Experiment 5

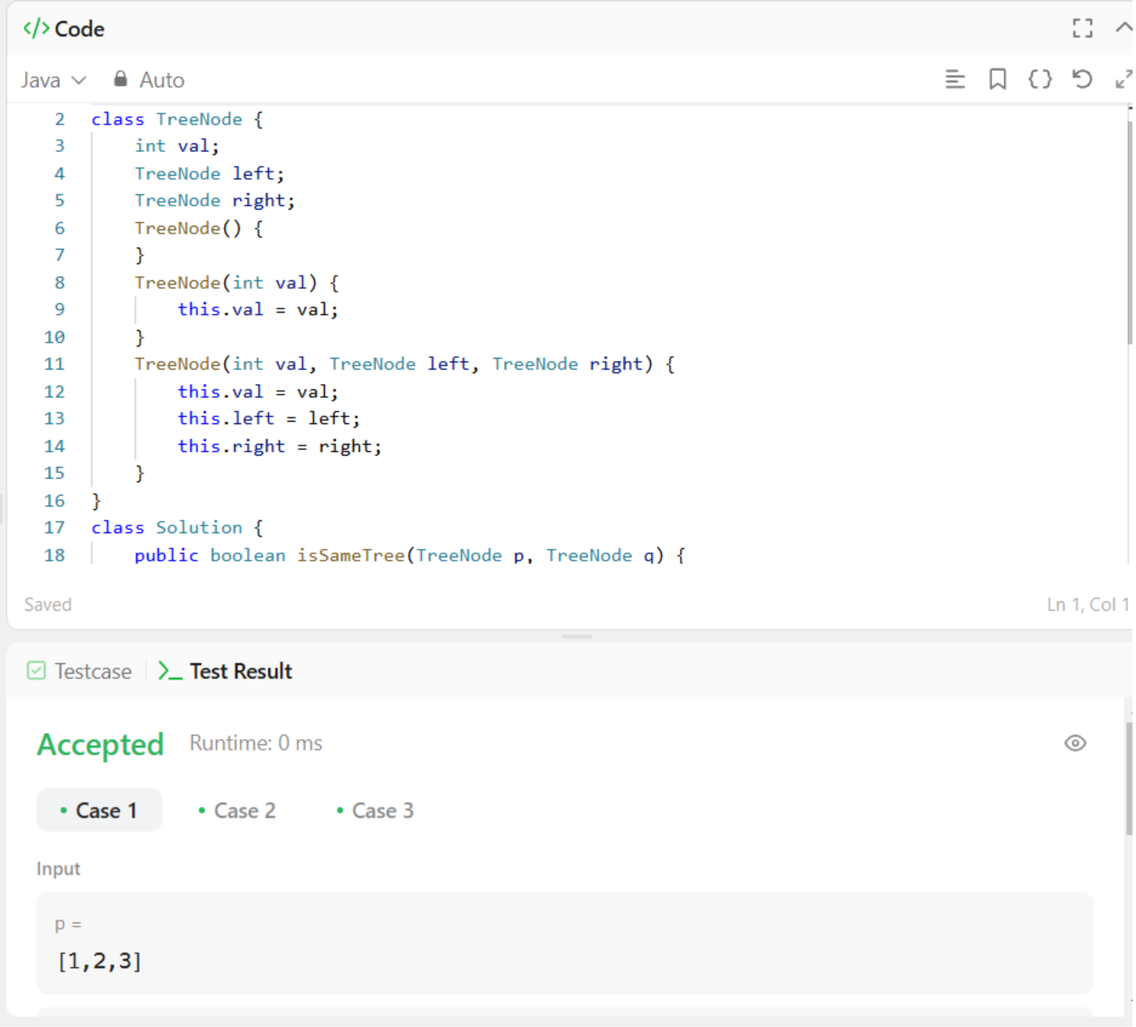
Student Name: Amit Kumar **UID:22BCS14056**

Branch: BE-CSE **Section/Group:637/B**

Semester:6 **Date of Performance:14-2-25**

Subject Name: AP LAB **Subject Code:22CSH-359**

1. **Problem statement-** Given the roots of two binary trees p and q, write a function to check if they are the same or not.Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.



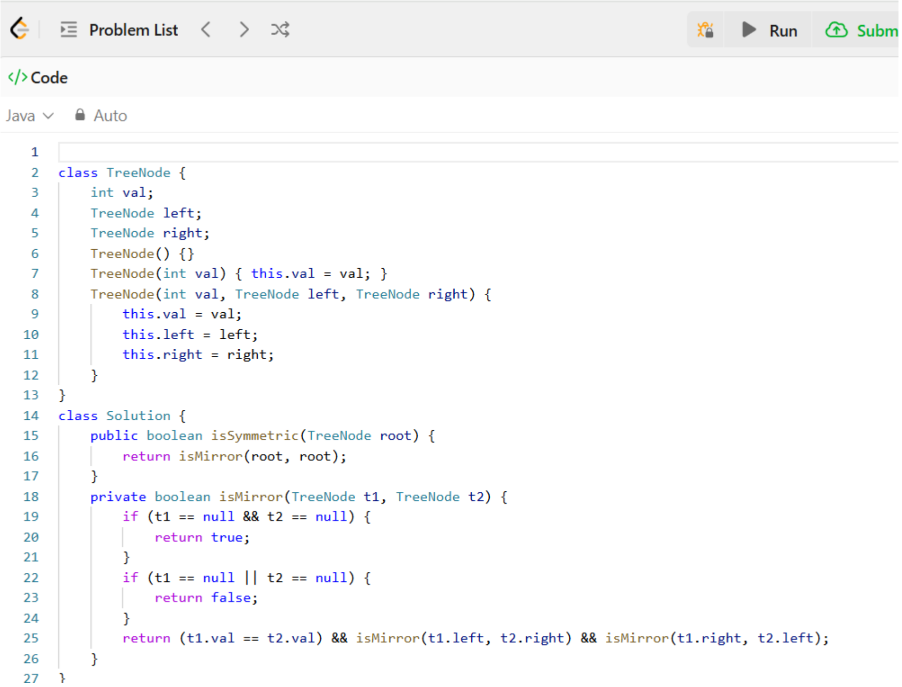
```java
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode() {
    }
    TreeNode(int val) {
        this.val = val;
    }
    TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}
class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {
```

Accepted Runtime: 0 ms

Input

```
p =
[1,2,3]
```

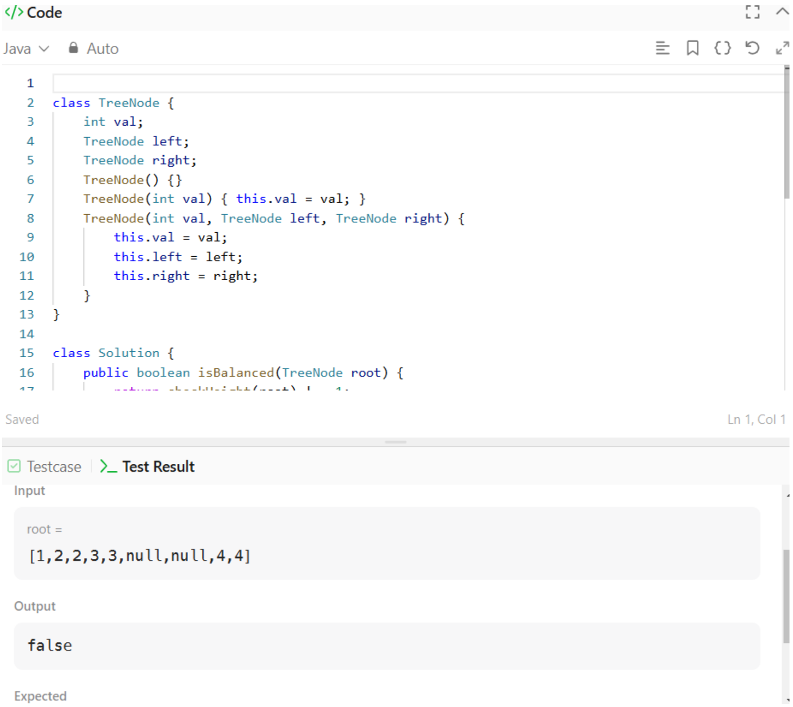
2. Problem statement- Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center)



```java

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode() {}
    TreeNode(int val) { this.val = val; }
    TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}
class Solution {
    public boolean isSymmetric(TreeNode root) {
        return isMirror(root, root);
    }
    private boolean isMirror(TreeNode t1, TreeNode t2) {
        if (t1 == null && t2 == null) {
            return true;
        }
        if (t1 == null || t2 == null) {
            return false;
        }
        return (t1.val == t2.val) && isMirror(t1.left, t2.right) && isMirror(t1.right, t2.left);
    }
}
```

3. Problem statement- Given a binary tree, determine if it is height-balanced.



```java

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode() {}
    TreeNode(int val) { this.val = val; }
    TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}

class Solution {
    public boolean isBalanced(TreeNode root) {
        return checkHeight(root) != -1;
```

Input

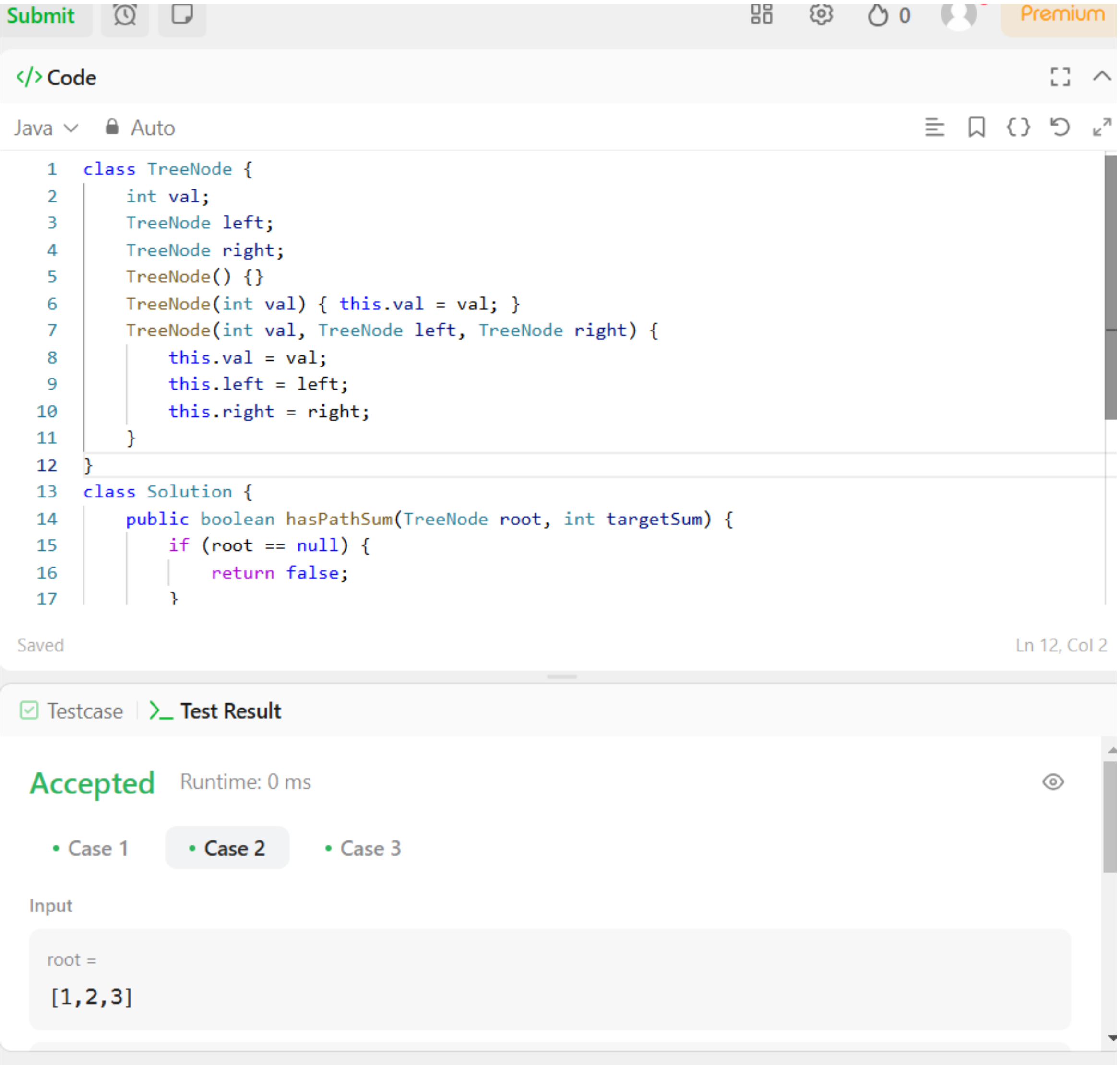
```
root =
[1,2,2,3,3,null,null,4,4]
```

Output

```
false
```

Expected

4. Problem statement- Given the root of a binary tree and an integer targetSum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals targetSum.



```java
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode() {}
    TreeNode(int val) { this.val = val; }
    TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}
class Solution {
    public boolean hasPathSum(TreeNode root, int targetSum) {
        if (root == null) {
            return false;
        }
```

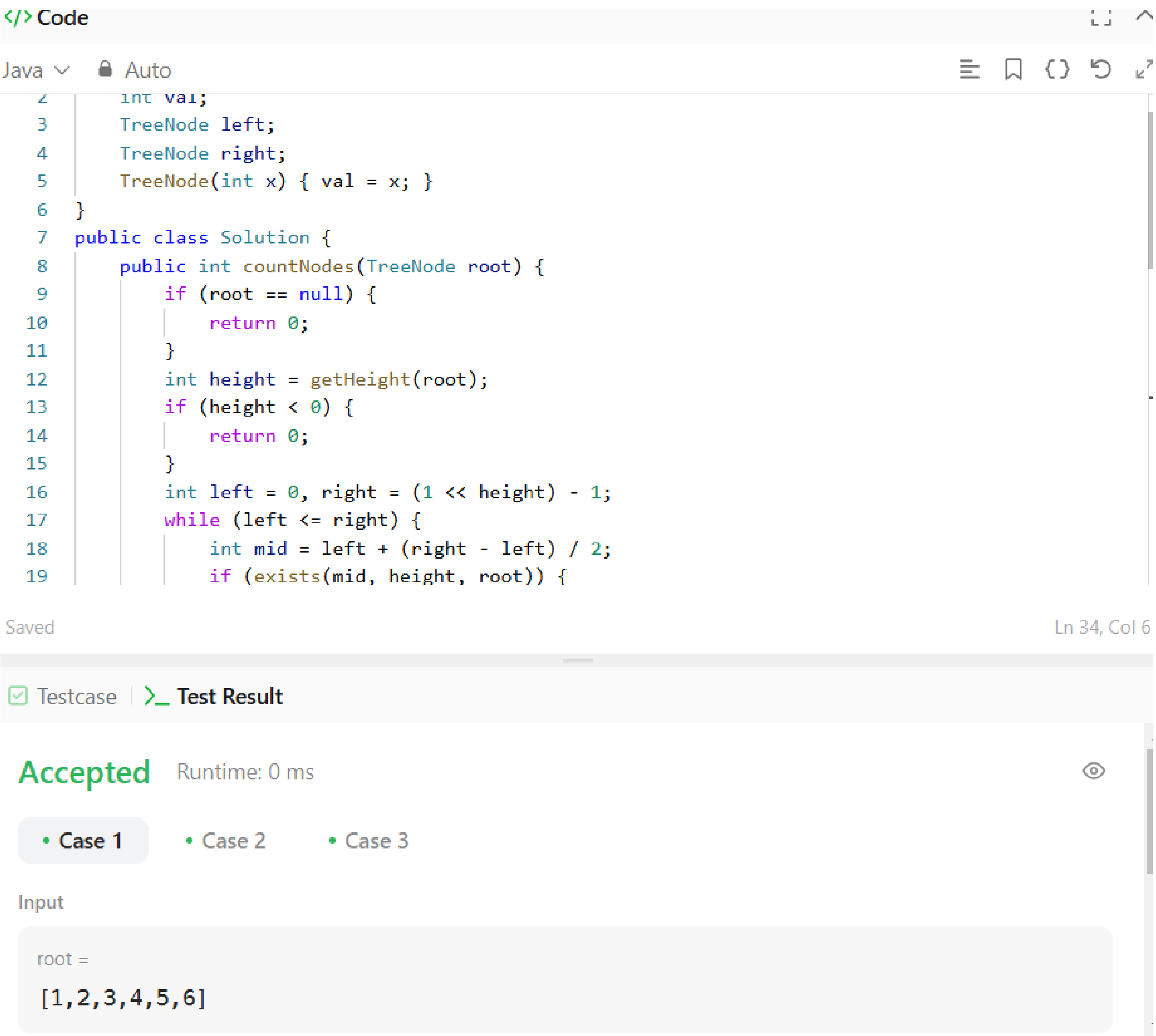
Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

root =
[1,2,3]

5. Problem statement- Given the root of a complete binary tree, return the number of the nodes in the tree. According to Wikipedia, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes CO3 inclusive at the last level h. Design an algorithm that runs in less than O(n) time complexity



```java
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}
public class Solution {
    public int countNodes(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int height = getHeight(root);
        if (height < 0) {
            return 0;
        }
        int left = 0, right = (1 << height) - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (exists(mid, height, root)) {
```

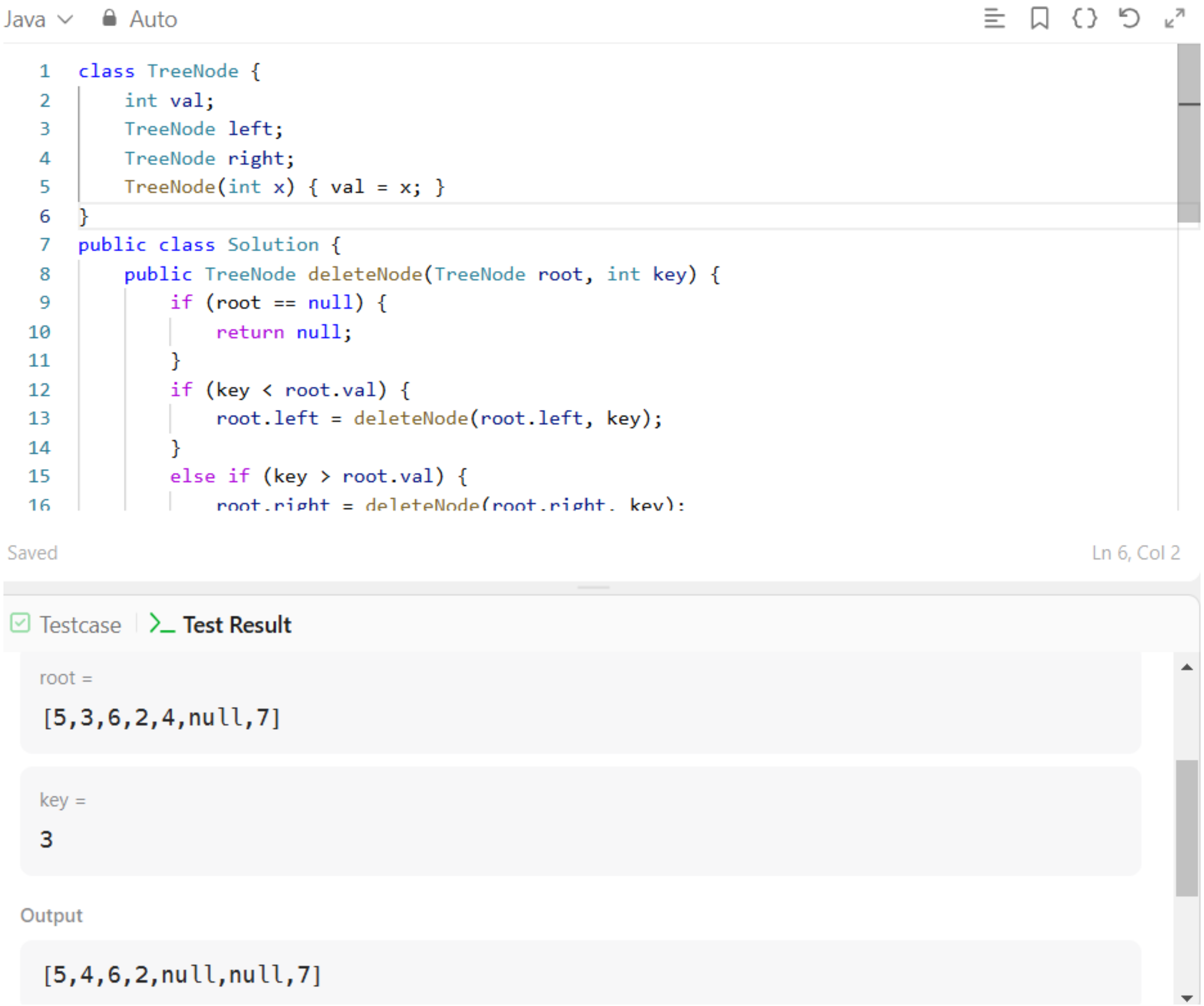
Accepted Runtime: 0 ms

Input

root =

```
[1,2,3,4,5,6]
```

6. Problem statement- Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the root node reference (possibly updated) of the BST. Basically, the deletion can be divided into two stages: Search for a node to remove. If the node is found, delete the node.



```java
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}
public class Solution {
    public TreeNode deleteNode(TreeNode root, int key) {
        if (root == null) {
            return null;
        }
        if (key < root.val) {
            root.left = deleteNode(root.left, key);
        }
        else if (key > root.val) {
            root.right = deleteNode(root.right, key);
```

root =

```
[5,3,6,2,4,null,7]
```

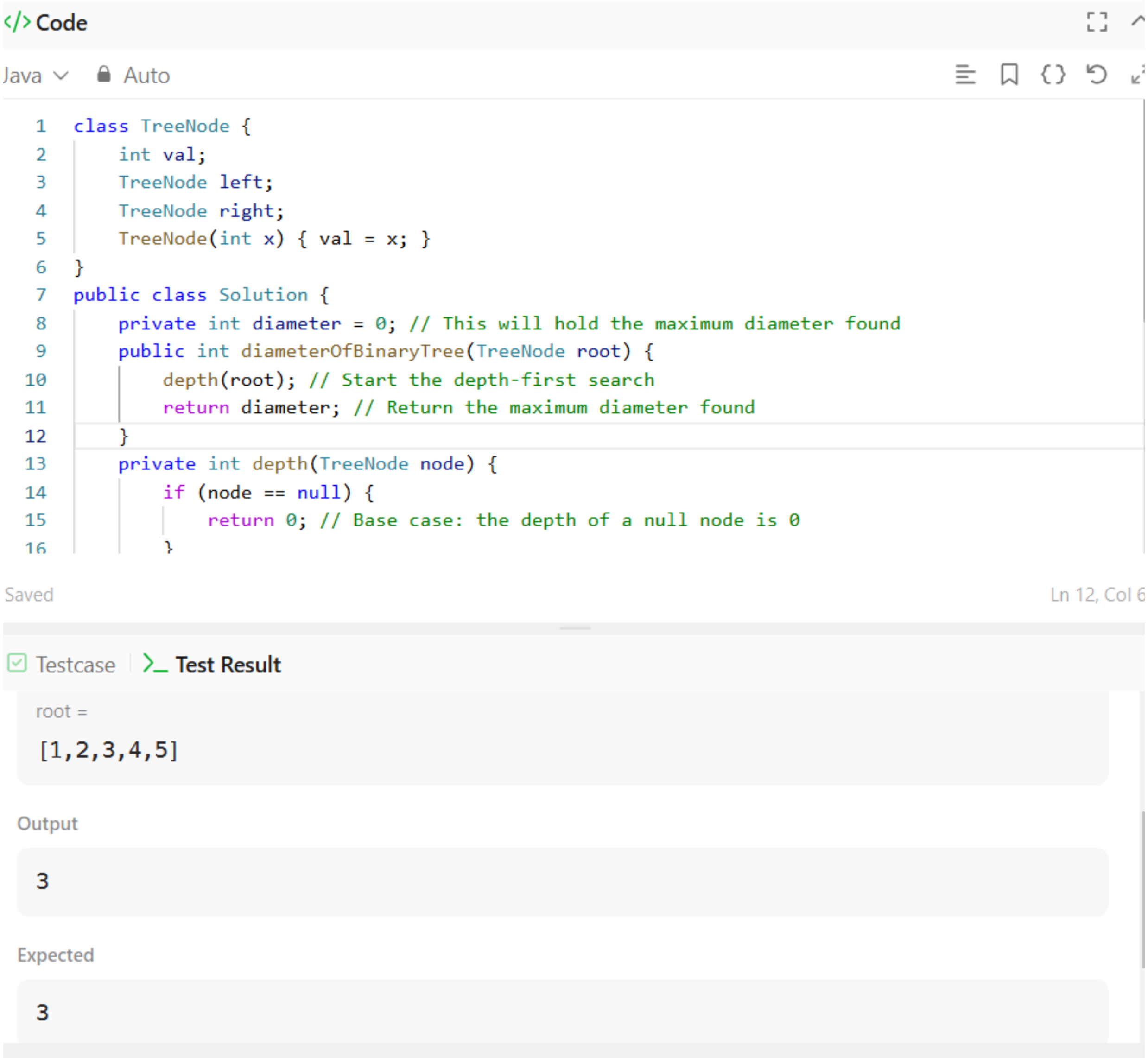
key =

```
3
```

Output

```
[5,4,6,2,null,null,7]
```

7. Problem statement- Given the root of a binary tree, return the length of the diameter of the tree. The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root. The length of a path between two nodes is represented by the number of edges between them.



```java
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}
public class Solution {
    private int diameter = 0; // This will hold the maximum diameter found
    public int diameterOfBinaryTree(TreeNode root) {
        depth(root); // Start the depth-first search
        return diameter; // Return the maximum diameter found
    }
    private int depth(TreeNode node) {
        if (node == null) {
            return 0; // Base case: the depth of a null node is 0
        }
```

root =

```
[1,2,3,4,5]
```

Output

```
3
```

Expected

```
3
```