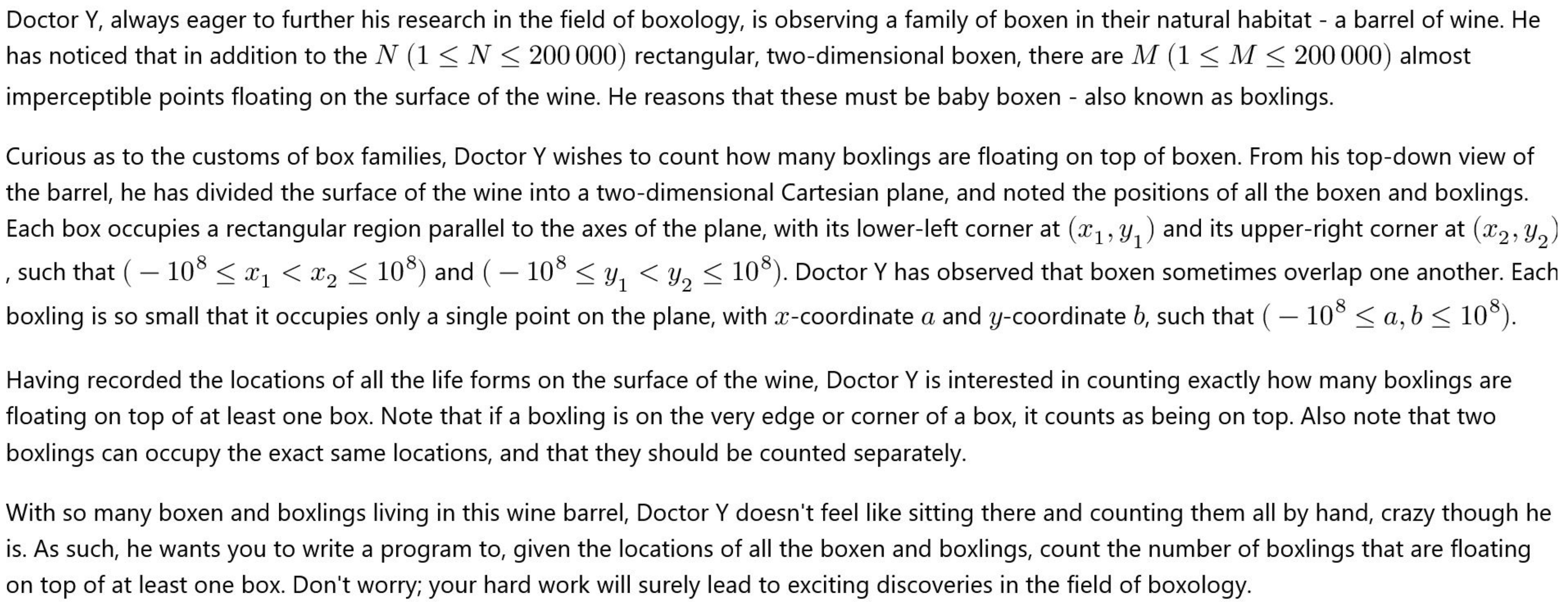
Line sweep

Overview

The general concept of line sweep is that you have a vertical line that you move through the plane from left to right and perform operations when it passes through an object.

This may seem a bit abstract at first, so let’s do an actual example.

Example



Doctor Y, always eager to further his research in the field of boxology, is observing a family of boxen in their natural habitat - a barrel of wine. He has noticed that in addition to the N $(1 \leq N \leq 200\,000)$ rectangular, two-dimensional boxen, there are M $(1 \leq M \leq 200\,000)$ almost imperceptible points floating on the surface of the wine. He reasons that these must be baby boxen - also known as boxlings.

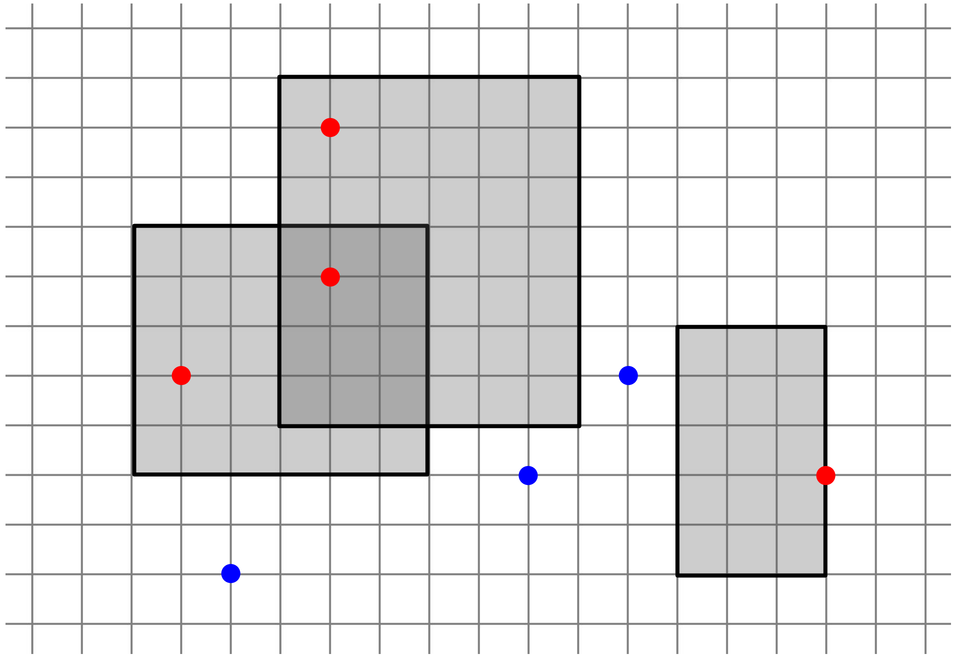
Curious as to the customs of box families, Doctor Y wishes to count how many boxlings are floating on top of boxen. From his top-down view of the barrel, he has divided the surface of the wine into a two-dimensional Cartesian plane, and noted the positions of all the boxen and boxlings. Each box occupies a rectangular region parallel to the axes of the plane, with its lower-left corner at (x_1, y_1) and its upper-right corner at (x_2, y_2), such that $(-10^8 \leq x_1 < x_2 \leq 10^8)$ and $(-10^8 \leq y_1 < y_2 \leq 10^8)$. Doctor Y has observed that boxen sometimes overlap one another. Each boxling is so small that it occupies only a single point on the plane, with x-coordinate a and y-coordinate b, such that $(-10^8 \leq a, b \leq 10^8)$.

Having recorded the locations of all the life forms on the surface of the wine, Doctor Y is interested in counting exactly how many boxlings are floating on top of at least one box. Note that if a boxling is on the very edge or corner of a box, it counts as being on top. Also note that two boxlings can occupy the exact same locations, and that they should be counted separately.

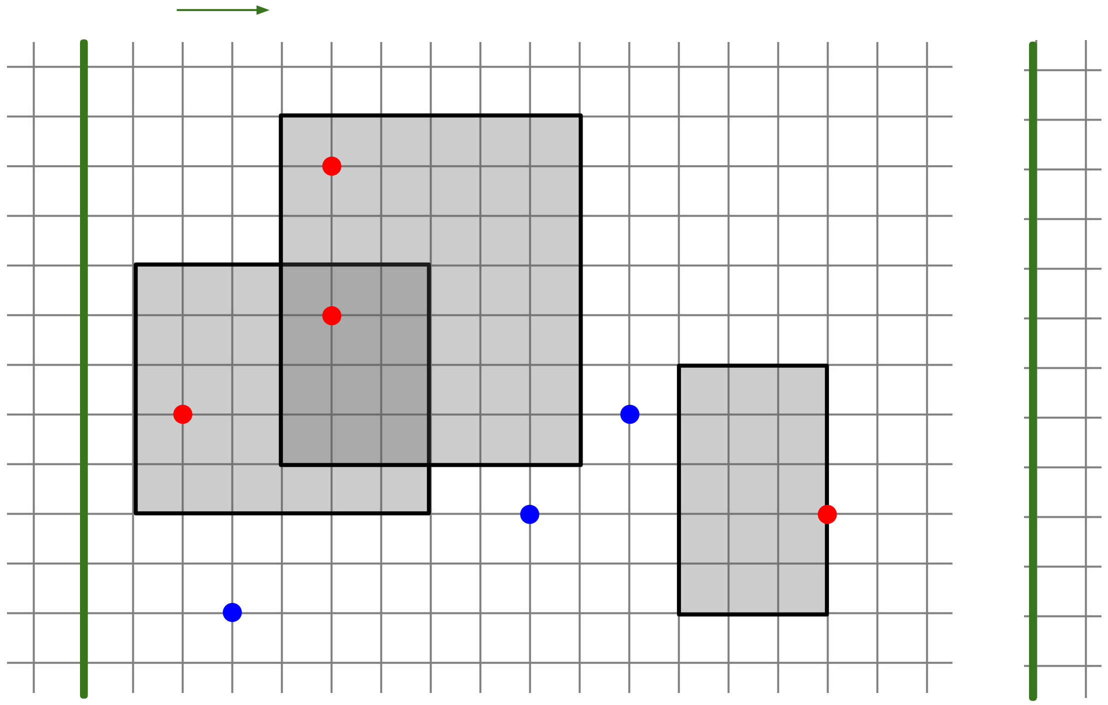
With so many boxen and boxlings living in this wine barrel, Doctor Y doesn't feel like sitting there and counting them all by hand, crazy though he is. As such, he wants you to write a program to, given the locations of all the boxen and boxlings, count the number of boxlings that are floating on top of at least one box. Don't worry; your hard work will surely lead to exciting discoveries in the field of boxology.

https://dmoj.ca/problem/boxl

Example

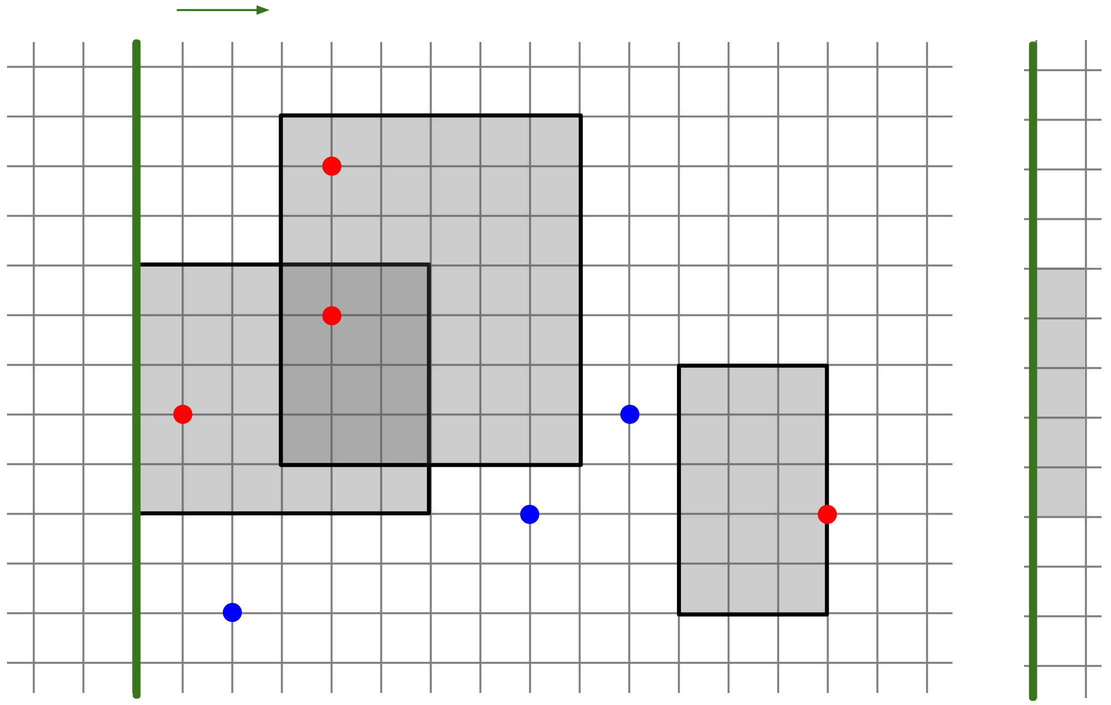


Line sweep



To solve this problem, we can sweep through the plane with a vertical line and keep track of how many boxes there are at each y value right in front of the line.

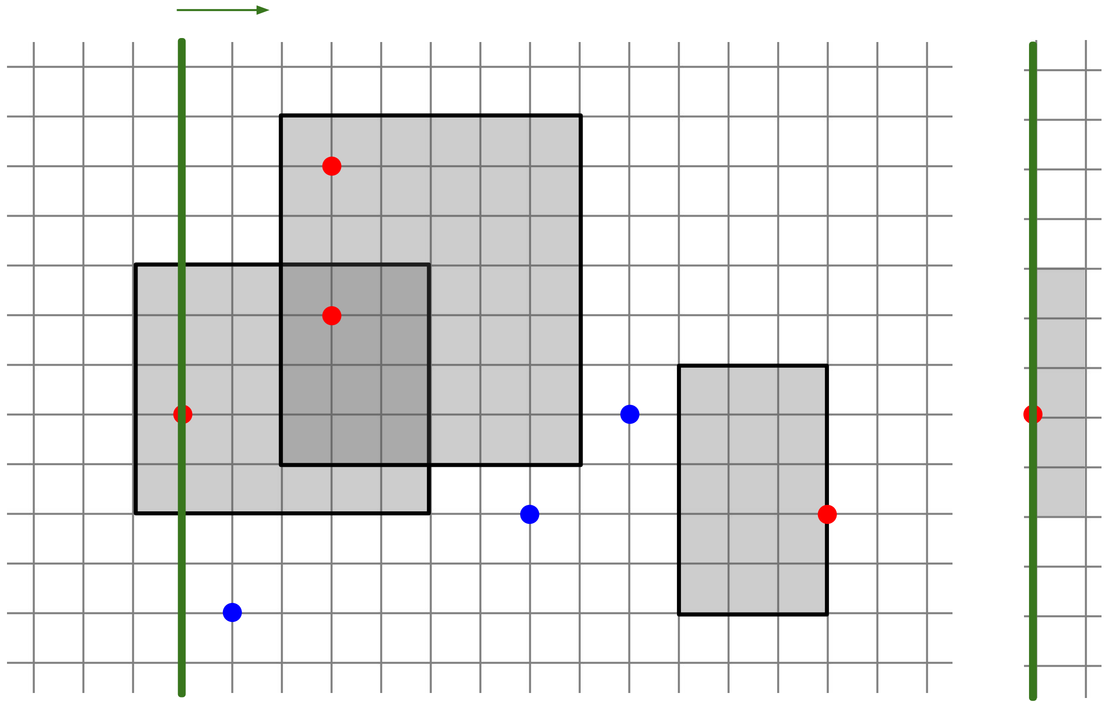
Line sweep



To solve this problem, we can sweep through the plane with a vertical line and keep track of how many boxes there are at each y value right in front of the line.

Every time the line meets the start of a rectangle, we have to increase each value between the two y coordinates of the rectangle.

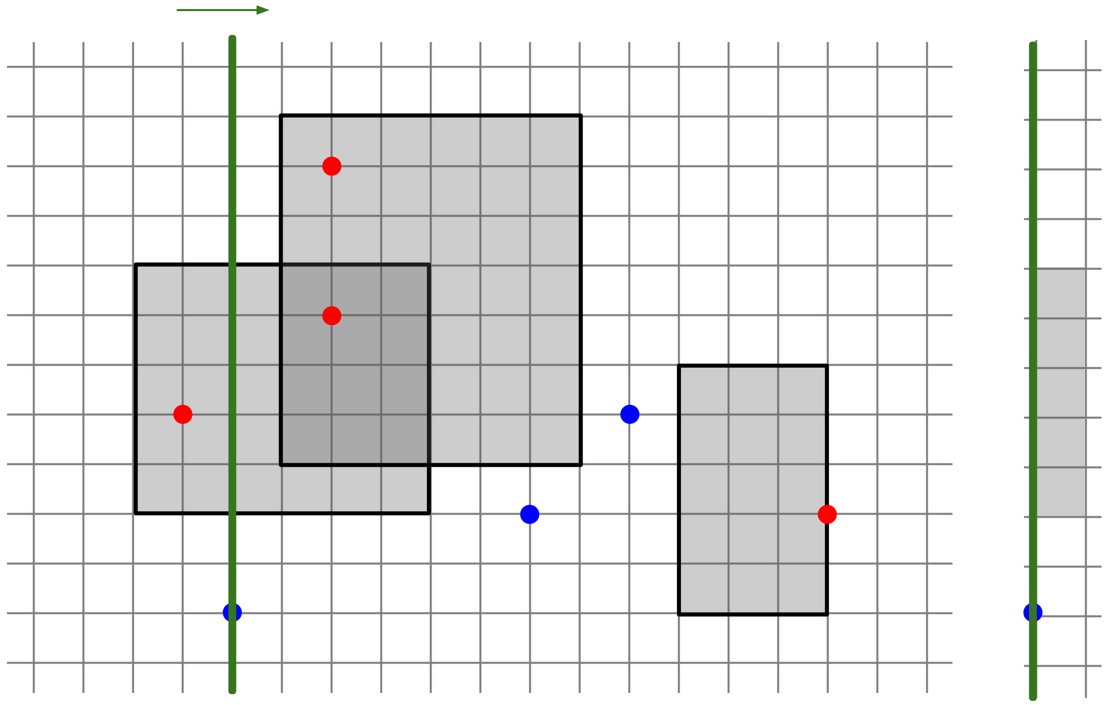
Line sweep



To solve this problem, we can sweep through the plane with a vertical line and keep track of how many boxes there are at each y value right in front of the line.

Every time the line passes through a point, we can check whether it's in a rectangle by checking the array at the y value of the point.

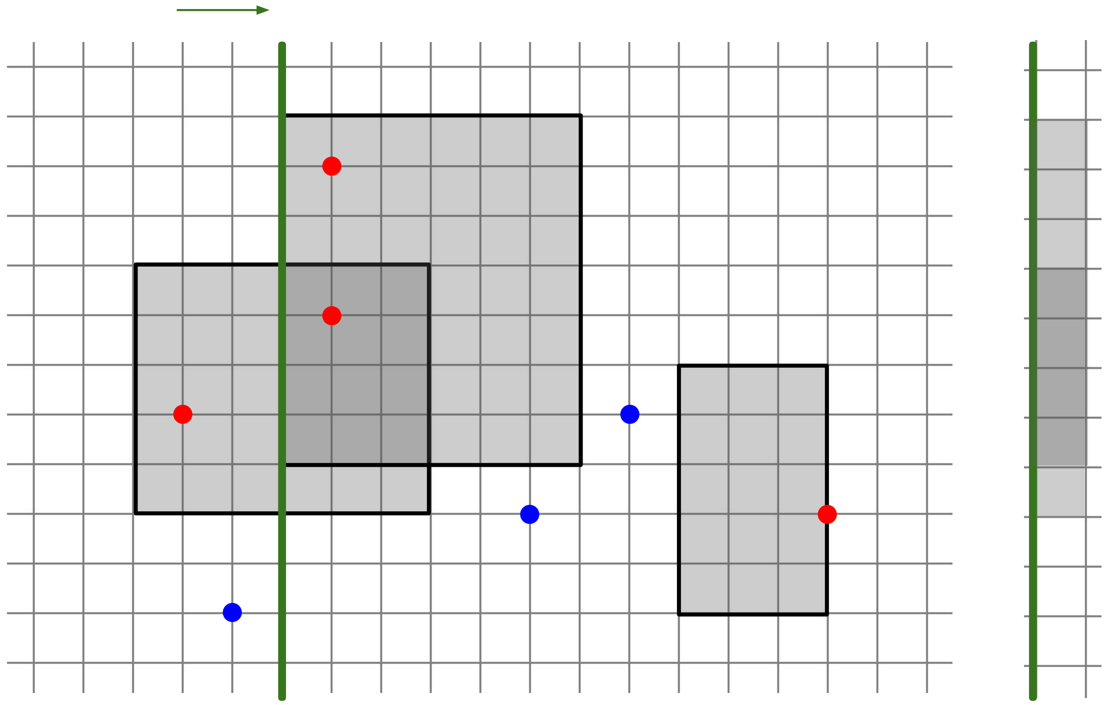
Line sweep



To solve this problem, we can sweep through the plane with a vertical line and keep track of how many boxes there are at each y value right in front of the line.

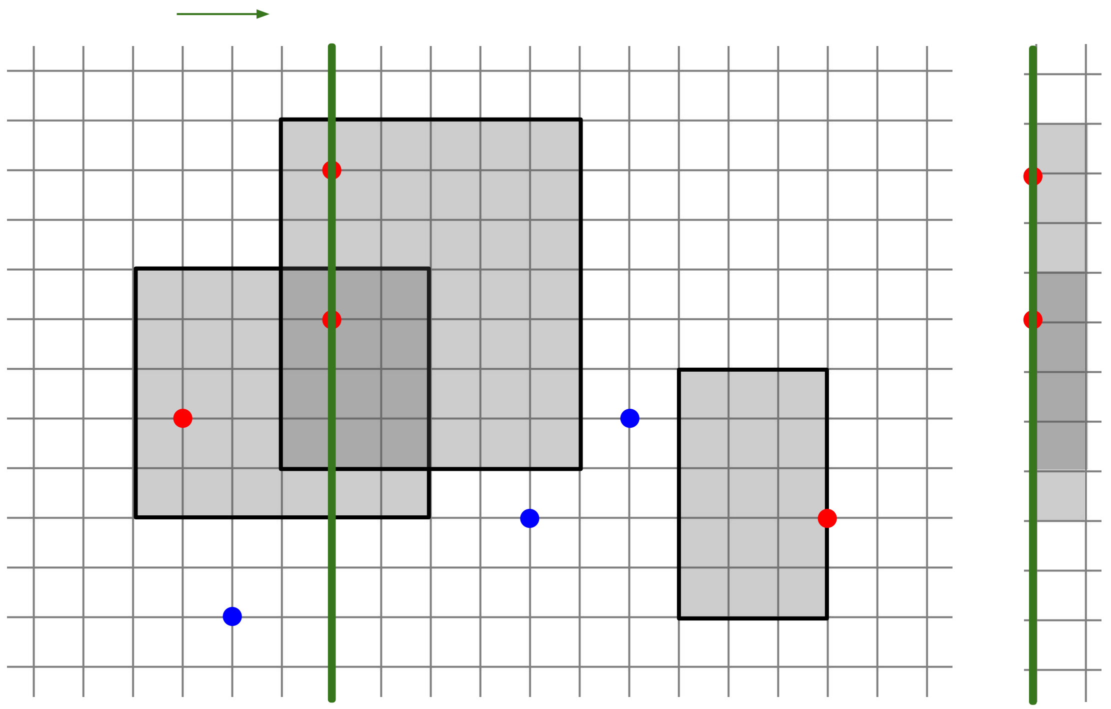
Every time the line passes through a point, we can check whether it's in a rectangle by checking the array at the y value of the point.

Line sweep



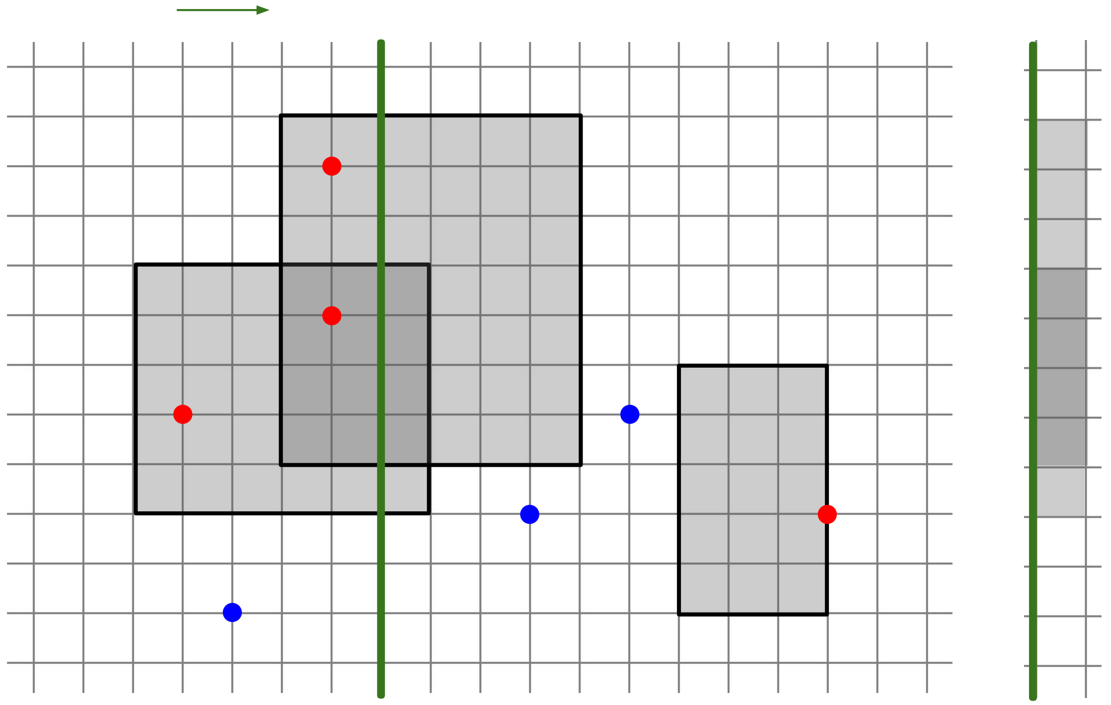
To solve this problem, we can sweep through the plane with a vertical line and keep track of how many boxes there are at each y value right in front of the line.

Line sweep



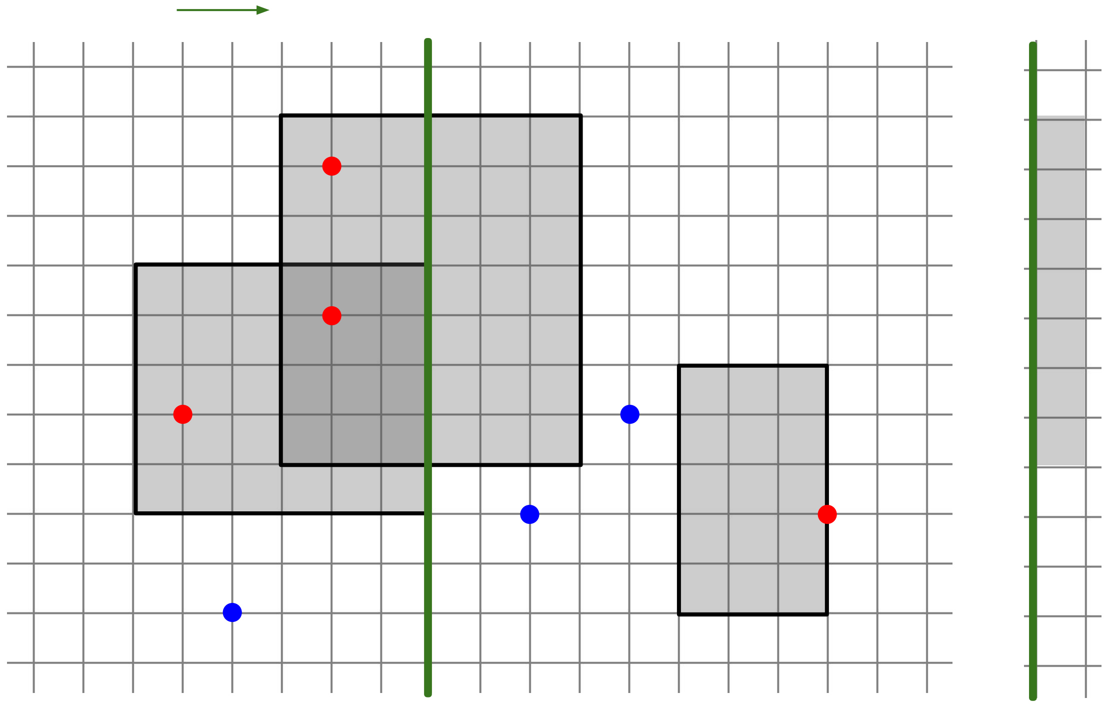
To solve this problem, we can sweep through the plane with a vertical line and keep track of how many boxes there are at each y value right in front of the line.

Line sweep



To solve this problem, we can sweep through the plane with a vertical line and keep track of how many boxes there are at each y value right in front of the line.

Line sweep



To solve this problem, we can sweep through the plane with a vertical line and keep track of how many boxes there are at each y value right in front of the line.

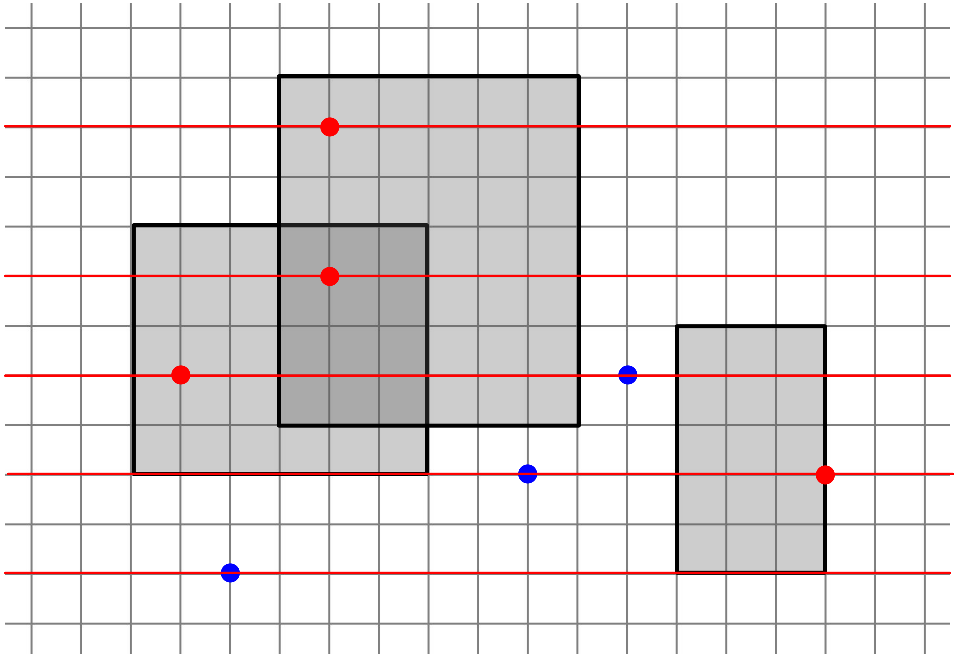
Every time the line meets the end of a rectangle, we have to decrease each value between the two y coordinates of the rectangle.

Optimizations

Right now, this algorithm is kinda garbage, probably slower than a naive O(NM) solution. However, there are a few key things we can do to get it down to O((N + M) logM):

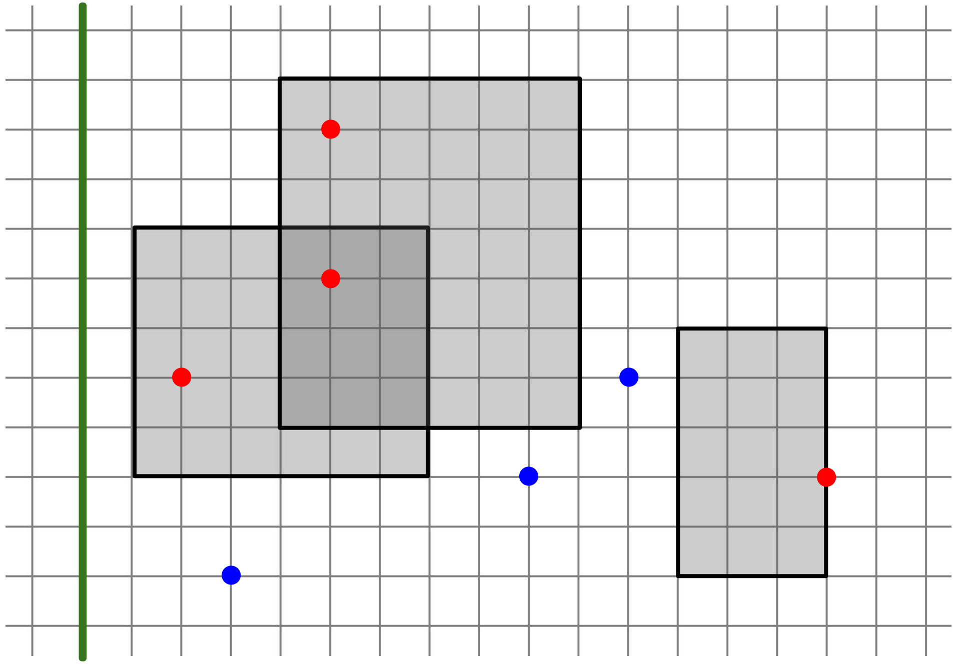
- Don’t actually go through all possible x values. We can instead sort all “events” (rectangle begin, point, rectangle end) based on their x value and iterate through those.
- Use a range update + point query binary indexed tree.
- Instead of having a binary indexed tree that stores all 2e8 y values, we can have a binary indexed tree that stores at most M different y values for each of the M points. This means that to perform operations on it, we will have to binary search through all the y values it does store.

Example



Unique y values of points:
1, 3, 5, 7, 10

Example



Events:

- `Update+,  x = 1, y1 = 3, y2 = 8`
- `Query,    x = 2, y = 5`
- `Query,    x = 3, y = 1`
- `Update+,  x = 4, y1 = 4, y2 = 11`
- `Query,    x = 5, y = 7`
- `Query,    x = 5, y = 10`
- `Update-,  x = 7, y1 = 3, y2 = 8`
- `Query,    x = 9, y = 3`
- `Update-,  x = 10, y1 = 4, y2 = 11`
- `Query,    x = 11, y = 5`
- `Update+,  x = 12, y1 = 1, y2 = 6`
- `Query,    x = 14, y = 3`
- `Update-,  x = 14, y1 = 1, y2 = 6`

Sorted by x.
In case of a tie: `Update+ < Query < Update-`

Implementation

```cpp
#include <bits/stdc++.h>

using namespace std;

struct Event {
    int type; // -1 - rect start, 0 - point, 1 - rect end
    int x, y1, y2; // y2 is unused when type is 0

    const bool operator<(const Event& e) const {
        // Order by x first and break ties based on type
        // Ties must be broken this way for this problem since being on the edge of a rectangle still counts
        if(x != e.x) return x < e.x;
        return type < e.type;
    }
};
```

Implementation

```
int n, m;
vector<Event> events;
vector<int> yValues; // All unique y values of points
int sz; // Size of BITree
int tree[200001];

// BITree functions
void update(int i, int v){
    for(++i; i <= sz; i += i & -i) tree[i] += v;
}

int query(int i){
    int res = 0;
    for(++i; i > 0; i -= i & -i) res += tree[i];
    return res;
}
```

Implementation

```
int main(){
    cin.sync_with_stdio(0); cin.tie(0); cout.tie(0);
    cin >> n >> m;

    for (int i = 0; i < n; ++i) {
        int x1, y1, x2, y2;
        cin >> x1 >> y1 >> x2 >> y2;
        events.push_back({-1, x1, y1, y2});
        events.push_back({1, x2, y1, y2});
    }

    for(int i = 0; i < m; ++i){
        int x, y;
        cin >> x >> y;
        events.push_back({0, x, y});
        yValues.push_back(y);
    }
```

Implementation

```
    sort(events.begin(), events.end());
    sort(yValues.begin(), yValues.end());
    yValues.erase(unique(yValues.begin(), yValues.end()), yValues.end());
    sz = yValues.size();

    int total = 0;
    for(Event event : events){
        int ind1 = lower_bound(yValues.begin(), yValues.end(), event.y1) - yValues.begin();
        int ind2 = upper_bound(yValues.begin(), yValues.end(), event.y2) - yValues.begin();
        if(event.type == 0){
            if(query(ind1) > 0) ++total;
        }
        else if(event.type == -1){
            update(ind1, 1);
            update(ind2, -1);
        }
        else {
            update(ind1, -1);
            update(ind2, 1);
        }
    }
    cout << total << '\n';
}
```

Applying this to other problems

There are two main things we did to solve this problem, which can be used separately, but are often used together:

- First, we had a line that we pushed through the plane
- Second, we looked at the list of y values that were relevant, and created a data structure that only stores those y values, saving a ton of memory and some performance

Line sweep-like problems

- https://dmoj.ca/problem/boxl
- https://dmoj.ca/problem/dmopc20c4p3
- https://dmoj.ca/problem/segments
- https://dmoj.ca/problem/ccc14s4
- https://dmoj.ca/problem/cco16p2