223. Rectangle Area

Medium Geometry Math

Problem Description

The problem is to calculate the total area covered by two rectilinear rectangles in a 2D plane. We are given the coordinates of the bottom-left and top-right corners for both rectangles. The coordinates for the first rectangle are `(ax1, ay1)` for the bottom-left corner and `(ax2, ay2)` for the top-right corner. Similarly, the second rectangle is defined by `(bx1, by1)` for the bottom-left corner and `(bx2, by2)` for the top-right corner. The rectangles have sides parallel to the x and y axes, which means they are aligned vertically and horizontally without any rotation.

The task is to find the sum of the areas of both rectangles minus any overlapping area.

Intuition

To solve this problem, we first calculate the areas of both rectangles independently. To find the area of a rectangle, we multiply the width (`x2 - x1`) by the height (`y2 - y1`). However, if the rectangles overlap, we need to avoid double-counting the shared area.

We determine the overlapping region by comparing the rectangles' edges. The width of the overlap is the smaller of the widths of the endings (`min(ax2, bx2)`) minus the larger of the starts (`max(ax1, bx1)`), and similarly for the height with respect to the y-coordinates.

The crucial insight is that if there is no overlap, either the width or the height (or both) of the overlapping area will be negative, since one rectangle's start will be greater than the other's end. Since we cannot have a negative area for rectangles, which would imply no overlap, we take the maximum of the computed width and height with 0.

The total area covered by the two rectangles is then the sum of the areas of both rectangles minus the area of the overlap, which is the product of the overlapping width and height (only subtracted if they are positive).

Solution Approach

The solution uses a straightforward approach leveraging arithmetic calculations without the need for additional data structures or complex algorithms. The process can be broken down into the following steps:

1. Calculate the area of both rectangles separately:
   - For the first rectangle, the area `a` is computed as `(ax2 - ax1) * (ay2 - ay1)`.
   - Similarly, for the second rectangle, the area `b` is computed as `(bx2 - bx1) * (by2 - by1)`.
2. Determine the dimensions of the overlap (if any):
   - Compute the overlap width as the difference between the minimum of the right sides (`min(ax2, bx2)`) and the maximum of the left sides (`max(ax1, bx1)`).
   - Similarly, compute the overlap height as the difference between the minimum of the top sides (`min(ay2, by2)`) and the maximum of the bottom sides (`max(ay1, by1)`).
   - If the rectangles don't overlap, either the calculated width or height (or both) will be negative, which implies that there is no overlap.
3. Calculate the area of the overlapping region if it exists:
   - The area of the overlap should only be considered if both width and height are positive.
   - The overlap area is calculated by multiplying the positive values of the computed width and height: `max(height, 0) * max(width, 0)`.
   - Using `max` function ensures that a negative width or height (indicating no overlap) results in zero overlapping area.
4. Combine the results to get the total area covered by the rectangles:
   - Add the area of both rectangles: `a + b`.
   - Subtract the overlap area only if it is positive, which is ensured by the previous step.

The final result, which is the total area covered by the two rectangles, is thus calculated by `a + b - max(height, 0) * max(width, 0)`.

No additional data structures or complex algorithms are necessary because the solution relies solely on conditional arithmetic. This approach is efficient as it calculates the required value with a constant number of arithmetic operations and in constant time complexity, i.e., `O(1)`.

Example Walkthrough

Let us illustrate the solution approach with a small example:

Assume we have two rectangles. The first rectangle A has coordinates of its bottom-left and top-right corners as `(1, 2)` and `(5, 5)`, respectively. The second rectangle B has corners at `(3, 1)` and `(7, 4)`.

Step 1: Calculate the area of both rectangles separately:

- Area of rectangle A (`a`): `(5 - 1) * (5 - 2) = 4 * 3 = 12`
- Area of rectangle B (`b`): `(7 - 3) * (4 - 1) = 4 * 3 = 12`

Step 2: Determine the dimensions of the overlap:

- Overlap width: `min(ax2, bx2) - max(ax1, bx1) = min(5, 7) - max(1, 3) = 5 - 3 = 2`
- Overlap height: `min(ay2, by2) - max(ay1, by1) = min(5, 4) - max(2, 1) = 4 - 2 = 2`

Step 3: Calculate the area of the overlapping region if it exists:

- Since both the overlap width and height are positive, we calculate the overlap area:
- Overlap area: `max(height, 0) * max(width, 0) = max(2, 0) * max(2, 0) = 2 * 2 = 4`

Step 4: Combine the results to get the total area covered by the rectangles:

- Sum of both rectangles' areas: `a + b = 12 + 12 = 24`
- Subtract the overlap area (positive) to avoid double counting: `24 - 4 = 20`

The total area covered by the two rectangles is 20 square units. This example demonstrates the efficient use of conditional arithmetic to handle potential overlaps in the calculation of combined rectangular areas.

Solution Implementation

Python

```python
class Solution:
    def computeArea(self, A_x1: int, A_y1: int, A_x2: int, A_y2: int,
                    B_x1: int, B_y1: int, B_x2: int, B_y2: int) -> int:
        # Area of the first rectangle (A)
        area_A = (A_x2 - A_x1) * (A_y2 - A_y1)

        # Area of the second rectangle (B)
        area_B = (B_x2 - B_x1) * (B_y2 - B_y1)

        # Computing the width of the overlapping area
        # The overlap width is the smaller of the rightmost x minus the larger of the leftmost x
        overlap_width = min(A_x2, B_x2) - max(A_x1, B_x1)

        # Computing the height of the overlapping area
        # The overlap height is the smaller of the topmost y minus the larger of the bottommost y
        overlap_height = min(A_y2, B_y2) - max(A_y1, B_y1)

        # Computing the area of overlapping if both width and height are positive (there is an overlap)
        overlap_area = max(overlap_width, 0) * max(overlap_height, 0)

        # Total combined area is the sum of both rectangle areas minus the overlapping area
        return area_A + area_B - overlap_area
```

Java

```java
class Solution {
    public int computeArea(int A_x1, int A_y1, int A_x2, int A_y2,
                           int B_x1, int B_y1, int B_x2, int B_y2) {
        // Calculate the area of the first rectangle
        int areaA = (A_x2 - A_x1) * (A_y2 - A_y1);

        // Calculate the area of the second rectangle
        int areaB = (B_x2 - B_x1) * (B_y2 - B_y1);

        // Calculate the width of the overlapping area
        int overlapWidth = Math.min(A_x2, B_x2) - Math.max(A_x1, B_x1);

        // Calculate the height of the overlapping area
        int overlapHeight = Math.min(A_y2, B_y2) - Math.max(A_y1, B_y1);

        // Ensure that the overlap width and height are non-negative
        overlapWidth = Math.max(overlapWidth, 0);
        overlapHeight = Math.max(overlapHeight, 0);

        // Calculate the area of the overlapping region
        int overlapArea = overlapWidth * overlapHeight;

        // Combine the areas of both rectangles and subtract the overlapping area
        return areaA + areaB - overlapArea;
    }
}
```

C++

```cpp
class Solution {
public:
    int computeArea(int ax1, int ay1, int ax2, int ay2,
                    int bx1, int by1, int bx2, int by2) {
        // Compute the area of the first rectangle: AreaA
        int areaA = (ax2 - ax1) * (ay2 - ay1);

        // Compute the area of the second rectangle: AreaB
        int areaB = (bx2 - bx1) * (by2 - by1);

        // Calculate the width of the overlapping area
        int overlapWidth = std::min(ax2, bx2) - std::max(ax1, bx1);

        // Calculate the height of the overlapping area
        int overlapHeight = std::min(ay2, by2) - std::max(ay1, by1);

        // Calculate the area of the overlapping region, guarding against no overlap scenario
        // If no overlap, result is zero (overlapWidth or overlapHeight could be negative)
        int overlapArea = std::max(overlapHeight, 0) * std::max(overlapWidth, 0);

        // Return the sum of both areas minus the overlapping area
        return areaA + areaB - overlapArea;
    }
};
```

TypeScript

```typescript
/**
 * Computes the total area covered by two rectangles in a 2D plane.
 * The rectangles are aligned with the x and y axes.
 *
 * @param {number} ax1 - The x-coordinate of the bottom-left corner of the first rectangle.
 * @param {number} ay1 - The y-coordinate of the bottom-left corner of the first rectangle.
 * @param {number} ax2 - The x-coordinate of the upper-right corner of the first rectangle.
 * @param {number} ay2 - The y-coordinate of the upper-right corner of the first rectangle.
 * @param {number} bx1 - The x-coordinate of the bottom-left corner of the second rectangle.
 * @param {number} by1 - The y-coordinate of the bottom-left corner of the second rectangle.
 * @param {number} bx2 - The x-coordinate of the upper-right corner of the second rectangle.
 * @param {number} by2 - The y-coordinate of the upper-right corner of the second rectangle.
 * @return {number} - The combined area of both rectangles.
 */
function computeArea(
    ax1: number,
    ay1: number,
    ax2: number,
    ay2: number,
    bx1: number,
    by1: number,
    bx2: number,
    by2: number,
): number {
    // Area of the first rectangle
    const areaOfFirstRectangle = (ax2 - ax1) * (ay2 - ay1);
    // Area of the second rectangle
    const areaOfSecondRectangle = (bx2 - bx1) * (by2 - by1);

    // Calculate the width of the overlapping area
    const overlapWidth = Math.min(ax2, bx2) - Math.max(ax1, bx1);
    // Calculate the height of the overlapping area
    const overlapHeight = Math.min(ay2, by2) - Math.max(ay1, by1);

    // Calculate the area of the overlapping region; if there is no overlap,
    // this would be zero as Math.max would handle negative values.
    const overlapArea = Math.max(overlapWidth, 0) * Math.max(overlapHeight, 0);

    // The total area is the sum of the areas of both rectangles minus the overlap area
    return areaOfFirstRectangle + areaOfSecondRectangle - overlapArea;
}
```


```python
class Solution:
    def computeArea(self, A_x1: int, A_y1: int, A_x2: int, A_y2: int,
                    B_x1: int, B_y1: int, B_x2: int, B_y2: int) -> int:
        # Area of the first rectangle (A)
        area_A = (A_x2 - A_x1) * (A_y2 - A_y1)

        # Area of the second rectangle (B)
        area_B = (B_x2 - B_x1) * (B_y2 - B_y1)

        # Computing the width of the overlapping area
        # The overlap width is the smaller of the rightmost x minus the larger of the leftmost x
        overlap_width = min(A_x2, B_x2) - max(A_x1, B_x1)

        # Computing the height of the overlapping area
        # The overlap height is the smaller of the topmost y minus the larger of the bottommost y
        overlap_height = min(A_y2, B_y2) - max(A_y1, B_y1)

        # Computing the area of overlapping if both width and height are positive (there is an overlap)
        overlap_area = max(overlap_width, 0) * max(overlap_height, 0)

        # Total combined area is the sum of both rectangle areas minus the overlapping area
        return area_A + area_B - overlap_area
```


Time and Space Complexity

Time Complexity

The time complexity of the given code is `O(1)`. This is because all operations performed to calculate the areas of rectangles and the overlapping region take a constant amount of time. Each operation involves basic arithmetic calculations that do not depend on the size of the input.

Space Complexity

The space complexity of the code is also `O(1)`. The amount of memory used does not increase with the size of the input, as there are only a fixed number of integer variables assigned (`a`, `b`, `width`, and `height`) and no use of any data structures that could scale with input size.