DATA STRUCTURES PROJECT REPORT

Muhammad Ahmed Baig 20i-1884 CS-C

Naufil Moten 20i-0642 CS-C

Objective: This project's purpose is to build a data compression method which is to read specific data from, our goal is to express the same data in a smaller amount of space. Our objective is to compress the text files.

Working: The code works in the way.

1st Task

1. Firstly, it asks the user to enter the file name if the user entered a wrong input the code will end.
2. After that we have used file handling and read the text from the text file. After that we have stored the text data in a string and passed in a duplicate code we made.
3. The duplicate code will remove all the duplicates from the string by adding an unused character to the duplicating positions.
4. After that we made a frequency code in the main that will count the frequency of the string and store it in a frequency array we made. After that we will output all the all the characters along with their frequencies.
5. Now we made a Node with the members of right (node pointer), left (node pointer), next (node pointer), ch (character), freq (int), code (string) and codelen (int).
6. After that we made a Tree class for our first Huffman Tree (not optimal) added a Node named root in it the functions in the Tree class is a constructor, an insert function and an display function (preorder).
7. In insert we have insert (passing the duplicated string) each node in a way that the first node will be inserted in the left and second node will be inserted in the right of the parent node which we will make before inside the insert function it will have junk value of character and frequency. After that we will attach that parent node to the left another parent node (newly made) and inserting the passed node to the right. This will continue until out last node is inserted from the main. That's how will make our first tree which is Task1.
8. After that we traverse the tree into binary. We did this by passing the node and the duplicated string to this function. This function will convert the string we passed into a binary code (character by character) and save it in the code data member of the Node and will also save it length in the codelen member of the node.
9. Finally we display our first Huffman (Non-Optimized) Tree in the output.

2nd Task

1. In this part we build an Optimized Huffman Tree.
2. Now the duplicated string into the Priority Queue we made. In the Priority Queue firstly we are prioritizing on the basis of frequency. For Enqueue, by inserting if the frequency of the head is more than the frequency of the new node which we have inserted, it will

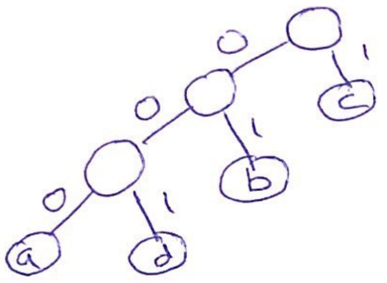
insert the new node in head and the head will be inserted in its next. If not, we will make a temp and insert head in the temp node and will insert it to next and next until we find the nodes whose frequency is greater than the first node. After that we will insert that new node before that node. For Dequeue, we will pass the node if head is NULL queue will be empty if not we will place the head in its next and after that return the node

3. After that we made the optimized Huffman Tree. To make the Huffman tree we first dequeue two nodes from the priority queue, and then sum their frequencies to make a new Parent node and add the two nodes on its left and right and enqueue the parent node back into the queue. This process is repeated until there is only one node in the queue which then becomes the Head of the Huffman tree.
4. After that we transverse the tree into binary. We did this by passing the node and the duplicated string to this function. This function will convert the string we passed into a binary code and save it in the code member of the Node and will also save it length in the codelen member of the node.
5. Finally we display our first Huffman (Optimized) Tree in the output.
6. Now we want to calculate the Average Bit Rate of both the trees and Formula for finding ABR is given in the project description: (ABR = (F1L1 + F2L2 + ... + FmLm)/N). N is the length of the binary coded string in this formula while Length corresponds to Binary Code of single character and F stands for frequency of that Character.
7. We have now both the Average Bit Rates, we will divide the greater one which would be uncompressed ABR with smaller which would be compressed ABR.
8. This marks the end of our project.

Sample Output is given on the next page. Thank you for reading this report out.

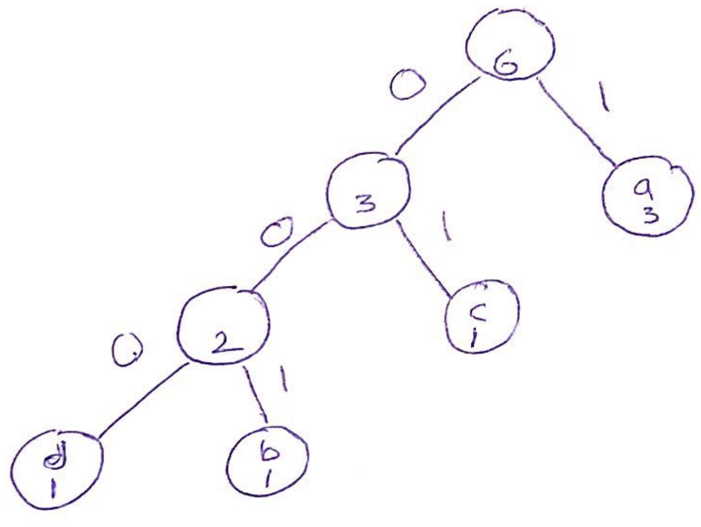
Input : aadbac
After Duplication : adbc
Frequencies : a = 3, d = 1, b = 1, c = 1

Task 1 : (Not Optimized Huffman)



Character	Binary	Length
a	000	3
d	001	3
b	01	2
c	1	1

Tast 2 : (Optimized Huffman)



Character	Binary	Length
d	000	3
b	001	3
c	01	2
a	1	1