最小圆覆盖 hdu 3007 - Howe_Young - 博客园

今天学习了一下最小圆覆盖, 看了一下午都没看懂, 晚上慢慢的摸索这代码,接合着别人的讲解, 画着图跟着代码一步一步的走着,竟然有些理解了.

最小圆覆盖:给定n个点, 求出半径最小的圆可以把这些点全部包围, 可以在圆的边界上

下面是我的个人理解. 如果不对, 还请路过大牛指出

先找一个点, 让圆心等于这个点的坐标, 半径等于0, 显然这个对的, 接着找下一个点, 如果只有两个点的话, 那么最小的圆一定是以他们为直径做圆, 接着找第三个点, 如果第三个点在园内或者在边界上, 那么不用更新当前的最小圆, 如果不在的话, 就要更新当前的最小圆了,使它包括这三个点, 那么更新的办法就是从他开始做圆, 依次判断它前面的点是否满足在最小圆内, 如果不在的话, 就需要根据两个点或者三个点来确定圆了, 它的外接圆最多三个点就确定了,刚开始一直不理解这个为什么三个点,后来画画图,走走就出来了. 我这可能说的比较笼统. 表达能力太差. 还是把大牛的原话拷过来把.

最小覆盖圆， 增量法

假设圆O是前i-1个点得最小覆盖圆，加入第i个点，如果在圆内或边上则什么也不做。否,新得到的最小覆盖圆肯定经过第i个点。

然后以第i个点为基础（半径为0），重复以上过程依次加入第j个点，若第j个点在圆外，则最小覆盖圆必经过第j个点。

重复以上步骤（因为最多需要三个点来确定这个最小覆盖圆，所以重复三次）。遍历完所有点之后，所得到的圆就是覆盖所有点得最小圆。

证明可以考虑这么做：

最小圆必定是可以通过不断放大半径，直到所有以任意点为圆心，半径为半径的圆存在交点，此时的半径就是最小圆。所以上述定理可以通过这个思想得到。这个做法复杂度是O(n)的，当加入圆的顺序随机时，因为三点定一圆，所以不在圆内概率是3/i,求出期望可得是O(n)。

下面是代码(模板)

```
/*************************************************************************
    > File Name:            hdu_3007.cpp
    > Author:               Howe_Young
    > Mail:                 1013410795@qq.com
    > Created Time:         2015年05月04日 星期一 18时42分33秒
 ************************************************************************/
/*最小圆覆盖*/
/*给定n个点, 让求半径最小的圆将n个点全部包围,可以在圆上*/
#include <cstdio>
#include <iostream>
#include <cstring>
#include <cmath>
#include <cstdlib>
#include <algorithm>
#define EPS 1e-8
using namespace std;
const int maxn = 550;
struct point{
    double x, y;
};
int sgn(double x)
{
    if (fabs(x) < EPS)
        return 0;
    return x < 0 ? -1 : 1;
}
double get_distance(const point a, const point b)//两点之间的距离
{
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
point get_circle_center(const point a, const point b, const point c)//得到三角形外接圆的圆心
{
    point center;
    double a1 = b.x - a.x;
    double b1 = b.y - a.y;
    double c1 = (a1 * a1 + b1 * b1) / 2.0;
    double a2 = c.x - a.x;
    double b2 = c.y - a.y;
    double c2 = (a2 * a2 + b2 * b2) / 2.0;
    double d = a1 * b2 - a2 * b1;
    center.x = a.x + (c1 * b2 - c2 * b1) / d;
    center.y = a.y + (a1 * c2 - a2 * c1) / d;
    return center;
}
//p表示定点, n表示顶点的个数, c代表最小覆盖圆圆心, r是半径
void min_cover_circle(point *p, int n, point &c, double &r)//找最小覆盖圆(这里没有用全局变量p[], 因为是为了封装一个函数便于调用)
{
```

```
    random_shuffle(p, p + n);//随机函数,使用了之后使程序更快点,也可以不用
    c = p[0];
    r = 0;
    for (int i = 1; i < n; i++)
    {
        if (sgn(get_distance(p[i], c) - r) > 0)//如果p[i]在当前圆的外面, 那么以当前点为圆心开始找
        {
            c = p[i];//圆心为当前点
            r = 0;//这时候这个圆只包括他自己.所以半径为0
            for (int j = 0; j < i; j++)//找它之前的所有点
            {
                if (sgn(get_distance(p[j], c) - r) > 0)//如果之前的点有不满足的, 那么就是以这两点为直径的圆
                {
                    c.x = (p[i].x + p[j].x) / 2.0;
                    c.y = (p[i].y + p[j].y) / 2.0;
                    r = get_distance(p[j], c);
                    for (int k = 0; k < j; k++)
                    {
                        if (sgn(get_distance(p[k], c) - r) > 0)//找新作出来的圆之前的点是否还有不满足的, 如果不满足一定就是三个点都在圆上
了
                        {
                            c = get_circle_center(p[i], p[j], p[k]);
                            r = get_distance(p[i], c);
                        }
                    }
                }
            }
        }
    }
}
int main()
{
    int n;
    point p[maxn];
    point c; double r;
    while (~scanf("%d", &n) && n)
    {
        for (int i = 0; i < n; i++)
            scanf("%lf %lf", &p[i].x, &p[i].y);
        min_cover_circle(p, n, c, r);
        printf("%.2lf %.2lf %.2lf\n", c.x, c.y, r);
    }
    return 0;
}
```