Problem 2768: Number of Black Blocks

Problem Information

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

Problem Description

You are given two integers

m

and

n

representing the dimensions of a

0-indexed

m x n

grid.

You are also given a

0-indexed

2D integer matrix

coordinates

, where

coordinates[i] = [x, y]

indicates that the cell with coordinates

[x, y]

is colored

black

. All cells in the grid that do not appear in

coordinates

are

white

.

A block is defined as a

2 x 2

submatrix of the grid. More formally, a block with cell

[x, y]

as its top-left corner where

0 <= x < m - 1

and

0 <= y < n - 1

contains the coordinates

[x, y]

,

[x + 1, y]

,

[x, y + 1]

, and

[x + 1, y + 1]

.

Return

a

0-indexed

integer array

arr

of size

5

such that

arr[i]

is the number of blocks that contains exactly

i

black

cells

.

Example 1:

Input:

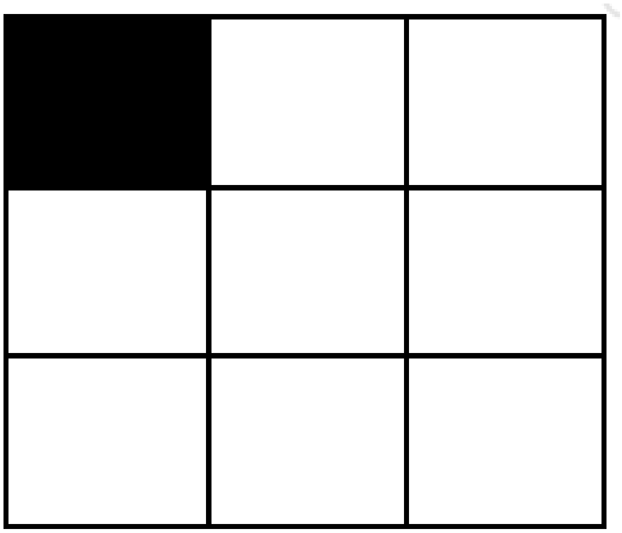
m = 3, n = 3, coordinates = [[0,0]]

Output:

[3,1,0,0,0]

Explanation:

The grid looks like this:



There is only 1 block with one black cell, and it is the block starting with cell [0,0]. The other 3 blocks start with cells [0,1], [1,0] and [1,1]. They all have zero black cells. Thus, we return [3,1,0,0,0].

Example 2:

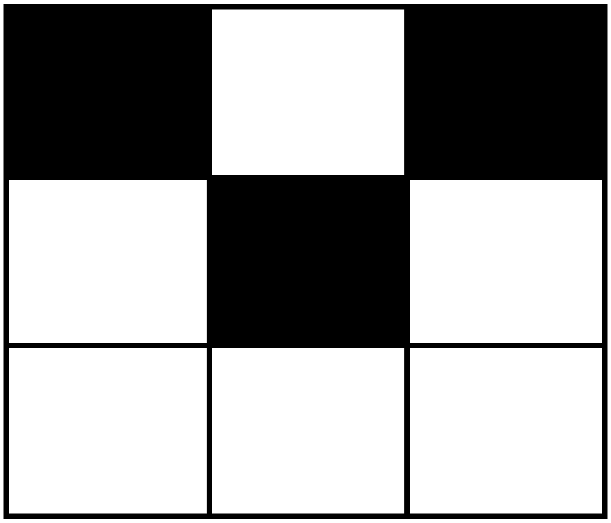
Input:

m = 3, n = 3, coordinates = [[0,0],[1,1],[0,2]] Output:

[0,2,2,0,0]

Explanation:

The grid looks like this:



There are 2 blocks with two black cells (the ones starting with cell coordinates [0,0] and [0,1]). The other 2 blocks have starting cell coordinates of [1,0] and [1,1]. They both have 1 black cell. Therefore, we return [0,2,2,0,0].

Constraints:

2 <= m <= 10

5

2 <= n <= 10

5

0 <= coordinates.length <= 10

4

coordinates[i].length == 2

0 <= coordinates[i][0] < m

0 <= coordinates[i][1] < n

It is guaranteed that

coordinates

contains pairwise distinct coordinates.

Code Snippets

C++:

```
class Solution {
public:
vector<long long> countBlackBlocks(int m, int n, vector<vector<int>>&
coordinates) {

}
};
```

Java:

```
class Solution {
public long[] countBlackBlocks(int m, int n, int[][] coordinates) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def countBlackBlocks(self, m, n, coordinates):
```

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

JavaScript:

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

};
```

TypeScript:

```
function countBlackBlocks(m: number, n: number, coordinates: number[][]):
number[] {

};
```

C#:

```
public class Solution {
public long[] CountBlackBlocks(int m, int n, int[][] coordinates) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* countBlackBlocks(int m, int n, int** coordinates, int
coordinatesSize, int* coordinatesColSize, int* returnSize) {
}
```

Go:

```
func countBlackBlocks(m int, n int, coordinates [][]int) []int64 {

}
```

Kotlin:

```
class Solution {
fun countBlackBlocks(m: Int, n: Int, coordinates: Array<IntArray>): LongArray
{

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn count_black_blocks(m: i32, n: i32, coordinates: Vec<Vec<i32>>) ->
Vec<i64> {

}
}
```

Ruby:

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

```

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<int> countBlackBlocks(int m, int n, List<List<int>> coordinates) {

}
}
```

Scala:

```
object Solution {
def countBlackBlocks(m: Int, n: Int, coordinates: Array[Array[Int]]):
Array[Long] = {

}
}
```

Elixir:

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

end
```

```
end
```

Erlang:

```
-spec count_black_blocks(M :: integer(), N :: integer(), Coordinates ::
[[integer()]]) -> [integer()].
count_black_blocks(M, N, Coordinates) ->
.
```

Racket:

```
(define/contract (count-black-blocks m n coordinates)
(-> exact-integer? exact-integer? (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Black Blocks
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<long long> countBlackBlocks(int m, int n, vector<vector<int>>&
coordinates) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Black Blocks
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public long[] countBlackBlocks(int m, int n, int[][] coordinates) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Black Blocks
Difficulty: Medium
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

```
class Solution(object):
def countBlackBlocks(self, m, n, coordinates):
"""
:type m: int
:type n: int
:type coordinates: List[List[int]]
```

```
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Black Blocks
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Number of Black Blocks
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function countBlackBlocks(m: number, n: number, coordinates: number[][]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Number of Black Blocks
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public long[] CountBlackBlocks(int m, int n, int[][] coordinates) {

}
}
```

C Solution:

```
/*
* Problem: Number of Black Blocks
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* countBlackBlocks(int m, int n, int** coordinates, int
coordinatesSize, int* coordinatesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Number of Black Blocks
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func countBlackBlocks(m int, n int, coordinates [][]int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countBlackBlocks(m: Int, n: Int, coordinates: Array<IntArray>): LongArray
{

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Black Blocks
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn count_black_blocks(m: i32, n: i32, coordinates: Vec<Vec<i32>>) ->
Vec<i64> {
}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<int> countBlackBlocks(int m, int n, List<List<int>> coordinates) {

}
}
```

Scala Solution:

```
object Solution {
def countBlackBlocks(m: Int, n: Int, coordinates: Array[Array[Int]]):
Array[Long] = {
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec count_black_blocks(M :: integer(), N :: integer(), Coordinates ::
[[integer()]]) -> [integer()].
count_black_blocks(M, N, Coordinates) ->
.
```

Racket Solution:

```
(define/contract (count-black-blocks m n coordinates)
(-> exact-integer? exact-integer? (listof (listof exact-integer?)) (listof
exact-integer?))
)
```