

Problem 1893: Check if All the Integers in a Range Are Covered

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a 2D integer array

ranges

and two integers

left

and

right

. Each

`ranges[i] = [start`

`i`

`, end`

`i`

`]`

represents an

inclusive

interval between

start

i

and

end

i

.

Return

true

if each integer in the inclusive range

[left, right]

is covered by

at least one

interval in

ranges

. Return

false

otherwise

An integer

x

is covered by an interval

ranges[i] = [start

i

, end

i

]

if

start

i

<= x <= end

i

Example 1:

Input:

ranges = [[1,2],[3,4],[5,6]], left = 2, right = 5

Output:

true

Explanation:

Every integer between 2 and 5 is covered: - 2 is covered by the first range. - 3 and 4 are covered by the second range. - 5 is covered by the third range.

Example 2:

Input:

```
ranges = [[1,10],[10,20]], left = 21, right = 21
```

Output:

```
false
```

Explanation:

21 is not covered by any range.

Constraints:

```
1 <= ranges.length <= 50
```

```
1 <= start
```

```
i
```

```
<= end
```

```
i
```

```
<= 50
```

```
1 <= left <= right <= 50
```

Code Snippets

C++:

```
class Solution {  
public:  
    bool isCovered(vector<vector<int>>& ranges, int left, int right) {  
  
    }  
};
```

Java:

```
class Solution {  
public boolean isCovered(int[][] ranges, int left, int right) {  
  
}  
}
```

Python3:

```
class Solution:  
    def isCovered(self, ranges: List[List[int]], left: int, right: int) -> bool:
```

Python:

```
class Solution(object):  
    def isCovered(self, ranges, left, right):  
        """  
        :type ranges: List[List[int]]  
        :type left: int  
        :type right: int  
        :rtype: bool  
        """
```

JavaScript:

```
/**  
 * @param {number[][]} ranges  
 * @param {number} left  
 * @param {number} right  
 * @return {boolean}  
 */  
var isCovered = function(ranges, left, right) {
```

```
};
```

TypeScript:

```
function isCovered(ranges: number[][][], left: number, right: number): boolean
{
    ...
}
```

C#:

```
public class Solution {
    public bool IsCovered(int[][][] ranges, int left, int right) {
        ...
    }
}
```

C:

```
bool isCovered(int** ranges, int rangesSize, int* rangesColSize, int left,
int right) {
    ...
}
```

Go:

```
func isCovered(ranges [][]int, left int, right int) bool {
    ...
}
```

Kotlin:

```
class Solution {
    fun isCovered(ranges: Array<IntArray>, left: Int, right: Int): Boolean {
        ...
    }
}
```

Swift:

```
class Solution {
    func isCovered(_ ranges: [[Int]], _ left: Int, _ right: Int) -> Bool {
```

```
}
```

```
}
```

Rust:

```
impl Solution {
    pub fn is_covered(ranges: Vec<Vec<i32>>, left: i32, right: i32) -> bool {
        }
}
```

Ruby:

```
# @param {Integer[][]} ranges
# @param {Integer} left
# @param {Integer} right
# @return {Boolean}
def is_covered(ranges, left, right)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[][] $ranges
     * @param Integer $left
     * @param Integer $right
     * @return Boolean
     */
    function isCovered($ranges, $left, $right) {

    }
}
```

Dart:

```
class Solution {
    bool isCovered(List<List<int>> ranges, int left, int right) {
```

```
}
```

```
}
```

Scala:

```
object Solution {  
    def isCovered(ranges: Array[Array[Int]], left: Int, right: Int): Boolean = {  
  
    }  
    }  
}
```

Elixir:

```
defmodule Solution do  
    @spec is_covered([integer()]) :: integer(), integer() :: boolean  
    def is_covered(ranges, left, right) do  
  
    end  
    end
```

Erlang:

```
-spec is_covered([[integer()]], integer(), integer()) ::  
    boolean().  
is_covered(Ranges, Left, Right) ->  
.
```

Racket:

```
(define/contract (is-covered ranges left right)  
  (-> (listof (listof exact-integer?)) exact-integer? exact-integer? boolean?)  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Check if All the Integers in a Range Are Covered
```

```

* Difficulty: Easy
* 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:
    bool isCovered(vector<vector<int>>& ranges, int left, int right) {

    }
};

```

Java Solution:

```

/**
 * Problem: Check if All the Integers in a Range Are Covered
 * Difficulty: Easy
 * 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 boolean isCovered(int[][] ranges, int left, int right) {

}
}

```

Python3 Solution:

```

"""
Problem: Check if All the Integers in a Range Are Covered
Difficulty: Easy
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 isCovered(self, ranges: List[List[int]], left: int, right: int) -> bool:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):

def isCovered(self, ranges, left, right):
"""

:type ranges: List[List[int]]
:type left: int
:type right: int
:rtype: bool
"""


```

JavaScript Solution:

```

/**
 * Problem: Check if All the Integers in a Range Are Covered
 * Difficulty: Easy
 * 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[][]} ranges
 * @param {number} left
 * @param {number} right
 * @return {boolean}
 */
var isCovered = function(ranges, left, right) {

};


```

TypeScript Solution:

```
/**  
 * Problem: Check if All the Integers in a Range Are Covered  
 * Difficulty: Easy  
 * 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 isCovered(ranges: number[][][], left: number, right: number): boolean  
{  
  
};
```

C# Solution:

```
/*  
 * Problem: Check if All the Integers in a Range Are Covered  
 * Difficulty: Easy  
 * 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 bool IsCovered(int[][] ranges, int left, int right) {  
  
    }  
}
```

C Solution:

```
/*  
 * Problem: Check if All the Integers in a Range Are Covered  
 * Difficulty: Easy  
 * 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
*/
bool isCovered(int** ranges, int rangesSize, int* rangesColSize, int left,
int right) {

}

```

Go Solution:

```

// Problem: Check if All the Integers in a Range Are Covered
// Difficulty: Easy
// 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 isCovered(ranges [][]int, left int, right int) bool {
}

```

Kotlin Solution:

```

class Solution {
    fun isCovered(ranges: Array<IntArray>, left: Int, right: Int): Boolean {
        }
    }
}

```

Swift Solution:

```

class Solution {
    func isCovered(_ ranges: [[Int]], _ left: Int, _ right: Int) -> Bool {
        }
    }
}

```

Rust Solution:

```

// Problem: Check if All the Integers in a Range Are Covered
// Difficulty: Easy
// 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 is_covered(ranges: Vec<Vec<i32>>, left: i32, right: i32) -> bool {
        }

    }
}

```

Ruby Solution:

```

# @param {Integer[][]} ranges
# @param {Integer} left
# @param {Integer} right
# @return {Boolean}
def is_covered(ranges, left, right)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer[][] $ranges
     * @param Integer $left
     * @param Integer $right
     * @return Boolean
     */
    function isCovered($ranges, $left, $right) {

    }
}

```

Dart Solution:

```
class Solution {  
    bool isCovered(List<List<int>> ranges, int left, int right) {  
        }  
    }  
}
```

Scala Solution:

```
object Solution {  
    def isCovered(ranges: Array[Array[Int]], left: Int, right: Int): Boolean = {  
        }  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
    @spec is_covered(ranges :: [[integer]], left :: integer, right :: integer) ::  
        boolean  
    def is_covered(ranges, left, right) do  
  
    end  
    end
```

Erlang Solution:

```
-spec is_covered(Ranges :: [[integer()]], Left :: integer(), Right ::  
    integer()) -> boolean().  
is_covered(Ranges, Left, Right) ->  
.
```

Racket Solution:

```
(define/contract (is-covered ranges left right)  
  (-> (listof (listof exact-integer?)) exact-integer? exact-integer? boolean?)  
)
```