Internet played a main role where we got to know about many new things and we were able to implement that in our project.