

计算几何

```
#include <bits/stdc++.h>
using namespace std;

const double eps = 1e-8;
const double inf = 1e20;
const double pi = acos(-1.0);
const int maxp = 200005;
const int maxn = 200005;
int sgn(const double x){
    if(fabs(x)<=eps) return 0;
    return x>0?1:-1;
}
inline double sqr(double x){return x*x;}
struct Point{
    double x,y;
    Point(){}
    Point(double _x,double _y){x=_x;y=_y;}
    void input(){scanf("%lf%lf",&x,&y);}
    void output(){printf("%.2f %.2f",x,y);}
    bool operator == (Point b) const{return sgn(x-b.x)==0&&sgn(y-b.y)==0;}
    bool operator < (Point b) const{return sgn(x-b.x)==0?sgn(y-b.y)<0:x<b.x;}
    Point operator +(const Point &b) const{return Point(x+b.x,y+b.y);}
    Point operator -(const Point &b) const{return Point(x-b.x,y-b.y);}
    Point operator *(const double &k) const{return Point(x*k,y*k);}
    Point operator /(const double &k) const{return Point(x/k,y/k);}
    double operator ^(const Point &b) const{return x*b.y-y*b.x;}
//叉积
    double operator *(const Point &b) const{return x*b.x+y*b.y;}
//点积
    double len(){return hypot(x,y);}           //求长度
    double len2(){return x*x+y*y;}
    double distance(Point p){return hypot(x-p.x,y-p.y);} //两点之间距离
    double rad(Point a,Point b){                  //求pa,pb所成角
        Point p=*this;
        return fabs(atan2(fabs((a-p)^*(b-p)),(a-p)*(b-p)));
    }
    Point trunc(double r){                     //转成长度为r的向量
        double l=len();
        if(!sgn(l))return *this;
        r/=l;
        return Point(x*r,y*r);
    }
    Point rotleft(){return Point(-y,x);}      //逆时针转九十度
    Point rotright(){return Point(y,-x);}     //顺时针转九十度
    Point rotate(Point p,double angle){        //绕p点转angle度
        Point v=(*this)-p;
        double c=cos(angle),s=sin(angle);
        return Point(p.x+v.x*c-v.y*s,p.y+v.x*s+v.y*c);
    }
};

struct Line{
    Point s,e;
```

```

Line(){}
Line(Point _s,Point _e){s=_s;e=_e;}
bool operator ==(Line v){return (s==v.s)&&(e==v.e);}
Line(Point p,double angle){           //根据点和倾斜角确定斜率 0<=angle<pi
    s=p;
    if(sgn(angle-pi/2)==0) e=(s+Point(0,1));
    else e=s+Point(1,tan(angle));
}
Line(double a,double b,double c){      //ax+by+c=0
    if(sgn(a)==0){s=Point(0,-c/b);e=Point(1,-c/b);}
    else if(sgn(b)==0){s=Point(-c/a,0);e=Point(-c/a,1);}
    else {s=Point(0,-c/b);e=Point(1,(-c-a)/b);}
}
void input(){s.input();e.input();}
void adjust(){if(e<s)swap(s,e);}
double length(){return s.distance(e);}
double angle(){           //求倾斜角
    double k=atan2(e.y-s.y,e.x-s.x);
    if(sgn(k)<0) k+=pi;
    if(sgn(k-pi)==0) k-=pi;
    return k;
}
int relation(Point p){           //点和直线关系: 1左侧, 2右侧, 3直线上
    int c=sgn((p-s)^*(e-s));           //注意直线要满足y小的为s！！！
    if(c<0) return 1;
    else if(c>0) return 2;
    else return 3;
}
bool pointonseg(Point p){return sgn((p-s)^*(e-s))==0&&sgn((p-s)^*(p-e))<=0;}
//判断点是否在线段上
bool parallel(Line v){return sgn((e-s)^*(v.e-v.s))==0;}           //判断两向量平行
int segcrossseg(Line v){
    int d1 = sgn((e-s)^*(v.s-s));           //线段和线段关系: 0不相交, 1非规范相交, 2规范相交
    int d2 = sgn((e-s)^*(v.e-s));
    int d3 = sgn((v.e-v.s)^*(s-v.s));
    int d4 = sgn((v.e-v.s)^*(e-v.s));
    if( (d1^d2)==-2 && (d3^d4)==-2 )return 2;
    return (d1==0 && sgn((v.s-s)^*(v.s-e))<=0) ||
(d2==0 && sgn((v.e-s)^*(v.e-e))<=0) ||
(d3==0 && sgn((s-v.s)^*(s-v.e))<=0) ||
(d4==0 && sgn((e-v.s)^*(e-v.e))<=0);
}
int linecrossseg(Line v){           //直线与线段v相交: 0不相交 1非规范相交 2规范相交
    int d1 = sgn((e-s)^*(v.s-s));
    int d2 = sgn((e-s)^*(v.e-s));
    if((d1^d2)==-2) return 2;
    return (d1==0 || d2==0);
}
int linecrossline(Line v){           //两直线关系0平行 1重合 2相交
    if(*this).parallel(v))
    return v.relation(s)==3;
    return 2;
}
Point crosspoint(Line v){           //求两直线交点
    double a1 = (v.e-v.s)^*(s-v.s);

```

```

        double a2 = (v.e-v.s)^(e-v.s);
        return Point((s.x*a2-e.x*a1)/(a2-a1),(s.y*a2-e.y*a1)/(a2-a1));
    }
    double dispointtoline(Point p){return fabs((p-s)^(e-s))/length();} //点到直线
的距离
    double dispointtoseg(Point p){                                         // 点到线段的距
离
        if(sgn((p-s)*(e-s))<0 || sgn((p-e)*(s-e))<0) return
min(p.distance(s),p.distance(e));
        return dispointtoline(p);
    }
    double dissegtoseg(Line v){return
min(min(dispointtoseg(v.s),dispointtoseg(v.e)),min(v.dispointtoseg(s),v.dispoint
toseg(e)));} //线段到线段的距离
    Point lineprog(Point p){return s + ((e-s)*(e-s)*(p-s))/(e-s).len2();}
);} //p在直线上的投影
    Point symmetrypoint(Point p){Point q = lineprog(p);return Point(2*q.x-
p.x,2*q.y-p.y);} //p关于直线的对称点
};

struct circle{
    Point p;      //圆心
    double r;     //半径
    circle(){}
    circle(Point _p,double _r){p = _p;r = _r;}
    circle(double x,double y,double _r){p = Point(x,y);r = _r;}
    circle(Point a,Point b,Point c){                     //三角形外接圆
        Line u = Line((a+b)/2,((a+b)/2)+((b-a).rotleft()));
        Line v = Line((b+c)/2,((b+c)/2)+((c-b).rotleft()));
        p = u.crosspoint(v);
        r = p.distance(a);
    }
    circle(Point a,Point b,Point c,bool t){           //三角形内切圆
        Line u,v;
        double m = atan2(b.y-a.y,b.x-a.x), n = atan2(c.y-a.y,c.x-a.x);
        u.s = a;
        u.e = u.s + Point(cos((n+m)/2),sin((n+m)/2));
        v.s = b;
        m = atan2(a.y-b.y,a.x-b.x), n = atan2(c.y-b.y,c.x-b.x);
        v.e = v.s + Point(cos((n+m)/2),sin((n+m)/2));
        p = u.crosspoint(v);
        r = Line(a,b).dispointtoseg(p);
    }
    void input(){p.input();scanf("%lf",&r);}
    void output(){printf("%.2f%.2f%.2f\n",p.x,p.y,r);}
    bool operator == (circle v){return (p==v.p) && sgn(r-v.r)==0;}
    bool operator < (circle v) const{return ((p<v.p)||((p==v.p)&&sgn(r-v.r)<0));}
    double area(){return pi*r*r;}
    double circumference(){return 2*pi*r;}
    int relation(Point b){                                //点和圆的关系: 0外1上2内
        double dst = b.distance(p);
        if(sgn(dst-r) < 0) return 2;
        else if(sgn(dst-r)==0) return 1;
        return 0;
    }
    int relationseg(Line v){                           //线段和圆的关系
        double dst = v.dispointtoseg(p);
        if(sgn(dst-r) < 0) return 2;
        else if(sgn(dst-r) == 0) return 1;
    }
}

```

```

        return 0;
    }
    int relationline(Line v){ //直线和圆的关系
        double dst = v.dispointtoline(p);
        if(sgn(dst-r) < 0) return 2;
        else if(sgn(dst-r) == 0) return 1;
        return 0;
    }
    int relationcircle(circle v){ //两个圆的关系: 5相离4外切3相交2
内切1内含
        double d = p.distance(v.p);
        if(sgn(d-r-v.r) > 0) return 5;
        if(sgn(d-r-v.r) == 0) return 4;
        double l = fabs(r-v.r);
        if(sgn(d-r-v.r)<0 && sgn(d-l)>0) return 3;
        if(sgn(d-l)==0) return 2;
        if(sgn(d-l)<0) return 1;
    }
    int pointcrosscircle(circle v,Point &p1,Point &p2){ //两个圆的交点数量 返回数量
与交点
        int rel = relationcircle(v);
        if(rel == 1 || rel == 5) return 0;
        double d = p.distance(v.p);
        double l = (d*d+r*r-v.r*v.r)/(2*d);
        double h = sqrt(r*r-l*l);
        Point tmp = p + (v.p-p).trunc(l);
        p1 = tmp + ((v.p-p).rotleft().trunc(h));
        p2 = tmp + ((v.p-p).rotright().trunc(h));
        if(rel == 2 || rel == 4) return 1;
        return 2;
    }
    int pointcrossline(Line v,Point &p1,Point &p2){ //直线与圆的交点数量 返回数量与交
点
        if(!(*this).relationline(v)) return 0;
        Point a = v.lineprog(p);
        double d = v.dispointtoline(p);
        d = sqrt(r*r-d*d);
        if(sgn(d) == 0){p1 = a;p2 = a;return 1;}
        p1 = a + (v.e-v.s).trunc(d);
        p2 = a - (v.e-v.s).trunc(d);
        return 2;
    }
    int gercircle(Point a,Point b,double r1,circle &c1,circle &c2){ //得到过a,b半径
为r1的两个圆
        circle x(a,r1),y(b,r1);
        int t = x.pointcrosscircle(y,c1.p,c2.p);
        if(!t) return 0;
        c1.r = c2.r = r1;
        return t;
    }
    int getcircle(Line u,Point q,double r1,circle &c1,circle &c2){ //得到与直线相
切, 过点q, 半径为r1的两个圆
        double dis = u.dispointtoline(q);
        if(sgn(dis-r1*2)>0) return 0;
        if(sgn(dis) == 0){
            c1.p = q + ((u.e-u.s).rotleft().trunc(r1));
            c2.p = q + ((u.e-u.s).rotright().trunc(r1));
            c1.r = c2.r = r1;
        }
    }
}

```

```

        return 2;
    }
    Line u1 = Line((u.s + (u.e-u.s).rotleft().trunc(r1)),(u.e +(u.e-
u.s).rotleft().trunc(r1)));
    Line u2 = Line((u.s + (u.e-u.s).rotright().trunc(r1)),(u.e +(u.e-
u.s).rotright().trunc(r1)));
    circle cc = circle(q,r1);
    Point p1,p2;
    if(!cc.pointcrossline(u1,p1,p2))cc.pointcrossline(u2,p1,p2);
    c1 = circle(p1,r1);
    if(p1 == p2){c2 = c1;return 1;}
    c2 = circle(p2,r1);
    return 2;
}
int getcircle(Line u,Line v,double r1,circle &c1,circle &c2,circle
&c3,circle &c4){//与u,v直线相切半径为r1的圆
if(u.parallel(v))return 0;//两直线平行
Line u1 = Line(u.s + (u.e-u.s).rotleft().trunc(r1),u.e +(u.e-
u.s).rotleft().trunc(r1));
Line u2 = Line(u.s + (u.e-u.s).rotright().trunc(r1),u.e +(u.e-
u.s).rotright().trunc(r1));
Line v1 = Line(v.s + (v.e-v.s).rotleft().trunc(r1),v.e +(v.e-
v.s).rotleft().trunc(r1));
Line v2 = Line(v.s + (v.e-v.s).rotright().trunc(r1),v.e +(v.e-
v.s).rotright().trunc(r1));
c1.r = c2.r = c3.r = c4.r = r1;
c1.p = u1.crosspoint(v1);
c2.p = u1.crosspoint(v2);
c3.p = u2.crosspoint(v1);
c4.p = u2.crosspoint(v2);
return 4;
}
int getcircle(circle cx,circle cy,double r1,circle &c1,circle &c2){ //与cx,cy
两圆相切, 半径为r1的圆
circle x(cx.p,r1+cx.r),y(cy.p,r1+cy.r);
int t = x.pointcrosscircle(y,c1.p,c2.p);
if(!t)return 0;
c1.r = c2.r = r1;
return t;
}
int tangentline(Point q,Line &u,Line &v){//过一点q作圆的切线
int x = relation(q);
if(x == 2) return 0;
if(x == 1){
    u = Line(q,q + (q-p).rotleft());
    v = u;
    return 1;
}
double d = p.distance(q);
double l = r*r/d;
double h = sqrt(r*r-l*l);
u = Line(q,p + ((q-p).trunc(l) + (q-p).rotleft().trunc(h)));
v = Line(q,p + ((q-p).trunc(l) + (q-p).rotright().trunc(h)));
return 2;
}
double areacircle(circle v){ //求两圆相交的面积
int rel = relationcircle(v);
if(rel >= 4) return 0.0;
}

```

```

        if(rel <= 2) return min(area(),v.area());
        double d = p.distance(v.p);
        double hf = (r+v.r+d)/2.0;
        double ss = 2*sqrt(hf*(hf-r)*(hf-v.r)*(hf-d));
        double a1 = acos((r*r+d*d-v.r*v.r)/(2.0*r*d));
        a1 = a1*r*r;
        double a2 = acos((v.r*v.r+d*d-r*r)/(2.0*v.r*d));
        a2 = a2*v.r*v.r;
        return a1+a2-ss;
    }
    double areatriangle(Point a,Point b){ //圆与三角形pab相交的面积
        if(sgn((p-a)^*(p-b)) == 0) return 0.0;
        Point q[5];
        int len = 0;
        q[len++] = a;
        Line l(a,b);
        Point p1,p2;
        if(pointcrossline(l,q[1],q[2])==2){
            if(sgn((a-q[1])*(b-q[1]))<0)q[len++] = q[1];
            if(sgn((a-q[2])*(b-q[2]))<0)q[len++] = q[2];
        }
        q[len++] = b;
        if(len == 4 && sgn((q[0]-q[1])*(q[2]-q[1]))>0)swap(q[1],q[2]);
        double res = 0;
        for(int i = 0;i < len-1;i++){
            if(relation(q[i])==0||relation(q[i+1])==0){
                double arg = p.rad(q[i],q[i+1]);
                res += r*r*arg/2.0;
            }
            else res += fabs((q[i]-p)^*(q[i+1]-p))/2.0;
        }
        return res;
    }
};

struct polygon{
    int n;
    Point p[maxp];
    Line l[maxp];
    void input(int _n){n=_n;for(int i=0;i<n;i++) p[i].input();}
    void add(Point q){p[n++]=q;}
    void getline(){for(int i=0;i<n;i++){l[i]=Line(p[i],p[(i+1)%n]);}}
    struct cmp{
        Point p;
        cmp(const Point &p0){p=p0;}
        bool operator()(const Point &aa,const Point &bb){
            Point a=aa,b=bb;
            int d=sgn((a-p)^*(b-p));
            if(d==0) return sgn(a.distance(p)-b.distance(p))<0;
            return d>0;
        }
    };
    void norm(){ //极角排序
        Point mi=p[0];
        for(int i=1;i<n;i++) mi=min(mi,p[i]);
        sort(p,p+n,cmp(mi));
    }
    void getconvex(polygon &convex){
        sort(p,p+n);
    }
};

```

```

convex.n=n;
for(int i=0;i<min(n,2);i++) convex.p[i]=p[i];
if(convex.n==2&&(convex.p[0]==convex.p[1])) convex.n--;
if(n<=2) return ;
int &top=convex.n;
top=1;
for(int i=2;i<n;i++){
    while (top&&sgn((convex.p[top]-p[i])^(convex.p[top-1]-p[i]))<=0)
top--;
    convex.p[+top]=p[i];
}
int temp=top;
convex.p[+top]=p[n-2];
for(int i=n-3;i>=0;i--){
    while (top!=temp&&sgn((convex.p[top]-p[i])^(convex.p[top-1]-p[i]))<=0) top--;
    convex.p[+top]=p[i];
}
if(convex.n==2&&(convex.p[0]==convex.p[1])) convex.n--;
convex.norm();
}

bool isconvex(){ //判断是不是凸的
    bool s[2];
    memset(s,false,sizeof(s));
    for(int i = 0;i < n;i++){
        int j = (i+1)%n;
        int k = (j+1)%n;
        s[sgn((p[j]-p[i])^(p[k]-p[i]))+1] = true;
        if(s[0] && s[2]) return false;
    }
    return true;
}

int relationpoint(Point q){ //判断点和任意多边形的关系
    for(int i = 0;i < n;i++){if(p[i] == q) return 3;}
    getline(); //3 点上 2 边上 1 内部 0 外部
    for(int i = 0;i < n;i++) {if(l[i].pointonseg(q))return 2;}
    int cnt = 0;
    for(int i = 0;i < n;i++){
        int j = (i+1)%n;
        int k = sgn((q-p[j])^(p[i]-p[j]));
        int u = sgn(p[i].y-q.y);
        int v = sgn(p[j].y-q.y);
        if(k > 0 && u < 0 && v >= 0)cnt++;
        if(k < 0 && v < 0 && u >= 0)cnt--;
    }
    return cnt != 0;
}

void convexecut(Line u,polygon &po){
    int &top = po.n; //注意引用
    top = 0;
    for(int i = 0;i < n;i++){
        int d1 = sgn((u.e-u.s)^(p[i]-u.s));
        int d2 = sgn((u.e-u.s)^(p[(i+1)%n]-u.s));
        if(d1 >= 0)po.p[top++]= p[i];
        if(d1*d2 < 0)po.p[top++] = u.crosspoint(Line(p[i],p[(i+1)%n]));
    }
}

double getcircumference(){

```

```

        double sum = 0;
        for(int i = 0;i < n;i++){sum += p[i].distance(p[(i+1)%n]);}
        return sum;
    }
    double getarea(){
        double sum = 0;
        for(int i = 0;i < n;i++){sum += (p[i]^p[(i+1)%n]);}
        return fabs(sum)/2;
    }
    bool getdir(){      //得到方向1逆时针0顺时针
        double sum = 0;
        for(int i = 0;i < n;i++)sum += (p[i]^p[(i+1)%n]);
        if(sgn(sum) > 0)return 1;
        return 0;
    }
    Point getbarycentre(){           //返回重心
        Point ret(0,0);
        double area = 0;
        for(int i = 1;i < n-1;i++){
            double tmp = (p[i]-p[0])^(p[i+1]-p[0]);
            if(sgn(tmp) == 0)continue;
            area += tmp;
            ret.x += (p[0].x+p[i].x+p[i+1].x)/3*tmp;
            ret.y += (p[0].y+p[i].y+p[i+1].y)/3*tmp;
        }
        if(sgn(area)) ret = ret/area;
        return ret;
    }
    double areacircle(circle c){
        double ans = 0;
        for(int i = 0;i < n;i++){
            int j = (i+1)%n;
            if(sgn((p[j]-c.p)^p[i]-c.p) ) >= 0) ans += c.areastriangle(p[i],p[j]);
            else ans -= c.areastriangle(p[i],p[j]);
        }
        return fabs(ans);
    }
    int relationcircle(circle c){      //多边形与圆位置关系: 2圆完全在多边形里, 1圆触碰
        to多边形边界, 0其他
        getline();
        int x = 2;
        if(relationpoint(c.p) != 1)return 0;//圆心不在内部
        for(int i = 0;i < n;i++){
            if(c.relationseg(l[i])==2)return 0;
            if(c.relationseg(l[i])==1)x = 1;
        }
        return x;
    }
};

double cross(Point A,Point B,Point C){return (B-A)^C-A;}          //AB X AC
double dot(Point A,Point B,Point C){return (B-A)*(C-A);}          //AB * AC
double minRectangleCover(polygon A){
    if(A.n < 3)return 0.0; //要特判 A.n < 3 的情况
    A.p[A.n] = A.p[0];
    double ans = -1;
    int r = 1, p = 1, q;
    for(int i = 0;i < A.n;i++){//卡出离边 A.p[i] - A.p[i+1] 最远的点

```

```

        while( sgn( cross(A.p[i],A.p[i+1],A.p[r+1]) -
cross(A.p[i],A.p[i+1],A.p[r]) ) >= 0 ) r = (r+1)%A.n;
        while(sgn( dot(A.p[i],A.p[i+1],A.p[p+1]) - dot(A.p[i],A.p[i+1],A.p[p]) )
>= 0 )//卡出 A.p[i] - A.p[i+1] 方向上正向 n 最远的点
        p = (p+1)%A.n;
        if(i == 0)q = p;
        while(sgn(dot(A.p[i],A.p[i+1],A.p[q+1]) - dot(A.p[i],A.p[i+1],A.p[q])) 
<= 0)//卡出 A.p[i] - A.p[i+1] 方向上负向最远的点
        q = (q+1)%A.n;
        double d = (A.p[i] - A.p[i+1]).len2();
        double tmp = cross(A.p[i],A.p[i+1],A.p[r]) *(dot(A.p[i],A.p[i+1],A.p[p]) 
- dot(A.p[i],A.p[i+1],A.p[q]))/d;
        if(ans < 0 || ans > tmp)ans = tmp;
    }
    return ans;
}

struct halfplane:public Line{
    double angle;
    halfplane(){//表示向量 s->e 逆时针 (左侧) 的半平面
    halfplane(Point _s,Point _e){s = _s;e = _e;}
    halfplane(Line v){s = v.s;e = v.e;}
    void calcangle(){angle = atan2(e.y-s.y,e.x-s.x);}
    bool operator <(const halfplane &b) const{return angle < b.angle;}
};

struct halfplanes{
    int n;
    halfplane hp[maxp];
    Point p[maxp];
    int que[maxp],st,ed;
    void push(halfplane tmp){hp[n++] = tmp;}
    void unique(){//去重
        int m = 1;
        for(int i = 1;i < n;i++){
            if(sgn(hp[i].angle-hp[i-1].angle) != 0)hp[m++] = hp[i];
            else if(sgn( (hp[m-1].e-hp[m-1].s)*(hp[i].s-hp[m-1].s)) > 0) hp[m-1]
= hp[i];
        }
        n = m;
    }
    bool halfplaneinsert(){
        for(int i = 0;i < n;i++)hp[i].calcangle();
        sort(hp,hp+n);
        unique();
        que[st=0] = 0;que[ed=1] = 1;
        p[1] = hp[0].crosspoint(hp[1]);
        for(int i = 2;i < n;i++){
            while(st<ed && sgn((hp[i].e-hp[i].s)*(p[ed]-hp[i].s))<0)ed--;
            while(st<ed && sgn((hp[i].e-hp[i].s)*(p[st+1]-hp[i].s))<0)st++;
            que[++ed] = i;
            if(hp[i].parallel(hp[que[ed-1]]))return false;
            p[ed]=hp[i].crosspoint(hp[que[ed-1]]);
        }
        while(st<ed && sgn((hp[que[st]].e-hp[que[st]].s)*(p[ed]-hp[que[st]].s))
<0) ed--;
        while(st<ed && sgn((hp[que[ed]].e-hp[que[ed]].s)*(p[st+1]-
hp[que[ed]].s))<0)st++;
        if(st+1>=ed) return false;
        return true;
    }
}

```

```

}

void getconvex(polygon &con){//得到最后半平面交得到的凸多边形 需要先调用
halfplaneinsert() 且返回 true
    p[st] = hp[que[st]].crosspoint(hp[que[ed]]);
    con.n = ed-st+1;
    for(int j = st,i = 0;j <= ed;i++,j++) con.p[i] = p[j];
}
};

struct circles{
    circle c[maxn];
    double ans[maxn],pre[maxn];//ans[i] 表示被覆盖了 i 次的面积
    int n;
    circles(){}
    void add(circle cc){c[n++] = cc;}
    bool inner(circle x,circle y){//x 包含在 y 中
        if(x.relationcircle(y) != 1) return 0;
        return sgn(x.r-y.r)<=0?1:0;
    }
    void init_or{//圆的面积并去掉内含的圆
        bool mark[maxn] = {0};
        int i,j,k=0;
        for(i = 0;i < n;i++){
            for(j = 0;j < n;j++){
                if(i != j && !mark[j]){
                    if( (c[i]==c[j]) || inner(c[i],c[j]) )break;
                }
                if(j < n)mark[i] = 1;
            }
            for(i = 0;i < n;i++){
                if(!mark[i]) c[k++] = c[i];
                n = k;
            }
        }
    }
    void init_add{//圆的面积交去掉内含的圆
        int i,j,k;
        bool mark[maxn] = {0};
        for(i = 0;i < n;i++){
            for(j = 0;j < n;j++){
                if(i != j && !mark[j]){
                    if( (c[i]==c[j]) || inner(c[j],c[i]) )break;
                }
            }
            if(j < n)mark[i] = 1;
        }
        for(i = 0;i < n;i++){
            if(!mark[i]) c[k++] = c[i];
            n = k;
        }
    }
}//半径为 r 的圆，弧度为 th 对应的弓形的面积
double areaarc(double th,double r){return 0.5*r*r*(th-sin(th));}
void getarea{//求 n 个圆并的面积，需要加上 init_or() 去掉重复圆（否则WA）
    memset(ans,0,sizeof(ans));//SPOJCIRUT 是求被覆盖 k 次的面积，不能加 init_or()
    vector<pair<double,int>>v;//对于求覆盖多少次面积的问题，不能解决相同圆，而且不能
init_or()
    for(int i = 0;i < n;i++){
        v.clear();
        v.push_back(make_pair(-pi,1));
        v.push_back(make_pair(pi,-1));

```

```

        for(int j = 0;j < n;j++){
            if(i != j){
                Point q = (c[j].p - c[i].p);
                double ab = q.len(),ac = c[i].r, bc = c[j].r;
                if(sgn(ab+ac-bc)<=0){
                    v.push_back(make_pair(-pi,1));
                    v.push_back(make_pair(pi,-1));
                    continue;
                }
                if(sgn(ab+bc-ac)<=0)continue;
                if(sgn(ab-ac-bc)>0)continue;
                double th = atan2(q.y,q.x), fai = acos((ac*ac+ab*ab-
bc*bc)/(2.0*ac*ab));
                double a0 = th-fai;
                if(sgn(a0+pi)<0)a0+=2*pi;
                double a1 = th+fai;
                if(sgn(a1-pi)>0)a1-=2*pi;
                if(sgn(a0-a1)>0){
                    v.push_back(make_pair(a0,1));
                    v.push_back(make_pair(pi,-1));
                    v.push_back(make_pair(-pi,1));
                    v.push_back(make_pair(a1,-1));
                }
                else{
                    v.push_back(make_pair(a0,1));
                    v.push_back(make_pair(a1,-1));
                }
            }
        }
        sort(v.begin(),v.end());
        int cur = 0;
        for(int j = 0;j < v.size();j++){
            if(cur && sgn(v[j].first-pre[cur])){
                ans[cur] += areaarc(v[j].first-pre[cur],c[i].r);
                ans[cur] += 0.5*
                (Point(c[i].p.x+c[i].r*cos(pre[cur]),c[i].p.y+c[i].r*sin(pre[cur]))^Point(c[i].p
.x+c[i].r*cos(v[j].first),c[i].p.y+c[i].r*sin(v[j].first)));
            }
            cur += v[j].second;
            pre[cur] = v[j].first;
        }
    }
    for(int i = 1;i < n;i++) ans[i] -= ans[i+1];
}
};

int main(){
    return 0;
}

```

