HZOJ-244 总结

宋星霖

2024 年 2 月 8 日

1 题目描述

1.1 题目描述

约翰打算建一个围栏来圈养他的奶牛。作为最挑剔的兽类, 奶牛们要求这个围栏必须是正方形的, 而且围栏里至少要有 $C(1 \leq C \leq 500)$ 个草场, 来供应她们的午餐。

约翰的土地上共有 $N(C \leq N \leq 500)$ 个草场, 每个草场在一块 1x1 的方格内, 而且这个方格的坐标不会超过 10000。有时候, 会有多个草场在同一个方格内, 那他们的坐标就会相同。

现求围栏的最小边长为多少。

1.2 输入

第一行输入两个数 C, N 。

接下来 N 行每行两个数, 表示每个草场的坐标 X_i, Y_i 。

1.3 输出

输出围栏的最小边长。

2 算法设计

2.1 构建一个二分模型 (0000011111)

原问题可以转化为：假定可以构造一个函数 $check()$ ，它可以实现给定一个围栏边长 L ，是否可以至少拥有 C 个草场。假定拥有 C 个草场的最小边长为 x ，则 $L < x$ ，返回 0，$L \geq x$ ，返回 1。显然，该函数单调递增，可用二分法查找到 x 。

2.2 用二位扫描法实现 *check*() 函数

第一步，将所有点按 X 坐标排序。检查边长小于等于 L 的范围内是否存在草场数量大于等于 C ，如图 1 所示。

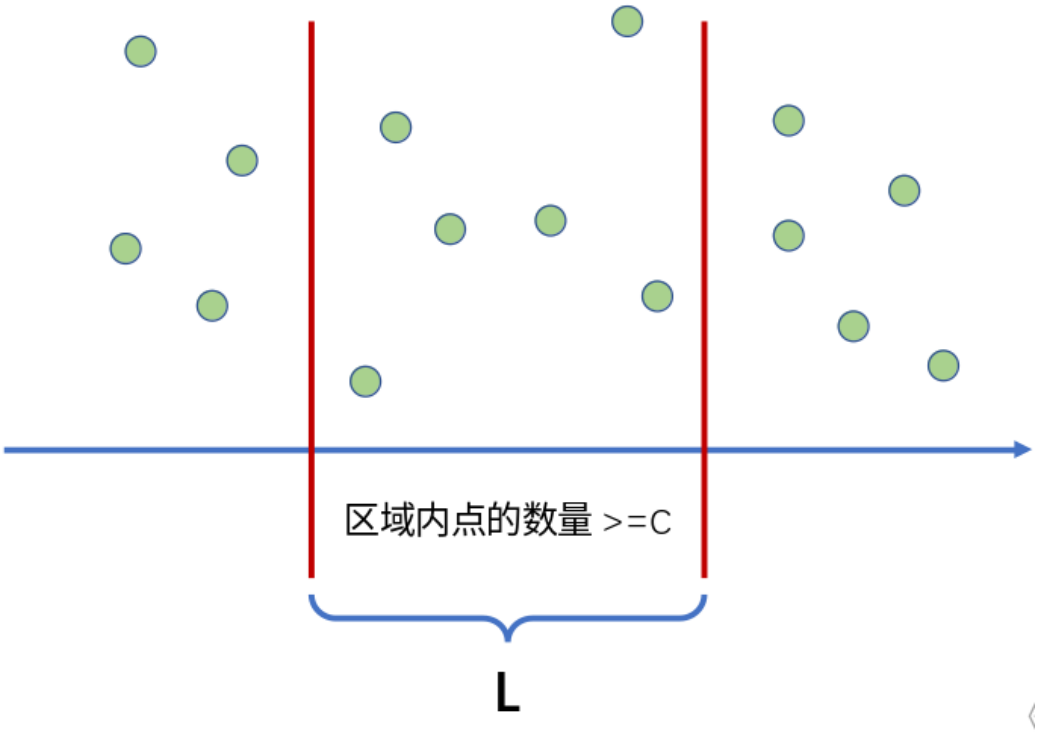

图 1: 二维扫描线法 1

第二步，取出区域内的点的 Y 坐标并排序。检查边长小于等于 L 的范围内是否存在草场数量大于等于 C，如图 2 所示。

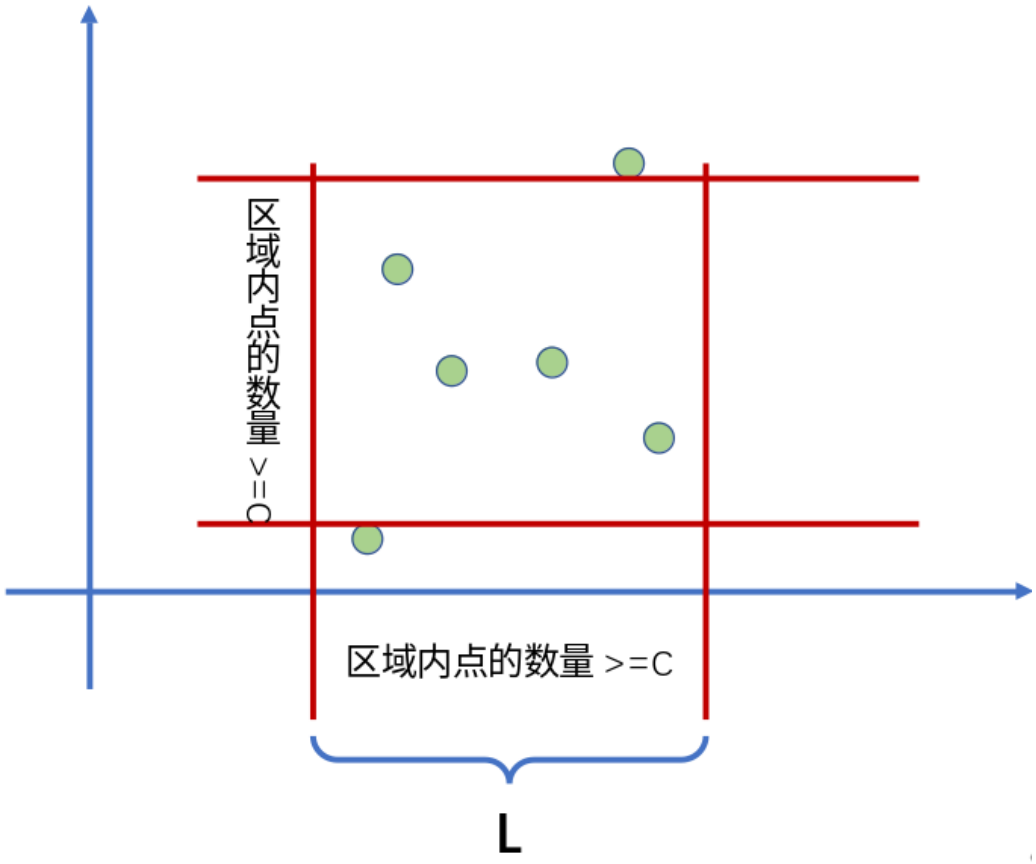

图 2: 二维扫描线法 2

3 代码展示

Listing 1: HZOJ-244.cpp

```cpp
#include <bits/stdc++.h>
#define MAX_N 500
using namespace std;
struct point {
    int mX, mY;
} gArr[MAX_N + 5];
int gTmp[MAX_N + 5];
bool check_kid (int n, int c, int left, int right, int len) {
    for (int i = left; i <= right; i++) {
        gTmp[i] = gArr[i].mY;
    }
    sort (gTmp + left, gTmp + right + 1);
    for (int i = left + c - 1; i <= right; i++) {
        if (gTmp[i] - gTmp[i - c + 1] < len)
            return true;
    }
    return false;
}
bool check (int len, int n, int c) {
    int j = 1;
    for (int i = 1; i <= n; i++) {
        while (gArr[i].mX - gArr[j].mX >= len)
            j++;
        if (i - j + 1 < c)
            continue;
        if (check_kid (n, c, j, i, len))
            return true;
    }
    return false;
}
int solve (int left, int right, int n, int c) {
    int mid = 0;
    while (left < right) {
        mid = (left + right) / 2;
        if (check (mid, n, c))
            right = mid;
        else
            left = mid + 1;
    }
    return left;
}
int main () {
    int c, n;
    scanf ("%d%d", &c, &n);
    for (int i = 1; i <= n; i++) {
        scanf ("%d%d", &gArr[i].mX, &gArr[i].mY);
    }
    sort (gArr + 1, gArr + 1 + n,
```

```
49          [] (const point& i, const point& j) -> bool {
50              if (i.mX != j.mX)
51                  return i.mX < j.mX;
52              return i.mY < j.mY;
53          });
54      printf ("%d\n", solve (0, 10000, n, c));
55      return 0;
56  }
```