几何

原始库

```
#define maxn 2000

int sig(double d);
struct Point;
Point rotate(Point p, double radian);
double cross(Point o, Point a, Point b);
double dot(Point &o, Point &a, Point &b);
double dot(Point &a, Point &b);
double dis(Point a, Point b);
double cos(Point o, Point a, Point b);
double sin(Point o, Point a, Point b);
int btw(Point &x, Point &a, Point &b);
int segCross(Point a, Point b, Point c, Point d, Point &p);
int segLineCross(Point a, Point b, Point c, Point d, Point &p);
```

```
int lineCross(Point a, Point b, Point c, Point d, Point &p);
struct Circle;
int graham(Point *p, int n, int *ch);
int jarvis(Point *p, int n, int *ch);
double MEC(Point *p, int n, Point &center);
int MEP(Point*p, int n);
void calPolygon(Point*p, int num, double &area, double &perimeter, double &gX, double
&gY, bool &shun);
struct PointInt;
void pick(PointInt *p, int num, int &A, int &E, int &I);
struct VerSeg;
void rectCal(VerSeg*segs, int n, double &area, double &perimeter);
struct Rec;
void recDfs(int step, int cnt, Rec tmp);
void zzy(Line* l, int n, int &b, int &t);
bool calCore(Point *ps, int &n);
int find_intersections(VS segs);
int sig(double d) {
    return fabs(d) < 1E-6 ? 0 : d < 0 ? -1 : 1;
}
/**
    点的结构体
*/
struct Point {
    double x, y;
    double k;
    Point(){}
    Point(double x, double y): x(x), y(y) {}
    void set(double x, double y) {
        this->x = x;
        this->y = y;
    }
    double mod() {
        return sqrt(x*x+y*y);
    }
    double mod_pow() {
        return x*x + y*y;
    }
    void output() {
        printf("x = %f, y = %f\n", x, y);
    }
    bool operator < (const Point &p) const {
        return sig(x-p.x) != 0 ? x < p.x : sig(y-p.y) < 0;
    }
};


 /**
    向量p绕着圆点转动radian（弧度）
    返回得到的点
*/
Point rotate(Point p, double radian) {
    double c = cos(radian), s = sin(radian);
    p.set(p.x*c-p.y*s, p.y*c+p.x*s);
    return p;
}
/**
    叉乘
    --------------------------
    返回 oa * ob;
*/
double cross(Point o, Point a, Point b) {
```

```
    return (a.x - o.x)*(b.y - o.y)-(b.x - o.x)*(a.y - o.y);
}
/**
    点积
    --------------------------
    返回 oa •ob;
*/
double dot(Point &o, Point &a, Point &b) {
    return (a.x-o.x)*(b.x-o.x) + (a.y-o.y)*(b.y-o.y);
}
/**
    点积
    --------------------------
    返回 a •b;
*/
double dot(Point &a, Point &b) {
    return a.x*b.x + a.y*b.y;  //(a.x-o.x)*(b.x-o.x) + (a.y-o.y)*(b.y-o.y);
}
/**
    两点距离
    ----------------------
*/
double dis(Point a, Point b) {
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
/**
    oa, ob夹角的余弦值(弧度制)
*/
double cos(Point o, Point a, Point b) {
    return dot(o,a,b)/dis(o,a)/dis(o,b);
}
/**
    oa, ob夹角的正弦值(弧度制)
*/
double sin(Point o, Point a, Point b) {
    return fabs(cross(o,a,b)/dis(o,a)/dis(o,b));
}
/*
    前提假设a、b、x共线
    返回:
        x在seg(a,b)内: -1
        x在seg(a,b)上: 0
        x在seg(a,b)外: 1
*/
int btw(Point &x, Point &a, Point &b) {
    return sig(dot(x, a, b));
}
/*  判断线段a,b  和  c,d是否相交

    类型         返回 p
---------------------------------------
1.  不相交           0       不变
2.  规范相交     1       交点
3.  非规范相交       2       不变
*/
int segCross(Point a, Point b, Point c, Point d, Point &p) {
    double s1, s2;
    int d1, d2, d3, d4;
    d1 = sig(s1=cross(a,b,c));
    d2 = sig(s2=cross(a,b,d));
```

```
    d3 = sig(cross(c,d,a));
    d4 = sig(cross(c,d,b));
    if((d1^d2)==-2 && (d3^d4)==-2) {
        p.x = (c.x*s2-d.x*s1)/(s2-s1);
        p.y = (c.y*s2-d.y*s1)/(s2-s1);
        return 1;
    }
    if( d1==0 && btw(c,a,b)<=0 ||
        d2==0 && btw(d,a,b)<=0 ||
        d3==0 && btw(a,c,d)<=0 ||
        d4==0 && btw(b,c,d)<=0)
        return 2;
    return 0;
}
/**
    和segCross(...)类似
    只是ab为直线，cd为线段
*/
int segLineCross(Point a, Point b, Point c, Point d, Point &p) {
    double s1, s2;
    int d1, d2;
    d1 = sig(s1=cross(a,b,c));
    d2 = sig(s2=cross(a,b,d));
    if((d1^d2)==-2) {
        p.x = (c.x*s2-d.x*s1)/(s2-s1);
        p.y = (c.y*s2-d.y*s1)/(s2-s1);
        return 1;
    }
    if(d1==0 || d2==0)  return 2;
    return 0;
}
/*
    判断直线a,b   和   c,d是否相交

    类型             返回 p
---------------------------------------
1.  不相交（平行）        0       不变
2.  规范相交          1       交点
3.  非规范相交（重合） 2       不变
*/
int lineCross(Point a, Point b, Point c, Point d, Point &p) {
    double s1, s2;
    s1=cross(a,b,c);
    s2=cross(a,b,d);
    if(sig(s1)==0 && sig(s2)==0)   return 2;
    if(sig(s2-s1)==0)   return 0;
    p.x = (c.x*s2-d.x*s1)/(s2-s1);
    p.y = (c.y*s2-d.y*s1)/(s2-s1);
    return 1;
}
struct Circle {
    Point center;
    double radis;
    bool contains(Point& p) {
        return sig(dis(center, p)-radis) <= 0;
    }
    void output() {
        printf("%.2f %.2f %.2f\n", center.x, center.y, radis);
    }
};
int g_cmp(const void *a, const void *b) {
    int d = sig(((Point*)a)->y-((Point*)b)->y);
```

```
    return d ? d : sig(((Point*)a)->x-((Point*)b)->x);
}
int graham(Point*p, int n, int*ch) {
#define push(x)    ch[len ++]=x
#define pop()      len --
sort(p, p+n);
    int len = 0, len0 = 1, i;
    for(i = 0; i < n; i ++) {
        while(len > len0 && sig(cross(p[ch[len-1]], p[ch[len-2]], p[i]))<=0)
            pop();
        push(i);
    }
    len0 = len;
    for(i = n-2; i >= 0; i --) {
        while(len > len0 && sig(cross(p[ch[len-1]], p[ch[len-2]], p[i]))<=0)
            pop();
        push(i);
    }
    return len-1;
}
/**
    凸包: jarvis步进法
    ------------------
    p: 原始的点
    n: 点的个数
    ch:存储凸包的点（回路，首位相接）

    与graham不同，不会改变p的位置
*/
int jarvis(Point *p, int n, int *ch) {
    static int d, i, o, s, l, t;
    for(d = i = 0; i < n; i ++)
        if(p[i] < p[d])
            d = i;
    l = s = *ch = d;
    d = 1;
    do {
        o = l;
        for(i = 0; i < n; i ++)
            if((t=sig(cross(p[o], p[l], p[i])))>0
|| (t==0 && btw(p[l], p[o], p[i])<=0))
                l = i;
        ch[d ++] = l;
    } while(l != s);
    return d-1;
}
/**
    minimal enclosing circle(最小覆盖圆)
    ------------------------------------
    p: 点集合
    n: 个数
    center: 存储这些点的中心
    返回:半径

    效率：nlgn

    需要调用Jarvis等函数
*/
double MEC(Point *p, int n, Point &center) {
#define g1(a,b,c) p##a.x = pp##b.x-pp##c.x;    p##a.y = pp##b.y-pp##c.y
    static int idx, i, ch[1000010], num, s1, s2;
```

```
    static double tmp, cos_v, cos_s1, cos_s2, a, b, c, d;
    static Point p1, p2, p3;
    num = graham(p, n, ch);
    s1 = 0, s2 = 1;
    while(1) {
        idx = 0;
        cos_v = -100;
        Point &pp1 = p[ch[s1]], &pp2 = p[ch[s2]];
        for(i = 0; i < num; i ++) {
            tmp = cos(p[ch[i]], pp1, pp2);
            if(tmp > cos_v) {
                cos_v = tmp;
                idx = i;
            }
        }
        Point &pp3 = p[ch[idx]];
        if(sig(cos_v) <= 0) {
            center.x = (pp1.x+pp2.x)/2.0;
            center.y = (pp1.y+pp2.y)/2.0;
            break;
        }
        cos_s1 = cos(pp1, pp2, pp3);
        cos_s2 = cos(pp2, pp1, pp3);

        if(sig(cos_s1)>=0 && sig(cos_s2)>=0) { //这个三角形就是
            g1(1,2,3);  g1(2,1,3);  g1(3,1,2);
            d = cross(pp2, pp1, pp3);
            d = d*d*2.0;
            a = p1.mod_pow()*dot(p3,p2)/d;
            b = -p2.mod_pow()*dot(p3,p1)/d;
            c = p3.mod_pow()*dot(p2,p1)/d;
            center.x = a*pp1.x+b*pp2.x+c*pp3.x;
            center.y = a*pp1.y+b*pp2.y+c*pp3.y;
            break;
        }
        if(sig(cos_s1)<0)   s1 = idx;
        else            s2 = idx;
    }
    return dis(center, p[ch[s1]]);
}

/**
    maximum enclosing points
    单位圆的最多覆盖点
*/
struct MEP_Nod {
    double A;   //极角[0, 2*M_PI)
    int M;      //模式：1为进入，2为退出
    bool operator < (const MEP_Nod &nod) const {
        return sig(A-nod.A) ? A < nod.A : M > nod.M;
    }
    void set(double A, int M) {
        if(sig(A-2*M_PI)>=0)    A -= 2*M_PI;
        else if(sig(A)<0)       A += 2*M_PI;
        this->A = A;    this->M = M;
    }
};
int MEP(Point*p, int n) {
    static MEP_Nod N[2000];
    int s, tmp, tmpMax, ans, i, j;
    double theta, alpha, d, di;
    ans = 0;
```

```
    for(i = 0; i < n; i ++) {
        s = 0;
        for(j = 0; j < n; j ++) {
            if(i==j || sig((di=dis(p[i], p[j]))-2)>0)      continue;
            theta = atan2(p[j].y-p[i].y, p[j].x-p[i].x);
            alpha = acos(di / 2.0);
            N[s++].set(theta - alpha, 1);
            N[s++].set(theta + alpha, -1);
            if(sig(N[s-2].A-N[s-1].A) > 0) {
                d = N[s-1].A;
                N[s-1].A = 2*M_PI;
                N[s ++].set(0, 1);
                N[s ++].set(d, -1);
            }
        }
        sort(N, N+s);
        tmp = tmpMax = 0;
        for(j = 0; j < s; j ++) {
            tmp += N[j].M;
            if(tmp > tmpMax)    tmpMax = tmp;
        }
        if(tmpMax > ans)    ans = tmpMax;
    }
    return ans+1;
}
/**
    多边形的面积、重心、周长、转向 的计算
    要求：多边形首尾相接(p[n] = p[0])
-------------------------------
    参数:
        p:      多边形的顶点（要求p[n]=p[0]）
        num:    多边形顶点个数

        area:   返回面积
        perimeter:返回周长
        gX  :   返回重心横坐标
        gY  :   返回重心纵坐标
        shun:   true代表顺时针
*/
void calPolygon(Point*p, int num, double &area, double &perimeter, double &gX, double
&gY, bool &shun) {
    area = gX = gY = perimeter = 0;
    double tmp;
    for(int i=0; i < num; i ++) {
        tmp = p[i].x*p[i+1].y - p[i].y*p[i+1].x;
        area += tmp;
        gX += tmp*(p[i].x + p[i+1].x);
        gY += tmp*(p[i].y + p[i+1].y);
        perimeter += dis(p[i], p[i+1]);
    }
    area /= 2.0;
    gX /= 6*area;
    gY /= 6*area;
    if(shun=area<0)
        area = -area;
}
/**
    皮克定理
        输入：顶点为整点的多边形(顶点n个，首尾相接)
        得到:   1.面积      A
                2.边上点数   E
```

```
                3.内部点数    I
-------------------------------------
    注意:
        1.依据皮克公式: A = E / 2 + I - 1, 其中:
                A: 为多边形(顶点都在格点上)面积
                E: 为多边形边上的格点数
                I: 为多边形内部的格点数
        2.参数中的A将设为: 2*面积
        3.需要调用gcd(int,int)函数
*/
struct PointInt {
    int x, y;
};
int gcd(int a,int b) {
    static int t;
    for(; t=b; b=a%b,a=t);
    return a;
}
void pick(PointInt *p, int num, int &A, int &E, int &I) {
    A = E = 0;
    for(int i = 0; i < num; i ++) {
        A += p[i].x*p[i+1].y - p[i].y*p[i+1].x;
        E += abs(gcd(p[i].x-p[i+1].x, p[i].y-p[i+1].y));
    }
    A = abs(A);
    I = (A-E)/2+1;
}
/**
    计算矩形的面积和周长
    -----------------------------
    需要掉用线段树,
    传入:
            VerSeg(矩形的竖直线,注意y1 < y2)
            n    竖直线个数(为矩形数的2倍)
*/

struct VerSeg {
    double x, y1, y2;
    int mode;//1为进入线段, -1为退出线段
    bool operator <(const VerSeg& vs) const {
        return x < vs.x;
    }
    void set(double x, double y1, double y2, int mode) {
        this->x = x;
        this->y1 = y1;
        this->y2 = y2;
        this->mode = mode;
    }
};
double val[2*maxn];//保存离散化后的数值

struct SegTree {
#define SIZE 2*maxn+1000
#define MEM 2*SIZE+10
#define SET(x) memset(x, 0, sizeof(x))
    int L[MEM], R[MEM], V[MEM];//L、R:线段的区间范围, V:线段的值
    double M[MEM],C[MEM];       //M此区间包含的非零线段长度, C此区间包含的连续线段个数, S
此线段的V+递归子线段的V
    bool P[MEM], Q[MEM];        //P,Q左右端点是否被覆盖, 辅助C
    int n;
```

```
    void init(int size) {
        for(n = 1; n < size; n <<= 1);
        int i;
        for(i = n; i < 2*n; i ++) {
            L[i] = i-n;
            R[i] = i+1-n;
        }
        for(i = n-1; i >= 1; i --) {
            L[i] = L[2*i];
            R[i] = R[2*i+1];
        }
        SET(V); SET(M); SET(C); SET(P); SET(Q);
    }
    void update1(int i) {//设置M 和 C
        if(V[i]) {
            M[i] = val[R[i]] - val[L[i]];
        } else if(i >= n) {
            M[i] = 0;
        } else {
            M[i] = M[2*i]+M[2*i+1];
        }
    }
    void update2(int i) {//设置C
        if(V[i]) {
            P[i] = Q[i] = C[i] = 1;
        } else if(i >= n) {
            P[i] = Q[i] = C[i] = 0;
        } else {
            P[i] = P[2*i];
            Q[i] = Q[2*i+1];
            C[i] = C[2*i]+C[2*i+1] - Q[2*i]*P[2*i+1];
        }
    }
    void insert(int l, int r, int v, int i=1) {
        if(l <= L[i] && r >= R[i]) V[i] += v;
        else {
            if(l < (L[i]+R[i])/2)  insert(l, r, v, 2*i);
            if(r > (L[i]+R[i])/2)  insert(l, r, v, 2*i+1);
        }
        update1(i);
        update2(i);
    }
} st;

void rectCal(VerSeg*segs, int n, double &area, double &perimeter) {
    map<double, int> ma;
    int i;
    double lastM, lastX;
    for(i = 0; i < n; i ++) {
        ma[segs[i].y1];    ma[segs[i].y2];
    }
    i = 0;
    memset(val, 0, sizeof(val));
    for(map<double,int>::iterator iter = ma.begin(); iter!=ma.end(); iter ++, i ++)
{
        iter->second = i;
        val[i] = iter->first;
    }
    sort(segs, segs+n);
    st.init(n);
    area = perimeter = 0;
    lastX = lastM = 0;
    for(i = 0; i < n; i ++) {
```

```
        lastM = st.M[1];
        area += st.M[1] * (segs[i].x - lastX);
        perimeter += 2 * st.C[1] * (segs[i].x - lastX);
        st.insert(ma[segs[i].y1], ma[segs[i].y2], segs[i].mode);
        perimeter += fabs(lastM - st.M[1]);
        lastX = segs[i].x;
    }
}
/**
    求解相交矩形面积---容斥原理
    ---------------------------
*/
struct Rec {
    int x1, y1, x2, y2;
    int square() {
        return (y2-y1)*(x2-x1);
    }
};
Rec rec[25], full;
int num, area;
#define max(a,b)    a>b?a:b
#define min(a,b)    a<b?a:b
void init() {
    //将full设置为最大场景
    full.x1 = full.y1 = 0;
    full.x2 = full.y2 = 1000;
    area = full.square();
}
bool intersect(Rec&r, Rec s) {
    r.x1 = max(r.x1, s.x1);
    r.x2 = min(r.x2, s.x2);
    r.y1 = max(r.y1, s.y1);
    r.y2 = min(r.y2, s.y2);
    return r.x1<r.x2 && r.y1<r.y2;
}
void recDfs(int step, int cnt, Rec tmp) {
    if(step == num) {
        if(cnt&1)   area += tmp.square();
        else        area -= tmp.square();
        return;
    }
    recDfs(step+1, cnt, tmp);
    if(intersect(tmp, rec[step]))
        recDfs(step+1, cnt+1,tmp);
}
/** 下面开始计算半平面交!!!!!!!!!!!
    如果直线l1,l2,l3过一个点，未删除中间的那条边！（照顾点的退化情况）因此核可能有重复的点
    退化情况：
        1.如果是多边形，无大碍，参考calCore函数
        2.如果是线性规划，建议加入边界来约束
*/
/**
    精简了上面的calPolygon，只保留了area和shun的返回
*/
void calPolygon(Point*p, int num, double &area, bool &shun) {
    p[num] = p[0];
    area = 0;
    double tmp;
    for(int i=0; i < num; i ++) {
        tmp = p[i].x*p[i+1].y - p[i].y*p[i+1].x;
        area += tmp;
    }
```

```
    area /= 2.0;
    if(shun=area<0)
        area = -area;
}
/**
    一条直线，a和b再此直线上，并且a指向b
    angle指向了向量(a, b) 左面 的方向
    因此反映了半平面！！！
*/
struct Line {
    Point a, b;
    double angle;
    Line(){}
    Line(Point a, Point b) : a(a), b(b) {
        angle = atan2(b.x-a.x, a.y-b.y);
    }
    bool operator < (const Line & l) const {
        return (angle-l.angle)!=0 ? angle<l.angle : cross(a, b, l.b)<0;
    }
/**
    叉乘
*/
    double operator * (const Line & l) const {
        return (b.x-a.x) * (l.b.y-l.a.y) - (l.b.x-l.a.x) * (b.y-a.y);
    }
};
/**
    zzy的副主函数，用于判断m点是否应该被弹出
    如果m和n平行，返回true
    如果m和n的交点严格再l之外，返回true
    如果m和n的交点严格再l之内，返回false
    如果m和n的交点再l上，则判断n和l的交是否完全被m包含，若是，返回true。此步具体做法：
    判断m和n在l的同侧，并且n和l再m的异侧
（由于本算法是ax+by+c<=0的，如果计算ax+by+c<0，不要轻易改变out返回值，建议就这样计算，然后看面积是否为0）
*/
bool out(Line m, Line n, Line l) {
    Point p;
    if(lineCross(m.a, m.b, n.a, n.b, p) != 1) {
        return true;
    }
    double d = cross(l.a, l.b, p);
    if(sig(d) != 0) return d < 0;
    double a = m*l, b = n*l, c = m * n;
    return sig(a)*sig(b)>0 && sig(a)*sig(c)<0;}
/**
    用zzy大牛的方法计算半年平面交
    将传入的半平面原地筛选，选出有用的半平面
    参数：
        l:  传入的半平面，程序中将作为队列，程序结束保存有用的半平面
        n:  传入的半平面个数
        b:  计算后l的队列底
        t:  计算后l的队列顶
        [b, t]  闭区间!
*/
void zzy(Line* l, int n, int &b, int &t) {
    sort(l, l+n);
    int i, j;
    for(i=1, j=0; i < n; i ++)
        if(sig(l[i].angle-l[j].angle) != 0)
```

```
            l[++ j] = l[i];
    for(b=0, t=1, i=2; i <= j; i ++) {
        while(b<t && out(l[t], l[t-1], l[i]))  t--;
        while(b<t && out(l[b], l[b+1], l[i]))  b++;
        l[++ t] = l[i];
    }
    while(n != t-b) {
        n = t-b;
        while(b<t && out(l[t], l[t-1], l[b]))  t--;
        while(b<t && out(l[b], l[b+1], l[t]))  b++;
    }
}

/**
    求多边形的核----zzy计算半平面交的进一步扩展
    同zzy函数一样，也是原地筛选出核
    参数:
        ps: 传入的多边形，程序结束后会成为核的多边形
        n:  传入的多边形的边数，程序结束后会成为核的多边形的边数
    返回:
        返回是否有核
*/
bool calCore(Point *ps, int &n) {
    static Line l[maxn];
    ps[n] = ps[0];
    bool shun;
    double area;
    int b, t, i;
    calPolygon(ps, n, area, shun);
    if(shun)
        for(i = 0; i < n; i ++)
            l[i] = Line(ps[i+1], ps[i]);
    else
        for(i = 0; i < n; i ++)
            l[i] = Line(ps[i], ps[i+1]);
    zzy(l, n, b, t);
    l[t+1] = l[b];
    n = t-b+1;
    if(n < 3)   return false;
    for(i = b; i <= t; i ++) {
        lineCross(l[i].a, l[i].b, l[i+1].a, l[i+1].b, ps[i-b]);
    }
    return true;
}
/**
    下面开始线段求交。。。。
    未考虑的退化情况:
        1.线段退化为点未考虑
        2.线段有部分重合未考虑
*/

Point P;
int mode = 0;

struct Seg {
    Point a, b;
    Seg(const Point &a, const Point &b) : a(a), b(b) {}
    void format() {
        if(b < a) {
            Point t = b;    b = a;  a = t;
        }
```

```
    }
    double getY() const {
        if(sig(a.x - b.x) == 0)return P.y;
        return (a.y*(b.x-P.x) - b.y*(a.x-P.x)) / (b.x - a.x);
    }
    bool operator < (const Seg & s) const {
        double y1 = getY(), y2 = s.getY();
        if(mode == 1 || sig(y1-y2) != 0)   return sig(y1 - y2) < 0;
        else                               return sig(cross(P, b, s.b)) > 0;
    }
};
struct Event {
    Point p;
    Event(const Point &p) : p(p) {}
    mutable vector<Seg> L;
    bool operator < (const Event& e) const {
        return p < e.p;
    }
};

typedef set<Seg>::iterator SI;
typedef vector<Seg> VS;
typedef pair<Point, VS> AP;
set<Event> E;   //Events
set<Seg> S;     //Segments in State
vector<AP> ans; //保存交点，和交于此点的直线，和为节省空间可以不要
VS segs;

void findNewEvent(SI pre) {
    if(pre == S.end())  return;
    SI nxt = pre --;
    if(pre == S.end())  return;
    Point p;
    if(segCross(pre->a, pre->b, nxt->a, nxt->b, p)==1 && P < p)
        E.insert(Event(p));
}
bool handleEventPoint(const Event& e) {
    P = e.p;
    Seg tmp(P, P);
    VS &L = e.L, C, R;
    mode = 1;
    for(SI iter = S.find(tmp); iter != S.end() && !(tmp < *iter); ) {
        Seg seg = *iter;
        S.erase(iter ++);
        if(seg.b<P || P<seg.b)  C.push_back(seg);
        else                    R.push_back(seg);
    }
    mode = 0;
    S.insert(L.begin(), L.end());
    S.insert(C.begin(), C.end());
    mode = 1;
    findNewEvent(S.lower_bound(tmp));
    findNewEvent(S.upper_bound(tmp));
    mode = 0;
    if(L.size() + C.size() + R.size() > 1) {
        ans.push_back(AP(P, VS()));
        vector<Seg> &v = ans.back().second;
        v.insert(v.end(), L.begin(), L.end());
        v.insert(v.end(), C.begin(), C.end());
        v.insert(v.end(), R.begin(), R.end());
        return true;
    }
    return false;
```

```
}
int find_intersections(VS segs) {
    E.clear();
    S.clear();
    for(int i = 0; i < segs.size(); i ++) {
        segs[i].format();
        E.insert(Event(segs[i].a)).first->L.push_back(segs[i]);
        E.insert(Event(segs[i].b));
    }
    int count = 0;
    while(!E.empty()) {
        Event p = *E.begin();
        E.erase(E.begin());
        count += handleEventPoint(p);
    }
    return count;
}
```

判断任意两条线段是否相交

```
const double eps = 1E-6;
int sig(double d) {
    return (d>eps) - (d<-eps);
}
struct Point {
    double x, y;
    bool operator < (const Point & p) const {
        return sig(x-p.x)!=0 ? x<p.x : sig(y-p.y)<0;
    }
};
double cross(Point o, Point a, Point b) {
    return (a.x - o.x)*(b.y - o.y)-(b.x - o.x)*(a.y - o.y);
}
double dot(Point &o, Point &a, Point &b) {
    return (a.x-o.x)*(b.x-o.x) + (a.y-o.y)*(b.y-o.y);
}
int btw(Point &x, Point &a, Point &b) {
    return sig(dot(x, a, b));
}
/*  判断线段a,b  和  c,d是否相交

    类型         返回 p
---------------------------------------
1.  不相交           0       不变
2.  规范相交     1       交点
3.  非规范相交       2       不变
*/
int segCross(Point a, Point b, Point c, Point d, Point &p) {
    double s1, s2;
    int d1, d2, d3, d4;
    d1 = sig(s1=cross(a,b,c));
    d2 = sig(s2=cross(a,b,d));
    d3 = sig(cross(c,d,a));
    d4 = sig(cross(c,d,b));
    if((d1^d2)==-2 && (d3^d4)==-2) {
        p.x = (c.x*s2-d.x*s1)/(s2-s1);
        p.y = (c.y*s2-d.y*s1)/(s2-s1);
        return 1;
    }
    if( d1==0 && btw(c,a,b)<=0 ||
        d2==0 && btw(d,a,b)<=0 ||
        d3==0 && btw(a,c,d)<=0 ||
        d4==0 && btw(b,c,d)<=0)
```

```
        return 2;
    return 0;
}

//下面开始判断任意两条线段是否正交（不包括端点相交，以及点与边相交）
//不能处理线段退化为点的情况
//不能处理线段重合的情况

Point P;
int mode = 0;

struct Seg {
    Point a, b;
    Seg(const Point &a, const Point &b) : a(a), b(b) {}
    double getY() const {
        if(sig(a.x - b.x) == 0) return P.y;
        return (a.y*(b.x-P.x) - b.y*(a.x-P.x)) / (b.x - a.x);
    }
    bool operator < (const Seg & s) const {
        double y1 = getY(), y2 = s.getY();
        if(mode == 1 || sig(y1-y2) != 0)   return sig(y1 - y2) < 0;
        else                               return sig(cross(P, b, s.b)) > 0;
    }
    void output() {
        printf("(%.2f, %.2f)\t(%.2f, %.2f)\n", a.x, a.y, b.x, b.y);
    }
};
struct Event {
    Point p;
    Event(const Point &p) : p(p) {}
    mutable vector<Seg> L;
    bool operator < (const Event& e) const {
        return p < e.p;
    }
};
typedef set<Seg>::iterator SI;
typedef vector<Seg> VS;

set<Seg> S;     //Segments in State

bool findNewEvent(SI pre) {
    if(pre == S.end())  return false;
    SI nxt = pre --;
    if(pre == S.end())  return false;
    Point p;
    int val = segCross(pre->a, pre->b, nxt->a, nxt->b, p);
    return val==1;
}
bool handleEventPoint(const Event& e) {
    P = e.p;
    Seg tmp(P, P);
    VS &L = e.L, C;
    mode = 1;
    for(SI iter = S.find(tmp); iter != S.end() && !(tmp < *iter); ) {
        Seg seg = *iter;
        S.erase(iter ++);
        if(seg.b<P || P<seg.b)  C.push_back(seg);
    }
    mode = 0;
    S.insert(L.begin(), L.end());
    S.insert(C.begin(), C.end());
    mode = 1;
    if(findNewEvent(S.lower_bound(tmp)))   return true;
```

```
    if(findNewEvent(S.upper_bound(tmp)))   return true;
    return false;
}
bool find_intersections(VS segs) {
    static set<Event> E;    //Events
    E.clear();
    S.clear();
    for(int i = 0; i < segs.size(); i ++) {
        if(segs[i].b < segs[i].a)  swap(segs[i].a, segs[i].b);
        E.insert(Event(segs[i].a)).first->L.push_back(segs[i]);
        E.insert(Event(segs[i].b));
    }
    for(set<Event>::iterator iter = E.begin(); iter != E.end(); iter ++) {
        if(handleEventPoint(*iter))return true;
    }
    return false;
}
```

多边形＋点集

```
#define maxn 10010
#define M_PI       3.14159265358979323846 /* pi */
double my_sqrt(double d) {return sqrt(max(d, 0.0));}
double my_acos(double d) {return acos(d<-1?-1:d>1?1:d);}
#define sqr(v) ((v)*(v))
const double eps = 1E-6;
int sig(double d) {
    return (d>eps) - (d<-eps);
}
struct Point {
    double x, y;
    Point(double x, double y) : x(x), y(y) {}
    Point() {}
    Point operator - (const Point & p) const {
        return Point(x-p.x, y-p.y);
    }
    Point operator + (const Point & p) const {
        return Point(x+p.x, y+p.y);
    }
    Point operator * (double d) const {
        return Point(x*d, y*d);
    }
    bool operator == (const Point & p) const {
        return sig(x-p.x)==0 && sig(y-p.y)==0;
    }
    bool operator < (const Point & p) const {
        return sig(x-p.x)!=0 ? x<p.x : sig(y-p.y)<0;
    }
    Point resize(double d) {
        d /= my_sqrt(sqr(x)+sqr(y));
        return Point(x*d, y*d);
    }
    Point left90() {
        return Point(-y, x);
    }
    Point rotate(double radian) {  //逆时针转
        double c = cos(radian), s = sin(radian);
        return Point(x*c-y*s, y*c+x*s);
    }
    double mod() {
        return my_sqrt(sqr(x)+sqr(y));
    }
    void output() {
        printf("x = %.2f, y = %.2f\n", x, y);
```

```
    }
};
double cross(const Point & o, const Point & a, const Point & b) {
    return (a.x-o.x)*(b.y-o.y) - (b.x-o.x)*(a.y-o.y);
}
double dot(Point o, Point a, Point b) {
    return (a.x-o.x)*(b.x-o.x) + (a.y-o.y)*(b.y-o.y);
}
double dis(Point a, Point b) {
    return my_sqrt(sqr(a.x-b.x) + sqr(a.y-b.y));
}
int lineCross(Point a, Point b, Point c, Point d, Point &p) {
    double s1, s2;
    s1=cross(a,b,c);
    s2=cross(a,b,d);
    if(sig(s1)==0 && sig(s2)==0)   return 2;
    if(sig(s2-s1)==0)   return 0;
    p.x = (c.x*s2-d.x*s1)/(s2-s1);
    p.y = (c.y*s2-d.y*s1)/(s2-s1);
    return 1;
}
double area(Point * p, int n) {
    double res = 0;
    p[n] = p[0];
    for(int i = 0; i < n; i ++) {
        res += p[i].x*p[i+1].y - p[i+1].x*p[i].y;
    }
    return res / 2;
}
//在ax+by+c=0 直线上找两个点p1、p2，并且使得向量(p1,p2)的左面为ax+by+c<0 一面
//返回ax+by+c是否为一直线，即a^2+b^2 != 0
bool axis(double a, double b, double c, Point & p1, Point & p2) {
    double d = a*a+b*b;
    if(sig(d) > 0) {
        p1 = Point(-a*c/d, -b*c/d);
        p2 = p1 + Point(-b, a);
        return true;
    }
    return false;
}
//下面是新的基本操作
//返回点o到直线ab的距离，res保存o在ab上的投影
double pointToLine(Point o, Point a, Point b, Point & res) {
    double d = dis(a, b);
    double s = cross(a, b, o) / d;
    res = o + (a-b).left90()*(s/d);
    return fabs(s);
}
//oa围绕o点，逆时针转到ob，所转的角度。返回值在(-pi, pi]之间
double angle(Point o, Point a, Point b) {
    double cr = cross(o, a, b);
    double dt = dot(o, a, b);
    if(sig(cr)==0)  cr = 0;
    if(sig(dt)==0)  dt = 0;
    return atan2(cr, dt);
}
/**
 * a1a2 逆时针绕点转到b1b2，绕的点为p，转的角度为ang (-pi, pi]
 * 返回：可以转true，不可以转false 当且仅当两向量平行且同向时无解!
 * assume |a1a2| = |b1b2|
 */
bool angle(Point a1, Point a2, Point b1, Point b2, Point & p, double & ang) {
```

```
    Point vecA = a2-a1, vecB = b2-b1;
    if(sig(vecA.mod()-vecB.mod()) != 0)    return false;

    double cr = vecA.x*vecB.y-vecA.y*vecB.x;
    double dt = vecA.x*vecB.x+vecA.y*vecB.y;

    if(sig(cr)==0)  cr = 0;
    if(sig(dt)==0)  dt = 0;

    ang = atan2(cr, dt);

    if(sig(cr)==0) {
        if(sig(dt)>=0)  return false;
        p = (a1+b1) * 0.5;
        return true;
    }
    Point m1 = (a1+b1)*0.5, m2 = (a2+b2)*0.5;
    if(1!=lineCross(m1, m1+(a1-b1).left90(), m2, m2+(a2-b2).left90(), p))//中垂线
交点
        lineCross(a1, a2, b1, b2, p);  //中垂线平行，现在assume 八字型
    return true;
}
//判断p是否在线段ab上，在端点处也返回true
bool onSeg(Point p, Point a, Point b) {
    return sig(cross(p, a, b))==0 && sig(dot(p, a, b))<=0;
}
//---------上面是基本函数-----------
//---------下面是新函数-----------
//判断一个多边形是否是凸多边形，初始为瞬时针、逆时针均可!
bool isConvex(Point * ps, int n) {
    bool s[3] = {1, 1, 1};
    ps[n] = ps[0];
    ps[n+1] = ps[1];
    for(int i = 0; i < n && (s[0] | s[2]); i ++) {
        s[ 1+sig(cross(ps[i+1], ps[i+2], ps[i])) ] = 0;
    }
    return (s[0] | s[2]);          //允许三点共线
    return (s[1] && s[0] | s[2]);  //不允许三点共线，要用这个for的条件最好也改...
}
//0 不在里面  2 在里面 1 边上
int inside_convex(Point * ps, int n, Point q) {
    bool s[3] = {1, 1, 1};
    ps[n] = ps[0];
    for(int i = 0; i < n && (s[0] | s[2]); i ++) {
        s[ 1+sig(cross(ps[i+1], q, ps[i])) ] = 0;
    }
    if(s[0] | s[2]) return s[1]+1;  //里面或者边上
    return 0;
}
/**
 * 判断点是否在凸多边形内O(n)初始化，log(n)效率，assume多边形为convex并且没有三点共线
 * 教程: http://hi.baidu.com/aekdycoin/blog/item/7abf85026f0d7e85d43f7cfe.html
 * 题目: http://acm.sgu.ru/problem.php?contest=0&problem=253
 */
Point ps[maxn];
double ang[maxn];
int n;
void init() {
    if(area(ps, n) < 0) reverse(ps, ps+n);
    rotate(ps, min_element(ps, ps+n), ps+n);
    for(int i = 1; i < n; i ++) {
```

```
        ang[i] = atan2(ps[i].y-ps[0].y, ps[i].x-ps[0].x);
    }
    ang[0] = -M_PI/2;
}
bool dcmp(double a, double b) {
    return sig(a-b) < 0;
}
//0 外 2 内 1 边上
int judge(Point p) {
    if(onSeg(p, ps[0], ps[1]) || onSeg(p, ps[0], ps[n-1])) return 1;
    int idx = lower_bound(ang, ang+n, atan2(p.y-ps[0].y, p.x-ps[0].x), dcmp) - ang;
    if(idx <= 1 || idx == n)    return 0;   //外面!
    return 1 + sig(cross(ps[idx-1], ps[idx], p));
}
/*sgu 253，快速判断点是否在凸多边形内
    int main() {
        int m, k;
        while(scanf("%d%d%d", &n, &m, &k) != EOF) {
            for(int i = 0; i < n; i ++)scanf("%lf%lf", &ps[i].x, &ps[i].y);
            init();
            int sum = 0;
            Point p;
            while(m --) {
                scanf("%lf%lf", &p.x, &p.y);
                if(judge(p)) sum ++;
            }
            if(sum >= k) {
                printf("YES\n");
            } else {
                printf("NO\n");
            }
        }
        return 0;
    }
*/
/**
 * 判断点p是否在任意多边形ps中
 * 0 外，1 内，2 边上
 */
int inPolygon(Point * ps, int n, Point p) {
    int cnt = 0;
    ps[n] = ps[0];
    for(int i = 0; i < n; i ++) {
        if(onSeg(p, ps[i], ps[i+1]))   return 2;
        int k = sig( cross(ps[i], ps[i+1], p) );
        int d1 = sig( ps[i+0].y-p.y );
        int d2 = sig( ps[i+1].y-p.y );
        if(k>0 && d1<=0 && d2>0)    cnt ++;
        if(k<0 && d2<=0 && d1>0)    cnt --;
    }
    return cnt != 0;
}
//多边形切割
//用直线ab切割多边形p，切割后的在向量(a,b)的左侧，并原地保存切割结果
//如果退化为一个点，也会返回去，此时n为1
void polygon_cut(Point * p, int & n, Point a, Point b) {
    static Point pp[maxn];
    int m = 0;
    p[n] = p[0];
    for(int i = 0; i < n; i ++) {
        if(sig(cross(a, b, p[i])) > 0)
            pp[m ++] = p[i];
```

```
        if(sig(cross(a, b, p[i])) != sig(cross(a, b, p[i+1])))
            lineCross(a, b, p[i], p[i+1], pp[m ++]);
    }
    n = 0;
    for(int i = 0; i < m; i ++)
        if(!i || !(pp[i]==pp[i-1]))
            p[n ++] = pp[i];
    while(n>1 && p[n-1]==p[0]) n --;
}
//多边形求核，类似于zzy的calCore
const double BOUND = 10000000.0;
bool calCore(Point * p, int & n) {
    static Point pp[maxn];
    if(sig(area(p, n)) >= 0)    copy(p, p+n, pp);
    else                reverse_copy(p, p+n, pp);
    pp[n] = pp[0];
    int nn = n;         //pp,nn init over!

    p[0] = Point(-BOUND, -BOUND);
    p[1] = Point(BOUND, -BOUND);
    p[2] = Point(BOUND, BOUND);
    p[3] = Point(-BOUND, BOUND);
    n = 4;              //p, n init over!

    for(int i = 0; i < nn && n; i ++)
        polygon_cut(p, n, pp[i], pp[i+1]);
    return n != 0;
}
/**求最近点对，分两个版本：
 * 1.    【版本 1】：将靠近中心线的点按照y进行排序，每次递归都进行sort操作。。【类似吉大】
 *       算法效率n^n logn，但是实际效果可能更好一些！
 * 2.    【版本 2】：将所有的点按照y进行归并排序，每次递归进行inplace_merge操作。【类似上交】
 *       算法效率nlogn，但是纯的nlogn，实际效果不一定最好。。。
 * 3.    【求集合间的最近点对】
 *类似poj3714，如果有两个点集，求点集间的最短距离，只需给Point加id域，且在close0 的倒数第二行：
 *  res = min(res, dis(psy[i], psy[j]));
 *的前面加一个if语句就可以了，比如
 *  if(psy[i].id^psy[j].id)
 *      res = min(res, dis(psy[i], psy[j]));
 *并且用【版本 2】的模板速度会快一点!!!
 */
bool cmp_x(const Point & a, const Point & b) {return a.x<b.x;}
bool cmp_y(const Point & a, const Point & b) {return a.y<b.y;}
//【版本 1】：将靠近中心线的点按照y进行排序，每次递归都进行sort操作。。【类似吉大】
//算法效率n^n logn，但是实际效果可能更好一些！
double close0(Point * ps, int l, int r) {
    static Point psy[maxn];
    if(r-l <= 1)    return 1E100;
    int m = (l + r) / 2, i;
    double res = 1E100;
    for(i = m; i >= l && sig(ps[i].x-ps[m].x)==0; i --);
          res = min(res, close0(ps, l, i+1));
    for(i = m; i < r  && sig(ps[i].x-ps[m].x)==0; i ++);
          res = min(res, close0(ps, i, r));
    int len = 0;
    for(i = m;   i>= l && sig(ps[m].x-res-ps[i].x)<0; i --)   psy[len ++] = ps[i];
    for(i = m+1; i < r && sig(ps[m].x+res-ps[i].x)>0; i ++)   psy[len ++] = ps[i];
    sort(psy, psy+len, cmp_y);
    for(int i = 0; i < len; i ++)
        for(int j = i+1; j<len && psy[j].y<psy[i].y+res; j ++)
            res = min(res, dis(psy[i], psy[j]));
```

```
    return res;
}
double close(Point * ps, int n) {
    sort(ps, ps+n, cmp_x);
    return close0(ps, 0, n);
}
//【版本 2】：将所有的点按照y进行归并排序，每次递归进行inplace_merge操作。【类似上交】
//算法效率nlogn，但是纯的nlogn，实际效果不一定最好。。。
double close0_(Point * ps, int l, int r) {
    static Point psy[maxn];
    if(r-l <= 1)    return 1E300;
    int m = (l + r) / 2;
    double midX = ps[m].x;
    double res = min( close0_(ps, l, m), close0_(ps, m, r) );
    inplace_merge(ps+l, ps+m, ps+r, cmp_y);
    double x1 = midX-res, x2 = midX+res;
    int len = 0;
    for(int i = l; i < r; i ++)
        if(sig(x1-ps[i].x)<0 && sig(ps[i].x-x2)<0)
            psy[len ++] = ps[i];
    for(int i = 0; i < len; i ++)
        for(int j = i+1; j<len && psy[j].y<psy[i].y + res; j ++)
            res = min(res, dis(psy[i], psy[j]));
    return res;
}
double close_(Point * ps, int n) {
    sort(ps, ps+n, cmp_x);
    return close0_(ps, 0, n);
}
/**
 * 下面开始【【【【【旋转卡壳】】】】】!!!
 * 旋转卡壳全集教程：http://blog.csdn.net/ACMaker/archive/2008/10/29/3176910.aspx
 */
/**
 * 求凸多边形直径！注意传入凸多边形！
 */
double diam(Point *p, int n) {
    if(area(p, n)<0)    reverse(p, p+n);
    p[n] = p[0];
    double res = 0;
    for(int i = 0, j = 1; i < n; i ++) {
        while(sig(cross(p[i], p[i+1], p[j])-cross(p[i], p[i+1], p[(j+1)%n])) < 0)
            j = (j+1)%n;
        res = max(res, dis(p[i], p[j]));
        res = max(res, dis(p[i+1], p[j]));
    }
    return res;
}
//计算两个凸多边形间最近，最远距离，assume两个凸多边形分离
//返回o到线段ab的最短距离
double minDis(Point o, Point a, Point b) {
    if(sig(dot(b, a, o))<0) return dis(o, b);
    if(sig(dot(a, b, o))<0) return dis(o, a);
    return fabs(cross(o, a, b)/dis(a, b));
}
//计算从curAng逆时针转到ab的角
double calRotate(Point a, Point b, double curAng) {
    double x = fmod(atan2(b.y-a.y, b.x-a.x)-curAng, M_PI*2);
    if(x<0)         x += M_PI*2;
    if(x>1.9*M_PI)  x = 0;      //in case x is nearly 2*M_PI
    if(x >= 1.01*M_PI)  while(1);
    return x;
```

```
}
//求凸多边形间最小距离，断言P和Q分离
//传入P、Q：凸多边形。n、m：P和Q的顶点个数
//如果P和Q非逆时针，则reverse!
//题目：POJ-3608
//教程：http://blog.csdn.net/ACMaker/archive/2008/10/29/3178696.aspx
double mind2p(Point *P, int n, Point *Q, int m) {
    if(area(P, n) < 0)  reverse(P, P+n);    //需要逆时针的
    if(area(Q, m) < 0)  reverse(Q, Q+m);
    int p0 = min_element(P, P+n)-P, q0 = max_element(Q, Q+m)-Q;

    double res = dis(P[p0], Q[q0]);

    double ang = -M_PI/2, rotateP, rotateQ;
    int pi, pj, qi, qj, totP, totQ, val;
    for(pi=p0, qi=q0, totP=0, totQ=0; totP<n && totQ<m; ) {
        pj = (pi+1) % n;
        qj = (qi+1) % m;

        rotateP = calRotate(P[pi], P[pj], ang);
        rotateQ = calRotate(Q[qi], Q[qj], ang+M_PI);

        val = sig(rotateP - rotateQ);
        ang += min(rotateP, rotateQ);

        if(val == -1)   res = min(res, minDis(Q[qi], P[pi], P[pj]));
        else if(val==1) res = min(res, minDis(P[pi], Q[qi], Q[qj]));
        else {
            res = min(res, minDis(P[pi], Q[qi], Q[qj]));
            res = min(res, minDis(P[pj], Q[qi], Q[qj]));
            res = min(res, minDis(Q[qi], P[pi], P[pj]));
            res = min(res, minDis(Q[qj], P[pi], P[pj]));
        }
        if(val != 1)    pi=pj, totP ++;
        if(val != -1)   qi=qj, totQ ++;
    }
    return res;
}
//求凸多边形间最大距离，断言P和Q分离
//传入P、Q：凸多边形。n、m：P和Q的顶点个数
//如果P和Q非逆时针，则reverse!
//教程：http://blog.csdn.net/ACMaker/archive/2008/10/29/3178794.aspx
double maxd2p(Point *P, int n, Point *Q, int m) {  //【【【待测】】】.......!!!
    if(area(P, n) < 0)  reverse(P, P+n);    //需要逆时针的
    if(area(Q, m) < 0)  reverse(Q, Q+m);
    int p0 = min_element(P, P+n)-P, q0 = max_element(Q, Q+m)-Q;

    double res = dis(P[p0], Q[q0]);

    double ang = -M_PI/2, rotateP, rotateQ;
    int pi, pj, qi, qj, totP, totQ, val;
    for(pi=p0, qi=q0, totP=0, totQ=0; totP<n && totQ<m; ) {
        pj = (pi+1) % n;
        qj = (qi+1) % m;

        rotateP = calRotate(P[pi], P[pj], ang);
        rotateQ = calRotate(Q[qi], Q[qj], ang+M_PI);

        val = sig(rotateP - rotateQ);
        ang += min(rotateP, rotateQ);
```

```
        if(val == -1)   res = max(res, max(dis(Q[qi], P[pi]), dis(Q[qi], P[pj])));
        else if(val==1) res = max(res, max(dis(P[pi], Q[qi]), dis(P[pi], Q[qj])));
        else {
            res = max(res, dis(P[pi], Q[qi]));
            res = max(res, dis(P[pi], Q[qj]));
            res = max(res, dis(P[pj], Q[qi]));
            res = max(res, dis(P[pj], Q[qj]));
        }
        if(val != 1)    pi=pj, totP ++;
        if(val != -1)   qi=qj, totQ ++;
    }
    return res;
}
/**
//poj3608
    Point P[maxn], Q[maxn];
    int n, m;

    int main() {
        while(scanf("%d%d", &n, &m), n||m) {
            for(int i = 0; i < n; i ++)scanf("%lf%lf", &P[i].x, &P[i].y);
            for(int i = 0; i < m; i ++)scanf("%lf%lf", &Q[i].x, &Q[i].y);
            printf("%.5f\n", mind2p(P, n, Q, m));
        }
        return 0;
    }
 */
/**
 * 计算凸多边形的最小外接矩形面积
 * 如果要计算点集的最小外接矩形，先求凸包
 * 点集的最小面积/周长外接矩形，UVA10173
 * 教程：http://blog.csdn.net/ACMaker/archive/2008/10/30/3188123.aspx
 */
/**
 * graham求凸包-水平序，结果是按照瞬时针给出的
 */
int graham(Point*p, int n, int*ch) {
#define push(x)    ch[len ++]=x
#define pop()      len --
    sort(p, p+n);
    int len = 0, len0 = 1, i;
    for(i = 0; i < n; i ++) {
        while(len > len0 && sig(cross(p[ch[len-1]], p[ch[len-2]], p[i]))<=0)
            pop();
        push(i);
    }
    len0 = len;
    for(i = n-2; i >= 0; i --) {
        while(len > len0 && sig(cross(p[ch[len-1]], p[ch[len-2]], p[i]))<=0)
            pop();
        push(i);
    }
    return len-1;
}
bool cmp2(const Point &a, const Point &b) {
    return sig(a.y-b.y)!=0 ? a.y<b.y : sig(a.x-b.x)>0;
}
//计算凸多边形的最小外接矩形面积
double minRect0(Point *p, int n) {
    if(n<=2)    return 0;
    if(area(p, n) < 0)  reverse(p, p+n);
```

```
    int ai = min_element(p, p+n)     -p;
    int bi = min_element(p, p+n, cmp2)-p;
    int ci = max_element(p, p+n)     -p;
    int di = max_element(p, p+n, cmp2)-p;
    int aj, bj, cj, dj;

    double res = 1E100, ang = -M_PI/2;
    double ra, rb, rc, rd, r, s, c, ac, bd;

    while(ang < M_PI * 0.51) {
        aj=(ai+1)%n;    ra = calRotate(p[ai], p[aj], ang);
        bj=(bi+1)%n;    rb = calRotate(p[bi], p[bj], ang+0.5*M_PI);
        cj=(ci+1)%n;    rc = calRotate(p[ci], p[cj], ang+1.0*M_PI);
        dj=(di+1)%n;    rd = calRotate(p[di], p[dj], ang+1.5*M_PI);
        r = min(min(ra,rb), min(rc,rd));
        ang += r;

        s = sin(ang), c = cos(ang);

        ac = -s*(p[ci].x-p[ai].x) + c*(p[ci].y-p[ai].y);
        bd =  c*(p[di].x-p[bi].x) + s*(p[di].y-p[bi].y);

        res = min(res, fabs(ac*bd));
            //改为(fabs(ac)+fabs(bd))*2 就是求最小周长外接矩形

        if(sig(ra-r)==0)    ai=aj;
        if(sig(rb-r)==0)    bi=bj;
        if(sig(rc-r)==0)    ci=cj;
        if(sig(rd-r)==0)    di=dj;
    }
    return res==1E100 ? 0 : res;
}
//计算点集的最小外接举行面积，底层调用minRect0
double minRect(Point *p, int n) {
    static Point q[maxn];
    static int ch[maxn];
    int len = graham(p, n, ch);
    for(int i = 0; i < len; i ++)  q[i] = p[ch[len-1-i]];
    return minRect0(q, len);
}
/*
//UVA10173
    Point P[maxn];
    int n;
    int main() {
        while(scanf("%d", &n), n) {
            for(int i = 0; i < n; i ++)scanf("%lf%lf", &P[i].x, &P[i].y);
            printf("%.4f\n", minRect(P, n));
        }
        return 0;
    }
*/
/**
 * 计算最多共线的点的个数(必须是整点，不能有重点，效率n^2),不能有重点!!
 */
typedef pair<int,int> T;
int gcd(int a, int b) {
    for(int t; t=b; b=a%b, a=t);
    return a;
}
int calNumLine(T *ts, int n) {
    static T tmp[maxn];
    int res = 0;
```

```
    int i, j, k;
    for(i = 0; i < n; i ++) {
        for(j = 0; j < i; j ++) {
            int dx = ts[j].first - ts[i].first;
            int dy = ts[j].second-ts[i].second;
            int g = gcd(dx, dy);
            dx /= g;    dy /= g;
            if(dx<0)    dx=-dx,dy=-dy;
            tmp[j] = T(dx, dy);
        }
        sort(tmp, tmp+j);
        for(j = 0; j < i; j = k) {
            for(k = j; k<i && tmp[k]==tmp[j]; k ++);
            res = max(res, k-j);
        }
    }
    return res+1;
}
int main() {


    return 0;
}
```

费马点_fix

```
/**
以前的费马点可能有问题，比如(0,0)(0,1)(1,0)这个等腰直角三角形，就会死锁
更新了搜索方式，向八个方向搜索
*/

struct Point {
    double x, y;
    Point(){}
    Point(double x, double y) {
        this->x = x;
        this->y = y;
    }
};

double dis(Point a, Point b) {
    return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
double allDis(Point *ps, int n, Point o) {
    double sum = 0;
    for(int i = 0; i < n; i ++)
        sum += dis(o, ps[i]);
    return sum;
}
double fermat(Point *ps, int n) {
    int dx[8] = {1,-1,0,0,1,1,-1,-1}, dy[8]={0,0,1,-1,1,-1,1,-1};
    Point o(0, 0);
    double res = allDis(ps, n, o);
    for(double step = 2048.0; step>=0.49; step /= 2.0) {
        for(int i = 0; i < 8; i ++) {
            Point no(o.x+step*dx[i], o.y+step*dy[i]);
            double tmp = allDis(ps, n, no);
            if(tmp < res) {
                res = tmp;
                o = no;
                i = -1; /// again!!
            }
        }
    }
```

```
    return res;
}

Point ps[110];
int n;

//poj-2420
int main() {
    Point o, no;
    int i;
    while(cin >> n) {
        for(i = 0; i < n; i ++) {
            cin >> ps[i].x >> ps[i].y;
        }
        printf("%.0f\n", fermat(ps, n));
    }
    return 0;
}

//--------------------------------以下代码是牛顿迭代法
-------------------------------------

/*
 所谓费马点 就是到n个点距离和最小的点
 该题大部分人用的都是随机退火 于是研究牛顿迭代法
 注意到 虽然距离和f(x,y)很讨厌 但是其二次偏导数恰恰是恒正的
 也就是说 对于任意一个y f'x(x,y)有唯一 0 点 于是使用牛顿法逼近这个 0 点
 但是 这个函数是否是单峰的呢?如果不是 那么牛顿法就很可能会陷入局部最小点而达不到最优解
 并且实际实现的时候发现 初始迭代点不同的确会导致这种情况发生 也就是说 要么我程序写丑了
 要么这个函数的确有多个不同的局部最小值 求各位大牛证明
 ////////////////////////////

 证明如下:
 如果有两个局部最小值 由该函数连续 该函数在这两个点的连线上必然是先下 后上 再下 再上 反正必然有
2 个起伏
 于是 假设其中一个点是原点 令这两点的方向角为t 该函数在t方向的导函数不是单调的
 但是 将x用x0+rcost,y=y0+rsint带入 其所有方向角的二次偏导都是恒正的 这样上面的情况就不存在
了
 更直观的:将整张图旋转 让那条直线转到x轴方向 由于对所有{(xi,yi)}有x二次偏导是正的 就得出方向
角二次偏导是正的 同时 由于f的连续性和累次极限存在连续得到重极限存在的定理也可以得出相同结论
 ////////////////

 研究发现 虽然该函数求偏导是单调的 但是单调递增有上界 于是当初始坐标选取不好 就选不到下凸函数段
了 于是就无法利用牛顿收敛法 程序会越算越大最后溢出
 所以 正确的牛顿法应该是对方向导求导 即 求t的三次导 然后沿着下降最快的那个方向往下走
 于是 退化为关于(r,t)的函数的模拟退火 于是退化为关于(x,y)的模拟退火 这就是为什么模拟退火可以
保证出最优解的原因 这是个漏斗形的曲面
 */

const int maxn = 110;
const double eps = 1e-6;
inline double sqr(double a) {
    return a * a;
}
inline double dis(double x1, double y1, double x2, double y2) {
    return sqrt(sqr(x1 - x2) + sqr(y1 - y2));
}

double x[maxn], y[maxn];
int n;
```

```
double f(double x0, double y0) {
    int i;
    double tot = 0;
    for (i = 0; i < n; i++)
        tot += dis(x0, y0, x[i], y[i]);
    return tot;
}
double countdfx(double x0, double y0) {
    int i;
    double tot = 0;
    for (i = 0; i < n; i++)
        tot += (x0 - x[i]) / dis(x0, y0, x[i], y[i]);
    return tot;
}
double countdfy(double x0, double y0) {
    int i;
    double tot = 0;
    for (i = 0; i < n; i++)
        tot += (y0 - y[i]) / dis(x0, y0, x[i], y[i]);
    return tot;
}
double countdfxx(double x0, double y0) {
    int i;
    double tot = 0;
    for (i = 0; i < n; i++)
        tot += sqr(y0 - y[i]) / pow(dis(x0, y0, x[i], y[i]), 3);
    return tot;
}
double countdfyy(double x0, double y0) {
    int i;
    double tot = 0;
    for (i = 0; i < n; i++)
        tot += sqr(x0 - x[i]) / pow(dis(x0, y0, x[i], y[i]), 3);
    return tot;
}
double work() {
    const double maxl = 1e5;
    double x0 = maxl, y0 = maxl;
    int i;
    for (i = x0 = y0 = 0.0; i < n; i++)
        x0 += x[i], y0 += y[i];
    x0 /= n;
    y0 /= n;
    while (1) {
        double xp = x0, yp = y0;
        for (i = 0; i < n; i++)
            if (dis(x0, y0, x[i], y[i]) < eps) {
                x0 += eps;
                y0 += eps;
            }
        double xd = countdfx(xp, yp), yd = countdfy(xp, yp);
        double xdd = countdfxx(xp, yp), ydd = countdfyy(xp, yp);
        x0 -= xd / xdd;
        y0 -= yd / ydd;
        if (f(x0, y0) > f(xp, yp)) {
            x0 = xp;
            y0 = yp;
            break;
        }
        if (dis(x0, y0, xp, yp) < eps)
            break;
    }
    return f(x0, y0);
```

```
}
int main() {
    int i;
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%lf%lf", &x[i], &y[i]);
    printf("%.0f\n", work());
    return 0;
}
```

平面嵌入

```
/**
    判定一个图是否为平面图，输入完以后直接调用judge函数即可
    注意：
        0.点数n>0
        1.点从 1 开始计算
        2.不能有重边/自环（否则自己先去重一下）
 */
typedef pair<int,int> T;
#define maxn 20000

struct node {
    int dep, fa, infc, used, vst, dfi, ec, lowp, bflag, flag, lowpoint;
};
int n, m, indee, lk[maxn * 3][2], w1[maxn], w2[maxn], que[maxn],
        child[maxn * 3][3], bedg[maxn * 3][2], p1, p2, sdlist[maxn * 6][3],
        buk[maxn * 6][2], ps, exf[maxn * 3][2], p, proots[maxn * 3][3],
        stk[maxn * 3][2], infap[maxn * 3];
node dot[maxn];
void init(T * ts) {//读入函数
    ps = 0;
    int i, k1, k2;
    for (i = 1; i <= n; i++)
        w1[i] = i;//尾巴指针
    p1 = n;
    for (i = 0; i < m; i++) {
        k1 = ts[i].first;
        k2 = ts[i].second;
        lk[++p1][0] = k2;//记录每一条边
        lk[p1][1] = 0;
        lk[w1[k1]][1] = p1;
        w1[k1] = p1;
        lk[++p1][0] = k1;
        lk[p1][1] = 0;
        lk[w1[k2]][1] = p1;
        w1[k2] = p1;
    }
    for (i = 1; i <= n; i++)
        que[i] = i;
}
int deep(int a) {//深度优先搜索,其中包括取得所有节点的lowpoint
    dot[a].used = 1;
    indee++;//用于DFI序,标记每个节点
    dot[a].dfi = indee;
    int s, t = lk[a][1], tmp;
    while (t != 0) {
        tmp = lk[t][0];
        if (!dot[tmp].used) {
            dot[tmp].fa = a;//记录父亲
            dot[tmp].dep = dot[a].dep + 1;//表示深度
            dot[tmp].ec = dot[a].dep;//表示直接或者间接连接的最早祖先
```

```
            dot[tmp].lowp = dot[a].dep;//表示直接连接的最早祖先
            child[++p1][0] = tmp;
            child[p1][1] = 0;
            child[w1[a]][1] = p1;
            w1[a] = p1;
            s = deep(tmp);
            if (s < dot[a].ec)
                dot[a].ec = s;
        } else {
            if (dot[a].fa != tmp) {
                if (dot[a].lowp > dot[tmp].dep)
                    dot[a].lowp = dot[tmp].dep;
                if (dot[a].dfi > dot[tmp].dfi) {
                    bedg[++p2][0] = a;
                    bedg[p2][1] = 0;
                    bedg[w2[tmp]][1] = p2;
                    w2[tmp] = p2;
                }
            }
        }
        t = lk[t][1];
    }
    if (dot[a].ec > dot[a].lowp)
        dot[a].ec = dot[a].lowp;
    return dot[a].ec;
}
void sortvtx() {//将节点按照其lowpoint排序
    int i, tmp;
    for (i = 1; i <= n; i++)
        w1[i] = i;
    p1 = n;
    p2 = 0;
    for (i = 1; i <= n; i++) {
        buk[++p1][0] = i;
        buk[p1][1] = 0;
        buk[w1[dot[i].dfi]][1] = p1;
        w1[dot[i].dfi] = p1;
    }
    for (i = n; i >= 1; i--) {
        tmp = buk[i][1];
        while (tmp != 0) {
            que[++p2] = buk[tmp][0];
            tmp = buk[tmp][1];
        }
    }
}
void getsdlist() {//取得所有节点的SDlist
    int tmp, i, fa;
    memset(buk, 0, sizeof(buk));
    for (i = 1; i <= n; i++) {
        w1[i] = i;
        w2[i] = i;
        buk[i][1] = 0;
    }
    p1 = n;
    p2 = n;
    for (i = 1; i <= n; i++) {//将所有的节点按照lowpoint排序
        buk[++p1][0] = i;
        buk[p1][1] = 0;
        buk[w1[dot[i].ec]][1] = p1;
        w1[dot[i].ec] = p1;
    }
```

```
    for (i = 1; i <= n; i++) {
        tmp = buk[i][1];
        while (tmp != 0) {//按顺序插入其父亲的SDlist
            fa = dot[buk[tmp][0]].fa;
            sdlist[++p2][0] = i;
            sdlist[p2][1] = 0;
            sdlist[w2[fa]][1] = p2;
            dot[buk[tmp][0]].infc = p2;
            sdlist[p2][2] = w2[fa];
            w2[fa] = p2;
            tmp = buk[tmp][1];
        }
    }
}
void getnextvtx(int v, int v1, int &m, int &m1) {//到达下一节点
    m = exf[v][v1 ^ 1];
    if (exf[m][0] == v)
        m1 = 0;
    else
        m1 = 1;
}
void addwei(int a) {//将儿子加入其父亲的SDlist首部
    int fa;
    fa = dot[a - n].fa;
    p1++;
    proots[p1][0] = a;
    proots[p1][1] = 0;
    proots[w1[fa]][1] = p1;
    proots[p1][2] = w1[fa];
    w1[fa] = p1;
    infap[a] = p1;
}
void addsou(int a) {//将儿子加入其父亲的SDlist尾部
    int fa;
    fa = dot[a - n].fa;
    p1++;
    proots[p1][0] = a;
    proots[p1][1] = 0;
    proots[p1][1] = proots[fa][1];
    proots[p1][2] = fa;
    proots[fa][1] = p1;
    proots[proots[p1][1]][2] = p1;
    infap[a] = p1;
    if (w1[fa] == fa)
        w1[fa] = p1;
}
void walkup(int v, int w) {//向上遍历取得相关信息
    dot[w].bflag = v;
    int c, x, x1, y, y1, z, z1;
    x = w, x1 = 1;
    y = w, y1 = 0;
    while (x != v) {
        if (dot[x].vst == v || dot[y].vst == v)//如果遍历到遍历过的节点,终止遍历
            break;
        dot[x].vst = v;
        dot[y].vst = v;
        z1 = 0;
        if (x > n)
            z1 = x;
        if (y > n)
            z1 = y;
        if (z1 != 0) {//如果到达块的根节点
```

```
            c = z1 - n;
            z = dot[c].fa;
            if (z != v) {
                if (dot[c].lowpoint < dot[v].dep)
                    addwei(z1);
                else
                    addsou(z1);
            }
            x = z;
            x1 = 1;
            y = z;
            y1 = 0;
        } else {
            getnextvtx(x, x1, x, x1);
            getnextvtx(y, y1, y, y1);
        }
    }
}
void getactivenext(int v, int v1, int &m, int &m1, int vt) {//取得下一活跃节点
    m = v;
    m1 = v1;
    getnextvtx(m, m1, m, m1);
    while (dot[m].bflag != vt && proots[m][1] == 0 && dot[m].ec >= dot[vt].dep
            && m != v)
        getnextvtx(m, m1, m, m1);
}
void addstack(int a, int b) {//压入栈
    ps++;
    stk[ps][0] = a;
    stk[ps][1] = b;
}
void mergestack() {//合并操作
    int t, t1, k, k1, tmp, s, s1, fa;
    t = stk[ps][0];
    t1 = stk[ps][1];
    k = stk[ps - 1][0];
    k1 = stk[ps - 1][1];
    ps -= 2;
    s = exf[t][1 ^ t1];
    if (exf[s][1] == t)
        s1 = 1;
    else
        s1 = 0;
    exf[k][k1] = s;
    exf[s][s1] = k;
    tmp = dot[t - n].infc;//将儿子从父亲的SDlist中删除
    sdlist[sdlist[tmp][2]][1] = sdlist[tmp][1];
    sdlist[sdlist[tmp][1]][2] = sdlist[tmp][2];
    tmp = dot[t - n].fa;
    if (sdlist[tmp][1] == 0)
        dot[tmp].ec = dot[tmp].lowp;
    else
        dot[tmp].ec = min(dot[tmp].lowp, sdlist[sdlist[tmp][1]][0]);
    tmp = infap[t];
    fa = dot[t - n].fa;
    proots[proots[tmp][2]][1] = proots[tmp][1];//将节点从父亲的proots中删除
    if (proots[tmp][1] != 0)
        proots[proots[tmp][1]][2] = proots[tmp][2];
    else
        w1[fa] = proots[tmp][2];
}
void embededg(int v, int v1, int w, int w1) {//加边操作
```

```
    exf[v][v1] = w;
    exf[w][w1] = v;
}
void walkdown(int v) {//向下遍历的过程
    ps = 0;
    int v2, w, w1, x1, x, y, y1, w0, w2, vt = dot[v - n].fa;
    for (v2 = 0; v2 <= 1; v2++) {//进行顺时针和逆时针两次遍历
        getnextvtx(v, 1 ^ v2, w, w1);
        while (w != v) {
            if (dot[w].bflag == vt) {//加入回落边
                while (ps != 0)
                    mergestack();
                embededg(v, v2, w, w1);
                dot[w].bflag = 0;//去掉它的相关性标记
            }
            if (proots[w][1] != 0) {//有子块需要继续遍历
                addstack(w, w1);
                w0 = proots[proots[w][1]][0];
                getactivenext(w0, 1, x, x1, vt);
                getactivenext(w0, 0, y, y1, vt);
                if (dot[x].ec >= dot[vt].dep) {//找到我们应该走向哪一个方向
                    w = x;
                    w1 = x1;
                } else if (dot[y].ec >= dot[vt].dep) {
                    w = y;
                    w1 = y1;
                } else if (dot[x].bflag == vt || proots[x][1] != 0) {
                    w = x;
                    w1 = x1;
                } else {
                    w = y;
                    w1 = y1;
                }
                if (w == x)
                    w2 = 0;
                else
                    w2 = 1;
                addstack(w0, w2);
            } else {
                if (w > n || dot[w].ec >= dot[vt].dep)
                    getnextvtx(w, w1, w, w1);
                else {
                    if (w <= n && dot[w].ec < dot[vt].dep && ps == 0)
                        embededg(v, v2, w, w1);
                    break;//遭遇终止节点
                }
            }
        }
        if (ps != 0)
            break;
    }
}
bool chainvtx(int a) {//处理节点
    int t = child[a][1], tmp;
    while (t != 0) {
        tmp = child[t][0];
        exf[tmp][1] = tmp + n;//更新外部面
        exf[tmp][0] = tmp + n;
        exf[tmp + n][1] = tmp;
        exf[tmp + n][0] = tmp;
        t = child[t][1];
    }
```

```
    t = bedg[a][1];
    while (t != 0) {//向上遍历
        walkup(a, bedg[t][0]);
        t = bedg[t][1];
    }
    t = child[a][1];
    while (t != 0) {//向下遍历
        walkdown(child[t][0] + n);
        t = child[t][1];
    }
    t = bedg[a][1];
    while (t != 0) {
        if (dot[bedg[t][0]].bflag != 0)
            return false;
        t = bedg[t][1];
    }
    return true;
}

bool judge(int N, int M, T * ts) {
    n = N;  m = M;
    if(n == 1)      return true;
    if(m > 3*n-5)   return false;//如果边数大于 3*n-5 条,输出非平面图
    init(ts);

    int i;
    for (i = 1; i <= n; i++) {//初始化
        proots[i][1] = 0;
        p = 0;
        child[i][1] = 0;
        proots[i + n][1] = 0;
        buk[i][1] = 0;
        buk[i + n][1] = 0;
        sdlist[i][1] = 0;
        sdlist[i + n][1] = 0;
        dot[i].bflag = 0;
        dot[i + n].flag = 0;
    }
    for (i = 1; i <= n; i++) {
        w1[i] = i;
        w2[i] = i;
        child[i][1] = 0;
        bedg[i][1] = 0;
        dot[i].used = 0;
    }
    indee = 0;
    p1 = n;
    p2 = n;
    for (i = 1; i <= n; i++) {//深度优先搜索
        if (!dot[i].used) {
            dot[i].dep = 1;
            deep(i);
        }
    }
    sortvtx();//将节点排序
    getsdlist();//取得所有节点的SDlist
    for (i = 1; i <= n; i++) {
        dot[i].lowpoint = dot[i].ec;
        dot[i].vst = 0;
        dot[i + n].vst = 0;
        proots[i][1] = 0;
        w1[i] = i;
```

```
    }
    p1 = n;
    for (i = 1; i <= n; i++) {//按照逆向的深度优先搜索序处理每一个节点
        if (!chainvtx(que[i])) {
            return false;
        }
    }
    return true;
}

T ts[maxn];

int main() {
    int N, M;
    while(scanf("%d%d", &N, &M) != EOF) {
        for(int i = 0; i < M; i ++) {
            scanf("%d%d", &ts[i].first, &ts[i].second);
        }
        if(judge(N, M, ts)) {
            printf("YES\n");
        } else {
            printf("NO\n");
        }
    }
    return 0;
}
```

三角形

```
#define sqr(v) ((v)*(v))
const double eps = 1E-6;
int sig(double d) {
    return (d>eps) - (d<-eps);
}
struct Point {
    double x, y;
    Point(double x, double y) : x(x), y(y) {}
    Point() {}
    Point operator + (const Point & p) const {return Point(x+p.x, y+p.y); }
    Point operator - (const Point & p) const {return Point(x-p.x, y-p.y); }
    Point operator * (const double& d) const {return Point(x*d, y*d);     }
    Point left90() {   return Point(-y, x);                              }
    void output() { printf("x = %.2f, y = %.2f\n", x, y);}
};
struct Circle {
    Point c;
    double r;
    Circle(Point c, double r) : c(c), r(r) {}
    Circle() {}
};
double cross(const Point & o, const Point & a, const Point & b) {
    return (a.x-o.x)*(b.y-o.y) - (b.x-o.x)*(a.y-o.y);
}
double dot(const Point & o, const Point & a, const Point & b) {
    return (a.x-o.x)*(b.x-o.x) + (a.y-o.y)*(b.y-o.y);
}
double dis(Point a, Point b) {
    return sqrt(sqr(a.x-b.x) + sqr(a.y-b.y));
}
int lineCross(Point a, Point b, Point c, Point d, Point &p) {
    double s1, s2;
    s1=cross(a,b,c);
    s2=cross(a,b,d);
    if(sig(s1)==0 && sig(s2)==0)   return 2;
```

```
    if(sig(s2-s1)==0)   return 0;
    p.x = (c.x*s2-d.x*s1)/(s2-s1);
    p.y = (c.y*s2-d.y*s1)/(s2-s1);
    return 1;
}
/*=====================================================*\
```

上面是公用函数，下面开始------三角形------

【一些性质】:

0.前提：

设三角形的三条边为a, b, c

A,B,C为a,b,c所对的角

令x^ = x*x

1.海伦公式三角形的面积可以根据海伦公式算得，如下：

```
s = sqrt(p * (p - a) * (p - b) * (p - c));
p = (a + b + c) / 2;
```

2. 外接圆和内切圆半径：

令 tmp = sqrt((a+b+c)(b+c-a)(a+c-b)(a+b-c))

外接圆半径：R = abc / tmp

内切圆半径：r = tmp / (2(a+b+c))

3.正弦定理：

a/sin(A) = b/sin(B) = c/sin(C) = 2R

（其中R为外接圆半径）

4.余弦定理：

```
a^ = b^ + c^ - 2bc*cos(A)
b^ = a^ + c^ - 2ac*cos(B)
c^ = a^ + b^ - 2ab*cos(C)
```

5.五心+费马点 的性质：

1、垂心

三角形三边上的高的交点称为三角形的垂心。三角形垂心有下列有趣的性质：设△ABC 的三条高为AD、BE、CF，其中D、E、F为垂足,垂心为H。

性质 1 垂心H关于三边的对称点，均在△ABC的外接圆上。

性质 2 △ABC中,有六组四点共圆，有三组（每组四个）相似的直角三角形，且AHÂ·HD=BHÂ·HE=CHÂ·HF。

性质 3 H、A、B、C四点中任一点是其余三点为顶点的三角形的垂心(并称这样的四点为一垂心组)。

性质 4 △ABC,△ABH,△BCH,△ACH的外接圆是等圆。

性质 5 在非直角三角形中,过H的直线交AB、AC所在直线分别于P、Q，则 AB/APÂ·tanB+AC/AQÂ·tanC=tanA+tanB+tanC。

性质 6 三角形任一顶点到垂心的距离，等于外心到对边的距离的 2 倍。

性质 7 设O，H分别为△ABC的外心和垂心，则∠BAO=∠HAC，∠ABH=∠OBC，∠BCO=∠HCA。

性质 8 锐角三角形的垂心到三顶点的距离之和等于其内切圆与外接圆半径之和的 2 倍。

性质 9 锐角三角形的垂心是垂足三角形的内心；锐角三角形的内接三角形(顶点在原三角形的边上)中，以垂足三角形的周长最短。

2、内心

三角形的内切圆的圆心简称为三角形的内心，即三角形三个角平分线的交点。内心有下列优美的性质：

性质 1 设I为△ABC的内心,则I为其内心的充要条件是：到△ABC三边的距离相等。

性质 2 设I为△ABC的内心，则∠BIC=90Â°+12∠A，类似地还有两式；反之亦然。

性质 3 设I为△ABC内一点，AI所在直线交△ABC的外接圆于D。I为△ABC内心的充要条件是ID=DB=DC。

性质 4 设I为△ABC的内心，BC=a，AC=b，AB=c，I在BC、AC、AB上的射影分别为D、E、F；内切圆半径为r，令 p= (1/2)(a+b+c)，则 (1)S △ ABC=pr ； (2)r=2S △ ABC/a+b+c ；(3)AE=AF=p-a,BD=BF=p-b,CE=CD=p-c；(4)abcr=pÂ·AIÂ·BIÂ·CI。

性质 5 三角形一内角平分线与其外接圆的交点到另两顶点的距离与到内心的距离相等；反之，若I为△ABC的∠A平分线AD(D在△ABC的外接圆上)上的点，且 DI=DB，则I为△ABC的内心。

性质 6 设I为△ABC的内心，BC=a，AC=b，AB=c，∠A的平分线交BC于K,交△ABC的外接圆于D，则 AI/KI =AD/DI =DI/DK = (b+c)/a。

3、外心

三角形的外接圆的圆心简称三角形的外心.即三角形三边中垂线的交点。外心有如下一系列优美性质：

性质 1 三角形的外心到三顶点的距离相等，反之亦然。

性质 2 设O为△ABC的外心，则∠BOC=2∠A，或∠BOC=360Â°-2∠A(还有两式)。

性质 3 设三角形的三条边长，外接圆的半径、面积分别为a、b、c,R、S△，则R=abc/4S△。

性质 4 过△ABC的外心O任作一直线与边AB、AC(或延长线)分别相交于P、Q两点，则AB/APÂ·sin2B+ AC/AQÂ·sin2C=sin2A+sin2B+sin2C。

性质 5 锐角三角形的外心到三边的距离之和等于其内切圆与外接圆半径之和。

4、重心

性质 1 设G为△ABC的重心，△ABC内的点Q在边BC、CA、AB边上的射影分别为D、E、F，则当Q与G重合时QDÂ·QEÂ·QF最大；反之亦然。

性质 2 设G为△ABC的重心，AG、BG、CG的延长线交△ABC的三边于D、E、F,则S△AGF=S△BGD=S△CGE；反之亦然。

性质 3 设G为△ABC的重心，则S△ABG=S△BCG=S△ACG= (1/3)S△ABC；反之亦然。

5、旁心

1、三角形一内角平分线和另外两顶点处的外角平分线交于一点，该点即为三角形的旁心。

2、每个三角形都有三个旁心。

3、旁心到三边的距离相等。

6、费马点

1.平面内一点P到△ABC三顶点的之和为PA+PB+PC，当点P为费马点时，距离之和最小。

在特殊三角形中：

2.三内角皆小于 120Â°的三角形，分别以 AB,BC,CA，为边，向三角形外侧做正三角形ABC1,ACB1,BCA1,然后连接AA1,BB1,CC1,则三线交于一点P,则点P就是所求的费马点.

3.若三角形有一内角大于或等于 120 度,则此钝角的顶点就是所求.

4.当△ABC为等边三角形时,此时外心与费马点重合

```
\*================================================*/
//外心：
//外心-计算法：
Point wai(Point a, Point b, Point c) {
    double a1 = a.x - b.x;
    double a2 = a.x - c.x;
    double b1 = a.y - b.y;
    double b2 = a.y - c.y;
    double c1 = a.x*a.x + a.y*a.y - b.x*b.x - b.y*b.y;
    double c2 = a.x*a.x + a.y*a.y - c.x*c.x - c.y*c.y;

    double t = (a1*b2-a2*b1)*2.0;
    return Point ((c1*b2-c2*b1)/t, (a1*c2-a2*c1)/t);
}
//外心-线段求交法：
Circle point3Circle(Point a, Point b, Point c) {
    Circle res;
    Point ab((a.x+b.x)/2, (a.y+b.y)/2), ac((a.x+c.x)/2, (a.y+c.y)/2);
    Point pab=(a-b).left90()+ab, pac=(a-c).left90()+ac;
    lineCross(ab, pab, ac, pac, res.c);
    res.r = dis(res.c, a);
    return res;
}
//重心：
//到三角形三顶点距离的平方和最小的点
//三角形内到三边距离之积最大的点
Point zhong(Point a, Point b, Point c) {
    return Point((a.x+b.x+c.x)/3, (a.y+b.y+c.y)/3);
}
//内心：
Point nei(Point a, Point b, Point c) {
    double A = dis(b, c), B = dis(a, c), C = dis(a, b);
```

```
    double P = A+B+C;
    return a*(A/P)+b*(B/P)+c*(C/P);
}
//垂心：垂心在数字较大(10^6 以上)的时候会有点不精确。。
Point chui(Point a, Point b, Point c) {
    return a+b+c-wai(a, b, c)*2;
}
//费马点：
Point fermat(Point a, Point b, Point c) {
    if(sig(cross(a, b, c)) < 0)swap(b, c);

    double ab = dis(a, b);
    double bc = dis(b, c);
    double ac = dis(a, c);

    double cosA = dot(a,b,c)/(ab*ac);
    double cosB = dot(b,a,c)/(ab*bc);
    double cosC = dot(c,a,b)/(ac*bc);
    if(sig(cosA+0.5)<=0)    return a;
    if(sig(cosB+0.5)<=0)    return b;
    if(sig(cosC+0.5)<=0)    return c;
    Point res;
    Point cc = (a-b).left90()*(sqrt(3)/2) + (a+b)*0.5;
    Point aa = (b-c).left90()*(sqrt(3)/2) + (b+c)*0.5;
    lineCross(a, aa, c, cc, res);
    return res;
}
//Find three numbers r, s, t which make p = r*a + s*b + t*c and r + s + t = 1
void parametric(Point p, Point a, Point b, Point c, double&r,double&s,double&t) {
   double d = cross(a, b, c);
   r = cross(p, b, c) / d;
   s = cross(a, p, c) / d;
   t = cross(a, b, p) / d;
}
```

极角排序

```
#include <vector>
#include <list>
#include <map>
#include <set>
#include <queue>
#include <deque>
#include <stack>
#include <bitset>
#include <algorithm>
#include <functional>
#include <numeric>
#include <utility>
#include <sstream>
#include <iostream>
#include <iomanip>
#include <cstdio>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <string.h>
#include <limits>

using namespace std;

#define maxn 1010
#define M_PI 3.14159265358979323846
```

```
const double eps = 1E-8;
int sig(double d) {
    return (d>eps) - (d<-eps);
}
struct Point {
    double x, y;
};
double cross(const Point & o, const Point & a, const Point & b) {
    return (a.x-o.x)*(b.y-o.y) - (b.x-o.x)*(a.y-o.y);
}

/**
 * 以下是极角排序：先init输入ps,n和原点o
 * ps不能三点共线！如果ps中发现了o，则将len-1
 * 各成员变量意思看注释
 * 统计的时候注意j-i=len的情况（共线）
 */
double ang[maxn*2];
bool cmp(int a, int b) {
    return ang[a] < ang[b];
}
struct Polar {
    int len;
    int num[maxn];
//返回射线(o,ps[i])逆时针转180度角以内的点的个数（不包括两边界），这里的ps[i]为消去o以后的i
    int ni[maxn];
//idx[i]表示排名为i的点是idx[i]; ni是idx的反函数，表示i号节点排名为ni[i]

    void init(Point * ps, int n, Point o) {
        static int idx[maxn*2];
        len = 0;
        for(int i = 0; i < n; i ++) {
            if(sig(ps[i].x-o.x)==0 && sig(ps[i].y-o.y)==0) continue;//跳过点o
            idx[len] = len;
            ang[len] = atan2(ps[i].y-o.y, ps[i].x-o.x);
            len ++;
        }
        sort(idx, idx+len, cmp);
        for(int i = 0; i < len; i ++) {
            idx[i+len] = idx[i]+len;
            ang[i+len] = ang[i]+M_PI*2.0;
        }
        for(int i = 0; i < len; i ++)  ni[idx[i]] = i;
        int j = 1;
        for(int i = 0; i < len; i ++) {
            while(ang[idx[j]]-ang[idx[i]]<M_PI)    j ++;
            num[idx[i]] = j-i-1;
        }
    }
    //由ps[a]逆时针转到ps[b]中间点的个数，不算边上的点；这里的ps[i]为消去o以后的i
    int numBtw(int a, int b) { //assume a!=b
        a = ni[a];
        b = ni[b];
        if(a<b) return b-a-1;
        return len-(a-b+1);
    }
};
//-----------------------华丽的分隔线-----------------------
/**
 * 2010天津网赛：Convex
 * 思路：极角排序+容斥原理
```

```
 * You will get the coordinates of n points in a plane.
 * It is guaranteed that there are no three points on a straight line.
 * You can choose any four points from these points to construct a quadrangle(凸四边形).
 * Now, please tell me how many convex quadrangles you can construct.
 */
/*
Point ps[maxn];
Polar po;
long long C(int n, int k) {
    if(k<0 || k>n)  return 0;
    k = min(k, n-k);
    long long res = 1;
    for(int i = 1; i <= k; i ++)    res *= i+n-k;
    for(int i = 1; i <= k; i ++)    res /= i;
    return res;
}
int main() {
    int t, n;
    for(scanf("%d", &t); t --; ) {
        scanf("%d", &n);
        for(int i = 0; i < n; i ++)
            scanf("%lf%lf", &ps[i].x, &ps[i].y);
        long long ans = 0;
        for(int i = 0; i < n; i ++) {
            po.init(ps, n, ps[i]);
            long long num = 0;
            for(int j = 0; j < n-1; j ++) {
                num += C(po.num[j], 2);
            }
            ans += C(n-1,3)-num;
        }
        ans = C(n,4)-ans;
        cout << ans << endl;
    }
    return 0;
}
*/

/*
 * 福州网络赛：How many stars
 * 思路：极角排序+容斥原理
 * There are N points. no three共线
 * He chooses three different points random to form a triangle
 * calculate how many points are strictly in the triangle

Point ps[maxn];
Polar po[maxn];
int t, n, m, a, b, c, ans;
int main() {
    scanf("%d", &t);
    for(int idx = 1; idx<=t; idx ++) {
        scanf("%d", &n);
        for(int i = 0; i < n; i ++)
            scanf("%lf%lf", &ps[i].x, &ps[i].y);
        for(int i = 0; i < n; i ++)
            po[i].init(ps, n, ps[i]);
        //init over!
        scanf("%d", &m);
        printf("Case %d:\n", idx);
        while(m --) {
            scanf("%d%d%d", &a, &b, &c);
```

```
            if(cross(ps[a],ps[b],ps[c])<0) swap(b,c);

            ans = -2*(n-3);
            ans += po[a].numBtw(b-(b>a),c-(c>a));
            ans += po[b].numBtw(c-(c>b),a-(a>b));
            ans += po[c].numBtw(a-(a>c),b-(b>c));
            ans += po[a].num[c-(c>a)];
            ans += po[b].num[a-(a>b)];
            ans += po[c].num[b-(b>c)];

            printf("%d\n", ans);
        }
    }
    return 0;
}
*/
```

圆

```
#define M_PI 3.14159265358979323846
#define sqr(v) ((v)*(v))
double my_acos(double d)    {    return acos(d>1?1:d<-1?-1:d);    }
double my_sqrt(double d)    {    return sqrt(max(d, 0.0));       }
int sig(double d) {
    return fabs(d)<1E-6 ? 0 : d > 0 ? 1 : -1;
}
struct Point {
    double x, y;
    Point(){}
    Point(double x, double y) : x(x), y(y) {}
    Point resize(double d) {
        d /= my_sqrt(x*x+y*y);
        return Point(x*d, y*d);
    }
    //左转 90 度
    Point left90() {
        return Point(-y, x);
    }
    Point operator - (const Point & o) const {
        return Point(this->x-o.x, this->y-o.y);
    }
    Point operator + (const Point & o) const {
        return Point(this->x+o.x, this->y+o.y);
    }
    Point operator * (double d) const {
        return Point(x*d, y*d);
    }
    void output() {
        printf("x = %.2f, y = %.2f\n", x, y);
    }
};
struct Circle {
    Point center;
    double radis;
};
double cross(Point o, Point a, Point b) {
    return (a.x-o.x)*(b.y-o.y) - (b.x-o.x)*(a.y-o.y);
}
double dot(Point o, Point a, Point b) {
    return (a.x-o.x)*(b.x-o.x) + (a.y-o.y)*(b.y-o.y);
}
double dis(Point a, Point b) {
    return my_sqrt(sqr(a.x-b.x) + sqr(a.y-b.y));
}
```

```
double radian(Point o, Point a, Point b) {
    return my_acos(dot(o,a,b)/(dis(o,a)*dis(o,b)));
}
//---------------------下面是新的函数--------------【【待测】】
//------------直线与圆相交-----------------
//返回点o到直线ab的距离，res保存o在ab上的投影
double pointToLine(Point o, Point a, Point b, Point & res) {
    double d = dis(a, b);
    double s = cross(a, b, o) / d;
    res = o + (a-b).left90()*(s/d);
    return fabs(s);
}
//判断直线与圆相交
//1 相交，0 相切，-1 不相交
int intersect(Point a, Point b, Circle c, Point &p1, Point &p2) {
    Point p;
    double d = pointToLine(c.center, a, b, p);
    int v = sig(c.radis-d);
    if(v != -1) {
        Point vec = (b-a).resize( my_sqrt(sqr(c.radis)-sqr(d)) );
        p1 = p+vec;
        p2 = p-vec;
    }
    return v;
}
//------------圆与圆相交-----------------
//判断圆与圆相交
/* ab重合，返回 0
 * a包含b，返回 1
 * b包含a，返回-1
 * ab相离，返回 2
 *
 + a内切b，返回 3（a大）
 + b内切a，返回-3
 + ab外切，返回 4
 + ab相交，返回 5
 */
int intersect(Circle a, Circle b, Point &p1, Point &p2) {
    double d = dis(a.center, b.center);
    int s1 = sig(d-fabs(a.radis-b.radis));
    int s2 = sig(d-fabs(a.radis+b.radis));
    if(sig(d)==0 || s1<0)   return sig(a.radis-b.radis);    //重合/内含 0/+-1
    if(s2>0)    return 2;                                   //相离 2

    double t = (sqr(a.radis)+sqr(d)-sqr(b.radis)) / (2*d); //t=cos(theta)*r1
    Point p = (b.center-a.center).resize(t) + a.center;        //中间那个点
    Point                                vec                                =
(b.center-a.center).left90().resize(my_sqrt(sqr(a.radis)-sqr(t)));
    p1 = p+vec;
    p2 = p-vec;

    if(s1==0)   return 3 * sig(a.radis-b.radis);            //内切+-3
    if(s2==0)   return 4;                                   //外切 4
    return 5;                                               //相交 5
}
//------------点与圆的最近点-----------------
//返回o到c的最近点，如果o==c.center，则返回c.center
//Point to Circle
Point p2c(Point &p, Circle &c) {
```

```
    if(sig(dis(p, c.center))==0)   return c.center;
    return (p-c.center).resize(c.radis)+c.center;
}
//---------------以下函数【【测完了】】!!!
//------------两圆交面积--------------------
//得到弓形面积，r是半径，theta是圆心角，theta属于[0, 2*PI]
double getGong(double r, double ang) {
    return (ang-sin(ang))*r*r/2;
}
//返回a和b的交面积
double intersectArea(Circle a, Circle b) {
    double r1 = max(a.radis, b.radis);
    double r2 = min(a.radis, b.radis);
    double d = dis(a.center, b.center);
    if(sig(d-(r1+r2)) >= 0) return 0.0;
    if(sig(d-(r1-r2)) <= 0) return M_PI*r2*r2;
    double ang1 = 2 * my_acos((r1*r1+d*d-r2*r2) / (2*d*r1));
    double ang2 = 2 * my_acos((r2*r2+d*d-r1*r1) / (2*d*r2));
    return getGong(r1, ang1) + getGong(r2, ang2);
}
int main() {    //poj2546
    Circle a, b;
    Point p1, p2;
    while(cin >> a.center.x >> a.center.y >> a.radis >> b.center.x >> b.center.y >>
b.radis) {
        printf("%.3f\n", intersectArea(a, b));
    }
    return 0;
}
```

圆并_离散化

```
#define maxn 310
#define sqr(v) ((v)*(v))
const double eps = 1E-6;
int sig(double d)          {   return (d>eps)-(d<-eps);       }
double my_acos(double d)   {   return acos(d>1?1:d<-1?-1:d);  }
double my_sqrt(double d)   {   return sqrt(max(d, 0.0));      }
struct Point {
    double x, y;
};
double dis(Point a, Point b) {
    return my_sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
struct Circle {
    Point c;
    double r;
};
//---------------------下面是新的函数----------------------
//////////////////////////////////////////////////////---------------
Circle * cs;
double x1, x2;
struct Arc {
    int id;
    double y1, y2;
    char type;  //1 进-1 出
    Arc() {}
    Arc(int id, double y1, double y2, char type) : id(id),y1(y1),y2(y2),type(type){}
    bool operator < (const Arc & arc) const {
        int v = sig(y1+y2-arc.y1-arc.y2);
        if(v != 0)              return v<0;
        if(type != arc.type)    return type>arc.type;
        return (type==-1) ^ (sig(cs[id].r-cs[arc.id].r)<0);
```

```
    }
    double area() {
        double x = cs[id].c.x, y = cs[id].c.y;
        double ang = my_acos(((x1-x)*(x2-x)+(y1-y)*(y2-y))/sqr(cs[id].r));
        return (ang-sin(ang))*sqr(cs[id].r)/2;
    }
};
bool dcmp(double a, double b) {
    return sig(a-b)<0;
}
double union_area(Circle *cs0, int cn) {
    static double xs[maxn*maxn+10*maxn+10];
    static Arc as[maxn*2];

    cs = cs0;
    int xn = 0;

    for(int i = 0; i < cn; i ++) {
        Circle ci = cs[i];
        if(sig(ci.r)<=0)    continue;
        xs[xn ++] = ci.c.x-ci.r;
        xs[xn ++] = ci.c.x+ci.r;

        for(int j = i+1; j < cn; j ++) {
            Circle cj = cs[j];
            double d = dis(ci.c, cj.c);
            if(sig(d)==0  ||  sig(ci.r+cj.r-d)<0  ||  sig(fabs(ci.r-cj.r)-d)>0)
    continue;

            double d1 = ( sqr(d)+sqr(ci.r)-sqr(cj.r) ) / (2*d);
            double d2 = my_sqrt( sqr(ci.r)-sqr(d1) );

            double vx = (ci.c.y-cj.c.y) * d2 / d;
            double vvx= (cj.c.x-ci.c.x) * d1 / d;

            xs[xn ++] = ci.c.x+vvx+vx;
            xs[xn ++] = ci.c.x+vvx-vx;
        }
    }
    sort(xs, xs+xn, dcmp);
    int idx = 1;
    for(int i = 1; i < xn; i ++)
        if(sig(xs[i]-xs[i-1])!=0)  xs[idx++]=xs[i];
    xn = min(xn, idx);

    double res = 0;

    for(int i = 0; i < xn-1; i ++) {
        x1 = xs[i], x2 = xs[i+1];
        double mid = (x1+x2)/2;
        int an = 0;
        for(int j = 0; j < cn; j ++) {
            Circle c = cs[j];
            if(sig(c.c.x-c.r-mid)>=0 || sig(c.c.x+c.r-mid)<=0) continue;
            double d1 = my_sqrt(sqr(c.r)-sqr(c.c.x-x1));
            double d2 = my_sqrt(sqr(c.r)-sqr(c.c.x-x2));
            as[an ++] = Arc(j, c.c.y+d1, c.c.y+d2, -1);
            as[an ++] = Arc(j, c.c.y-d1, c.c.y-d2, 1);
        }
        sort(as, as+an);
        int i, j, deg;
        for(i = 0; i < an; i = j + 1) {
            for(j=i, deg=1; deg; deg+=as[++ j].type);
```

```
            res += (x2-x1)*(as[j].y1+as[j].y2-as[i].y1-as[i].y2)/2;
            res += as[i].area() + as[j].area();
        }
    }
    return res;
}
Circle C[maxn*3];
int main() {
    freopen("ChaeYeon.in", "r", stdin);
    int t;
    for(scanf("%d", &t); t --; ) {
        int rNum, gNum, bNum;
        scanf("%d%d%d", &rNum, &gNum, &bNum);
        for(int i=0;i<rNum;i++)scanf("%lf%lf%lf", &C[i].c.x, &C[i].c.y, &C[i].r);
        for(int i=rNum; i<rNum+gNum; i++)
                    scanf("%lf%lf%lf",&C[i].c.x,&C[i].c.y, &C[i].r);
        for(int i = rNum+gNum; i < rNum+gNum+bNum; i ++)
                    scanf("%lf%lf%lf", &C[i].c.x, &C[i].c.y, &C[i].r);

        double r, g, b, rg, rb, gb, rgb;

        r = union_area(C, rNum);
        g = union_area(C+rNum, gNum);
        b = union_area(C+rNum+gNum, bNum);

        rg = union_area(C, rNum+gNum);
        gb = union_area(C+rNum, gNum+bNum);
        rgb =union_area(C, rNum+gNum+bNum);

        for(int i = rNum; i < rNum+bNum; i ++) {
            C[i] = C[i+gNum];
        }
        rb = union_area(C, rNum+bNum);

        rgb = r+g+b-rg-rb-gb+rgb;

        rg = r+g-rg;
        gb = g+b-gb;
        rb = r+b-rb;

        printf("%.2f %.2f %.2f %.2f %.2f %.2f %.2f\n",
                r-rg-rb+rgb+eps,
                g-gb-rg+rgb+eps,
                b-rb-gb+rgb+eps,
                rgb+eps,
                rg-rgb+eps,
                rb-rgb+eps,
                gb-rgb+eps
        );
    }
    return 0;
}
```

圆并离散化＋矩形交(spoj orz)

```
//题目: http://www.spoj.pl/problems/ORZ/
/**
 * ks个small圆，半径为0.58
 * kl个large圆，半径为1.31
 * 求ks+kl个圆并 和 矩形[(0,0),(maxX,maxY)]的交面积
 */
double maxX, maxY;
#define maxn 310
```

```
#define sqr(v) ((v)*(v))
const double eps = 1E-6;
int sig(double d)       {   return (d>eps)-(d<-eps);        }
double my_acos(double d)    {   return acos(d>1?1:d<-1?-1:d);  }
double my_sqrt(double d)    {   return sqrt(max(d, 0.0));       }
struct Point {
    double x, y;
};
double dis(Point a, Point b) {
    return my_sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
struct Circle {
    Point c;
    double r;
};
//---------------------下面是新的函数----------------------
////////////////////////////////////////////////////---------------
Circle * cs;
double x1, x2;
struct Arc {
    int id;
    double y1, y2;
    char type;  //1 进-1 出
    Arc() {}
    Arc(int id, double y1, double y2, char type):id(id),y1(y1),y2(y2),type(type){}
    bool operator < (const Arc & arc) const {
        int v = sig(y1+y2-arc.y1-arc.y2);
        if(v != 0)              return v<0;
        if(type != arc.type)    return type>arc.type;
        return (type==-1) ^ (sig(cs[id].r-cs[arc.id].r)<0);
    }
    double area() {
        double x = cs[id].c.x, y = cs[id].c.y;
        double ang = my_acos(((x1-x)*(x2-x)+(y1-y)*(y2-y))/sqr(cs[id].r));
        return (ang-sin(ang))*sqr(cs[id].r)/2;
    }
    void output() {
        printf("id = %d, y1 = %.2f, y2 = %.2f, type = %d\n", id, y1, y2, type);
    }
};
double getArea(Arc s, Arc e) {
    double res =
     (x2-x1) * (min(e.y1,maxY)+min(e.y2,maxY)-max(s.y1,0.0)-max(s.y2,0.0)) / 2.0;
    if(sig(s.y1)>0 || sig(s.y2)>0)             res += s.area();
    if(sig(e.y1-maxY)<0 || sig(e.y2-maxY)<0)   res += e.area();
    return res;
}
bool dcmp(double a, double b) {
    return sig(a-b)<0;
}
#define filt if(xs[xn-1]>maxX||xs[xn-1]<0)xn--;
double union_area(Circle *cs0, int cn) {
    static double xs[maxn*maxn+10*maxn+10];
    static Arc as[maxn*2];

    cs = cs0;
    int xn = 0;
    xs[xn ++] = 0;
    xs[xn ++] = maxX;

    for(int i = 0; i < cn; i ++) {
        Circle ci = cs[i];
        if(sig(ci.r)<=0)    continue;
```

```
        xs[xn ++] = ci.c.x-ci.r;    filt;
        xs[xn ++] = ci.c.x+ci.r;    filt;

        if(sig(ci.c.y+ci.r-maxY)>0) {
            double dx = my_sqrt(sqr(ci.r)-sqr(maxY-ci.c.y));
            xs[xn ++] = ci.c.x+dx;  filt;
            xs[xn ++] = ci.c.x-dx;  filt;
        }
        if(sig(ci.c.y-ci.r)<0) {
            double dx = my_sqrt(sqr(ci.r)-sqr(ci.c.y));
            xs[xn ++] = ci.c.x+dx;  filt;
            xs[xn ++] = ci.c.x-dx;  filt;
        }
        for(int j = i+1; j < cn; j ++) {
            Circle cj = cs[j];
            double d = dis(ci.c, cj.c);
            if(sig(d)==0 || sig(ci.r+cj.r-d)<0 || sig(fabs(ci.r-cj.r)-d)>0)
                continue;

            double d1 = ( sqr(d)+sqr(ci.r)-sqr(cj.r) ) / (2*d);
            double d2 = my_sqrt( sqr(ci.r)-sqr(d1) );

            double vx = (ci.c.y-cj.c.y) * d2 / d;
            double vvx= (cj.c.x-ci.c.x) * d1 / d;

            xs[xn ++] = ci.c.x+vvx+vx;  filt;
            xs[xn ++] = ci.c.x+vvx-vx;  filt;
        }
    }
    sort(xs, xs+xn, dcmp);
    int idx = 1;
    for(int i = 1; i < xn; i ++)
        if(sig(xs[i]-xs[i-1])!=0)  xs[idx++]=xs[i];
    xn = min(xn, idx);

    double res = 0;

    for(int i = 0; i < xn-1; i ++) {
        x1 = xs[i], x2 = xs[i+1];
        double mid = (x1+x2)/2;
        int an = 0;
        for(int j = 0; j < cn; j ++) {
            Circle c = cs[j];
            if(sig(c.c.x-c.r-mid)>=0 || sig(c.c.x+c.r-mid)<=0) continue;
            double d1 = my_sqrt(sqr(c.r)-sqr(c.c.x-x1));
            double d2 = my_sqrt(sqr(c.r)-sqr(c.c.x-x2));
            as[an ++] = Arc(j, c.c.y+d1, c.c.y+d2, -1);
            as[an ++] = Arc(j, c.c.y-d1, c.c.y-d2, 1);
        }
        sort(as, as+an);
        int i, j, deg;
        for(i = 0; i < an; i = j + 1) {
            for(j=i, deg=1; deg; deg+=as[++ j].type);
            res += getArea(as[i], as[j]);
        }
    }
    return res;
}
Circle C[maxn*3];
int main() {
    int ks, kl;
    while(scanf("%lf%lf%d%d",&maxX,&maxY,&ks,&kl),!(ks==0&&kl==0&&maxX==0&&maxY==0))
{
```

```
        for(int i = 0; i < ks; i ++)
            scanf("%lf%lf", &C[i].c.x, &C[i].c.y), C[i].r = 0.58;
        for(int i = ks;i<ks+kl;i ++)
                    scanf("%lf%lf", &C[i].c.x, &C[i].c.y), C[i].r = 1.31;
        double ans = maxX*maxY-union_area(C, ks+kl);
        printf("%.2f\n", ans+eps);
    }
    return 0;
}
```

圆交圆并(hdu3229 jiajia's robot)

//不知道现在的为什么过不去hdu3467，囧。。。

```
#define SZ(x) ((int) x.size())
#define SQR(v) ((v) * (v))
double my_acos(double d)    {    return acos(d>1?1:d<-1?-1:d);    }
double my_sqrt(double d)    {    return sqrt(max(d, 0.0));       }
const double eps = 1e-6;
int sig(double a) {
    return (a > eps) - (a < -eps);
}
struct Point {
    double x, y;
    Point(double x, double y):x(x), y(y) {}
    Point() {}
    bool operator == (const Point &a) const {
        return sig(x - a.x) == 0 && sig(y - a.y) == 0;
    }
    Point operator + (const Point &a) const {
        return Point(x + a.x, y + a.y);
    }
    Point operator - (const Point &a) const {
        return Point(x - a.x, y - a.y);
    }
    Point operator * (const double &a) const {
        return Point(x * a, y * a);
    }
    Point resize(double a) {
        a /= my_sqrt(SQR(x) + SQR(y));
        return Point(x * a, y * a);
    }
    Point left90() {
        return Point(-y, x);
    }
    Point right90() {
        return Point(y, -x);
    }
};
double dis2(Point a, Point b) {
    return SQR(a.x - b.x) + SQR(a.y - b.y);
}
double dis(Point a, Point b) {
    return my_sqrt(SQR(a.x - b.x) + SQR(a.y - b.y));
}
double cross(Point o, Point a, Point b) {
    return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x);
}
double dot(Point o, Point a, Point b) {
    return (a.x - o.x) * (b.x - o.x) + (a.y - o.y) * (b.y - o.y);
}
struct Circle {
    Point c;
    double r;
    Circle(Point c, double r):c(c), r(r) {}
```

```
    Circle() {}
    bool operator == (const Circle &a) const {
        return c == a.c && sig(r - a.r) == 0;
    }
    bool in(Circle a) {
        return sig(r + dis(c, a.c) - a.r) <= 0;
    }
};
double cal_angle(Circle c, Point a, Point b) {
    double k = dot(c.c, a, b) / SQR(c.r);
    return my_acos(k);
}
double cal_area(Circle c, Point a, Point b) {
    return SQR(c.r) * cal_angle(c, a, b) / 2 - cross(c.c, a, b) / 2;
}
Point mid;    //cmp辅助专用!
bool cmp(Point a, Point b) {
    return atan2(a.y - mid.y, a.x - mid.x) < atan2(b.y - mid.y, b.x - mid.x);
}
bool circles_intersection(Circle a, Circle b, Point &c1, Point &c2) {
    double dd = dis(a.c, b.c);
    if (sig(dd - (a.r + b.r)) > 0)       return false;
    double l = (dd + (SQR(a.r) - SQR(b.r)) / dd) / 2;
    double h = my_sqrt(SQR(a.r) - SQR(l));
    c1 = a.c + (b.c - a.c).resize(l) + (b.c - a.c).left90().resize(h);
    c2 = a.c + (b.c - a.c).resize(l) + (b.c - a.c).right90().resize(h);
    return true;
}
bool cover1(Circle c, Point a, Point b, vector<Circle> &cir) {
    Point p = c.c + ((a + b) * 0.5- c.c).resize(c.r);
    for (vector<Circle>::iterator it = cir.begin(); it != cir.end(); ++it) {
        if (sig(dis2(p, it->c) - SQR(it->r)) > 0) {
            return false;
        }
    }
    return true;
}
//////////////下面开始计算圆交，如果要找唯一的一点的时候，就测试poc的每一个点
double circle_inter_area(vector<Circle> &cs) {
    int zx[] = {0, 1, 0, -1   };
    int zy[] = {1, 0, -1, 0   };
    vector<Circle> cir;
    for (int i = 0; i < SZ(cs); ++i) {
        if (sig(cs[i].r) == 0)
            goto out;
        for (int j = i + 1; j < SZ(cs); ++j)
            if (cs[i] == cs[j])
                goto out;
        for (int j = 0; j < SZ(cs); ++j)
            if (!(cs[i] == cs[j]) && cs[j].in(cs[i]))
                goto out;
        cir.push_back(cs[i]);
        out:;
    }
    vector<vector<Point> >poc(SZ(cir));    //points on circles
    for (int i = 0; i < SZ(cir); ++i) {
        for (int z = 0; z < 4; ++z) {
            poc[i].push_back(cir[i].c + Point(zx[z], zy[z]).resize(cir[i].r));
        }
    }
    for (int i = 0; i < SZ(cir); ++i) {
        for (int j = i + 1; j < SZ(cir); ++j) {
            Point a, b;
```

```
            if (circles_intersection(cir[i], cir[j], a, b)) {
                poc[i].push_back(a);
                poc[i].push_back(b);
                poc[j].push_back(a);
                poc[j].push_back(b);
            } else {
                return 0;
            }
        }
    }
    for (int i = 0; i < SZ(cir); ++i) {
        mid = (cir[i].c);
        sort(poc[i].begin(), poc[i].end(), cmp);
        poc[i].erase(unique(poc[i].begin(), poc[i].end()), poc[i].end());
    }
    double ans = 0;
    bool ok = false;
    for (int i = 0; i < SZ(cir); ++i) {
        poc[i].push_back(poc[i][0]);
        for (int j = 0; j < SZ(poc[i])-1; ++j) {
            Point a = poc[i][j];
            Point b = poc[i][j+1];

            if (cover1(cir[i], a, b, cir)) {
                ans += cross(Point(0, 0), a, b) / 2;
                ans += cal_area(cir[i], a, b);
                ok = true;
            }
        }
    }
    return ans;
}
//////////////下面开始计算圆并
bool cover2(Circle c, Point a, Point b, vector<Circle> &cir) {
    Point p = c.c + ((a + b) * 0.5- c.c).resize(c.r);
    for (vector<Circle>::iterator it = cir.begin(); it != cir.end(); ++it) {
        if (sig(dis2(p, it->c) - SQR(it->r)) < 0) {
            return true;
        }
    }
    return false;
}
double circle_union_area(vector<Circle> &cs) {
    int zx[] = {0, 1, 0, -1};
    int zy[] = {1, 0, -1, 0};
    vector<Circle> cir;
    for (int i = 0; i < SZ(cs); ++i) {
        if (sig(cs[i].r) == 0)
            goto out;
        for (int j = i + 1; j < SZ(cs); ++j)
            if (cs[i] == cs[j])
                goto out;
        for (int j = 0; j < SZ(cs); ++j)
            if (!(cs[i] == cs[j]) && cs[i].in(cs[j]))
                goto out;
        cir.push_back(cs[i]);
        out:;
    }
    vector<vector<Point> >poc(SZ(cir));    //points on circles
    for (int i = 0; i < SZ(cir); ++i) {
        for (int z = 0; z < 4; ++z) {
            poc[i].push_back(cir[i].c + Point(zx[z], zy[z]).resize(cir[i].r));
        }
```

```
    }
    for (int i = 0; i < SZ(cir); ++i) {
        for (int j = i + 1; j < SZ(cir); ++j) {
            Point a, b;
            if (circles_intersection(cir[i], cir[j], a, b)) {
                poc[i].push_back(a);
                poc[i].push_back(b);
                poc[j].push_back(a);
                poc[j].push_back(b);
            }
        }
    }
    for (int i = 0; i < SZ(cir); ++i) {
        mid = (cir[i].c);
        sort(poc[i].begin(), poc[i].end(), cmp);
        poc[i].erase(unique(poc[i].begin(), poc[i].end()), poc[i].end());
    }
    double ans = 0;
    for (int i = 0; i < SZ(cir); ++i) {
        poc[i].push_back(poc[i][0]);
        for (int j = 0; j < SZ(poc[i])-1; ++j) {
            Point a = poc[i][j];
            Point b = poc[i][j+1];
            if (!cover2(cir[i], a, b, cir)) {
                ans += cross(Point(0, 0), a, b) / 2;
                ans += cal_area(cir[i], a, b);
            }
        }
    }
   return ans;
}
Circle get(Point a, Point b) {
    Point o = (a+b) * 0.5;
    return Circle(o, dis(a,b)/2);
}
int main() {
    Circle cir[10];

    Point a, b, c, d;
    for(int idx = 1; cin>>a.x>>a.y>>b.x>>b.y>>c.x>>c.y>>d.x>>d.y; idx ++) {
        if(a.x==0&&a.y==0&&b.x==0&&b.y==0&&c.x==0&&c.y==0&&d.x==0&&d.y==0)break;
        cir[1] = get(a, d);
        cir[2] = get(a, c);
        cir[3] = get(b, c);
        cir[4] = get(b, d);

        vector<Circle> vec;
        vec.push_back(cir[1]);
        vec.push_back(cir[2]);
        vec.push_back(cir[3]);
        vec.push_back(cir[4]);

        double sub = circle_inter_area(vec);
        double all = circle_union_area(vec);

        printf("Case %d: %.3f\n\n", idx, all-sub+eps);
    }
   return 0;
}
```

球

```
//lat1,lat2 纬度.lng1,lng2:经度.弧度制！！
double ang(double lat1, double lng1, double lat2, double lng2) {
```

```
    return acos(cos(lat1)*cos(lat2)*cos(lng1-lng2)+sin(lat1)*sin(lat2));
}
//直线距离
double dis_line(double lat1, double lng1, double lat2, double lng2, double r) {
    return r*sqrt(2-2*(cos(lat1)*cos(lat2)*cos(lng1-lng2)+sin(lat1)*sin(lat2)));
}
//球面距离
double dis_sphere(double lat1, double lng1, double lat2, double lng2, double r) {
    return r*ang(lat1, lng1, lat2, lng2);
}
//下面是 3407 的题目
/**
 * Input:
 *  55 0 N 40 0 E
 *  59 0 N 49 30 E
 * Output:
 *  725.979
 */
void scan(double &lat, double &lng) {
    int a0, a1, b0, b1;
    char c, d;
    scanf("%d%d %c%d%d %c", &a0, &a1, &c, &b0, &b1, &d);
    lat = (a0+a1/60.0)*M_PI/180.0; if(c=='S')  lat = -lat;
    lng = (b0+b1/60.0)*M_PI/180.0; if(d=='W')  lng = -lng;
}
int main() {
    double lat1, lng1, lat2, lng2;
    scan(lat1, lng1);
    scan(lat2, lng2);
    printf("%.3f\n", dis_sphere(lat1, lng1, lat2, lng2, 6370));
    return 0;
}
```

3D 几何类库

```
using namespace std;
#define maxn 10010
# define M_PI       3.14159265358979323846 /* pi */
#define sqr(v) ((v)*(v))
const double eps = 1E-8;
int sig(double d) {
    return (d>eps) - (d<-eps);
}
struct Point {
    double x, y, z;
    Point(double x, double y, double z) : x(x), y(y), z(z) {}
    Point() {}
    Point operator + (const Point & p) const {
        return Point(x+p.x, y+p.y, z+p.z);
    }
    Point operator - (const Point & p) const {
        return Point(x-p.x, y-p.y, z-p.z);
    }
    Point operator * (const double & d) const {
        return Point(x*d, y*d, z*d);
    }
    Point resize(double d) {
        d /= sqrt(x*x+y*y+z*z);
        return Point(x*d, y*d, z*d);
    }
    double len() const {
        return sqrt(x*x+y*y+z*z);
    }
    bool operator < (const Point & p) const {
```

```
        return sig(x-p.x)!=0 ? x<p.x : sig(y-p.y)!=0 ? y<p.y : sig(z-p.z)<0;
    }
    bool operator == (const Point & p) const {
        return sig(x-p.x)==0 && sig(y-p.y)==0 && sig(z-p.z)==0;
    }
    void input() {
        scanf("%lf%lf%lf", &x, &y, &z);
    }
    void output() const {
        printf("x = %.2f, y = %.2f, z = %.2f\n", x, y, z);
    }
};
//点乘
double dot(const Point & a, const Point & b) {
    return a.x*b.x + a.y*b.y + a.z*b.z;
}
double dot(const Point & o, const Point & a, const Point & b) {
    return dot(a-o, b-o);
}
//叉乘
Point cross(const Point & a, const Point & b) {
    return Point(a.y*b.z-a.z*b.y, a.z*b.x-a.x*b.z, a.x*b.y-a.y*b.x);
}
Point cross(const Point & o, const Point & a, const Point & b) {
    return cross(a-o,b-o);
}
//两点距离
double dis(const Point & a, const Point & b) {
    return sqrt(sqr(a.x-b.x) + sqr(a.y-b.y) + sqr(a.z-b.z));
}
//点到直线距离
double dis(const Point & o, const Point & a, const Point & b) {
    return cross(o, a, b).len() / dis(a,b);
}
//返回o到ab直线的投影点，对于o在ab上也适合
Point pointToLine(const Point & o, const Point & a, const Point & b) {
    Point fa = cross(o, a, b); //oab的垂向量
    Point vec = cross(b-a, fa);//vec为在oab平面上，且由o指向ab的一条向量
    return o + vec.resize(fa.len()/dis(a,b));
}
//判断三点共线
bool sameLine(const Point & a, const Point & b, const Point & c) {
    return sig(cross(a, b, c).len())==0;
}
//判断四点共面
bool sameFace(const Point & a, const Point & b, const Point & c, const Point & d)
{
    return sig(dot(b-a, cross(a, c, d))) == 0;
}
//p绕着se向量，逆时针转ang角度
Point rotate(Point p, Point s, Point e, double ang) {
    if(sameLine(p, s, e)) {
        //sprintf("keke..\n"); error!
        return p;
    }
    Point fa1 = cross(s, e, p);
    Point fa2 = cross(e-s, fa1);

    double len = fabs(cross(p, e, s).len() / dis(e, s));   //圆心角
    fa2 = fa2.resize(len);
    fa1 = fa1.resize(len);
```

```
    Point h = p + fa2;
    Point pp = h + fa1;

    Point res = h + (p-h)*cos(ang) + (pp-h)*sin(ang);
    return res;
}
//vec是一个向量，将vec转到z轴向量，返回p点所转到的位置
Point rotate(Point p, Point vec) {
    Point z(0, 0, 1);
    Point fa = cross(vec, z);
    return rotate(p, Point(0,0,0), fa, acos( dot(z,vec)/vec.len() ));
}

struct Face {
    Point a, b, c;
    Face() {}
    Face(Point a, Point b, Point c) : a(a), b(b), c(c) {}
    Point fa() const {
        return cross(a, b, c);
    }
    bool same_side(Point q , Point  p){
        return sig ( dot(a - q, cross(q, b, c))
            * dot(a - p , cross(p, b, c)) ) > 0 ;
    }
    bool inFace(Point q) const {    //判断某点是否在该平面上
        return sameFace(a, b, c, q);
    }
    bool operator == (const Face & face) const {
        Point fa1 = fa();
        Point fa2 = face.fa();
        if(sig(cross(fa1,fa2).len())!=0)    return false;
        return inFace(face.a);
    }
};
//判断两直线平行
bool px(const Point & a, const Point & b, const Point & c, const Point & d) {
    return sig(cross(b-a, d-c).len()) == 0;
}
//判断直线与平面平行
bool px(const Face & f, const Point & a, const Point & b) {
    return sig(dot(f.fa(), b-a))==0;
}
//判断两平面平行
bool px(const Face & f1, const Face & f2) {
    return sig(cross(f1.fa(),f2.fa()).len()) == 0;
}
//判断两直线垂直
bool cz(const Point & a, const Point & b, const Point & c, const Point & d) {
    return sig(dot(b-a, d-a)) == 0;
}
//判断直线与平面垂直
bool cz(const Face & f, const Point & a, const Point & b) {
    return sig(cross(f.fa(), b-a).len()) == 0;
}
//判断两平面垂直
bool cz(const Face & f1, const Face & f2) {
    return sig(dot(f1.fa(), f2.fa())) == 0;
}
//点到平面距离
double dis(const Face & f, const Point & p) {
    Point fa = f.fa();
    return fabs(dot(fa, f.a-p) / fa.len());
```

```
}
//两直线距离
double dis(const Point & a, const Point & b, const Point & c, const Point & d) {
    Point fa = cross(b-a,d-c);
    if(fa.len() == 0) { //两直线平行
        return cross(c, a, b).len() / dis(a, b);
        return dis(c, a, b);        //或者用这个，点到直线距离
    } else {
        return fabs(dot(c-a, fa) / fa.len());
    }
}

//求ab直线与f的交点，断言ab于f不平行
Point intersect(const Face & f, const Point & a, const Point & b) {
    Point fa = f.fa();
    double t = dot(fa,f.a-a) / dot(fa,b-a);
    return a+(b-a)*t;
}
//求两直线交点,【断言】两直线共面且不平行
Point intersect(const Point & a, const Point & b, const Point & c, const Point &
d) {
    Point e = d + cross(a-b, c-d);
    return intersect(Face(c,d,e), a, b);
    //或者用下面代码：
    Point fa = cross(c, d, e);
    double t = dot(fa, c-a) / dot(fa, b-a);
    return a+(b-a) * t;
}
//求f1,f2 的交线，断言f1 与f2 不平行
void intersect(const Face & f1, const Face & f2, Point & p1, Point & p2) {
    p1 = false==px(f2, f1.a, f1.b) ? intersect(f2, f1.a, f1.b) : intersect(f2, f1.b,
f1.c);
    p2 = p1 + cross(f1.fa(), f2.fa());
}
//求两直线的共垂线，p1 返回ab直线上的公垂线交点，p2 返回cd直线上的公垂线交点
void gc(const Point & a, const Point & b, const Point & c, const Point & d, Point
& p1, Point & p2) {
    Point e = d + cross(a-b, c-d);
    p1 = intersect(Face(c,d,e), a, b);
    p2 = pointToLine(p1, c, d);
}
//判断点是否在空间三角形内部，包括边界，【断言】abc不共线，并且oabc共面
bool inTriangle(const Point & o, const Point & a, const Point & b, const Point &
c) {
    double s = cross(a, b, c).len();
    double s1 = cross(o, a, b).len();
    double s2 = cross(o, b, c).len();
    double s3 = cross(o, a, c).len();
//  return sig(s - s1 - s2 - s3) == 0;
    return sig(s-s1-s2-s3)==0 && sig(s1)==1 && sig(s2)==1 && sig(s3)==1;//不包括边
界的版本
}
//判断某个点是否在线段上，断言oab共线        -1 表示在上面，0 表示在线段边上，1 表示在线段外
int onSeg(const Point & o, const Point & a, const Point & b) {
    return sig(dot(o, a, b));
}

/*
//ural1754  金字塔内一点的最近距离
double m, h, H;
Point p0, p1, p2, p3, p4;
```

```
int main() {
    while(cin >> m >> h >> H) {
        cin >> p0.x >> p0.y;    p0.z = h;
        cin >> p1.x >> p1.y;    p1.z = 0;
        cin >> p2.x >> p2.y;    p2.z = 0;
        cin >> p3.x >> p3.y;    p3.z = 0;
        cin >> p4.x >> p4.y;    p4.z = H;

        Face f1(p1, p2, p3);
        Face f2(p1, p2, p4);
        Face f3(p1, p3, p4);
        Face f4(p2, p3, p4);

        double ans = 1E30;
        ans = min(ans, dis(f1, p0));
        ans = min(ans, dis(f2, p0));
        ans = min(ans, dis(f3, p0));
        ans = min(ans, dis(f4, p0));

        if(sig(ans + h - m) <= 0) {
            cout << "YES" << endl;
        } else {
            cout << "NO" << endl;
        }
    }
    return 0;
}
*/
/*
//poj-2852  两条直线公垂线的中点
int main() {
    double t;
    double ha, hb;

    double r = 10000.0;

    double a1, b1, a2, b2;
    cin >> t >> ha >> hb;

    Point A1(0, 0, ha), B1(100, 0, hb);

    for(int idx = 1; idx <= t; idx ++) {
        cin >> a2 >> b2 >> a1 >> b1;
        a1 = a1 * M_PI / 180;
        b1 = b1 * M_PI / 180;
        a2 = a2 * M_PI / 180;
        b2 = b2 * M_PI / 180;

        Point A2 = A1 + Point(cos(a1), sin(a1), tan(a2))*r;
        Point B2 = B1 + Point(cos(b1), sin(b1), tan(b2))*r;
        Point p, q;
        gc(A1, A2, B1, B2, p, q);

        printf("%d: %.0f\n", idx, (p.z+q.z)/2);
    }
    return 0;
}
*/
/*
//uva-11275 判断两个空间三角形是否有交点
//判断两个三角形是否有交点
bool intersect(Point * P1, Point * P2) {
    Face f1(P1[0], P1[1], P1[2]);
```

```
    Face f2(P2[0], P2[1], P2[2]);
    if(px(f1,f2)) { //两三角形共面
        if(!sameFace(P1[0], P1[1], P1[2], P2[0]))  return false;
        for(int i = 0; i < 3; i ++) {
            if(inTriangle(P1[i], P2[0], P2[1], P2[2]) ||
                    inTriangle(P2[i], P1[0], P1[1], P1[2]))
                return true;
        }
        return false;
    }
    Point l1, l2;
    intersect(f1, f2, l1, l2); //公垂线

    Point arr1[3], arr2[3];
    int len1 = 0, len2 = 0, a, b;

    for(int i = 0; i < 3; i ++) {
        a = 0;  a+=a==i;
        b = a+1;b+=b==i;
        if(!px( P1[a], P1[b], l1,l2)) {
            arr1[len1] = intersect(P1[a], P1[b], l1, l2);
            if(onSeg(arr1[len1], P1[a], P1[b]) <= 0)   len1 ++;
        }
        if(!px( P2[a], P2[b], l1,l2)) {
            arr2[len2] = intersect(P2[a], P2[b], l1, l2);
            if(onSeg(arr2[len2], P2[a], P2[b]) <= 0)   len2 ++;
        }
    }
    sort(arr1, arr1+len1);
    sort(arr2, arr2+len2);
    len1 = unique(arr1, arr1+len1) - arr1;
    len2 = unique(arr2, arr2+len2) - arr2;
    if(len1 == 3 || len2 == 3) while(1);   //assume not !
    if(len1 == 0 || len2 == 0) return false;
    if(len1 == 1 && len2 == 1) return arr1[0]==arr2[0];
    if(len1 == 2)
        for(int i = 0; i < len2; i ++)
            if(onSeg(arr2[i],arr1[0],arr1[1]) <= 0)return true;
    if(len2 == 2)
        for(int i = 0; i < len1; i ++)
            if(onSeg(arr1[i],arr2[0],arr2[1]) <= 0)return true;
    return false;
}
int main() {
    int t;
    Point p1[3], p2[3];
    for(scanf("%d", &t); t --; ) {
        for(int i = 0; i < 3; i ++)p1[i].input();
        for(int i = 0; i < 3; i ++)p2[i].input();
        printf("%d\n", intersect(p1, p2));
    }
    return 0;
}*/
/*
//福州网络赛fzu1981  求弓箭手射穿敌人的个数（所有弓箭手都沿着vec方向射箭）
Point ps1[maxn], ps2[maxn];
int n, m;
int main() {
    int t;
    scanf("%d", &t);
    for(int index=1; index<=t; index++) {
        scanf("%d%d", &n, &m);
        for(int i = 0; i < n; i ++)ps1[i].input();
```

```
        for(int i = 0; i < m; i ++)ps2[i].input();
        Point vec;
        vec.input();

        for(int i = 0; i < n; i ++)ps1[i] = rotate(ps1[i],vec);
        for(int i = 0; i < m; i ++)ps2[i] = rotate(ps2[i],vec);

        sort(ps1, ps1+n);
        sort(ps2, ps2+m);

        int ans = 0;
        for(int i = 0; i < n; i ++) {
            int idx = upper_bound(ps2, ps2+m, ps1[i])-ps2-1;
            if(idx>=0 && idx<n) {
                if(sig(ps1[i].x-ps2[idx].x)==0 && sig(ps1[i].y-ps2[idx].y)==0) {
                    ans ++;
                }
            }
        }
        printf("Case %d: %d\n", index, ans);
    }
    return 0;
}
*/


/*
//hdu-2693 3D camera
//O发出三条射线OA、OB、OC，判断三角形DEF是否有一点绝对在O-ABC包围的范围之内

double cross2d(const Point & o, const Point & a, const Point & b) {
    return (a.x-o.x)*(b.y-o.y) - (b.x-o.x)*(a.y-o.y);
}
double dot2d(Point o, Point a, Point b) {
    return (a.x-o.x)*(b.x-o.x) + (a.y-o.y)*(b.y-o.y);
}
int lineCross(Point a, Point b, Point c, Point d, Point &p) {
    double s1, s2;
    s1=cross2d(a,b,c);
    s2=cross2d(a,b,d);
    if(sig(s1)==0 && sig(s2)==0)   return 2;
    if(sig(s2-s1)==0)   return 0;
    p.x = (c.x*s2-d.x*s1)/(s2-s1);
    p.y = (c.y*s2-d.y*s1)/(s2-s1);
    return 1;
}
double area(Point * p, int n) {
    double res = 0;
    p[n] = p[0];
    for(int i = 0; i < n; i ++) {
        res += p[i].x*p[i+1].y - p[i+1].x*p[i].y;
    }
    return res / 2;
}
void polygon_cut(Point * p, int & n, Point a, Point b) {
    static Point pp[maxn];
    int m = 0;
    p[n] = p[0];
    for(int i = 0; i < n; i ++) {
        if(sig(cross2d(a, b, p[i])) > 0)
            pp[m ++] = p[i];
        if(sig(cross2d(a, b, p[i])) != sig(cross2d(a, b, p[i+1])))
            lineCross(a, b, p[i], p[i+1], pp[m ++]);
```

```
    }
    n = 0;
    for(int i = 0; i < m; i ++)
        if(!i || !(pp[i]==pp[i-1]))
            p[n ++] = pp[i];
    while(n>1 && p[n-1]==p[0]) n --;
}

Point poly[maxn];
int polyN;

bool deal(Face surf, Point o, Point a, Point b, Point c) {
    Face oab(o,a,b);
    bool ok = true;
    if( px(oab,surf) ) {
        if(sig(c.z-o.z)==0) while(1);
        if(sig(o.z-surf.a.z)==0 || sig(c.z-o.z)>0) {
            ok = false;
        }
    } else {
        Point l1,l2;
        intersect(oab,surf,l1,l2);

        Point fa = cross(o,b,a);
        Point faL = l2-l1;

        faL = Point(-faL.y, faL.x, faL.z);

        if(sig(dot(faL,fa))<0) swap(l1,l2);
        polygon_cut(poly, polyN, l1, l2);  //l1 l2 的左侧!!!
    }
    return ok;
}

int main() {
    int t;
    for(scanf("%d", &t); t --; ) {
        Point o, a, b, c, d, e, f;

        o.input();
        a.input();  b.input();  c.input();
        d.input();  e.input();  f.input();

        Point vec = cross(d,e,f);
        if(sig(vec.len())==0)  while(1);

        o = rotate(o, vec);
        a=rotate(a,vec);   b=rotate(b,vec);   c=rotate(c,vec);
        d=rotate(d,vec);   e=rotate(e,vec);   f=rotate(f,vec);

        if(sig(o.z-d.z) < 0) {
            Point tmp(0,0,-1);
            o = rotate(o, tmp);
            a=rotate(a,tmp);   b=rotate(b,tmp);   c=rotate(c,tmp);
            d=rotate(d,tmp);   e=rotate(e,tmp);   f=rotate(f,tmp);
        }

        Face surf(d,e,f);       //水平面

        polyN = 3;
        poly[0] = d;
        poly[1] = e;
        poly[2] = f;
```

```
        Face oab(o,a,b);
        Face oac(o,a,c);
        Face obc(o,b,c);

        bool ok = true;

        ok &= deal(surf,o,a,b,c);
        ok &= deal(surf,o,b,c,a);
        ok &= deal(surf,o,c,a,b);
        if(ok) {
            if(sig(area(poly,polyN))==0) {
                ok = false;
            }
        }
        if(ok)  printf("YES\n");
        else    printf("NO\n");
    }
    return 0;
}
*/
```

计算几何_三维凸包（面积＋面数）

```
int faces;      //凸包的面数
int sig(double x) {    return (x > 1E-6) - (x < -1E-6);    }
#define N 505
struct Point {
   double x, y, z;
   Point() {}
   Point(double x, double y, double z) : x(x), y(y), z(z) {}
   Point operator +(Point b) {
      return Point(x + b.x, y + b.y, z + b.z);
   }
   Point operator -(Point b) {
      return Point(x - b.x, y - b.y, z - b.z);
   }
   Point operator /(double t) {
      return Point(x / t, y / t, z / t);
   }
   double len() {
      return sqrt(x * x + y * y + z * z);
   }
};
double dot(Point a, Point b){
   return a.x * b.x + a.y * b.y + a.z * b.z;
}
Point cross(Point a, Point b) {
   return Point(a.y * b.z - a.z * b.y,
             -(a.x * b.z - a.z * b.x),
               a.x * b.y - a.y * b.x);
}
Point ps[N];
struct Face {
   int a,b,c;
   Face(int a ,int b , int c ): a (a), b(b) , c(c) {}
   double area () {
      return cross( ps[b]-ps[a] , ps[c]-ps[a] ).len();
   }
   Point fa() const {
      return cross( ps[b]-ps[a] , ps[c]-ps[a] );
   }
   bool same_side(Point q , Point  p){
      return sig ( dot(ps[a] - q, cross(ps[b] - q, ps[c] - q))
```

```
            * dot(ps[a] - p , cross( ps[b] - p , ps[c] - p)) ) > 0 ;
    }
    bool inFace(Point q) const {    //判断某点是否在该平面上
        return sig(dot(ps[a] - q, cross(ps[b] - q, ps[c] - q)))==0;
    }
    bool operator == (const Face & face) const {
        Point fa1 = fa();
        Point fa2 = face.fa();
        if(sig(cross(fa1,fa2).len())!=0)    return false;
        return inFace(ps[face.a]);
    }
};
struct line {
    int a, b;
    line(int a, int b) : a(a),b(b){}
};
double convexHull(Point *ps, int n) {
#define judge(S, T)                                                 \
map[C[j].S][C[j].T]=map[C[j].T][C[j].S]= map[C[j].S][C[j].T]==0;\
LT.push_back(line(C[j].S, C[j].T))
    static bool map[N][N];
    static vector <Face> C , FT;    //convex、face_tmp
    static vector <line> LT;       //line_tmp
    int i, j;
    if(n <= 2)    return 0.0;
    if(n == 3)    return cross(ps[1]-ps[0] , ps[2]-ps[0]).len()*0.5;
    C.clear();
    memset(map, 0 , sizeof(map));
    for(i = 0; i < 4; i ++)
        C.push_back(Face(i, (i+1)%4, (i+2)%4));
    Point center = (ps[0] + ps[1] + ps[2] + ps[3]) / 4;
    for(i = 4 ; i < n ; i ++ ) {
        FT.clear();
        LT.clear();
        for (j = 0 ; j < C.size() ; j ++ )
            if ( ! (C[j].same_side( center , ps[i] )) ) {
                judge(a, b);    judge(c, b);    judge(c, a);
            } else    FT.push_back(C[j]);
        C.clear();
        for(j = 0 ;j < FT.size() ; j ++ )
            C.push_back( FT[j] );
        for(j = 0 ;j < LT.size() ; j ++ )
            if (map [ LT[j].a ][ LT[j].b ]) {
                C.push_back( Face ( LT[j].a , LT[j].b , i ) );
                map[ LT[j].a ][ LT[j].b ] = map[ LT[j].b ][ LT[j].a ] = 0;
            }
    }
    double area = 0 ;
    for ( i = 0 ; i < C.size() ; i ++ )
        area += C[i].area();
    area /= 2.0;
    //以下代码用于统计面数
    faces = 0;
    for(int i = 0; i < C.size(); i ++) {
        bool ok = true;
        for(int j = i+1; j < C.size(); j ++) {
            if(C[i]==C[j]) {
                ok = false;
                break;
            }
        }
        faces += ok;
    }
```

```
    return area;
}
int main() {
    int n;
    double x, y, z;
    while(scanf("%d", &n) != EOF) {
        for(int i = 0; i < n; i ++) {
            scanf("%lf%lf%lf", &x, &y, &z);
            ps[i] = Point(x, y, z);
        }
        while(sig(convexHull(ps, n)) == 0) {
            for(int j = n-1; j > 0; j --) {  //随机化，因为前四个必须不在一个平面上!
                swap(ps[j], ps[rand()%j]);
            }
        }
        printf("%d\n", faces);
    }
    return 0;
}
```

三角剖分

多边形与圆交面积

```
#define maxn 60
#define M_PI 3.14159265358979323846
#define sqr(v) ((v)*(v))
double my_acos(double d)    {    return acos(d>1?1:d<-1?-1:d);    }
double my_sqrt(double d)    {    return sqrt(max(d, 0.0));       }
const double eps = 1E-8;
int sig(double d) {
    return (d>eps) - (d<-eps);
}
struct Point {
    double x, y;
    Point(){}
    Point(double x, double y) : x(x), y(y) {}
    Point resize(double d) {
        d /= my_sqrt(x*x+y*y);
        return Point(x*d, y*d);
    }
    //左转90度
    Point left90() {
        return Point(-y, x);
    }
    Point operator - (const Point & o) const {
        return Point(this->x-o.x, this->y-o.y);
    }
    Point operator + (const Point & o) const {
        return Point(this->x+o.x, this->y+o.y);
    }
    Point operator * (double d) const {
        return Point(x*d, y*d);
    }
    void output() {
        printf("x = %.2f, y = %.2f\n", x, y);
    }
};
double cross(Point o, Point a, Point b) {
    return (a.x-o.x)*(b.y-o.y) - (b.x-o.x)*(a.y-o.y);
}
double dot(Point o, Point a, Point b) {
    return (a.x-o.x)*(b.x-o.x) + (a.y-o.y)*(b.y-o.y);
```

```
}
double dis(Point a, Point b) {
    return my_sqrt(sqr(a.x-b.x) + sqr(a.y-b.y));
}
int btw(Point o, Point a, Point b) {
    return sig(dot(o,a,b));
}
struct Circle {
    Point center;
    double radis;
};

//返回点o到直线ab的距离，res保存o在ab上的投影
double pointToLine(Point o, Point a, Point b, Point & res) {
    double d = dis(a, b);
    double s = cross(a, b, o) / d;
    res = o + (a-b).left90()*(s/d);
    return fabs(s);
}
//判断直线与圆相交
//1相交，0相切，-1不相交
int intersect(Point a, Point b, Circle c, Point &p1, Point &p2) {
    Point p;
    double d = pointToLine(c.center, a, b, p);
    int v = sig(c.radis-d);
    if(v != -1) {
        Point vec = (b-a).resize( my_sqrt(sqr(c.radis)-sqr(d)) );
        p1 = p+vec;
        p2 = p-vec;
    }
    return v;
}

//返回圆c和三角形oab相交的有向面积
double intersectArea(Circle c, Point a, Point b) {
    if(sig(cross(c.center,a,b))==0)return 0.0; //共线，三角形退化

    Point o = c.center, p[5];
    double r = c.radis;
    int len = 0;

    p[len++] = a;
    if(1 == intersect(a,b,c,p[1],p[2])) {  //正交
        if(btw(p[1],a,b)<0) p[len++]=p[1];
        if(btw(p[2],a,b)<0) p[len++]=p[2];
    }
    p[len++]=b;
    if(len==4 && btw(p[1],p[0],p[2])>0)    swap(p[1], p[2]);
    //init over!
    double res = 0;
    for(int i = 0; i < len-1; i ++) {
        if( sig(dis(o,p[i])-r)>0 || sig(dis(o,p[i+1])-r)>0 ) { //外部! 扇形
            double theta = my_acos( dot(o,p[i],p[i+1])/dis(o,p[i])/dis(o,p[i+1]) );
            res += theta * r * r / 2.0;
        } else {    //内部，三角形
            res += fabs(cross(o, p[i], p[i+1])/2.0);
        }
    }
    if(sig(cross(o,a,b))<0) res = -res; //有向面积
    return res;
}
//返回圆c与多边形ps相交面积（ps为任意简单多边形）
```

```
double intersectArea(Circle c, Point * ps, int n) {
    ps[n] = ps[0];
    double res = 0;
    for(int i = 0; i < n; i ++) {
        res += intersectArea(c,ps[i],ps[i+1]);
    }
    return fabs(res);
}

//poj-3675  任意多边形与圆的交面积
Point ps[maxn];
int n;
int main() {
    Circle c;
    c.center = Point(0,0);
    while(scanf("%lf%d", &c.radis, &n) != EOF) {
        for(int i = 0; i < n; i ++) {
            scanf("%lf%lf", &ps[i].x, &ps[i].y);
        }
        printf("%.2f\n", intersectArea(c, ps, n));
    }
    return 0;
}
```

多边形与多边形交面积

```
#define maxn 510
const double eps = 1E-8;
int sig(double d) {
    return (d>eps) - (d<-eps);
}
struct Point {
    double x, y;
    Point(){}
    Point(double x, double y) : x(x), y(y) {}
    bool operator == (const Point & p) const {
        return sig(x-p.x)==0 && sig(y-p.y)==0;
    }
};
double cross(Point o, Point a, Point b) {
    return (a.x-o.x)*(b.y-o.y) - (b.x-o.x)*(a.y-o.y);
}
double area(Point * ps, int n) {
    ps[n] = ps[0];
    double res = 0;
    for(int i = 0; i < n; i ++) {
        res += ps[i].x*ps[i+1].y - ps[i].y*ps[i+1].x;
    }
    return res / 2.0;
}
int lineCross(Point a, Point b, Point c, Point d, Point &p) {
    double s1, s2;
    s1=cross(a,b,c);
    s2=cross(a,b,d);
    if(sig(s1)==0 && sig(s2)==0)   return 2;
    if(sig(s2-s1)==0)   return 0;
    p.x = (c.x*s2-d.x*s1)/(s2-s1);
    p.y = (c.y*s2-d.y*s1)/(s2-s1);
    return 1;
}
//多边形切割
//用直线ab切割多边形p，切割后的在向量(a,b)的左侧，并原地保存切割结果
//如果退化为一个点，也会返回去，此时n为1
void polygon_cut(Point * p, int & n, Point a, Point b) {
```

```
    static Point pp[maxn];
    int m = 0;
    p[n] = p[0];
    for(int i = 0; i < n; i ++) {
        if(sig(cross(a, b, p[i])) > 0)
            pp[m ++] = p[i];
        if(sig(cross(a, b, p[i])) != sig(cross(a, b, p[i+1])))
            lineCross(a, b, p[i], p[i+1], pp[m ++]);
    }
    n = 0;
    for(int i = 0; i < m; i ++)
        if(!i || !(pp[i]==pp[i-1]))
            p[n ++] = pp[i];
    while(n>1 && p[n-1]==p[0]) n --;
}
//----------------------------华丽的分隔线-------------------------

//返回三角形oab和三角形ocd的有向交面积，o是原点
double intersectArea(Point a, Point b, Point c, Point d) {
    Point o(0,0);
    int s1 = sig(cross(o,a,b));
    int s2 = sig(cross(o,c,d));
    if(s1==0||s2==0)    return 0.0; //退化，面积为0

    if(s1==-1)  swap(a,b);
    if(s2==-1)  swap(c,d);

    Point p[10] = {o,a,b};
    int n = 3;

    polygon_cut(p,n,o,c);
    polygon_cut(p,n,c,d);
    polygon_cut(p,n,d,o);

    double res = fabs( area(p,n) );
    if(s1*s2==-1)   res=-res;
    return res;
}

//求两多边形的交面积
double intersectArea(Point * ps1, int n1, Point * ps2, int n2) {
    if(area(ps1,n1)<0)  reverse(ps1,ps1+n1);
    if(area(ps2,n2)<0)  reverse(ps2,ps2+n2);

    ps1[n1] = ps1[0];
    ps2[n2] = ps2[0];
    double res = 0;
    for(int i = 0; i < n1; i ++) {
        for(int j = 0; j < n2; j ++) {
            res += intersectArea(ps1[i],ps1[i+1],ps2[j],ps2[j+1]);
        }
    }
    return res; //assume res is positive !
}

//hdu-3060  求两个任意简单多边形的并面积
Point ps1[maxn], ps2[maxn];
int n1, n2;

int main() {
    while(scanf("%d%d", &n1, &n2) != EOF) {
        for(int i = 0; i < n1; i ++)    scanf("%lf%lf", &ps1[i].x, &ps1[i].y);
```

```
        for(int i = 0; i < n2; i ++)    scanf("%lf%lf", &ps2[i].x, &ps2[i].y);

        double ans = intersectArea(ps1, n1, ps2, n2);
        ans = fabs(area(ps1,n1))+fabs(area(ps2,n2)) - ans; //容斥
        printf("%.2f\n", ans);
    }
    return 0;
}
```

Delaunay 三角剖分

```
//Deaunay的对偶图是Voronoi，因此求Deaunay可以解决很多问题
//Deaunay三角形的外接圆，不包含其他点!
//最小生成树的边是Deaunay边的子集
//   【【【【【【【【【【下面是基本元素】】】】】】】】】】
#define maxn 110100
#define maxm 310001 //无向边
const double eps = 1e-8;
int sig(double d) {return (d>eps)-(d<-eps);}
struct Point {
    double x,y;
    struct Edge *e;
    bool operator < (const Point & p) const {
        return sig(x-p.x)!=0?x<p.x:sig(y-p.y)<0;
    }
    bool operator ==(const Point & p) const {
        return sig(x-p.x)==0&&sig(y-p.y)==0;
    }
};
double cross(Point & o, Point & a, Point & b) {
    return (a.x-o.x)*(b.y-o.y) - (b.x-o.x)*(a.y-o.y);
}
double dot(Point & o, Point & a, Point & b) {
    return (a.x-o.x)*(b.x-o.x) + (a.y-o.y)*(b.y-o.y);
}
struct Edge {
    Point *o, *d;
    Edge *on, *op, *dn, *dp;
};
//   【【【【【【【【【【下面开始Delaunay】】】】】】】】】】
#define Op(e,p)    ((e)->o==p?(e)->d:(e)->o)
#define Next(e,p)  ((e)->o==p?(e)->on:(e)->dn)
#define Prev(e,p)  ((e)->o==p?(e)->op:(e)->dp)
struct Delaunay {
    void solve(Point * ps, int n) {     //请保证调用以前ps已经sort和unique完毕!
        edge_num = 0;
        rubb = NULL;

        for(int i = 0; i < n; i ++)ps[i].e = NULL;
        Edge* l_cw, *r_ccw;
        divide(ps, 0, n, l_cw, r_ccw);
    }
    //以下是私有函数!
    Edge es[maxm], * rubb;
    int edge_num;
    Edge *make_edge(Point &u, Point &v) {
       Edge * e;
       if(rubb==NULL) {
          e = es + edge_num++;
       } else {
          e = rubb;
          rubb = rubb->dn;
       }
```

```
    e->on=e->op=e->dn=e->dp=e;
    e->o=&u;    e->d=&v;
    if (u.e==NULL) u.e=e;
    if (v.e==NULL) v.e=e;
    return e;
}
void delete_edge(Edge *e) {
    Point *u=e->o, *v=e->d;
    if (u->e==e) u->e=e->on;
    if (v->e==e) v->e=e->dn;
    Prev(e->on, u) = e->op;
    Next(e->op, u) = e->on;
    Prev(e->dn, v) = e->dp;
    Next(e->dp, v) = e->dn;
    e->dn = rubb;
    rubb = e;
}

void splice(Edge *a, Edge *b, Point *v) {
    Edge *n;
    n=Next(a, v);    Next(a, v)=b;
    Prev(n, v) = b;
    Next(b, v)=n;    Prev(b, v)=a;
}
Edge *join(Edge *a, Point *u, Edge *b, Point *v, int s) {
    Edge *e = make_edge(*u, *v);
    if (s == 0) {
        splice(Prev(a,u), e, u);
        splice(b, e, v);
    } else {
        splice(a, e, u);
        splice(Prev(b, v), e, v);
    }
    return e;
}

void lower_tangent(Edge * & l, Edge * & r, Point * & s, Point * & u){
    Point *dl=Op(l,s), *dr=Op(r,u);
    while(1) {
        if (sig(cross((*s),(*dl),(*u))) > 0) {
            l=Prev(l,dl);    s=dl;    dl=Op(l,s);
        } else if (sig(cross((*u),(*dr),(*s))) < 0) {
            r=Next(r,dr);    u=dr;    dr=Op(r,u);
        } else break;
    }
}

void merge(Edge *r_cw_l, Point *s, Edge *l_ccw_r, Point *u, Edge **l_tangent){
    Edge *b, *lc, *rc;
    Point *dlc, *drc;
    double crc, clc;

    lower_tangent(r_cw_l, l_ccw_r, s, u);
    b = join(r_cw_l, s, l_ccw_r, u, 1);

    *l_tangent = b;

    do{
        lc=Next(b,s); rc=Prev(b,u); dlc=Op(lc,s); drc=Op(rc,u);

        double cplc = cross(*dlc, *s, *u);
        double cprc = cross(*drc, *s, *u);
```

```
        bool alc = sig(cplc)>0, arc = sig(cprc)>0;
        if (!alc && !arc) break;
        if (alc){
            clc = dot(*dlc, *s, *u) / cplc;
            do{
                Edge * next = Next(lc, s);
                Point & dest = * Op(next, s);
                double cpn = cross(dest, *s, *u);
                if(sig(cpn)<=0)   break;

                double cn = dot(dest, *s, *u) / cpn;
                if (sig(cn-clc)>0) break;
                delete_edge(lc);
                lc = next;
                clc = cn;
            } while(1);
        }
        if (arc) {
            crc = (double)dot(*drc, *s, *u) / cprc;
            do{
                Edge * prev = Prev(rc, u);
                Point & dest = * Op(prev, u);
                double cpp = cross(dest, *s, *u);
                if(sig(cpp)<=0)   break;

                double cp = dot(dest, *s, *u) / cpp;
                if (sig(cp-crc) > 0) break;
                delete_edge(rc);
                rc = prev;
                crc = cp;
            } while (1);
        }
        dlc=Op(lc, s); drc=Op(rc, u);
        if (!alc || (alc && arc && sig(crc-clc)<0)){
            b = join(b, s, rc, drc, 1);
            u = drc;
        } else {
            b = join(lc, dlc, b, u, 1);
            s = dlc;
        }
    } while(1);
}

void divide(Point *p, int l, int r, Edge * & l_ccw, Edge * & r_cw) {
    int n=r-l;
    Edge *l_ccw_l, *r_cw_l, *l_ccw_r, *r_cw_r, *l_tangent, *c;
    if (n == 2) {
        l_ccw = r_cw = make_edge(p[l], p[l+1]);
    } else if (n == 3) {
        Edge * a = make_edge(p[l], p[l+1]), *b = make_edge(p[l+1], p[l+2]);
        splice(a,b,&p[l+1]);
        double c_p = cross(p[l], p[l+1], p[l+2]);
        if (sig(c_p)>0){ c=join(a,&p[l],b,&p[l+2],1); l_ccw=a; r_cw=b; }
        else if (sig(c_p)<0) {
                    c=join(a,&p[l],b,&p[l+2],0); l_ccw=c; r_cw=c;
                } else { l_ccw=a; r_cw=b; }
    } else if (n > 3) {
        int split=(l+r)/2;
        divide(p, l, split, l_ccw_l, r_cw_l);
        divide(p, split, r, l_ccw_r, r_cw_r);
        merge(r_cw_l,&p[split-1],l_ccw_r,&p[split],&l_tangent);
        if(l_tangent->o == &p[l]) l_ccw_l=l_tangent;
        if(l_tangent->d == &p[r-1]) r_cw_r=l_tangent;
```

```
            l_ccw=l_ccw_l; r_cw=r_cw_r;
        }
    }
} de;
//  【【【【【【【【下面开始MST】】】】】】】】】】
double dis(const Point & p1, const Point & p2) {
   return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
int p[maxn], r[maxn];
void make() {
    memset(r, 0, sizeof(r));
    memset(p, 255, sizeof(p));
}
int find(int x) {
    int px, i;
    for(px = x; p[px] != -1; px = p[px]);
    while(x != px) {
        i = p[x];
        p[x] = px;
        x = i;
    }
    return px;
}
int unio(int x, int y) {    //失败返回-1，否则返回新祖先
    x = find(x);    y = find(y);
    if(x == y)  return -1;
    if(r[x]>r[y]) {
        p[y]=x;
        return x;
    } else {
        p[x] = y;
        if(r[x] == r[y])    r[y] ++;
        return y;
    }
}
//以上是并查集
double V[maxm];
int A[maxm], B[maxm], R[maxm];
int cmp(const int & a, const int & b) {return V[a]<V[b];}
struct Kruskal {
    int n, len;

    void init(int n) {
        this->n = n;
        len = 0;
    }
    void addEdge(int a, int b, double v) {
        A[len] = a; B[len] = b; V[len] = v;
        len ++;
    }
    //返回最小生成树的权值和，并且[0,len)定为为最小生成树的边
    double solve() {
        for(int i = 0; i < len; i ++)  R[i] = i;
        sort(R, R+len, cmp);
        make();
        double res = 0;
        int j = 0;
        for(int i = 0; i < len && j<n-1; i ++) {
            if(unio(A[R[i]], B[R[i]]) != -1) {
                res += V[R[i]];
                R[j ++] = R[i];
            }
```

```
        }
        len = j;
        return res;
    }
} kr;
//以上是Kruskal
double MST(Point * ps, int n) {         //请保证调用以前ps已经sort和unique完毕!
    if(n <= 1)  return 0;
    de.solve(ps, n);
    kr.init(n);
    Edge *e_start, *e;
    Point *u, *v;
    for (int i = 0; i < n; i++) {
        u = &ps[i];
        e_start = e = u->e;
        do{
            v = Op(e, u);
            if (u < v)  kr.addEdge(u-ps, v-ps, dis(*u, *v));
        } while ((e=Next(e, u)) != e_start);
    }
    // enum edges over!
    return kr.solve();
}
//      【【【MST题目】】】
/*
//  hdu-1162        裸的欧几里得MST
Point ps[maxn];
int n;
int main() {
    int i;
    while(scanf("%d", &n) != EOF) {
        for (i = 0; i < n; i++) {
            scanf("%lf%lf", &ps[i].x, &ps[i].y);
        }
        sort(ps, ps+n);
        n=unique(ps, ps+n)-ps;
        printf("%.2f\n", MST(ps, n));
    }
    return 0;
}*/
//  POJ-3391        求最小生成树的第k大边
/*Point ps[maxn];
int n, k;
int main() {
    int t, x, y;
    for(scanf("%d", &t); t --; ) {
        scanf("%d", &k);
        n = 0;
        while(scanf("%d", &x), x!=-1.0) {
            scanf("%d", &y);
            ps[n].x = x;
            ps[n].y = y;
            n ++;
        }
        sort(ps, ps+n);
        n = unique(ps,ps+n)-ps;
        if (n <= 1 || n-k-1<0)
        {
            printf("0\n");
            continue;
        }
        MST(ps, n);
```

```
        printf("%d\n", int(ceil(V[R[n - k - 1]]) + 0.00000001));
    }
    return 0;
}*/
//  【【【【枚举三角形】】】】 and 【【【【求最大空圆】】】】
void enum_triangle(Point *ps, int n) {
    Edge *e_start, *e, *nxt;
    Point *u, *v, *w;
    for (int i = 0; i < n; i++) {
        u = &ps[i];
        e_start = e = u->e;
        do{
            v = Op(e, u);
            if (u < v) {
                nxt = Next(e, u);
                w = Op(nxt, u);
                if(u < w && Next(nxt,w)==Prev(e,v)) {
//  now, (u v w) is a triangle!!!!!!
//  这时，uvw的外接圆是空的（不包含ps中的其他点），如果要求最大空圆，则计算uvw的外接圆就可以！
                }
            }
        } while ((e=Next(e, u)) != e_start);
    }
}
```

AWT

Area_Path2d.Double_URAL1464

```
/**
    有关Path2D.Double的：
    Path2D.Double需要关注的函数：
        moveTo(x,y)移动到某个点
        lineTo(x,y)连线到某点
        closePath(x,y)关闭此条路线
        reset()清空
    moveTo和closePath呼应！
    Path2D.Double可以连续添加多个多边形，要注意绕向相同，在awt中瞬时针为正，逆时针为负。
    如果绕向不同，是异或运算，不过最好不要这么做，用Area.exclusiveOr比较安全！
*/
//给定一个多边形，和其中一点，求能被这个点照到的面积
//实质上是求每条边的阴影所形成的多边形的并，然后在用总面积减去这个并就可以了
//最终mle在 32 个点上，死的很壮烈，代码很有魅力。尤其是Path2D.Double。可以连续的构造n个多边形！
import static java.awt.geom.PathIterator.SEG_CLOSE;
import static java.awt.geom.PathIterator.SEG_LINETO;
import static java.awt.geom.PathIterator.SEG_MOVETO;
import java.awt.geom.Area;
import java.awt.geom.Path2D;
import java.awt.geom.PathIterator;
import java.awt.geom.Rectangle2D;
import java.util.Scanner;
public class Main {
    static Scanner scan = new Scanner(System.in);

    static int n;
    static double x0, y0;

    static double vec1X, vec1Y, vec2X, vec2Y;
    static Path2D.Float pd = new Path2D.Float();

    static double cross(double x0, double y0, double x1, double y1) {
        return x0*y1-x1*y0;
```

```
    }

    static void process(double x1, double y1, double x2, double y2) {
        vec1X = (x1-x0) * 2000000.0;
        vec1Y = (y1-y0) * 2000000.0;

        vec2X = (x2-x0)*2000000.0;
        vec2Y = (y2-y0)*2000000.0;

        pd.moveTo(x1, y1);

        double cr = cross(vec1X, vec1Y, vec2X, vec2Y);

        if(cr > 0) {
            pd.lineTo(x1+vec1X,   y1+vec1Y);
            pd.lineTo(x2+vec2X, y2+vec2Y);
            pd.lineTo(x2, y2);
        } else {
            pd.lineTo(x2, y2);
            pd.lineTo(x2+vec2X, y2+vec2Y);
            pd.lineTo(x1+vec1X,   y1+vec1Y);
        }
    }

    public static void main(String[] args) {
        x0 = scan.nextDouble();
        y0 = scan.nextDouble();
        n = scan.nextInt();

        double x0, y0, x1, y1, x2, y2;
        x0 = scan.nextDouble();
        y0 = scan.nextDouble();
        x1 = x0;
        y1 = y0;
        for(int i = 1; i < n; i ++) {
            x2 = scan.nextDouble();
            y2 = scan.nextDouble();
            process(x1, y1, x2, y2);
            x1 = x2;    y1 = y2;
//          if(i % 1000 == 0)   System.gc();
        }
//      System.gc();
        process(x1, y1, x0, y0);

        Area ans = new Area(new Rectangle2D.Double(0, 0, 1000, 1000));
        ans.subtract(new Area(pd));

        System.out.printf("%.2f\n", Math.abs(deal(ans)));
    }

    static double arr[] = new double[6];

    static double dealOne(PathIterator pi) {
        double lastX = 0, lastY = 0, area = 0, firstX = 0, firstY = 0;

        for(;; pi.next()) {
            int v = pi.currentSegment(arr);
            switch(v) {
                case SEG_MOVETO:
                    firstX = lastX = arr[0];
                    firstY = lastY = arr[1];
                    break;
                case SEG_LINETO:
```

```
                    area += lastX * arr[1] - lastY * arr[0];
                    lastX = arr[0];
                    lastY = arr[1];
                    break;
                case SEG_CLOSE:
                    area += lastX * firstY - lastY * firstX;
                    return area;
            }
        }
    }
    static double deal(Area a) {
        double ans = 0;
        PathIterator pi = a.getPathIterator(null);
        for(; !pi.isDone(); pi.next()) {
            double tmp = dealOne(pi);
            ans += tmp;
        }
        return ans / 2.0;
    }
}
/**
1.0 1.0
0 0
3 0
3 2
2 2
2 3
0 3
*/
```

area类(poj1688 NB AC)

```
import static java.awt.geom.PathIterator.SEG_CLOSE;
import static java.awt.geom.PathIterator.SEG_CUBICTO;
import static java.awt.geom.PathIterator.SEG_LINETO;
import static java.awt.geom.PathIterator.SEG_MOVETO;
import java.awt.geom.Area;
import java.awt.geom.Ellipse2D;
import java.awt.geom.PathIterator;
import java.util.Scanner;
public class Main {
    static Scanner scan = new Scanner(System.in);

    public static void main(String[] args) {
        for(int t = scan.nextInt(); t != 0; t --) {
            Area area = new Area();
            for(int n = scan.nextInt(); n != 0; n --) {
                int x = scan.nextInt(), y = scan.nextInt(), r = scan.nextInt();
                area.add(new Area(new Ellipse2D.Double(x-r, y-r, r*2, r*2)));
            }
            System.out.println(deal(area));
        }
    }

    static double dealOne(PathIterator pi) {
        double lastX = 0, lastY = 0, area = 0, firstX = 0, firstY = 0;
        double arr[] = new double[6];
        double a0,a1,a2,a3,b0,b1,b2,b3;
        for(;; pi.next()) {
            int v = pi.currentSegment(arr);
            switch(v) {
                case SEG_MOVETO:
                    firstX = lastX = arr[0];
                    firstY = lastY = arr[1];
```

```
                    break;
                case SEG_LINETO:
                    area += lastX * arr[1] - lastY * arr[0];
                    lastX = arr[0];
                    lastY = arr[1];
                    break;
                case SEG_CUBICTO:
                    a0 = lastX;
                    a1 = -3 * (lastX-arr[0]);
                    a2 = 3 * (lastX - 2*arr[0] + arr[2]);
                    a3 = -(lastX - 3*arr[0] + 3*arr[2] - arr[4]);

                    b0 = lastY;
                    b1 = -3 * (lastY-arr[1]);
                    b2 = 3 * (lastY - 2*arr[1] + arr[3]);
                    b3 = -(lastY - 3*arr[1] + 3*arr[3] - arr[5]);

                    area     +=     (a0*b1-b0*a1)     +     (a0*b2-b0*a2)     +
(3*a0*b3+a1*b2-a2*b1-3*b0*a3)/3+(2*a1*b3-2*a3*b1)/4+(a2*b3-a3*b2)/5;

                    lastX = arr[4];
                    lastY = arr[5];
                    break;
                case SEG_CLOSE:
                    area += lastX * firstY - lastY * firstX;
                    return area;
            }
        }
    }

    static int deal(Area a) {
        int ans = 0;
        PathIterator pi = a.getPathIterator(null);
        for(; !pi.isDone(); pi.next()) {
            double tmp = dealOne(pi);
            if(tmp > 1E-6)  ans ++;
        }
        return ans;
    }
}
```

最小正方形覆盖

```
//poj-3301
//最小正方形覆盖
//给定点集，输出最小覆盖正方形的面积

#define maxn 31
#define M_PI        3.14159265358979323846  /* pi */

int times = 100;

const double eps = 1E-12;
int sig(double d) {
    return (d>eps) - (d<-eps);
}
struct Point {
    double x, y;
    Point(double x, double y) : x(x), y(y) {}
    Point() {}
    Point rotate(double radian) {  //逆时针转
        double c = cos(radian), s = sin(radian);
        return Point(x*c-y*s, y*c+x*s);
```

```
    }
    void output() {
        if(sig(x)==0)   x = 0;
        if(sig(y)==0)   y = 0;
        printf("%.8f %.8f\n", x, y);
    }
};

Point ps[10010];
int n;

double minX, maxX, minY, maxY;

double get(double ang) {
    minX = 1E30, maxX = -1E30;
    minY = 1E30, maxY = -1E30;
    Point p;
    for(int i = 0; i < n; i ++) {
        p = ps[i].rotate(ang);
        minX = min(minX, p.x);
        minY = min(minY, p.y);
        maxX = max(maxX, p.x);
        maxY = max(maxY, p.y);
    }
    double res = max(maxX-minX,maxY-minY);
    return res*res;
}

void one(double jing) {
    double nowVal, nowAng, nxtVal, nxtAng;
    nowAng=0, nowVal=get(0);

    double tgtAng = 0;
    double tgtVal = nowVal;

    for(int i = 0; i <= times; i ++) {
        nxtAng = jing * i / times;
        nxtVal = get(nxtAng);
        if(nxtVal < tgtVal) {
            tgtVal = nxtVal;
            tgtAng = nowAng;
        }
        nowAng = nxtAng;
        nowVal = nxtVal;
    }
    for(int i = 0; i < n; i ++) {
        ps[i] = ps[i].rotate(tgtAng);
    }
}

int main() {
    int t;
    scanf("%d", &t);
    for(int idx=1; idx<=t; idx ++) {
        scanf("%d",&n);
        for(int i = 0; i < n; i ++) {
            scanf("%lf%lf", &ps[i].x, &ps[i].y);
        }
        for(double jing=M_PI/2.0; jing>1E-8; jing=jing/times*2.0) {
            //注意最后乘2.0，因为搜索两个区间
            one(jing);
        }
        printf("%.2f\n", get(0));
```

```
    }
    return 0;
}
```

最近圆对

```
/**
 * 最近圆对，hdu-3124 09武汉网赛，momodi的代码
 * 思想：二分+任意两圆是否碰撞（扫描线检测）
 *       扫描线是水平扫描线，然后tree中存储的顺序是竖直的，这样可以判定是否ok
 */

#define sqr(v) ((double)(v) * (v))
const int maxn = 101000;
const double eps = 1e-10;

int sig(double d) {
    return (d > eps) - (d < -eps);
}
int X[maxn], Y[maxn], R[maxn]; //原始圆
int n;                         //圆的个数
int LL[maxn], RR[maxn];        //按照左边界和右边界排序的圆下标，这两个是事件点
int tmp[maxn],num[maxn];       //tmp[i]表示纵坐标排序第i名的为tmp[i],num是tmp的反函数

set<int> tree;                 //树，保存信息
double mid;                    //当前搜索的点

bool collid(int a, int b) {    //排名为a的和排名为b的是否碰撞!
    a = tmp[a];
    b = tmp[b];
    if (sqr(X[a] - X[b]) + sqr(Y[a] - Y[b]) - sqr(R[a] + R[b] + mid + mid) <= 0) {
        return true;
    }
    return false;
}
bool insert(int v) {           //是否将排名为v的点插入成功
    set<int>::iterator it = tree.insert(v).first;
    if (it != tree.begin()) {
        if (collid(v, *--it)) {
            return false;
        }
        ++it;
    }
    if (++it != tree.end()) {
        if (collid(v, *it)) {
            return false;
        }
    }
    return true;
}
bool remove(int v) {           //是否将排名为v的点删除成功
    set<int>::iterator it = tree.find(v);
    if (it == tree.end()) {
        printf("error\n");
    }
    if (it != tree.begin() && it != --tree.end()) {
        int a = *--it;
        ++it;
        int b = *++it;
```

```
        if (collid(a, b)) {
            return false;
        }
    }
    tree.erase(v);
    return true;
}

bool ok(double mid) {           //而分的每一次判断是否可以!
    ::mid = mid;
    tree.clear();
    int l = 0, r = 0;
    while (l < n || r < n) {
        if (l == n) {
            if (!remove(num[RR[r++]])) {
                return false;
            }
        } else if (r == n) {
            if (!insert(num[LL[l++]])) {
                return false;
            }
        } else if (sig(X[LL[l]] - R[LL[l]] - mid - X[RR[r]] - R[RR[r]] - mid) <= 0){
            if (!insert(num[LL[l++]])) {
                return false;
            }
        } else {
            if (!remove(num[RR[r++]])) {
                return false;
            }
        }
    }
    return true;
}
bool cmpy(int a, int b) {           //按照y坐标关键字排序
    if (Y[a] == Y[b]) {
        return X[a] < X[b];
    }
    return Y[a] < Y[b];
}
bool cmpl(int a, int b) {           //进入圆时的事件点
    return X[a] - R[a] < X[b] - R[b];
}
bool cmpr(int a, int b) {           //离开圆时的事件点
    return X[a] + R[a] < X[b] + R[b];
}
double solve() {                    //求最近圆对
    for (int i = 0; i < n; ++i) {
        tmp[i] = LL[i] = RR[i] = i;
    }
    sort(tmp, tmp + n, cmpy);
    for (int i = 0; i < n; ++i) {
        num[tmp[i]] = i;
    }
    sort(LL, LL + n, cmpl);
    sort(RR, RR + n, cmpr);
    double s = 0, t = sqr(sqrt(sqr(X[0] - X[1]) + sqr(Y[0] - Y[1])) - R[0] - R[1]);
    while (t - s > 1e-8) {
        double mid = (s + t) / 2;
        if (ok(mid)) {
            s = mid;
        } else {
            t = mid;
```

```
        }
    }
    return s + t;
}
int main() {
    int ca;
    scanf("%d", &ca);
    while (ca--) {
        scanf("%d", &n);
        for (int i = 0; i < n; ++i) {
            scanf("%d %d %d", X + i, Y + i, R + i);
        }
        printf("%.6lf\n", solve());
    }
    return 0;
}
```

计算线段覆盖的网格整点数

```
int sig(int d) {
    return (d>0)-(d<0);
}
int gcd(int a, int b) {
    for(int t; t=b; b=a%b, a=t);
    return a;
}
struct Point {
    int x, y;
    bool operator < (const Point & p) const {
        return x!=p.x ? x<p.x : y<p.y;
    }
};
int cross(Point o, Point a, Point b) {
    return (int)(a.x-o.x)*(int)(b.y-o.y) - (int)(b.x-o.x)*(int)(a.y-o.y);
}
int dot(Point o, Point a, Point b) {
    return (int)(a.x-o.x)*(int)(b.x-o.x) + (int)(a.y-o.y)*(int)(b.y-o.y);
}
bool btw(Point o, Point a, Point b) {
    return dot(o, a, b) <= 0;
}
bool overLop(Point a, Point b, Point c, Point d) {
    return cross(a, c, d)==0 && cross(b, c, d)==0 &&
            (btw(a, c, d) || btw(b, c, d) || btw(c, a, b) || btw(d, a, b));
}
bool onSeg(Point o, Point a, Point b) {
    return cross(o,a,b)==0 && btw(o,a,b);
}
bool inte(Point a, Point b, Point c, Point d, Point & p) {
    if(onSeg(a, c, d)) return p=a, true;
    if(onSeg(b, c, d)) return p=b, true;
    if(onSeg(c, a, b)) return p=c, true;
    if(onSeg(d, a, b)) return p=d, true;

    int s1, s2, s3, s4;
    int d1, d2, d3, d4;
    d1 = sig(s1=cross(a,b,c));
    d2 = sig(s2=cross(a,b,d));
    d3 = sig(cross(c,d,a));
    d4 = sig(cross(c,d,b));
    if((d1^d2)==-2 && (d3^d4)==-2) {
        s3 = ((int)c.x*(int)s2-(int)d.x*(int)s1);
        s4 = ((int)c.y*(int)s2-(int)d.y*(int)s1);
```

```
        if(s3%(s2-s1) != 0 || s4%(s2-s1)!=0)   return false;   //非整数解!
        p.x = s3/(s2-s1);
        p.y = s4/(s2-s1);
        return 1;
    }
    return false;
}
int compute(Point *A, Point *B, int n) {
    static Point ps[1001000];
    int off = 0;
    for(int i = 0; i < n; i ++) {
        for(int j = off; j < i; j ++) {
            if(overLop(A[i], B[i], A[j], B[j])) {
                Point mi = min(min(A[i], B[i]), min(A[j], B[j]));
                Point ma = max(max(A[i], B[i]), max(A[j], B[j]));
                A[i] = mi;
                B[i] = ma;
                swap(A[off], A[j]);
                swap(B[off], B[j]);
                off ++;
            }
        }
    }
    int ans = 0;
    for(int i = off; i < n; i ++) {
        ans += 1 + abs(gcd(A[i].x-B[i].x, A[i].y-B[i].y));
    }
    int len = 0;
    for(int i = off; i < n; i ++) {
        for(int j = off; j < i; j ++) {
            if(inte(A[i],B[i], A[j], B[j],ps[len]))    len ++;
        }
    }
    sort(ps, ps+len);
    int j;
    for(int i = 0; i < len; i = j) {
        for(j = i+1; j<len && !(ps[i]<ps[j]); j ++);
        int x = j - i;
        n = (int)  round((1.0+sqrt(1.0+x*8.0))/2.0);
        ans -= n-1;
    }
    return ans;
}
/**
 * zoj-3400
    Input:
        6
        0 0 2 4
        0 0 4 2
        2 0 2 4
        3 0 0 3
        2 4 5 5
        4 2 5 5
    Output:
        11
 *
 */
Point A[1010], B[1010];
int n;
int main() {
    while(scanf("%d", &n) != EOF) {
        for(int i = 0; i < n; i ++)
            scanf("%d%d%d%d", &A[i].x, &A[i].y, &B[i].x, &B[i].y);
```

```
        int ans = compute(A, B, n);
        cout << ans << endl;
    }
    return 0;
}
```

计算几何_矩形内最远点_模拟退火

```
//ZOJ-2009
/**
    给定一个矩形，矩形内有n个点，求矩形内的一个点，使得该点离所给点的最短距离最大
    算法：模拟退火
*/
#define sqr(x) (x)*(x)
#define FOR(i,a,b) for(int i=a;i<b;i++)
#define FF(i,a) for(int i=0;i<a;i++)
const double eps=1e-3;
const double INF=1e20;
const double pi=acos(-1.0);
struct Houxuan
{
   double x,y,dist;
} pp[30000];
struct PT
{
   double x,y;
}hole[100000];
double dis(double x1,double y1,double x2,double y2)
{
   return sqrt(sqr(x1-x2)+sqr(y1-y2));
}
int n,cas;
double X,Y;
int main()
{
   int P=15,L=25;
   scanf("%d",&cas);
   srand((unsigned)time(NULL));
   while(cas--)
   {
    scanf("%lf%lf%d", &X, &Y, &n);
      if(n <= 0)   while(1);
      double delta=double(max(X,Y))/(sqrt(1.0*n));
      FF(i,n)
         scanf("%lf%lf",&hole[i].x,&hole[i].y);
      FF(i,P)
      {
         pp[i].x=double(rand()%1000+1)/1000.000*X;
         pp[i].y=double(rand()%1000+1)/1000.000*Y;
         pp[i].dist=INF;
         FF(j,n)

pp[i].dist=min(pp[i].dist,dis(pp[i].x,pp[i].y,hole[j].x,hole[j].y));
      }
      while(delta>eps)
      {
         FF(i,P)
         {
            Houxuan tmp=pp[i];
            FF(j,L)
            {
               double theta=double(rand()%1000+1)/1000.000*10*pi;
               Houxuan t;
               t.x=tmp.x+delta*cos(theta);
```

```
                t.y=tmp.y+delta*sin(theta);
                if(t.x>X||t.x<0||t.y>Y||t.y<0)
                    continue;
                t.dist=INF;
                FF(k,n)
                {
                    t.dist=min(t.dist,dis(t.x,t.y,hole[k].x,hole[k].y));
                }
                if(t.dist>pp[i].dist)
                    pp[i]=t;
            }
        }
        delta*=0.8;
    }
    int idx=0;
    FOR(i,1,P)
    {
        if(pp[i].dist>pp[idx].dist)
            idx=i;
    }
    printf("The safest point is (%.1f, %.1f).\n",pp[idx].x,pp[idx].y);
  }
  return 0;
}
```

计算几何_线段围成的面积最大多边形_ural1159

```
//给定n条线段长度，求围成的最大面积多边形
//算法是将求n条线段的外接圆
//注意这个多边形跨过圆心与否需要讨论！也就是优弧和劣弧的区别！
//eps取的是 1E-10
int n;
double len[110];
int needBigger(double r) {
    double sumAng = 0, myAng;
    for(int i = 1; i < n; i ++)
        sumAng += my_acos(1-len[i]*len[i]/(2*r*r));

    myAng = my_acos(1-len[0]*len[0]/(2*r*r));

    if(sig(sumAng-myAng)<0)             return true;    //劣弧上，下面是优弧上
    if(sig(sumAng-(2*M_PI-myAng)) < 0) return false;
    else                                return true;
}
double getArea(double r) {
    double sumAng = 0, myAng, sumArea = 0, myArea = 0;
    for(int i = 1; i < n; i ++) {
        double ang = my_acos(1-len[i]*len[i]/(2*r*r));
        sumAng += ang;
        sumArea += r*r*sin(ang)/2;
    }

    myAng = my_acos(1-len[0]*len[0]/(2*r*r));
    myArea = r*r*sin(myAng)/2;

    if(fabs(sumAng-myAng) - fabs(2*M_PI-myAng-sumAng) < 0)//劣弧上
        return sumArea-myArea;
    return sumArea+myArea;
}
/** ural_1159 fence
    input       4       10 5 5 4
    output      28.00
 */
```

```
int main() {
    while(scanf("%d", &n) != EOF) {
        int maxIdx = 0;
        double sum = 0, ans = 0.00;
        for(int i = 0; i < n; i ++) {
            scanf("%lf", len+i);
            if(len[i]>len[maxIdx])maxIdx = i;
            sum += len[i];
        }
        swap(len[maxIdx], len[0]); //let len[0] to be the maximum !

        if(sig(len[0]-(sum-len[0])) < 0) {
            double l = len[0]/2.0; //the minimum radius !
            double r = 1000000.0;  //enough

            while(r-l > 0.0000001) {   //great high precision is needed!
                double m = (l+r)/2;
                if(needBigger(m))   l = m;
                else                r = m;
            }
            ans = getArea((l+r)/2);
        }
        printf("%.2f\n", ans);
    }
    return 0;
}
```

矩形面积并(覆盖 k 次)

```
#define hash(x) ((x) & 131071)      //(x&(2^n-1))
int a1[140000];
double a2[140000];
void initHash() {
    memset(a1, 255, sizeof(a1));
}
int gen(int x) {
    int z = hash(x);
    while(a1[z]!=-1 && a1[z]!=x)    z = hash(z+1);
    if(a1[z] == -1) a1[z]=x, a2[z]=0;   //don't forget to init a2 !
    return z;
}
int get(int x) {
    int z = hash(x);
    while(a1[z]!=-1 && a1[z]!=x)    z=hash(z+1);
    if(a1[z]==-1)   return -1;
    return z;
}
//以上是hash表
#define maxn 20100
const double eps = 1E-6;
int sig(double d) {   return (d>eps) - (d<-eps);    }
double arr[maxn];                   //每个区间的刻度值
struct List {
#define pos(i, j)   ((j+1)*(num+10)+(i+1))  //表示第i块的小块们，覆盖了j次的长度
    int p[maxn], n, k;              //每个小快覆盖次数，小快个数，要统计覆盖k次(k>0)
    int block, q[210], num;         //每个大块的长度，每个大块的覆盖次数，块数
    double ans;                     //覆盖k次的总长度
    void init(int n, int k) {       //assume n>=2
        this->n = n;
        this->k = k;
        memset(p, 0, sizeof(p));
        memset(q, 0, sizeof(q));
        block = (int)sqrt((double)n);
```

```
        ans = 0;
        num = (n-1+block-1)/block;
        initHash();
        for(int i = 0; i < num-1; i ++)
         a2[ gen(pos(i,0)) ] = arr[(i+1)*block]-arr[i*block];
        a2[ gen(pos(num-1,0)) ] = arr[n-1]-arr[(num-1)*block];
    }
    void fill(int l, int r, int v) {
        int idx;
        double tmp;
        for(int j = l; j < r; j ++) {
            idx = j/block;
            tmp = arr[j+1] - arr[j];
            a2[ gen(pos(idx,p[j])) ] -= tmp; if(p[j]+q[idx] == k)    ans -= tmp;
            p[j] += v;
            a2[ gen(pos(idx,p[j])) ] += tmp;    if(p[j]+q[idx] == k)    ans += tmp;
        }
    }
    void insert(double l, double r, int v) {
        int lw = lower_bound(arr, arr+n, l-eps)-arr;    //减去eps比较安全!
        int rw = lower_bound(arr, arr+n, r-eps)-arr;

        int L = (lw+block-1)/block, R = rw/block;
        if(R-L>0) {
         int j;
           for(int i = L; i < R; i ++) {
             if(k-q[i]>=0 && (j=get(pos(i, k-q[i])))!=-1)   ans-=a2[j];
               q[i] += v;
               if(k-q[i]>=0 && (j=get(pos(i, k-q[i])))!=-1) ans+=a2[j];
           }
           fill(R*block, rw, v);
           fill(lw, L*block, v);
        } else {
           fill(lw, rw, v);
        }
    }
} lst;
struct Nod {
    double x, y1, y2;
    int mode;           //进入 1，退出 2
    void init(double x, double y1, double y2, int mode) {
        this->x = x;
        this->y1= y1;
        this->y2= y2;
        this->mode=mode;
    }
    bool operator < (const Nod & nod) const {
     int val = sig(x-nod.x);
     return val==0 ? mode>nod.mode : val<0; //进入优先!
    }
};
bool cmpEq(double a, double b) {   return sig(a-b)==0; }
double compute(Nod * nods, int n, int k) { //n传入的是挡板的个数，不是矩形个数!
    if(n == 0)  return 0;
    int len = 0;
    for(int i = 0; i < n; i += 2) {         //assume奇偶是一样的y1 和y2
        arr[len ++] = nods[i].y1;
        arr[len ++] = nods[i].y2;
    }
    sort(arr, arr+len);
    len = unique(arr, arr+len, cmpEq) - arr;
    lst.init(len, k);
```

```
    sort(nods, nods+n);
    double ans = 0, lastX = nods[0].x;
    for(int i = 0; i < n; i ++) {
        ans += lst.ans*(nods[i].x-lastX);
        lst.insert(nods[i].y1, nods[i].y2, nods[i].mode);
        lastX = nods[i].x;
    }
    return ans;
}
int n, k;
Nod nods[maxn];
/**
 * hdu-3621
 * Area k(2010 天津网赛)
 */
int main() {
   for(int idx = 1; scanf("%d",&n), n; idx ++) {
       int x, y, z, l;
       int newN = 0;
       for(int i = 0; i < n; i ++) {
        scanf("%d%d%d%d", &x, &y, &z, &l);
          if(2*z<=l) {
              nods[newN ++].init(x-l/2.0, y-l/2.0, y+l/2.0, 1);
              nods[newN ++].init(x+l/2.0, y-l/2.0, y+l/2.0, -1);
          }
       }
       scanf("%d", &k);
       double ans = compute(nods, newN, k);
       printf("Case %d: %.3f\n", idx, ans);
   }
   return 0;
}
```

点集_正方形个数

```
/**
 * poj-3432
 * 给的点集构成的正方形个数（点不重合）
 * 效率：n^2*lgn
 */

typedef pair<int,int> T;

T ts[2010];
int n;

int main() {
    while(scanf("%d", &n) != EOF) {
        for(int i = 0; i < n; i ++)scanf("%d%d", &ts[i].first, &ts[i].second);
        sort(ts, ts+n);
        int ans = 0;
        for(int i = 0; i < n; i ++) {
            for(int j = i+1; j < n; j ++) {//枚举对角线
                int dx = (ts[j].first - ts[i].first);
                int dy = (ts[j].second-ts[i].second);

                int mx = (ts[j].first+ts[i].first);
                int my = (ts[j].second+ts[i].second);

                if((mx-dy)&1 || (my+dx)&1) continue;

                if(binary_search(ts, ts+n, T((mx-dy)/2, (my+dx)/2)) &&
```

```
                    binary_search(ts, ts+n, T((mx+dy)/2, (my-dx)/2)))   ans ++;
                }
            }
            printf("%d\n", ans/2);
        }
        return 0;
}
```

高维立方体_空间切割求容积

```
int jc[20]; //阶乘值，一开始应该先初始化好

//n维空间的立方体，平行于各个坐标轴，对定点坐标为(0,0,...0)和(x0,x1,x2..xn-1)
//求x0+x1+x2+...+xn-1<=u截该立方体的容积
//思想是容斥原理
double get(double * x, int n, double u) {
    double res = 0;
    for(int i = 0; i < (1<<n); i ++) {
        double tmp = 0;
        int num = 0;
        for(int j = 0; j < n; j ++)
            if((i>>j)&1)    tmp += x[j], num ++;
        tmp = u - tmp;
        if(tmp >= 0) {
            tmp = pow(tmp, n);
            if(num & 1) res -= tmp;
            else        res += tmp;
        }
    }
    return res / jc[n];
}

double xs[10];

/**
 * spoj-2002
 * n维立方体，平行于各个坐标轴，以原点为中心，每个坐标轴的取值范围为[-xi,xi]
 * 计算a <= x0+x1+...+xn-1 <= b的容积 所占总容积的百分比
 * 我首先将立方体平移到原点为立方体的顶点，在第一象限内讨论
 */
int main() {
    jc[0] = 1;
    for(int i = 1; i <= 20; i ++)  jc[i] = jc[i-1]*i;  //init jc
    int t, n, a, b;
    for(scanf("%d", &t); t --; ) {
        scanf("%d%d%d", &n, &a, &b);
        double sum = 0, all = 1;
        for(int i = 0; i < n; i ++) {
            scanf("%lf", xs+i);
            sum += xs[i];
            xs[i] *= 2;
            all *= xs[i];
        }
        printf("%.9f\n", (get(xs, n, b+sum) - get(xs, n, a+sum))/all);
    }
    return 0;
}
```

两条线段储水量

```
//poj-2826  An easy problem?
//两条线段相交，问最多可以接住多少雨水。貌似不复杂，不过得想清楚哪些情况下是肯定接不住的：线段不相交，线段重合，一条平行于 x 轴等等多种情况。
```

```
struct Point {
    double x, y;
};
int sig(double d) {
    if(fabs(d) < 1E-8)  return 0;
    return d < 0 ? -1 : 1;
}
double cross(Point &o, Point &a, Point &b) {
    return (a.x - o.x)*(b.y - o.y)-(b.x - o.x)*(a.y - o.y);
}
double dot(Point &o, Point &a, Point &b) {
    return (a.x-o.x)*(b.x-o.x) + (a.y-o.y)*(b.y-o.y);
}
int btw(Point &x, Point &a, Point &b) {
    return sig(dot(x, a, b));
}
int segCross(Point a, Point b, Point c, Point d, Point &p) {
    double s1, s2;
    int d1, d2, d3, d4;
    d1 = sig(s1=cross(a,b,c));
    d2 = sig(s2=cross(a,b,d));
    d3 = sig(cross(c,d,a));
    d4 = sig(cross(c,d,b));
    if((d1^d2)==-2 && (d3^d4)==-2) {
        p.x = (c.x*s2-d.x*s1)/(s2-s1);
        p.y = (c.y*s2-d.y*s1)/(s2-s1);
        return 1;
    }
    if( d1==0 && btw(c,a,b)<=0 ||
        d2==0 && btw(d,a,b)<=0 ||
        d3==0 && btw(a,c,d)<=0 ||
        d4==0 && btw(b,c,d)<=0) {
        if(d1==0 && d2==0)  return 0;

        if(d1==0 && btw(c,a,b)<=0)  p = c;
        if(d2==0 && btw(d,a,b)<=0)  p = d;
        if(d3==0 && btw(a,c,d)<=0)  p = a;
        if(d4==0 && btw(b,c,d)<=0)  p = b;

        return 2;
    }
    return -1;
}

Point a, b, c, d;

double compute() {
    if(sig(a.y-b.y)==0 || sig(c.y-d.y)==0) return 0.0;
    Point p;
    int x = segCross(a, b, c, d, p);
    if(x <= 0)  return 0.0;

    Point m, n;
    if(a.y > b.y) {
        m = a;
    } else {
        m = b;
    }
    if(c.y > d.y) {
        n = c;
    } else {
        n = d;
    }
```

```
    if(sig(m.y-n.y) < 0) {
        Point t;
        t = m;
        m = n;
        n = t;
    }
    double y = m.y < n.y ? m.y : n.y;
    if(sig(p.y-y) >= 0) return 0.0;

    if(sig(m.x-n.x)>=0 && sig(n.x-p.x)>=0 && cross(p, m, n)<0) {
        return 0.0;
    }
    if(sig(m.x-n.x)<=0 && sig(n.x-p.x)<=0 && cross(p, m, n)>0) {
        return 0.0;
    }


    Point u, v;
    u.x = -10010.0;
    v.x = 10010.0;
    u.y = v.y = y;

    Point mm, nn;

    if((x=segCross(u, v, p, m, mm)) <= 0)  while(1);
    if((x=segCross(u, v, p, n, nn)) <= 0)  while(1);
    return fabs((y-p.y) * (mm.x-nn.x) / 2.0);
}

int main() {
    int t;
    for(cin >> t; t --; ) {
        cin >> a.x >> a.y >> b.x >> b.y >> c.x >> c.y >> d.x >> d.y;
        double ans = compute();
        if(sig(ans)==0) ans = 0;
        printf("%.2f\n", ans);
    }
    return 0;
}
```

曼哈顿最远点

```
//求曼哈顿距离最远点，效率：n*2^(k-1) (k是维数)
//POJ-2926

struct Point {
    double x[5];
    void input() {
        for(int i = 0; i < 5; i ++) {
            scanf("%lf", x+i);
        }
    }
    double calDis(int mask) {
        double res = x[4];
        for(int i = 0; i < 4; i ++) {
            if((mask>>i) & 1)   res += x[i];
            else                res -= x[i];
        }
        return res;
    }
};

Point ps[100010];
int n;
```

```
int main() {
    scanf("%d", &n);
    for(int i = 0; i < n; i ++)ps[i].input();
    double ans = -1.0;
    for(int mask = 0; mask < 16; mask ++) {
        double maxDis = -100000000000000.0, minDis = 100000000000000.0;
        for(int i = 0; i < n; i ++) {
            double tmp = ps[i].calDis(mask);
            maxDis = max(maxDis, tmp);
            minDis = min(minDis, tmp);
        }
        ans = max(ans, maxDis - minDis);
    }
    printf("%.2f\n", ans);
    return 0;
}
```

Poj-2043 Area of Polygons

求整点多边形覆盖的网格数目

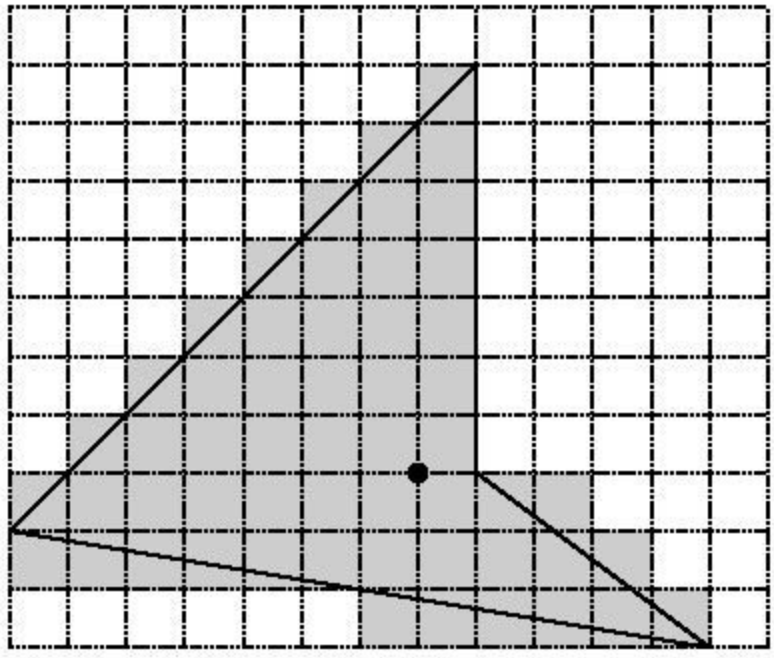

Figure 1: A polygon and unit squares intersecting it

```
int X[110], Y[110];
int n;

const int inf = 0x3f3f3f3f;
int area;

typedef pair<double, double> T;
T ts[110];

bool vis[5010];

void deal(int x) {
    int len = 0;
    for(int i = 0; i < n; i ++) {
```

```
        int xa = X[i], ya = Y[i];
        int xb = X[i+1], yb = Y[i+1];
        if(xa > xb) swap(xa, xb), swap(ya, yb);
        if(xa > x || xb < x+1)  continue;
        ts[len].first = ((double)(x-xa)*(yb-ya))/(xb-xa)+ya;
        ts[len].second =((double)(x+1-xa)*(yb-ya))/(xb-xa)+ya;
        len ++;
    }
    sort(ts, ts+len);
    memset(vis, 0, sizeof(vis));
    for(int i = 0; i < len; i += 2) {
        if(ts[i].first > ts[i].second) swap(ts[i].first, ts[i].second);
        if(ts[i+1].first > ts[i+1].second) swap(ts[i+1].first, ts[i+1].second);
        int newYA = floor(ts[i].first);
        int newYD = ceil(ts[i+1].second);

        for(int y = newYA; y < newYD; y ++) {
            vis[y + 2500] = true;
        }
    }
    for(int i = 0; i <5010; i ++) {
        if(vis[i])  area ++;
    }
}

int main() {
    while(cin >> n, n) {
        int minX = inf;
        int maxX = -inf;
        for(int i = 0; i < n; i ++){
            cin >> X[i] >> Y[i];
            minX = min(minX, X[i]);
            maxX = max(maxX, X[i]);
        }
        X[n] = X[0];    Y[n] = Y[0];

        area = 0;
        for(int i = minX; i < maxX; i ++) {
            deal(i);
        }
        cout << area << endl;
    }
    return 0;
}
```