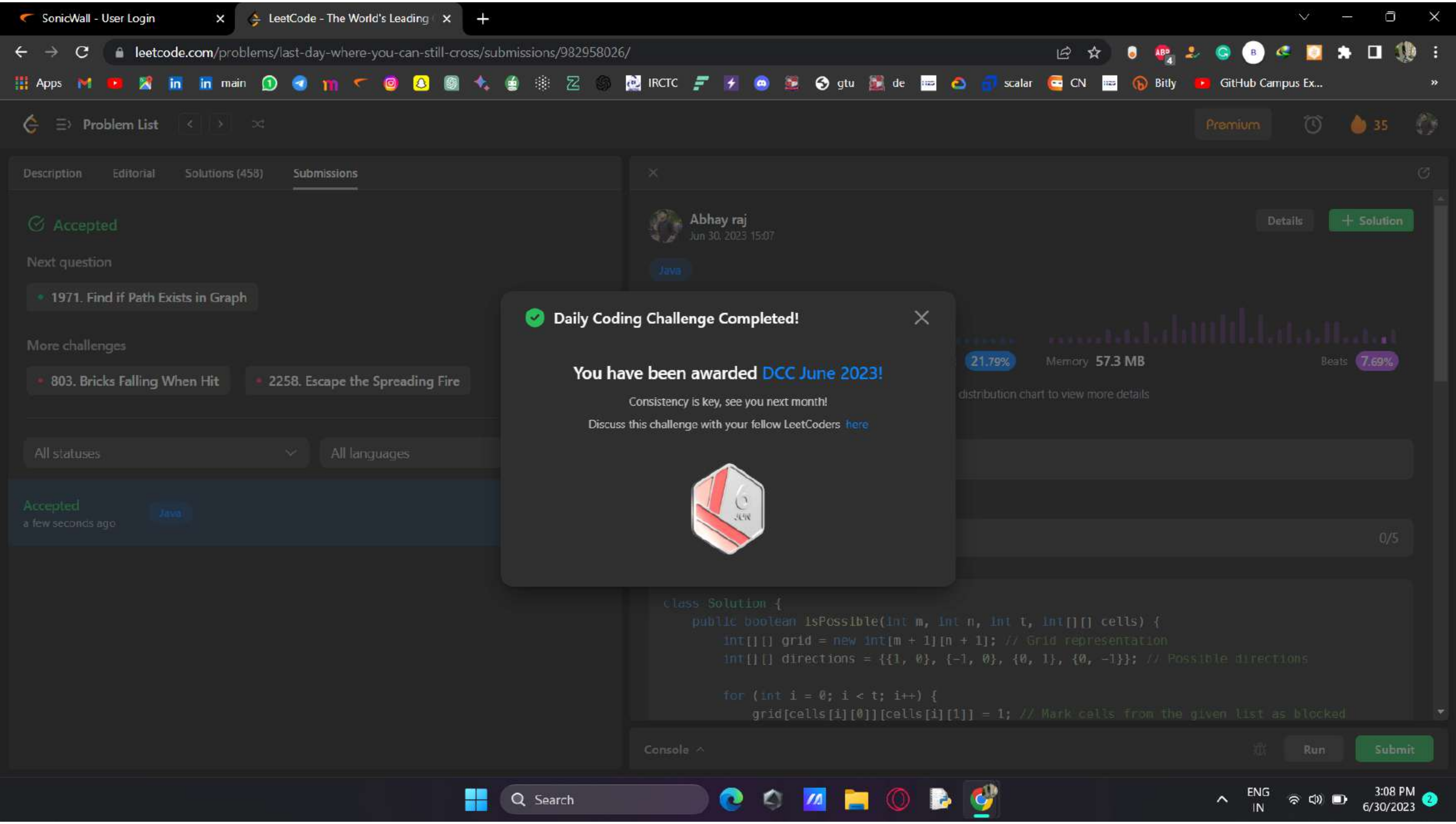


Accepted

Next question

1971. Find if Path Exists in Graph

More challenges

803. Bricks Falling When Hit

2258. Escape the Spreading Fire

Accepted a few seconds ago Java

Abhay raj
Jun 30, 2023 15:07

Java

21.79% Memory 57.3 MB Beats 7.69%

distribution chart to view more details

0/5

Daily Coding Challenge Completed!

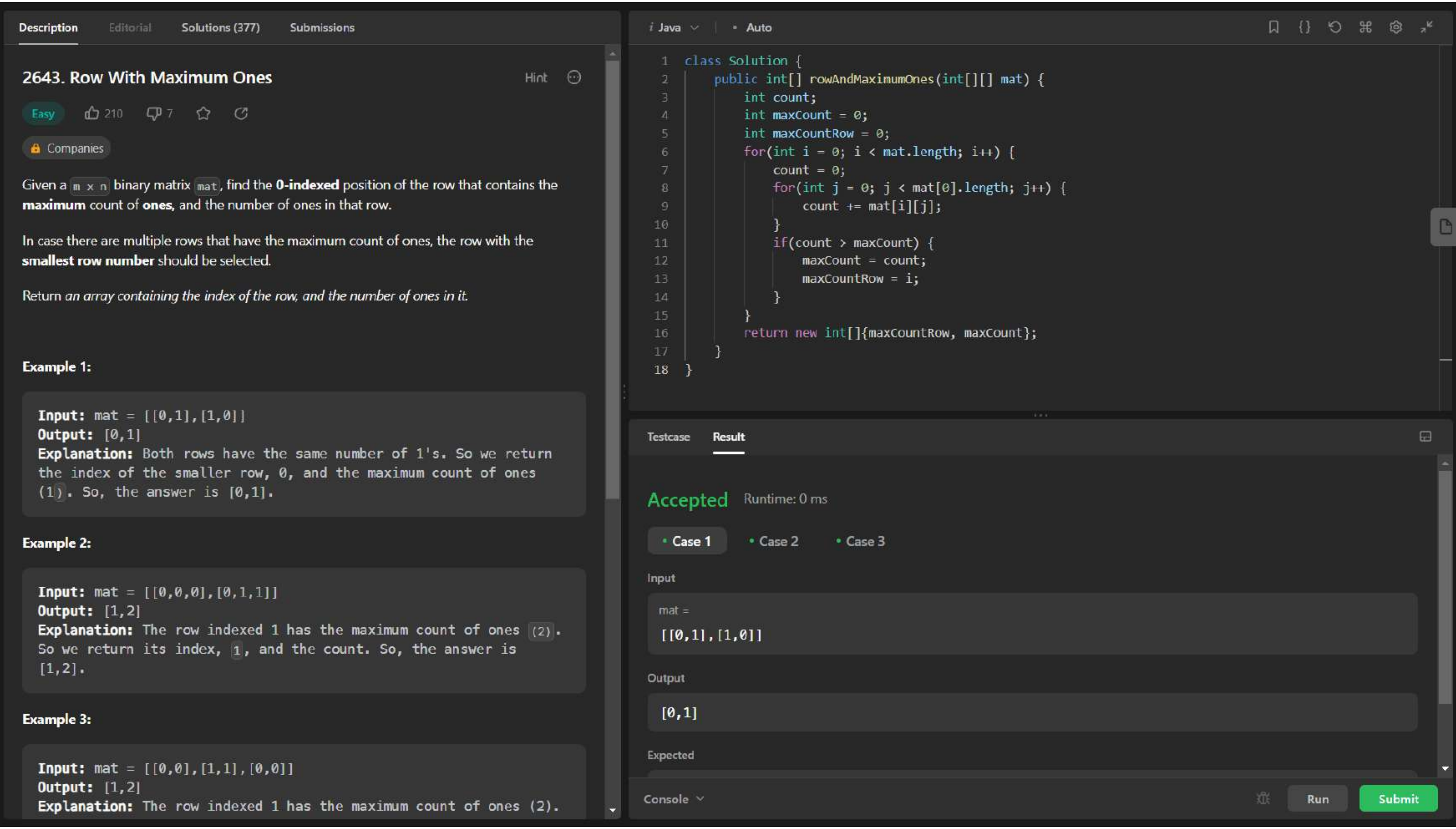
You have been awarded DCC June 2023!

Consistency is key, see you next month!

Discuss this challenge with your fellow LeetCoders here

```
class Solution {
    public boolean isPossible(int m, int n, int t, int[][] cells) {
        int[][] grid = new int[m + 1][n + 1]; // Grid representation
        int[][] directions = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}}; // Possible directions

        for (int i = 0; i < t; i++) {
            grid[cells[i][0]][cells[i][1]] = 1; // Mark cells from the given list as blocked
```



Description | Editorial | Solutions (377) | Submissions

2643. Row With Maximum Ones

Easy · 210 · 7 · Companies

Given a `m x n` binary matrix `mat`, find the **0-indexed** position of the row that contains the **maximum** count of **ones,** and the number of ones in that row.

In case there are multiple rows that have the maximum count of ones, the row with the **smallest row number** should be selected.

Return *an array containing the index of the row, and the number of ones in it.*

Example 1:

```
Input: mat = [[0,1],[1,0]]
Output: [0,1]
Explanation: Both rows have the same number of 1's. So we return
the index of the smaller row, 0, and the maximum count of ones
(1). So, the answer is [0,1].
```

Example 2:

```
Input: mat = [[0,0,0],[0,1,1]]
Output: [1,2]
Explanation: The row indexed 1 has the maximum count of ones (2).
So we return its index, 1, and the count. So, the answer is
[1,2].
```

Example 3:

```
Input: mat = [[0,0],[1,1],[0,0]]
Output: [1,2]
Explanation: The row indexed 1 has the maximum count of ones (2).
```

Java · Auto

```java
class Solution {
    public int[] rowAndMaximumOnes(int[][] mat) {
        int count;
        int maxCount = 0;
        int maxCountRow = 0;
        for(int i = 0; i < mat.length; i++) {
            count = 0;
            for(int j = 0; j < mat[0].length; j++) {
                count += mat[i][j];
            }
            if(count > maxCount) {
                maxCount = count;
                maxCountRow = i;
            }
        }
        return new int[]{maxCountRow, maxCount};
    }
}
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 · Case 2 · Case 3

Input

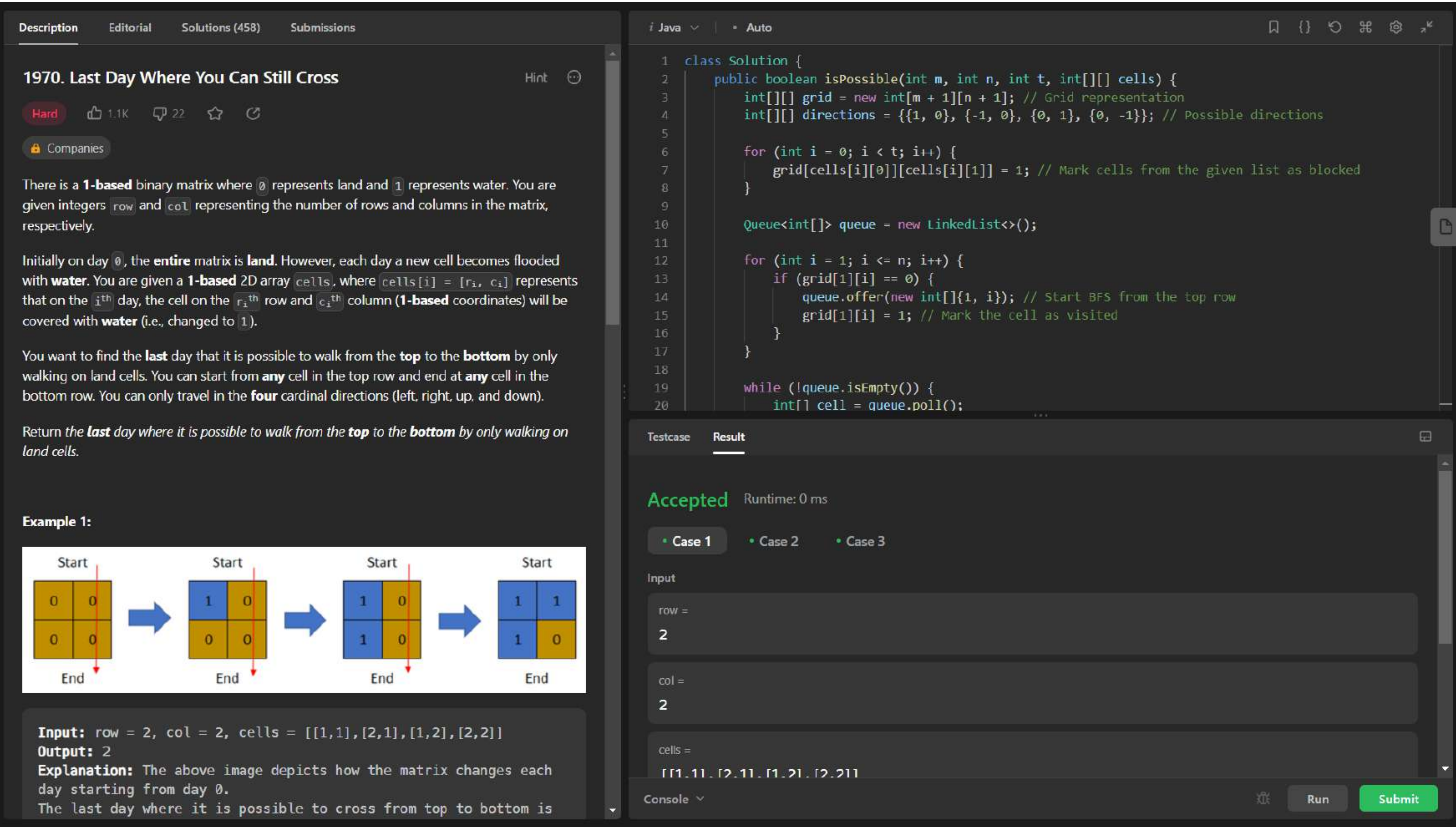
mat =

```
[[0,1],[1,0]]
```

Output

```
[0,1]
```

Expected



1970. Last Day Where You Can Still Cross

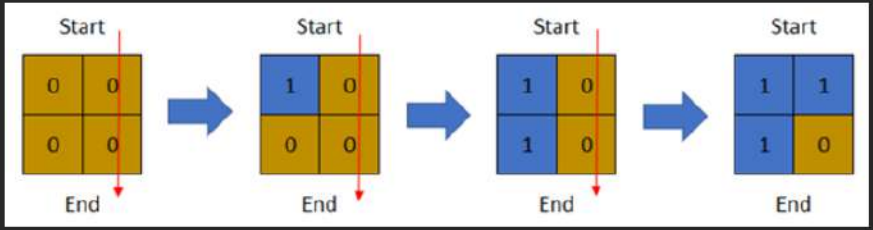
There is a **1-based** binary matrix where `0` represents land and `1` represents water. You are given integers `row` and `col` representing the number of rows and columns in the matrix, respectively.

Initially on day `0`, the **entire** matrix is **land**. However, each day a new cell becomes flooded with **water**. You are given a **1-based** 2D array `cells`, where `cells[i] = [ri, ci]` represents that on the `i`th day, the cell on the `ri`th row and `ci`th column (**1-based** coordinates) will be covered with **water** (i.e., changed to `1`).

You want to find the **last** day that it is possible to walk from the **top** to the **bottom** by only walking on land cells. You can start from **any** cell in the top row and end at **any** cell in the bottom row. You can only travel in the **four** cardinal directions (left, right, up, and down).

Return *the **last** day where it is possible to walk from the **top** to the **bottom** by only walking on land cells.*

Example 1:



```
Input: row = 2, col = 2, cells = [[1,1],[2,1],[1,2],[2,2]]
Output: 2
Explanation: The above image depicts how the matrix changes each
day starting from day 0.
The last day where it is possible to cross from top to bottom is
```

```java
class Solution {
    public boolean isPossible(int m, int n, int t, int[][] cells) {
        int[][] grid = new int[m + 1][n + 1]; // Grid representation
        int[][] directions = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}}; // Possible directions

        for (int i = 0; i < t; i++) {
            grid[cells[i][0]][cells[i][1]] = 1; // Mark cells from the given list as blocked
        }

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

        for (int i = 1; i <= n; i++) {
            if (grid[1][i] == 0) {
                queue.offer(new int[]{1, i}); // Start BFS from the top row
                grid[1][i] = 1; // Mark the cell as visited
            }
        }

        while (!queue.isEmpty()) {
            int[] cell = queue.poll();
```

Accepted Runtime: 0 ms

Input

row =
2

col =
2

cells =
[[1,1],[2,1],[1,2],[2,2]]