Problem 757: Set Intersection Size At Least Two

Problem Information

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

Problem Description

You are given a 2D integer array

intervals

where

intervals[i] = [start

i

, end

i

]

represents all the integers from

start

i

to

end

i

inclusively.

A

containing set

is an array

nums

where each interval from

intervals

has

at least two

integers in

nums

.

For example, if

intervals = [[1,3], [3,7], [8,9]]

, then

[1,2,4,7,8,9]

and

[2,3,4,8,9]

are

containing sets

.

Return

the minimum possible size of a containing set

.

Example 1:

Input:

intervals = [[1,3],[3,7],[8,9]]

Output:

5

Explanation:

let nums = [2, 3, 4, 8, 9]. It can be shown that there cannot be any containing array of size 4.

Example 2:

Input:

intervals = [[1,3],[1,4],[2,5],[3,5]]

Output:

3

Explanation:

let nums = [2, 3, 4]. It can be shown that there cannot be any containing array of size 2.

Example 3:

Input:

intervals = [[1,2],[2,3],[2,4],[4,5]]

Output:

5

Explanation:

let nums = [1, 2, 3, 4, 5]. It can be shown that there cannot be any containing array of size 4.

Constraints:

1 <= intervals.length <= 3000

intervals[i].length == 2

0 <= start

i

< end

i

<= 10

8

Code Snippets

C++:

```
class Solution {
public:
```

```
int intersectionSizeTwo(vector<vector<int>>& intervals) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
```

```
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (intersection-size-two intervals)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Set Intersection Size At Least Two
* Difficulty: Hard
* Tags: array, greedy, 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 intersectionSizeTwo(vector<vector<int>>& intervals) {

}
};
```

Java Solution:

```
/**
* Problem: Set Intersection Size At Least Two
* Difficulty: Hard
* Tags: array, greedy, 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 intersectionSizeTwo(int[][] intervals) {

}
}
```

Python3 Solution:

```
"""
Problem: Set Intersection Size At Least Two
Difficulty: Hard
Tags: array, greedy, 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 intersectionSizeTwo(self, intervals: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Set Intersection Size At Least Two
* Difficulty: Hard
* Tags: array, greedy, 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 intersectionSizeTwo = function(intervals) {

};
```

TypeScript Solution:

```
/**
* Problem: Set Intersection Size At Least Two
* Difficulty: Hard
* Tags: array, greedy, 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 intersectionSizeTwo(intervals: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Set Intersection Size At Least Two
* Difficulty: Hard
* Tags: array, greedy, 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 IntersectionSizeTwo(int[][] intervals) {

}
}
```

C Solution:

```
/*
* Problem: Set Intersection Size At Least Two
* Difficulty: Hard
* Tags: array, greedy, 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 intersectionSizeTwo(int** intervals, int intervalsSize, int*
intervalsColSize) {

}
```

Go Solution:

```
// Problem: Set Intersection Size At Least Two
// Difficulty: Hard
// Tags: array, greedy, 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 intersectionSizeTwo(intervals [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Set Intersection Size At Least Two
// Difficulty: Hard
// Tags: array, greedy, 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 intersection_size_two(intervals: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (intersection-size-two intervals)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```