Advanced Programming LAB II

EXPERIMENT - 9

Submitted by,

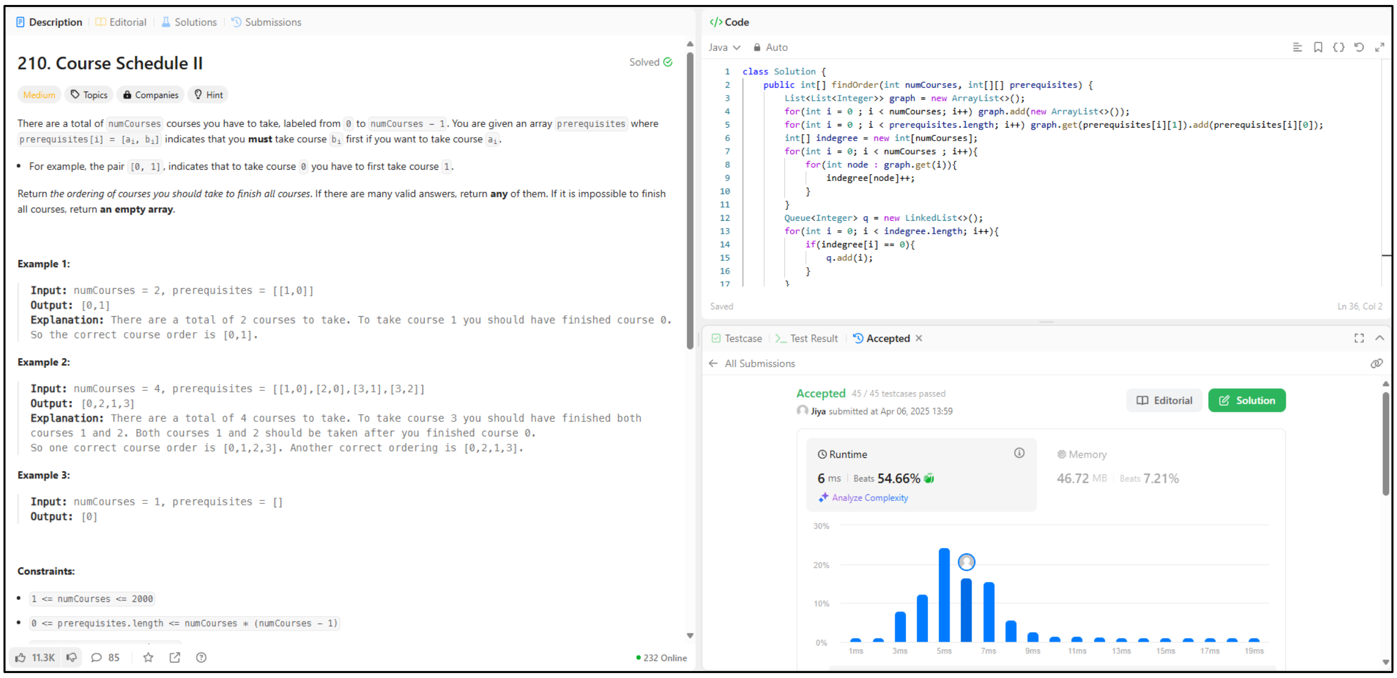
Jiya | 22BCS14856

22BCS_FL_IOT-601 (A)

210. Course Schedule II

https://leetcode.com/problems/course-schedule-ii/description/

```java
class Solution {
    public int[] findOrder(int numCourses, int[][] prerequisites) {
        List<List<Integer>> graph = new ArrayList<>();
        for(int i = 0 ; i < numCourses; i++) graph.add(new ArrayList<>());
        for(int i = 0 ; i < prerequisites.length; i++)
graph.get(prerequisites[i][1]).add(prerequisites[i][0]);
        int[] indegree = new int[numCourses];
        for(int i = 0; i < numCourses ; i++){
            for(int node : graph.get(i)){
                indegree[node]++;
            }
        }
        Queue<Integer> q = new LinkedList<>();
        for(int i = 0; i < indegree.length; i++){
            if(indegree[i] == 0){
                q.add(i);
            }
        }
        int[] ts = new int[numCourses];
        int i = 0;

        while(!q.isEmpty()){
            int node = q.remove();
            ts[i++] = node;

            for(int nbr : graph.get(node)){
                indegree[nbr]--;
                if(indegree[nbr] == 0){
                    q.add(nbr);
                }
            }

        }
        if(i == 0 || i < numCourses) return new int[]{};
        return ts;
    }
}
```


Description | Editorial | Solutions | Submissions
Code
Java Auto
Medium Topics Companies Hint
Solved
Testcase | Test Result | Accepted
All Submissions
Editorial
Solution
Analyze Complexity
11.3K
85
232 Online
Saved
Ln 36, Col 2

210. Course Schedule II

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [ai, bi]` indicates that you **must** take course `bi` first if you want to take course `ai`.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

Return *the ordering of courses you should take to finish all courses*. If there are many valid answers, return **any** of them. If it is impossible to finish all courses, return **an empty array**.

Example 1:

```
Input: numCourses = 2, prerequisites = [[1,0]]
Output: [0,1]
Explanation: There are a total of 2 courses to take. To take course 1 you should have finished course 0.
So the correct course order is [0,1].
```

Example 2:

```
Input: numCourses = 4, prerequisites = [[1,0],[2,0],[3,1],[3,2]]
Output: [0,2,1,3]
Explanation: There are a total of 4 courses to take. To take course 3 you should have finished both
courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0.
So one correct course order is [0,1,2,3]. Another correct ordering is [0,2,1,3].
```

Example 3:

```
Input: numCourses = 1, prerequisites = []
Output: [0]
```

Constraints:

- `1 <= numCourses <= 2000`
- `0 <= prerequisites.length <= numCourses * (numCourses - 1)`

```java
class Solution {
    public int[] findOrder(int numCourses, int[][] prerequisites) {
        List<List<Integer>> graph = new ArrayList<>();
        for(int i = 0 ; i < numCourses; i++) graph.add(new ArrayList<>());
        for(int i = 0 ; i < prerequisites.length; i++) graph.get(prerequisites[i][1]).add(prerequisites[i][0]);
        int[] indegree = new int[numCourses];
        for(int i = 0; i < numCourses ; i++){
            for(int node : graph.get(i)){
                indegree[node]++;
            }
        }
        Queue<Integer> q = new LinkedList<>();
        for(int i = 0; i < indegree.length; i++){
            if(indegree[i] == 0){
                q.add(i);
            }
        }
```

Accepted 45 / 45 testcases passed
Jiya submitted at Apr 06, 2025 13:59

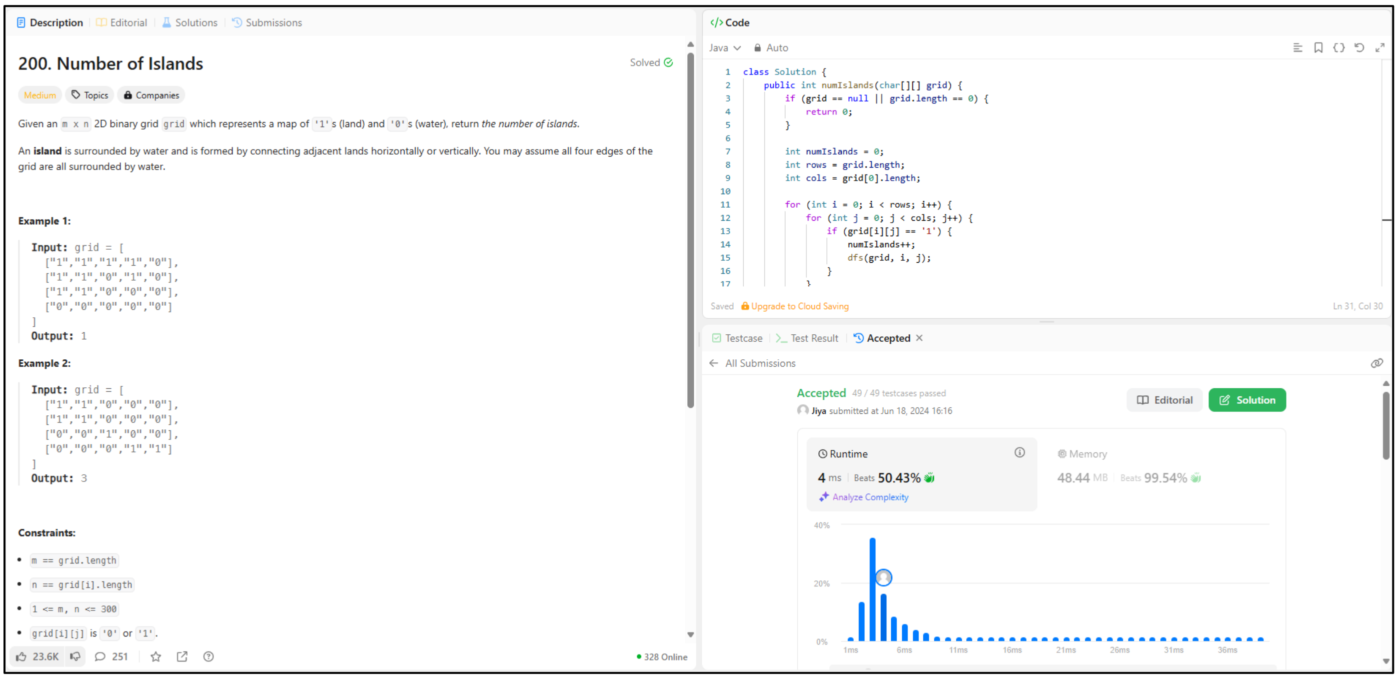
Runtime
6 ms | Beats 54.66%

Memory
46.72 MB | Beats 7.21%

200. Number of Islands

https://leetcode.com/problems/number-of-islands/description/

```
class Solution {
    public int numIslands(char[][] grid) {
        if (grid == null || grid.length == 0) {
            return 0;
        }
        int numIslands = 0;
        int rows = grid.length;
        int cols = grid[0].length;

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (grid[i][j] == '1') {
                    numIslands++;
                    dfs(grid, i, j);
                }
            }
        }
        return numIslands;
    }
    private void dfs(char[][] grid, int row, int col) {
        int rows = grid.length;
        int cols = grid[0].length;

        if (row < 0 || row >= rows || col < 0 || col >= cols || grid[row][col] ==
'0') {
            return;
        }

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

        dfs(grid, row - 1, col);
        dfs(grid, row + 1, col);
        dfs(grid, row, col - 1);
        dfs(grid, row, col + 1);
    }
}
```



200. Number of Islands

Given an `m x n` 2D binary grid `grid` which represents a map of `'1'`s (land) and `'0'`s (water), return *the number of islands*.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
Input: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
Output: 1
```

Example 2:

```
Input: grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
Output: 3
```

Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 300`
- `grid[i][j]` is `'0'` or `'1'`.

```
class Solution {
    public int numIslands(char[][] grid) {
        if (grid == null || grid.length == 0) {
            return 0;
        }

        int numIslands = 0;
        int rows = grid.length;
        int cols = grid[0].length;

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (grid[i][j] == '1') {
                    numIslands++;
                    dfs(grid, i, j);
                }
```

Accepted 49 / 49 testcases passed
Jiya submitted at Jun 18, 2024 16:16

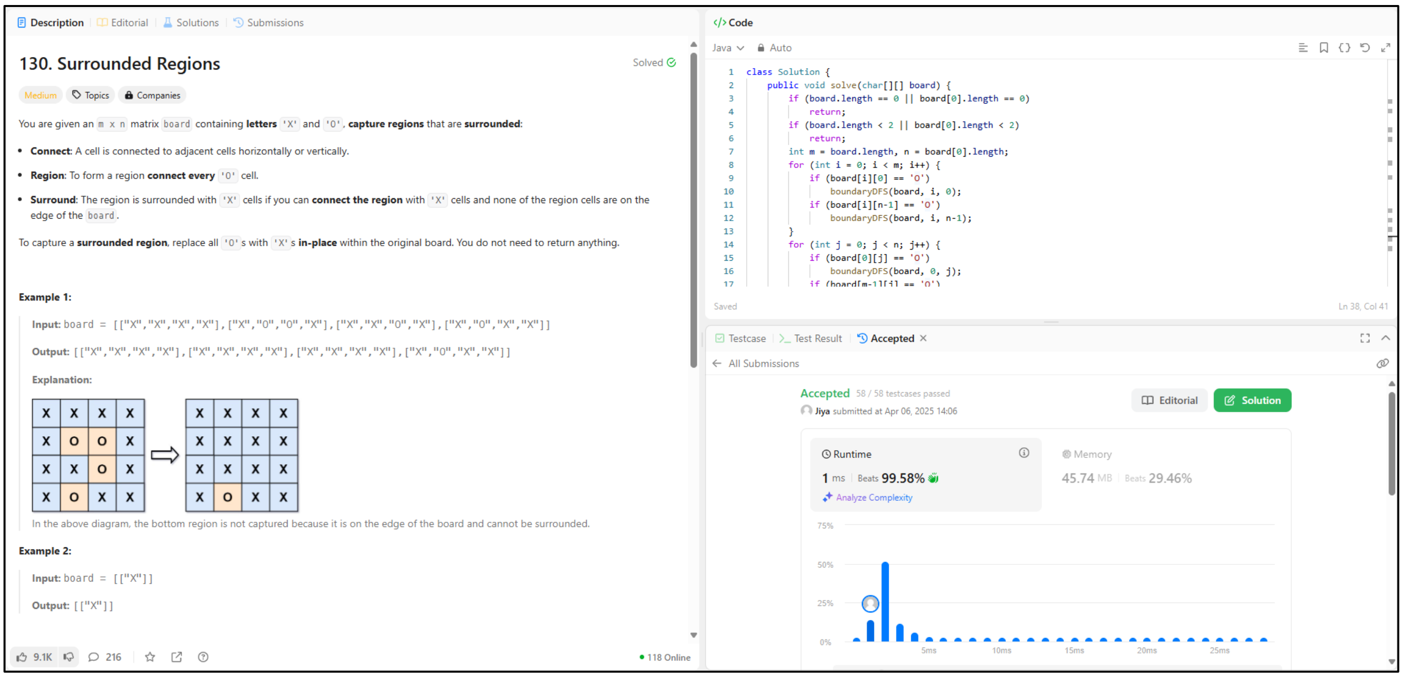
Runtime: 4 ms | Beats 50.43%

Memory: 48.44 MB | Beats 99.54%

130. Surrounded Regions

https://leetcode.com/problems/surrounded-regions/description/

```java
class Solution {
    public void solve(char[][] board) {
        if (board.length == 0 || board[0].length == 0)
            return;
        if (board.length < 2 || board[0].length < 2)
            return;
        int m = board.length, n = board[0].length;
        for (int i = 0; i < m; i++) {
            if (board[i][0] == 'O') boundaryDFS(board, i, 0);
            if (board[i][n-1] == 'O') boundaryDFS(board, i, n-1);
        }
        for (int j = 0; j < n; j++) {
            if (board[0][j] == 'O') boundaryDFS(board, 0, j);
            if (board[m-1][j] == 'O') boundaryDFS(board, m-1, 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] == '*')
                    board[i][j] = 'O';
            }
        }
    }
    private void boundaryDFS(char[][] board, int i, int j) {
        if (i < 0 || i > board.length - 1 || j <0 || j > board[0].length - 1)
            return;
        if (board[i][j] == 'O')
            board[i][j] = '*';
        if (i > 1 && board[i-1][j] == 'O')
            boundaryDFS(board, i-1, j);
        if (i < board.length - 2 && board[i+1][j] == 'O')
            boundaryDFS(board, i+1, j);
        if (j > 1 && board[i][j-1] == 'O')
            boundaryDFS(board, i, j-1);
        if (j < board[i].length - 2 && board[i][j+1] == 'O' )
            boundaryDFS(board, i, j+1);
    }
}
```



130. Surrounded Regions

Solved

You are given an `m x n` matrix `board` containing **letters** `'X'` and `'O'`, **capture regions** that are **surrounded**:

- **Connect**: A cell is connected to adjacent cells horizontally or vertically.
- **Region**: To form a region **connect every** `'O'` cell.
- **Surround**: The region is surrounded with `'X'` cells if you can **connect the region** with `'X'` cells and none of the region cells are on the edge of the `board`.

To capture a **surrounded region**, replace all `'O'`s with `'X'`s **in-place** within the original board. You do not need to return anything.

Example 1:

Input: board = [["X","X","X","X"],["X","O","O","X"],["X","X","O","X"],["X","O","X","X"]]

Output: [["X","X","X","X"],["X","X","X","X"],["X","X","X","X"],["X","O","X","X"]]

Explanation:

X	X	X	X
X	O	O	X
X	X	O	X
X	O	X	X

⇨

X	X	X	X
X	X	X	X
X	X	X	X
X	O	X	X

In the above diagram, the bottom region is not captured because it is on the edge of the board and cannot be surrounded.

Example 2:

Input: board = [["X"]]

Output: [["X"]]

```java
class Solution {
    public void solve(char[][] board) {
        if (board.length == 0 || board[0].length == 0)
            return;
        if (board.length < 2 || board[0].length < 2)
            return;
        int m = board.length, n = board[0].length;
        for (int i = 0; i < m; i++) {
            if (board[i][0] == 'O')
                boundaryDFS(board, i, 0);
            if (board[i][n-1] == 'O')
                boundaryDFS(board, i, n-1);
        }
        for (int j = 0; j < n; j++) {
            if (board[0][j] == 'O')
                boundaryDFS(board, 0, j);
            if (board[m-1][j] == 'O')
```

Accepted 58 / 58 testcases passed

Jiya submitted at Apr 06, 2025 14:06

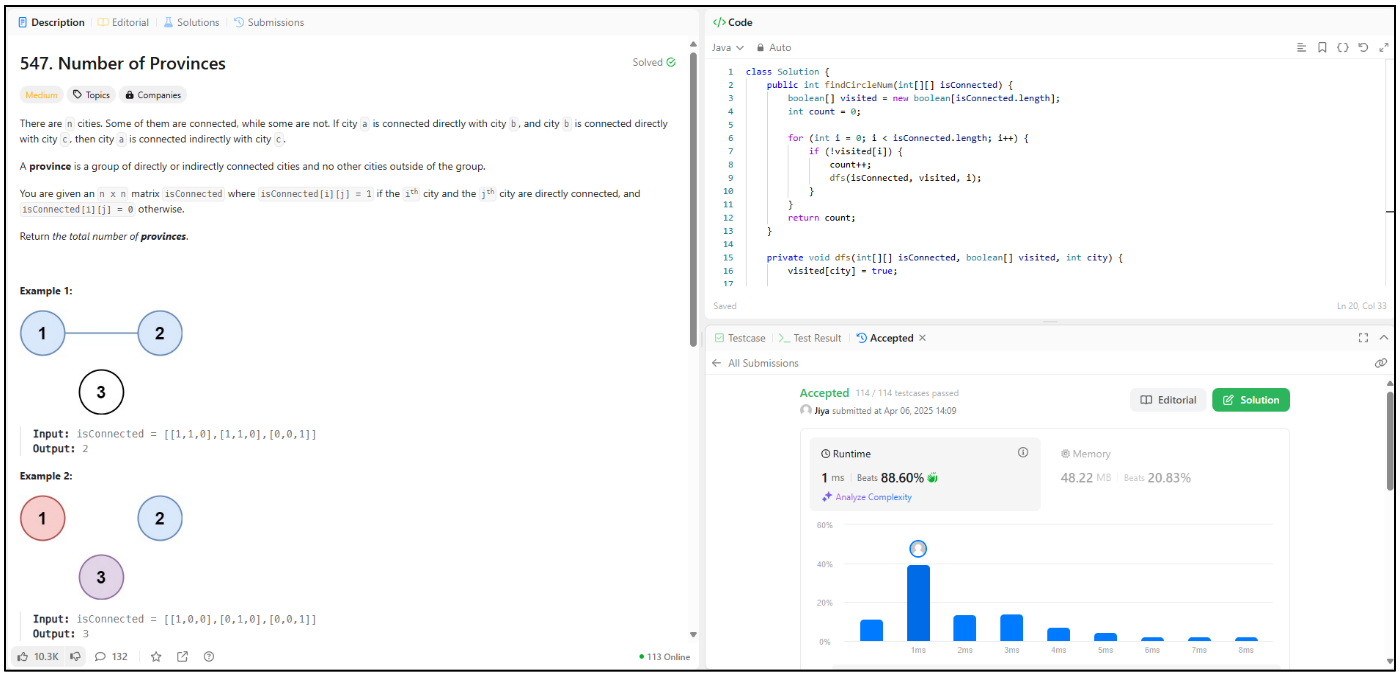
Runtime: 1 ms | Beats 99.58%

Memory: 45.74 MB | Beats 29.46%

547. Number of Provinces

https://leetcode.com/problems/number-of-provinces/description/

```
class Solution {
    public int findCircleNum(int[][] isConnected) {
        boolean[] visited = new boolean[isConnected.length];
        int count = 0;

        for (int i = 0; i < isConnected.length; i++) {
            if (!visited[i]) {
                count++;
                dfs(isConnected, visited, i);
            }
        }
        return count;
    }

    private void dfs(int[][] isConnected, boolean[] visited, int city) {
        visited[city] = true;

        for (int i = 0; i < isConnected.length; i++) {
            if (isConnected[city][i] == 1 && !visited[i]) {
                dfs(isConnected, visited, i);
            }
        }
    }
}
```



547. Number of Provinces

There are `n` cities. Some of them are connected, while some are not. If city `a` is connected directly with city `b`, and city `b` is connected directly with city `c`, then city `a` is connected indirectly with city `c`.

A **province** is a group of directly or indirectly connected cities and no other cities outside of the group.

You are given an `n x n` matrix `isConnected` where `isConnected[i][j] = 1` if the `i`th city and the `j`th city are directly connected, and `isConnected[i][j] = 0` otherwise.

Return *the total number of **provinces***.

Example 1:

```
Input: isConnected = [[1,1,0],[1,1,0],[0,0,1]]
Output: 2
```

Example 2:

```
Input: isConnected = [[1,0,0],[0,1,0],[0,0,1]]
Output: 3
```

```
class Solution {
    public int findCircleNum(int[][] isConnected) {
        boolean[] visited = new boolean[isConnected.length];
        int count = 0;

        for (int i = 0; i < isConnected.length; i++) {
            if (!visited[i]) {
                count++;
                dfs(isConnected, visited, i);
            }
        }
        return count;
    }

    private void dfs(int[][] isConnected, boolean[] visited, int city) {
        visited[city] = true;
```

Accepted 114 / 114 testcases passed

Jiya submitted at Apr 06, 2025 14:09

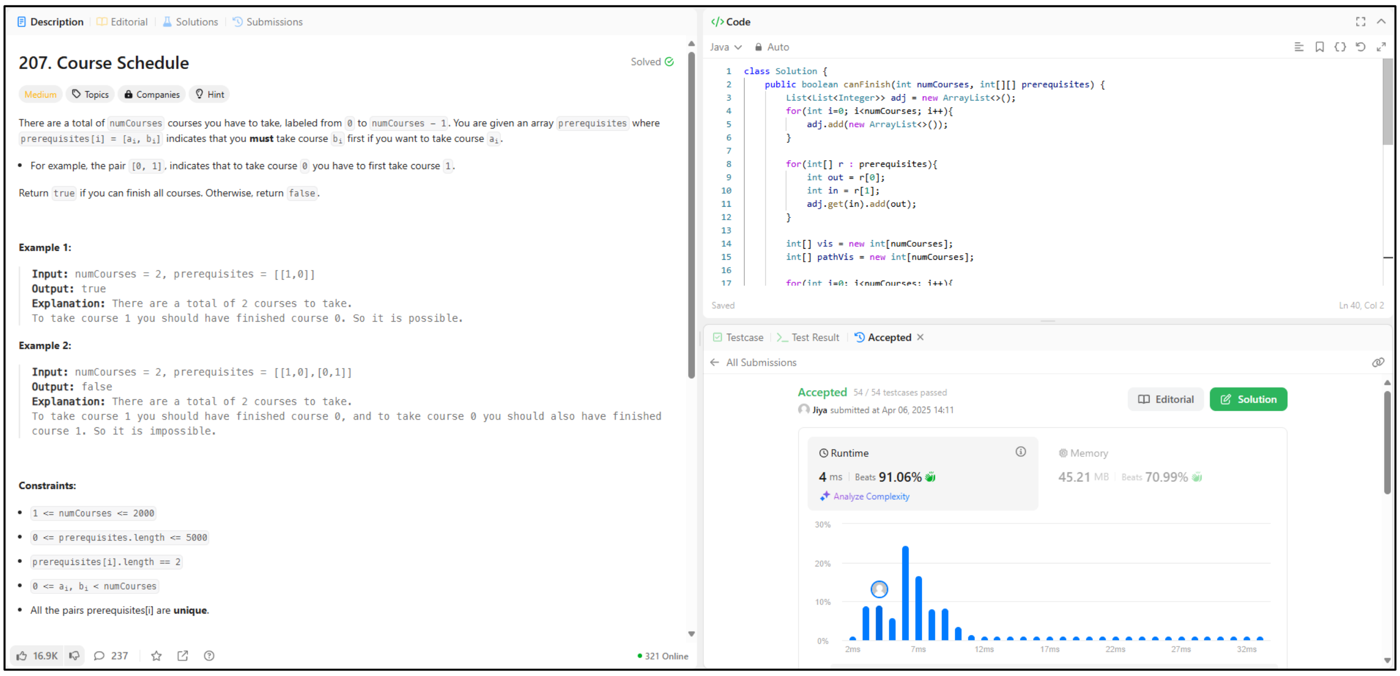
Runtime 1 ms | Beats 88.60%

Memory 48.22 MB | Beats 20.83%

207. Course Schedule

https://leetcode.com/problems/course-schedule/description/

```java
class Solution {
    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[] r : prerequisites){
            int out = r[0];
            int in = r[1];
            adj.get(in).add(out);
        }
        int[] vis = new int[numCourses];
        int[] pathVis = new int[numCourses];

        for(int i=0; i<numCourses; i++){
            if(vis[i] == 0){
                if(isCycle(adj, i, vis, pathVis)) return false;
            }
        }
        return true;
    }
    private boolean isCycle(List<List<Integer>> adj, int curr, int[] vis, int[]
pathVis){
        vis[curr] = 1;
        pathVis[curr] = 1;

        for(int nbr : adj.get(curr)){
            if(vis[nbr] == 0){
                if(isCycle(adj, nbr, vis, pathVis)) return true;
            }else if(vis[nbr] == 1 && pathVis[nbr] == 1){
                return true;
            }
        }
        pathVis[curr] = 0;
        return false;
    }
}
```



207. Course Schedule

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [`a_i`, `b_i`]` indicates that you **must** take course b_i first if you want to take course a_i.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

Return `true` if you can finish all courses. Otherwise, return `false`.

Example 1:

```
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0. So it is possible.
```

Example 2:

```
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0, and to take course 0 you should also have finished
course 1. So it is impossible.
```

Constraints:

- `1 <= numCourses <= 2000`
- `0 <= prerequisites.length <= 5000`
- `prerequisites[i].length == 2`
- `0 <=` a_i, b_i `< numCourses`
- All the pairs prerequisites[i] are **unique**.

16.9K 237 321 Online

Code

Java Auto

```java
class Solution {
    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[] r : prerequisites){
            int out = r[0];
            int in = r[1];
            adj.get(in).add(out);
        }

        int[] vis = new int[numCourses];
        int[] pathVis = new int[numCourses];

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

Saved Ln 40, Col 2

Testcase | Test Result | Accepted

All Submissions

Accepted 54 / 54 testcases passed

Jiya submitted at Apr 06, 2025 14:11

Editorial Solution

Runtime

4 ms | Beats 91.06%

Analyze Complexity

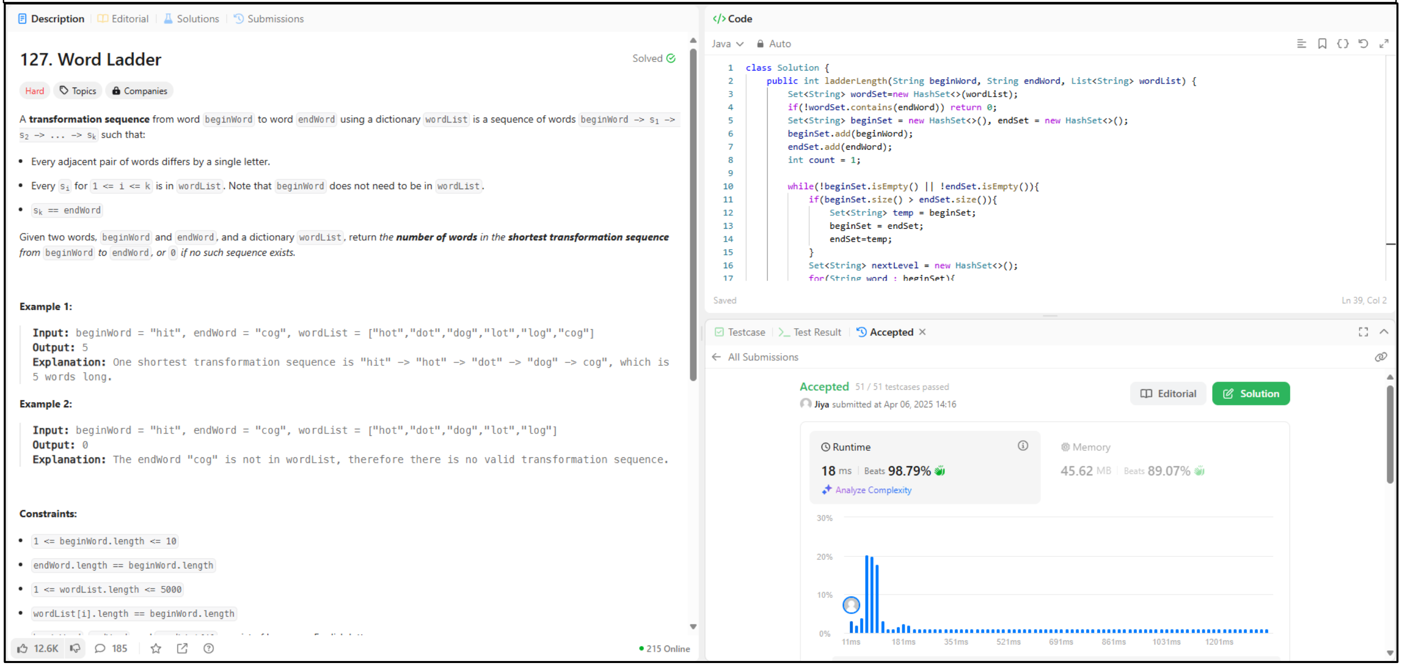
Memory

45.21 MB | Beats 70.99%

127. Word Ladder

https://leetcode.com/problems/word-ladder/description/

```java
class Solution {
    public int ladderLength(String beginWord, String endWord, List<String> wordList)
{
        Set<String> wordSet=new HashSet<>(wordList);
        if(!wordSet.contains(endWord)) return 0;
        Set<String> beginSet = new HashSet<>(), endSet = new HashSet<>();
        beginSet.add(beginWord);
        endSet.add(endWord);
        int count = 1;
        while(!beginSet.isEmpty() || !endSet.isEmpty()){
            if(beginSet.size() > endSet.size()){
                Set<String> temp = beginSet;
                beginSet = endSet;
                endSet=temp;
            }
            Set<String> nextLevel = new HashSet<>();
            for(String word : beginSet){
                char[] wordCh = word.toCharArray();
                for(int i=0; i<wordCh.length; i++){
                    char org = wordCh[i];
                    for(char c = 'a'; c <= 'z'; c++){
                        if(c==org)continue;
                        wordCh[i]=c;
                        String nWord = new String(wordCh);
                        if(endSet.contains(nWord))return count + 1;
                        if(wordSet.contains(nWord)){
                            nextLevel.add(nWord);
                            wordSet.remove(nWord);
                        }
                    }
                    wordCh[i] = org;
                }
            }
            if (nextLevel.isEmpty()) return 0;
            beginSet = nextLevel;
            count++;
        }
        return 0;
    }
}
```



127. Word Ladder

A **transformation sequence** from word `beginWord` to word `endWord` using a dictionary `wordList` is a sequence of words `beginWord -> s1 -> s2 -> ... -> sk` such that:

- Every adjacent pair of words differs by a single letter.
- Every `si` for `1 <= i <= k` is in `wordList`. Note that `beginWord` does not need to be in `wordList`.
- `sk == endWord`

Given two words, `beginWord` and `endWord`, and a dictionary `wordList`, return *the **number of words** in the **shortest transformation sequence** from* `beginWord` *to* `endWord`*, or* `0` *if no such sequence exists.*

Example 1:

```
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]
Output: 5
Explanation: One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> cog", which is
5 words long.
```

Example 2:

```
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]
Output: 0
Explanation: The endWord "cog" is not in wordList, therefore there is no valid transformation sequence.
```

Constraints:

- `1 <= beginWord.length <= 10`
- `endWord.length == beginWord.length`
- `1 <= wordList.length <= 5000`
- `wordList[i].length == beginWord.length`

```java
class Solution {
    public int ladderLength(String beginWord, String endWord, List<String> wordList) {
        Set<String> wordSet=new HashSet<>(wordList);
        if(!wordSet.contains(endWord)) return 0;
        Set<String> beginSet = new HashSet<>(), endSet = new HashSet<>();
        beginSet.add(beginWord);
        endSet.add(endWord);
        int count = 1;

        while(!beginSet.isEmpty() || !endSet.isEmpty()){
            if(beginSet.size() > endSet.size()){
                Set<String> temp = beginSet;
                beginSet = endSet;
                endSet=temp;
            }
            Set<String> nextLevel = new HashSet<>();
            for(String word : beginSet){
```

Accepted 51 / 51 testcases passed
Jiya submitted at Apr 06, 2025 14:16

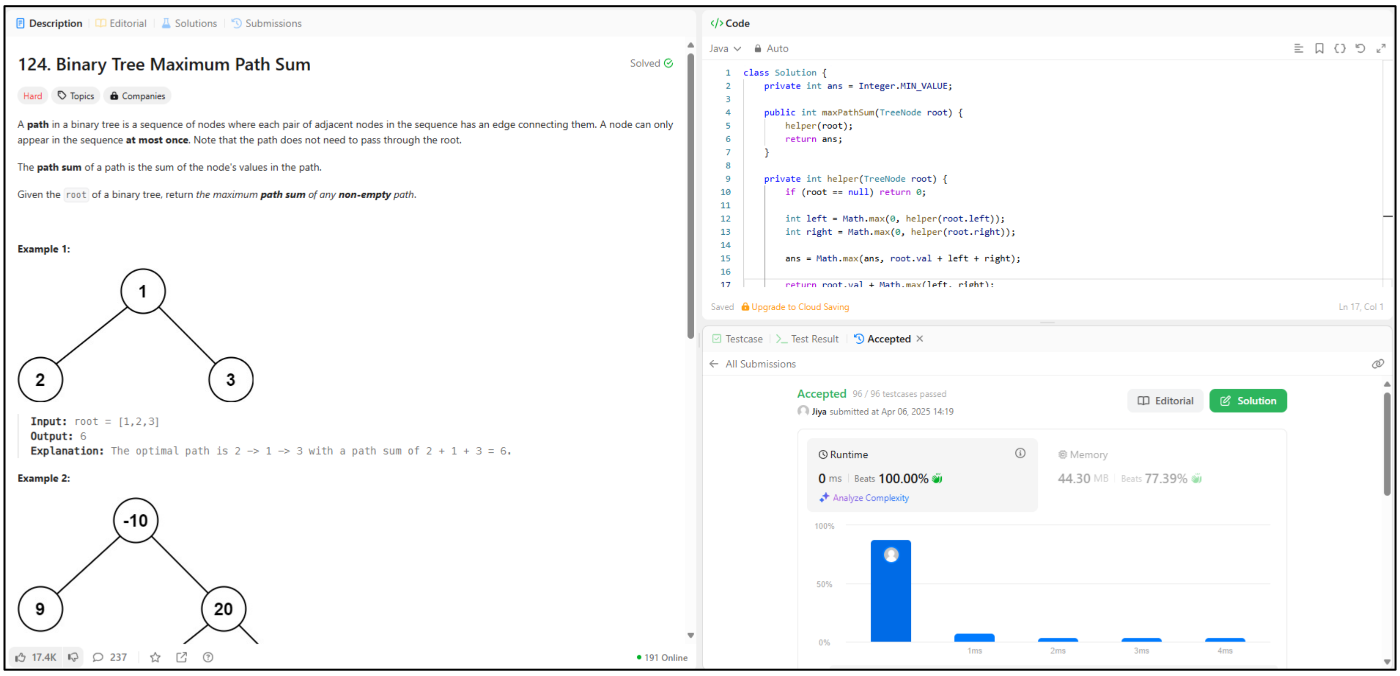
Runtime
18 ms | Beats 98.79%

Memory
45.62 MB | Beats 89.07%

124. Binary Tree Maximum Path Sum

https://leetcode.com/problems/binary-tree-maximum-path-sum/description/

```java
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);
    }
}
```



124. Binary Tree Maximum Path Sum

A **path** in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence **at most once**. Note that the path does not need to pass through the root.

The **path sum** of a path is the sum of the node's values in the path.

Given the `root` of a binary tree, return *the maximum* ***path sum*** *of any* ***non-empty*** *path.*

Example 1:

```
Input: root = [1,2,3]
Output: 6
Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.
```

Example 2:

```java
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);
```

Saved Upgrade to Cloud Saving Ln 17, Col 1

Accepted 96 / 96 testcases passed
Jiya submitted at Apr 06, 2025 14:19

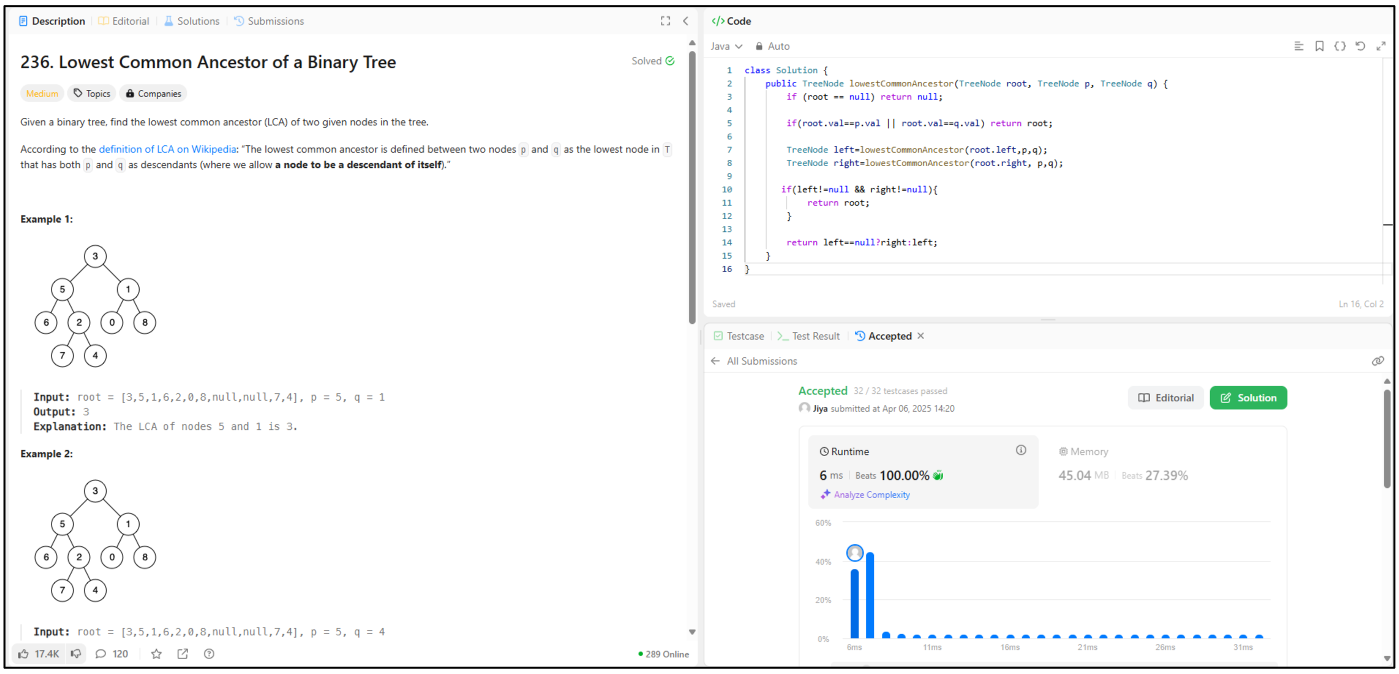
Runtime
0 ms | Beats 100.00%
Analyze Complexity

Memory
44.30 MB | Beats 77.39%

236. Lowest Common Ancestor of a Binary Tree

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/description/

```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if (root == null) return null;

        if(root.val==p.val || root.val==q.val) return root;

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

       if(left!=null && right!=null){
            return root;
        }

        return left==null?right:left;
    }
}
```



236. Lowest Common Ancestor of a Binary Tree

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

Example 1:

```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.
```

Example 2:

```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4
```

```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if (root == null) return null;

        if(root.val==p.val || root.val==q.val) return root;

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

       if(left!=null && right!=null){
            return root;
        }

        return left==null?right:left;
    }
}
```

Accepted 32 / 32 testcases passed
Jiya submitted at Apr 06, 2025 14:20

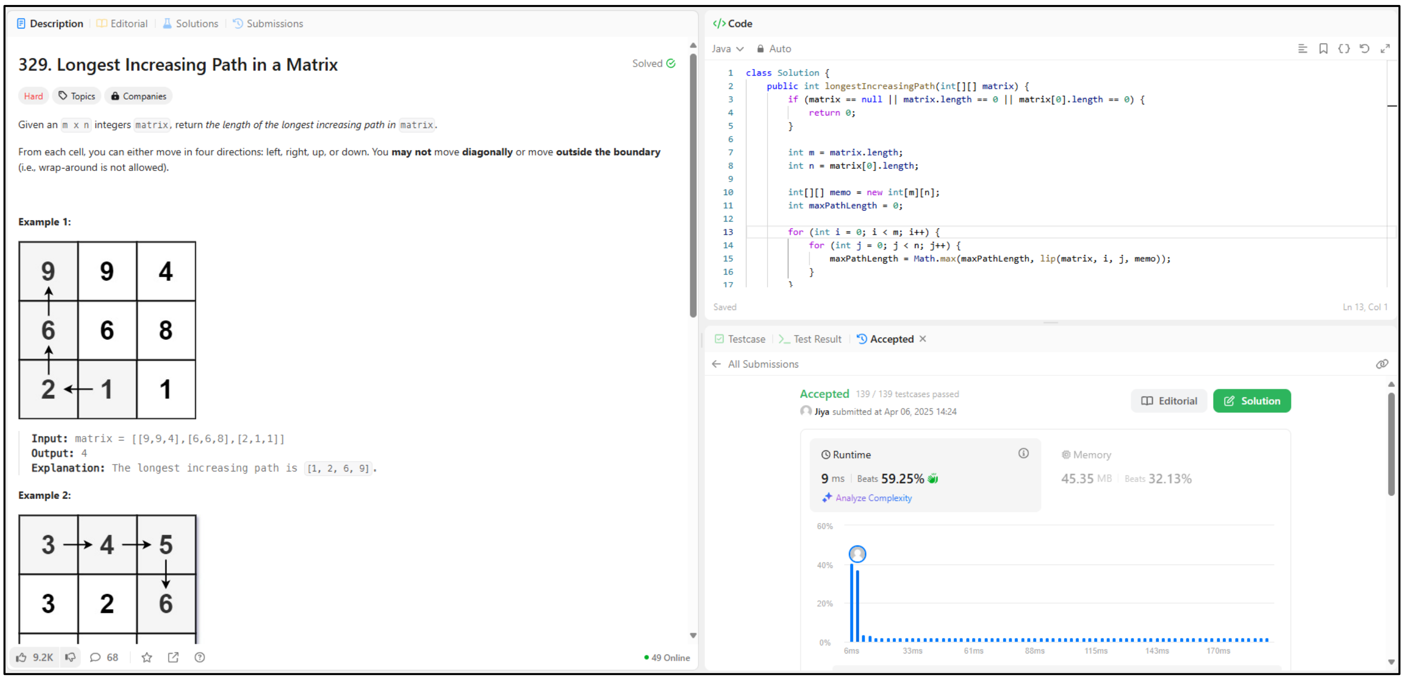
Runtime: 6 ms | Beats 100.00%

Memory: 45.04 MB | Beats 27.39%

329. Longest Increasing Path in a Matrix

https://leetcode.com/problems/longest-increasing-path-in-a-matrix/description/

```java
public class Solution {
    public int longestIncreasingPath(int[][] matrix) {
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {
            return 0;
        }
        int[][] cache = new int[matrix.length][matrix[0].length];
        int max = 0;
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                int length = findSmallAround(i, j, matrix, cache,
Integer.MAX_VALUE);
                max = Math.max(length, max);
            }
        }
        return max;
    }
    private int findSmallAround(int i, int j, int[][] matrix, int[][] cache, int
pre) {
        if (i < 0 || i >= matrix.length || j < 0 || j >= matrix[0].length ||
matrix[i][j] >= pre) {
            return 0;
        }
        if (cache[i][j] > 0) return cache[i][j];
        else {
            int cur = matrix[i][j];
            int tempMax = 0;
            tempMax = Math.max(findSmallAround(i - 1, j, matrix, cache, cur),
tempMax);
            tempMax = Math.max(findSmallAround(i + 1, j, matrix, cache, cur),
tempMax);
            tempMax = Math.max(findSmallAround(i, j - 1, matrix, cache, cur),
tempMax);
            tempMax = Math.max(findSmallAround(i, j + 1, matrix, cache, cur),
tempMax);
            cache[i][j] = ++tempMax;  tempMax;
        }
    }
}
```



329. Longest Increasing Path in a Matrix

Given an `m x n` integers `matrix`, return *the length of the longest increasing path in* `matrix`.

From each cell, you can either move in four directions: left, right, up, or down. You **may not** move **diagonally** or move **outside the boundary** (i.e., wrap-around is not allowed).

Example 1:

```
Input: matrix = [[9,9,4],[6,6,8],[2,1,1]]
Output: 4
Explanation: The longest increasing path is [1, 2, 6, 9].
```

Example 2:

```java
class Solution {
    public int longestIncreasingPath(int[][] matrix) {
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {
            return 0;
        }

        int m = matrix.length;
        int n = matrix[0].length;

        int[][] memo = new int[m][n];
        int maxPathLength = 0;

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                maxPathLength = Math.max(maxPathLength, lip(matrix, i, j, memo));
            }
        }
```

Accepted 139 / 139 testcases passed
Jiya submitted at Apr 06, 2025 14:24

Runtime: 9 ms | Beats 59.25%
Memory: 45.35 MB | Beats 32.13%