Problem 1288: Remove Covered Intervals

Problem Information

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

Problem Description

Given an array

intervals

where

intervals[i] = [l

i

, r

i

]

represent the interval

[l

i

, r

i

)

, remove all intervals that are covered by another interval in the list.

The interval

[a, b)

is covered by the interval

[c, d)

if and only if

c <= a

and

b <= d

.

Return

the number of remaining intervals

.

Example 1:

Input:

intervals = [[1,4],[3,6],[2,8]]

Output:

2

Explanation:

Interval [3,6] is covered by [2,8], therefore it is removed.

Example 2:

Input:

intervals = [[1,4],[2,3]]

Output:

1

Constraints:

1 <= intervals.length <= 1000

intervals[i].length == 2

0 <= l

i

< r

i

<= 10

5

All the given intervals are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int removeCoveredIntervals(vector<vector<int>>& intervals) {

}
};
```

Java:

```
class Solution {
public int removeCoveredIntervals(int[][] intervals) {

}
}
```

Python3:

```
class Solution:
def removeCoveredIntervals(self, intervals: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function removeCoveredIntervals(intervals: number[][]): number {

};
```

C#:

```
public class Solution {
public int RemoveCoveredIntervals(int[][] intervals) {

}
}
```

C:

```
int removeCoveredIntervals(int** intervals, int intervalsSize, int*
intervalsColSize) {

}
```

Go:

```
func removeCoveredIntervals(intervals [][]int) int {

}
```

Kotlin:

```
class Solution {
fun removeCoveredIntervals(intervals: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func removeCoveredIntervals(_ intervals: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn remove_covered_intervals(intervals: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} intervals
# @return {Integer}
def remove_covered_intervals(intervals)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $intervals
* @return Integer
*/
function removeCoveredIntervals($intervals) {

}
}
```

Dart:

```
class Solution {
int removeCoveredIntervals(List<List<int>> intervals) {

}
}
```

Scala:

```
object Solution {
def removeCoveredIntervals(intervals: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec remove_covered_intervals(intervals :: [[integer]]) :: integer
def remove_covered_intervals(intervals) do

end
end
```

Erlang:

```
-spec remove_covered_intervals(Intervals :: [[integer()]]) -> integer().
remove_covered_intervals(Intervals) ->
.
```

Racket:

```
(define/contract (remove-covered-intervals intervals)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Covered Intervals
* Difficulty: Medium
* Tags: array, sort
*
* 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 removeCoveredIntervals(vector<vector<int>>& intervals) {

}
};
```

Java Solution:

```
/**
* Problem: Remove Covered Intervals
* Difficulty: Medium
* Tags: array, sort
*
* 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 removeCoveredIntervals(int[][] intervals) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove Covered Intervals
Difficulty: Medium
Tags: array, sort

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 removeCoveredIntervals(self, intervals: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Remove Covered Intervals
* Difficulty: Medium
* Tags: array, sort
*
* 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[][]} intervals
* @return {number}
*/
var removeCoveredIntervals = function(intervals) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove Covered Intervals
* Difficulty: Medium
* Tags: array, sort
*
* 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 removeCoveredIntervals(intervals: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Remove Covered Intervals
* Difficulty: Medium
* Tags: array, sort
*
* 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 RemoveCoveredIntervals(int[][] intervals) {

}
}
```

C Solution:

```
/*
* Problem: Remove Covered Intervals
* Difficulty: Medium
* Tags: array, sort
*
* 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 removeCoveredIntervals(int** intervals, int intervalsSize, int*
intervalsColSize) {

}
```

Go Solution:

```
// Problem: Remove Covered Intervals
// Difficulty: Medium
// Tags: array, sort
//
// 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 removeCoveredIntervals(intervals [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun removeCoveredIntervals(intervals: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func removeCoveredIntervals(_ intervals: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Remove Covered Intervals
// Difficulty: Medium
// Tags: array, sort
//
// 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 remove_covered_intervals(intervals: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} intervals
# @return {Integer}
```

```
def remove_covered_intervals(intervals)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $intervals
* @return Integer
*/
function removeCoveredIntervals($intervals) {

}
}
```

Dart Solution:

```
class Solution {
int removeCoveredIntervals(List<List<int>> intervals) {

}
}
```

Scala Solution:

```
object Solution {
def removeCoveredIntervals(intervals: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec remove_covered_intervals(intervals :: [[integer]]) :: integer
def remove_covered_intervals(intervals) do

end
end
```

Erlang Solution:

```
-spec remove_covered_intervals(Intervals :: [[integer()]]) -> integer().
remove_covered_intervals(Intervals) ->
.
```

Racket Solution:

```
(define/contract (remove-covered-intervals intervals)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```