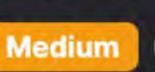
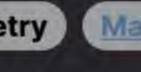
# 1401. Circle and Rectangle Overlapping









Math

## **Problem Description**

You are given a circle and an axis-aligned rectangle. The circle is defined by its radius and the coordinates of its center (xcenter, yCenter). The rectangle is defined by the coordinates of its bottom-left corner (x1, y1) and its top-right corner (x2, y2). Your task is to determine if there exists at least one point that is inside both the given circle and rectangle. The problem is asking you to return true if such a point exists (meaning the circle and rectangle overlap), and false otherwise.

Leetcode Link

# Intuition

Any corner of the rectangle is inside the circle.

To solve this problem, let's first understand the scenarios where a circle and a rectangle can overlap:

- 2. Any part of the circle's circumference intersects the rectangle. 3. The circle is entirely inside the rectangle, or vice versa.
- However, checking every point on the circumference directly would be impractical. The key insight here is that the closest point on

to the circle's center, greatly simplifying the problem.

the rectangle to the circle's center can indicate whether they overlap without checking every point. If this closest point is also inside the circle, then they overlap. The solution code uses a helper function f to find the closest point on the rectangle to a given point (xcenter, ycenter). The

function f calculates the minimum distance along the x or y dimension from the rectangle to the center of the circle. If the center of

the circle is within the bounds of the rectangle along one axis, then the minimum distance is 0 for that axis. Otherwise, f returns the distance to the closest edge of the rectangle along that axis. Once we calculate the minimum distances a and b for the x and y axes, the point on the rectangle closest to the circle's center is at (xCenter + a, yCenter + b). If that point is within the circle (meaning the sum of squares a \* a + b \* b is less than or equal to

radius \* radius), then we have an overlap, and the function returns true. If the closest point is outside the circle, the function returns false. This approach effectively checks whether the rectangle and the circle overlap by considering only the closest point on the rectangle

Solution Approach

## comparing them to the circle's radius. Here's how it's implemented:

1. Function Declaration: The Solution class contains a method checkOverlap which takes as its arguments the circle's radius and its center coordinates, along with the coordinates of the rectangle's bottom-left and top-right corners.

The solution employs a straightforward geometric approach, converting the overlap problem into one of measuring distances and

- 2. Distance Function (f): The method uses a nested helper function f, which is designed to calculate the shortest one-dimensional distance from a point (either xCenter or yCenter) to an interval defined by two bounds (either x1, x2 or y1, y2). The bounds represent the sides of the rectangle along one axis.
- It works as follows: if the center coordinate (for one dimension) is between the rectangle's edge coordinates, then the circle's center is directly above or alongside the edge (when projected onto that axis), and thus the shortest distance is 0. If the circle's center is outside the rectangle along that axis, the function returns the distance to the nearest edge.

3. Finding the Closest Point: The checkOverlap method then calls f for both the x and y dimensions to find the x-distance (a) and

y-distance (b) from the circle's center to the rectangle's closest edge or point. 4. Checking Overlap: After the closest point is identified by its x and y distances to the circle's center, we check if this point is

inside the circle. The point lies inside if the sum of the squares of the distances a \* a + b \* b does not exceed the square of

- 5. Returning the Result: Finally, the checkOverlap method returns true if the closest point is inside the circle (hence, an overlap) or false otherwise.
- looping or complex structures are necessary, making the solution very efficient in terms of time complexity, which is O(1), as the calculations are done in a fixed number of steps regardless of input size.

This algorithm is particularly effective because it reduces a 2D problem into a comparative evaluation of calculated 1D distances. No

Example Walkthrough Let's go through a small example to illustrate the solution approach.

### Radius of the circle (radius): 2 Coordinates of the circle's center (xCenter, yCenter): (2, 2)

Coordinates of the bottom-left corner of the rectangle (x1, y1): (3, 1)

Coordinates of the top-right corner of the rectangle (x2, y2): (5, 3)

Assume we are given the following inputs:

the radius of the circle radius \* radius.

- Now, we need to determine if at least one point on the rectangle is also inside the circle.
- Following the solution approach:
- o In this case, the xCenter is 2, and the x bounds of the rectangle are 3 and 5. Since the xCenter is less than x1, the shortest

x-distance to the rectangle is x1 - xCenter, which is 3 - 2 = 1.

alongside the rectangle on the y-axis, so the shortest y-distance is 0.

• The yCenter is 2, and the y bounds are 1 and 3. Since the yCenter is within these bounds, the circle's center is directly

2. Next, we use the helper function f to calculate the closest x-distance (a) from the circle's center to the rectangle's closest edge.

1. We start by preparing to invoke the <a href="mailto:check0verlap">check0verlap</a> method from the <a href="mailto:solution">Solution</a> class, which will take our given inputs.

3. We apply the same logic to find the closest y-distance (b) from the circle's center to the rectangle's closest edge.

- 4. With the closest x-distance (a) as 1 and the closest y-distance (b) as 0, we now determine whether the closest point on the rectangle to the circle's center is inside the circle.
- The square of the circle's radius is radius \* radius or 2 \* 2 = 4. 5. Since 1 (the sum of squares of the distances) is less than 4 (the square of the radius), the closest point on the rectangle to the

6. The checkOverlap method would return true for these inputs, indicating that the rectangle and the circle do indeed overlap.

This example shows that you do not need to check every point on the rectangle or circle to see if there is an overlap. Instead, by cleverly using the closest point and comparing distance squared, we can efficiently solve the problem with a constant time

We calculate a \* a + b \* b which equals 1 \* 1 + 0 \* 0 = 1.

circle's center is within the circle. Therefore, there is an overlap.

# along one dimension (either 'x' or 'y').

if min\_edge <= center <= max\_edge:</pre>

return min\_edge - center

distance\_y = distance\_to\_edge(y1, y2, y\_center)

# the circle overlaps with the rectangle.

return 0

if center < min\_edge:</pre>

def distance\_to\_edge(min\_edge: int, max\_edge: int, center: int) -> int:

# Check if the sum of squares of the distances is less than or equal to

// Method to check if a circle overlaps with an axis-aligned rectangle

int deltaX = shortestDistanceToSegment(x1, x2, xCenter);

int deltaY = shortestDistanceToSegment(y1, y2, yCenter);

if (segmentStart <= center && center <= segmentEnd) {</pre>

// If it is, the circle and rectangle overlap.

return deltaX \* deltaX + deltaY \* deltaY <= radius \* radius;</pre>

return deltaX \* deltaX + deltaY \* deltaY <= radius \* radius;</pre>

# the square of the radius. According to the Pythagorean theorem, this means

return distance\_x \* distance\_x + distance\_y \* distance\_y <= radius \* radius</pre>

private int shortestDistanceToSegment(int segmentStart, int segmentEnd, int center) {

public boolean checkOverlap(int radius, int xCenter, int yCenter, int x1, int y1, int x2, int y2) {

// If the center is before the start, calculate the distance from the center to the start

// Find the shortest distance from the circle's center to the rectangle's border along the x axis

// Find the shortest distance from the circle's center to the rectangle's border along the y axis

// Check if the combined distance of both axes' deltas is within the radius (using the Pythagorean theorem)

// If the center is between the start and end points of the segment, it means the shortest distance is 0

// Helper method to find the shortest distance from a point 'center' to a line segment defined by 'segmentStart' and 'segmentEnd'

# If center is less than the minimum edge, return distance to the minimum edge.

# If center is between the edges, the distance is 0.

- Python Solution
- x1: int, y1: int, x2: int, 10 y2: int ) -> bool: 11 12 # Helper function to calculate distance from center to rectangle edge

```
21
               # If center is greater than the maximum edge, return distance to the maximum edge.
22
               return center - max_edge
23
24
           # Calculate the distance from the center of the circle to the closest point
25
           # of the rectangle along the x-axis.
26
           distance_x = distance_to_edge(x1, x2, x_center)
27
           # Calculate the distance from the center of the circle to the closest point
28
           # of the rectangle along the y-axis.
```

Java Solution

class Solution {

algorithm.

13

14

15

16

17

18

19

20

29

30

31

32

33

34

35

10

11

12

13

14

15

16

17

18

19

class Solution:

def checkOverlap(

radius: int,

x\_center: int,

y\_center: int,

self,

### 20 if (center < segmentStart) {</pre> return segmentStart - center; 22 23 // If the center is past the end, calculate the distance from the center to the end 24 return center - segmentEnd;

return 0;

```
25
26 }
27
C++ Solution
 1 class Solution {
2 public:
       // Function to check if the circle with a given radius and center overlaps with a given rectangle.
       // The rectangle is defined by its bottom-left (x1, y1) and top-right (x2, y2) coordinates.
       bool checkOverlap(int radius, int xCenter, int yCenter, int x1, int y1, int x2, int y2) {
 6
           // Helper function to calculate the smallest distance from the center of the circle
           // to the edge of the rectangle along one axis (either x or y).
           // It takes the minimum and maximum edge coordinate values along one axis
           // and the center's coordinate along the same axis.
10
           auto getMinimumDistance = [](int minEdge, int maxEdge, int centerCoord) -> int {
11
12
               if (centerCoord >= minEdge && centerCoord <= maxEdge) {</pre>
13
                   // The center is inside the rectangle along this axis, so distance is 0.
14
                   return 0;
15
16
               // The center is outside the rectangle; calculate the distance to the closer edge.
                return centerCoord < minEdge ? minEdge - centerCoord : centerCoord - maxEdge;</pre>
17
           };
18
19
20
           // Calculate the smallest distances from the circle's center to the rectangle's edges
21
           // along the x and y axes.
22
           int deltaX = getMinimumDistance(x1, x2, xCenter);
23
           int deltaY = getMinimumDistance(y1, y2, yCenter);
24
25
           // Check if the sum of squares of smallest distances is less than or equal to the square of the radius.
```

### // Function to calculate the minimum distance between a point and a projected point on a line segment function calculateMinDistance(lesserCoordinate: number, greaterCoordinate: number, pointCoordinate: number): number { // If the point is between the coordinates, distance is zero (point is within the line segment)

Typescript Solution

26

27

28

30

29 };

```
if (pointCoordinate >= lesserCoordinate && pointCoordinate <= greaterCoordinate) {</pre>
           return 0;
       // If point is before the start of the line segment, return the distance from start
       if (pointCoordinate < lesserCoordinate) {</pre>
           return lesserCoordinate - pointCoordinate;
9
10
       // If point is after the end of the line segment, return the distance from the end
11
       return pointCoordinate - greaterCoordinate;
12
13 }
14
   // Function to check if a circle overlaps with an axis-aligned rectangle
   function checkOverlap(
                            // Circle's radius
       radius: number,
       circleXCenter: number, // Circle's X-coordinate center
       circleYCenter: number, // Circle's Y-coordinate center
       rectX1: number,
                                // Rectangle's top-left X-coordinate
21
       rectY1: number,
                                // Rectangle's top-left Y-coordinate
22
       rectX2: number,
                                // Rectangle's bottom-right X-coordinate
                                // Rectangle's bottom-right Y-coordinate
       rectY2: number
   ): boolean {
       // Calculate the distance from the circle's center to the closest point on the X-axis of the rectangle
25
       const deltaX = calculateMinDistance(rectX1, rectX2, circleXCenter);
26
       // Calculate the distance from the circle's center to the closest point on the Y-axis of the rectangle
27
       const deltaY = calculateMinDistance(rectY1, rectY2, circleYCenter);
28
29
       // Check if the square of the distances is less than or equal to the square of the radius
30
       // If this condition is true, the circle and rectangle overlap
       return deltaX * deltaX + deltaY * deltaY <= radius * radius;</pre>
31
33
Time and Space Complexity
```

## The checkOverlap function consists of a nested helper f function. The operations inside the f function are simple arithmetic and conditional operations. These operations take constant time. Since there are no loops or recursive calls, the overall time complexity of the function is 0(1), which means it executes in constant time regardless of the size of the input parameters.

# Space Complexity: 0(1)

Time Complexity: 0(1)

The space complexity is also 0(1) because the function only uses a fixed amount of additional space. The helper function f does not utilize additional memory that scales with the input size, and no extra data structures are employed. The variables a and b are the only additional memory used, and their space requirement does not depend on the size of the input.