**Experiment 5**

**Student Name: Ananya kapil UID: 22BCS16219**

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

**Semester: 6th Date of Performance:20/2/25**

**Subject Name: Advanced Programming - 2 Subject Code: 22CSH-351**

**Ques 1:**

**Aim:** Same Tree

**Code:**

class Solution(object):

    def isSameTree(self, p, q):

        if not p and not q:

            return True

        if not p or not q:

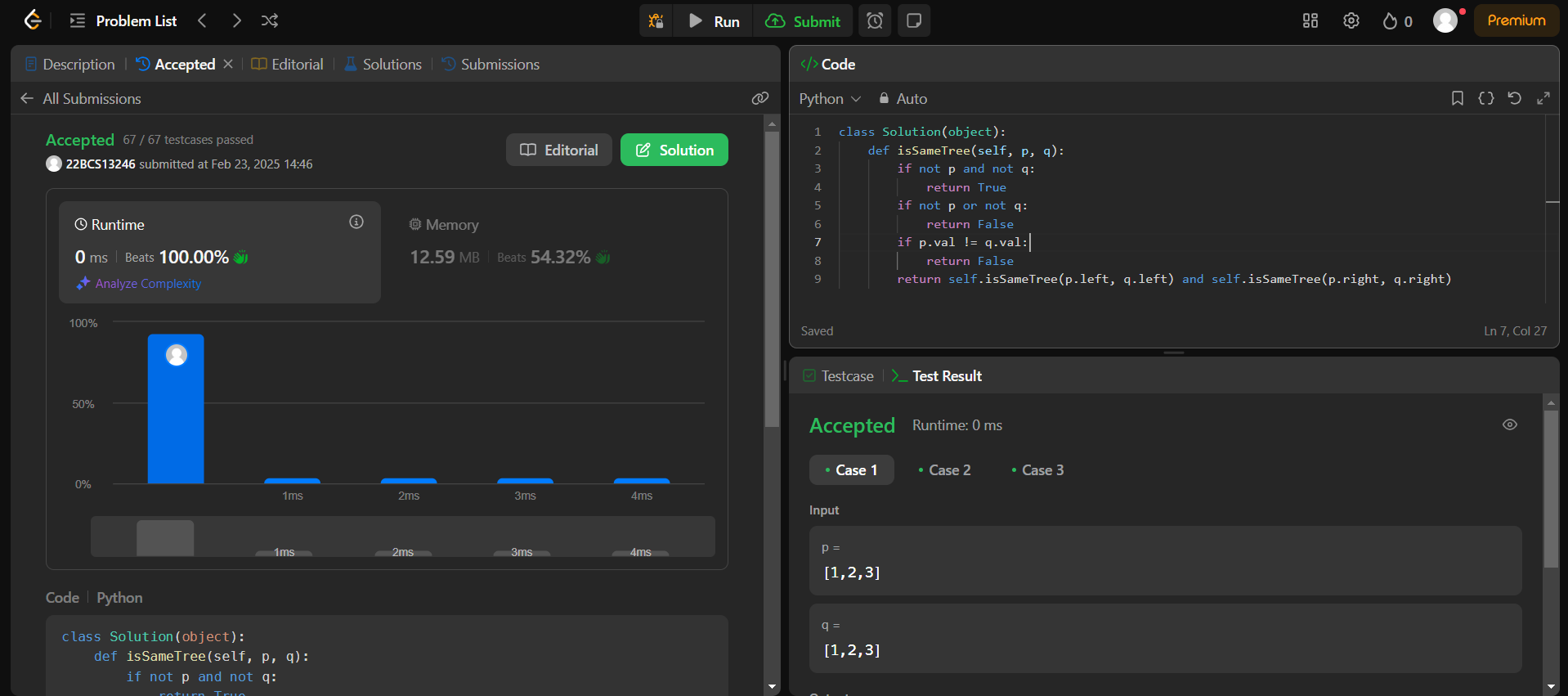
            return False

        if p.val != q.val:

            return False

        return self.isSameTree(p.left, q.left) and self.isSameTree(p.right, q.right)

**Submission Screenshot:**



**Ques 2:**

**Aim:** Symmetric Tree

**Code:**

class TreeNode:

    def \_\_init\_\_(self, val=0, left=None, right=None):

        self.val = val

        self.left = left

        self.right = right

class Solution:

    def isSymmetric(self, root):

        if not root:

            return True

        return self.isMirror(root.left, root.right)

    def isMirror(self, t1, t2):

        if not t1 and not t2:

            return True

        if not t1 or not t2:

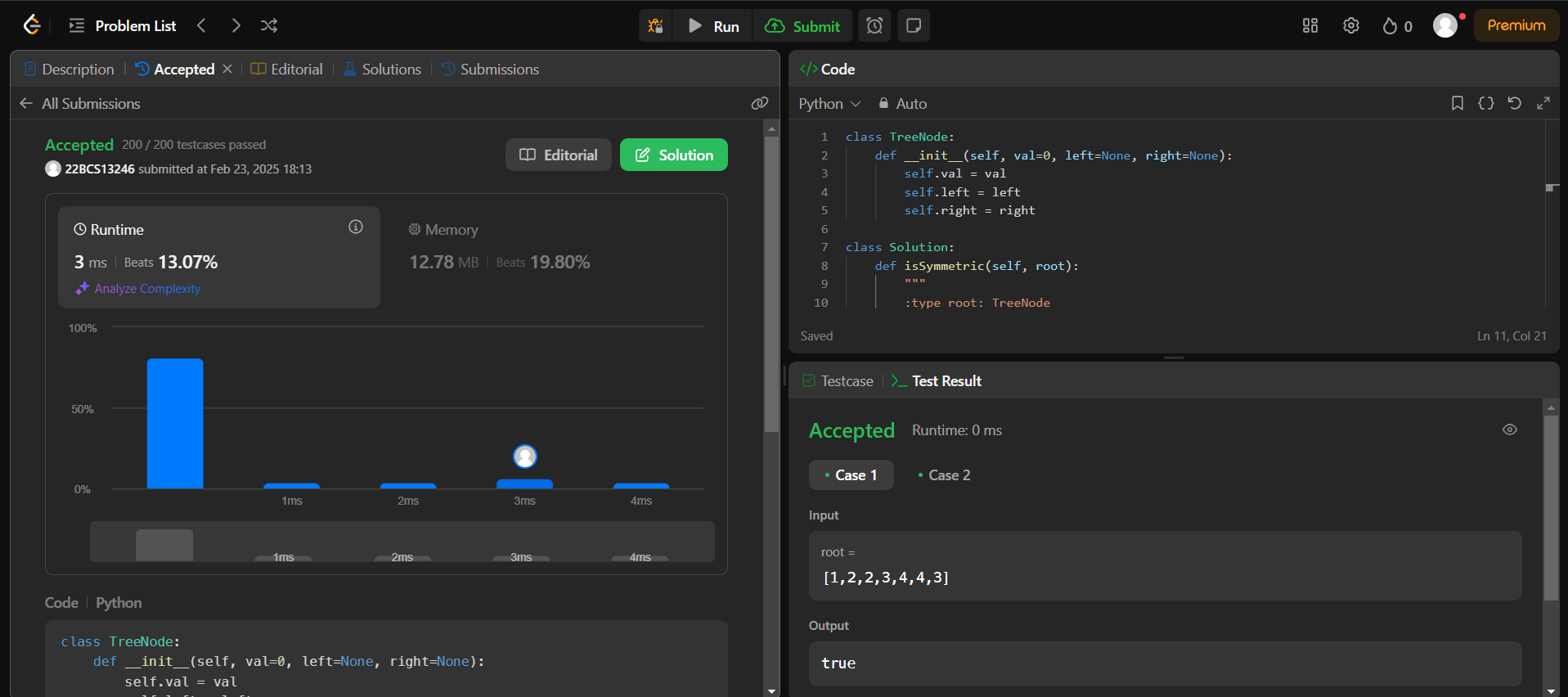
            return False

        if t1.val != t2.val:

            return False

        return self.isMirror(t1.left, t2.right) and self.isMirror(t1.right, t2.left)

**Submission Screenshot:**

****

**Ques 3:**

**Aim:** Balanced Binary Tree

**Code:**

class TreeNode:

    def \_\_init\_\_(self, val=0, left=None, right=None):

        self.val = val

        self.left = left

        self.right = right

class Solution:

    def isBalanced(self, root):

        def height(node):

            if not node:

                return 0

            left, right = height(node.left), height(node.right)

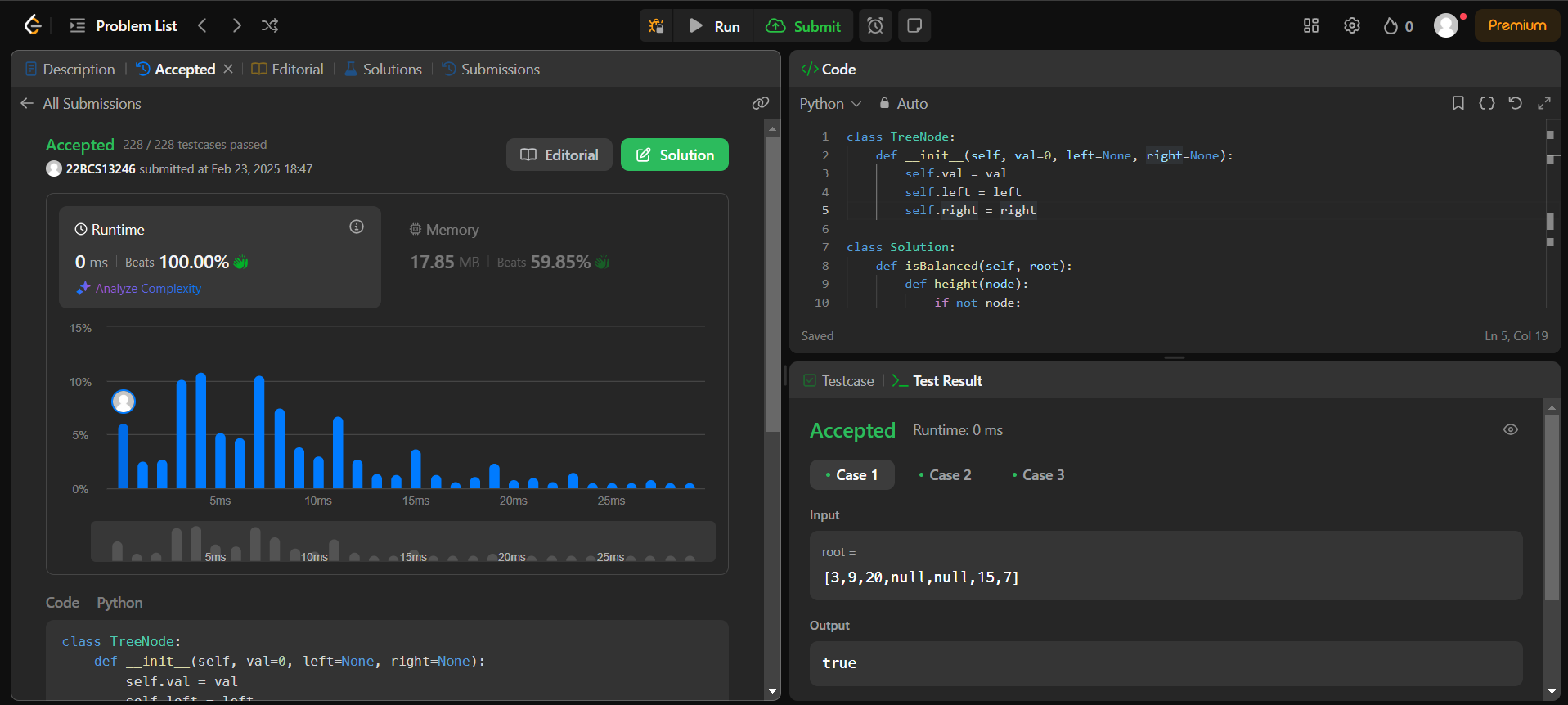
            if abs(left - right) > 1 or left == -1 or right == -1:

                return -1

            return max(left, right) + 1

        return height(root) != -1

**Submission Screenshot:**

****

**Ques 4:**

**Aim:** Path Sum

**Code:**

class TreeNode:

    def \_\_init\_\_(self, val=0, left=None, right=None):

        self.val = val

        self.left = left

        self.right = right

class Solution:

    def hasPathSum(self, root, targetSum):

        if not root:

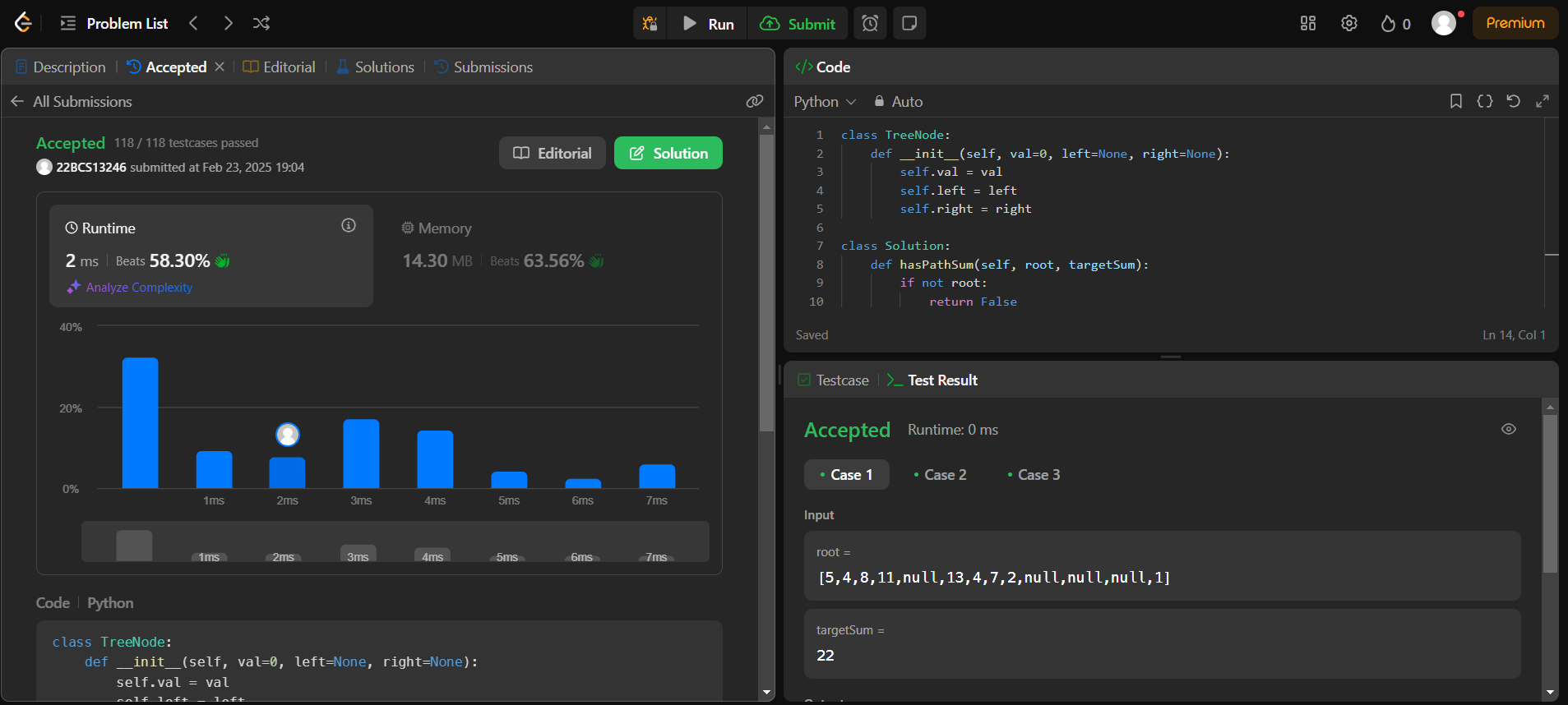
            return False

        if not root.left and not root.right and root.val == targetSum:

            return True

        return self.hasPathSum(root.left, targetSum - root.val) or self.hasPathSum(root.right, targetSum - root.val)

**Submission Screenshot:**

****

**Ques 5:**

**Aim:** Count Complete Tree Nodes

**Code:**

class TreeNode:

    def \_\_init\_\_(self, val=0, left=None, right=None):

        self.val = val

        self.left = left

        self.right = right

class Solution:

    def countNodes(self, root):

        if not root:

            return 0

        lh, rh = self.getHeight(root.left), self.getHeight(root.right)

        if lh == rh:

            return (1 << lh) + self.countNodes(root.right)

        else:

            return (1 << rh) + self.countNodes(root.left)

    def getHeight(self, node):

        h = 0

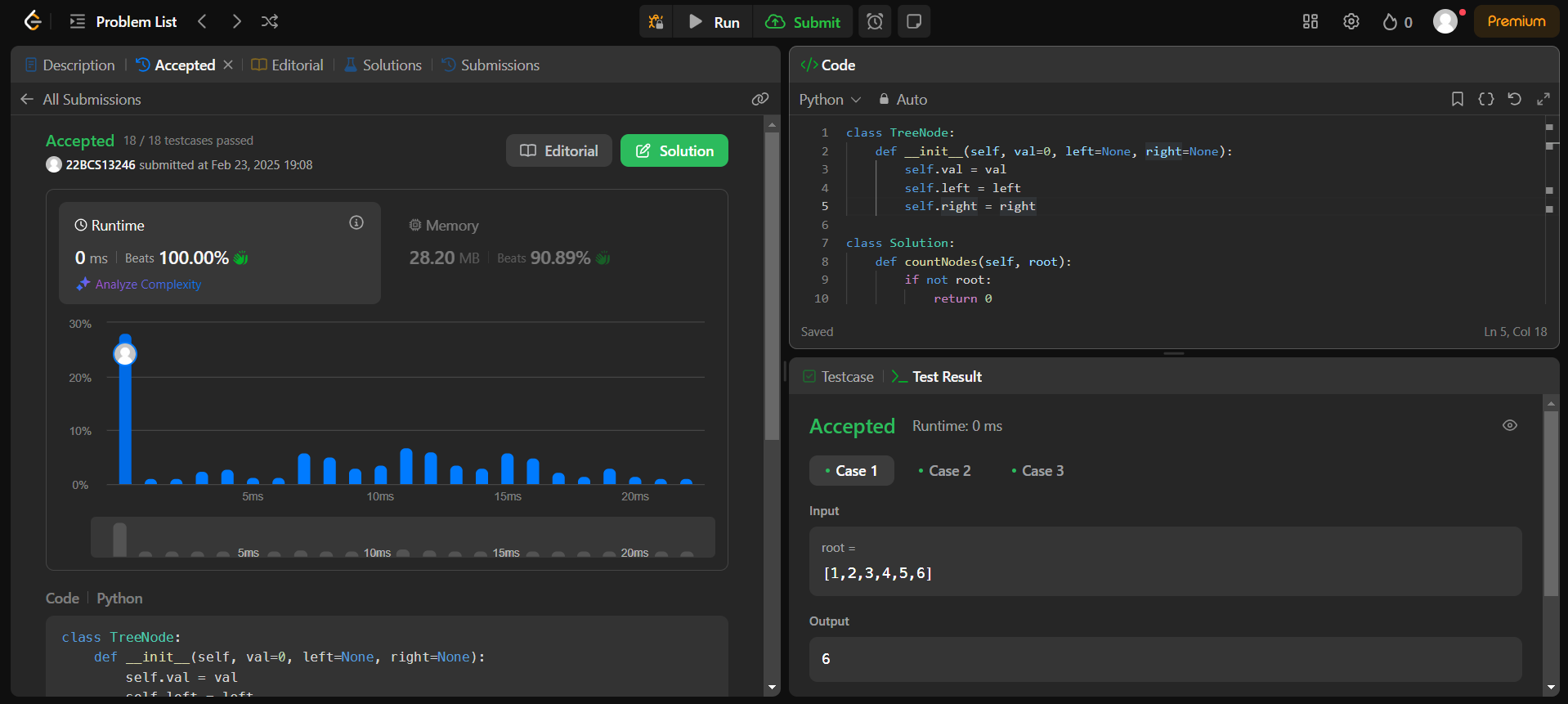
while node:

            h += 1

            node = node.left

        return h

**Submission Screenshot:**

****

**Ques 6:**

**Aim:** Delete node in a BST

**Code:**

class TreeNode:

    def \_\_init\_\_(self, val=0, left=None, right=None):

        self.val = val

        self.left = left

        self.right = right

class Solution:

    def deleteNode(self, root, key):

        if not root: return None

        if key < root.val: root.left = self.deleteNode(root.left, key)

        elif key > root.val: root.right = self.deleteNode(root.right, key)

        else:

            if not root.left: return root.right

            if not root.right: return root.left

            minNode = self.getMin(root.right)

            root.val, root.right = minNode.val, self.deleteNode(root.right, minNode.val)

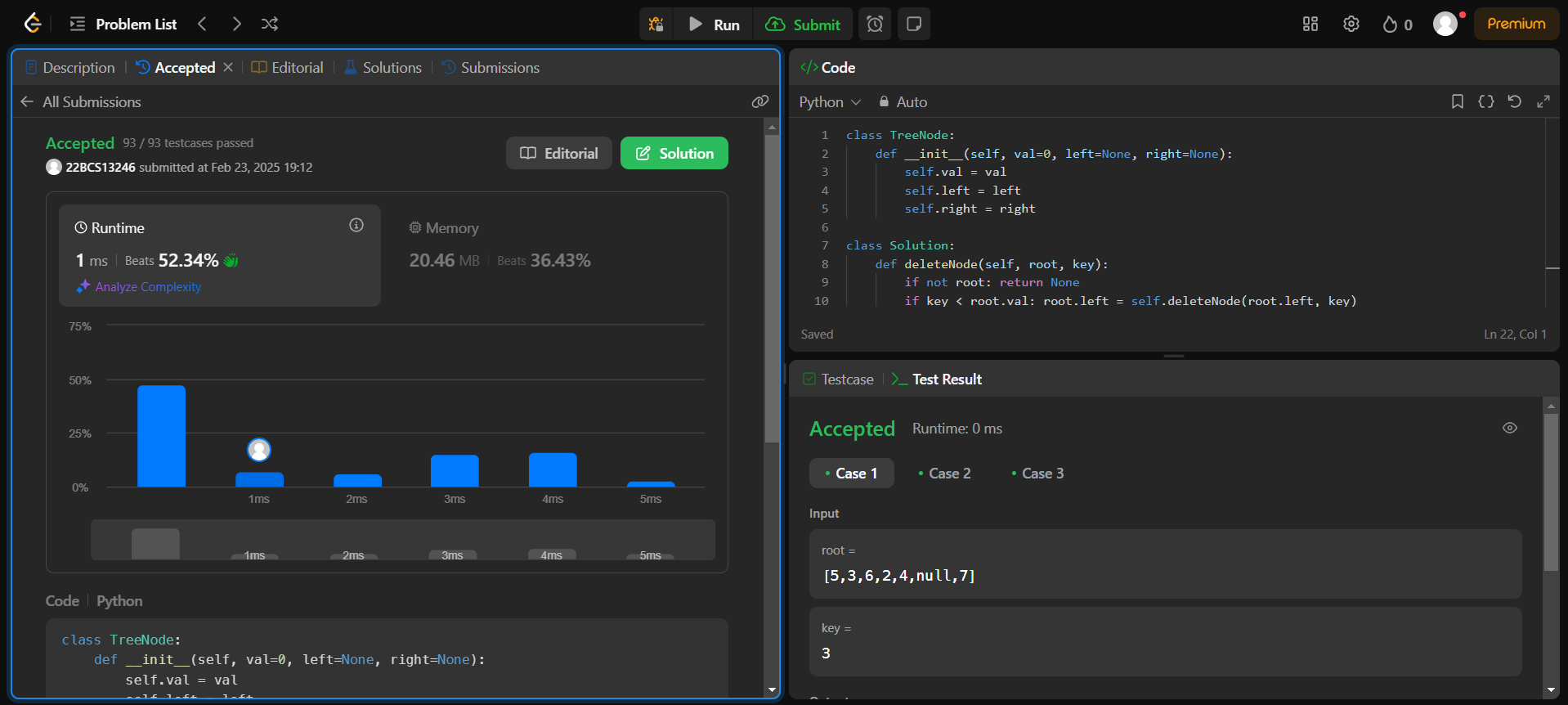
        return root

    def getMin(self, node):

        while node.left: node = node.left

        return node

**Submission Screenshot:**

****

**Ques 7:**

**Aim:** Diameter of Binary Tree

**Code:**

class TreeNode:

    def \_\_init\_\_(self, val=0, left=None, right=None):

        self.val = val

        self.left = left

        self.right = right

class Solution:

    def diameterOfBinaryTree(self, root):

        self.diameter = 0

        def depth(node):

            if not node: return 0

            left, right = depth(node.left), depth(node.right)

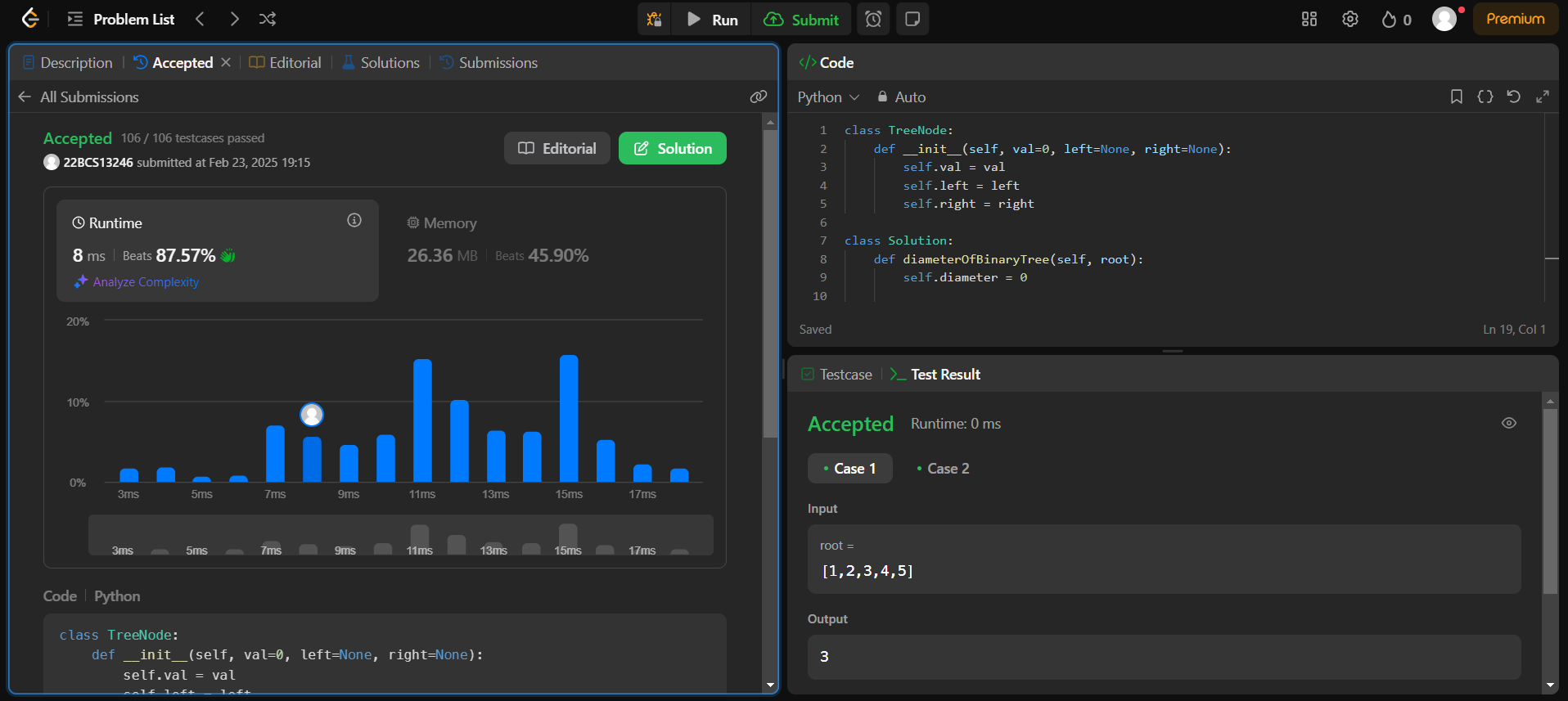
            self.diameter = max(self.diameter, left + right)

            return 1 + max(left, right)

        depth(root)

        return self.diameter

**Submission Screenshot:**

****