圆的离散化在计算几何中的应用

Felicia

2008-07-08

摘要 本文面向一定计算几何基础的 ICPC 参赛选手。文章首先介绍解决许多计算几何问题的核心思想——离散化思想，着重讨论该思想在处理有关圆的问题方面的应用。然后介绍两种经典的圆的离散化方法——水平离散化和极角离散化，并结合例题说明它们在 ICPC 的计算几何题中的具体应用。

关键词 计算几何 离散化 圆

目录

1 引言

1.1 计算几何中的覆盖问题

在计算几何中，各种覆盖问题很常见。例如：已知平面上N个不透明矩形，这些矩形的边都平行于坐标轴，它们按$1-N$的顺序自底向上叠放在坐标平面上。这N个矩形的面积并是多少？它们的并的轮廓线长是多少？哪些矩形是从上方可见的？

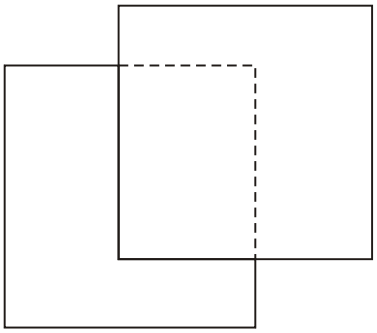

图 1: 矩形覆盖问题

1.2 解决覆盖问题的利器——离散化

遇到这样的问题，如果已知矩形的顶角坐标都是一定范围内的整数，那么最朴素的做法就是枚举范围内每个整数对(x,y)，以确定边长为1的小正方形，再对每个小正方形进行判断。这样效率及其低下，并且当坐标范围很大或者是实数的时候就无能为力了。

朴素方法为什么失败呢？原因是连续性问题不适合计算机处理，计算机擅长处理整数，也就是处理**离散性**的问题。在计算几何中，将原本看似连续的几何问题转化成适合计算机处理的离散性问题，我们称之为**离散化**。许多覆盖问题经过离散化处理之后，往往能化繁为简，迎刃而解。

尝试用离散化思想解决矩形覆盖问题。根据矩形的性质，很容易想到用水平线切割矩形，切割位置就是所有矩形的上下边所在位置。

经过这样的处理之后，我们很容易结合扫描线方法和线段树解决很多问题。

求面积 从y_-到y_+扫描，每发生一个事件就将遇到的矩形下边插入线段树，并从线段树中删除矩形上边。这样扫描线每次扫过的面积就是线段树保存的线段长度乘上扫描线扫过的距离。时间复杂度是$O(nlogn)$。

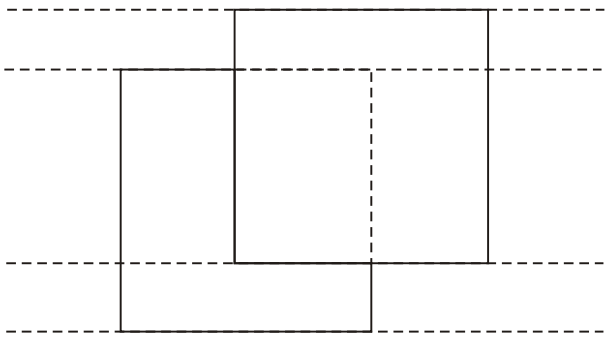

图 2: 水平线切割矩形

求周长 从y_-到y_+扫描，每发生一个事件就将遇到的矩形下边插入线段树，并从线段树中删除矩形上边。这样扫描线每次扫过的竖线部分的周长就是线段树保存的独立区间数乘上扫描线扫过的距离再乘2。

再从x_-到x_+扫描，类似的可以求出横线部分的周长。

时间复杂度是$O(nlogn)$。

判断可见性 用两组分别平行与x轴和y轴的切割线切割矩形，得到的线段称为元线段。定义元线段的中点为关键点。首先初始化所有矩形均不可见。然后枚举关键点，对每个关键点标记覆盖它的最上矩形为可见。这样就能求出所有可见矩形。时间复杂度是$O(n^3)$。

从上面的例子可以看出，离散化思想对于解决计算几何中的覆盖问题，起着至关重要的作用：离散化能化繁为简，提炼出几何问题中的关键信息，击中要害，进而为我们解决问题提供帮助。

2 圆的离散化方法

2.1 有关圆的覆盖问题

我们把刚才遇到的问题稍微变形：

已知平面上N个不透明圆，它们按$1-N$的顺序自底向上叠放在坐标平面上。这N个圆的面积并是多少？它们的并的轮廓线长是多少？哪些圆是从上方可见的？

矩形一旦变成了圆，问题就复杂了许多。但是运用离散化思想，这样的问题还是能够解决的。

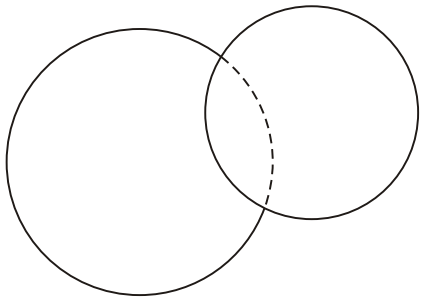

图 3: 圆覆盖问题

2.2 解决方法

仿照矩形的离散化方法，我们用一些水平线切割圆，就是所谓的**水平离散化**。

离散化的过程中总是要遵循某种**序**，例如水平离散化就遵循**水平序**。我们能不能换一种思路，根据圆的特殊性质，找到另一种序进行离散化呢？

3 水平离散化

3.1 核心思想

为什么水平离散化能够很好的处理矩形覆盖问题？我们注意到在用离散化方法解决矩形覆盖问题中，所选择的水平切割线的位置就是所有矩形上下边的位置。这样切割之后，矩形变成了一些横条。

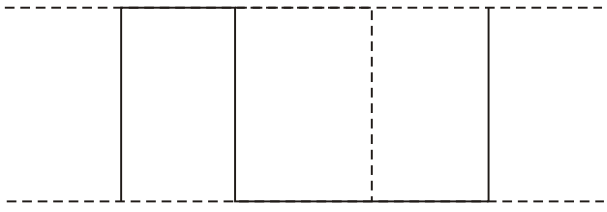

图 4: 矩形被切割成性质很好的横条

这些横条有非常好的性质——内部没有水平线。这个性质使得我们能够把横条看成区间，区间关系只可能在切割线处改变（称为发生事件），因此我们可以在切割线处利用线段树进行区间操作，从而降低时间复杂度。

同样我们把水平离散化应用到圆覆盖问题上来，首先需要选择水平切割线的位置。如果仿照矩形覆盖问题，让这些水平切割线通过所有圆的上下端点，会发现切割出来的横条性质不够好，仍然难以处理。

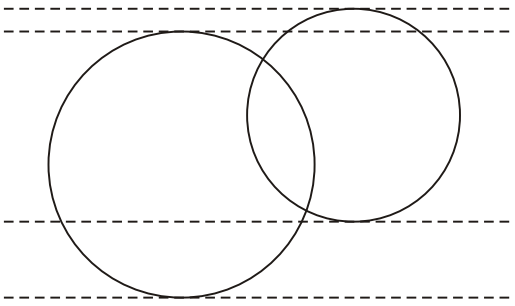

图 5: 通过上下端点的水平切割线

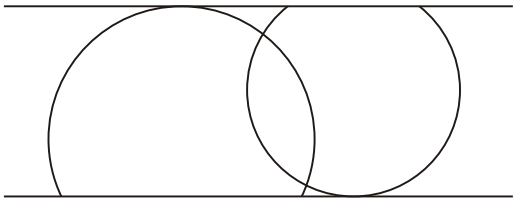

图 6: 得到的横条性质不好，无法直接处理

因此我们考虑增加一些切割点。增加在哪里呢？最自然的就是所有圆的交点了。

现在我们发现增加了这些切割点之后，切出的横条的性质好了很多：总是能很方便的划分成弓形和梯形（或三角形），而这两种形状我们都能够直接计算面积。

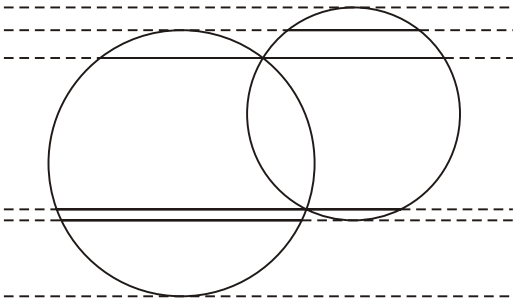

图 7: 通过上下端点和交点的水平切割线

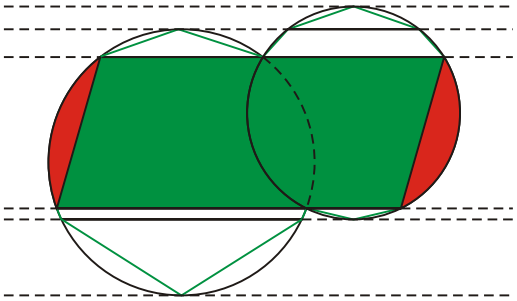

图 8: 横条可以方便计算面积

3.2 实现方法

3.2.1 求面积并

有了以上讨论，我们很容易得到一个用水平离散化求圆面积并的算法。

1. 求出所有的圆的交点以及圆的上下端点，作为切割点
2. 过每个切割点作水平切割线
3. 从y_-到y_+扫描，将每个横条剖分成弓形和梯形，计算面积

具体怎样剖分每个横条呢？我们发现，在横条内任意画一条水平线，每个圆都对应其上的一段区间，而对于任意这样一条水平线，区间的关系都是相同的。

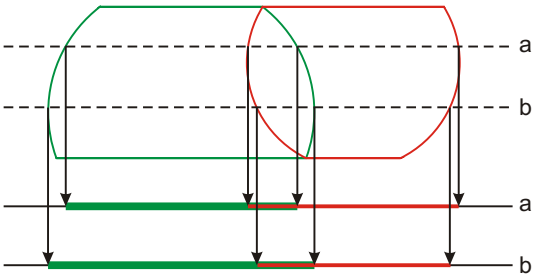

图 9: 任意水平线对应的区间关系相同

因此我们不妨取中线，将所有圆对应其上的区间求并，得到若干个独立的并区间。独立并区间的左右端点对应着横条剖分的弓形，内部点对应着横条剖分成的梯形（可能退化成三角形）。

有了剖分的方法我们就能计算出每个横条的面积了。采用朴素的方法对于每个横条扫描每个圆，时间复杂度是$O(n^3)$。

3.2.2 求周长

如果需要计算轮廓线周长，仍然是作刚才的水平切割线。所不同的是，我们只关心求并后的独立区间的左右端点对应的弧长。采用朴素的方法，时间复杂度是$O(n^3)$。

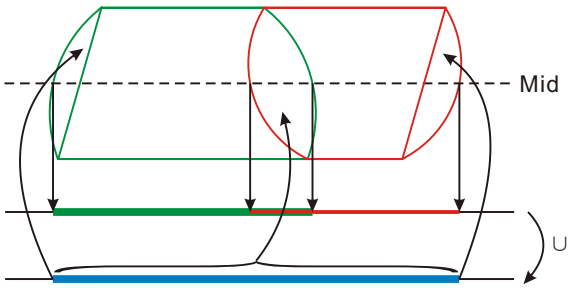

图 10: 独立并区间的各部分与基本形状的对应关系

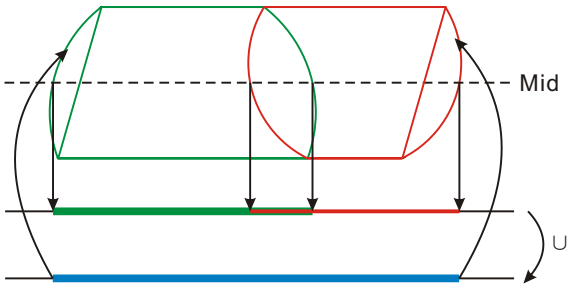

图 11: 稍作改动即可计算周长

3.2.3 判断可见性

离散化剖分之后，每个圆都对应若干个横条上的区间。如果一个圆对应的所有区间均不可见，那么这个圆就不可见。因此我们只要逐一扫描横条，判断每个横条上的区间可见性，就能确定所有圆的可见性了。采用朴素的方法，时间复杂度是$O(n^3)$。

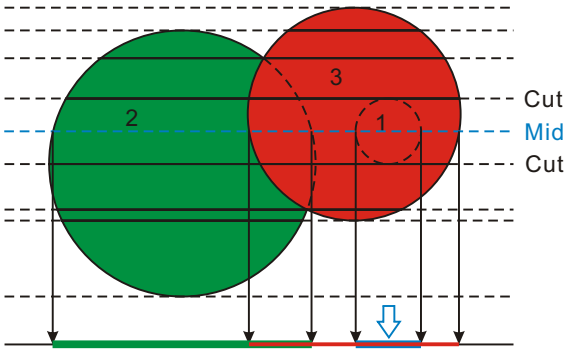

图 12: 圆3对应的所有区间(蓝色箭头所指的区间)均不可见，所以圆3不可见

4 极角离散化

4.1 核心思想

通过以上的讨论，我们似乎已经找到了解决圆覆盖问题的完美方法。但是还有没有更好的方法呢？

让我们尝试用另一种序——极角序进行离散化。

首先求出所有圆的交点。每个圆被其上的交点分成一些圆弧，称之为小圆弧（对应于元线段）。这些小圆弧不可再分，无需再分，其内部任意一点的可见性可以代表整段小圆弧的可见性。

4.2 例题

让我们看看怎样用极角离散化解决判断可见性的问题。

这个问题曾在日本区域赛中出现过，可以到这里提交代码。

分析问题，得到如下性质：

1. 任何一个圆都覆盖了一个闭区域。
2. 对于任意一个点，覆盖它的最上面的那个圆，一定是可见的。

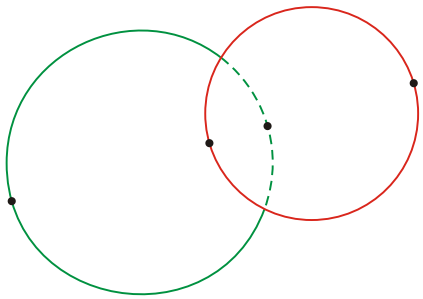

图 13: 极角序离散化——黑点是小圆弧中点，代表了小圆弧的可见性

3. 如果一个圆不可见（它被完全覆盖），那么它的边界是被完全覆盖的。
4. n个圆最多有$2(n-1)^2$个交点，这些交点把n个圆分成最多$2(n-1)^2$条小圆弧。
5. 对于每个小圆弧，要么它全被覆盖，要么它全不被覆盖。

根据1和2，问题转化为恰当地找出一些点，对于每个点，把覆盖它的最上面的圆标记为可见。

根据3，这些点一定在所有圆的边界集合内。

根据5，所有小圆弧构成边界集合。每个小圆弧上只要任意取一个点就能代表整个小圆弧（边界）。不妨取中点。

至此得到算法：取所有小圆弧的中点，对每个点找到覆盖它的最上面的圆。

根据4，最多取$2(n-1)^2$个点。对每个点找到覆盖它的最上面的圆，需要$O(n)$次运算。总复杂度是$O(n^3)$。

以下是我的代码，采用复数运算，总体上比水平离散化简洁很多。具体判断圆弧A上一个点是否被某圆C覆盖时，把这个点向A的圆心移动$\pm\epsilon$，如果均被C覆盖，那么认为该点是被覆盖的，否则认为该点不被覆盖。请读者仔细思考这是为什么。

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <complex>
#include <cmath>
using namespace std;
```

```
typedef complex <double> xy;

const double PI = acos(-1.0);

double normalize(double r) {
    while (r < 0.0) r += 2 * PI;
    while (r >= 2 * PI) r -= 2 * PI;
    return r;
}

int highest_cover(xy p, vector <xy> &points, vector <double>
    &rs) {
    for (int i = rs.size() - 1; i >= 0; i--)
        if (abs(points[i] - p) < rs[i])
            return i;
    return -1;
}

int main() {
    while (1) {
        int n;
        cin >> n;
        if (!n) break;
        vector <xy> points;
        vector <double> rs;
        for (int i = 0; i < n; i++) {
            double x, y, r;
            cin >> x >> y >> r;
            points.push_back(xy(x, y));
            rs.push_back(r);
        }
        vector <bool> visible(n, false);
        for (int i = 0; i < n; i++) {
            vector <double> rads;
            rads.push_back(0.0);
            rads.push_back(2.0 * PI);
            for (int j = 0; j < n; j++) {
                double a = rs[i];
                double b = abs(points[j] - points[i]);
                double c = rs[j];
                if (a + b < c || a + c < b || b + c < a)
                    continue;
```

```
                double d = arg(points[j] - points[i]);
                double e = acos((a * a + b * b - c * c) / (2
                    * a * b));
                rads.push_back(normalize(d + e));
                rads.push_back(normalize(d - e));
            }
            sort(rads.begin(), rads.end());
            for (int j = 0; j < rads.size() - 1; j++) {
                double rad = (rads[j + 1] + rads[j]) / 2.0;
                double diff = 4E-13;
                for (int k = -1; k <= 1; k += 2) {
                    int t = highest_cover(xy(points[i].real()
                        + (rs[i] + diff * k) * cos(rad),
                        points[i].imag() + (rs[i] + diff * k)
                        * sin(rad)), points, rs);
                    if (t != -1) visible[t] = true;
                }
            }
        }
        int ans = 0;
        for (int i = 0; i < n; i++)
            if (visible[i])
                ans++;
        cout << ans << endl;
    }
    return 0;
}
```

参考文献

[1] 《算法艺术与信息学竞赛》刘汝佳黄亮

[2] 《与圆有关的离散化方法》高逸涵

[3] 《数据结构的选择与算法效率》陈宏