Problem 3531: Count Covered Buildings

Problem Information

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

Problem Description

You are given a positive integer

n

, representing an

n x n

city. You are also given a 2D grid

buildings

, where

buildings[i] = [x, y]

denotes a

unique

building located at coordinates

[x, y]

.

A building is

covered

if there is at least one building in all

four

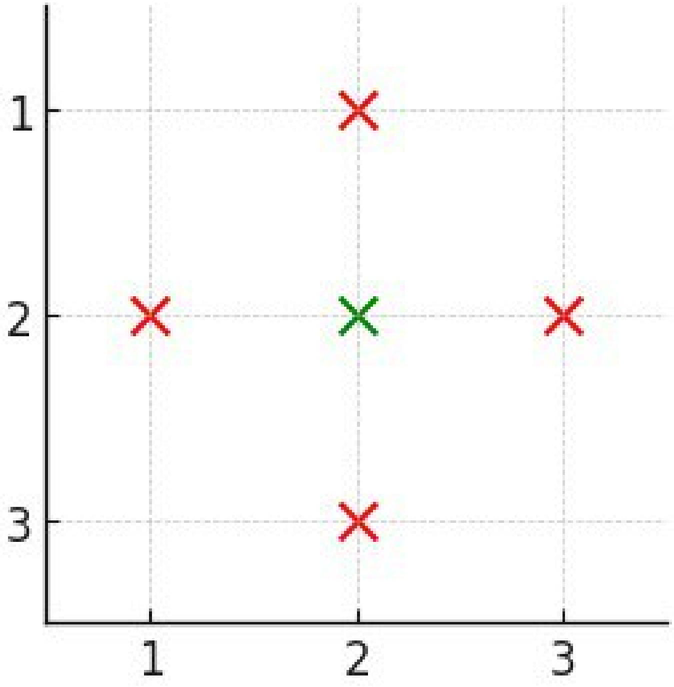
directions: left, right, above, and below.

Return the number of

covered

buildings.

Example 1:



Input:

n = 3, buildings = [[1,2],[2,2],[3,2],[2,1],[2,3]] Output:

1

Explanation:

Only building

[2,2]

is covered as it has at least one building:

above (

[1,2]

)

below (

[3,2]

)

left (

[2,1]

)

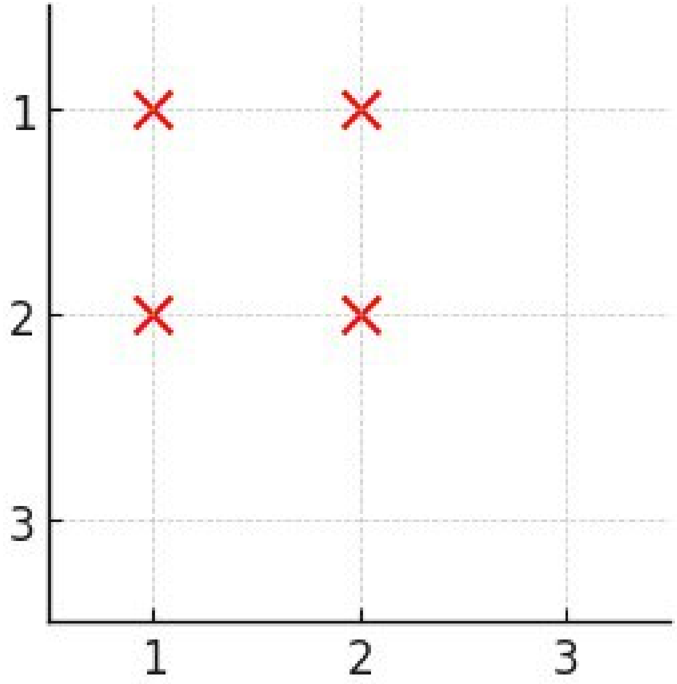
right (

[2,3]

)

Thus, the count of covered buildings is 1.

Example 2:



Input:

n = 3, buildings = [[1,1],[1,2],[2,1],[2,2]]

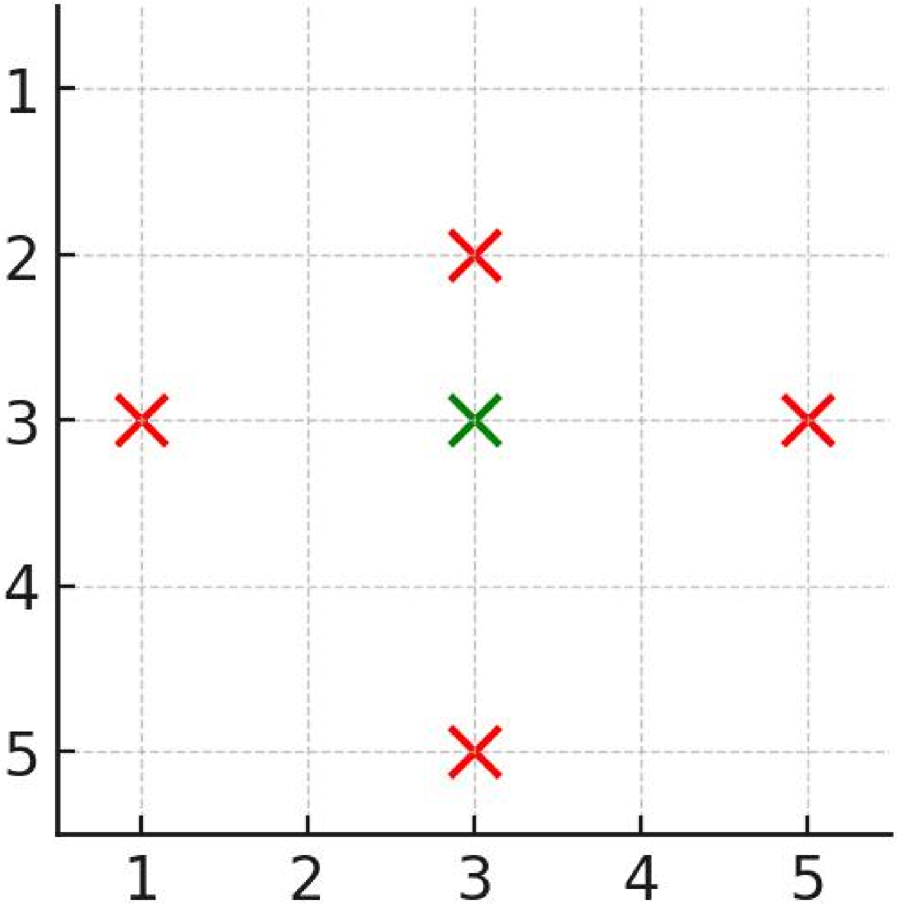
Output:

0

Explanation:

No building has at least one building in all four directions.

Example 3:



Input:

n = 5, buildings = [[1,3],[3,2],[3,3],[3,5],[5,3]]

Output:

1

Explanation:

Only building

[3,3]

is covered as it has at least one building:

above (

[1,3]

)

below (

[5,3]

)

left (

[3,2]

)

right (

[3,5]

)

Thus, the count of covered buildings is 1.

Constraints:

2 <= n <= 10

5

1 <= buildings.length <= 10

5

buildings[i] = [x, y]

1 <= x, y <= n

All coordinates of

buildings

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int countCoveredBuildings(int n, vector<vector<int>>& buildings) {

}
};
```

Java:

```
class Solution {
public int countCoveredBuildings(int n, int[][] buildings) {

}
}
```

Python3:

```
class Solution:
def countCoveredBuildings(self, n: int, buildings: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def countCoveredBuildings(self, n, buildings):
"""
:type n: int
```

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

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} buildings
* @return {number}
*/
var countCoveredBuildings = function(n, buildings) {

};
```

TypeScript:

```
function countCoveredBuildings(n: number, buildings: number[][]): number {

};
```

C#:

```
public class Solution {
public int CountCoveredBuildings(int n, int[][] buildings) {

}
}
```

C:

```
int countCoveredBuildings(int n, int** buildings, int buildingsSize, int*
buildingsColSize) {

}
```

Go:

```
func countCoveredBuildings(n int, buildings [][]int) int {

}
```

Kotlin:

```
class Solution {
fun countCoveredBuildings(n: Int, buildings: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func countCoveredBuildings(_ n: Int, _ buildings: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_covered_buildings(n: i32, buildings: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} buildings
# @return {Integer}
def count_covered_buildings(n, buildings)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $buildings
* @return Integer
*/
function countCoveredBuildings($n, $buildings) {


}
}
```

Dart:

```
class Solution {
int countCoveredBuildings(int n, List<List<int>> buildings) {

}
}
```

Scala:

```
object Solution {
def countCoveredBuildings(n: Int, buildings: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_covered_buildings(n :: integer, buildings :: [[integer]]) ::
integer
def count_covered_buildings(n, buildings) do

end
end
```

Erlang:

```
-spec count_covered_buildings(N :: integer(), Buildings :: [[integer()]]) ->
integer().
count_covered_buildings(N, Buildings) ->
.
```

Racket:

```
(define/contract (count-covered-buildings n buildings)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Covered Buildings
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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:
int countCoveredBuildings(int n, vector<vector<int>>& buildings) {

}
};
```

Java Solution:

```
/**
* Problem: Count Covered Buildings
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 int countCoveredBuildings(int n, int[][] buildings) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Covered Buildings
Difficulty: Medium
Tags: array, hash, sort

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 countCoveredBuildings(self, n: int, buildings: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countCoveredBuildings(self, n, buildings):
"""
:type n: int
:type buildings: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Covered Buildings
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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} n
* @param {number[][]} buildings
* @return {number}
*/
```

```
var countCoveredBuildings = function(n, buildings) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Covered Buildings
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 countCoveredBuildings(n: number, buildings: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Count Covered Buildings
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 int CountCoveredBuildings(int n, int[][] buildings) {

}
}
```

C Solution:

```
/*
* Problem: Count Covered Buildings
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int countCoveredBuildings(int n, int** buildings, int buildingsSize, int*
buildingsColSize) {

}
```

Go Solution:

```
// Problem: Count Covered Buildings
// Difficulty: Medium
// Tags: array, hash, sort
//
// 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 countCoveredBuildings(n int, buildings [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countCoveredBuildings(n: Int, buildings: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func countCoveredBuildings(_ n: Int, _ buildings: [[Int]]) -> Int {
}
}
```

Rust Solution:

```
// Problem: Count Covered Buildings
// Difficulty: Medium
// Tags: array, hash, sort
//
// 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_covered_buildings(n: i32, buildings: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} buildings
# @return {Integer}
def count_covered_buildings(n, buildings)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $buildings
* @return Integer
*/
function countCoveredBuildings($n, $buildings) {

}
}
```

Dart Solution:

```
class Solution {
int countCoveredBuildings(int n, List<List<int>> buildings) {

}
}
```

Scala Solution:

```
object Solution {
def countCoveredBuildings(n: Int, buildings: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_covered_buildings(n :: integer, buildings :: [[integer]]) ::
integer
def count_covered_buildings(n, buildings) do

end
end
```

Erlang Solution:

```
-spec count_covered_buildings(N :: integer(), Buildings :: [[integer()]]) ->
integer().
count_covered_buildings(N, Buildings) ->
.
```

Racket Solution:

```
(define/contract (count-covered-buildings n buildings)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```