Problem 2061: Number of Spaces Cleaning Robot Cleaned

Problem Information

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

Problem Description

A room is represented by a

0-indexed

2D binary matrix

room

where a

0

represents an

empty

space and a

1

represents a space with an

object

. The top left corner of the room will be empty in all test cases.

A cleaning robot starts at the top left corner of the room and is facing right. The robot will continue heading straight until it reaches the edge of the room or it hits an object, after which it will turn 90 degrees

clockwise

and repeat this process. The starting space and all spaces that the robot visits are

cleaned

by it.

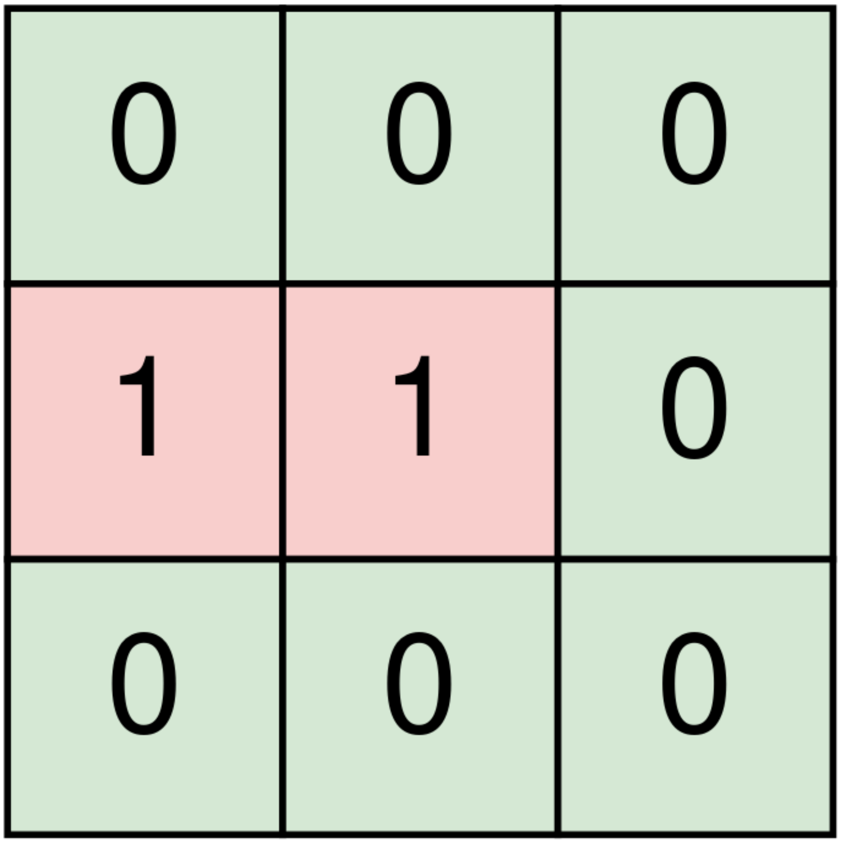
Return

the number of

clean

spaces in the room if the robot runs indefinitely.

Example 1:



0	0	0
1	1	0
0	0	0

Input:

room = [[0,0,0],[1,1,0],[0,0,0]]

Output:

7

Explanation:

The robot cleans the spaces at (0, 0), (0, 1), and (0, 2).

The robot is at the edge of the room, so it turns 90 degrees clockwise and now faces down.

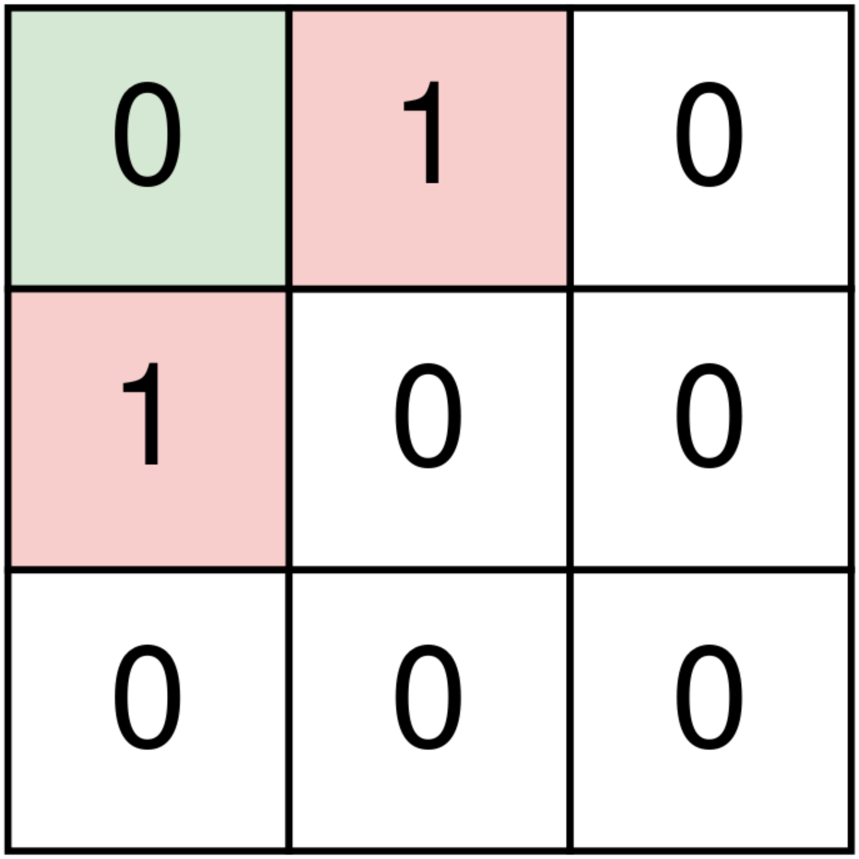
The robot cleans the spaces at (1, 2), and (2, 2).

The robot is at the edge of the room, so it turns 90 degrees clockwise and now faces left.

The robot cleans the spaces at (2, 1), and (2, 0).

The robot has cleaned all 7 empty spaces, so return 7.

Example 2:



0	1	0
1	0	0
0	0	0

Input:

room = [[0,1,0],[1,0,0],[0,0,0]]

Output:

1

Explanation:

The robot cleans the space at (0, 0).

The robot hits an object, so it turns 90 degrees clockwise and now faces down.

The robot hits an object, so it turns 90 degrees clockwise and now faces left.

The robot is at the edge of the room, so it turns 90 degrees clockwise and now faces up.

The robot is at the edge of the room, so it turns 90 degrees clockwise and now faces right.

The robot is back at its starting position.

The robot has cleaned 1 space, so return 1.

Example 3:

Input:

room = [[0,0,0],[0,0,0],[0,0,0]]

Output:

8

Constraints:

m == room.length

n == room[r].length

1 <= m, n <= 300

room[r][c]

is either

0

or

1

.

room[0][0] == 0

Code Snippets

C++:

```
class Solution {
public:
int numberOfCleanRooms(vector<vector<int>>& room) {

}
};
```

Java:

```
class Solution {
public int numberOfCleanRooms(int[][] room) {

}
}
```

Python3:

```
class Solution:
def numberOfCleanRooms(self, room: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def numberOfCleanRooms(self, room):
"""
:type room: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} room
* @return {number}
*/
var numberOfCleanRooms = function(room) {

};
```

TypeScript:

```
function numberOfCleanRooms(room: number[][]): number {

};
```

C#:

```
public class Solution {
public int NumberOfCleanRooms(int[][] room) {

}
}
```

C:

```
int numberOfCleanRooms(int** room, int roomSize, int* roomColSize) {

}
```

Go:

```
func numberOfCleanRooms(room [][]int) int {

}
```

Kotlin:

```
class Solution {
fun numberOfCleanRooms(room: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func numberOfCleanRooms(_ room: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_of_clean_rooms(room: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} room
# @return {Integer}
def number_of_clean_rooms(room)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $room
* @return Integer
*/
function numberOfCleanRooms($room) {

}
}
```

Dart:

```
class Solution {
int numberOfCleanRooms(List<List<int>> room) {

}
}
```

Scala:

```
object Solution {
def numberOfCleanRooms(room: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_clean_rooms(room :: [[integer]]) :: integer
def number_of_clean_rooms(room) do

end
end
```

Erlang:

```
-spec number_of_clean_rooms(Room :: [[integer()]]) -> integer().
number_of_clean_rooms(Room) ->
.
```

Racket:

```
(define/contract (number-of-clean-rooms room)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Spaces Cleaning Robot Cleaned
* Difficulty: Medium
* Tags: array
*
* 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 numberOfCleanRooms(vector<vector<int>>& room) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Spaces Cleaning Robot Cleaned
* Difficulty: Medium
* Tags: array
*
* 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 numberOfCleanRooms(int[][] room) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Spaces Cleaning Robot Cleaned
Difficulty: Medium
Tags: array

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 numberOfCleanRooms(self, room: List[List[int]]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def numberOfCleanRooms(self, room):
"""
:type room: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Spaces Cleaning Robot Cleaned
* Difficulty: Medium
* Tags: array
*
* 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[][]} room
* @return {number}
*/
var numberOfCleanRooms = function(room) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Spaces Cleaning Robot Cleaned
* Difficulty: Medium
* Tags: array
*
* 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 numberOfCleanRooms(room: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Spaces Cleaning Robot Cleaned
* Difficulty: Medium
* Tags: array
*
* 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 NumberOfCleanRooms(int[][] room) {

}
}
```

C Solution:

```
/*
* Problem: Number of Spaces Cleaning Robot Cleaned
* Difficulty: Medium
* Tags: array
*
* 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 numberOfCleanRooms(int** room, int roomSize, int* roomColSize) {

}
```

Go Solution:

```
// Problem: Number of Spaces Cleaning Robot Cleaned
// Difficulty: Medium
// Tags: array
//
// 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 numberOfCleanRooms(room [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfCleanRooms(room: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func numberOfCleanRooms(_ room: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Spaces Cleaning Robot Cleaned
// Difficulty: Medium
// Tags: array
//
// 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 number_of_clean_rooms(room: Vec<Vec<i32>>) -> i32 {

}

}
```

Ruby Solution:

```
# @param {Integer[][]} room
# @return {Integer}
def number_of_clean_rooms(room)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $room
* @return Integer
*/
function numberOfCleanRooms($room) {

}
}
```

Dart Solution:

```
class Solution {
int numberOfCleanRooms(List<List<int>> room) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfCleanRooms(room: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_clean_rooms(room :: [[integer]]) :: integer
def number_of_clean_rooms(room) do

end
end
```

Erlang Solution:

```
-spec number_of_clean_rooms(Room :: [[integer()]]) -> integer().
number_of_clean_rooms(Room) ->
.
```

Racket Solution:

```
(define/contract (number-of-clean-rooms room)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```