Range Addition II

Given an m * n matrix **M** initialized with all **0**'s and several update operations.

Operations are represented by a 2D array, and each operation is represented by an array with two **positive** integers **a** and **b**, which means **M[i][j]** should be **added by one** for all **0 and 0 .**

You need to count and return the number of maximum integers in the matrix after performing all the operations.

Example 1:

```
Input:
m = 3, n = 3
operations = [[2,2],[3,3]]
Output: 4
Explanation:
Initially, M =
[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]

After performing [2,2], M =
[[1, 1, 0],
 [1, 1, 0],
 [0, 0, 0]]

After performing [3,3], M =
[[2, 2, 1],
 [2, 2, 1],
 [1, 1, 1]]

So the maximum integer in M is 2, and there are four of it in M. So return 4.
```

Note:

1. The range of m and n is [1,40000].
2. The range of a is [1,m], and the range of b is [1,n].
3. The range of operations size won't exceed 10,000.

Solution 1

```
public class Solution {
    public int maxCount(int m, int n, int[][] ops) {
        if (ops == null || ops.length == 0) {
            return m * n;
        }

        int row = Integer.MAX_VALUE, col = Integer.MAX_VALUE;
        for(int[] op : ops) {
            row = Math.min(row, op[0]);
            col = Math.min(col, op[1]);
        }

        return row * col;
    }
}
```

written by shawngao original link here

Solution 2

C++

```
class Solution {
public:
    int maxCount(int m, int n, vector<vector<int>>& ops) {
        for (auto op : ops) {
//          if (op[0] == 0 || op[1] == 0)  continue;
            m = min(op[0], m);
            n = min(op[1], n);
        }
        return m * n;
    }
};
```

Java

```
public class Solution {
    public int maxCount(int m, int n, int[][] ops) {
        for (int[] op : ops) {
            m = Math.min(op[0], m);
            n = Math.min(op[1], n);
        }
        return m * n;
    }
}
```

Because there is no test case for operation contains 0. Without the sanity check it still pass. In fact the test implementation behind is not checking this and returns 0 for customized test case like:

```
3
3
[[2,2],[3,3], [3,0]]
```

Update

Thanks @sqfan pointing out this check is not needed:

```
//          if (op[0] == 0 || op[1] == 0)  continue;
```

written by alexander original link here

Solution 3

Say the operations are `[(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)]`. The top left square is clearly incremented by every operation. If some square `(x, y)` has `x >= x_i`, then it will not be marked by operation `i`. So all squares `(x, y)` with `x >= min_i(x_i)` do not get marked.

Thus, when there is atleast one operation, all squares `(x, y)` with `0 <= x < min(x_1, x_2, ..., x_n) and 0 <= y < min(y_1, y_2, ..., y_n)` get marked; and there are `min_i(x_i) * min_i(y_i)` of them. If there are no operations, then what is marked is the entire board.

```
def maxCount(self, R, C, ops):
    if not ops: return R * C
    X, Y = zip(*ops)
    return min(X) * min(Y)
```

written by awice original link here