**Assignment- 9**

**Student Name: Avin Kumar UID: 22BCS10600 Section/Group: IOT-614 B Semester: 6th**

**Subject Name: Advanced Programming Subject Code:** **22CSP-351**

**Lab-2**

[**200. Number of Islands**](https://leetcode.com/problems/number-of-islands/):

**Code:**

class Solution {

    public void helper(char[][] grid, int x, int y){

        int m = grid.length;

        int n = grid[0].length;

        Queue<int[]> queue = new LinkedList<>();

        queue.offer(new int[]{x,y});

        grid[x][y] = '0';

        int[] dx = {0, 1, 0, -1};

        int[] dy = {1, 0, -1, 0};

        while(!queue.isEmpty()){

            int[] pair = queue.poll();

            for(int d = 0; d<4; d++){

                int x\_ = pair[0]+dx[d];

                int y\_ = pair[1]+dy[d];

                if(x\_ < 0 || y\_ < 0 || x\_ >= m || y\_ >= n || grid[x\_][y\_] == '0') // line 2

                    continue;

                queue.offer(new int[]{x\_,y\_});

                grid[x\_][y\_] = '0';

            }

        }

    }

    public int numIslands(char[][] grid) {

        int m = grid.length;

        int n = grid[0].length;

        int count = 0 ;

        for(int x = 0; x < m; x++){

            for(int y = 0; y < n; y++){

                if(grid[x][y] == '1'){

                    helper(grid,x,y);

                    count++;

                }

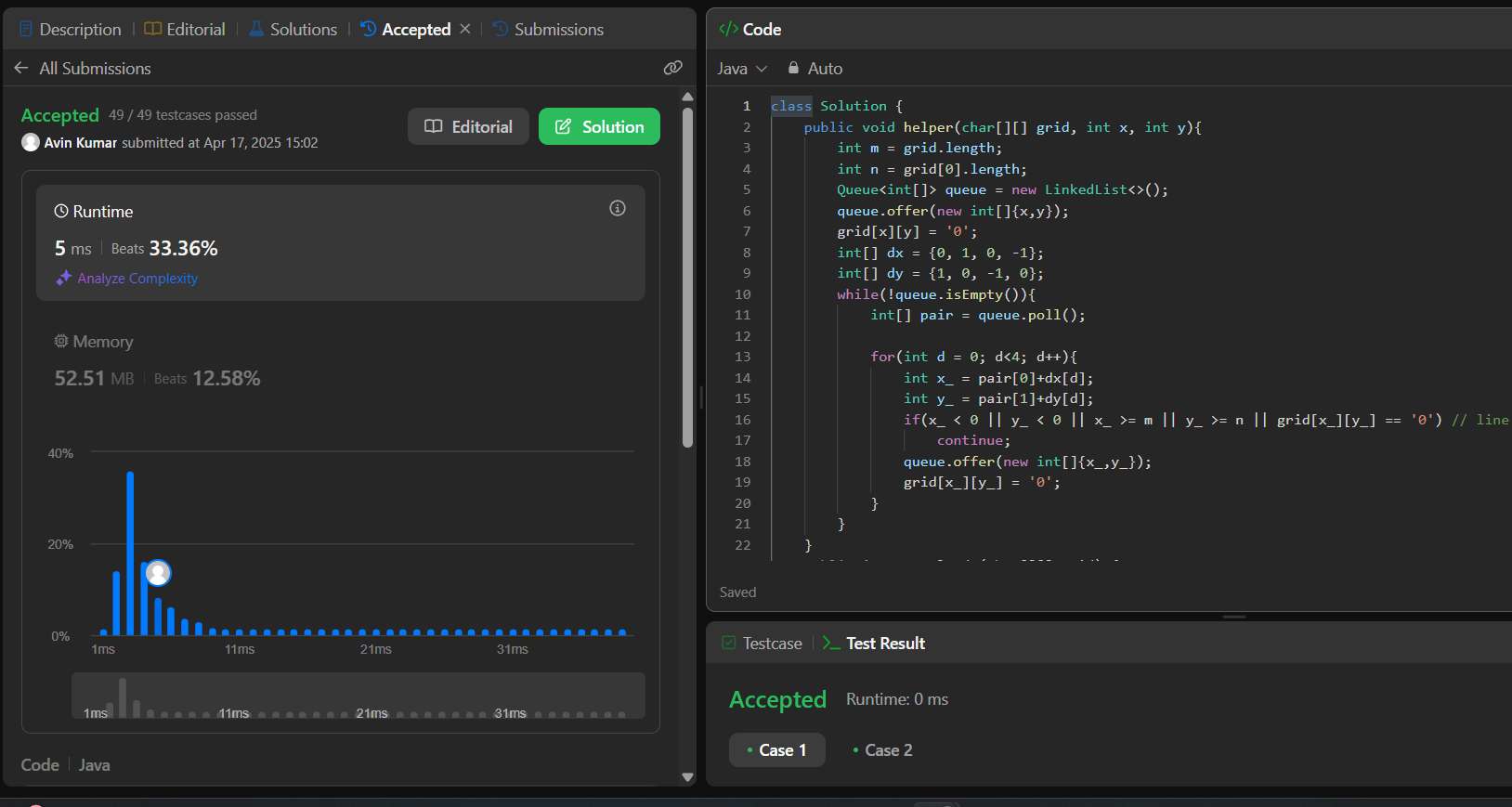
            }

        }

        return count;

    }

}



[**127. Word Ladder**](https://leetcode.com/problems/word-ladder/):

**Code:**

class Solution {

    public int ladderLength(String beginWord, String endWord, List<String> L) {

        Queue<Tuple> q = new LinkedList<>();

        Set<String> set = new HashSet<>(L);

        set.remove(beginWord);

        q.add(new Tuple(beginWord, 1));

        while (!q.isEmpty()) {

            Tuple tuple = q.poll();

            String node = tuple.node;

            int x = tuple.x;

            if(node.equals(endWord)) return x;

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

                for (char ch = 'a'; ch <= 'z'; ch++) {

                    char[] temp = node.toCharArray();

                    temp[i] = ch;

                    String newString = new String(temp);

                    if (set.contains(newString)) {

                        q.add(new Tuple(newString, x + 1));

                        set.remove(newString);

                    }

                }

            }

        }

        return 0;

    }

}

class Tuple{

    String node;

    int x;

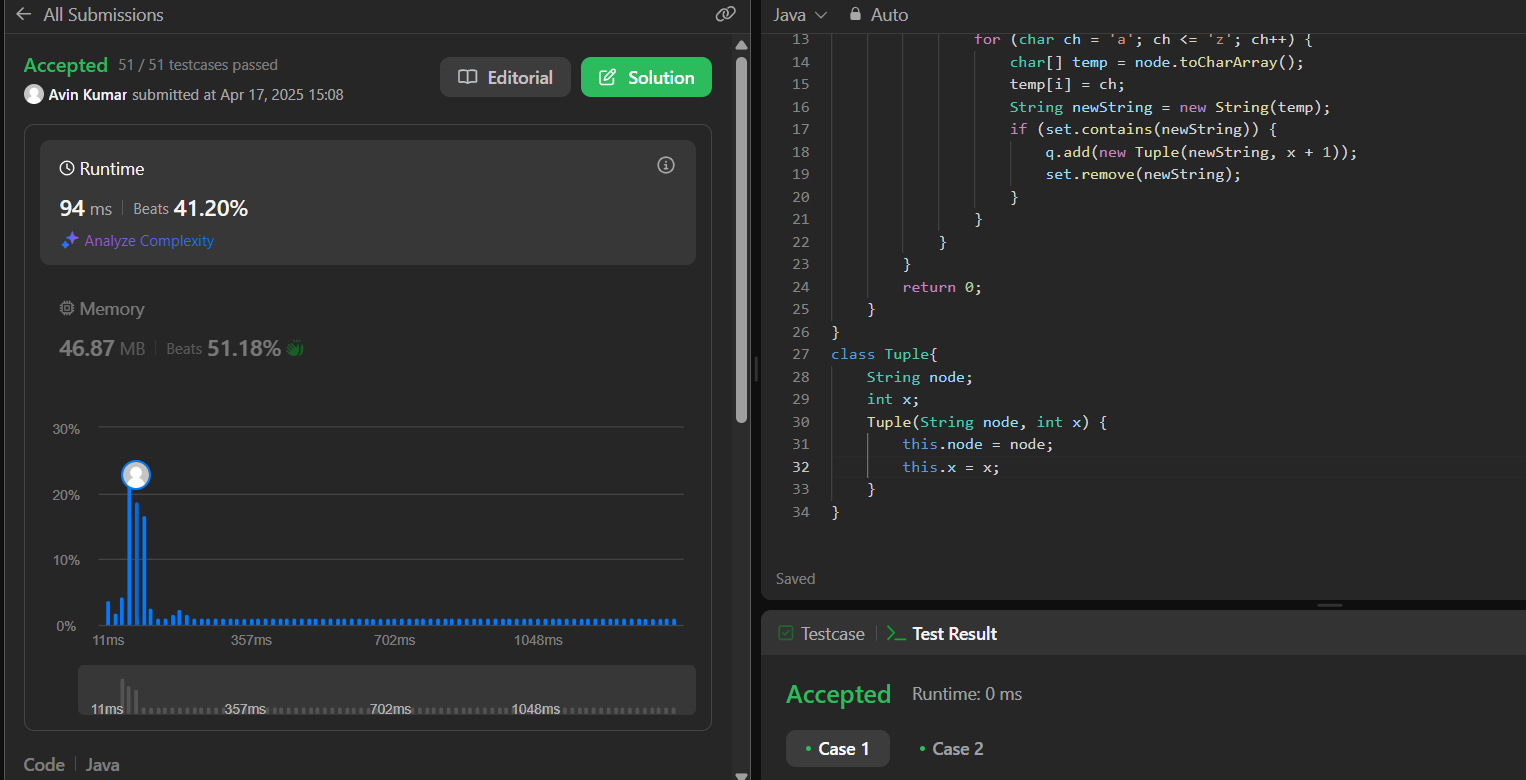
    Tuple(String node, int x) {

        this.node = node;

        this.x = x;

    }

}



[**130. Surrounded Regions**](https://leetcode.com/problems/surrounded-regions/):

**Code:**

class Solution {

    public void solve(char[][] board) {

        if (board == null || board.length == 0) return;

        int n = board.length;

        int m = board[0].length;

        int[][] vis = new int[n][m];

        int[] delRow = {-1, 0, 1, 0};

        int[] delCol = {0, 1, 0, -1};

        for (int j = 0; j < m; j++) {

            if (board[0][j] == 'O' && vis[0][j] == 0) {

                dfs(0, j, board, vis, delRow, delCol);

            }

            if (board[n - 1][j] == 'O' && vis[n - 1][j] == 0) {

                dfs(n - 1, j, board, vis, delRow, delCol);

            }

        }

        for (int i = 0; i < n; i++) {

            if (board[i][0] == 'O' && vis[i][0] == 0) {

                dfs(i, 0, board, vis, delRow, delCol);

            }

            if (board[i][m - 1] == 'O' && vis[i][m - 1] == 0) {

                dfs(i, m - 1, board, vis, delRow, delCol);

            }

        }

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < m; j++) {

                if (board[i][j] == 'O' && vis[i][j] == 0) {

                    board[i][j] = 'X';

                }

            }

        }

    }

    private void dfs(int row, int col, char[][] board, int[][] vis, int[] delRow, int[] delCol) {

        vis[row][col] = 1;

        int n = board.length;

        int m = board[0].length;

        for (int i = 0; i < 4; i++) {

            int nrow = row + delRow[i];

            int ncol = col + delCol[i];

            if (nrow >= 0 && nrow < n && ncol >= 0 && ncol < m

                && board[nrow][ncol] == 'O' && vis[nrow][ncol] == 0) {

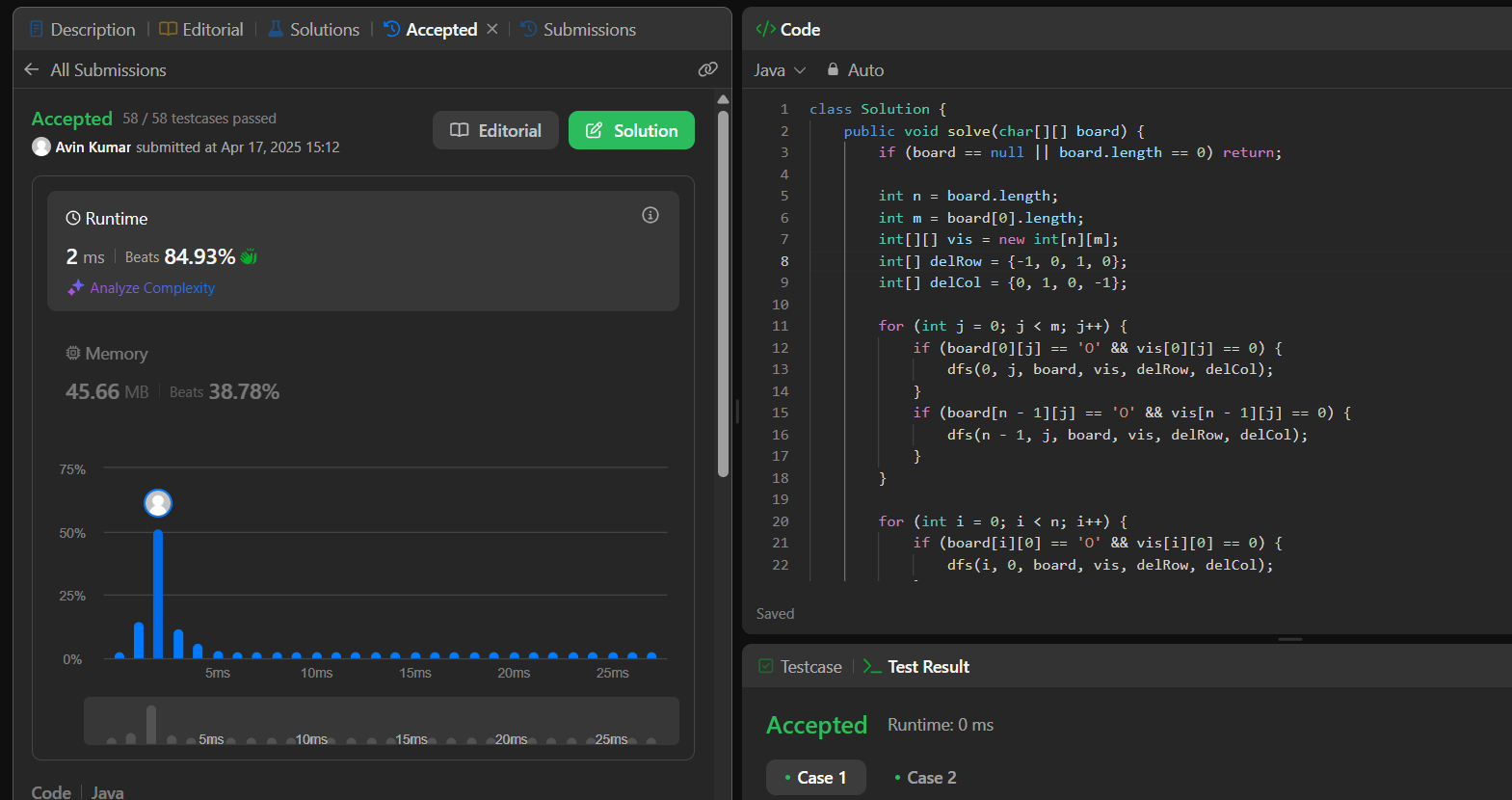
                dfs(nrow, ncol, board, vis, delRow, delCol);

            }

        }

    }

}



[**124. Binary Tree Maximum Path Sum**](https://leetcode.com/problems/binary-tree-maximum-path-sum/):

**Code:**

/\*\*

 \* Definition for a binary tree node.

 \* public 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 TreeNode {

    int val;

    TreeNode left, right;

    TreeNode(int val) { this.val = val; }

}

class Solution {

    private int ans = Integer.MIN\_VALUE;

    public int maxPathSum(TreeNode root) {

        helper(root);

        return ans;

    }

    private int helper(TreeNode root) {

        if (root == null) return 0;

        int left = Math.max(0, helper(root.left));

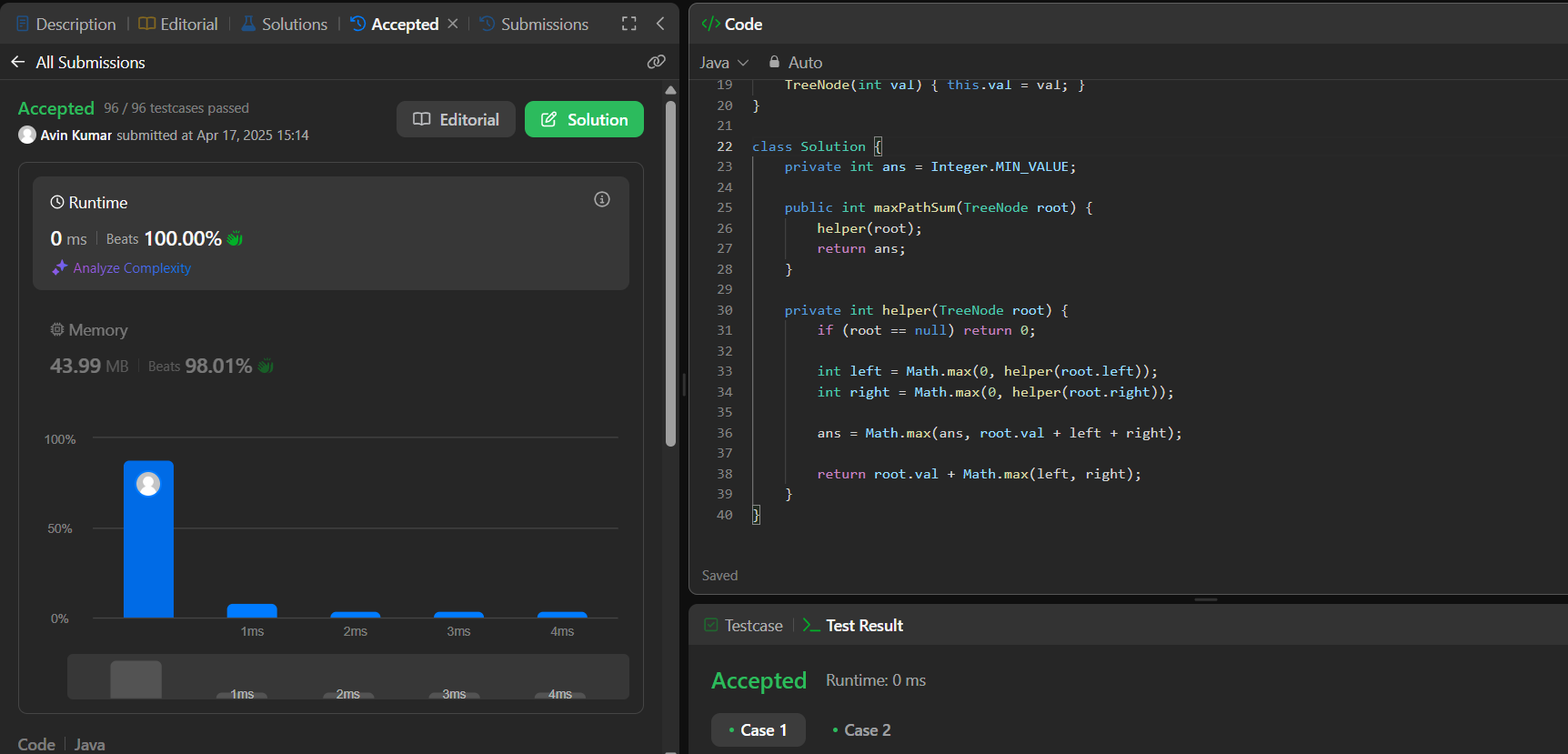
        int right = Math.max(0, helper(root.right));

        ans = Math.max(ans, root.val + left + right);

        return root.val + Math.max(left, right);

    }

}



[**547. Number of Provinces**](https://leetcode.com/problems/number-of-provinces/):

**Code:**

class Solution {

public void dfs(List<Integer>[] adjList, boolean[] visited, int curr) {

visited[curr] = true;

for (int neighbor : adjList[curr]) {

if (!visited[neighbor]) {

dfs(adjList, visited, neighbor);

}

}

}

public int findCircleNum(int[][] isConnected) {

int V = isConnected.length;

List<Integer>[] adjList = new ArrayList[V];

for (int i = 0; i < V; i++) {

adjList[i] = new ArrayList<>();

}

for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) {

if (isConnected[i][j] == 1 && i != j) {

adjList[i].add(j);

adjList[j].add(i); // Since it's undirected

}

}

}

boolean[] visited = new boolean[V];

int provinces = 0;

for (int i = 0; i < V; i++) {

if (!visited[i]) {

provinces++;

dfs(adjList, visited, i);

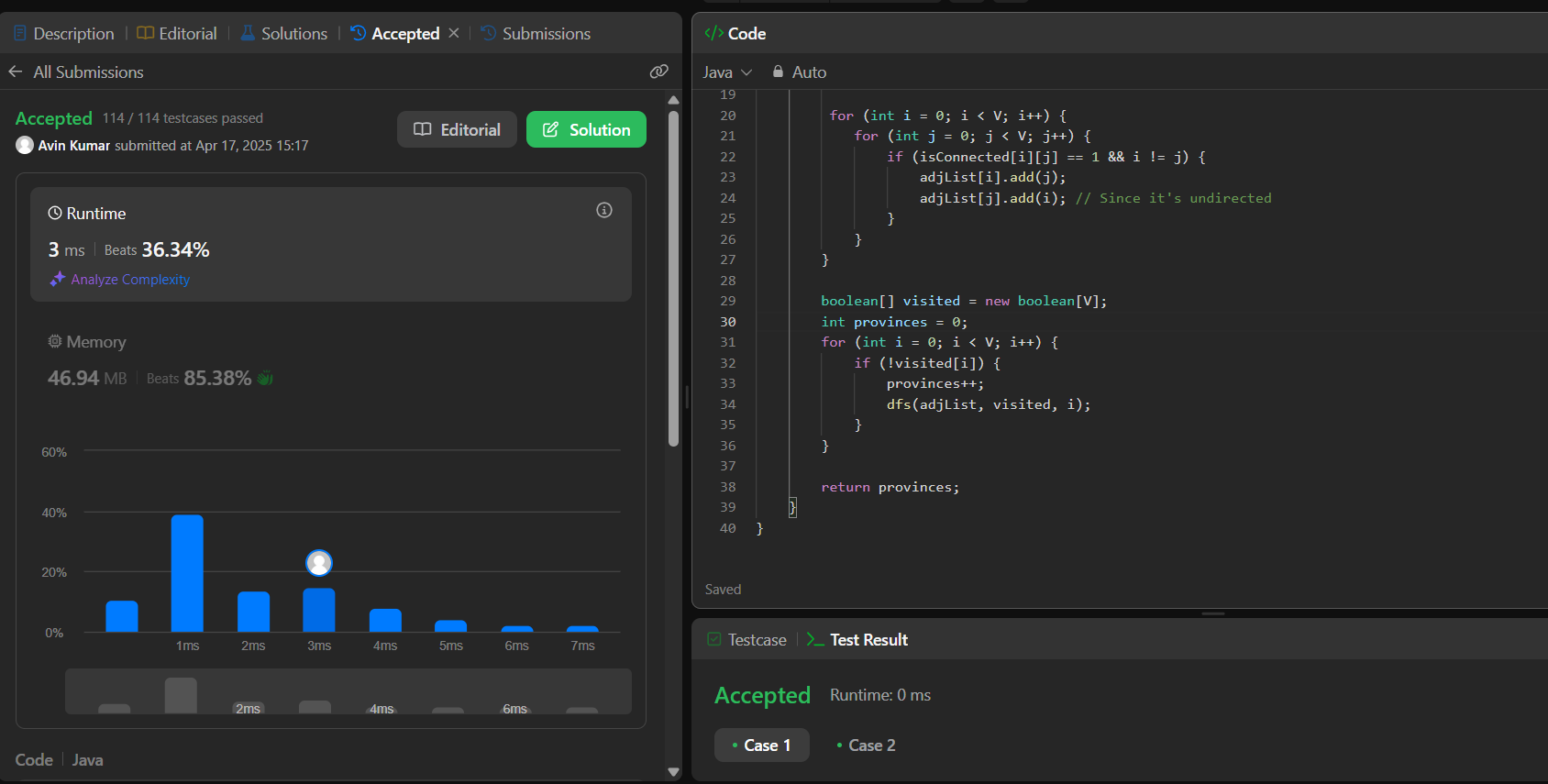
}

}

return provinces;

}

}



[**236. Lowest Common Ancestor of a Binary Tree**](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/):

**Code:**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

class Solution {

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

if(root==null||root==p||root==q){

return root;

}

TreeNode leftlca=lowestCommonAncestor(root.left,p,q);

TreeNode rightlca=lowestCommonAncestor(root.right,p,q);

if(rightlca==null){

return leftlca;

}

if(leftlca==null){

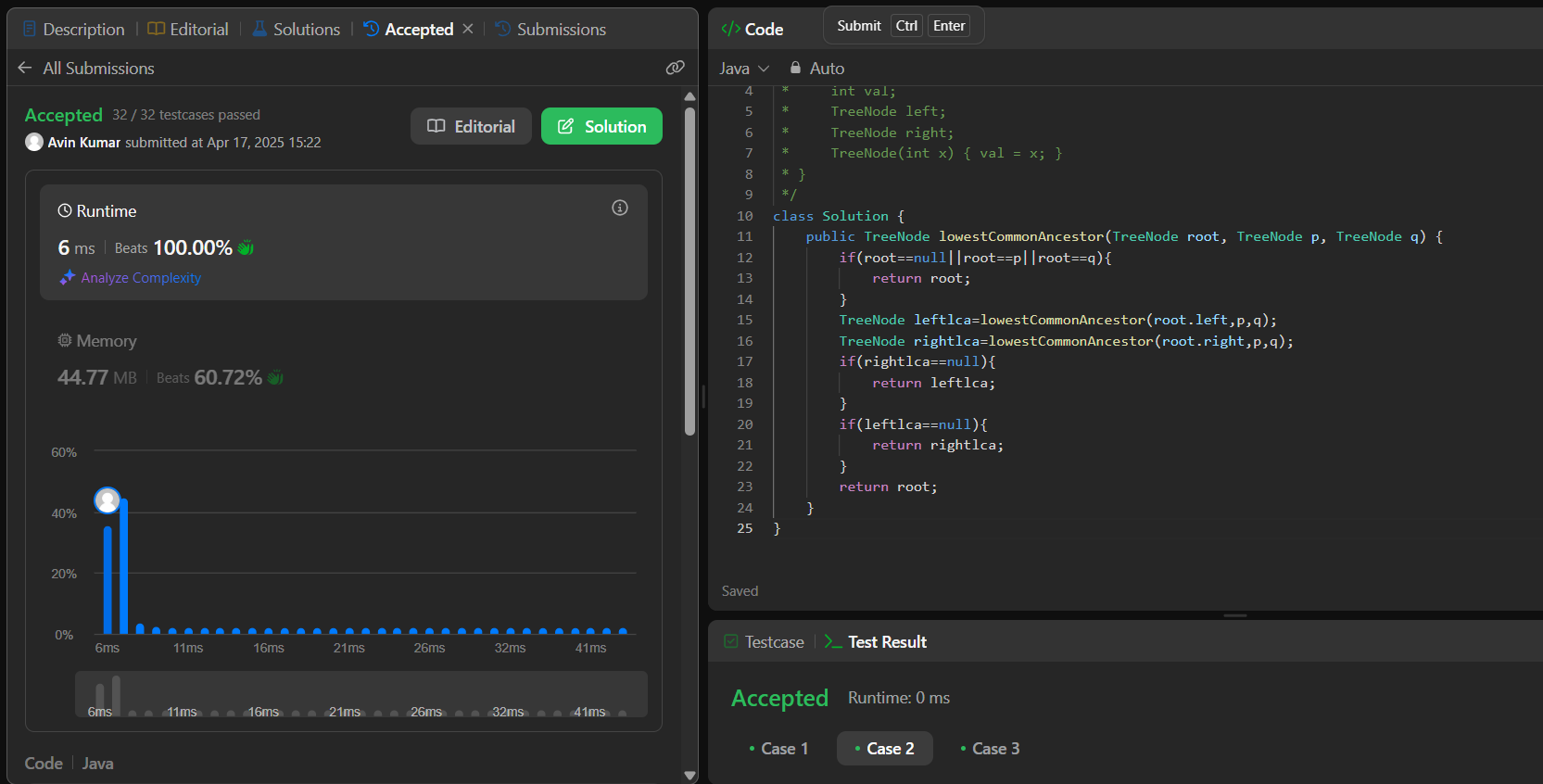
return rightlca;

}

return root;

}

}



[**207. Course Schedule**](https://leetcode.com/problems/course-schedule/):

**Code:**

class Solution {

private boolean dfs(int i,List<List<Integer>> adj,HashSet<Integer> set,boolean[] visited)

{

set.add(i);

visited[i] = true;

for(int num:adj.get(i))

{

if(!visited[num] && !dfs(num,adj,set,visited)) return false;

else if(set.contains(num)) return false;

}

set.remove(i);

return true;

}

public boolean canFinish(int numCourses, int[][] prerequisites) {

List<List<Integer>> adj = new ArrayList<>();

for(int i=0;i<numCourses;i++)

adj.add(new ArrayList<>());

for(int[] edge:prerequisites)

adj.get(edge[1]).add(edge[0]);

boolean acyclic = true;

boolean[] visited = new boolean[numCourses];

for(int i=0;i<numCourses;i++)

{

if(!visited[i])

{

HashSet<Integer> set = new HashSet<>();

acyclic = dfs(i,adj,set,visited);

if(!acyclic) return false;

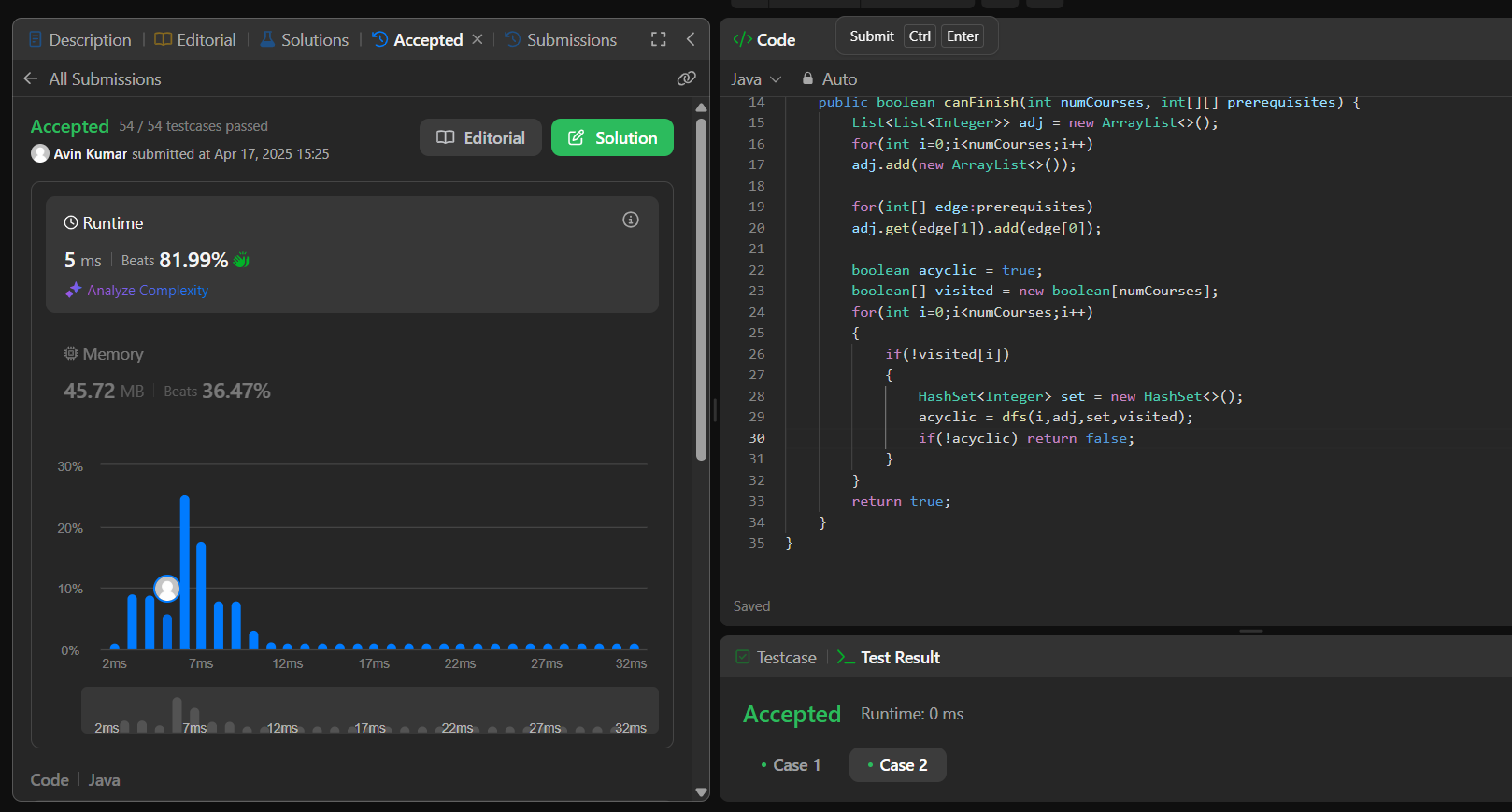
}

}

return true;

}

}



[**329. Longest Increasing Path in a Matrix**](https://leetcode.com/problems/longest-increasing-path-in-a-matrix/)**:**

**Code:**

class Solution {

    private int dfs(int r,int c,int[][] dp,int[][] matrix,int m,int n)

    {

        if(r<0 || c<0 || r>=m || c>=n) return 0;

        if(dp[r][c]!=-1) return dp[r][c];

        int up=0,down = 0,left =0,right = 0;

        if(r>0 && matrix[r-1][c]> matrix[r][c]) up = 1 + dfs(r-1,c,dp,matrix,m,n);

        if(r<m-1 && matrix[r+1][c]> matrix[r][c]) down = 1 + dfs(r+1,c,dp,matrix,m,n);

        if(c>0 && matrix[r][c-1]> matrix[r][c]) left = 1 + dfs(r,c-1,dp,matrix,m,n);

        if(c<n-1 && matrix[r][c+1]> matrix[r][c]) right = 1 + dfs(r,c+1,dp,matrix,m,n);

        return  dp[r][c] = Math.max(Math.max(up,down),Math.max(left,right));

    }

    public int longestIncreasingPath(int[][] matrix) {

        int m = matrix.length;

        int n = matrix[0].length;

        int ans = Integer.MIN\_VALUE;

        int[][] dp = new int[m][n];

        for(int[] row:dp) Arrays.fill(row,-1);

        for(int i=0;i<=m;i++)

        {

            for(int j=0;j<=n;j++)

            {

                int val = 1 + dfs(i,j,dp,matrix,m,n);

                ans = Math.max(ans,val);

            }

        }

        return ans;

    }

}

