1401. Circle and Rectangle Overlapping

Medium Geometry 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.

Intuition

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

1. Any corner of the rectangle is inside the circle.
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 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 to the circle's center, greatly simplifying the problem.

Solution Approach

The solution employs a straightforward geometric approach, converting the overlap problem into one of measuring distances and 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.
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 the radius of the circle `radius * radius`.
5. **Returning the Result:** Finally, the `checkOverlap` method returns `true` if the closest point is inside the circle (hence, an overlap) or `false` otherwise.

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

Example Walkthrough

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

Assume we are given the following inputs:

- 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)

Now, we need to determine if at least one point on the rectangle is also inside the circle.

Following the solution approach:

1. We start by preparing to invoke the `checkOverlap` method from the `Solution` class, which will take our given inputs.
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.
   - 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`.
3. We apply the same logic to find the closest y-distance (`b`) from the circle's center to the rectangle's closest edge.
   - 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 alongside the rectangle on the y-axis, so the shortest y-distance is `0`.
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.
   - We calculate `a * a + b * b` which equals `1 * 1 + 0 * 0 = 1`.
   - 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 circle's center is within the circle. Therefore, there is an overlap.
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 algorithm.

Solution Implementation

Python

```python
class Solution:
    def checkOverlap(
        self,
        radius: int,
        x_center: int,
        y_center: int,
        x1: int,
        y1: int,
        x2: int,
        y2: int,
    ) -> bool:
        # Helper function to calculate distance from center to rectangle edge
        # along one dimension (either 'x' or 'y').
        def distance_to_edge(min_edge: int, max_edge: int, center: int) -> int:
            # If center is between the edges, the distance is 0.
            if min_edge <= center <= max_edge:
                return 0
            # If center is less than the minimum edge, return distance to the minimum edge.
            if center < min_edge:
                return min_edge - center
            # If center is greater than the maximum edge, return distance to the maximum edge.
            return center - max_edge

        # Calculate the distance from the center of the circle to the closest point
        # of the rectangle along the x-axis.
        distance_x = distance_to_edge(x1, x2, x_center)
        # Calculate the distance from the center of the circle to the closest point
        # of the rectangle along the y-axis.
        distance_y = distance_to_edge(y1, y2, y_center)

        # Check if the sum of squares of the distances is less than or equal to
        # the square of the radius. According to the Pythagorean theorem, this means
        # the circle overlaps with the rectangle.
        return distance_x * distance_x + distance_y * distance_y <= radius * radius
```

Java

```java
class Solution {

    // Method to check if a circle overlaps with an axis-aligned rectangle
    public boolean checkOverlap(int radius, int xCenter, int yCenter, int x1, int y1, int x2, int y2) {
        // Find the shortest distance from the circle's center to the rectangle's border along the x axis
        int deltaX = getMinimumDistance(x1, x2, xCenter);
        // Find the shortest distance from the circle's center to the rectangle's border along the y axis
        int deltaY = getMinimumDistance(y1, y2, yCenter);
        // Check if the combined distance of both axes' deltas is within the radius (using the Pythagorean theorem)
        return deltaX * deltaX + deltaY * deltaY <= radius * radius;
    }

    // Helper method to find the shortest distance from a point 'center' to a line segment defined by 'segmentStart' and 'segmentEnd'
    private int shortestDistanceToSegment(int segmentStart, int segmentEnd, int center) {
        // If the center is between the start and end points of the segment, it means the shortest distance is 0
        if (segmentStart <= center && center <= segmentEnd) {
            return 0;
        }
        // If the center is before the start, calculate the distance from the center to the start
        if (center < segmentStart) {
            return segmentStart - center;
        }
        // If the center is past the end, calculate the distance from the center to the end
        return center - segmentEnd;
    }
}
```

C++

```cpp
class Solution {
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) {

        // 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.
        auto getMinimumDistance = [](int minEdge, int maxEdge, int centerCoord) -> int {
            if (centerCoord <= minEdge && centerCoord <= maxEdge) {
                // The center is inside the rectangle along this axis, so distance is 0.
                return 0;
            }
            // The center is outside the rectangle: calculate the distance to the closer edge.
            return centerCoord < minEdge ? minEdge - centerCoord : centerCoord - maxEdge;
        };

        // Calculate the smallest distances from the circle's center to the rectangle's edges
        // along the x and y axes.
        int deltaX = getMinimumDistance(x1, x2, xCenter);
        int deltaY = getMinimumDistance(y1, y2, yCenter);

        // Check if the sum of squares of smallest distances is less than or equal to the square of the radius.
        // If it is, the circle and rectangle overlap.
        return deltaX * deltaX + deltaY * deltaY <= radius * radius;
    }
};
```

TypeScript

```typescript
// 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)
    if (pointCoordinate >= lesserCoordinate && pointCoordinate <= greaterCoordinate) {
        return 0;
    }
    // If point is before the start of the line segment, return the distance from the start
    if (pointCoordinate < lesserCoordinate) {
        return lesserCoordinate - pointCoordinate;
    }
    // If point is after the end of the line segment, return the distance from the end
    return pointCoordinate - greaterCoordinate;
}

// Function to check if a circle overlaps with an axis-aligned rectangle
function checkOverlap(
    radius: number,         // Circle's radius
    circleXCenter: number,  // Circle's X-coordinate center
    circleYCenter: number,  // Circle's Y-coordinate center
    rectX1: number,         // Rectangle's top-left X-coordinate
    rectY1: number,         // Rectangle's top-left Y-coordinate
    rectX2: number,         // Rectangle's bottom-right X-coordinate
    rectY2: number          // Rectangle's bottom-right Y-coordinate
): boolean {
    // Calculate the distance from the circle's center to the closest point on the X-axis of the rectangle
    const deltaX = calculateMinDistance(rectX1, rectX2, circleXCenter);
    // Calculate the distance from the circle's center to the closest point on the Y-axis of the rectangle
    const deltaY = calculateMinDistance(rectY1, rectY2, circleYCenter);
    // Check if the square of the distances is less than or equal to the square of the radius
    // If this condition is true, the circle and rectangle overlap
    return deltaX * deltaX + deltaY * deltaY <= radius * radius;
}


class Solution:
    def checkOverlap(
        self,
        radius: int,
        x_center: int,
        y_center: int,
        x1: int,
        y1: int,
        x2: int,
        y2: int,
    ) -> bool:
        # Helper function to calculate distance from center to rectangle edge
        # along one dimension (either 'x' or 'y').
        def distance_to_edge(min_edge: int, max_edge: int, center: int) -> int:
            # If center is between the edges, the distance is 0.
            if min_edge <= center <= max_edge:
                return 0
            # If center is less than the minimum edge, return distance to the minimum edge.
            if center < min_edge:
                return min_edge - center
            # If center is greater than the maximum edge, return distance to the maximum edge.
            return center - max_edge

        # Calculate the distance from the center of the circle to the closest point
        # of the rectangle along the x-axis.
        distance_x = distance_to_edge(x1, x2, x_center)
        # Calculate the distance from the center of the circle to the closest point
        # of the rectangle along the y-axis.
        distance_y = distance_to_edge(y1, y2, y_center)

        # Check if the sum of squares of the distances is less than or equal to
        # the square of the radius. According to the Pythagorean theorem, this means
        # the circle overlaps with the rectangle.
        return distance_x * distance_x + distance_y * distance_y <= radius * radius
```

Time and Space Complexity

Time Complexity: `O(1)`

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 `O(1)`, which means it executes in constant time regardless of the size of the input parameters.

Space Complexity: `O(1)`

The space complexity is also `O(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.