

**UNIVERSITY INSTITUTE OF ENGINEERING**

**Department of Computer Science & Engineering**

**Subject Name:** Competitive Coding

**Subject Code:** 20CSP-314

**Submitted to: Submitted by:**

Er. Mamta Punia Name: Sahil Kaundal

UID: 21BCS8197

Section: 616

Group: A

**INDEX**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ex. No** | **List of Experiments** | **Conduct (MM: 12)** | **Viva**  **(MM: 10)** | **Record (MM: 8)** | **Total**  **(MM: 30)** | **Remarks/Signature** |
| 1 | To demonstrate the concept of Array. |  |  |  |  |  |
| 2 | To demonstrate the concept of Stack and Queue. |  |  |  |  |  |
| 3 | To demonstrate the concept of Linked List. |  |  |  |  |  |
| 4 | Sorting and Searching: Implement the concept of Searching and Sorting techniques. |  |  |  |  |  |
| 5 | To implement the concept of Graphs. |  |  |  |  |  |
| 6. | To demonstrate the concept of Tree Data Structure |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Experiment 6.1**

**Student Name:** Sahil Kaundal **UID:** 21BCS8197

**Branch:** BE CSE (Lateral Entry) **Section/Group:** 616/A

**Semester:** 5th **Date of Performance:** 07/10/2022

**Subject Name:** CC Lab **Subject Code:** 20CSP-314

1. **Aim/Overview of the practical:**

To demonstrate the concept of Tree Data Structure

You are given a pointer to the root of a binary search tree and values to be inserted into the tree. Insert the values into their appropriate position in the binary search tree and return the root of the updated binary tree. You just have to complete the function.

<https://www.hackerrank.com/challenges/binary-search-tree-insertion/problem?isFullScreen=true>

1. **Apparatus / Simulator Used:**

* Windows 7 or above
* Google Chrome

1. **Objective:**

* To understand the concept of trees.
* To implement the concept of trees.

**4. Code:**

import java.util.\*;

import java.io.\*;

class Node {

    Node left;

    Node right;

    int data;

    Node(int data) {

        this.data = data;

        left = null;

        right = null;

    }

}

class Solution {

    public static void preOrder( Node root ) {

        if( root == null)

            return;

        System.out.print(root.data + " ");

        preOrder(root.left);

        preOrder(root.right);

    }

 /\* Node is defined as :

 class Node

    int data;

    Node left;

    Node right;

    \*/

public static Node insert(Node root,int value)

{

    if(root == null) {

        root = new Node(value);

    } else if(value < root.data){

        root.left = insert(root.left,value);

    } else if(value > root.data) {

        root.right = insert(root.right,value);

    }

    return root;

}

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int t = scan.nextInt();

        Node root = null;

        while(t-- > 0) {

            int data = scan.nextInt();

            root = insert(root, data);

        }

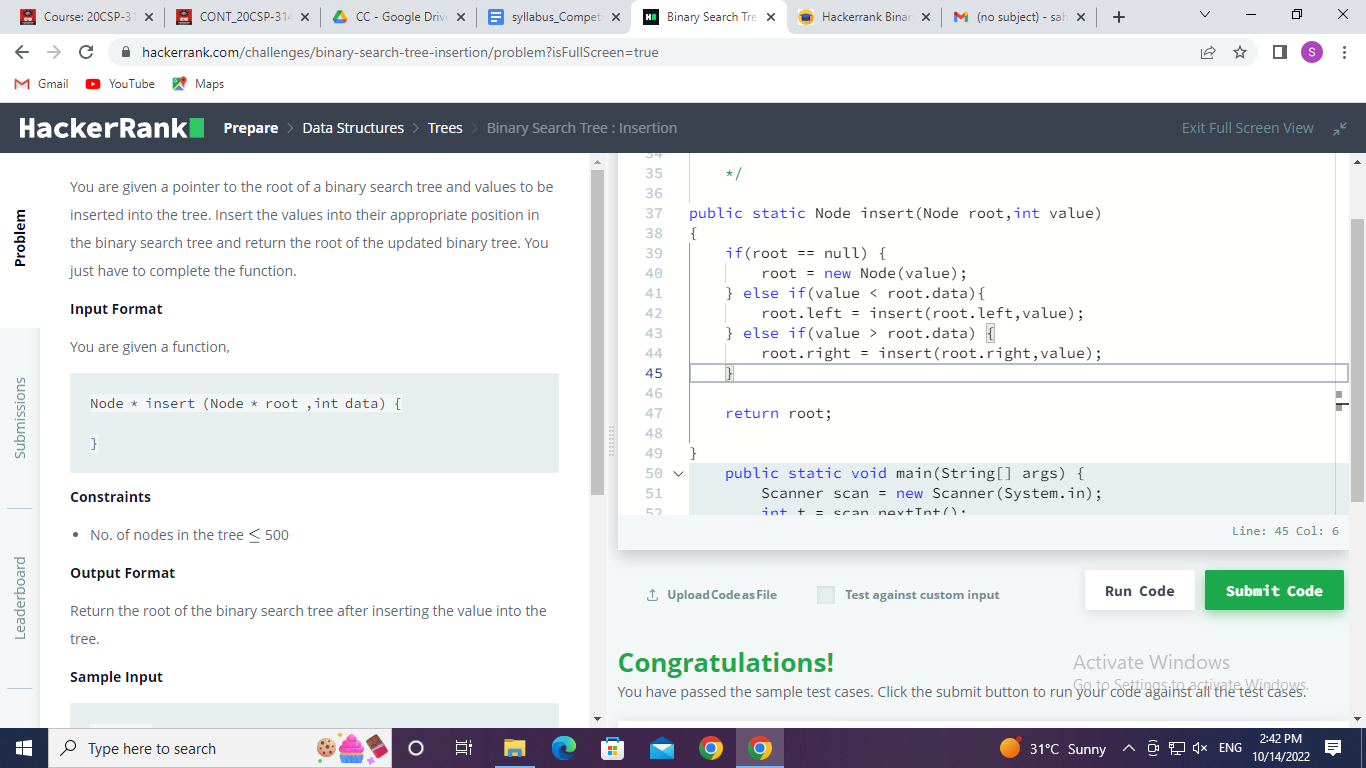
        scan.close();

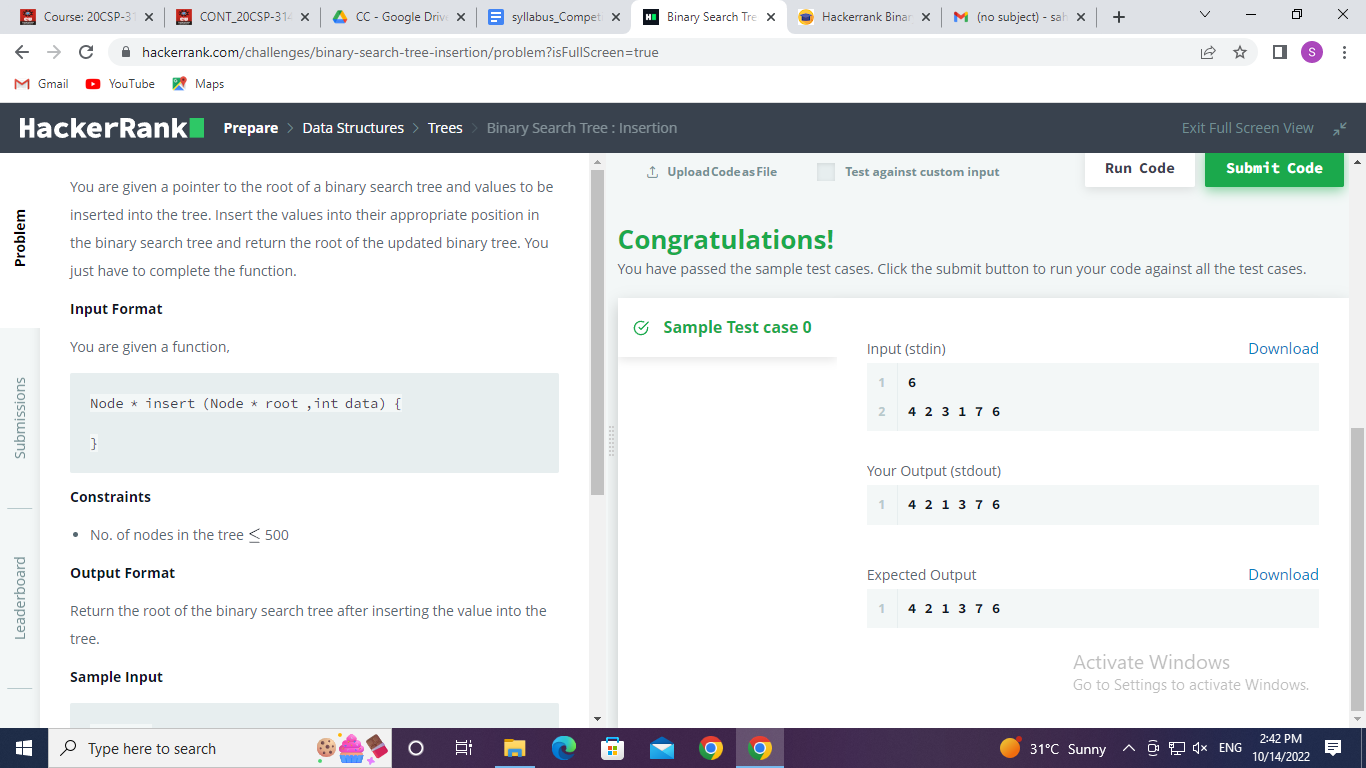
        preOrder(root);

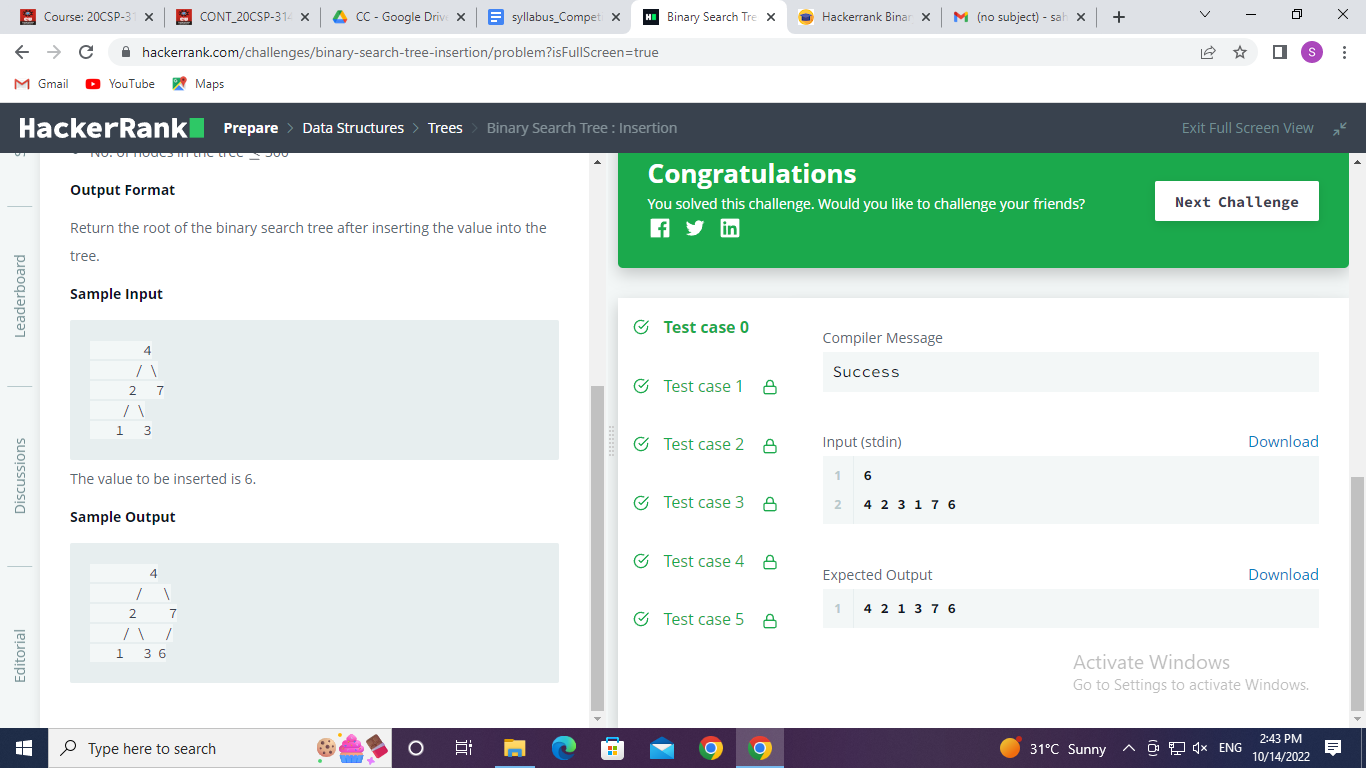
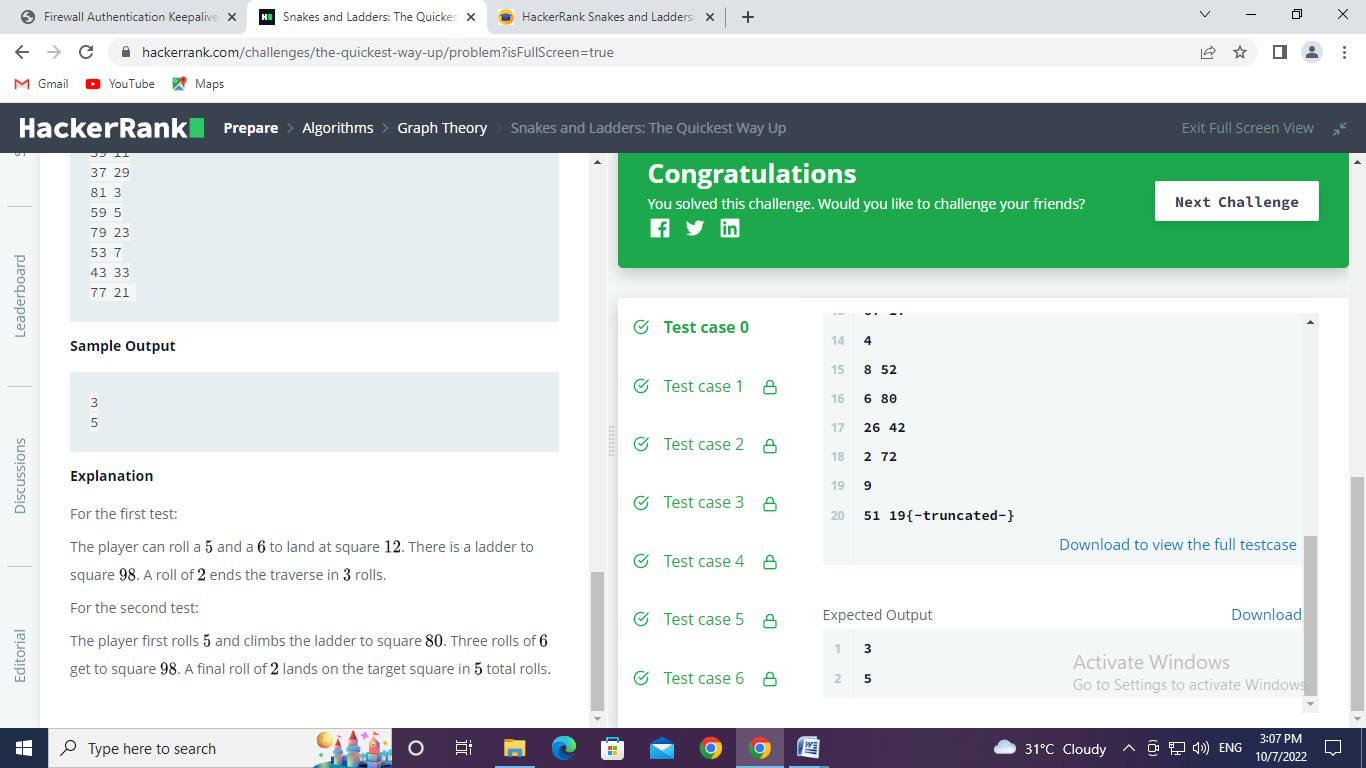
    }

}

**5. Result/Output/Writing Summary:**







**Experiment 6.2**

1. **Aim/Overview of the practical:**

To demonstrate the concept of Tree Data Structure

[Huffman coding](https://en.wikipedia.org/wiki/Huffman_coding) assigns variable length codewords to fixed length input characters based on their frequencies. More frequent characters are assigned shorter codewords and less frequent characters are assigned longer codewords. All edges along the path to a character contain a code digit. If they are on the left side of the tree, they will be a 0 (zero). If on the right, they'll be a 1 (one). Only the leaves will contain a letter and its frequency count. All other nodes will contain a null instead of a character, and the count of the frequency of all of it and its descendant characters.

[**https://www.hackerrank.com/challenges/tree-huffman-decoding/problem?isFullScreen=true**](https://www.hackerrank.com/challenges/tree-huffman-decoding/problem?isFullScreen=true)

1. **Apparatus / Simulator Used:**

* Windows 7 or above
* Google Chrome

1. **Objective:**

* To understand the concept of trees.
* To implement the concept of trees.

1. **Code:**

import java.util.\*;

abstract class Node implements Comparable<Node> {

    public  int frequency; // the frequency of this tree

    public  char data;

    public  Node left, right;

    public Node(int freq) {

        frequency = freq;

    }

    // compares on the frequency

    public int compareTo(Node tree) {

        return frequency - tree.frequency;

    }

}

class HuffmanLeaf extends Node {

    public HuffmanLeaf(int freq, char val) {

        super(freq);

        data = val;

    }

}

class HuffmanNode extends Node {

    public HuffmanNode(Node l, Node r) {

        super(l.frequency + r.frequency);

        left = l;

        right = r;

    }

}

class Decoding {

/\*

  class Node

    public  int frequency; // the frequency of this tree

      public  char data;

      public  Node left, right;

\*/

  void decode(String S, Node root)

{

    StringBuilder sb = new StringBuilder();

    Node c = root;

    for (int i = 0; i < S.length(); i++) {

        c = S.charAt(i) == '1' ? c.right : c.left;

        if (c.left == null && c.right == null) {

            sb.append(c.data);

            c = root;

        }

    }

    System.out.print(sb);

}

}

public class Solution {

    // input is an array of frequencies, indexed by character code

    public static Node buildTree(int[] charFreqs) {

        PriorityQueue<Node> trees = new PriorityQueue<Node>();

        // initially, we have a forest of leaves

        // one for each non-empty character

        for (int i = 0; i < charFreqs.length; i++)

            if (charFreqs[i] > 0)

                trees.offer(new HuffmanLeaf(charFreqs[i], (char)i));

        assert trees.size() > 0;

        // loop until there is only one tree left

        while (trees.size() > 1) {

            // two trees with least frequency

            Node a = trees.poll();

            Node b = trees.poll();

            // put into new node and re-insert into queue

            trees.offer(new HuffmanNode(a, b));

        }

        return trees.poll();

    }

    public static Map<Character,String> mapA=new HashMap<Character ,String>();

    public static void printCodes(Node tree, StringBuffer prefix) {

        assert tree != null;

        if (tree instanceof HuffmanLeaf) {

            HuffmanLeaf leaf = (HuffmanLeaf)tree;

            // print out character, frequency, and code for this leaf (which is just the prefix)

            //System.out.println(leaf.data + "\t" + leaf.frequency + "\t" + prefix);

            mapA.put(leaf.data,prefix.toString());

        } else if (tree instanceof HuffmanNode) {

            HuffmanNode node = (HuffmanNode)tree;

            // traverse left

            prefix.append('0');

            printCodes(node.left, prefix);

            prefix.deleteCharAt(prefix.length()-1);

            // traverse right

            prefix.append('1');

            printCodes(node.right, prefix);

            prefix.deleteCharAt(prefix.length()-1);

        }

    }

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        String test= input.next();

        // we will assume that all our characters will have

        // code less than 256, for simplicity

        int[] charFreqs = new int[256];

        // read each character and record the frequencies

        for (char c : test.toCharArray())

            charFreqs[c]++;

        // build tree

        Node tree = buildTree(charFreqs);

        // print out results

        printCodes(tree, new StringBuffer());

        StringBuffer s = new StringBuffer();

        for(int i = 0; i < test.length(); i++) {

          char c = test.charAt(i);

          s.append(mapA.get(c));

        }

        //System.out.println(s);

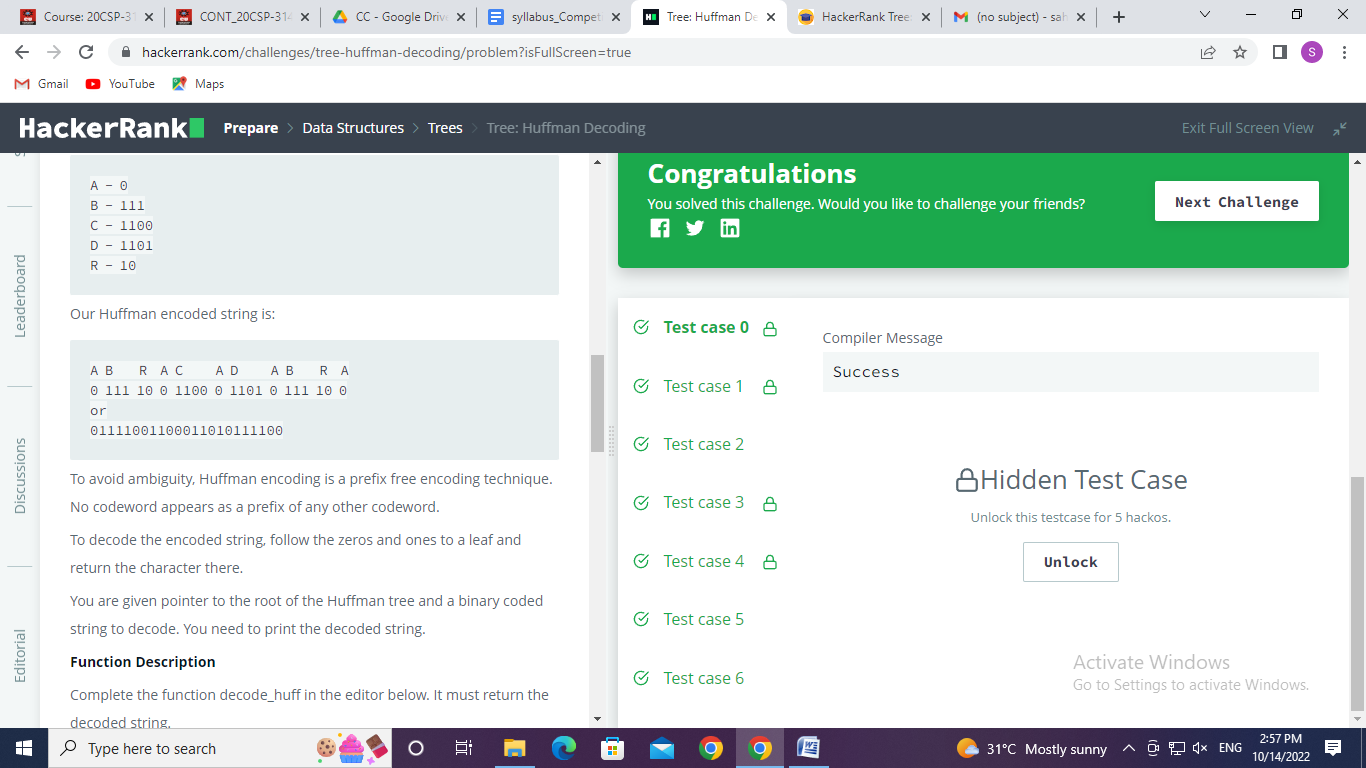
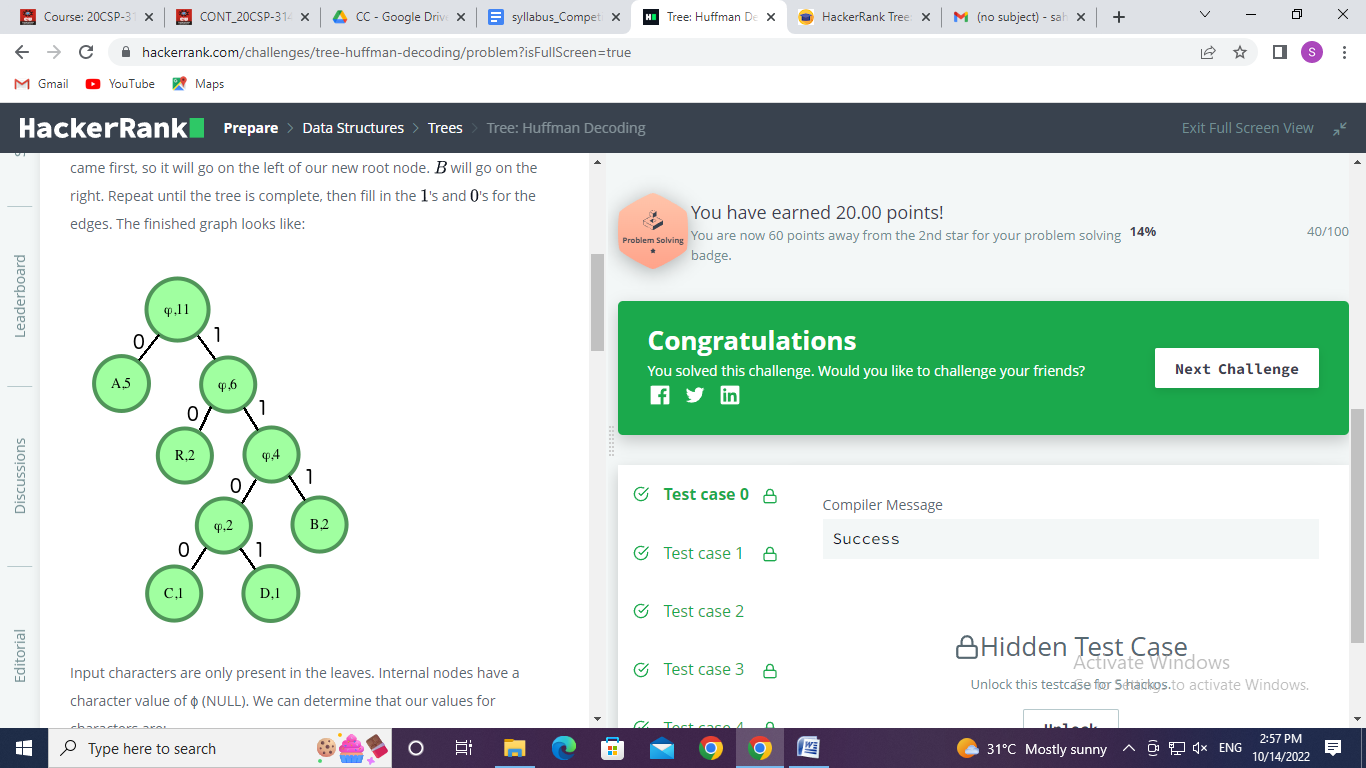
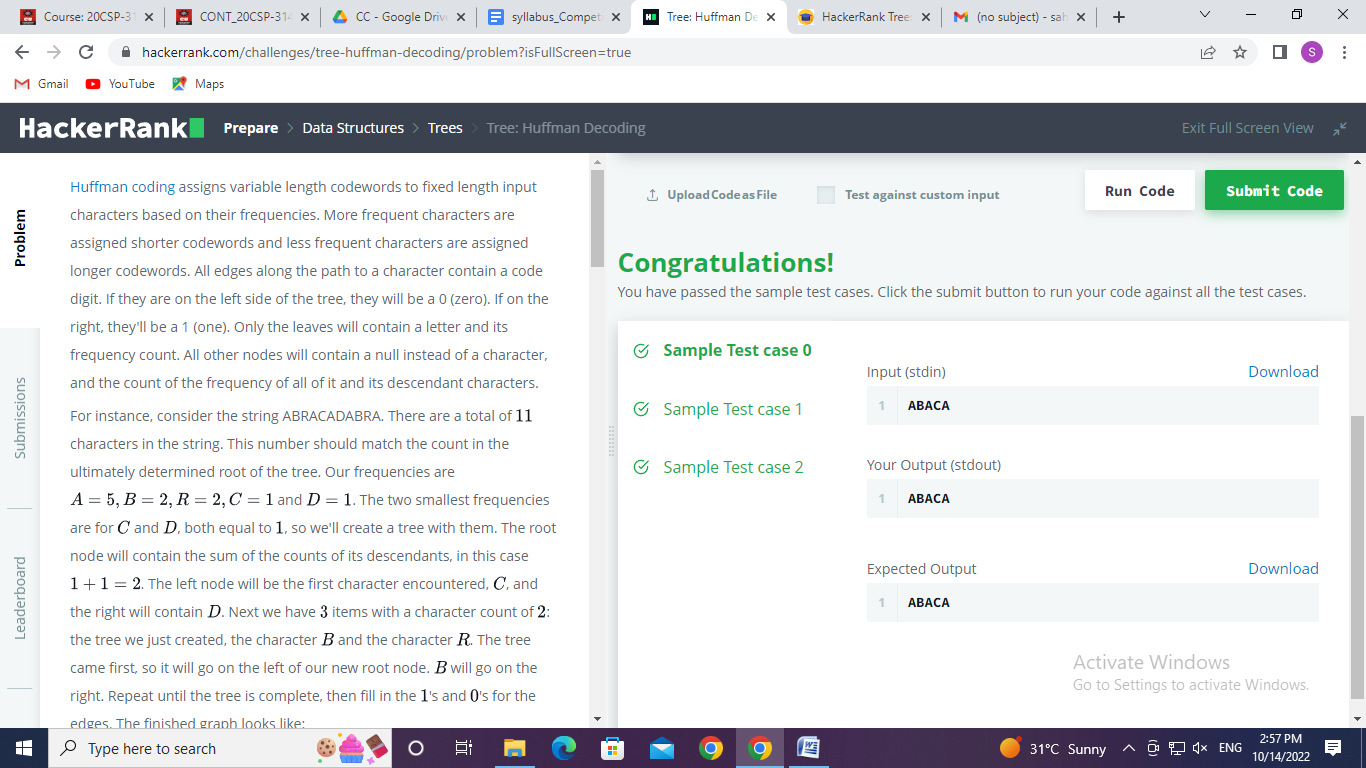
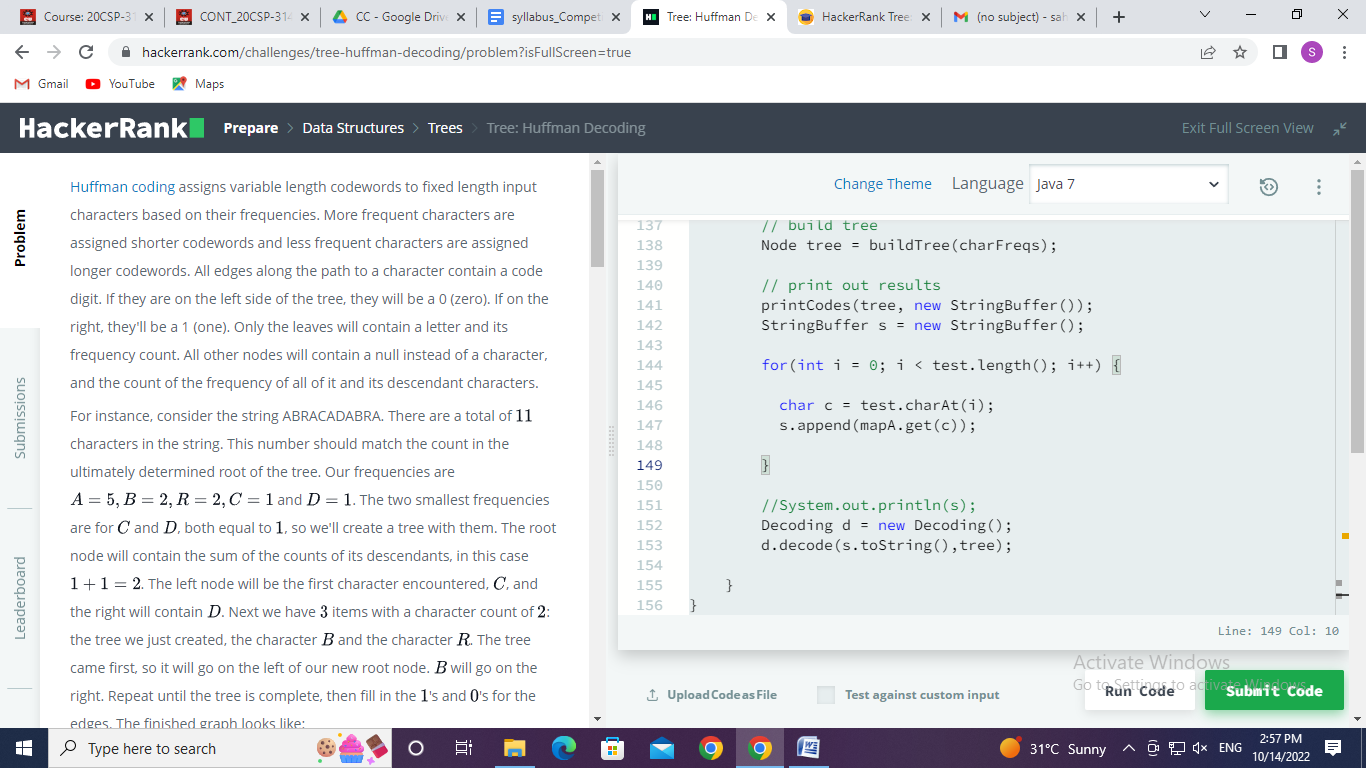
        Decoding d = new Decoding();

        d.decode(s.toString(),tree);

    }

}

1. **Result/Output/Writing Summary:**



**Learning outcomes (What I have learnt):**

* Learned about the concept of trees.
* Learned about implement the concept of trees.
* Learned about the Huffman Coding concept using trees.

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. |  |  |  |
|  |  |  |  |