

Problem 836: Rectangle Overlap

Problem Information

Difficulty: Easy

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

An axis-aligned rectangle is represented as a list

$[x1, y1, x2, y2]$

, where

$(x1, y1)$

is the coordinate of its bottom-left corner, and

$(x2, y2)$

is the coordinate of its top-right corner. Its top and bottom edges are parallel to the X-axis, and its left and right edges are parallel to the Y-axis.

Two rectangles overlap if the area of their intersection is

positive

. To be clear, two rectangles that only touch at the corner or edges do not overlap.

Given two axis-aligned rectangles

rec1

and

rec2

, return

true

if they overlap, otherwise return

false

.

Example 1:

Input:

rec1 = [0,0,2,2], rec2 = [1,1,3,3]

Output:

true

Example 2:

Input:

rec1 = [0,0,1,1], rec2 = [1,0,2,1]

Output:

false

Example 3:

Input:

rec1 = [0,0,1,1], rec2 = [2,2,3,3]

Output:

false

Constraints:

rec1.length == 4

rec2.length == 4

-10

9

<= rec1[i], rec2[i] <= 10

9

rec1

and

rec2

represent a valid rectangle with a non-zero area.

Code Snippets

C++:

```
class Solution {
public:
    bool isRectangleOverlap(vector<int>& rec1, vector<int>& rec2) {

    }
};
```

Java:

```

class Solution {
public boolean isRectangleOverlap(int[] rec1, int[] rec2) {

}

}

```

Python3:

```

class Solution:
def isRectangleOverlap(self, rec1: List[int], rec2: List[int]) -> bool:

```

Python:

```

class Solution(object):
def isRectangleOverlap(self, rec1, rec2):
"""
:type rec1: List[int]
:type rec2: List[int]
:rtype: bool
"""

```

JavaScript:

```

/**
 * @param {number[]} rec1
 * @param {number[]} rec2
 * @return {boolean}
 */
var isRectangleOverlap = function(rec1, rec2) {

};

```

TypeScript:

```

function isRectangleOverlap(rec1: number[], rec2: number[]): boolean {

};

```

C#:

```

public class Solution {
public bool IsRectangleOverlap(int[] rec1, int[] rec2) {

```

```
}  
}
```

C:

```
bool isRectangleOverlap(int* rec1, int rec1Size, int* rec2, int rec2Size) {  
  
}
```

Go:

```
func isRectangleOverlap(rec1 []int, rec2 []int) bool {  
  
}
```

Kotlin:

```
class Solution {  
    fun isRectangleOverlap(rec1: IntArray, rec2: IntArray): Boolean {  
  
    }  
}
```

Swift:

```
class Solution {  
    func isRectangleOverlap(_ rec1: [Int], _ rec2: [Int]) -> Bool {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn is_rectangle_overlap(rec1: Vec<i32>, rec2: Vec<i32>) -> bool {  
  
    }  
}
```

Ruby:

```

# @param {Integer[]} rec1
# @param {Integer[]} rec2
# @return {Boolean}
def is_rectangle_overlap(rec1, rec2)

end

```

PHP:

```

class Solution {

    /**
     * @param Integer[] $rec1
     * @param Integer[] $rec2
     * @return Boolean
     */
    function isRectangleOverlap($rec1, $rec2) {

    }

}

```

Dart:

```

class Solution {
  bool isRectangleOverlap(List<int> rec1, List<int> rec2) {

  }

}

```

Scala:

```

object Solution {
  def isRectangleOverlap(rec1: Array[Int], rec2: Array[Int]): Boolean = {

  }

}

```

Elixir:

```

defmodule Solution do
  @spec is_rectangle_overlap(rec1 :: [integer], rec2 :: [integer]) :: boolean
  def is_rectangle_overlap(rec1, rec2) do

```

```
end
end
```

Erlang:

```
-spec is_rectangle_overlap(Rec1 :: [integer()], Rec2 :: [integer()]) ->
boolean().
is_rectangle_overlap(Rec1, Rec2) ->
.
```

Racket:

```
(define/contract (is-rectangle-overlap rec1 rec2)
  (-> (listof exact-integer?) (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Rectangle Overlap
 * Difficulty: Easy
 * Tags: math
 *
 * Approach: Optimized algorithm based on problem constraints
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
public:
    bool isRectangleOverlap(vector<int>& rec1, vector<int>& rec2) {

    }
};
```

Java Solution:

```

/**
 * Problem: Rectangle Overlap
 * Difficulty: Easy
 * Tags: math
 *
 * Approach: Optimized algorithm based on problem constraints
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
public boolean isRectangleOverlap(int[] rec1, int[] rec2) {

}
}

```

Python3 Solution:

```

"""
Problem: Rectangle Overlap
Difficulty: Easy
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def isRectangleOverlap(self, rec1: List[int], rec2: List[int]) -> bool:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def isRectangleOverlap(self, rec1, rec2):
"""
:type rec1: List[int]
:type rec2: List[int]
:rtype: bool
"""

```

JavaScript Solution:

```
/**
 * Problem: Rectangle Overlap
 * Difficulty: Easy
 * Tags: math
 *
 * Approach: Optimized algorithm based on problem constraints
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

/**
 * @param {number[]} rec1
 * @param {number[]} rec2
 * @return {boolean}
 */
var isRectangleOverlap = function(rec1, rec2) {

};
```

TypeScript Solution:

```
/**
 * Problem: Rectangle Overlap
 * Difficulty: Easy
 * Tags: math
 *
 * Approach: Optimized algorithm based on problem constraints
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

function isRectangleOverlap(rec1: number[], rec2: number[]): boolean {

};
```

C# Solution:

```
/*
 * Problem: Rectangle Overlap
 * Difficulty: Easy
```

```

* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool IsRectangleOverlap(int[] rec1, int[] rec2) {

}
}

```

C Solution:

```

/*
* Problem: Rectangle Overlap
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

bool isRectangleOverlap(int* rec1, int rec1Size, int* rec2, int rec2Size) {

}

```

Go Solution:

```

// Problem: Rectangle Overlap
// Difficulty: Easy
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func isRectangleOverlap(rec1 []int, rec2 []int) bool {

```

```
}
```

Kotlin Solution:

```
class Solution {  
    fun isRectangleOverlap(rec1: IntArray, rec2: IntArray): Boolean {  
  
    }  
}
```

Swift Solution:

```
class Solution {  
    func isRectangleOverlap(_ rec1: [Int], _ rec2: [Int]) -> Bool {  
  
    }  
}
```

Rust Solution:

```
// Problem: Rectangle Overlap  
// Difficulty: Easy  
// Tags: math  
//  
// Approach: Optimized algorithm based on problem constraints  
// Time Complexity: O(n) to O(n^2) depending on approach  
// Space Complexity: O(1) to O(n) depending on approach  
  
impl Solution {  
    pub fn is_rectangle_overlap(rec1: Vec<i32>, rec2: Vec<i32>) -> bool {  
  
    }  
}
```

Ruby Solution:

```
# @param {Integer[]} rec1  
# @param {Integer[]} rec2  
# @return {Boolean}  
def is_rectangle_overlap(rec1, rec2)
```

```
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[] $rec1  
     * @param Integer[] $rec2  
     * @return Boolean  
     */  
    function isRectangleOverlap($rec1, $rec2) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
    bool isRectangleOverlap(List<int> rec1, List<int> rec2) {  
  
    }  
}
```

Scala Solution:

```
object Solution {  
    def isRectangleOverlap(rec1: Array[Int], rec2: Array[Int]): Boolean = {  
  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
    @spec is_rectangle_overlap(rec1 :: [integer], rec2 :: [integer]) :: boolean  
    def is_rectangle_overlap(rec1, rec2) do  
  
    end  
end
```

Erlang Solution:

```
-spec is_rectangle_overlap(Rec1 :: [integer()], Rec2 :: [integer()]) ->
boolean().
is_rectangle_overlap(Rec1, Rec2) ->
.
```

Racket Solution:

```
(define/contract (is-rectangle-overlap rec1 rec2)
  (-> (listof exact-integer?) (listof exact-integer?) boolean?)
)
```