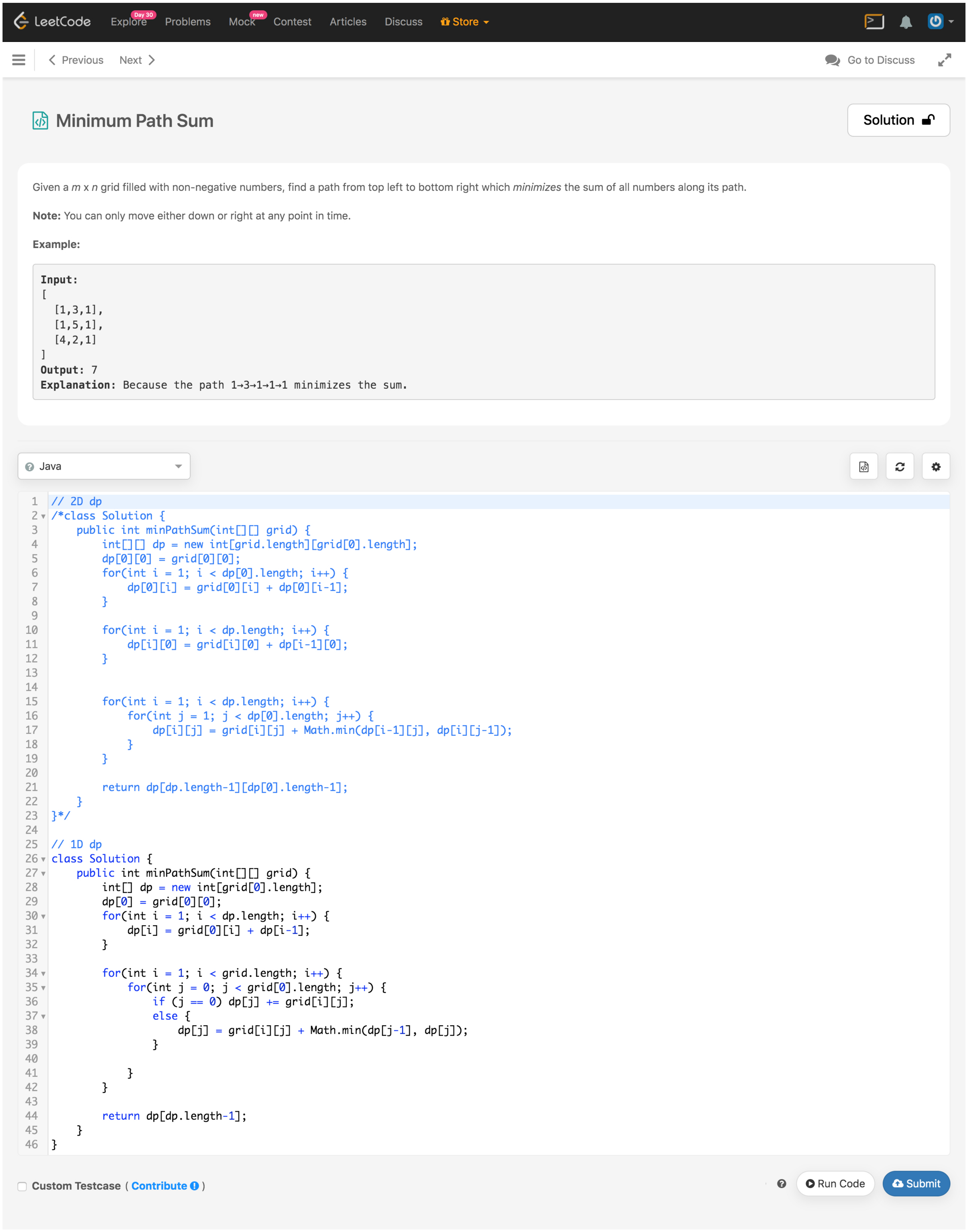

Minimum Path Sum

Given a *m* x *n* grid filled with non-negative numbers, find a path from top left to bottom right which *minimizes* the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

Example:

```
Input:
[
  [1,3,1],
  [1,5,1],
  [4,2,1]
]
Output: 7
Explanation: Because the path 1→3→1→1→1 minimizes the sum.
```

```java
// 2D dp
/*class Solution {
    public int minPathSum(int[][] grid) {
        int[][] dp = new int[grid.length][grid[0].length];
        dp[0][0] = grid[0][0];
        for(int i = 1; i < dp[0].length; i++) {
            dp[0][i] = grid[0][i] + dp[0][i-1];
        }

        for(int i = 1; i < dp.length; i++) {
            dp[i][0] = grid[i][0] + dp[i-1][0];
        }


        for(int i = 1; i < dp.length; i++) {
            for(int j = 1; j < dp[0].length; j++) {
                dp[i][j] = grid[i][j] + Math.min(dp[i-1][j], dp[i][j-1]);
            }
        }

        return dp[dp.length-1][dp[0].length-1];
    }
}*/

// 1D dp
class Solution {
    public int minPathSum(int[][] grid) {
        int[] dp = new int[grid[0].length];
        dp[0] = grid[0][0];
        for(int i = 1; i < dp.length; i++) {
            dp[i] = grid[0][i] + dp[i-1];
        }

        for(int i = 1; i < grid.length; i++) {
            for(int j = 0; j < grid[0].length; j++) {
                if (j == 0) dp[j] += grid[i][j];
                else {
                    dp[j] = grid[i][j] + Math.min(dp[j-1], dp[j]);
                }

            }
        }

        return dp[dp.length-1];
    }
}
```