Experiment 6

Student Name: Bhanu Pundir **UID:** 20BCS1439
Branch: BE CSE **Section/Group:** 620-B
Semester: 5th **Date of Performance:** 07 Oct 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

2. **Apparatus / Simulator Used:**
   - Windows 7 or above
   - Google Chrome

3. **Objective:**
   - To understand the concept of trees.
   - To implement the concept of trees.

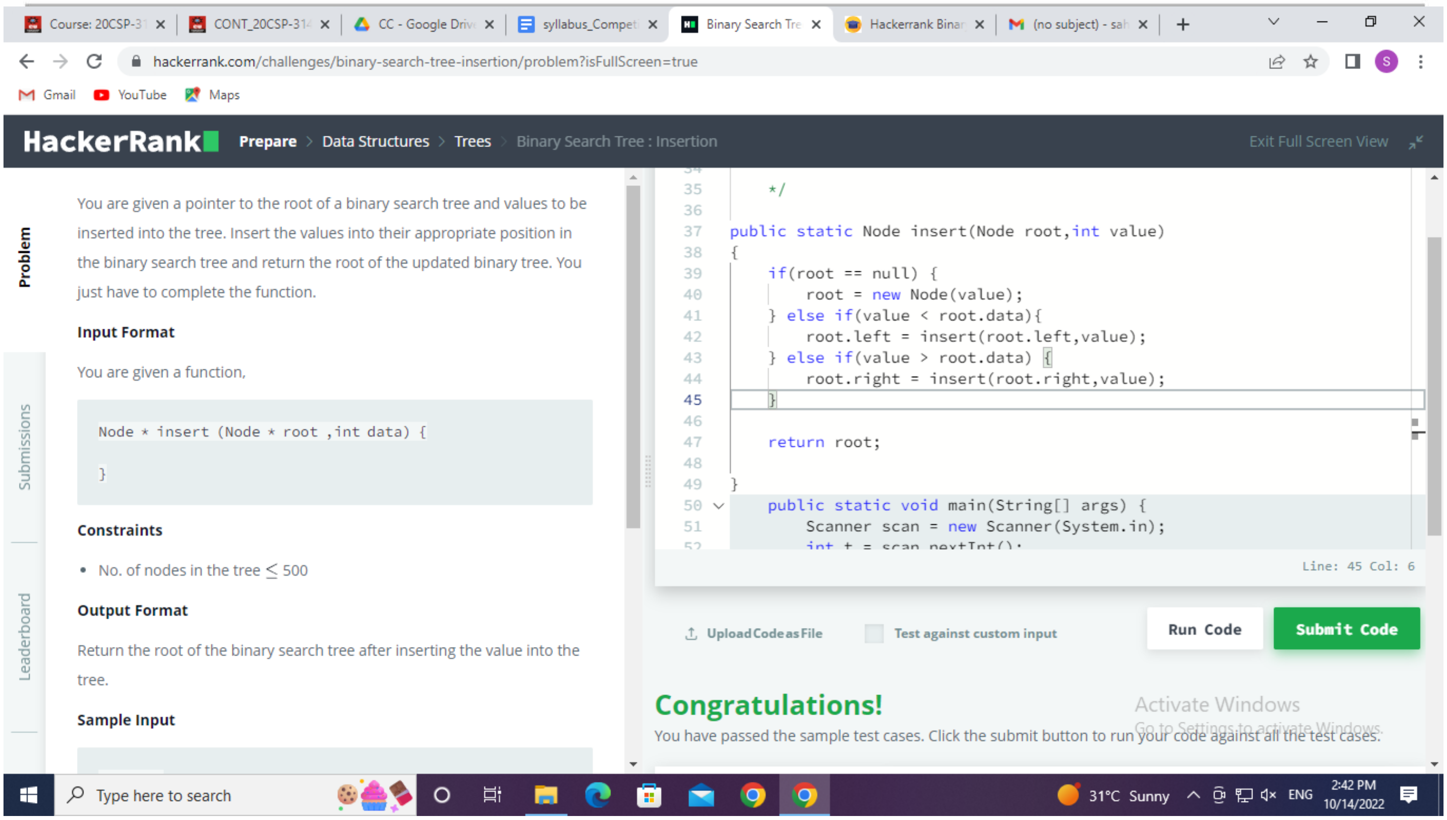
4. **Code:**

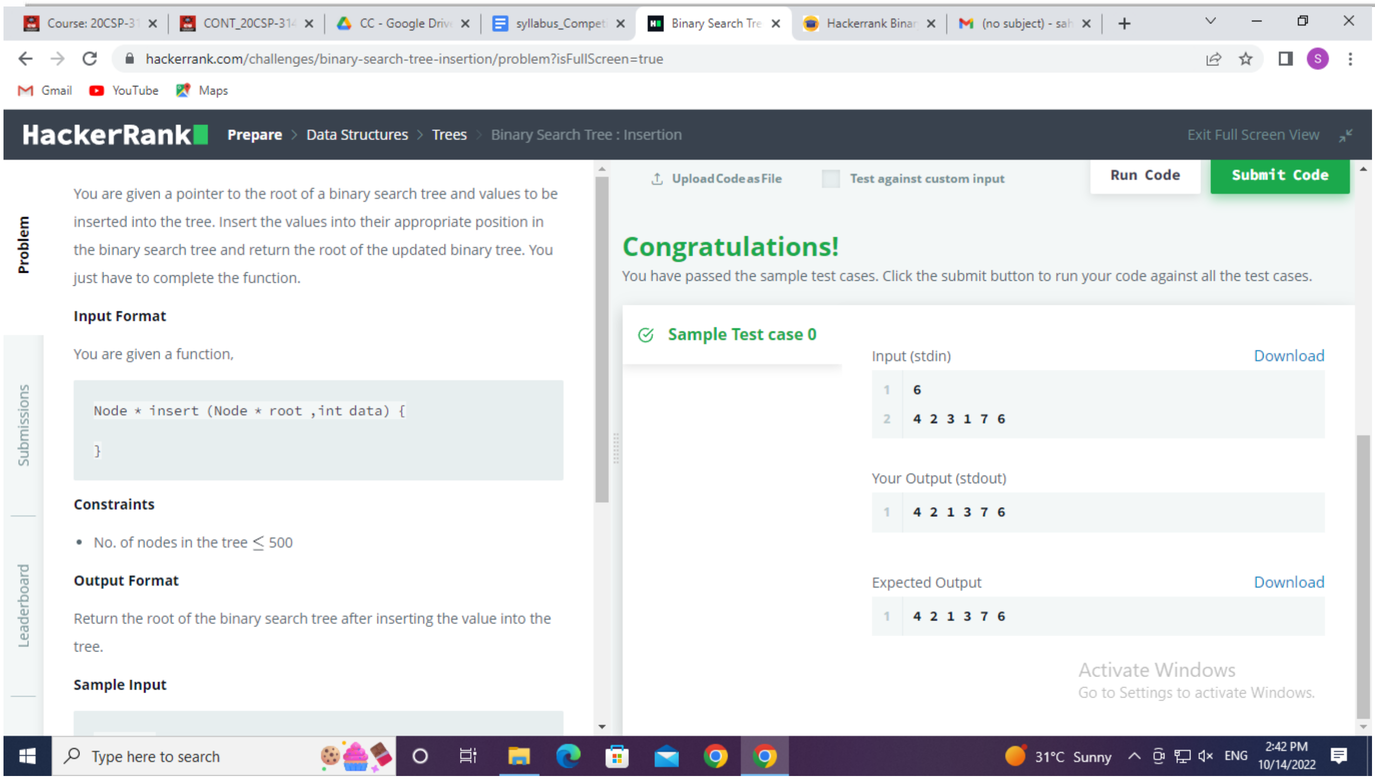
```
import java.util.*;
import java.io.*;

class Node {
    Node left;
    Node right;
    int data;

    Node(int data) {
```

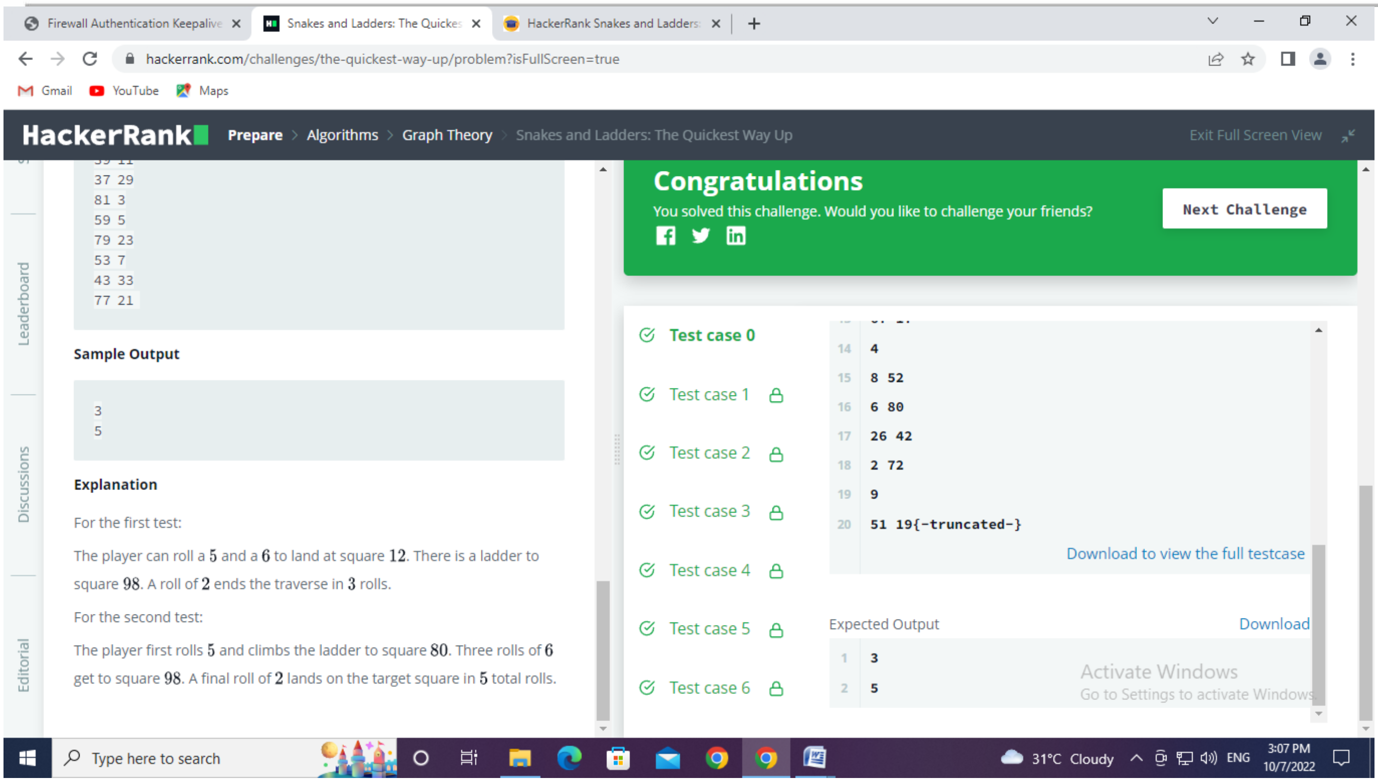
```
        this.data = data;
        left = null;
        right = null;
    }
}

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) {
```

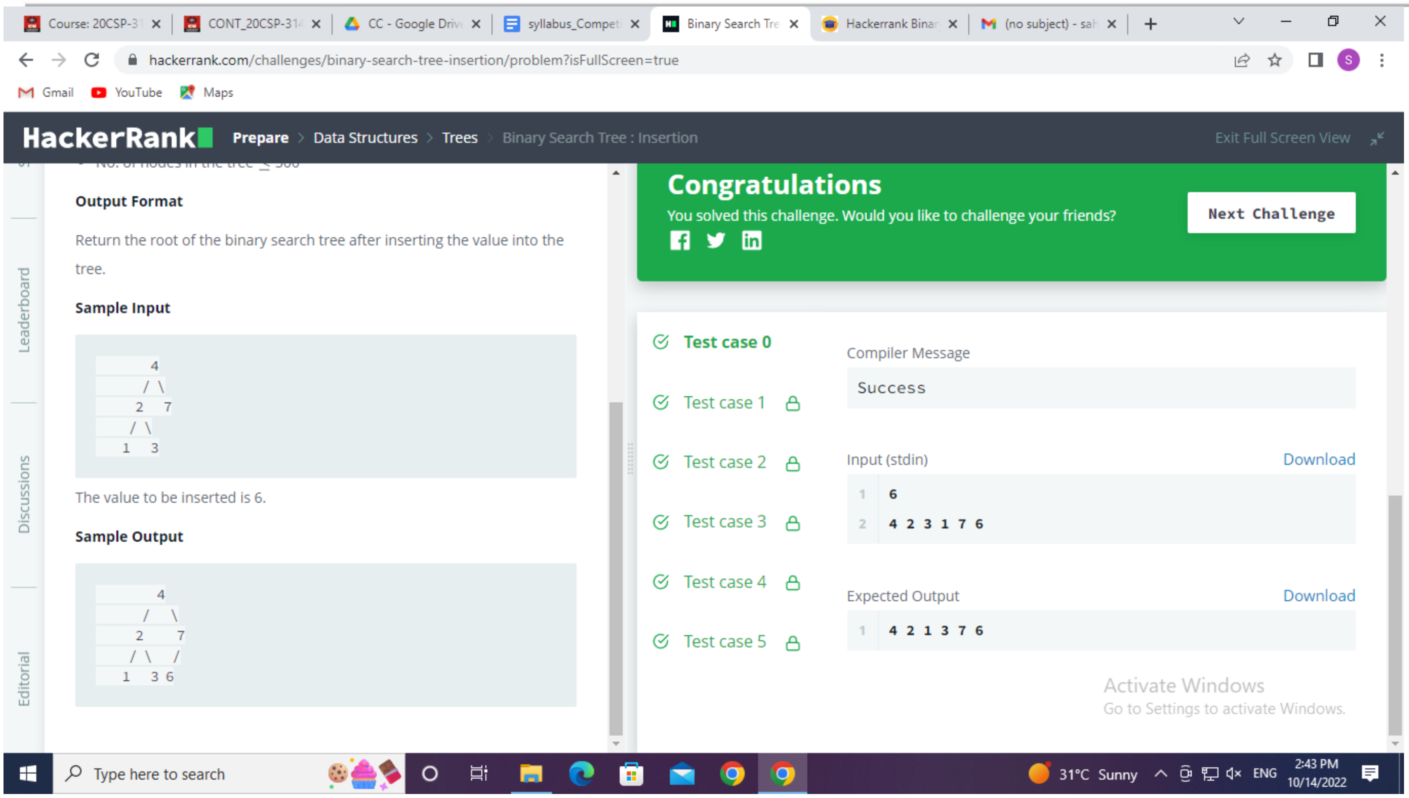
```
        root.right = insert(root.right,value);
    }

    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:


hackerrank.com/challenges/binary-search-tree-insertion/problem?isFullScreen=true
HackerRank
Prepare > Data Structures > Trees > Binary Search Tree : Insertion
Exit Full Screen View
Problem
Submissions
Leaderboard
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.
Input Format
You are given a function,
Node * insert (Node * root ,int data) {
}
Constraints
No. of nodes in the tree ≤ 500
Output Format
Return the root of the binary search tree after inserting the value into the tree.
Sample Input
*/
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);
Line: 45 Col: 6
Upload Code as File
Test against custom input
Run Code
Submit Code
Congratulations!
You have passed the sample test cases. Click the submit button to run your code against all the test cases.
Type here to search
31°C Sunny
ENG
2:42 PM
10/14/2022


hackerrank.com/challenges/binary-search-tree-insertion/problem?isFullScreen=true
HackerRank
Prepare > Data Structures > Trees > Binary Search Tree : Insertion
Exit Full Screen View
Problem
Submissions
Leaderboard
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.
Input Format
You are given a function,
Node * insert (Node * root ,int data) {
}
Constraints
No. of nodes in the tree ≤ 500
Output Format
Return the root of the binary search tree after inserting the value into the tree.
Sample Input
Upload Code as File
Test against custom input
Run Code
Submit Code
Congratulations!
You have passed the sample test cases. Click the submit button to run your code against all the test cases.
Sample Test case 0
Input (stdin)
Download
6
4 2 3 1 7 6
Your Output (stdout)
4 2 1 3 7 6
Expected Output
Download
4 2 1 3 7 6
Activate Windows
Go to Settings to activate Windows.
Type here to search
31°C Sunny
ENG
2:42 PM
10/14/2022


Firewall Authentication Keepalive
Snakes and Ladders: The Quicke
HackerRank Snakes and Ladders
hackerrank.com/challenges/the-quickest-way-up/problem?isFullScreen=true
Gmail
YouTube
Maps
HackerRank
Prepare > Algorithms > Graph Theory > Snakes and Ladders: The Quickest Way Up
Exit Full Screen View
37 29
81 3
59 5
79 23
53 7
43 33
77 21
Sample Output
3
5
Explanation
For the first test:
The player can roll a 5 and a 6 to land at square 12. There is a ladder to square 98. A roll of 2 ends the traverse in 3 rolls.
For the second test:
The player first rolls 5 and climbs the ladder to square 80. Three rolls of 6 get to square 98. A final roll of 2 lands on the target square in 5 total rolls.
Leaderboard
Discussions
Editorial
Congratulations
You solved this challenge. Would you like to challenge your friends?
Next Challenge
Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6
14 4
15 8 52
16 6 80
17 26 42
18 2 72
19 9
20 51 19{-truncated-}
Download to view the full testcase
Expected Output
Download
1 3
2 5
Activate Windows
Go to Settings to activate Windows.
Type here to search
31°C Cloudy
ENG
3:07 PM
10/7/2022



Experiment 6.2

1. Aim/Overview of the practical:

To demonstrate the concept of Tree Data Structure

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

2. Apparatus / Simulator Used:

- Windows 7 or above
- Google Chrome

3. Objective:

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

4. 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);
```

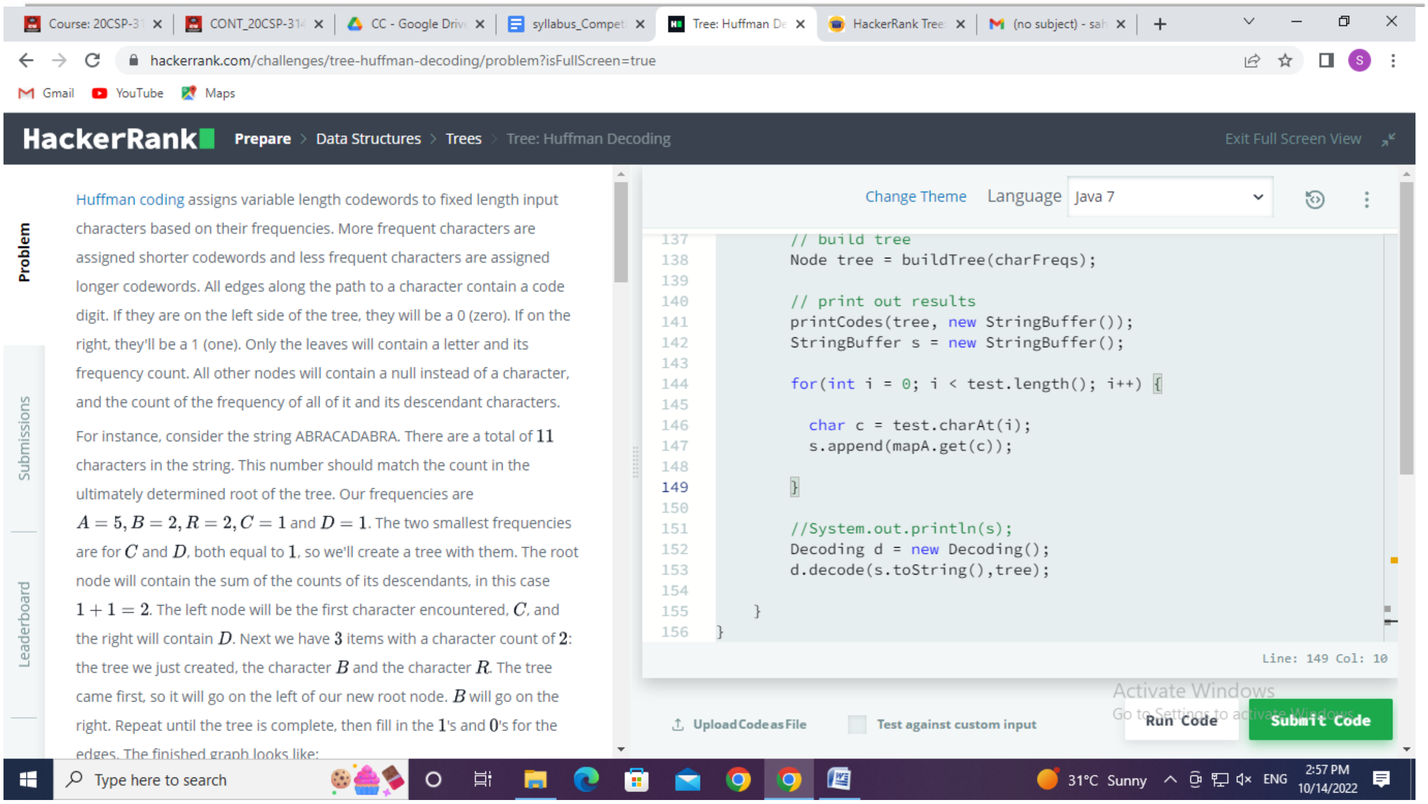
```
        data = val;
    }

}

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;
```

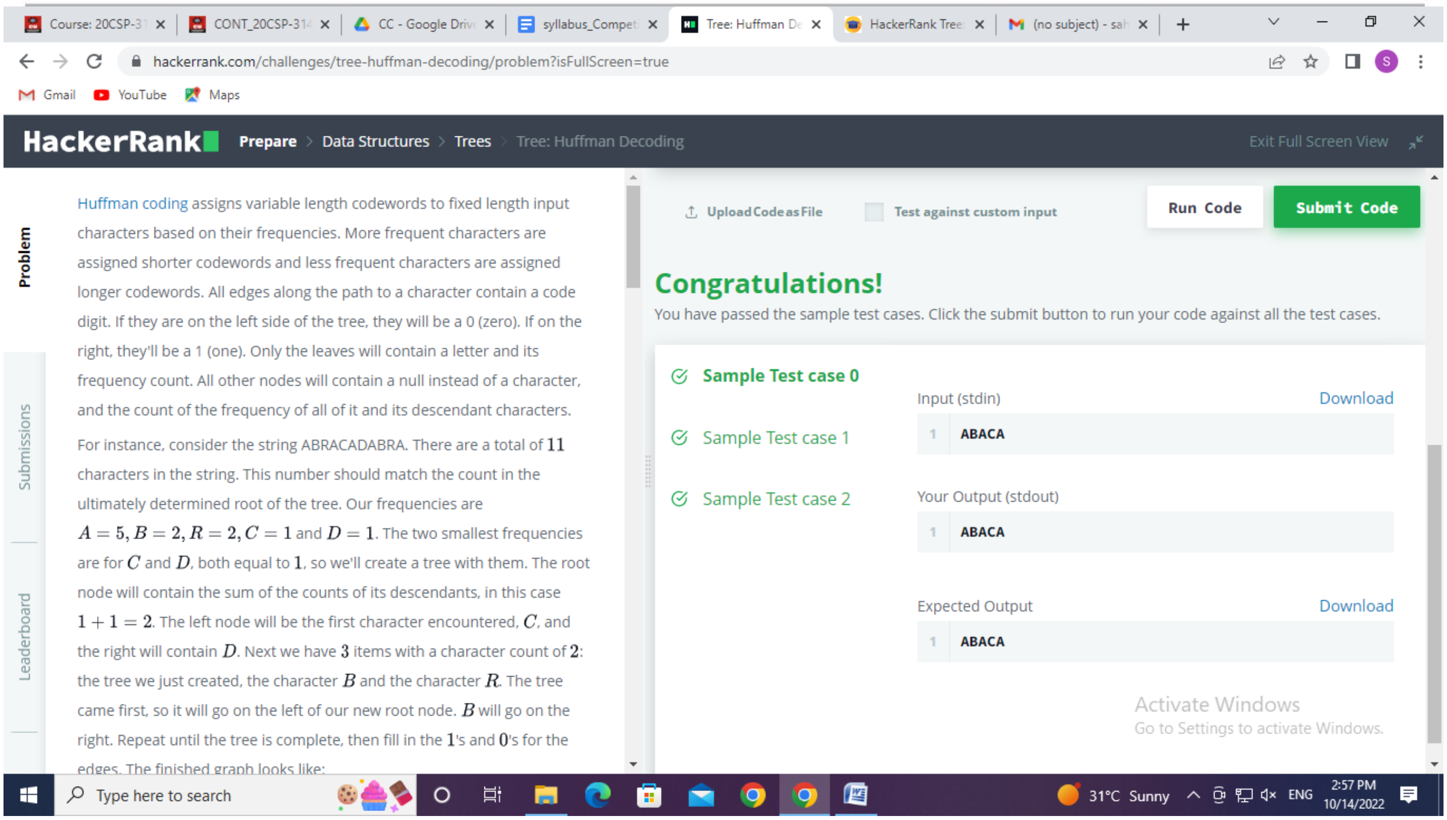
```
        }
    }
    System.out.print(sb);
}
}
public class Solution {

    // 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) {
```

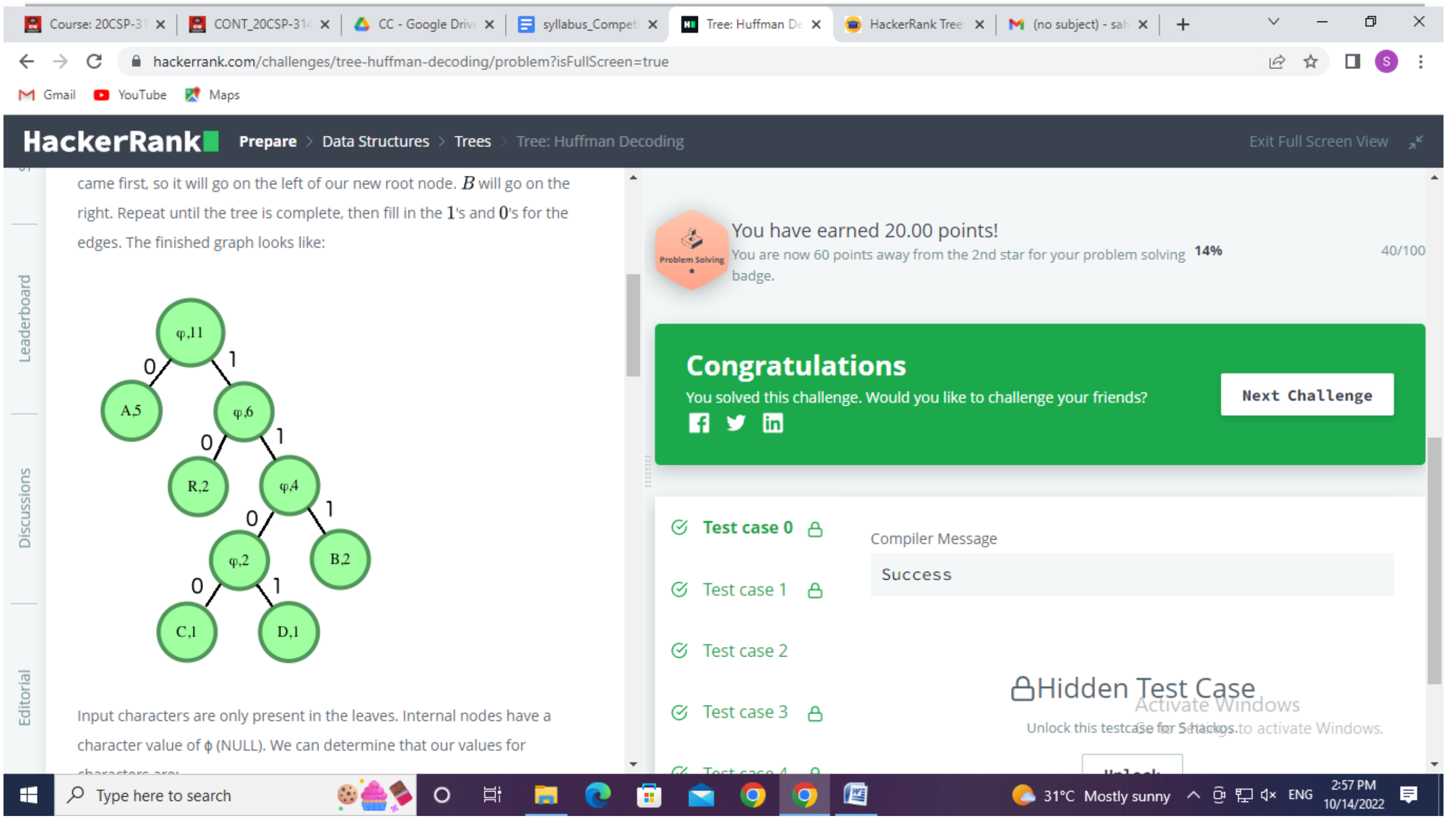
```
        assert tree != null;

        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" + pre
fix);
            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
```

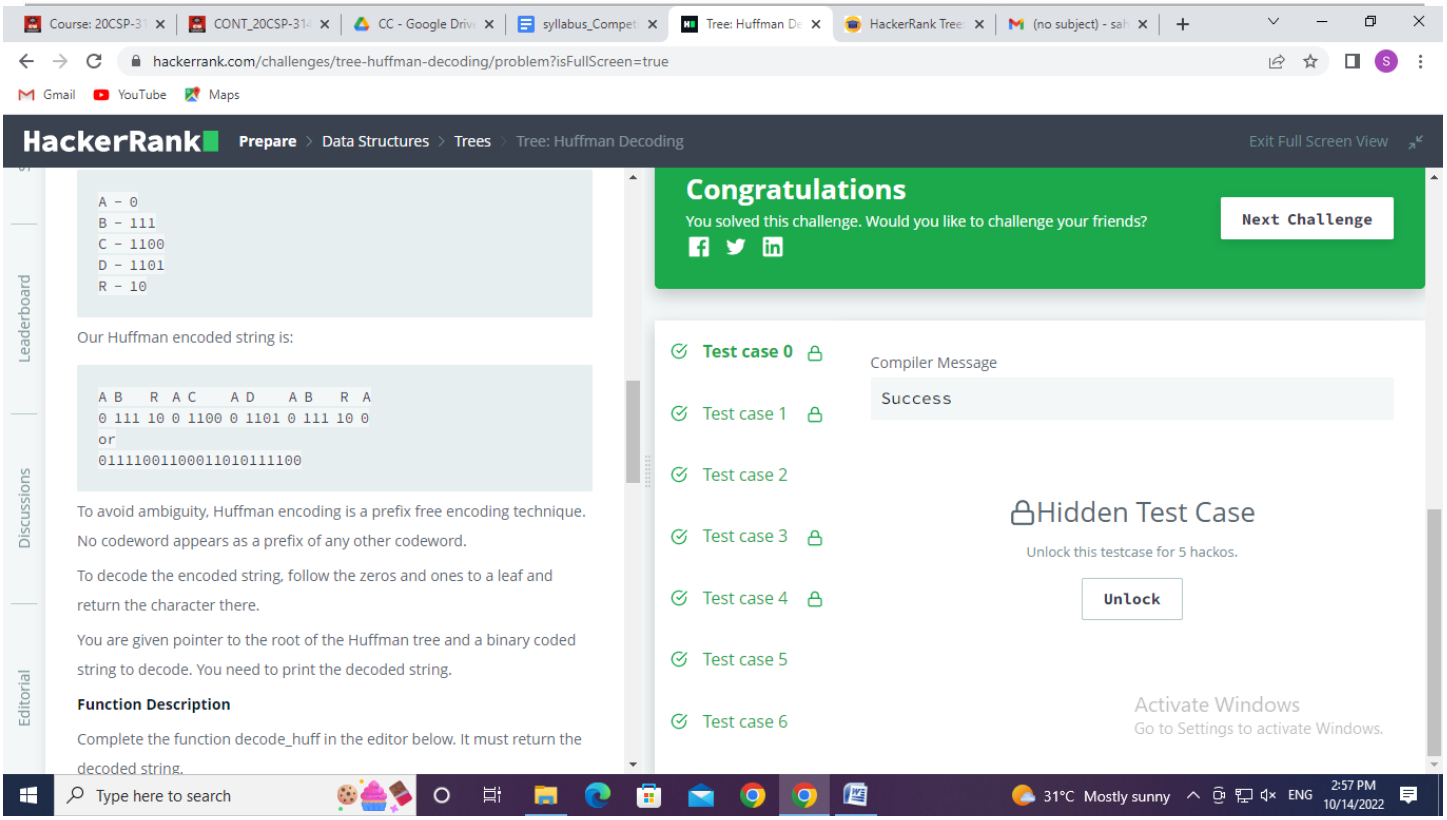
```java
        int[] charFreqs = new int[256];

        // 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);
    }
}
```

5. Result/Output/Writing Summary:


hackerrank.com/challenges/tree-huffman-decoding/problem?isFullScreen=true
HackerRank Prepare > Data Structures > Trees > Tree: Huffman Decoding
Exit Full Screen View
Problem
Submissions
Leaderboard
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.
For instance, consider the string ABRACADABRA. There are a total of 11 characters in the string. This number should match the count in the ultimately determined root of the tree. Our frequencies are A = 5, B = 2, R = 2, C = 1 and D = 1. The two smallest frequencies are for C and D, both equal to 1, so we'll create a tree with them. The root node will contain the sum of the counts of its descendants, in this case 1 + 1 = 2. The left node will be the first character encountered, C, and the right will contain D. Next we have 3 items with a character count of 2: the tree we just created, the character B and the character R. The tree came first, so it will go on the left of our new root node. B will go on the right. Repeat until the tree is complete, then fill in the 1's and 0's for the edges. The finished graph looks like:
Change Theme
Language
Java 7
// 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);
}
}
Line: 149 Col: 10
Upload Code as File
Test against custom input
Run Code
Submit Code
Type here to search
31°C Sunny
ENG
2:57 PM
10/14/2022


hackerrank.com/challenges/tree-huffman-decoding/problem?isFullScreen=true
HackerRank
Prepare > Data Structures > Trees > Tree: Huffman Decoding
Exit Full Screen View
Problem
Submissions
Leaderboard
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.
For instance, consider the string ABRACADABRA. There are a total of 11 characters in the string. This number should match the count in the ultimately determined root of the tree. Our frequencies are A = 5, B = 2, R = 2, C = 1 and D = 1. The two smallest frequencies are for C and D, both equal to 1, so we'll create a tree with them. The root node will contain the sum of the counts of its descendants, in this case 1 + 1 = 2. The left node will be the first character encountered, C, and the right will contain D. Next we have 3 items with a character count of 2: the tree we just created, the character B and the character R. The tree came first, so it will go on the left of our new root node. B will go on the right. Repeat until the tree is complete, then fill in the 1's and 0's for the edges. The finished graph looks like:
Upload Code as File
Test against custom input
Run Code
Submit Code
Congratulations!
You have passed the sample test cases. Click the submit button to run your code against all the test cases.
Sample Test case 0
Sample Test case 1
Sample Test case 2
Input (stdin)
Download
ABACA
Your Output (stdout)
ABACA
Expected Output
Download
ABACA
Activate Windows
Go to Settings to activate Windows.
Type here to search
31°C Sunny
ENG
2:57 PM
10/14/2022


hackerrank.com/challenges/tree-huffman-decoding/problem?isFullScreen=true
HackerRank
Prepare > Data Structures > Trees > Tree: Huffman Decoding
Exit Full Screen View
came first, so it will go on the left of our new root node. B will go on the right. Repeat until the tree is complete, then fill in the 1's and 0's for the edges. The finished graph looks like:
φ,11
A,5
φ,6
R,2
φ,4
φ,2
B,2
C,1
D,1
Input characters are only present in the leaves. Internal nodes have a character value of ϕ (NULL). We can determine that our values for
You have earned 20.00 points!
You are now 60 points away from the 2nd star for your problem solving badge.
14%
40/100
Congratulations
You solved this challenge. Would you like to challenge your friends?
Next Challenge
Test case 0
Test case 1
Test case 2
Test case 3
Compiler Message
Success
Hidden Test Case
Leaderboard
Discussions
Editorial
31°C Mostly sunny
2:57 PM
10/14/2022



Learning outcomes (What I have learnt):

- Learned about the concept of trees.
- Learned about implementing 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.			