The goal is to transform your char** (containing the map -- left below) into a int** (right below).

```
......  000000
......  000000
......  000000
....x.  000011
......  000011
......  000011
```

Other example:

```
............  000000000000
.......x....  000000011111
............  000000011111
....x...x...  000011123333
......x.....  000011234444
............  000011234444
```

The numbers represent the number of obstacles from the top left corner.
Here is how to get the number in red in the following map:

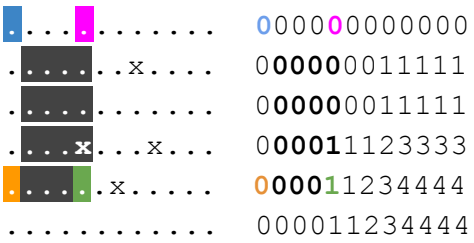
```
............  000000000000
.......x....  000000011111
............  000000011111
....x...x...  000011123333
......x.....  00001112
............
```

The operation: 1 + 1 - 1 + 1 = 2

The last +1 is because the position we're at contains an obstacle.

Once you built that, you just browse the array of numbers from the top left corner with a square of size 1x1. If it fits, you increase the size of the square. For instance, at the position (0, 0), you should find a square of size 4. You save that position and size and you will then try with a square of size 5, it doesn't fit, so you're going to the next position and try with a square of size 5 until you're at the end of the map.

It’s just like a normal brute force, the only difference is the way you get the number of obstacles (you don’t check for an obstacle at every position), you just make the following operation (for a square of **size 4** at position **(1,1)**):




1 - 0 - 0 + 0 = 1

If the result is >0, it means there is an obstacle in your square, and therefore that the square doesn’t fit.

Just look at the accepted answer on this stack overflow question to understand the operations:

https://stackoverflow.com/questions/20335427/most-efficient-algorithm-to-find-the-biggest-square-in-a-two-dimension-map

Don’t forget to upvote the question! :)

```
Count(D) = Count(A ∪ B ∪ C ∪ D) - Count(A ∪ C) - Count(A ∪ B) + Count(A)
```

Good luck, once you understand this algorithm, it is very easy to code and you’ll have incredible performance.