Problem 850: Rectangle Area II

Problem Information

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

Problem Description

You are given a 2D array of axis-aligned

rectangles

. Each

rectangle[i] = [x

i1

, y

i1

, x

i2

, y

i2

]

denotes the

i

th

rectangle where

(x

i1

, y

i1

)

are the coordinates of the

bottom-left corner

, and

(x

i2

, y

i2

)

are the coordinates of the

top-right corner

.

Calculate the

total area

covered by all

rectangles

in the plane. Any area covered by two or more rectangles should only be counted

once

.

Return

the

total area

. Since the answer may be too large, return it

modulo

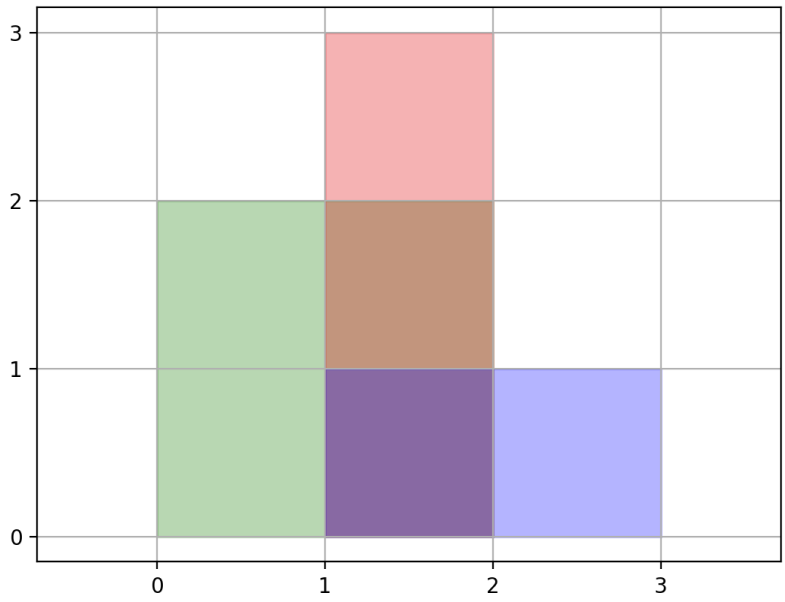
10

9

+ 7

.

Example 1:



Input:

rectangles = [[0,0,2,2],[1,0,2,3],[1,0,3,1]]

Output:

6

Explanation:

A total area of 6 is covered by all three rectangles, as illustrated in the picture. From (1,1) to (2,2), the green and red rectangles overlap. From (1,0) to (2,3), all three rectangles overlap.

Example 2:

Input:

rectangles = [[0,0,1000000000,1000000000]]

Output:

49

Explanation:

The answer is 10

18

modulo (10

9

+ 7), which is 49.

Constraints:

1 <= rectangles.length <= 200

rectanges[i].length == 4

0 <= x

i1

, y

i1

, x

i2

, y

i2

<= 10

9

x

i1 <=

x

i2

y

i1 <=

y

i2

All rectangles have non zero area.

Code Snippets

C++:

```
class Solution {
public:
int rectangleArea(vector<vector<int>>& rectangles) {

}
};
```

Java:

```
class Solution {
public int rectangleArea(int[][] rectangles) {

}
}
```

Python3:

```
class Solution:
def rectangleArea(self, rectangles: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function rectangleArea(rectangles: number[][]): number {

};
```

C#:

```
public class Solution {
public int RectangleArea(int[][] rectangles) {

}
}
```

C:

```
int rectangleArea(int** rectangles, int rectanglesSize, int*
rectanglesColSize) {

}
```

Go:

```
func rectangleArea(rectangles [][]int) int {

}
```

Kotlin:

```
class Solution {
fun rectangleArea(rectangles: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func rectangleArea(_ rectangles: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn rectangle_area(rectangles: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} rectangles
# @return {Integer}
def rectangle_area(rectangles)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $rectangles
* @return Integer
```

```
*/
function rectangleArea($rectangles) {

}
}
```

Dart:

```
class Solution {
int rectangleArea(List<List<int>> rectangles) {

}
}
```

Scala:

```
object Solution {
def rectangleArea(rectangles: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec rectangle_area(rectangles :: [[integer]]) :: integer
def rectangle_area(rectangles) do

end
end
```

Erlang:

```
-spec rectangle_area(Rectangles :: [[integer()]]) -> integer().
rectangle_area(Rectangles) ->
.
```

Racket:

```
(define/contract (rectangle-area rectangles)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Rectangle Area II
* Difficulty: Hard
* 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 rectangleArea(vector<vector<int>>& rectangles) {

}
};
```

Java Solution:

```
/**
* Problem: Rectangle Area II
* Difficulty: Hard
* 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 rectangleArea(int[][] rectangles) {

}
}
```

Python3 Solution:

```
"""
Problem: Rectangle Area II
Difficulty: Hard
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 rectangleArea(self, rectangles: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Rectangle Area II
* Difficulty: Hard
* 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[][]} rectangles
* @return {number}
*/
var rectangleArea = function(rectangles) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Rectangle Area II
* Difficulty: Hard
* 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 rectangleArea(rectangles: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Rectangle Area II
* Difficulty: Hard
* 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 RectangleArea(int[][] rectangles) {

}
}
```

C Solution:

```
/*
* Problem: Rectangle Area II
* Difficulty: Hard
```

```
* 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 rectangleArea(int** rectangles, int rectanglesSize, int*
rectanglesColSize) {

}
```

Go Solution:

```
// Problem: Rectangle Area II
// Difficulty: Hard
// 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 rectangleArea(rectangles [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun rectangleArea(rectangles: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func rectangleArea(_ rectangles: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Rectangle Area II
// Difficulty: Hard
// 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 rectangle_area(rectangles: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} rectangles
# @return {Integer}
def rectangle_area(rectangles)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $rectangles
* @return Integer
*/
function rectangleArea($rectangles) {

}
}
```

Dart Solution:

```
class Solution {
int rectangleArea(List<List<int>> rectangles) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def rectangleArea(rectangles: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec rectangle_area(rectangles :: [[integer]]) :: integer
def rectangle_area(rectangles) do

end
end
```

Erlang Solution:

```
-spec rectangle_area(Rectangles :: [[integer()]]) -> integer().
rectangle_area(Rectangles) ->
.
```

Racket Solution:

```
(define/contract (rectangle-area rectangles)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```