UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

FACULTAD DE PRODUCCION Y SERVICIOS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Curso: Laboratorio de Análisis y Diseño de Algoritmos

Aula 08

Presentado por:

Tacca Apaza, Nohelia Estefhania

Docente:

Alex Josue Florez Farfan

Grupo-"B"

Arequipa - Perú

Diciembre 2021

Ejercicio 01 - Unique Path II

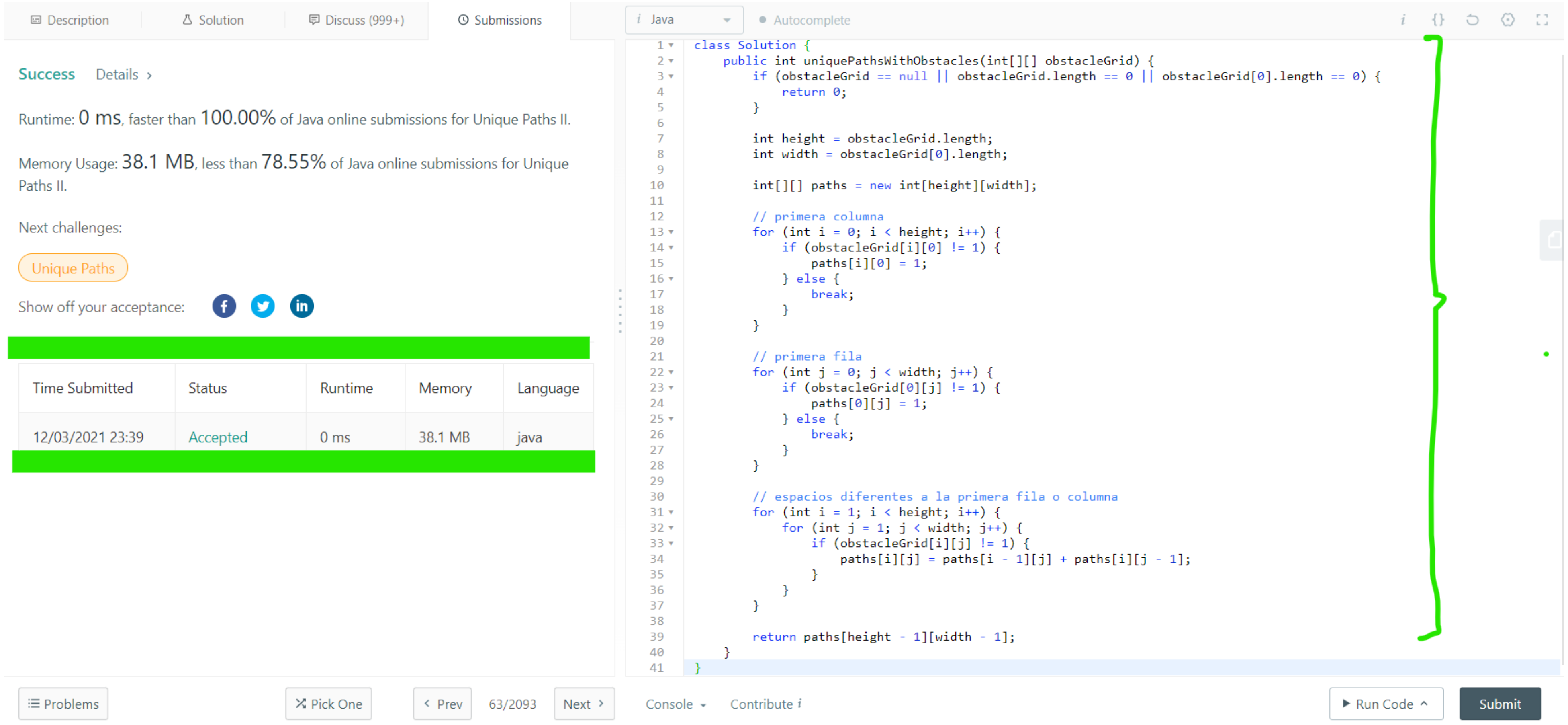
A robot is located at the top-left corner of a m x n grid.

The robot can only move either down or right at any point in time.

The robot is trying to reach the bottom-right corner of the grid.

Prueba en lenguaje de programación Java:

```
Input :

 0 0 0
 0 1 0
 0 0 0

Output : 2
```



Success Details ›

Runtime: 0 ms, faster than 100.00% of Java online submissions for Unique Paths II.

Memory Usage: 38.1 MB, less than 78.55% of Java online submissions for Unique Paths II.

Next challenges:

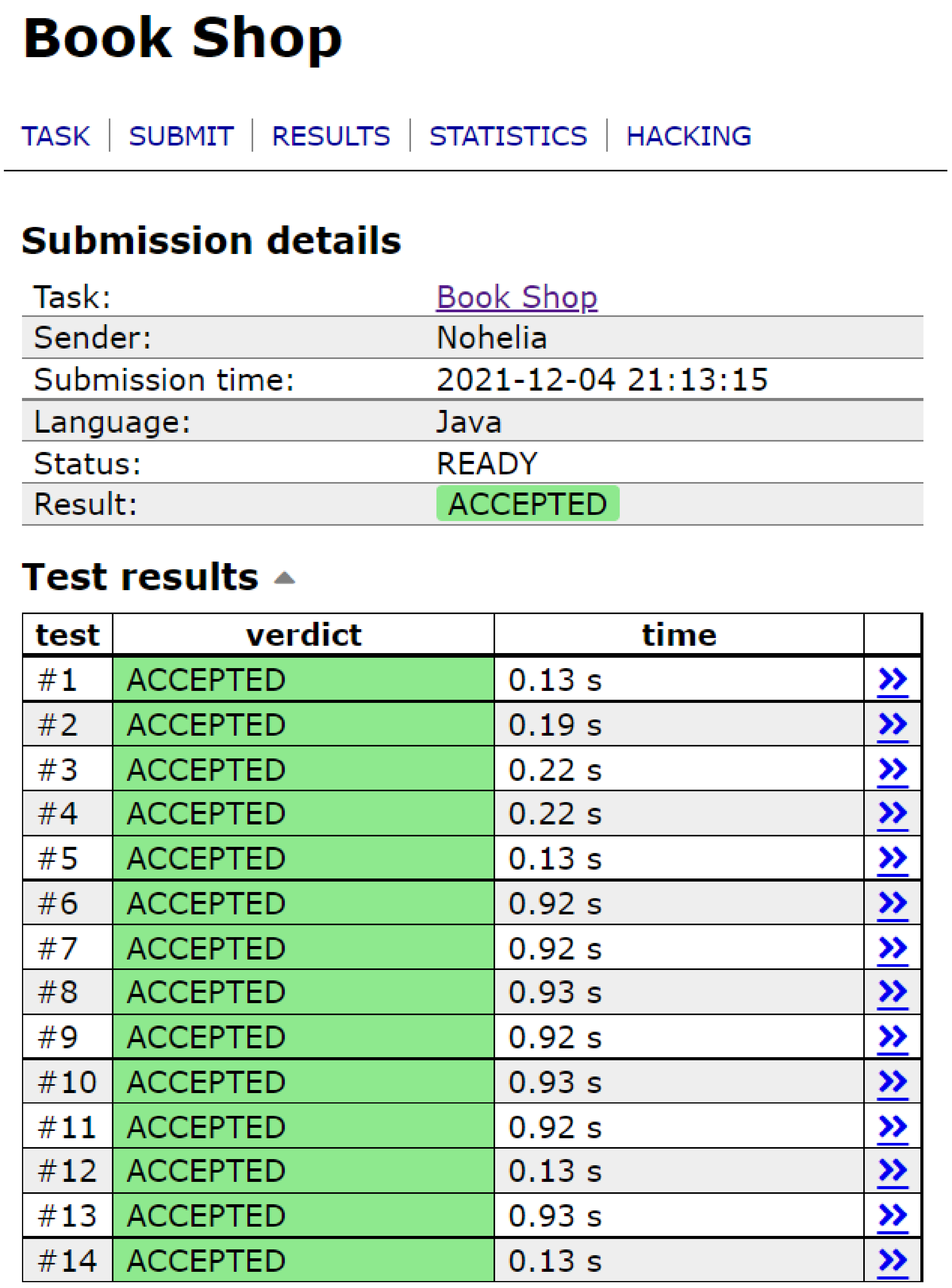
Unique Paths

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
12/03/2021 23:39	Accepted	0 ms	38.1 MB	java

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

        int height = obstacleGrid.length;
        int width = obstacleGrid[0].length;

        int[][] paths = new int[height][width];

        // primera columna
        for (int i = 0; i < height; i++) {
            if (obstacleGrid[i][0] != 1) {
                paths[i][0] = 1;
            } else {
                break;
            }
        }

        // primera fila
        for (int j = 0; j < width; j++) {
            if (obstacleGrid[0][j] != 1) {
                paths[0][j] = 1;
            } else {
                break;
            }
        }

        // espacios diferentes a la primera fila o columna
        for (int i = 1; i < height; i++) {
            for (int j = 1; j < width; j++) {
                if (obstacleGrid[i][j] != 1) {
                    paths[i][j] = paths[i - 1][j] + paths[i][j - 1];
                }
            }
        }

        return paths[height - 1][width - 1];
    }
}
```

Ejercicio 02 - Book Shop



Book Shop

TASK | SUBMIT | RESULTS | STATISTICS | HACKING

Submission details

Task:	Book Shop
Sender:	Nohelia
Submission time:	2021-12-04 21:13:15
Language:	Java
Status:	READY
Result:	ACCEPTED

Test results

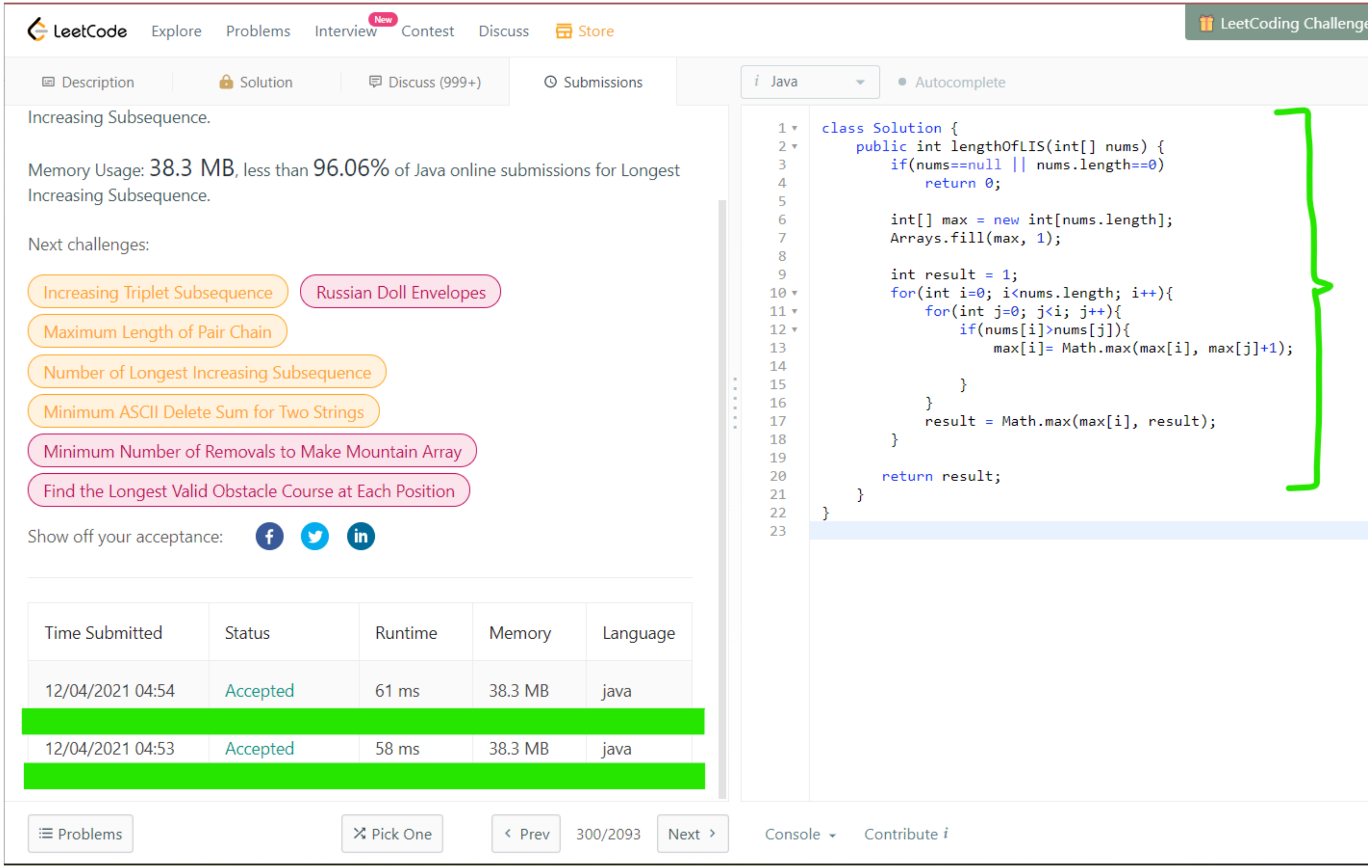
test	verdict	time	
#1	ACCEPTED	0.13 s	»
#2	ACCEPTED	0.19 s	»
#3	ACCEPTED	0.22 s	»
#4	ACCEPTED	0.22 s	»
#5	ACCEPTED	0.13 s	»
#6	ACCEPTED	0.92 s	»
#7	ACCEPTED	0.92 s	»
#8	ACCEPTED	0.93 s	»
#9	ACCEPTED	0.92 s	»
#10	ACCEPTED	0.93 s	»
#11	ACCEPTED	0.92 s	»
#12	ACCEPTED	0.13 s	»
#13	ACCEPTED	0.93 s	»
#14	ACCEPTED	0.13 s	»

Ejercicio 03 - Number of Longest Increasing Subsequence

Given an integer array nums, return the length of the longest strictly increasing subsequence.

A subsequence is a sequence that can be derived from an array by deleting some or no elements without changing the order of the remaining elements.

For example, [3,6,2,7] is a subsequence of the array [0,3,1,6,2,2,7]



Increasing Subsequence.

Memory Usage: 38.3 MB, less than 96.06% of Java online submissions for Longest Increasing Subsequence.

Next challenges:

Increasing Triplet Subsequence
Russian Doll Envelopes
Maximum Length of Pair Chain
Number of Longest Increasing Subsequence
Minimum ASCII Delete Sum for Two Strings
Minimum Number of Removals to Make Mountain Array
Find the Longest Valid Obstacle Course at Each Position

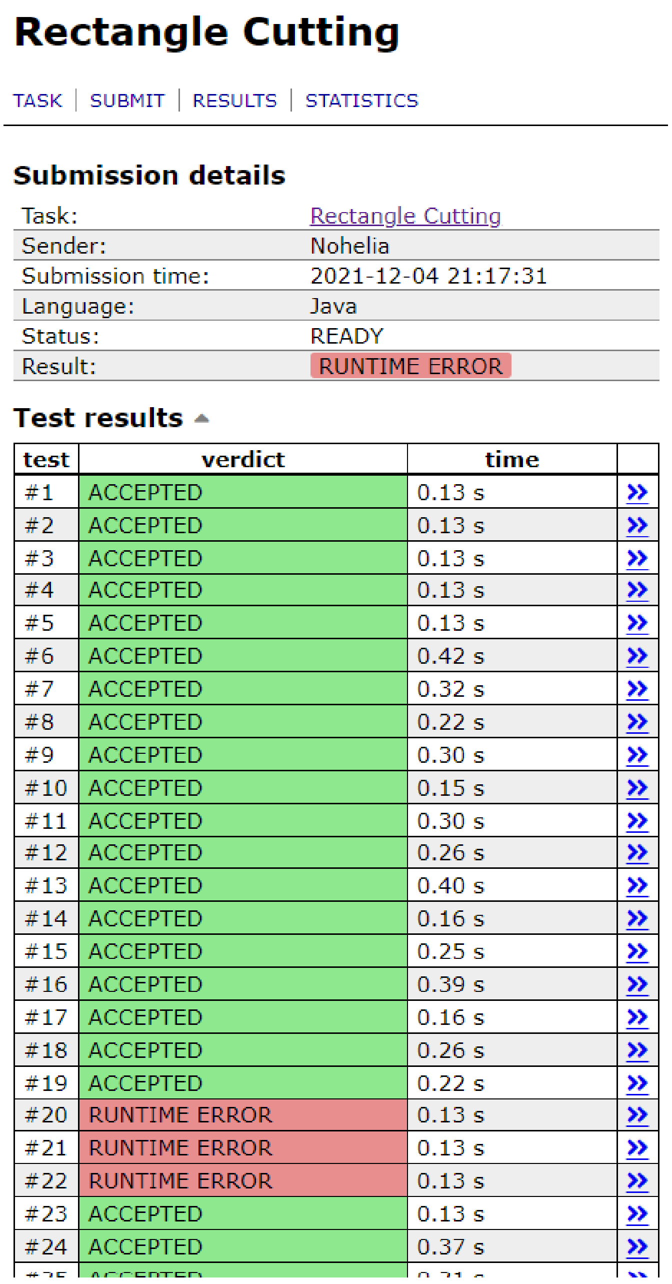
Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
12/04/2021 04:54	Accepted	61 ms	38.3 MB	java
12/04/2021 04:53	Accepted	58 ms	38.3 MB	java

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

        int[] max = new int[nums.length];
        Arrays.fill(max, 1);

        int result = 1;
        for(int i=0; i<nums.length; i++){
            for(int j=0; j<i; j++){
                if(nums[i]>nums[j]){
                    max[i]= Math.max(max[i], max[j]+1);

                }
            }
            result = Math.max(max[i], result);
        }

       return result;
    }
}
```

Ejercicio 04 - RectangleCutting

Given an a×b rectangle, your task is to cut it into squares. On each move you can select a rectangle and cut it into two rectangles in such a way that all side lengths remain integers. What is the minimum possible number of moves?



Rectangle Cutting

TASK | SUBMIT | RESULTS | STATISTICS

Submission details

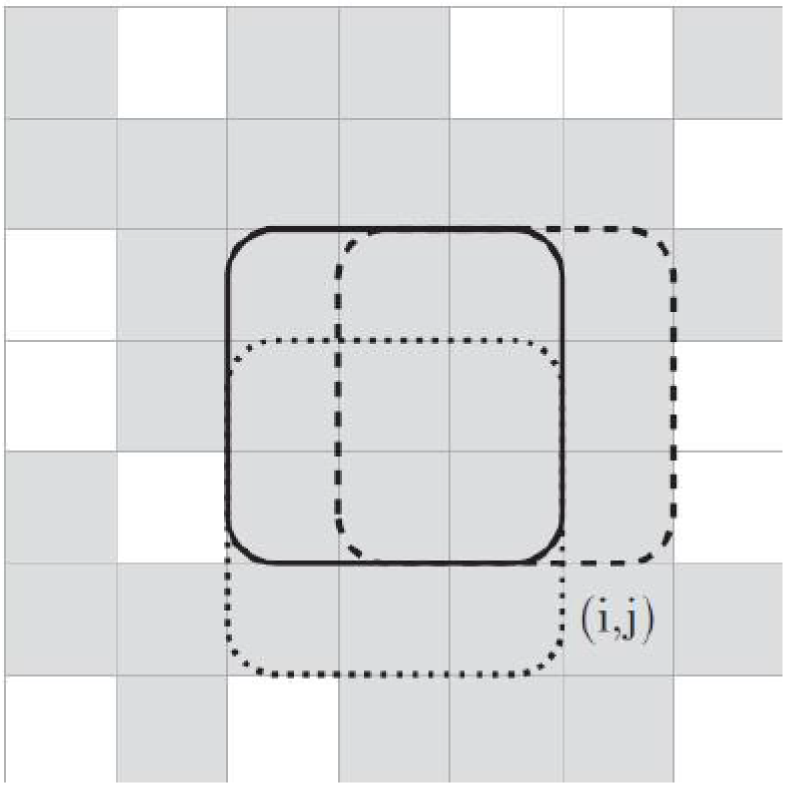
Task:	Rectangle Cutting
Sender:	Nohelia
Submission time:	2021-12-04 21:17:31
Language:	Java
Status:	READY
Result:	RUNTIME ERROR

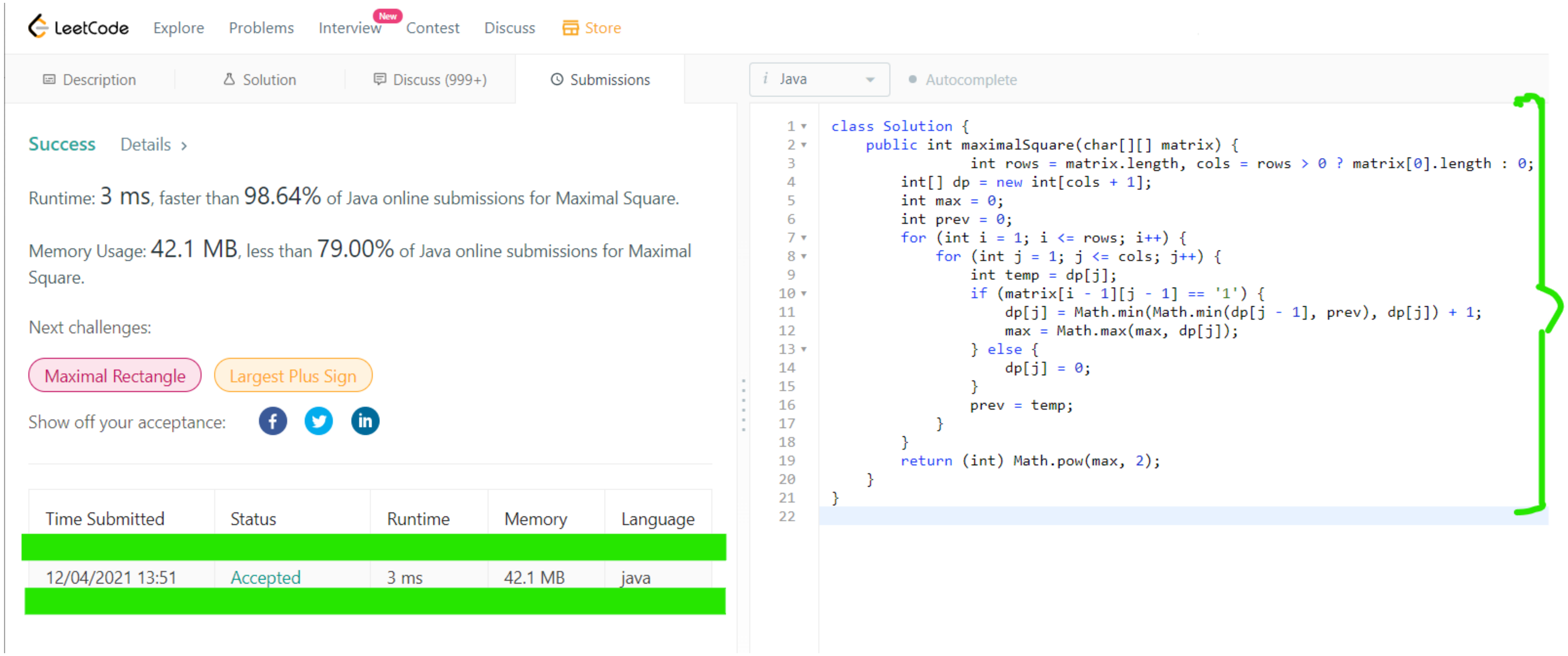
Test results

test	verdict	time	
#1	ACCEPTED	0.13 s	»
#2	ACCEPTED	0.13 s	»
#3	ACCEPTED	0.13 s	»
#4	ACCEPTED	0.13 s	»
#5	ACCEPTED	0.13 s	»
#6	ACCEPTED	0.42 s	»
#7	ACCEPTED	0.32 s	»
#8	ACCEPTED	0.22 s	»
#9	ACCEPTED	0.30 s	»
#10	ACCEPTED	0.15 s	»
#11	ACCEPTED	0.30 s	»
#12	ACCEPTED	0.26 s	»
#13	ACCEPTED	0.40 s	»
#14	ACCEPTED	0.16 s	»
#15	ACCEPTED	0.25 s	»
#16	ACCEPTED	0.39 s	»
#17	ACCEPTED	0.16 s	»
#18	ACCEPTED	0.26 s	»
#19	ACCEPTED	0.22 s	»
#20	RUNTIME ERROR	0.13 s	»
#21	RUNTIME ERROR	0.13 s	»
#22	RUNTIME ERROR	0.13 s	»
#23	ACCEPTED	0.13 s	»
#24	ACCEPTED	0.37 s	»

Ejercicio 05 - MaximalSquare

Given an m x n binary matrix filled with 0's and 1's, find the largest square containing only 1's and return its area.





Success Details ›

Runtime: 3 ms, faster than 98.64% of Java online submissions for Maximal Square.

Memory Usage: 42.1 MB, less than 79.00% of Java online submissions for Maximal Square.

Next challenges:

Maximal Rectangle Largest Plus Sign

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
12/04/2021 13:51	Accepted	3 ms	42.1 MB	java

```java
class Solution {
    public int maximalSquare(char[][] matrix) {
            int rows = matrix.length, cols = rows > 0 ? matrix[0].length : 0;
        int[] dp = new int[cols + 1];
        int max = 0;
        int prev = 0;
        for (int i = 1; i <= rows; i++) {
            for (int j = 1; j <= cols; j++) {
                int temp = dp[j];
                if (matrix[i - 1][j - 1] == '1') {
                    dp[j] = Math.min(Math.min(dp[j - 1], prev), dp[j]) + 1;
                    max = Math.max(max, dp[j]);
                } else {
                    dp[j] = 0;
                }
                prev = temp;
            }
        }
        return (int) Math.pow(max, 2);
    }
}
```