Tab 1

USACO Silver | Rectangular Pasture

Author: Hengsheng W.

Thinking Process

First, this problem may seem (and is, indeed) quite hard. Some ways you might want to start this problem and snatch the first few subtasks are:

- Simulate every single one of the 2^n possibilities for the cows, and check if each one works. This method will work for $n \leq 30$, but the idea seems very complicated, and the method certainly will not work for $n = 2500$.
- Iterate through the number of cows in the subset, and count the number of ways to do it successfully each time, using a map/dictionary. For 0 cows, the answer is 1. For 1 cow, the answer is n. Then for 2, 3, 4, and so on, we look at every single n-Choose-i way and store the pair of coordinates of the minimum-area rectangle that stores these cows in a set or map or something. Then, we will not overcount cases.
- Look at every possible pair of cows as corners of the subset rectangle, and then iterate through every single cell in the rectangle. We store the set of cows in this subset inside of a set of sets, to prevent overcounting.

All of these naive ways might be able to pass a few test cases, but this certainly is not enough to solve the whole problem. Let's take a look at the solution algorithm:

Solution - Algorithm

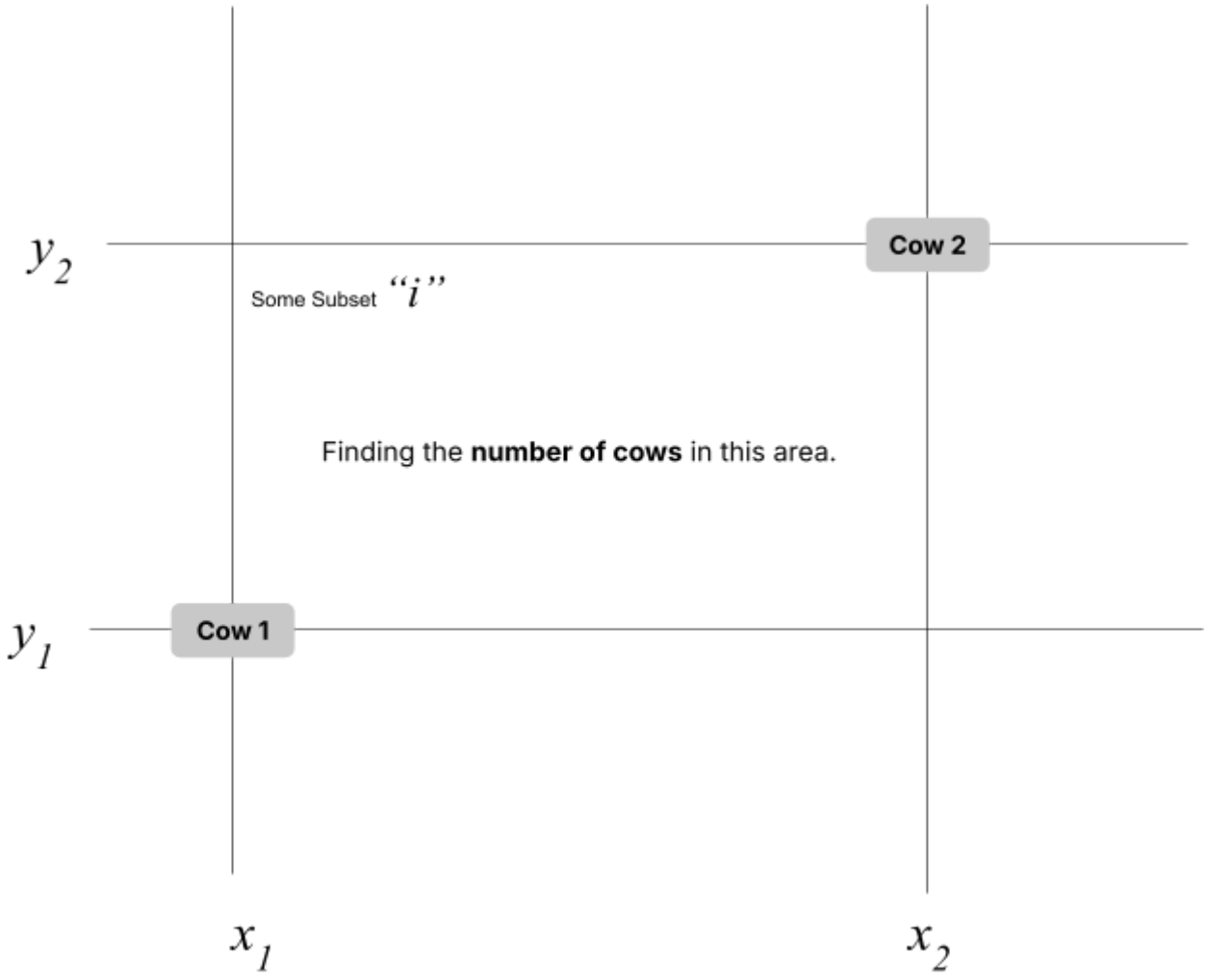
Observe:

- In order to have a unique subset of cows inside a pasture
- Then, the cows should be in a rectangle
- The rectangle should always have two opposite corners being the coordinates of a cow, in order to be the rectangle with minimal area

It is also an **extremely** important part that **any two cows will not share the same x or y**. If so, then this problem would be a little bit more complicated.

Using the previous observations, we realize that we can:

- Iterate through every distinct pair of cows, and use them as the minimum-x and maximum-x borders of the rectangle holding the current subset:



- Now, notice that the number of ways we can create a rectangle that has Cow 1 and Cow 2 as x-value "anchor" points is the number of cows between x1 and x2 while being below y1 (a), multiplied by the number of cows between x1 and x2, while being above y2 (b), including cow 1 and cow 2.
- This brings up an essential question: how do we calculate a and b? Well, we can:

- Define a 2D Prefix Sum which stores the number of cows from the coordinates $(0,\ 0)$ to $(i,\ j)$. Using this, we can find the number of cows in the area of the rectangle below the main “min”-rectangle and the number of cows above the main “min”-rectangle.

This brings up a problem; calculating the prefix sum for all 10^9 x and y values will be rather difficult. Instead, we can do something called “compression”. This will make prefix sums much easier and efficient to calculate:

There are certain “points of interest” on the huge chessboard; that is where the cows are located. So, instead of trying to analyze a chessboard looking like this:

```
0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
```

We could, instead, compress the grid of cows into this:

```
0 1 0 0
1 0 0 0
0 0 1 0
0 0 0 1
```

This way, creating a prefix sum will be much easier, and in this example, it will look like this:

```
0 1 1 1
1 2 2 2
1 2 3 3
1 2 3 4
```

- In order to create the 2D prefix sum, we will iterate through every row, and inside each iteration, iterate over every column. We will have to add the number of cows so far from the upper area and from the left-side area, and

then subtract the number of cows from the upper-left area to stop overcounting.

- This also brings up the issue that our prefix sum's index starts from 0, so we will 1-index the prefix sum.
- Now, the prefix sum will look like this:

```
0 0 0 0 0
0 0 1 1 1
0 1 2 2 2
0 1 2 3 3
0 1 2 3 4
```

Now that we have this prefix sum, the main idea of the solution should be clearer, and we can move onto the code.

CODE

By Hengsheng W.

Initialization of regular items and packages

```cpp
#include <bits/stdc++.h>
using namespace std;

using ll = long long;
using p = pair<ll, ll>;
```

Initialization and input of the vector that stores the locations.

```cpp
int main()
{
    // BASIC INPUT

    int n;
    cin >> n;
    vector<p> loc (n);
    for (auto& l : loc) {
        cin >> l.first >> l.second;
    }
```

Compression of the locations.

```cpp
// COW LOCATION COMPRESSION

    sort(loc.begin(), loc.end());
    for (int i = 0; i < n; i++) {
        loc[i].first = i + 1;
    }
    sort(loc.begin(), loc.end(), [](const p& a, const p& b) {
        return a.second < b.second;
    });
    for (int i = 0; i < n; i++) {
        loc[i].second = i + 1;
    }
    sort(loc.begin(), loc.end());
```

2D Prefix Sum Initialization, now that we have compressed the graph.

```
// INITIALIZATION OF 2D PREFIX SUM

vector<vector<int>> prefix (n + 1, vector<int> (n + 1, 0));
for (auto& cow : loc) {
    prefix[cow.first][cow.second] = 1;
}
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        prefix[i][j] += prefix[i - 1][j    ] +
                        prefix[i    ][j - 1] -
                        prefix[i - 1][j - 1];
    }
}
```

Find the answer:

- Iteration between all pairs of cows
    - Look at the coordinates, define coordinates of corners of "min"-rectangle
    - Find the number of cows on upper side of min-rectangle
    - Find number of cows on lower side of min-rectangle

```
// FINDING THE ANSWER

ll answer = n + 1;
for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
        p currA = loc[i];
        p currB = loc[j];
        int x1 = min(currA.first, currB.first);
        int x2 = max(currA.first, currB.first);
        int y1 = min(currA.second, currB.second);
        int y2 = max(currA.second, currB.second);
        ll  higher = 1 + prefix[x2][n]  -
```

```
                            prefix[x2][y2] -
                            prefix[x1][n]  +
                            prefix[x1][y2];
            ll  lower  = 1 + prefix[x2][y1 - 1] -
                            prefix[x1][y1 - 1];
            answer += higher * lower;
        }
    }
    cout << answer;
}
```

Hopefully this made some sense, goodbye!