Problem 3394: Check if Grid can be Cut into Sections

Problem Information

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

Problem Description

You are given an integer

n

representing the dimensions of an

n x n

grid, with the origin at the bottom-left corner of the grid. You are also given a 2D array of coordinates

rectangles

, where

rectangles[i]

is in the form

[start

x

, start

y

, end

x

, end

y

]

, representing a rectangle on the grid. Each rectangle is defined as follows:

(start

x

, start

y

)

: The bottom-left corner of the rectangle.

(end

x

, end

y

)

: The top-right corner of the rectangle.

Note

that the rectangles do not overlap. Your task is to determine if it is possible to make

either two horizontal or two vertical cuts

on the grid such that:

Each of the three resulting sections formed by the cuts contains

at least

one rectangle.

Every rectangle belongs to

exactly

one section.

Return

true

if such cuts can be made; otherwise, return

false

.

Example 1:

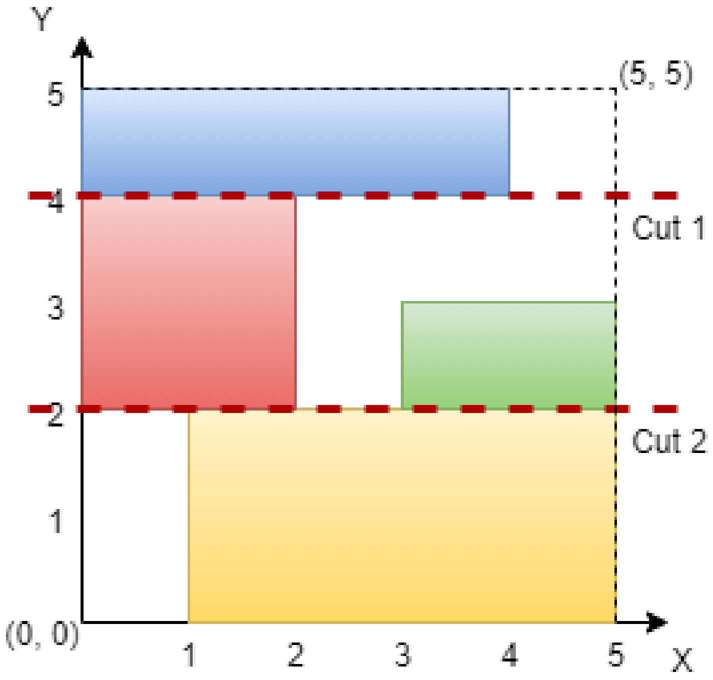
Input:

n = 5, rectangles = [[1,0,5,2],[0,2,2,4],[3,2,5,3],[0,4,4,5]]

Output:

true

Explanation:



The grid is shown in the diagram. We can make horizontal cuts at

y = 2

and

y = 4

. Hence, output is true.

Example 2:

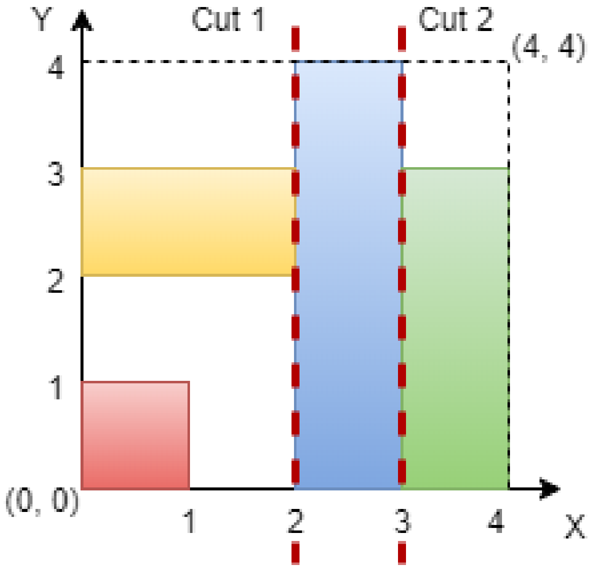
Input:

n = 4, rectangles = [[0,0,1,1],[2,0,3,4],[0,2,2,3],[3,0,4,3]]

Output:

true

Explanation:



We can make vertical cuts at

x = 2

and

x = 3

. Hence, output is true.

Example 3:

Input:

n = 4, rectangles = [[0,2,2,4],[1,0,3,2],[2,2,3,4],[3,0,4,2],[3,2,4,4]]

Output:

false

Explanation:

We cannot make two horizontal or two vertical cuts that satisfy the conditions. Hence, output is false.

Constraints:

3 <= n <= 10

9

3 <= rectangles.length <= 10

5

0 <= rectangles[i][0] < rectangles[i][2] <= n

0 <= rectangles[i][1] < rectangles[i][3] <= n

No two rectangles overlap.

Code Snippets

C++:

```
class Solution {
public:
bool checkValidCuts(int n, vector<vector<int>>& rectangles) {

}
};
```

Java:

```
class Solution {
public boolean checkValidCuts(int n, int[][] rectangles) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} rectangles
* @return {boolean}
*/
var checkValidCuts = function(n, rectangles) {

};
```

TypeScript:

```
function checkValidCuts(n: number, rectangles: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool CheckValidCuts(int n, int[][] rectangles) {

}
}
```

C:

```
bool checkValidCuts(int n, int** rectangles, int rectanglesSize, int*
rectanglesColSize) {

}
```

Go:

```
func checkValidCuts(n int, rectangles [][]int) bool {

}
```

Kotlin:

```
class Solution {
fun checkValidCuts(n: Int, rectangles: Array<IntArray>): Boolean {

}
}
```

Swift:

```
class Solution {
func checkValidCuts(_ n: Int, _ rectangles: [[Int]]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn check_valid_cuts(n: i32, rectangles: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} rectangles
# @return {Boolean}
def check_valid_cuts(n, rectangles)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $rectangles
* @return Boolean
*/
function checkValidCuts($n, $rectangles) {

}
}
```

Dart:

```
class Solution {
bool checkValidCuts(int n, List<List<int>> rectangles) {

}
}
```

Scala:

```
object Solution {
def checkValidCuts(n: Int, rectangles: Array[Array[Int]]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec check_valid_cuts(n :: integer, rectangles :: [[integer]]) :: boolean
def check_valid_cuts(n, rectangles) do

end
end
```

Erlang:

```
-spec check_valid_cuts(N :: integer(), Rectangles :: [[integer()]]) ->
boolean().
check_valid_cuts(N, Rectangles) ->
.
```

Racket:

```
(define/contract (check-valid-cuts n rectangles)
(-> exact-integer? (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if Grid can be Cut into Sections
* 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:
bool checkValidCuts(int n, vector<vector<int>>& rectangles) {

}
};
```

Java Solution:

```
/**
* Problem: Check if Grid can be Cut into Sections
* 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 boolean checkValidCuts(int n, int[][] rectangles) {
}
}
```

Python3 Solution:

```
"""
Problem: Check if Grid can be Cut into Sections
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 checkValidCuts(self, n: int, rectangles: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Check if Grid can be Cut into Sections
* 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} n
* @param {number[][]} rectangles
* @return {boolean}
*/
var checkValidCuts = function(n, rectangles) {

};
```

TypeScript Solution:

```
/**
* Problem: Check if Grid can be Cut into Sections
* 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 checkValidCuts(n: number, rectangles: number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Check if Grid can be Cut into Sections
* 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 bool CheckValidCuts(int n, int[][] rectangles) {
}
}
```

C Solution:

```
/*
* Problem: Check if Grid can be Cut into Sections
* 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
*/

bool checkValidCuts(int n, int** rectangles, int rectanglesSize, int*
rectanglesColSize) {

}
```

Go Solution:

```
// Problem: Check if Grid can be Cut into Sections
// 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 checkValidCuts(n int, rectangles [][]int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun checkValidCuts(n: Int, rectangles: Array<IntArray>): Boolean {

}

}
```

Swift Solution:

```
class Solution {
func checkValidCuts(_ n: Int, _ rectangles: [[Int]]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Check if Grid can be Cut into Sections
// 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 check_valid_cuts(n: i32, rectangles: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} rectangles
# @return {Boolean}
def check_valid_cuts(n, rectangles)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $rectangles
* @return Boolean
*/
function checkValidCuts($n, $rectangles) {

}
}
```

Dart Solution:

```
class Solution {
bool checkValidCuts(int n, List<List<int>> rectangles) {

}
}
```

Scala Solution:

```
object Solution {
def checkValidCuts(n: Int, rectangles: Array[Array[Int]]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec check_valid_cuts(n :: integer, rectangles :: [[integer]]) :: boolean
def check_valid_cuts(n, rectangles) do

end
end
```

Erlang Solution:

```
-spec check_valid_cuts(N :: integer(), Rectangles :: [[integer()]]) ->
boolean().
check_valid_cuts(N, Rectangles) ->
.
```

Racket Solution:

```
(define/contract (check-valid-cuts n rectangles)
(-> exact-integer? (listof (listof exact-integer?)) boolean?)
)
```