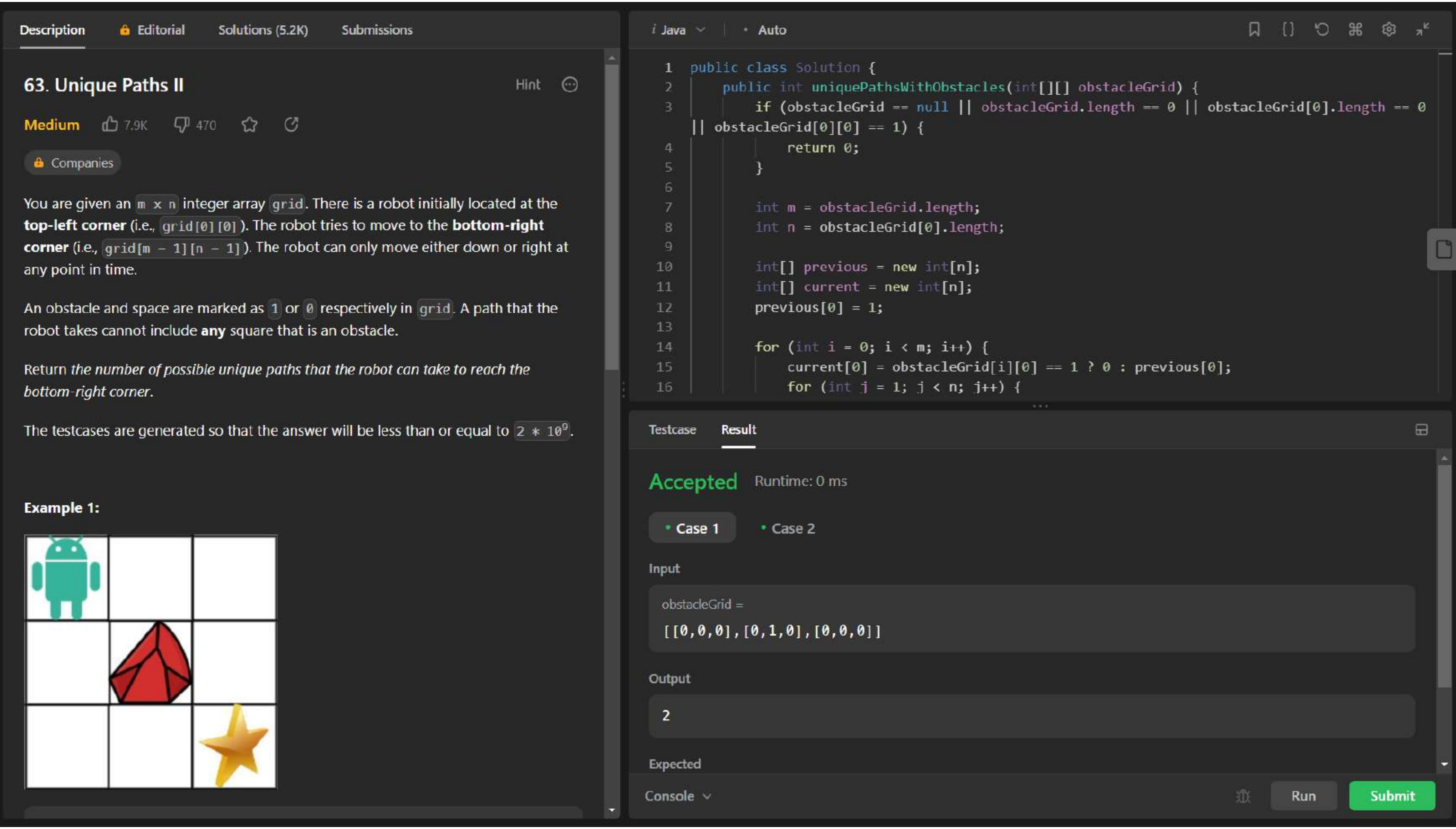


63. Unique Paths II

You are given an `m x n` integer array `grid`. There is a robot initially located at the **top-left corner** (i.e., `grid[0][0]`). The robot tries to move to the **bottom-right corner** (i.e., `grid[m - 1][n - 1]`). The robot can only move either down or right at any point in time.

An obstacle and space are marked as `1` or `0` respectively in `grid`. A path that the robot takes cannot include **any** square that is an obstacle.

Return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The testcases are generated so that the answer will be less than or equal to $2 * 10^9$.

Example 1:

```java
public class Solution {
    public int uniquePathsWithObstacles(int[][] obstacleGrid) {
        if (obstacleGrid == null || obstacleGrid.length == 0 || obstacleGrid[0].length == 0 || obstacleGrid[0][0] == 1) {
            return 0;
        }

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

        int[] previous = new int[n];
        int[] current = new int[n];
        previous[0] = 1;

        for (int i = 0; i < m; i++) {
            current[0] = obstacleGrid[i][0] == 1 ? 0 : previous[0];
            for (int j = 1; j < n; j++) {
```

Accepted Runtime: 0 ms

Input

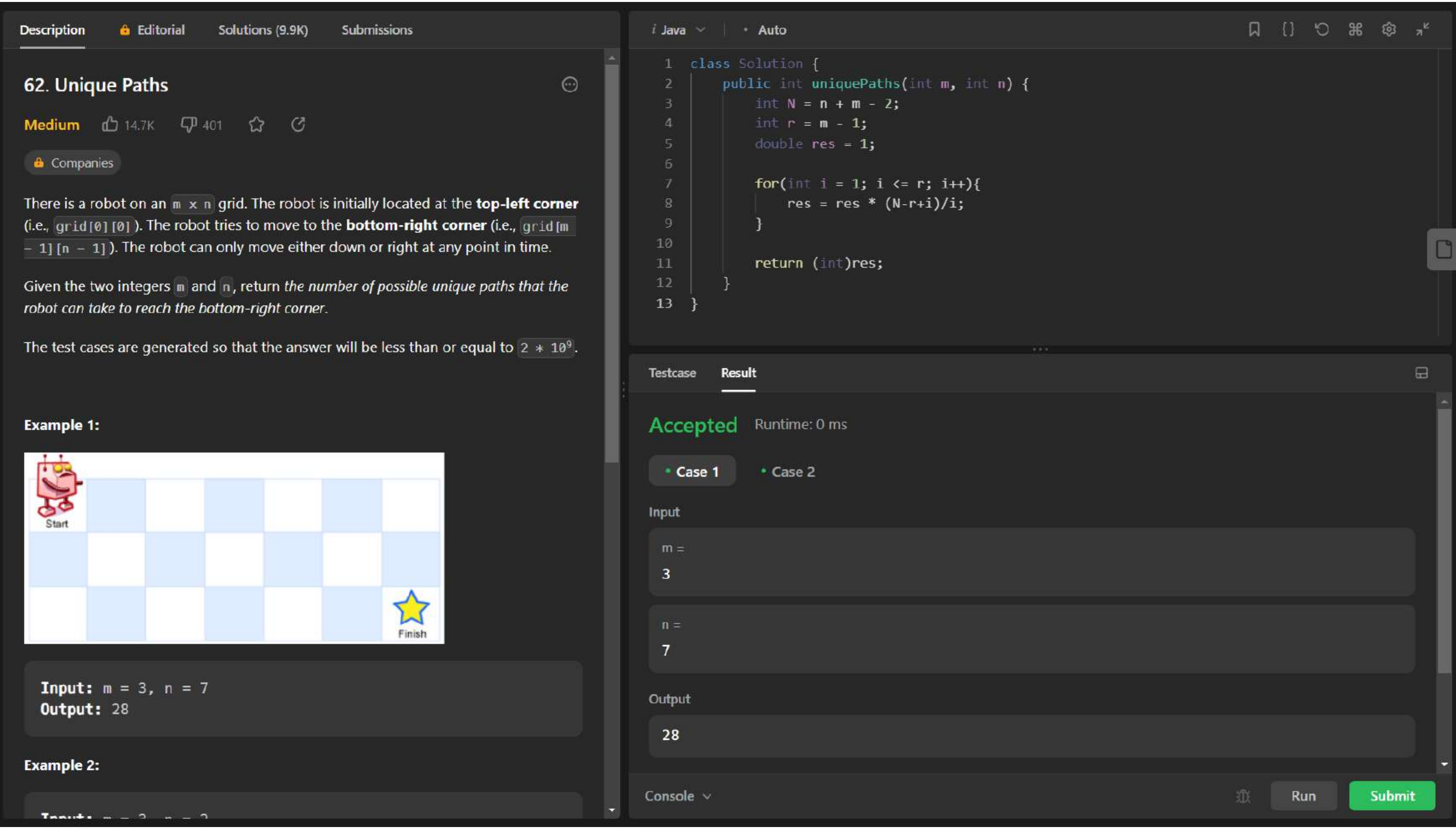
obstacleGrid =

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

Output

```
2
```

Expected



62. Unique Paths

Medium 14.7K 401

Companies

There is a robot on an `m x n` grid. The robot is initially located at the **top-left corner** (i.e., `grid[0][0]`). The robot tries to move to the **bottom-right corner** (i.e., `grid[m - 1][n - 1]`). The robot can only move either down or right at any point in time.

Given the two integers `m` and `n`, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The test cases are generated so that the answer will be less than or equal to `2 * 10`^{9}.

Example 1:

```
Input: m = 3, n = 7
Output: 28
```

Example 2:

```
Input: m = 3, n = 2
```

```java
class Solution {
    public int uniquePaths(int m, int n) {
        int N = n + m - 2;
        int r = m - 1;
        double res = 1;

        for(int i = 1; i <= r; i++){
            res = res * (N-r+i)/i;
        }

        return (int)res;
    }
}
```

Accepted Runtime: 0 ms

Input

m =

3

n =

7

Output

28