Problem 3235: Check if the Rectangle Corner Is Reachable

Problem Information

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

Problem Description

You are given two positive integers

xCorner

and

yCorner

, and a 2D array

circles

, where

circles[i] = [x

i

, y

i

, r

i

]

denotes a circle with center at

(x

i

, y

i

)

and radius

r

i

.

There is a rectangle in the coordinate plane with its bottom left corner at the origin and top right corner at the coordinate

(xCorner, yCorner)

. You need to check whether there is a path from the bottom left corner to the top right corner such that the

entire path

lies inside the rectangle,

does not

touch or lie inside

any

circle, and touches the rectangle

only

at the two corners.

Return

true

if such a path exists, and

false

otherwise.

Example 1:

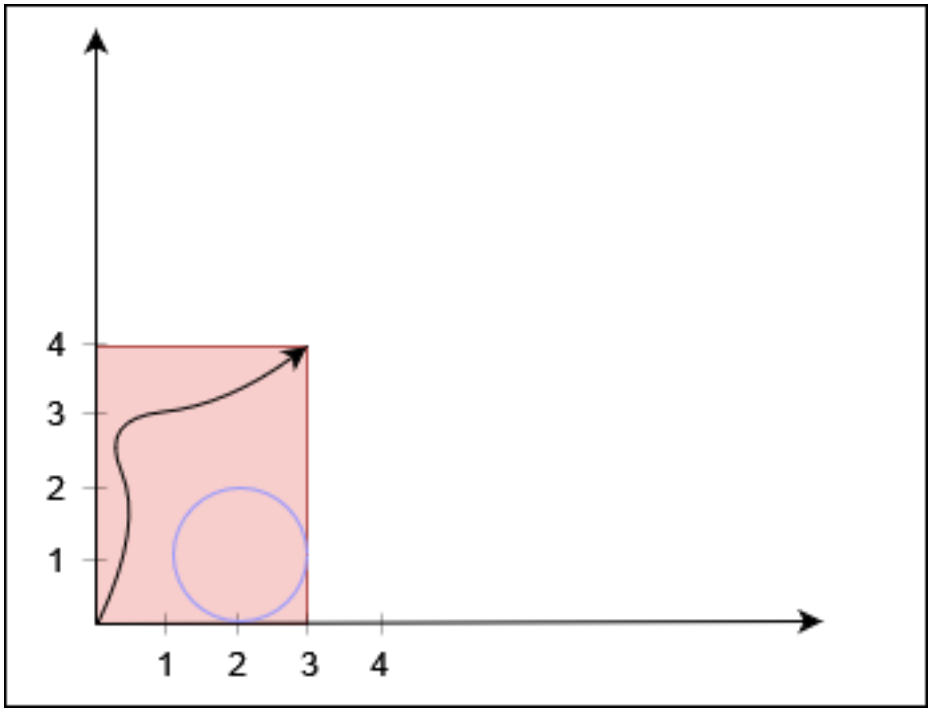
Input:

xCorner = 3, yCorner = 4, circles = [[2,1,1]]

Output:

true

Explanation:



The black curve shows a possible path between

(0, 0)

and

(3, 4)

.

Example 2:

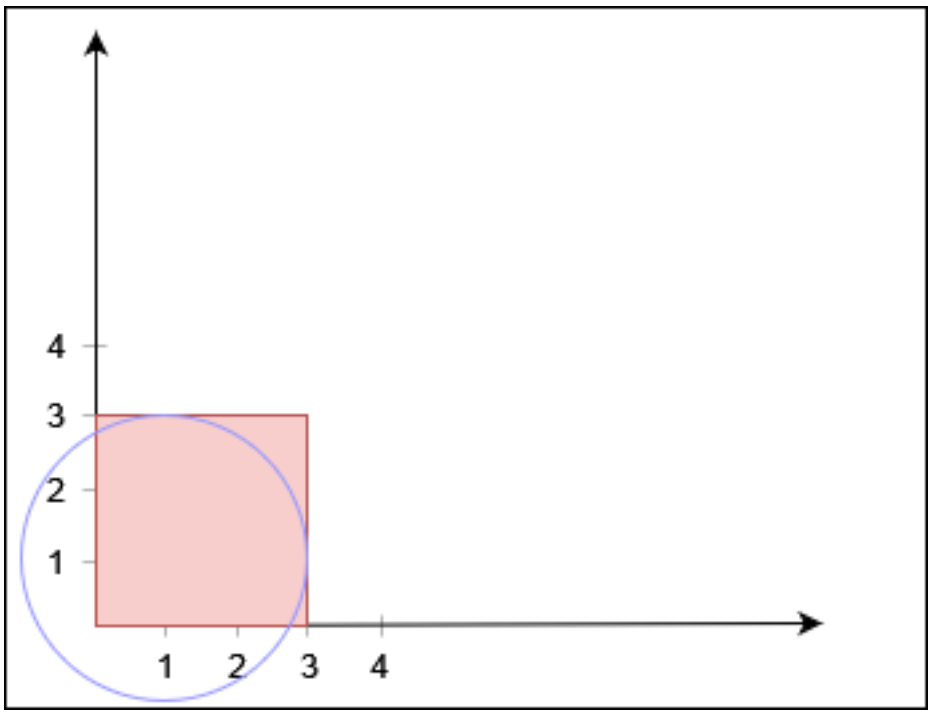
Input:

xCorner = 3, yCorner = 3, circles = [[1,1,2]]

Output:

false

Explanation:



No path exists from

(0, 0)

to

(3, 3)

.

Example 3:

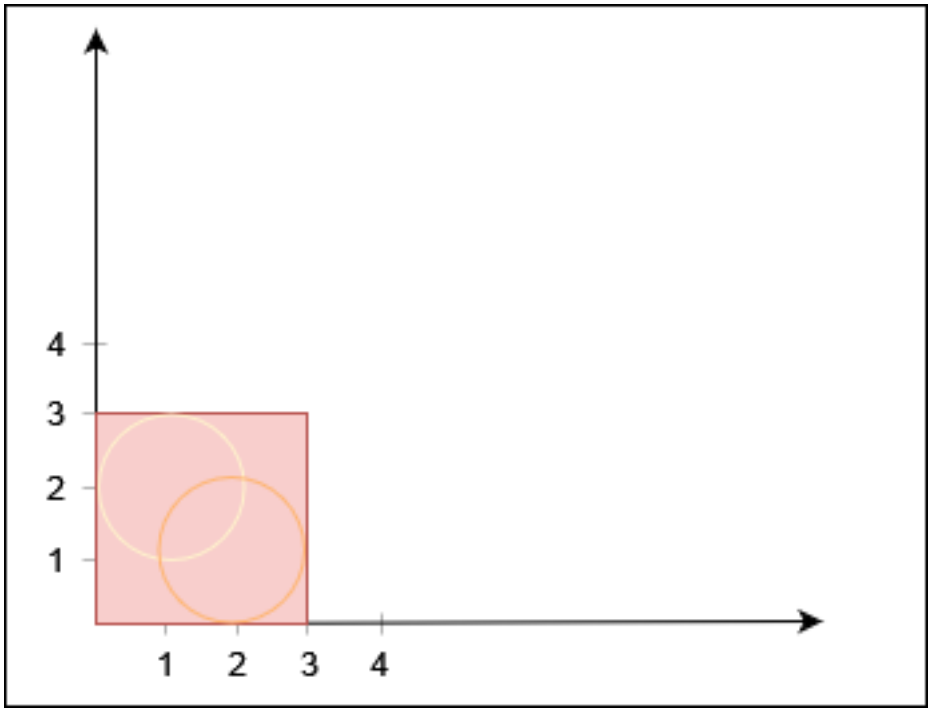
Input:

xCorner = 3, yCorner = 3, circles = [[2,1,1],[1,2,1]]

Output:

false

Explanation:



No path exists from

(0, 0)

to

(3, 3)

.

Example 4:

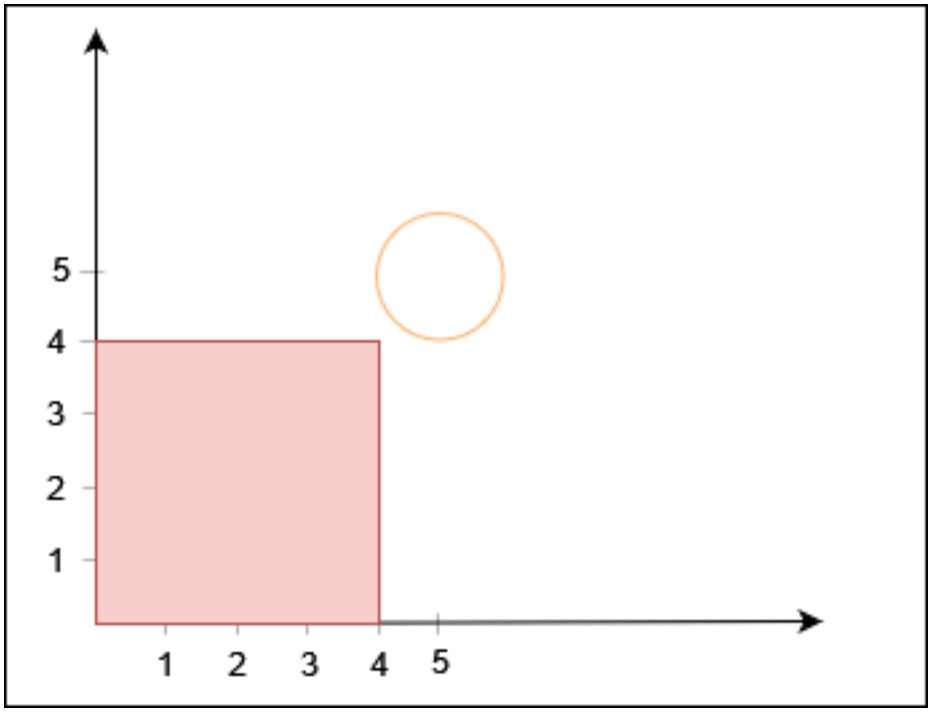
Input:

xCorner = 4, yCorner = 4, circles = [[5,5,1]]

Output:

true

Explanation:



Constraints:

3 <= xCorner, yCorner <= 10

9

1 <= circles.length <= 1000

circles[i].length == 3

1 <= x

i

, y

i

, r

i

<= 10

Code Snippets

C++:

```
class Solution {
public:
bool canReachCorner(int xCorner, int yCorner, vector<vector<int>>& circles) {

}
};
```

Java:

```
class Solution {
public boolean canReachCorner(int xCorner, int yCorner, int[][] circles) {

}
}
```

Python3:

```
class Solution:
def canReachCorner(self, xCorner: int, yCorner: int, circles:
List[List[int]]) -> bool:
```

Python:

```
class Solution(object):
def canReachCorner(self, xCorner, yCorner, circles):
"""
:type xCorner: int
:type yCorner: int
:type circles: List[List[int]]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number} xCorner
* @param {number} yCorner
* @param {number[][]} circles
* @return {boolean}
*/
var canReachCorner = function(xCorner, yCorner, circles) {

};
```

TypeScript:

```
function canReachCorner(xCorner: number, yCorner: number, circles:
number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool CanReachCorner(int xCorner, int yCorner, int[][] circles) {

}
}
```

C:

```
bool canReachCorner(int xCorner, int yCorner, int** circles, int circlesSize,
int* circlesColSize) {

}
```

Go:

```
func canReachCorner(xCorner int, yCorner int, circles [][]int) bool {

}
```

Kotlin:

```
class Solution {
fun canReachCorner(xCorner: Int, yCorner: Int, circles: Array<IntArray>):
Boolean {
```

```
}
}
```

Swift:

```
class Solution {
func canReachCorner(_ xCorner: Int, _ yCorner: Int, _ circles: [[Int]]) ->
Bool {

}
}
```

Rust:

```
impl Solution {
pub fn can_reach_corner(x_corner: i32, y_corner: i32, circles: Vec<Vec<i32>>)
-> bool {

}
}
```

Ruby:

```
# @param {Integer} x_corner
# @param {Integer} y_corner
# @param {Integer[][]} circles
# @return {Boolean}
def can_reach_corner(x_corner, y_corner, circles)

end
```

PHP:

```
class Solution {

/**
* @param Integer $xCorner
* @param Integer $yCorner
* @param Integer[][] $circles
* @return Boolean
*/
```

```
function canReachCorner($xCorner, $yCorner, $circles) {

}
}
```

Dart:

```
class Solution {
bool canReachCorner(int xCorner, int yCorner, List<List<int>> circles) {

}
}
```

Scala:

```
object Solution {
def canReachCorner(xCorner: Int, yCorner: Int, circles: Array[Array[Int]]):
Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_reach_corner(x_corner :: integer, y_corner :: integer, circles ::
[[integer]]) :: boolean
def can_reach_corner(x_corner, y_corner, circles) do

end
end
```

Erlang:

```
-spec can_reach_corner(XCorner :: integer(), YCorner :: integer(), Circles ::
[[integer()]]) -> boolean().
can_reach_corner(XCorner, YCorner, Circles) ->
.
```

Racket:

```
(define/contract (can-reach-corner xCorner yCorner circles)
(-> exact-integer? exact-integer? (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if the Rectangle Corner Is Reachable
* Difficulty: Hard
* Tags: array, graph, math, search
*
* 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 canReachCorner(int xCorner, int yCorner, vector<vector<int>>& circles) {

}
};
```

Java Solution:

```
/**
* Problem: Check if the Rectangle Corner Is Reachable
* Difficulty: Hard
* Tags: array, graph, math, search
*
* 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 canReachCorner(int xCorner, int yCorner, int[][] circles) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Check if the Rectangle Corner Is Reachable
Difficulty: Hard
Tags: array, graph, math, search

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 canReachCorner(self, xCorner: int, yCorner: int, circles:
List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def canReachCorner(self, xCorner, yCorner, circles):
"""
:type xCorner: int
:type yCorner: int
:type circles: List[List[int]]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Check if the Rectangle Corner Is Reachable
* Difficulty: Hard
* Tags: array, graph, math, search
*
* 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} xCorner
* @param {number} yCorner
* @param {number[][]} circles
* @return {boolean}
*/
var canReachCorner = function(xCorner, yCorner, circles) {

};
```

TypeScript Solution:

```
/**
* Problem: Check if the Rectangle Corner Is Reachable
* Difficulty: Hard
* Tags: array, graph, math, search
*
* 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 canReachCorner(xCorner: number, yCorner: number, circles:
number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Check if the Rectangle Corner Is Reachable
* Difficulty: Hard
* Tags: array, graph, math, search
*
* 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 CanReachCorner(int xCorner, int yCorner, int[][] circles) {

}
}
```

C Solution:

```
/*
* Problem: Check if the Rectangle Corner Is Reachable
* Difficulty: Hard
* Tags: array, graph, math, search
*
* 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 canReachCorner(int xCorner, int yCorner, int** circles, int circlesSize,
int* circlesColSize) {

}
```

Go Solution:

```
// Problem: Check if the Rectangle Corner Is Reachable
// Difficulty: Hard
// Tags: array, graph, math, search
//
// 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 canReachCorner(xCorner int, yCorner int, circles [][]int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canReachCorner(xCorner: Int, yCorner: Int, circles: Array<IntArray>):
```

```
Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canReachCorner(_ xCorner: Int, _ yCorner: Int, _ circles: [[Int]]) ->
Bool {

}
}
```

Rust Solution:

```
// Problem: Check if the Rectangle Corner Is Reachable
// Difficulty: Hard
// Tags: array, graph, math, search
//
// 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 can_reach_corner(x_corner: i32, y_corner: i32, circles: Vec<Vec<i32>>)
-> bool {

}
}
```

Ruby Solution:

```
# @param {Integer} x_corner
# @param {Integer} y_corner
# @param {Integer[][]} circles
# @return {Boolean}
def can_reach_corner(x_corner, y_corner, circles)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $xCorner
* @param Integer $yCorner
* @param Integer[][] $circles
* @return Boolean
*/
function canReachCorner($xCorner, $yCorner, $circles) {

}
}
```

Dart Solution:

```
class Solution {
bool canReachCorner(int xCorner, int yCorner, List<List<int>> circles) {

}
}
```

Scala Solution:

```
object Solution {
def canReachCorner(xCorner: Int, yCorner: Int, circles: Array[Array[Int]]):
Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_reach_corner(x_corner :: integer, y_corner :: integer, circles ::
[[integer]]) :: boolean
def can_reach_corner(x_corner, y_corner, circles) do

end
end
```

Erlang Solution:

```
-spec can_reach_corner(XCorner :: integer(), YCorner :: integer(), Circles ::
[[integer()]]) -> boolean().
can_reach_corner(XCorner, YCorner, Circles) ->
.
```

Racket Solution:

```
(define/contract (can-reach-corner xCorner yCorner circles)
(-> exact-integer? exact-integer? (listof (listof exact-integer?)) boolean?)
)
```