E3-H 点集闭包

时间限制：3000ms 内存限制：65536kb

题目描述

我们称一个二维点集 S 是优雅的，当且仅当对任意的点$(x_1,y_1),(x_2,y_2)\in S$，都有$(min(x_1,x_2),min(y_1,y_2))\in S$。

现在给出一个二维点集 S，请你向其中加入最少的点，使其变为优雅的点集 S'，求出 S' 中点的个数。

测试数据保证给出的点集 S 中没有重复的点。

输入

第一行为数据组数 T ($1\le T\le 10$)。

接下来 T 组数据：

第一行一个正整数 n ($1\le n\le 10^5$) 表示给出的点集中点的个数。

接下来 n 行，每行两个整数 x_i,y_i ($-10^9\le x_i,y_i\le 10^9$) 表示第 i 个点的坐标。

输出

对于每组数据，输出一行一个正整数，表示 S' 中点的个数。

输入样例

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

输出样例

```
3
7
```

解题思路

先将每个点按照x从大到小，y从小到大排序。此时，对于每个点(x_0, y_0)，需要增加的点数就等于满足以下条件的点（x，y）的个数：

1. $x > x_0$，即排序后位于该点之前；
2. 记点$（x_0，y_0）$的前一个横坐标也为x_0的点为$（x_0，u）$，则$u \leq y < y_0$；
3. 如果有多个相同的y，则**只算一次。**

这些点的个数可以用一个**线段树**来维护。

附注：

线段树（Segment Tree）主要用于维护**区间信息**（要求满足结合律）。与树状数组相比，它可以实现$O(logn)$的**区间修改**，还可以同时支持**多种操作**（加、乘），更具通用性。

在本题中，我们用它来求区间和（**满足上述条件的点(x, y)的个数**）。

注意：

题目中y的范围较大，直接使用线段树时的空间需求较大。又因为我们注意到每组数据的总点数小于等于10^5，此时可以使用**离散化的方式压缩线段树的空间**。具体的方法说明请见下文参考代码中的注释。

参考代码

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

using namespace std;

typedef long long LL;

const int SIZEN = 1e5 + 5;

struct point {
    int x, y;
    int id;
};

point p[SIZEN];

// 定义线段树
struct tree {
    int f[SIZEN << 2];

    int getSum(int now, int l, int r, int tl, int tr) {
        if (l == tl && r == tr)
            return f[now];
        int res = 0;
        int mid = (l + r) >> 1;
        if (mid >= tr)
            res += getSum(now << 1, l, mid, tl, tr);
        else if (mid < tl)
            res += getSum(now << 1 | 1, mid + 1, r, tl, tr);
```

```
        else
            res += getSum(now << 1, l, mid, tl, mid) + getSum(now << 1 | 1, mid + 1, r, mid + 1, tr);
        return res;
    }

    void replace(int now, int l, int r, const int target, const int num) {
        if (l == target && r == target) {
            f[now] = num;
            return;
        }
        int mid = (l + r) >> 1;
        if (target <= mid)
            replace(now << 1, l, mid, target, num);
        else
            replace(now << 1 | 1, mid + 1, r, target, num);
        f[now] = f[now << 1] + f[now << 1 | 1];
    }
};

tree f;

// 用于求解时的排序
bool cmp(const point a, const point b) {
    if (a.x == b.x)
        return a.y < b.y;
    return a.x > b.x;
}

// 用于离散化时的排序
bool ccmp(const point a, const point b) {
    return a.y < b.y;
}

int main() {
    int tt, n;
    scanf("%d", &tt);
    p[0].x = -1e9 - 10000;
    p[0].id = 0;
    while (tt--) {
        // 初始化线段树
        memset(f.f, 0, sizeof(f.f));

        // 输入部分
        scanf("%d", &n);
        for (int i = 1; i <= n; ++i) {
            cin >> p[i].x >> p[i].y;
        }

        // 离散化压缩线段树，将相同的y映射到同一个编号，以减少空间消耗
        sort(p + 1, p + 1 + n, ccmp);
        int len = 0;
        for (int i = 1; i <= n; i++) {
            if (p[i].y == p[i - 1].y)
                p[i].id = p[i - 1].id;
            else {
```

```
                p[i].id = p[i - 1].id + 1;
                len++;
            }
        }

        // 将每个点按照x从大到小，y从小到大排序
        sort(p + 1, p + 1 + n, cmp);
        // 计算答案
        LL ans = 0;
        int r = 0, begin = 1;
        for (int i = 1; i <= n; ++i) {
            // 维护线段树
            if (p[i].x != p[i - 1].x) {
                r = 0;
                for (int j = begin; j < i; ++j) {
                    f.replace(1, 0, n, p[j].id, 1);
                }
                begin = i;
            }
            int ref = p[i].id;
            ans += f.getSum(1, 0, n, r, ref - 1);
            r = ref;
        }
        printf("%lld\n", ans + n);
    }
    return 0;
}
```