Problem 598: Range Addition II

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given an

m x n

matrix

M

initialized with all

0

's and an array of operations

ops

, where

ops[i] = [a

i

, b

i

]

means

M[x][y]

should be incremented by one for all

0 <= x < a

i

and

0 <= y < b

i

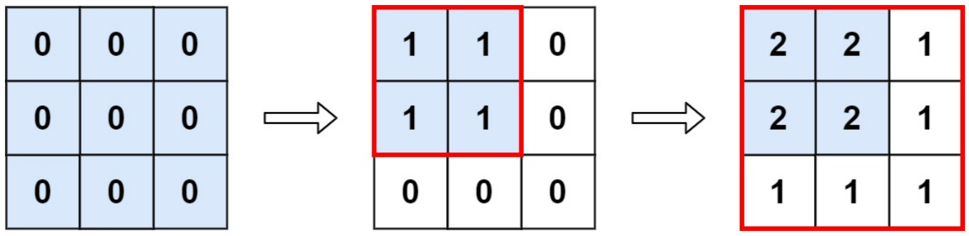
.

Count and return

the number of maximum integers in the matrix after performing all the operations

.

Example 1:



Input:

m = 3, n = 3, ops = [[2,2],[3,3]]

Output:

4

Explanation:

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

Example 2:

Input:

m = 3, n = 3, ops = [[2,2],[3,3],[3,3],[3,3],[2,2],[3,3],[3,3],[3,3],[2,2],[3,3],[3,3],[3,3]]

Output:

4

Example 3:

Input:

m = 3, n = 3, ops = []

Output:

9

Constraints:

1 <= m, n <= 4 * 10

4

0 <= ops.length <= 10

4

ops[i].length == 2

1 <= a

i

<= m

1 <= b

i

<= n

Code Snippets

C++:

```
class Solution {
public:
int maxCount(int m, int n, vector<vector<int>>& ops) {

}
};
```

Java:

```
class Solution {
public int maxCount(int m, int n, int[][] ops) {

}
}
```

Python3:

```
class Solution:
def maxCount(self, m: int, n: int, ops: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def maxCount(self, m, n, ops):
```

```
"""
:type m: int
:type n: int
:type ops: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} m
* @param {number} n
* @param {number[][]} ops
* @return {number}
*/
var maxCount = function(m, n, ops) {

};
```

TypeScript:

```
function maxCount(m: number, n: number, ops: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaxCount(int m, int n, int[][] ops) {

}
}
```

C:

```
int maxCount(int m, int n, int** ops, int opsSize, int* opsColSize) {

}
```

Go:

```
func maxCount(m int, n int, ops [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maxCount(m: Int, n: Int, ops: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func maxCount(_ m: Int, _ n: Int, _ ops: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_count(m: i32, n: i32, ops: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} m
# @param {Integer} n
# @param {Integer[][]} ops
# @return {Integer}
def max_count(m, n, ops)

end
```

PHP:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param Integer[][] $ops
* @return Integer
*/
function maxCount($m, $n, $ops) {

}
}
```

Dart:

```
class Solution {
int maxCount(int m, int n, List<List<int>> ops) {

}
}
```

Scala:

```
object Solution {
def maxCount(m: Int, n: Int, ops: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_count(m :: integer, n :: integer, ops :: [[integer]]) :: integer
def max_count(m, n, ops) do

end
end
```

Erlang:

```
-spec max_count(M :: integer(), N :: integer(), Ops :: [[integer()]]) ->
integer().
max_count(M, N, Ops) ->
.
```

Racket:

```
(define/contract (max-count m n ops)
(-> exact-integer? exact-integer? (listof (listof exact-integer?))
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Range Addition II
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int maxCount(int m, int n, vector<vector<int>>& ops) {

}
};
```

Java Solution:

```
/**
* Problem: Range Addition II
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
```

```
public int maxCount(int m, int n, int[][] ops) {

}
}
```

Python3 Solution:

```
"""
Problem: Range Addition II
Difficulty: Easy
Tags: array, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def maxCount(self, m: int, n: int, ops: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxCount(self, m, n, ops):
"""
:type m: int
:type n: int
:type ops: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Range Addition II
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} m
* @param {number} n
* @param {number[][]} ops
* @return {number}
*/
var maxCount = function(m, n, ops) {

};
```

TypeScript Solution:

```
/**
* Problem: Range Addition II
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function maxCount(m: number, n: number, ops: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Range Addition II
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
public class Solution {
public int MaxCount(int m, int n, int[][] ops) {

}
}
```

C Solution:

```
/*
* Problem: Range Addition II
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int maxCount(int m, int n, int** ops, int opsSize, int* opsColSize) {

}
```

Go Solution:

```
// Problem: Range Addition II
// Difficulty: Easy
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func maxCount(m int, n int, ops [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxCount(m: Int, n: Int, ops: Array<IntArray>): Int {
}
}
```

Swift Solution:

```
class Solution {
func maxCount(_ m: Int, _ n: Int, _ ops: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Range Addition II
// Difficulty: Easy
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn max_count(m: i32, n: i32, ops: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} m
# @param {Integer} n
# @param {Integer[][]} ops
# @return {Integer}
def max_count(m, n, ops)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param Integer[][] $ops
* @return Integer
*/
function maxCount($m, $n, $ops) {

}
}
```

Dart Solution:

```
class Solution {
int maxCount(int m, int n, List<List<int>> ops) {

}
}
```

Scala Solution:

```
object Solution {
def maxCount(m: Int, n: Int, ops: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_count(m :: integer, n :: integer, ops :: [[integer]]) :: integer
def max_count(m, n, ops) do

end
end
```

Erlang Solution:

```
-spec max_count(M :: integer(), N :: integer(), Ops :: [[integer()]]) ->
integer().
```

```
max_count(M, N, Ops) ->
.
```

Racket Solution:

```
(define/contract (max-count m n ops)
(-> exact-integer? exact-integer? (listof (listof exact-integer?))
exact-integer?)
)
```