Problem 435: Non-overlapping Intervals

Problem Information

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

Problem Description

Given an array of intervals

intervals

where

intervals[i] = [start

i

, end

i

]

, return

the minimum number of intervals you need to remove to make the rest of the intervals non-overlapping

.

Note

that intervals which only touch at a point are

non-overlapping

. For example,

[1, 2]

and

[2, 3]

are non-overlapping.

Example 1:

Input:

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

Output:

1

Explanation:

[1,3] can be removed and the rest of the intervals are non-overlapping.

Example 2:

Input:

intervals = [[1,2],[1,2],[1,2]]

Output:

2

Explanation:

You need to remove two [1,2] to make the rest of the intervals non-overlapping.

Example 3:

Input:

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

Output:

0

Explanation:

You don't need to remove any of the intervals since they're already non-overlapping.

Constraints:

1 <= intervals.length <= 10

5

intervals[i].length == 2

-5 * 10

4

<= start

i

< end

i

<= 5 * 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (erase-overlap-intervals intervals)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Non-overlapping Intervals
* Difficulty: Medium
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Non-overlapping Intervals
* Difficulty: Medium
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int eraseOverlapIntervals(int[][] intervals) {

}
}
```

Python3 Solution:

```
"""
Problem: Non-overlapping Intervals
Difficulty: Medium
Tags: array, dp, greedy, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def eraseOverlapIntervals(self, intervals: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Non-overlapping Intervals
* Difficulty: Medium
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Non-overlapping Intervals
* Difficulty: Medium
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

```
};
```

C# Solution:

```
/*
* Problem: Non-overlapping Intervals
* Difficulty: Medium
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int EraseOverlapIntervals(int[][] intervals) {

}
}
```

C Solution:

```
/*
* Problem: Non-overlapping Intervals
* Difficulty: Medium
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int eraseOverlapIntervals(int** intervals, int intervalsSize, int*
intervalsColSize) {

}
```

Go Solution:

```
// Problem: Non-overlapping Intervals
// Difficulty: Medium
// Tags: array, dp, greedy, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func eraseOverlapIntervals(intervals [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Non-overlapping Intervals
// Difficulty: Medium
// Tags: array, dp, greedy, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (erase-overlap-intervals intervals)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```