●●●● Rare 0/6

Rectangle Geometry

Authors: Darren Yao, Michael Cao, Benjamin Qi, Ben Dodge
Contributors: Allen Li, Andrew Wang, Dong Liu, Ryan Chou

Problems involving rectangles whose sides are parallel to the coordinate axes.

Language: Python Edit This Page

Prerequisites

- Bronze - Time Complexity

TABLE OF CONTENTS

RESOURCES

IUSACO	**7.1 - Rectangle Geometry**	module is based off this

Most problems in this category include only two or three squares or rectangles, in which case you can simply draw out cases on paper. This should logically lead to a solution.

Example - Fence Painting

Fence Painting
Bronze - Easy

Focus Problem – try your best to solve this problem before continuing!

Slow Solution

Since all the intervals lie between the range, $[0, 100]$, we can mark each interval of length 1 contained within each interval as painted using a loop. Then the answer will be the number of marked intervals.

Time Complexity: $\mathcal{O}(\text{max coordinate})$

Copy PYTHON

```python
import sys

MAX_POS = 100

sys.stdin = open("paint.in", "r")
sys.stdout = open("paint.out", "w")

a, b = map(int, input().split())
c, d = map(int, input().split())

painted = [False for _ in range(MAX_POS + 1)]
# Add Farmer John's painted interval
for i in range(a, b):
    painted[i] = True
# Add Bessie's painted interval
for i in range(c, d):
    painted[i] = True

print(sum(painted))  # Print the total amount of intervals painted
```

However, this solution would not work for higher constraints (ex. if the coordinates were up to 10^9).

Fast Solution

Calculate the answer by adding the original lengths and subtracting the intersection length.

$$(b - a) + (d - c) - \text{intersection}([a, b], [c, d])$$

The **official analysis** splits computing the intersection length into several cases. However, we can do it in a simpler way. An interval $[x, x + 1]$ is contained within both $[a, b]$ and $[c, d]$ if $a \le x$, $c \le x$, $x < b$, and $x < d$, or in other words if, $\max(a, c) \le x$ and $x < \min(b, d)$. So the length of the intersection is $\min(b, d) - \max(a, c)$ if this quantity is positive and zero otherwise!

Time Complexity: $\mathcal{O}(1)$

Copy PYTHON

```python
import sys

sys.stdin = open("paint.in", "r")
sys.stdout = open("paint.out", "w")

a, b = map(int, input().split())
c, d = map(int, input().split())
total = (b - a) + (d - c)  # the sum of the two intervals
intersection = max(min(b, d) - max(a, c), 0)  # subtract the intersection
union = total - intersection

print(union)
```

Example - Blocked Billboard

Think of this as the 2D analog of the previous example.

Blocked Billboard

Bronze - Normal

Focus Problem – try your best to solve this problem before continuing!

Slow Solution

Time Complexity: $\mathcal{O}((\max \text{coordinate})^2)$

Since all coordinates are in the range $[-1000, 1000]$, we can simply go through each of the 2000^2 possible visible squares and check which ones are visible using nested for loops.

Code runs marginally faster in Python **when placed in a function**, so we can use this to get all 10 test cases. When taken out of the function, the code passes only 9 test cases.

Copy PYTHON

```python
import sys

MAX_POS = 2000


def main():
    sys.stdin = open("billboard.in", "r")
    sys.stdout = open("billboard.out", "w")

    visible = [[False for _ in range(MAX_POS)] for _ in range(MAX_POS)]

    for i in range(3):
        x1, y1, x2, y2 = map(int, input().split())
        x1 += MAX_POS // 2
        y1 += MAX_POS // 2
        x2 += MAX_POS // 2
        y2 += MAX_POS // 2
        # Mark billboard area as visible, truck as not visible
        for x in range(x1, x2):
            for y in range(y1, y2):
                visible[x][y] = i < 2

    # Count all visible billboard squares
    ans = 0
    for x in range(MAX_POS):
        for y in range(MAX_POS):
            ans += visible[x][y]
    print(ans)


main()
```

This wouldn't suffice if the coordinates were changed to be up to 10^9.

Fast Solution

Time Complexity: $\mathcal{O}(1)$

Official Analysis

Note how creating a class `Rect` to represent a rectangle makes the code easier to understand.

Copy PYTHON

```python
import sys


class Rect:
    def __init__(self):
        # Read rectangle coordinates from input
        self.x1, self.y1, self.x2, self.y2 = map(int, input().split())

    def area(self):
        # Calculate area of the rectangle
        return (self.y2 - self.y1) * (self.x2 - self.x1)


def intersect(p, q):
    # Calculate overlap in x direction
    x_overlap = max(0, min(p.x2, q.x2) - max(p.x1, q.x1))
    # Calculate overlap in y direction
    y_overlap = max(0, min(p.y2, q.y2) - max(p.y1, q.y1))
    return x_overlap * y_overlap  # Area of intersection


sys.stdin = open("billboard.in", "r")
sys.stdout = open("billboard.out", "w")

rects = []
for _ in range(3):
    rects.append(Rect())  # Read the two billboards and the truck

print(
    rects[0].area()  # Area of first billboard
    + rects[1].area()  # Area of second billboard
    - intersect(rects[0], rects[2])  # Subtract area of first billboard covered by truck
    - intersect(
        rects[1], rects[2]
    )  # Subtract area of second billboard covered by truck
)
```

Common Formulas

Warning!

This entire section can be skipped if you already understand the fast solution to Blocked Billboard.

Certain tasks show up often in rectangle geometry problems. For example, many problems involve finding the overlapping area of two or more rectangles based on their coordinate points, or determining whether two rectangles intersect. Here, we'll discuss these formulas.

Note that these formulas only apply to rectangles which have sides parallel to the coordinate axes.

Coordinates

A rectangle can be represented with 2 points: its top right corner and bottom left corner. We'll label these points tr (top right) and bl (bottom left).

In this module, we'll assume that increasing x moves to the right and increasing y moves up.

Finding area

The formula for finding the area of an individual rectangle is $w \cdot l$.

`length` is the length of the vertical sides, and `width` is the length of the horizontal sides.

1. $\texttt{width} = \texttt{tr}_x - \texttt{bl}_x$
2. $\texttt{length} = \texttt{tr}_y - \texttt{bl}_y$
3. $\texttt{area} = \texttt{width} \cdot \texttt{length}$

Implementation

Copy PYTHON

```python
def area(bl_x: int, bl_y: int, tr_x: int, tr_y: int) -> int:
    length = tr_y - bl_y
    width = tr_x - bl_x
    return length * width
```

Checking if two rectangles intersect

Given two rectangles a and b, there are only two cases where they do not intersect:

1. $\texttt{tr}_{a_y} \leq \texttt{bl}_{b_y}$ or $\texttt{bl}_{a_y} \geq \texttt{tr}_{b_y}$.
2. $\texttt{bl}_{a_x} \geq \texttt{tr}_{b_x}$ or $\texttt{tr}_{a_x} \leq \texttt{bl}_{b_x}$.

In all other cases, the rectangles intersect.

Implementation

Copy PYTHON

```python
def intersect(s1, s2) -> bool:
    bl_a_x, bl_a_y, tr_a_x, tr_a_y = s1[0], s1[1], s1[2], s1[3]
    bl_b_x, bl_b_y, tr_b_x, tr_b_y = s2[0], s2[1], s2[2], s2[3]

    # no overlap
    if bl_a_x >= tr_b_x or tr_a_x <= bl_b_x or bl_a_y >= tr_b_y or tr_a_y <= bl_b_y:
        return False
    else:
        return True
```

Finding area of intersection

We'll assume that the shape formed by the intersection of two rectangles is itself a rectangle.

First, we'll find this rectangle's length and width. $\texttt{width} = \min(\texttt{tr}_{a_x}, \texttt{tr}_{b_x}) - \max(\texttt{bl}_{a_x}, \texttt{bl}_{b_x})$. $\texttt{length} = \min(\texttt{tr}_{a_y}, \texttt{tr}_{b_y}) - \max(\texttt{bl}_{a_y}, \texttt{bl}_{b_y})$.

If either of these values are negative, the rectangles do not intersect. If they are zero, the rectangles intersect at a single point. Multiply the length and width to find the overlapping area.

Implementation

Copy PYTHON

```python
def inter_area(s1, s2) -> int:
    bl_a_x, bl_a_y, tr_a_x, tr_a_y = s1[0], s1[1], s1[2], s1[3]
    bl_b_x, bl_b_y, tr_b_x, tr_b_y = s2[0], s2[1], s2[2], s2[3]

    return (min(tr_a_x, tr_b_x) - max(bl_a_x, bl_b_x)) * (
        min(tr_a_y, tr_b_y) - max(bl_a_y, bl_b_y)
    )
```

Problems