**Name:** Aditya Singh

**UID:** 22BCS15063

**Section:** FL\_IOT\_601 - A

**Assignment – 9 Solutions:-**

1. **Number of Islands:-**

class Solution {

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

int row = grid.length;

int col = grid[0].length;

int islands = 0;

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

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

if(grid[i][j] == '1'){

islands++;

dfs(i, j, grid);

}

}

}

return islands;

}

public void dfs(int row, int col, char[][] grid){

int newRow = grid.length;

int newCol = grid[0].length;

int[][] directions = new int[][]{{0,1}, {1,0}, {0,-1}, {-1,0}};

if(row<0 || row>=newRow || col<0 || col>=newCol || grid[row][col] == '0'){

return;

}

grid[row][col] = '0';

for(int[] dir: directions){

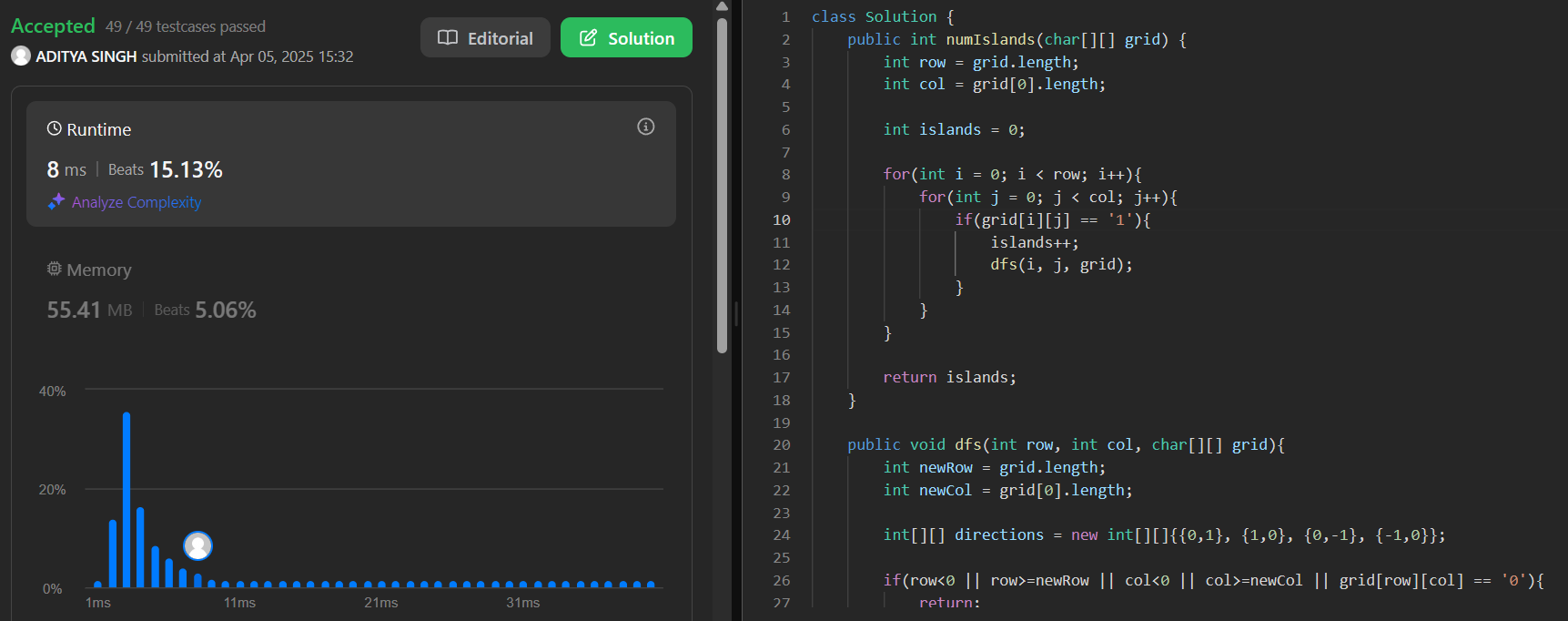
dfs(row+dir[0], col+dir[1], grid);

}

}

}

Result:-



1. **Word Ladder:-**

class Solution {

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

int l = beginWord.length();

Map<String, List<String>> allComboDict = new HashMap<>();

wordList.forEach(word -> {

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

String newWord = word.substring(0,i) + '\*' + word.substring(i+1, l);

List<String> transformations = allComboDict.getOrDefault(newWord, new ArrayList<>());

transformations.add(word);

allComboDict.put(newWord, transformations);

}

});

Queue<Pair<String, Integer>> Q = new LinkedList<>();

Q.add(new Pair(beginWord, 1));

Map<String, Boolean> visited = new HashMap<>();

visited.put(beginWord, true);

while(!Q.isEmpty()){

Pair<String, Integer> node = Q.remove();

String word = node.getKey();

int level = node.getValue();

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

String newWord = word.substring(0, i) + '\*' + word.substring(i + 1, l);

for(String adjacentWord : allComboDict.getOrDefault(newWord, new ArrayList<>())){

if(adjacentWord.equals(endWord)){

return level + 1;

}

if(!visited.containsKey(adjacentWord)){

visited.put(adjacentWord, true);

Q.add(new Pair(adjacentWord, level + 1));

}

}

}

}

return 0;

}

}

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Surrounded Regions:-**

class Solution {

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

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

int m = board.length, n = board[0].length;

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

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

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

dfs(board, i, j);

}

}

}

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

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

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

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

}

else if (board[i][j] == 'T'){

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

}

}

}

}

public void dfs(char[][] board, int i, int j){

if(i < 0 || i >= board.length || j < 0 || j >= board[i].length || board[i][j] != 'O'){

return;

}

board[i][j] = 'T';

dfs(board, i + 1, j);

dfs(board, i - 1, j);

dfs(board, i, j + 1);

dfs(board, i, j - 1);

}

}

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Binary Tree Maximum Path Sum:-**

class Solution {

int maxSum = Integer.MIN\_VALUE;

public int maxGain(TreeNode node){

if(node == null){

return 0;

}

int leftGain = Math.max(maxGain(node.left), 0);

int rightGain = Math.max(maxGain(node.right), 0);

int priceNewPath = node.val + leftGain + rightGain;

maxSum = Math.max(maxSum, priceNewPath);

return node.val + Math.max(leftGain, rightGain);

}

public int maxPathSum(TreeNode root) {

maxGain(root);

return maxSum;

}

}

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Friend Circles:-**

class Solution {

public:

void dfs(vector<vector<int>>& isConnected, vector<int>& visited, int i){

visited[i] = 1;

for (int j = 0; j < isConnected.size(); ++j) {

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

dfs(isConnected, visited, j);

}

}

}

int findCircleNum(vector<vector<int>>& isConnected) {

int n = isConnected.size();

vector<int> visited(n, 0);

int count = 0;

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

if (!visited[i]) {

dfs(isConnected, visited, i);

count++;

}

}

return count;

}

};

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Lowest Common Ancestor of a Binary Tree:-**

/\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* }; \*/

class Solution {

public:

TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

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

return root;

}

TreeNode\* left = lowestCommonAncestor(root->left, p, q);

TreeNode\* right = lowestCommonAncestor(root->right, p, q);

if (left != nullptr && right != nullptr) {

return root;

}

return left != nullptr ? left : right;

}

};

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Course Schedule:-**

class Solution {

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

HashMap<Integer, List<Integer>> courseGraph = new HashMap<>();

for(int[] pre: prerequisites){

if(courseGraph.containsKey(pre[1])){

courseGraph.get(pre[1]).add(pre[0]);

}

else{

List<Integer> nextCourses = new LinkedList<>();

nextCourses.add(pre[0]);

courseGraph.put(pre[1], nextCourses);

}

}

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

for(int currentCourse = 0; currentCourse < numCourses; currentCourse++){

if(courseSchedule(currentCourse, visited, courseGraph) == false){

return false;

}

}

return true;

}

public boolean courseSchedule(int course, HashSet<Integer> visited, HashMap<Integer, List<Integer>> courseGraph){

if(visited.contains(course)){

return false;

}

if(courseGraph.get(course) == null){

return true;

}

visited.add(course);

for(int pre: courseGraph.get(course)){

if(courseSchedule(pre, visited, courseGraph) == false){

return false;

}

}

visited.remove(course);

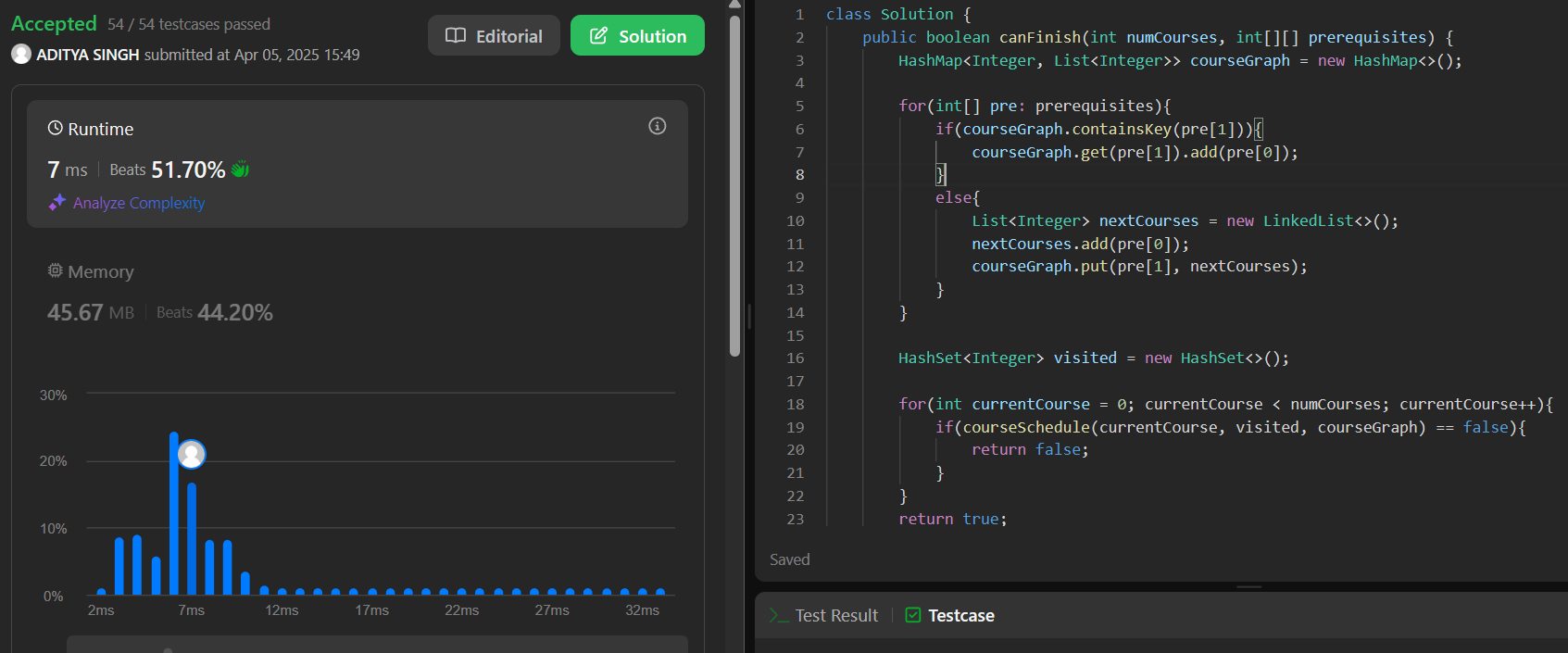
courseGraph.put(course, null);

return true;

}

}

Result:-



1. **Longest Increasing Path in a Matrix:-**

class Solution {

private static int[][] dirs = {{0,1}, {0,-1}, {-1,0}, {1,0}};

private int m, n;

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

if(matrix.length == 0) return 0;

m = matrix.length;

n = matrix[0].length;

int[][] dp = new int[m][n];

int ans = 0;

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

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

ans = Math.max(ans, dfs(matrix, i, j, dp));

}

}

return ans;

}

private int dfs(int[][] matrix, int i, int j, int dp[][]){

if(dp[i][j] != 0) return dp[i][j];

for(int[] d: dirs){

int x = i + d[0], y = j + d[1];

if(x >= 0 && x < m && y >= 0 && y < n && matrix[x][y] > matrix[i][j]){

dp[i][j] = Math.max(dp[i][j], dfs(matrix, x, y, dp));

}

}

return ++dp[i][j];

}

}

Result:-

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Course Schedule II:-**

class Solution {

static int WHITE = 1;

static int GRAY = 2;

static int BLACK = 3;

boolean isPossible;

Map<Integer, Integer> color;

Map<Integer, List<Integer>> adjList;

List<Integer> topologicalOrder;

private void init(int numCourses) {

this.isPossible = true;

this.color = new HashMap<Integer, Integer>();

this.adjList = new HashMap<Integer, List<Integer>>();

this.topologicalOrder = new ArrayList<Integer>();

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

this.color.put(i, WHITE);

}

}

private void dfs(int node) {

if (!this.isPossible) {

return;

}

this.color.put(node, GRAY);

for (Integer neighbor : this.adjList.getOrDefault(node, new ArrayList<Integer>())) {

if (this.color.get(neighbor) == WHITE) {

this.dfs(neighbor);

} else if (this.color.get(neighbor) == GRAY) {

this.isPossible = false;

}

}

this.color.put(node, BLACK);

this.topologicalOrder.add(node);

}

public int[] findOrder(int numCourses, int[][] prerequisites) {

this.init(numCourses);

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

int dest = prerequisites[i][0];

int src = prerequisites[i][1];

List<Integer> lst = adjList.getOrDefault(src, new ArrayList<Integer>());

lst.add(dest);

adjList.put(src, lst);

}

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

if (this.color.get(i) == WHITE) {

this.dfs(i);

}

}

int[] order;

if (this.isPossible) {

order = new int[numCourses];

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

order[i] = this.topologicalOrder.get(numCourses - i - 1);

}

} else {

order = new int[0];

}

return order;

}

}

Result:-

A screenshot of a computer

AI-generated content may be incorrect.