Problem 2257: Count Unguarded Cells in the Grid

Problem Information

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

Problem Description

You are given two integers

m

and

n

representing a

0-indexed

m x n

grid. You are also given two 2D integer arrays

guards

and

walls

where

guards[i] = [row

i

, col

i

]

and

walls[j] = [row

j

, col

j

]

represent the positions of the

i

th

guard and

j

th

wall respectively.

A guard can see

every

cell in the four cardinal directions (north, east, south, or west) starting from their position unless

obstructed

by a wall or another guard. A cell is

guarded

if there is

at least

one guard that can see it.

Return

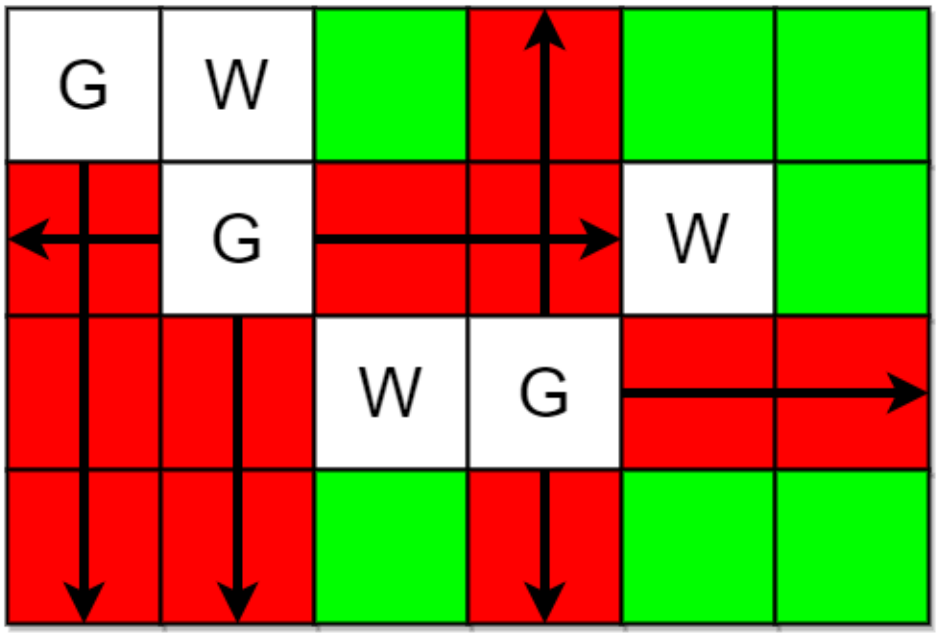
the number of unoccupied cells that are

not

guarded

.

Example 1:



Input:

m = 4, n = 6, guards = [[0,0],[1,1],[2,3]], walls = [[0,1],[2,2],[1,4]]

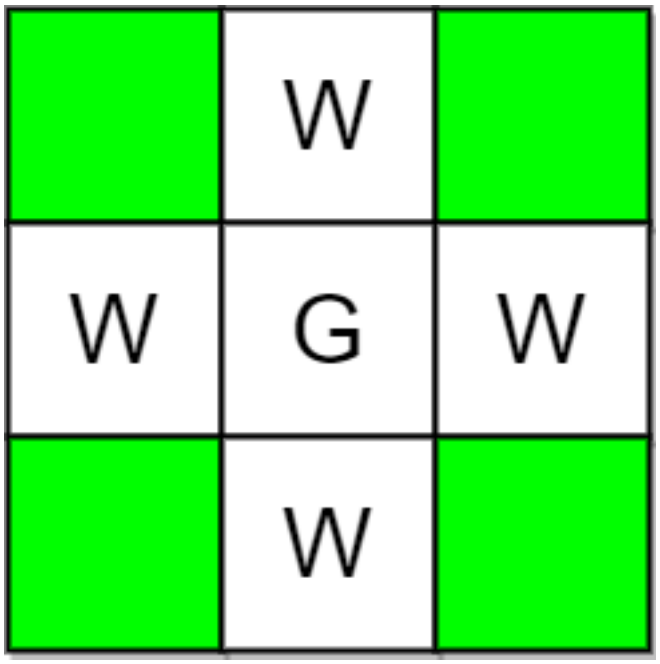
Output:

7

Explanation:

The guarded and unguarded cells are shown in red and green respectively in the above diagram. There are a total of 7 unguarded cells, so we return 7.

Example 2:



Input:

m = 3, n = 3, guards = [[1,1]], walls = [[0,1],[1,0],[2,1],[1,2]]

Output:

4

Explanation:

The unguarded cells are shown in green in the above diagram. There are a total of 4 unguarded cells, so we return 4.

Constraints:

1 <= m, n <= 10

5

2 <= m * n <= 10

5

1 <= guards.length, walls.length <= 5 * 10

4

2 <= guards.length + walls.length <= m * n

guards[i].length == walls[j].length == 2

0 <= row

i

, row

j

< m

0 <= col

i

, col

j

< n

All the positions in

guards

and

walls

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int countUnguarded(int m, int n, vector<vector<int>>& guards,
vector<vector<int>>& walls) {

}
};
```

Java:

```
class Solution {
public int countUnguarded(int m, int n, int[][] guards, int[][] walls) {

}
}
```

Python3:

```
class Solution:
def countUnguarded(self, m: int, n: int, guards: List[List[int]], walls:
List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def countUnguarded(self, m, n, guards, walls):
"""
:type m: int
:type n: int
:type guards: List[List[int]]
:type walls: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} m
* @param {number} n
* @param {number[][]} guards
* @param {number[][]} walls
* @return {number}
*/
var countUnguarded = function(m, n, guards, walls) {

};
```

TypeScript:

```
function countUnguarded(m: number, n: number, guards: number[][], walls:
number[][]): number {

};
```

C#:

```
public class Solution {
public int CountUnguarded(int m, int n, int[][] guards, int[][] walls) {

}
}
```

C:

```
int countUnguarded(int m, int n, int** guards, int guardsSize, int*
guardsColSize, int** walls, int wallsSize, int* wallsColSize) {

}
```

Go:

```
func countUnguarded(m int, n int, guards [][]int, walls [][]int) int {

}
```

Kotlin:

```
class Solution {
fun countUnguarded(m: Int, n: Int, guards: Array<IntArray>, walls:
```

```
Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func countUnguarded(_ m: Int, _ n: Int, _ guards: [[Int]], _ walls: [[Int]])
-> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_unguarded(m: i32, n: i32, guards: Vec<Vec<i32>>, walls:
Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} m
# @param {Integer} n
# @param {Integer[][]} guards
# @param {Integer[][]} walls
# @return {Integer}
def count_unguarded(m, n, guards, walls)

end
```

PHP:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param Integer[][] $guards
* @param Integer[][] $walls
```

```
* @return Integer
*/
function countUnguarded($m, $n, $guards, $walls) {

}
}
```

Dart:

```
class Solution {
int countUnguarded(int m, int n, List<List<int>> guards, List<List<int>>
walls) {

}
}
```

Scala:

```
object Solution {
def countUnguarded(m: Int, n: Int, guards: Array[Array[Int]], walls:
Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_unguarded(m :: integer, n :: integer, guards :: [[integer]],
walls :: [[integer]]) :: integer
def count_unguarded(m, n, guards, walls) do

end
end
```

Erlang:

```
-spec count_unguarded(M :: integer(), N :: integer(), Guards ::
[[integer()]], Walls :: [[integer()]]) -> integer().
count_unguarded(M, N, Guards, Walls) ->
.
```

Racket:

```
(define/contract (count-unguarded m n guards walls)
(-> exact-integer? exact-integer? (listof (listof exact-integer?)) (listof
(listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Unguarded Cells in the Grid
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int countUnguarded(int m, int n, vector<vector<int>>& guards,
vector<vector<int>>& walls) {

}
};
```

Java Solution:

```
/**
* Problem: Count Unguarded Cells in the Grid
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
class Solution {
public int countUnguarded(int m, int n, int[][] guards, int[][] walls) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Unguarded Cells in the Grid
Difficulty: Medium
Tags: array, tree

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

```
class Solution(object):
def countUnguarded(self, m, n, guards, walls):
"""
:type m: int
:type n: int
:type guards: List[List[int]]
:type walls: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Unguarded Cells in the Grid
* Difficulty: Medium
```

```
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number} m
* @param {number} n
* @param {number[][]} guards
* @param {number[][]} walls
* @return {number}
*/
var countUnguarded = function(m, n, guards, walls) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Unguarded Cells in the Grid
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function countUnguarded(m: number, n: number, guards: number[][], walls:
number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Count Unguarded Cells in the Grid
* Difficulty: Medium
* Tags: array, tree
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int CountUnguarded(int m, int n, int[][] guards, int[][] walls) {

}
}
```

C Solution:

```
/*
* Problem: Count Unguarded Cells in the Grid
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int countUnguarded(int m, int n, int** guards, int guardsSize, int*
guardsColSize, int** walls, int wallsSize, int* wallsColSize) {

}
```

Go Solution:

```
// Problem: Count Unguarded Cells in the Grid
// Difficulty: Medium
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func countUnguarded(m int, n int, guards [][]int, walls [][]int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun countUnguarded(m: Int, n: Int, guards: Array<IntArray>, walls:
Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func countUnguarded(_ m: Int, _ n: Int, _ guards: [[Int]], _ walls: [[Int]])
-> Int {

}
}
```

Rust Solution:

```
// Problem: Count Unguarded Cells in the Grid
// Difficulty: Medium
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn count_unguarded(m: i32, n: i32, guards: Vec<Vec<i32>>, walls:
Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} m
# @param {Integer} n
```

```
# @param {Integer[][]} guards
# @param {Integer[][]} walls
# @return {Integer}
def count_unguarded(m, n, guards, walls)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param Integer[][] $guards
* @param Integer[][] $walls
* @return Integer
*/
function countUnguarded($m, $n, $guards, $walls) {

}
}
```

Dart Solution:

```
class Solution {
int countUnguarded(int m, int n, List<List<int>> guards, List<List<int>>
walls) {

}
}
```

Scala Solution:

```
object Solution {
def countUnguarded(m: Int, n: Int, guards: Array[Array[Int]], walls:
Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_unguarded(m :: integer, n :: integer, guards :: [[integer]],
walls :: [[integer]]) :: integer
def count_unguarded(m, n, guards, walls) do

end
end
```

Erlang Solution:

```
-spec count_unguarded(M :: integer(), N :: integer(), Guards ::
[[integer()]], Walls :: [[integer()]]) -> integer().
count_unguarded(M, N, Guards, Walls) ->
.
```

Racket Solution:

```
(define/contract (count-unguarded m n guards walls)
(-> exact-integer? exact-integer? (listof (listof exact-integer?)) (listof
(listof exact-integer?)) exact-integer?)
)
```