计算几何

1、基本函数

1.1 Point 定义

```
const double eps = 1e-8;
const double PI = acos(-1.0);
int sgn(double x)
{
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
struct Point
{
    double x,y;
    Point(){}
    Point(double _x,double _y)
    {
        x = _x;y = _y;
    }
    Point operator -(const Point &b)const
    {
        return Point(x - b.x,y - b.y);
    }
    //叉积
    double operator ^(const Point &b)const
    {
        return x*b.y - y*b.x;
    }
    //点积
    double operator *(const Point &b)const
    {
        return x*b.x + y*b.y;
    }
    //绕原点旋转角度 B（弧度值），后 x,y 的变化
    void transXY(double B)
    {
        double tx = x,ty = y;
        x = tx*cos(B) - ty*sin(B);
        y = tx*sin(B) + ty*cos(B);
    }
};
```

1.2 Line 定义

```
struct Line
{
    Point s,e;
    Line(){}
    Line(Point _s,Point _e)
    {
        s = _s;e = _e;
    }
    //两直线相交求交点
    //第一个值为 0 表示直线重合，为 1 表示平行，为 0 表示相交,为 2 是相交
```

```
    //只有第一个值为 2 时，交点才有意义
    pair<int,Point> operator &(const Line &b)const
    {
        Point res = s;
        if(sgn((s-e)^(b.s-b.e)) == 0)
        {
            if(sgn((s-b.e)^(b.s-b.e)) == 0)
                return make_pair(0,res);//重合
            else return make_pair(1,res);//平行
        }
        double t = ((s-b.s)^(b.s-b.e))/((s-e)^(b.s-b.e));
        res.x += (e.x-s.x)*t;
        res.y += (e.y-s.y)*t;
        return make_pair(2,res);
    }
};
```

1.3 两点间距离

```
//*两点间距离
double dist(Point a,Point b)
{
    return sqrt((a-b)*(a-b));
}
```

1.4 判断：线段相交

```
//*判断线段相交
bool inter(Line l1,Line l2)
{
    return
    max(l1.s.x,l1.e.x) >= min(l2.s.x,l2.e.x) &&
    max(l2.s.x,l2.e.x) >= min(l1.s.x,l1.e.x) &&
    max(l1.s.y,l1.e.y) >= min(l2.s.y,l2.e.y) &&
    max(l2.s.y,l2.e.y) >= min(l1.s.y,l1.e.y) &&
    sgn((l2.s-l1.e)^(l1.s-l1.e))*sgn((l2.e-l1.e)^(l1.s-l1.e)) <= 0 &&
    sgn((l1.s-l2.e)^(l2.s-l2.e))*sgn((l1.e-l2.e)^(l2.s-l2.e)) <= 0;
}
```

1.5 判断：直线和线段相交

```
//判断直线和线段相交
bool Seg_inter_line(Line l1,Line l2) //判断直线 l1 和线段 l2 是否相交
{
    return sgn((l2.s-l1.e)^(l1.s-l1.e))*sgn((l2.e-l1.e)^(l1.s-l1.e)) <= 0;
}
```

1.6 点到直线距离

```
//点到直线距离
//返回为 result,是点到直线最近的点
Point PointToLine(Point P,Line L)
{
    Point result;
    double t = ((P-L.s)*(L.e-L.s))/((L.e-L.s)*(L.e-L.s));
    result.x = L.s.x + (L.e.x-L.s.x)*t;
    result.y = L.s.y + (L.e.y-L.s.y)*t;
    return result;
}
```

1.7 点到线段距离

```
//点到线段的距离
//返回点到线段最近的点
```

```
Point NearestPointToLineSeg(Point P,Line L)
{
    Point result;
    double t = ((P-L.s)*(L.e-L.s))/((L.e-L.s)*(L.e-L.s));
    if(t >= 0 && t <= 1)
    {
        result.x = L.s.x + (L.e.x - L.s.x)*t;
        result.y = L.s.y + (L.e.y - L.s.y)*t;
    }
    else
    {
        if(dist(P,L.s) < dist(P,L.e))
            result = L.s;
        else result = L.e;
    }
    return result;
}
```

1.8 计算多边形面积

```
//计算多边形面积
//点的编号从 0~n-1
double CalcArea(Point p[],int n)
{
    double res = 0;
    for(int i = 0;i < n;i++)
        res += (p[i]^p[(i+1)%n])/2;
    return fabs(res);
}
```

1.9 判断点在线段上

```
//*判断点在线段上
bool OnSeg(Point P,Line L)
{
    return
    sgn((L.s-P)^(L.e-P)) == 0 &&
    sgn((P.x - L.s.x) * (P.x - L.e.x)) <= 0 &&
    sgn((P.y - L.s.y) * (P.y - L.e.y)) <= 0;
}
```

1.10 判断点在凸多边形内

```
//*判断点在凸多边形内
//点形成一个凸包，而且按逆时针排序（如果是顺时针把里面的<0 改为>0）
//点的编号:0~n-1
//返回值:
//-1:点在凸多边形外
//0:点在凸多边形边界上
//1:点在凸多边形内
int inConvexPoly(Point a,Point p[],int n)
{
    for(int i = 0;i < n;i++)
    {
        if(sgn((p[i]-a)^(p[(i+1)%n]-a)) < 0)return -1;
        else if(OnSeg(a,Line(p[i],p[(i+1)%n])))return 0;
    }
    return 1;
}
```

1.11 判断点在任意多边形内

```
//*判断点在任意多边形内
//射线法，poly[]的顶点数要大于等于 3,点的编号 0~n-1
//返回值
//-1:点在凸多边形外
```

```
//0:点在凸多边形边界上
//1:点在凸多边形内
int inPoly(Point p,Point poly[],int n)
{
    int cnt;
    Line ray,side;
    cnt = 0;
    ray.s = p;
    ray.e.y = p.y;
    ray.e.x = -100000000000.0;//-INF,注意取值防止越界
    for(int i = 0;i < n;i++)
    {
        side.s = poly[i];
        side.e = poly[(i+1)%n];

        if(OnSeg(p,side))return 0;

        //如果平行轴则不考虑
        if(sgn(side.s.y - side.e.y) == 0)
            continue;

        if(OnSeg(side.s,ray))
        {
            if(sgn(side.s.y - side.e.y) > 0)cnt++;
        }
        else if(OnSeg(side.e,ray))
        {
            if(sgn(side.e.y - side.s.y) > 0)cnt++;
        }
        else if(inter(ray,side))
            cnt++;
    }
    if(cnt % 2 == 1)return 1;
    else return -1;
}
```

1.12 判断凸多边形

```
//判断凸多边形
//允许共线边
//点可以是顺时针给出也可以是逆时针给出
//点的编号 1~n-1
bool isconvex(Point poly[],int n)
{
    bool s[3];
    memset(s,false,sizeof(s));
    for(int i = 0;i < n;i++)
    {
        s[sgn( (poly[(i+1)%n]-poly[i])^(poly[(i+2)%n]-poly[i]) )+1] = true;
        if(s[0] && s[2])return false;
    }
    return true;
}
```

2、凸包

```
/*
 * 求凸包，Graham 算法
 * 点的编号 0~n-1
 * 返回凸包结果 Stack[0~top-1]为凸包的编号
 */
const int MAXN = 1010;
Point list[MAXN];
int Stack[MAXN],top;
//相对于 list[0]的极角排序
bool _cmp(Point p1,Point p2)
{
    double tmp = (p1-list[0])^(p2-list[0]);
    if(sgn(tmp) > 0)return true;
    else if(sgn(tmp) == 0 && sgn(dist(p1,list[0]) - dist(p2,list[0])) <= 0)
        return true;
    else return false;
}
void Graham(int n)
{
    Point p0;
    int k = 0;
    p0 = list[0];
    //找最下边的一个点
    for(int i = 1;i < n;i++)
    {
        if( (p0.y > list[i].y) || (p0.y == list[i].y && p0.x > list[i].x) )
        {
            p0 = list[i];
            k = i;
        }
    }
    swap(list[k],list[0]);
    sort(list+1,list+n,_cmp);
    if(n == 1)
    {
        top = 1;
        Stack[0] = 0;
        return;
    }
    if(n == 2)
    {
        top = 2;
        Stack[0] = 0;
        Stack[1] = 1;
        return ;
    }
    Stack[0] = 0;
    Stack[1] = 1;
    top = 2;
    for(int i = 2;i < n;i++)
    {
        while(top > 1 &&
sgn((list[Stack[top-1]]-list[Stack[top-2]])^(list[i]-list[Stack[top-2]])) <=
0)
```

```
            top--;
        Stack[top++] = i;
    }
}
```

3、平面最近点对（HDU 1007）

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <iostream>
#include <math.h>
using namespace std;
const double eps = 1e-6;
const int MAXN = 100010;
const double INF = 1e20;
struct Point
{
    double x,y;
};
double dist(Point a,Point b)
{
    return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
Point p[MAXN];
Point tmpt[MAXN];
bool cmpxy(Point a,Point b)
{
    if(a.x != b.x)return a.x < b.x;
    else return a.y < b.y;
}
bool cmpy(Point a,Point b)
{
    return a.y < b.y;
}
double Closest_Pair(int left,int right)
{
    double d = INF;
    if(left == right)return d;
    if(left + 1 == right)
        return dist(p[left],p[right]);
    int mid = (left+right)/2;
    double d1 = Closest_Pair(left,mid);
    double d2 = Closest_Pair(mid+1,right);
    d = min(d1,d2);
    int k = 0;
    for(int i = left;i <= right;i++)
    {
        if(fabs(p[mid].x - p[i].x) <= d)
            tmpt[k++] = p[i];
    }
    sort(tmpt,tmpt+k,cmpy);
    for(int i = 0;i <k;i++)
    {
        for(int j = i+1;j < k && tmpt[j].y - tmpt[i].y < d;j++)
        {
```

```
            d = min(d,dist(tmpt[i],tmpt[j]));
        }
    }
    return d;
}
int main()
{
    int n;
    while(scanf("%d",&n)==1 && n)
    {
        for(int i = 0;i < n;i++)
            scanf("%lf%lf",&p[i].x,&p[i].y);
        sort(p,p+n,cmpxy);
        printf("%.2lf\n",Closest_Pair(0,n-1)/2);
    }
    return 0;
}
```

4、旋转卡壳

4.1 求解平面最远点对（POJ 2187 Beauty Contest）

```
struct Point
{
    int x,y;
    Point(int _x = 0,int _y = 0)
    {
        x = _x; y = _y;
    }
    Point operator -(const Point &b)const
    {
        return Point(x - b.x, y - b.y);
    }
    int operator ^(const Point &b)const
    {
        return x*b.y - y*b.x;
    }
    int operator *(const Point &b)const
    {
        return x*b.x + y*b.y;
    }
    void input()
    {
        scanf("%d%d",&x,&y);
    }
};
//距离的平方
int dist2(Point a,Point b)
{
    return (a-b)*(a-b);
}
//******二维凸包，int***********
const int MAXN = 50010;
Point list[MAXN];
int Stack[MAXN],top;
bool _cmp(Point p1,Point p2)
{
```

```
    int tmp = (p1-list[0])^(p2-list[0]);
    if(tmp > 0)return true;
    else if(tmp == 0 && dist2(p1,list[0]) <= dist2(p2,list[0]))
        return true;
    else return false;
}
void Graham(int n)
{
    Point p0;
    int k = 0;
    p0 = list[0];
    for(int i = 1;i < n;i++)
        if(p0.y > list[i].y || (p0.y == list[i].y && p0.x > list[i].x))
        {
            p0 = list[i];
            k = i;
        }
    swap(list[k],list[0]);
    sort(list+1,list+n,_cmp);
    if(n == 1)
    {
        top = 1;
        Stack[0] = 0;
        return;
    }
    if(n == 2)
    {
        top = 2;
        Stack[0] = 0; Stack[1] = 1;
        return;
    }
    Stack[0] = 0; Stack[1] = 1;
    top = 2;
    for(int i = 2;i < n;i++)
    {
        while(top > 1 &&
((list[Stack[top-1]]-list[Stack[top-2]])^(list[i]-list[Stack[top-2]])) <= 0)
            top--;
        Stack[top++] = i;
    }
}
//旋转卡壳，求两点间距离平方的最大值
int rotating_calipers(Point p[],int n)
{
    int ans = 0;
    Point v;
    int cur = 1;
    for(int i = 0;i < n;i++)
    {
        v = p[i]-p[(i+1)%n];
        while((v^(p[(cur+1)%n]-p[cur])) < 0)
            cur = (cur+1)%n;
        ans = max(ans,max(dist2(p[i],p[cur]),dist2(p[(i+1)%n],p[(cur+1)%n])));
    }
    return ans;
}
Point p[MAXN];
int main()
```

```
{
    int n;
    while(scanf("%d",&n) == 1)
    {
        for(int i = 0;i < n;i++)list[i].input();
        Graham(n);
        for(int i = 0;i < top;i++)p[i] = list[Stack[i]];
        printf("%d\n",rotating_calipers(p,top));
    }
    return 0;
}
```

4.2 求解平面点集最大三角形

```
//旋转卡壳计算平面点集最大三角形面积
int rotating_calipers(Point p[],int n)
{
    int ans = 0;
    Point v;
    for(int i = 0;i < n;i++)
    {
        int j = (i+1)%n;
        int k = (j+1)%n;
        while(j != i && k != i)
        {
            ans = max(ans,abs((p[j]-p[i])^(p[k]-p[i])));
            while( ((p[i]-p[j])^(p[(k+1)%n]-p[k])) < 0 )
                k = (k+1)%n;
            j = (j+1)%n;
        }
    }
    return ans;
}
Point p[MAXN];
int main()
{
    int n;
    while(scanf("%d",&n) == 1)
    {
        if(n == -1)break;
        for(int i = 0;i < n;i++)list[i].input();
        Graham(n);
        for(int i = 0;i < top;i++)p[i] = list[Stack[i]];
        printf("%.2f\n",(double)rotating_calipers(p,top)/2);
    }
    return 0;
}
```

4.3 求解两凸包最小距离（POJ 3608）

```
const double eps = 1e-8;
int sgn(double x)
{
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
struct Point
{
    double x,y;
    Point(double _x = 0,double _y = 0)
```

```
    {
        x = _x; y = _y;
    }
    Point operator -(const Point &b)const
    {
        return Point(x - b.x, y - b.y);
    }
    double operator ^(const Point &b)const
    {
        return x*b.y - y*b.x;
    }
    double operator *(const Point &b)const
    {
        return x*b.x + y*b.y;
    }
    void input()
    {
        scanf("%lf%lf",&x,&y);
    }
};
struct Line
{
    Point s,e;
    Line(){}
    Line(Point _s,Point _e)
    {
        s = _s; e = _e;
    }
};
//两点间距离
double dist(Point a,Point b)
{
    return sqrt((a-b)*(a-b));
}
//点到线段的距离，返回点到线段最近的点
Point NearestPointToLineSeg(Point P,Line L)
{
    Point result;
    double t = ((P-L.s)*(L.e-L.s))/((L.e-L.s)*(L.e-L.s));
    if(t >=0 && t <= 1)
    {
        result.x = L.s.x + (L.e.x - L.s.x)*t;
        result.y = L.s.y + (L.e.y - L.s.y)*t;
    }
    else
    {
        if(dist(P,L.s) < dist(P,L.e))
            result = L.s;
        else result = L.e;
    }
    return result;
}

/*
 * 求凸包，Graham 算法
 * 点的编号 0~n-1
 * 返回凸包结果 Stack[0~top-1]为凸包的编号
 */
```

```
const int MAXN = 10010;
Point list[MAXN];
int Stack[MAXN],top;
//相对于 list[0]的极角排序
bool _cmp(Point p1,Point p2)
{
    double tmp = (p1-list[0])^(p2-list[0]);
    if(sgn(tmp) > 0)return true;
    else if(sgn(tmp) == 0 && sgn(dist(p1,list[0]) - dist(p2,list[0])) <= 0)
        return true;
    else return false;
}
void Graham(int n)
{
    Point p0;
    int k = 0;
    p0 = list[0];
    //找最下边的一个点
    for(int i = 1;i < n;i++)
    {
        if( (p0.y > list[i].y) || (p0.y == list[i].y && p0.x > list[i].x) )
        {
            p0 = list[i];
            k = i;
        }
    }
    swap(list[k],list[0]);
    sort(list+1,list+n,_cmp);
    if(n == 1)
    {
        top = 1;
        Stack[0] = 0;
        return;
    }
    if(n == 2)
    {
        top = 2;
        Stack[0] = 0;
        Stack[1] = 1;
        return ;
    }
    Stack[0] = 0;
    Stack[1] = 1;
    top = 2;
    for(int i = 2;i < n;i++)
    {
        while(top > 1 &&
sgn((list[Stack[top-1]]-list[Stack[top-2]])^(list[i]-list[Stack[top-2]])) <=
0)
            top--;
        Stack[top++] = i;
    }
}
//点 p0 到线段 p1p2 的距离
double pointtoseg(Point p0,Point p1,Point p2)
{
   return dist(p0,NearestPointToLineSeg(p0,Line(p1,p2)));
}
```

```
//平行线段 p0p1 和 p2p3 的距离
double dispallseg(Point p0,Point p1,Point p2,Point p3)
{
    double ans1 = min(pointtoseg(p0,p2,p3),pointtoseg(p1,p2,p3));
    double ans2 = min(pointtoseg(p2,p0,p1),pointtoseg(p3,p0,p1));
    return min(ans1,ans2);
}
//得到向量 a1a2 和 b1b2 的位置关系
double Get_angle(Point a1,Point a2,Point b1,Point b2)
{
    return (a2-a1)^(b1-b2);
}
double rotating_calipers(Point p[],int np,Point q[],int nq)
{
    int sp = 0, sq = 0;
    for(int i = 0;i < np;i++)
        if(sgn(p[i].y - p[sp].y) < 0)
            sp = i;
    for(int i = 0;i < nq;i++)
        if(sgn(q[i].y - q[sq].y) > 0)
            sq = i;
    double tmp;
    double ans = dist(p[sp],q[sq]);
    for(int i = 0;i < np;i++)
    {
        while(sgn(tmp = Get_angle(p[sp],p[(sp+1)%np],q[sq],q[(sq+1)%nq])) < 0)
            sq = (sq+1)%nq;
        if(sgn(tmp) == 0)
            ans = min(ans,dispallseg(p[sp],p[(sp+1)%np],q[sq],q[(sq+1)%nq]));
        else ans = min(ans,pointtoseg(q[sq],p[sp],p[(sp+1)%np]));
        sp = (sp+1)%np;
    }
    return ans;
}
double solve(Point p[],int n,Point q[],int m)
{
    return min(rotating_calipers(p,n,q,m),rotating_calipers(q,m,p,n));
}
Point p[MAXN],q[MAXN];
int main()
{
    int n,m;
    while(scanf("%d%d",&n,&m) == 2)
    {
        if(n == 0 && m == 0)break;
        for(int i = 0;i < n;i++)
            list[i].input();
        Graham(n);
        for(int i = 0;i < top;i++)
            p[i] = list[i];
        n = top;
        for(int i = 0;i < m;i++)
            list[i].input();
        Graham(m);
        for(int i = 0;i < top;i++)
            q[i] = list[i];
        m = top;
        printf("%.4f\n",solve(p,n,q,m));
```

```
    }
    return 0;
}
```

5、半平面交

5.1 半平面交模板(from UESTC)

```
const double eps = 1e-8;
const double PI = acos(-1.0);
int sgn(double x)
{
    if(fabs(x) < eps) return 0;
    if(x < 0) return -1;
    else return 1;
}
struct Point
{
    double x,y;
    Point(){}
    Point(double _x,double _y)
    {
        x = _x; y = _y;
    }
    Point operator -(const Point &b)const
    {
        return Point(x - b.x, y - b.y);
    }
    double operator ^(const Point &b)const
    {
        return x*b.y - y*b.x;
    }
    double operator *(const Point &b)const
    {
        return x*b.x + y*b.y;
    }
};
struct Line
{
    Point s,e;
    double k;
    Line(){}
    Line(Point _s,Point _e)
    {
        s = _s; e = _e;
        k = atan2(e.y - s.y,e.x - s.x);
    }
    Point operator &(const Line &b)const
    {
        Point res = s;
        double t = ((s - b.s)^(b.s - b.e))/((s - e)^(b.s - b.e));
        res.x += (e.x - s.x)*t;
        res.y += (e.y - s.y)*t;
        return res;
    }
};
```

```
//半平面交，直线的左边代表有效区域
bool HPIcmp(Line a,Line b)
{
    if(fabs(a.k - b.k) > eps)return a.k < b.k;
    return ((a.s - b.s)^(b.e - b.s)) < 0;
}
Line Q[110];
void HPI(Line line[], int n, Point res[], int &resn)
{
    int tot = n;
    sort(line,line+n,HPIcmp);
    tot = 1;
    for(int i = 1;i < n;i++)
        if(fabs(line[i].k - line[i-1].k) > eps)
            line[tot++] = line[i];
    int head = 0, tail = 1;
    Q[0] = line[0];
    Q[1] = line[1];
    resn = 0;
    for(int i = 2; i < tot; i++)
    {
        if(fabs((Q[tail].e-Q[tail].s)^(Q[tail-1].e-Q[tail-1].s)) < eps ||
fabs((Q[head].e-Q[head].s)^(Q[head+1].e-Q[head+1].s)) < eps)
            return;
        while(head < tail && (((Q[tail]&Q[tail-1]) -
line[i].s)^(line[i].e-line[i].s)) > eps)
            tail--;
        while(head < tail && (((Q[head]&Q[head+1]) -
line[i].s)^(line[i].e-line[i].s)) > eps)
            head++; Q[++tail]
        = line[i];
    }
    while(head < tail && (((Q[tail]&Q[tail-1]) -
Q[head].s)^(Q[head].e-Q[head].s)) > eps)
        tail--;
    while(head < tail && (((Q[head]&Q[head-1]) -
Q[tail].s)^(Q[tail].e-Q[tail].e)) > eps)
        head++;
    if(tail <= head + 1)return;
    for(int i = head; i < tail; i++)
        res[resn++] = Q[i]&Q[i+1];
    if(head < tail - 1)
        res[resn++] = Q[head]&Q[tail];
}
```

5.2 普通半平面交写法

```
POJ 1750
const double eps = 1e-18;
int sgn(double x)
{
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
struct Point
{
    double x,y;
    Point(){}
```

```
    Point(double _x,double _y)
    {
        x = _x; y = _y;
    }
    Point operator -(const Point &b)const
    {
        return Point(x - b.x, y - b.y);
    }
    double operator ^(const Point &b)const
    {
        return x*b.y - y*b.x;
    }
    double operator *(const Point &b)const
    {
        return x*b.x + y*b.y;
    }
};
//计算多边形面积
double CalcArea(Point p[],int n)
{
    double res = 0;
    for(int i = 0;i < n;i++)
        res += (p[i]^p[(i+1)%n]);
    return fabs(res/2);
}
//通过两点，确定直线方程
void Get_equation(Point p1,Point p2,double &a,double &b,double &c)
{
    a = p2.y - p1.y;
    b = p1.x - p2.x;
    c = p2.x*p1.y - p1.x*p2.y;
}
//求交点
Point Intersection(Point p1,Point p2,double a,double b,double c)
{
    double u = fabs(a*p1.x + b*p1.y + c);
    double v = fabs(a*p2.x + b*p2.y + c);
    Point t;
    t.x = (p1.x*v + p2.x*u)/(u+v);
    t.y = (p1.y*v + p2.y*u)/(u+v);
    return t;
}
Point tp[110];
void Cut(double a,double b,double c,Point p[],int &cnt)
{
    int tmp = 0;
    for(int i = 1;i <= cnt;i++)
    {
        //当前点在左侧，逆时针的点
        if(a*p[i].x + b*p[i].y + c < eps)tp[++tmp] = p[i];
        else
        {
            if(a*p[i-1].x + b*p[i-1].y + c < -eps)
                tp[++tmp] = Intersection(p[i-1],p[i],a,b,c);
            if(a*p[i+1].x + b*p[i+1].y + c < -eps) tp[++tmp]
                = Intersection(p[i],p[i+1],a,b,c);
        }
    }
```

```
    for(int i = 1;i <= tmp;i++)
        p[i] = tp[i];
    p[0] = p[tmp];
    p[tmp+1] = p[1];
    cnt = tmp;
}
double V[110],U[110],W[110];
int n;
const double INF = 100000000000.0;
Point p[110];
bool solve(int id)
{
    p[1] = Point(0,0);
    p[2] = Point(INF,0);
    p[3] = Point(INF,INF);
    p[4] = Point(0,INF);
    p[0] = p[4];
    p[5] = p[1];
    int cnt = 4;
    for(int i = 0;i < n;i++)
        if(i != id)
        {
            double a = (V[i] - V[id])/(V[i]*V[id]);
            double b = (U[i] - U[id])/(U[i]*U[id]);
            double c = (W[i] - W[id])/(W[i]*W[id]);
            if(sgn(a) == 0 && sgn(b) == 0)
            {
                if(sgn(c) >= 0)return false;
                else continue;
            }
            Cut(a,b,c,p,cnt);
        }
    if(sgn(CalcArea(p,cnt)) == 0)return false;
    else return true;
}
int main()
{
   while(scanf("%d",&n) == 1)
    {
        for(int i = 0;i < n;i++)
            scanf("%lf%lf%lf",&V[i],&U[i],&W[i]);
        for(int i = 0;i < n;i++)
        {
            if(solve(i))printf("Yes\n");
            else printf("No\n");
        }
    }
   return 0;
}
```

6、三点求圆心坐标（三角形外心）

```
//过三点求圆心坐标
Point waixin(Point a,Point b,Point c)
{
    double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1*a1 + b1*b1)/2;
```

```
    double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2*a2 + b2*b2)/2;
    double d = a1*b2 - a2*b1;
    return Point(a.x + (c1*b2 - c2*b1)/d, a.y + (a1*c2 -a2*c1)/d);
}
```

7、求两圆相交的面积

```
//两个圆的公共部分面积
double Area_of_overlap(Point c1,double r1,Point c2,double r2)
{
    double d = dist(c1,c2);
    if(r1 + r2 < d + eps)return 0;
    if(d < fabs(r1 - r2) + eps)
    {
        double r = min(r1,r2);
        return PI*r*r;
    }
    double x = (d*d + r1*r1 - r2*r2)/(2*d);
    double t1 = acos(x / r1);
    double t2 = acos((d - x)/r2);
    return r1*r1*t1 + r2*r2*t2 - d*r1*sin(t1);
}
```

8、Pick 公式

顶点坐标均是整点的简单多边形：面积=内部格点数目+边上格点数目/2-1