Problem 1252: Cells with Odd Values in a Matrix

Problem Information

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

Problem Description

There is an

m x n

matrix that is initialized to all

0

's. There is also a 2D array

indices

where each

indices[i] = [r

i

, c

i

]

represents a

0-indexed location

to perform some increment operations on the matrix.

For each location

indices[i]

, do

both

of the following:

Increment

all

the cells on row

r

i

.

Increment

all

the cells on column

c

i

.

Given

m

,

n

, and

indices

, return

the

number of odd-valued cells

in the matrix after applying the increment to all locations in

indices

.

Example 1:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 3 & 1 \\ 1 & 3 & 1 \end{bmatrix}$$

Input:

m = 2, n = 3, indices = [[0,1],[1,1]]

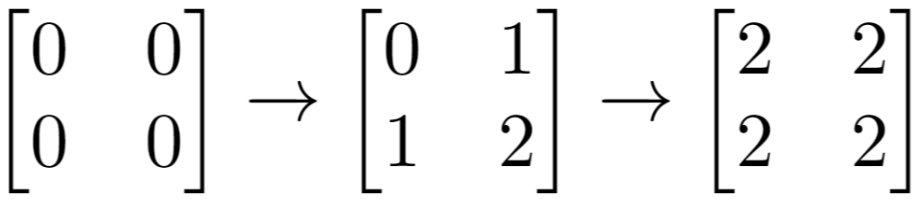
Output:

6

Explanation:

Initial matrix = [[0,0,0],[0,0,0]]. After applying first increment it becomes [[1,2,1],[0,1,0]]. The final matrix is [[1,3,1],[1,3,1]], which contains 6 odd numbers.

Example 2:



Input:

m = 2, n = 2, indices = [[1,1],[0,0]]

Output:

0

Explanation:

Final matrix = [[2,2],[2,2]]. There are no odd numbers in the final matrix.

Constraints:

1 <= m, n <= 50

1 <= indices.length <= 100

0 <= r

i

< m

0 <= c

i

< n

Follow up:

Could you solve this in

O(n + m + indices.length)

time with only

O(n + m)

extra space?

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int oddCells(int m, int n, int[][] indices) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function oddCells(m: number, n: number, indices: number[][]): number {

};
```

C#:

```
public class Solution {
public int OddCells(int m, int n, int[][] indices) {

}
}
```

C:

```
int oddCells(int m, int n, int** indices, int indicesSize, int*
indicesColSize) {

}
```

Go:

```
func oddCells(m int, n int, indices [][]int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer $m
* @param Integer $n
* @param Integer[][] $indices
* @return Integer
*/
function oddCells($m, $n, $indices) {

}
}
```

Dart:

```
class Solution {
int oddCells(int m, int n, List<List<int>> indices) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec odd_cells(M :: integer(), N :: integer(), Indices :: [[integer()]]) ->
integer().
odd_cells(M, N, Indices) ->
.
```

Racket:

```
(define/contract (odd-cells m n indices)
(-> exact-integer? exact-integer? (listof (listof exact-integer?))
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Cells with Odd Values in a Matrix
* 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 oddCells(int m, int n, vector<vector<int>>& indices) {

}
};
```

Java Solution:

```
/**
* Problem: Cells with Odd Values in a Matrix
* 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 oddCells(int m, int n, int[][] indices) {

}
}
```

Python3 Solution:

```
"""
Problem: Cells with Odd Values in a Matrix
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 oddCells(self, m: int, n: int, indices: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Cells with Odd Values in a Matrix
* 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[][]} indices
* @return {number}
*/
var oddCells = function(m, n, indices) {

};
```

TypeScript Solution:

```
/**
* Problem: Cells with Odd Values in a Matrix
* 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 oddCells(m: number, n: number, indices: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Cells with Odd Values in a Matrix
* 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 OddCells(int m, int n, int[][] indices) {

}
}
```

C Solution:

```
/*
* Problem: Cells with Odd Values in a Matrix
* 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 oddCells(int m, int n, int** indices, int indicesSize, int*
indicesColSize) {

}
```

Go Solution:

```
// Problem: Cells with Odd Values in a Matrix
// 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 oddCells(m int, n int, indices [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Cells with Odd Values in a Matrix
// 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 odd_cells(m: i32, n: i32, indices: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int oddCells(int m, int n, List<List<int>> indices) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec odd_cells(M :: integer(), N :: integer(), Indices :: [[integer()]]) ->
integer().
odd_cells(M, N, Indices) ->
.
```

Racket Solution:

```
(define/contract (odd-cells m n indices)
(-> exact-integer? exact-integer? (listof (listof exact-integer?))
exact-integer?)
)
```