Task1 Implementation Explanation:

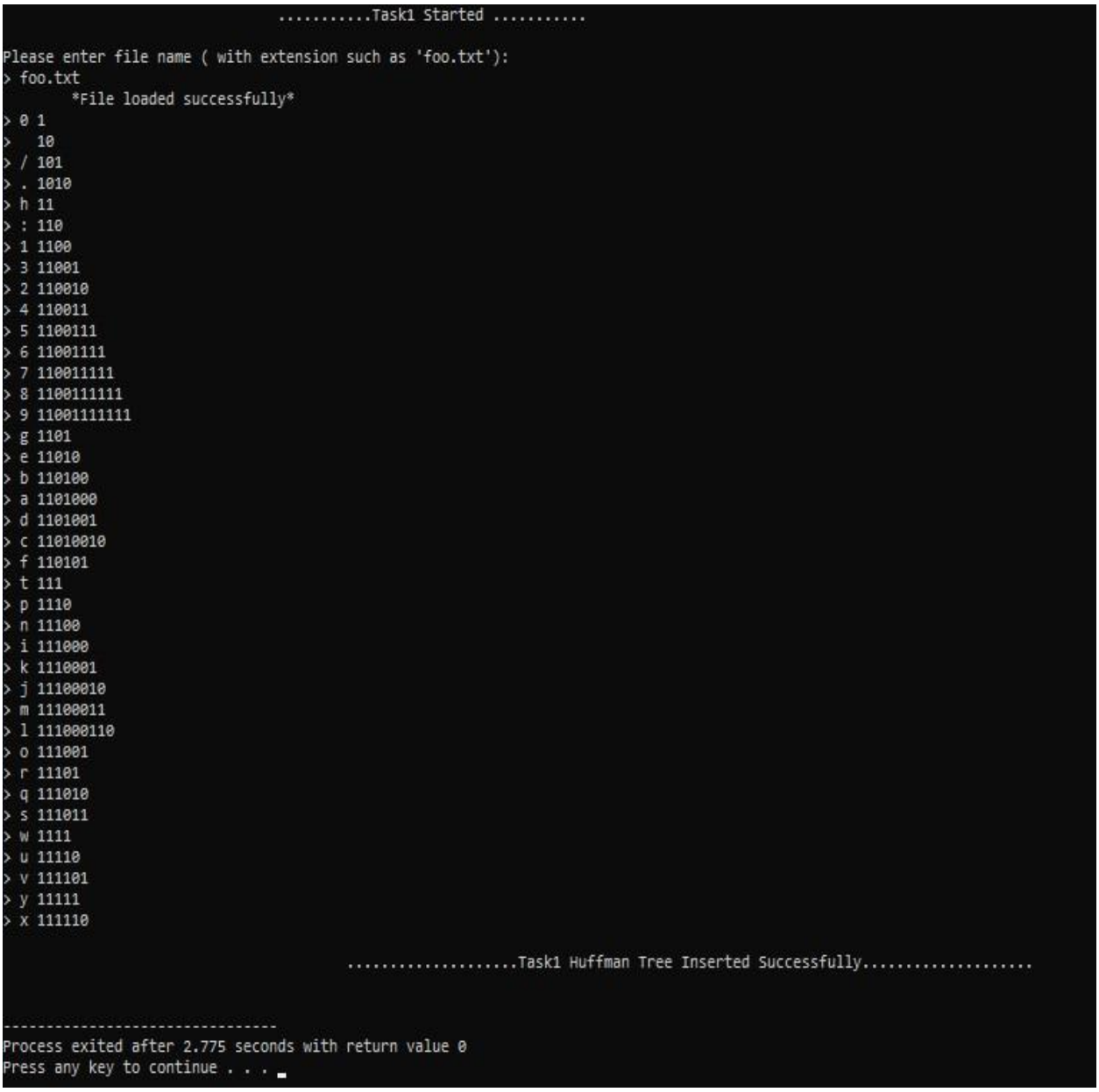
Firstly, user will be asked to enter file name. Then if file doesn't exist message of **file not loaded** will be displayed on screen otherwise **file loaded successfully.**

After that file be read by character by character and store in string. After that from **for loop** character by character from string will be sent to Tree function where at a time Encoding will be calculated and will be placed with character node.

This tree function is almost like binary search tree function, but it includes some checks and constraints on which basis insertion is of **Huffman Coding** type and in these checks Encoding part is also done accordingly.

Now afterwards complete insertion characters along encoded bits will be displayed through **Pre Order display** function. And task1 ends successfully.

Task1 Console Screen Shot:



Task 2 Implementation Explanation:

Firstly, user will be asked to enter file name. Then if file doesn't exist message of **file not loaded** will be displayed on screen otherwise **file loaded successfully.**

After that file is read character by character and all of the characters are stored in string. From there we iterate through the string character by character and insert it in the queue.

The data is stored in ascending order. After that Huffman function is called. Using this we will make a tree. In the Huffman function we use recursion. In it the function we use the a function which takes the top two nodes and creates a new node which has those two nodes as children and deletes these two top nodes and inserts the newly created node at its particular location. The function is called recursively until there is only one node left in the tree. After that we call a function encoding which encodes each character according to Huffman logic.

At last we also calculate the compression ratio. We iterate the tree using In-order logic. In this we multiply the frequency of character with the length of encoded character. Then we add it to a global variable. At last this sum of all the character's frequency and their encoding length is divided by overall frequency. Thus, we get our compression ratio of optimal Huffman code.

Task 2 Console Screen Shot: