Problem 2237: Count Positions on Street With Required Brightness

Problem Information

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

Problem Description

You are given an integer

n

. A perfectly straight street is represented by a number line ranging from

0

to

n - 1

. You are given a 2D integer array

lights

representing the street lamp(s) on the street. Each

lights[i] = [position

i

, range

i

]

indicates that there is a street lamp at position

position

i

that lights up the area from

[max(0, position

i

- range

i

), min(n - 1, position

i

+ range

i

)]

(

inclusive

).

The

brightness

of a position

p

is defined as the number of street lamps that light up the position

p

. You are given a

0-indexed

integer array

requirement

of size

n

where

requirement[i]

is the minimum

brightness

of the

i

th

position on the street.

Return

the number of positions

i

on the street between

0

and

n - 1

that have a

brightness

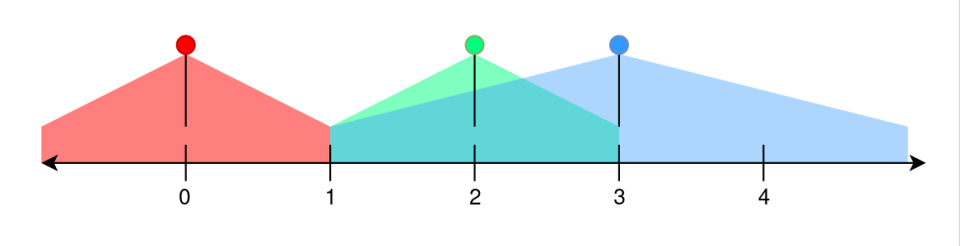
of

at least

requirement[i]

.

Example 1:



Input:

n = 5, lights = [[0,1],[2,1],[3,2]], requirement = [0,2,1,4,1]

Output:

4

Explanation:

- The first street lamp lights up the area from [max(0, 0 - 1), min(n - 1, 0 + 1)] = [0, 1] (inclusive). - The second street lamp lights up the area from [max(0, 2 - 1), min(n - 1, 2 + 1)] = [1, 3] (inclusive). - The third street lamp lights up the area from [max(0, 3 - 2), min(n - 1, 3 + 2)] = [1, 4] (inclusive).

- Position 0 is covered by the first street lamp. It is covered by 1 street lamp which is greater than requirement[0]. - Position 1 is covered by the first, second, and third street lamps. It is covered by 3 street lamps which is greater than requirement[1]. - Position 2 is covered by the second and third street lamps. It is covered by 2 street lamps which is greater than requirement[2]. - Position 3 is covered by the second and third street lamps. It is covered by 2 street lamps which is less than requirement[3]. - Position 4 is covered by the third street lamp. It is covered by 1 street lamp which is equal to requirement[4].

Positions 0, 1, 2, and 4 meet the requirement so we return 4.

Example 2:

Input:

n = 1, lights = [[0,1]], requirement = [2]

Output:

0

Explanation:

- The first street lamp lights up the area from [max(0, 0 - 1), min(n - 1, 0 + 1)] = [0, 0] (inclusive). - Position 0 is covered by the first street lamp. It is covered by 1 street lamp which is less than requirement[0]. - We return 0 because no position meets their brightness requirement.

Constraints:

1 <= n <= 10

5

1 <= lights.length <= 10

5

0 <= position

i

< n

0 <= range

i

<= 10

5

requirement.length == n

0 <= requirement[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int meetRequirement(int n, vector<vector<int>>& lights, vector<int>&
requirement) {

}
};
```

Java:

```
class Solution {
public int meetRequirement(int n, int[][] lights, int[] requirement) {

}
}
```

Python3:

```
class Solution:
def meetRequirement(self, n: int, lights: List[List[int]], requirement:
List[int]) -> int:
```

Python:

```
class Solution(object):
def meetRequirement(self, n, lights, requirement):
"""
:type n: int
:type lights: List[List[int]]
:type requirement: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} lights
* @param {number[]} requirement
* @return {number}
*/
var meetRequirement = function(n, lights, requirement) {

};
```

TypeScript:

```
function meetRequirement(n: number, lights: number[][], requirement:
number[]): number {

};
```

C#:

```
public class Solution {
public int MeetRequirement(int n, int[][] lights, int[] requirement) {

}
}
```

C:

```
int meetRequirement(int n, int** lights, int lightsSize, int* lightsColSize,
int* requirement, int requirementSize) {

}
```

Go:

```
func meetRequirement(n int, lights [][]int, requirement []int) int {

}
```

Kotlin:

```
class Solution {
fun meetRequirement(n: Int, lights: Array<IntArray>, requirement: IntArray):
Int {

}
}
```

Swift:

```
class Solution {
func meetRequirement(_ n: Int, _ lights: [[Int]], _ requirement: [Int]) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn meet_requirement(n: i32, lights: Vec<Vec<i32>>, requirement: Vec<i32>)
-> i32 {
```

```
}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} lights
# @param {Integer[]} requirement
# @return {Integer}
def meet_requirement(n, lights, requirement)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $lights
* @param Integer[] $requirement
* @return Integer
*/
function meetRequirement($n, $lights, $requirement) {

}
}
```

Dart:

```
class Solution {
int meetRequirement(int n, List<List<int>> lights, List<int> requirement) {

}
}
```

Scala:

```
object Solution {
def meetRequirement(n: Int, lights: Array[Array[Int]], requirement:
Array[Int]): Int = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec meet_requirement(n :: integer, lights :: [[integer]], requirement ::
[integer]) :: integer
def meet_requirement(n, lights, requirement) do

end
end
```

Erlang:

```
-spec meet_requirement(N :: integer(), Lights :: [[integer()]], Requirement
:: [integer()]) -> integer().
meet_requirement(N, Lights, Requirement) ->
.
```

Racket:

```
(define/contract (meet-requirement n lights requirement)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Positions on Street With Required Brightness
* 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 meetRequirement(int n, vector<vector<int>>& lights, vector<int>&
requirement) {

}
};
```

Java Solution:

```
/**
* Problem: Count Positions on Street With Required Brightness
* 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 meetRequirement(int n, int[][] lights, int[] requirement) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Positions on Street With Required Brightness
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 meetRequirement(self, n: int, lights: List[List[int]], requirement:
```

```
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def meetRequirement(self, n, lights, requirement):
"""
:type n: int
:type lights: List[List[int]]
:type requirement: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Positions on Street With Required Brightness
* 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} n
* @param {number[][]} lights
* @param {number[]} requirement
* @return {number}
*/
var meetRequirement = function(n, lights, requirement) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Positions on Street With Required Brightness
* 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 meetRequirement(n: number, lights: number[][], requirement:
number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Positions on Street With Required Brightness
* 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 MeetRequirement(int n, int[][] lights, int[] requirement) {

}
}
```

C Solution:

```
/*
* Problem: Count Positions on Street With Required Brightness
* 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 meetRequirement(int n, int** lights, int lightsSize, int* lightsColSize,
int* requirement, int requirementSize) {

}
```

Go Solution:

```
// Problem: Count Positions on Street With Required Brightness
// 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 meetRequirement(n int, lights [][]int, requirement []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun meetRequirement(n: Int, lights: Array<IntArray>, requirement: IntArray):
Int {

}
}
```

Swift Solution:

```
class Solution {
func meetRequirement(_ n: Int, _ lights: [[Int]], _ requirement: [Int]) ->
Int {

}
}
```

Rust Solution:

```
// Problem: Count Positions on Street With Required Brightness
// 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 meet_requirement(n: i32, lights: Vec<Vec<i32>>, requirement: Vec<i32>)
-> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} lights
# @param {Integer[]} requirement
# @return {Integer}
def meet_requirement(n, lights, requirement)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $lights
* @param Integer[] $requirement
* @return Integer
*/
function meetRequirement($n, $lights, $requirement) {

}
}
```

Dart Solution:

```
class Solution {
int meetRequirement(int n, List<List<int>> lights, List<int> requirement) {

}
}
```

Scala Solution:

```
object Solution {
def meetRequirement(n: Int, lights: Array[Array[Int]], requirement:
Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec meet_requirement(n :: integer, lights :: [[integer]], requirement ::
[integer]) :: integer
def meet_requirement(n, lights, requirement) do

end
end
```

Erlang Solution:

```
-spec meet_requirement(N :: integer(), Lights :: [[integer()]], Requirement
:: [integer()]) -> integer().
meet_requirement(N, Lights, Requirement) ->
.
```

Racket Solution:

```
(define/contract (meet-requirement n lights requirement)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```