



**《程序设计原理》实验报告**

**——问题 H: 矩阵内的点**

**学 号 3020205094**

**姓 名 刘宏伟**

**学 院 智能与计算学部**

**年 级 2021级**

**任课教师 杜朴风**

**2022年 12月 11日**

# 实验目的

掌握数据的离散化的理念、意义和其在C++中的实现方法，熟悉使用二维前缀和来求解矩阵面积，学会选择恰当的数据结构以及高效率的库函数来达成目的，学会认真考虑特殊情况以及面对大量数据的输入输出的处理方法。

# 二．实验内容

1. **设计思路**

考虑到点的坐标很大，而点的数量有限，这里考虑建立点的集合，并且保持横纵坐标的大小关系，以达到离散化的目的，同时又要保存好输入点时的序号（查询时要用）和横纵坐标的绑定关系（映射到新点时要用），所以这里考虑再次使用嵌套pair容器来存储，同时其sort函数也可以帮助我们快速确立大小关系，将输入点映射到大小不超过n\*n区域内的新点后，就可以构建二维前缀和数组，然后根据输入的查询序号找到相应点并运用公式求出答案。

1. **代码实现**（写上必要的注释）

#include <bits/stdc++.h>

using namespace std;

// 储存原始点的信息依次分别为x坐标原始值、y坐标大小排名、y坐标原始值、x坐标大小排名、点的序号

pair<pair<pair<int, int>, pair<int, int>>, int> raw\_point[5001];

// 储存点的二维前缀和

int point[5001][5001];

// 储存答案

int ans[100001][4];

// 自定义比较函数，用来给y坐标排序

int cmp(pair<pair<pair<int, int>, pair<int, int>>, int> a, pair<pair<pair<int, int>, pair<int, int>>, int> b) {

if (a.first.second != b.first.second)

return a.first.second < b.first.second;

else

return a.first.first > b.first.first;

}

int main() {

std::ios::sync\_with\_stdio(false);

int n, m;

cin >> n >> m;

for (int i = 0; i < n; i++) {

// 输入点的坐标，并带上序号存储

cin >> raw\_point[i].first.first.first >> raw\_point[i].first.second.first;

raw\_point[i].second = i;

}

// 按照y坐标优先进行排序

sort(raw\_point, raw\_point + n, cmp);

int ty = 0;

for (int i = 0; i < n; i++) {

// 特别处理一下坐标相等的情况，并将y坐标排名存储起来

if (i > 0 && raw\_point[i].first.second.first == raw\_point[i - 1].first.second.first) {

raw\_point[i].first.first.second = raw\_point[i - 1].first.first.second;

} else

raw\_point[i].first.first.second = ty++;

}

// 按照x坐标优先进行排序

sort(raw\_point, raw\_point + n);

int tx = 0;

for (int i = 1; i <= n; i++) {

// 特别处理一下坐标相等的情况，并将x坐标排名存储起来

if (i > 1 && raw\_point[i - 1].first.first.first == raw\_point[i - 2].first.first.first) {

raw\_point[i - 1].first.second.second = raw\_point[i - 2].first.second.second;

} else {

raw\_point[i - 1].first.second.second = tx++;

}

// 将点离散化存储到密集的点数组中

point[raw\_point[i - 1].first.second.second + 1][raw\_point[i - 1].first.first.second + 1]++;

}

for (int i = 1; i <= tx; i++) {

for (int j = 1; j <= ty; j++) {

// 将点数组构造为二维前缀和数组

point[i][j] = point[i][j] + point[i - 1][j] + point[i][j - 1] - point[i - 1][j - 1];

}

}

int x, y;

for (int i = 1; i <= m; i++) {

cin >> x >> y;

// 找到对应点的坐标排名

int posX1 = 0, posY1 = 0, posX2 = 0, posY2 = 0;

for (int j = 0; j < n; j++) {

if (posX1 == 0 && raw\_point[j].second == x) {

posX1 = raw\_point[j].first.second.second + 1;

posY1 = raw\_point[j].first.first.second + 1;

}

if (posX2 == 0 && raw\_point[j].second == y) {

posX2 = raw\_point[j].first.second.second + 1;

posY2 = raw\_point[j].first.first.second + 1;

}

if (posX1 != 0 && posX2 != 0)

break;

}

// 统一转换为右下角到左上角，否则答案算出来是错的，不满足公式

if (posX1 > posX2) {

ans[i][0] = posX1;

ans[i][2] = posX2;

} else {

ans[i][0] = posX2;

ans[i][2] = posX1;

}

if (posY1 > posY2) {

ans[i][1] = posY1;

ans[i][3] = posY2;

} else {

ans[i][1] = posY2;

ans[i][3] = posY1;

}

}

for (int i = 1; i <= m; i++) {

// 通过二维前缀和求解

cout << point[ans[i][0]][ans[i][1]] - point[ans[i][0]][ans[i][3] - 1] - point[ans[i][2] - 1][ans[i][1]] + point[ans[i][2] - 1][ans[i][3] - 1] << "\n";

}

return 0;

};

1. **结果展示**（Oj平台提交记录）



# 三．实验中遇到的问题和解决办法

遇到了一些问题：首先是算法方面的，一开始没有特别考虑好坐标相等的情况，花了一点时间改良了算法，可以合并相等的坐标。然后还有就是最后计算答案时，没有考虑好两个点的相对位置，导致答案会计算错误，这个最终也是通过自拟样例排查出来了。

然后就是阻碍比较大的问题了，首先是时间超限问题，第一次提交提示时间超限，一开始还以为是算法复杂度不过关，于是又看了一会，注释掉一些耗时的操作发现还是这样。于是我就开始使用排查法，由于不罚时，我就一个一个把有嫌疑的注释掉然后提交，最终剩下应该简单的输入输出居然还是时间超限，我就不得不怀疑输入输出的问题了，而且已经设置了std::ios::sync\_with\_stdio(false); 关闭io同步流，并且换用过scanf还是一样。于是就发现了如果cin完后面有个同步的cout的话，就会增加耗时，估计还是同步这块的问题，这里我改用了用一个数组先缓存答案，最后输入完了再进行输出，效果有所改善，但是仍然时间超限，于是我就终于发现了是endl的问题，当用了std::endl时，会刷新缓存，从而造成频繁的系统调用，这里我换成\n后瞬间豁然开朗，这给我了极深的感受，收益颇深。