**BUAA\_OhMyGold**

# Geometry

目录

[Math 0](#_Toc308012449)

[几何类 2](#_Toc308012450)

[基础函数 2](#_Toc308012451)

[点、线、向量 2](#_Toc308012452)

[两点距离 2](#_Toc308012453)

[判三点共线 3](#_Toc308012454)

[共线三点位置关系 3](#_Toc308012455)

[点到直线距离 3](#_Toc308012456)

[判点在线段内 3](#_Toc308012457)

[点到线段距离 3](#_Toc308012458)

[点关于直线的对称点 4](#_Toc308012459)

[判断线段相交 4](#_Toc308012460)

[线段求交 4](#_Toc308012461)

[两不相交线段最近距离 5](#_Toc308012462)

[两直线位置关系（交点） 6](#_Toc308012463)

[两不平行直线交点 6](#_Toc308012464)

[两向量夹角 6](#_Toc308012465)

[平面最近点距 6](#_Toc308012466)

[平面图形 7](#_Toc308012467)

[三角形 7](#_Toc308012468)

[四边形 10](#_Toc308012469)

[多边形 12](#_Toc308012470)

[圆 15](#_Toc308012471)

[三维几何 18](#_Toc308012472)

[体积 18](#_Toc308012473)

[三维凸包 18](#_Toc308012474)

[最小包围球 21](#_Toc308012475)

[旋转变换 24](#_Toc308012476)

[其它成题： 24](#_Toc308012477)

[N边形费马点(退火) 24](#_Toc308012478)

[简单多边形与圆的公共面积 26](#_Toc308012479)

[面积最大的三角形 29](#_Toc308012480)

[判定多边形是否对称 30](#_Toc308012481)

[贴海报 32](#_Toc308012482)

[矩形面积并 34](#_Toc308012483)

[半平面交求多边形的核 37](#_Toc308012484)

[半平面交求多条直线所围区域面积 38](#_Toc308012485)

[平面图欧拉定理 41](#_Toc308012486)

[光线遇圆反射 43](#_Toc308012487)

[其它几何模板 45](#_Toc308012488)

[一些可能用到的性质 56](#_Toc308012489)

# 几何类

## 基础函数

点：

struct Point

{

double x,y;//int

};

符号函数

int sig(double a)

{

if (fabs(a) <= EPS) return 0;

if (a>0) return 1;

else return -1;

}

//叉积

//原始叉积

int ocross(int x1,int y1,int x2,int y2)

{

return x1\*y2-x2\*y1;

}

//计算cross product (P1-P0)x(P2-P0)

double cross(Point p0,Point p1,Point p2)

{

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

double

cross(double x0,double y0,double x1,double y1,double x2,double y2)

{

return (x1-x0)\*(y2-y0)-(x2-x0)\*(y1-y0);

}

//点积，计算dot product (P1-P0).(P2-P0)

double dot(Point p0,Point p1,Point p2)

{

return (p1.x-p0.x)\*(p2.x-p0.x)+(p1.y-p0.y)\*(p2.y-p0.y);

}

double

dot(double x1,double y1,double x2,double y2,double x0,double y0)

{

return (x1-x0)\*(x2-x0)+(y1-y0)\*(y2-y0);

}

注意反三角函数：数据在正负1附近时有可能取到NAN引起精度问题。

## 点、线、向量

### 两点距离

double dis(struct Point a, struct Point b)

{

return sqrt((a.x - b.x)\*(a.x - b.x) + (a.y - b.y)\*(a.y - b.y));

}

### 判三点共线

int dots\_inline(Point p1,Point p2,Point p3)

{

return fabs(cross(p1,p2,p3)) < EPS;

}

int dots\_inline(double x1,double y1,double x2,double y2,double x3,double y3)

{

return fabs(cross(x1,y1,x2,y2,x3,y3)) < EPS;

}

### 共线三点位置关系

//共线三点a,b,c。

//判a点是不是在线段bc上，>0不在，=0与端点重合，<0在。

int point\_on\_line(Point a,Point b,Point c)

{

return sig(dot(a,b,c));

}

### 点到直线距离

double cross(Point p0,Point p1,Point p2)

{

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

double disptoline(Point p ,Point l1,Point l2)

{

return fabs(cross(p,l1,l2))/dis(l1,l2);

}

### 判点在线段内

//包括端点

int dot\_online\_in(Point p, Point l1,Point l2)

{

return (cross(p,l1,l2) < EPS) && (l1.x-p.x)\*(l2.x-p.x) < EPS &&(l1.y-p.y)\*(l2.y-p.y) < EPS;

}

### 点到线段距离

double distoline(struct Point p,struct Point a,struct Point b)

{

struct Point t;

t = p;

t.x += a.y - b.y;

t.y += b.x - a.x;

if (cross(p,a,t) \* cross(p,b,t) > EPS)

return dis(p,a) < dis(p,b) ? dis(p,a) : dis(p,b);

return fabs(cross(k,a,b))/dis(a,b);

}

### 点关于直线的对称点

//p点关于直线ab的对称点,直线方程为Ax+By+C=0,可直接传ABC

Point symmetricalPointofLine(Point p, Point a,Point b)

{

Point p2;

double A,B,C,d;

A = a.y - b.y;

B = b.x - a.x;

C = (a.x \* b.y - b.x \* a.y);

d = A \* A + B \* B;

p2.x = (B \* B \* p.x - A \* A \* p.x - 2 \* A \* B \* p.y - 2 \* A \* C) / d;

p2.y = (A \* A \* p.y - B \* B \* p.y - 2 \* A \* B \* p.x - 2 \* B \* C) / d;

return p2;

}

### 判断线段相交

//整型。浮点判EPS

bool seg\_cross(Point a, Point b, Point c, Point d)

{

return (cross(a,c,d)\*cross(b,c,d) < 0 && cross(c,a,b)\*cross(d,a,b) < 0);

}

### 线段求交

//求ab是否与cd相交，交点为p。

//返回值1规范相交，0交点是一线段的端点，-1不相交。

//！注意查1234有没打错！要考虑到共线但没有公共点的情况。

/\*double 类型\*/

int ab\_cross\_cd(Point a,Point b,Point c,Point d)

{

double s1,s2,s3,s4;

int d1,d2,d3,d4;

Point p;

d1 = sig(s1=cross(c,a,b));

d2 = sig(s2=cross(d,a,b));

d3 = sig(s3=cross(a,c,d));

d4 = sig(s4=cross(b,c,d));

if ((d1^d2) == -2 && (d3^d4) == -2)

{//规范相交

p.x=(c.x\*s2 - d.x\*s1)/(s2 - s1);

p.y=(c.y\*s2 - d.y\*s1)/(s2 - s1);

return 1;

}

//相交于某端点

if (d1==0 && point\_on\_line(c,a,b)<=0)

{

p = c;

return 0;

}

if (d2==0 && point\_on\_line(d,a,b)<=0)

{

p = d;

return 0;

}

if (d3==0 && point\_on\_line(a,c,d)<=0)

{

p = a;

return 0;

}

if (d4==0 && point\_on\_line(b,c,d)<=0)

{

p = b;

return 0;

}

return -1;

}

/\*int型\*/

int ab\_cross\_cd(Point a,Point b,Point c,Point d) //求ab是否与cd相交，交点为p。1规范相交，0交点是一线段的端点，-1不相交。

{

int d1,d2,d3,d4;

Point p;

d1 = cross(c,a,b);

d2 = cross(d,a,b);

d3 = cross(a,c,d);

d4 = cross(b,c,d);

if (d1\*d2 < 0 && d3\*d4 < 0)

{

p.x=(c.x\*s2-d.x\*s1)/(s2-s1);

p.y=(c.y\*s2-d.y\*s1)/(s2-s1);

return 1;

}

//如果不规范相交

if (d1 == 0 && point\_on\_line(c,a,b) <= 0)

{

p = c;

return 0;

}

if (d2 == 0 && point\_on\_line(d,a,b) <= 0)

{

p = d;

return 0;

}

if (d3 == 0 && point\_on\_line(a,c,d) <= 0)

{

p = a;

return 0;

}

if (d4 == 0 && point\_on\_line(b,c,d) <= 0)

{

p = b;

return 0;

}

return -1;

}

### 两不相交线段最近距离

double disseg(struct Point a,struct Point b,struct Point c,struct Point d)

{

return min(min(distoline(c,d,a), distoline(c,d,b)),

min(distoline(a,b,c), distoline(a,b,d)));

}

### 两直线位置关系（交点）

//求直线ab与cd关系，返回值0重合，-1平行，相交1且交点为p。

int ab\_cross\_cd2(Point a,Point b,Point c,Point d)

{

int d1,d2,d3,d4;

d1 = cross(c,a,b);

d2 = cross(d,a,b);

d3 = cross(a,c,d);

d4 = cross(b,c,d);

if (ocross(a.x - b.x,a.y - b.y,c.x - d.x,c.y - d.y) == 0)

{

if (d1 == 0)//重合

return 0;

else return -1;//平行

}

else

{//规范相交则求交点

px = (c.x\*d2 - d.x\*d1)\*1.0 / (d2-d1);

py = (c.y\*d2 - d.y\*d1)\*1.0 / (d2-d1);

return 1;

}

}

### 两不平行直线交点

Point intersection(Point a,Point b,Point c,Point d)

{

Point ret = a;

double t = ((a.x - c.x)\*(c.y - d.y) - (a.y - c.y)\*(c.x - d.x))

/((a.x - b.x)\*(c.y - d.y) - (a.y - b.y)\*(c.x - d.x));

ret.x += (b.x - a.x)\*t;

ret.y += (b.y - a.y)\*t;

return ret;

}

### 两向量夹角

//余弦定理

double cala(struct Point a,struct Point b)

{

return acos((a.x \* b.x + a.y \* b.y)/

((sqrt(a.x\*a.x + a.y\*a.y)) \* sqrt(b.x\*b.x + b.y\*b.y)) );

}

### 平面最近点距

/\*

使用：

ans = INF;

sort(v + 1,v + n + 1,cmp);

ans = cal(1,n);

\*/

Point v[100005];int n;//点集

double ans;

int cmp(const struct Point &s,const struct Point &t) //sort函数

{

if((t.x - s.x) >= EPS)

return 1;

else if(abs(s.x - t.x) < EPS)

{

if((t.y - s.y) >= EPS)

return 1;

else return 0;

}else return 0;

}

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 cal(int l,int r)

{

if(l == r)

return INF;

if(l + 1 == r)

return dis(l,r);

if(l + 2 == r)

return min(dis(l,r),min(dis(l,l + 1),dis(l + 1,r)));

int mid = (l + r)/2;

ans = min(ans,min(cal(l,mid),cal(mid + 1,r)));

for(int i = mid; i >= l; --i)

{

if((abs(a[i].x - a[mid].x) - ans) > EPS)

break;

for(int j = mid + 1; j <= mid + 5; ++j)//只计算后5个点即可

{

ans = min(ans,dis(i,j));

}

}

return ans;

}

## 平面图形

### 三角形

#### 三角形面积

//输入三顶点

double area\_triangle(Point p1,Point p2,Point p3)

{

return fabs(cross(p1,p2,p3))/2;

}

double area\_triangle(double x1,double y1,double x2,double y2,double x3,double y3)

{

return fabs(cross(x1,y1,x2,y2,x3,y3))/2;

}

//输入三边长

double area\_triangle(double a,double b,double c)

{

double s = (a + b + c)/2;

return sqrt(s\*(s - a)\*(s - b)\*(s - c));

}

#### 三角形外接圆半径

double triangle\_r(Point a,Point b,Point c)

{

double s = area\_triangle(a,b,c);

return fabs(dis(a,b)\*dis(a,c)\*dis(b,c)/(4\*s));

}

#### 三角形外心坐标

/\*

需要函数：两不平行直线交点

Point intersection(Point a,Point b,Point c,Point d)

\*/

//给三顶点坐标，求外心坐标

Point circumcenter(Point a,Point b,Point c)

{

Point ua,ub,va,vb;

ua.x = (a.x + b.x)/2;

ua.y = (a.y + b.y)/2;

ub.x = ua.x - a.y + b.y;

ub.y = ua.y + a.x - b.x;

va.x = (a.x+c.x)/2;

va.y = (a.y+c.y)/2;

vb.x = va.x-a.y+c.y;

vb.y = va.y+a.x-c.x;

return intersection(ua,ub,va,vb);

}

#### 三角形重心

//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

/\*

需要函数：两不平行直线交点

Point intersection(Point a,Point b,Point c,Point d)

\*/

Point barycenter(Point a,Point b,Point c)

{

Point ua,ub,va,vb;

ua.x = (a.x+b.x)/2;

ua.y = (a.y+b.y)/2;

ub = c;

va.x = (a.x+c.x)/2;

va.y = (a.y+c.y)/2;

vb = b;

return intersection(ua,ub,va,vb);

}

#### 三角形重心的等角共扼点

//到三边距离平方和最小的点

/\*先求原三角型的重心：m.x=(x1+x2+x3)/3.0; m.y=(y1+y2+y3)/3.0;

再求重心关于三角形三边的三个对称点：t1,t2,t3;

再求由t1，t2,t3组成的三角形的外心即可\*/

/\*

需要函数

//两不平行直线交点

Point intersection(Point a,Point b,Point c,Point d)

//三角形外心坐标

Point circumcenter(Point a,Point b,Point c)

//p点关于直线ab的对称点

Point symmetricalPointofLine(Point p, Point a,Point b)

\*/

Point tra\_conjugate(Point A,Point B,Point C)

{

Point v,a,b,c;

v.x = (A.x + B.x + C.x)/3.0;

v.y = (A.y + B.y + C.y)/3.0;

a = symmetricalPointofLine(v,B,C);

b = symmetricalPointofLine(v,A,C);

c = symmetricalPointofLine(v,A,B);

v = circumcenter(a,b,c);

return v;

}

#### 三角形内心

Point incenter(Point a,Point b,Point c)

{

Point ua,ub,va,vb;

double m,n;

ua = a;

m = atan2(b.y-a.y,b.x-a.x);

n = atan2(c.y-a.y,c.x-a.x);

ub.x = ua.x + cos((m + n)/2);

ub.y = ua.y + sin((m + n)/2);

va = b;

m = atan2(a.y - b.y,a.x - b.x);

n = atan2(c.y - b.y,c.x - b.x);

vb.x = va.x + cos((m + n)/2);

vb.y = va.y + sin((m + n)/2);

return intersection(ua,ub,va,vb);

}

#### 三角形垂心

Point perpencenter(Point a,Point b,Point c)

{

Point ua,ub,va,vb;

ua = c;

ub.x = ua.x - a.y + b.y;

ub.y = ua.y + a.x - b.x;

va = b;

vb.x = va.x - a.y + c.y;

vb.y = va.y + a.x - c.x;

return intersection(ua,ub,va,vb);

}

#### 三角形费马点

//到三角形三顶点距离之和最小的点

Point fermentPoint(Point a,Point b,Point c)

{

Point u,v;

double step = fabs(a.x)+fabs(a.y)+fabs(b.x)+fabs(b.y)+fabs(c.x)+fabs(c.y);

int i,j,k;

u.x = (a.x+b.x+c.x)/3;

u.y = (a.y+b.y+c.y)/3;

while (step > EPS)//1E-10

{

for (k = 0;k < 10; step /= 2,k++)

for (i = -1; i <= 1; i++)

for (j = -1; j <= 1; j++)

{

v.x = u.x + step\*i;

v.y = u.y + step\*j;

if(dis(u,a)+dis(u,b)+dis(u,c)>dis(v,a)+dis(v,b)+dis(v,c))

u = v;

}

}

return u;

}

### 四边形

#### 四边形费马点

//求四边形费马点与四点距离，凸凹均可

/\*

需要函数：

struct Point

{

double x,y;

}v[5];

int sig(double a)

int cross(Point a,Point b,Point c)

{

return sig((b.x-a.x)\*(c.y-a.y) - (b.y-a.y)\*(c.x-a.x));

}

//判断线段相交

bool seg\_cross(Point a, Point b, Point c, Point d)

{

return (cross(a,c,d)\*cross(b,c,d) == -1 && cross(c,a,b)\*cross(d,a,b) == -1);

}

//两点距离

double dis()

double cal(Point a, Point b, Point c, Point d)

{

return dis(a,b)+dis(c,d);

}

\*/

double tot()

{

ans = 999999999;

//以下是可以构成凸四边形的情况，对角线交点，求交即可。

if(seg\_cross(v[0],v[1],v[2],v[3]))

{

ans = min(ans,cal(v[0],v[1],v[2],v[3]));

}

if(seg\_cross(v[0],v[2],v[1],v[3]))

{

ans = min(ans,cal(v[0],v[2],v[1],v[3]));

}

if(seg\_cross(v[0],v[3],v[1],v[2]))

{

ans = min(ans,cal(v[0],v[3],v[1],v[2]));

}

//以下是各种不能构成图四边形的情况

//（包含点重合的情况），枚举四个顶点

ans = min(ans,dis(v[0],v[1])+dis(v[0],v[2])+dis(v[0],v[3]));

ans = min(ans,dis(v[1],v[0])+dis(v[1],v[2])+dis(v[1],v[3]));

ans = min(ans,dis(v[2],v[1])+dis(v[2],v[0])+dis(v[2],v[3]));

ans = min(ans,dis(v[3],v[1])+dis(v[3],v[2])+dis(v[3],v[0]));

return ans;

}

#### 两矩形面积交

//两个平行坐标轴矩形，给左上右下角

double Inter\_Rec(Point a,Point b,Point c,Point d)

{

if (b.x <= c.x || d.x <= a.x || a.y <= d.y || c.y <= b.y)

return 0;

return (min(b.x,d.x) - max(a.x,c.x))\*(min(a.y,c.y) - max(b.y,d.y));

}

#### 判定矩形cd是否能放进ab中

bool Rect\_Inside(double c, double d, double a, double b)

{//C\*D into A\*B

if (a \* b < c \* d) return false;

if (a < b) swap(a, b);

if (c < d) swap(c, d);

if (b < d) return false;

if ((a >= c) && (b >= d)) return true;

double dis = sqrt(c \* c + d \* d);

double p = asin(b / dis);

double q = asin(d / dis);

double sita = p - q;

double Len = c \* cos(sita) + d \* sin(sita);

if (Len <= a) return true;

return false;

}

### 多边形

#### 多边形面积

//顶点按顺时针或逆时针给出,下标0~n-1

double area\_polygon(int n,Point \*p)

{

double s1 = 0,s2 = 0;

int i;

for (i = 0; i < n; i++)

s1 += p[(i+1)%n].y \* p[i].x,s2 += p[(i+1)%n].y \* p[(i+2)%n].x;

return fabs(s1-s2)/2;

}

#### 判凸多边形

/\*需要

#define \_sign(x) ((x)>eps?1:((x)<-eps?2:0))

下标从0~n-1

\*/

int is\_convex(int n,Point \*p)

{

int i,s[3]={1,1,1};

for (i = 0;i < n && s[1]|s[2]; ++i)

s[\_sign(cross(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;

return s[1]|s[2];

}

#### 点与多边形关系

//q在凸多边形内或边上返回1

int inside\_convex(Point q,Point\* p,int n) {

int i,s[3]={1,1,1};

for (i = 0;i < n && s[1]|s[2]; i++)

s[\_sign(cross(p[(i+1)%n],q,p[i]))]=0;

return s[1]|s[2];

}

//q严格在凸多边形内部返回1

int inside\_convex\_v2(Point q,Point\* p，int n)

{

int i,s[3]={1,1,1};

for (i = 0; i < n && s[0] && s[1]|s[2]; ++i)

s[\_sign(cross( p[(i+1)%n], q ,p[i] ))] = 0;

return s[0] && s[1] | s[2];

}

#### 凸包

//水平序凸包P2187

//（严格极点，不计凸包边中的点，不计重点）

Point v[50005];//点数组

int s[50005];//求得凸包点在v中的下标

/\*

使用：

sort(v + 1,v + 1 + n,cmp);

create();

\*/

int cmp(const struct Point &a, const struct Point &b ){

if( a.y < b.y )

return 1;

else

if( a.y == b.y ){

if( a.x < b.x )

return 1;

else

return 0;

}

else

return 0;

}

int dir(int now,int p1,int p2)

{

return ((v[p1].x - v[now].x)\*(v[p2].y - v[now].y) - (v[p2].x - v[now].x)\*(v[p1].y - v[now].y));

}

void create()

{

bool used[50005];

memset(used,false,sizeof(used));

s[++p] = 1;

s[++p] = 2;

for(int i = 3; i <= n; ++i)

{

while(p > 1 && dir(s[p - 1],s[p],i) <= 0)

{

used[s[p]] = false;

p--;

}

s[++p] = i;

used[s[p]] = true;

}

int top = p;

int i = n - 1;

while(used[i] == true) i--;

s[++p] = i;

for(i = i - 1; i >= 1; i--)

{

if(!used[i])

{

while(p > top && dir(s[p - 1],s[p],i) <= 0)

p--;

s[++p] = i;

}

}

}

//极角序逆时针凸包（以左下角点为起点v[1]）P1113

//凸包点存s[1]~~s[p]中，并且s[1] = s[p]

struct Point

{

int x,y;

}v[1005],s[1005];

int n,p;

int cross(Point a,Point b,Point c)

{

return (b.x - a.x)\*(c.y - a.y) - (c.x - a.x)\*(b.y - a.y);

}

int cmp(const struct Point &a,const struct Point &b)

{

if(cross(v[1],a,b) > 0)

return 1;

else if(cross(v[1],a,b) == 0)

return a.x < b.x;

else return 0;

}

void c\_h()

{

int ty = 1;

for(int i = 1; i <= n; ++i)

{

if(v[i].y < v[ty].y || (v[i].y == v[ty].y && v[i].x < v[ty].x))

ty = i;

}

struct Point temp;

temp = v[1];

v[1] = v[ty];

v[ty] = temp;

v[n + 1] = v[1];

p = 0;

sort(v + 2, v + 1 + n, cmp);

s[++p] = v[1];

for(int i = 2; i <= n + 1; ++i)

{

while(p > 1 && cross(s[p - 1],s[p],v[i]) < 0)

p--;

s[++p] = v[i];

}

}

#### 最远点距

//先使用上面水平序凸包

int sqr\_dis(Point a,Point b)

{

return (a.x - b.x)\*(a.x - b.x) + (a.y - b.y)\*(a.y - b.y);

}

int RC()//返回最远点对距离平方

{

int x = 1,m = 0;

for(int i = 1; i < p; ++i)

{

while(dir(s[i],s[i + 1],s[x + 1]) < dir(s[i],s[i + 1],s[x + 2]))

{

x++;

if(x == p) x = 1;

}

m = max(m,max(sqr\_dis(v[s[i]],v[s[x + 1]]),sqr\_dis(v[s[i + 1]],v[s[x + 2]])));

}

}

### 圆

#### 直线和圆交点

/\*

需要函数：

int sig(double a)

两点距离

double dis(struct Point p1,struct Point p2)

两不平行直线交点

Point intersection(Point a,Point b,Point c,Point d)

！要求直线和圆有交点

\*/

void intersection\_line\_circle(Point l1,Point l2,Point& p1,Point& p2)

{

Point p = o;

double t;

p.x += l1.y - l2.y;

p.y += l2.x - l1.x;

p = intersection(p,o,l1,l2);

t = sqrt(r \* r - dis(p,o) \* dis(p,o)) / dis(l1,l2);

p1.x = p.x + (l2.x - l1.x) \* t;

p1.y = p.y + (l2.y - l1.y) \* t;

p2.x = p.x - (l2.x - l1.x) \* t;

p2.y = p.y - (l2.y - l1.y) \* t;

}

#### 线段和圆交点

/\*

1.线段全部在圆内: 返回 -1

2.线段全部在圆外: 返回 0

（圆心距线段所在直线大于半径和小于半径两种情况）

3.线段与圆有一个交点: 交点是p1上返回1，交点是p2返回2

4.线段与圆有两个交点: 返回 3，分别是p1,p2

圆心Point o,半径r,直线ab

需要函数：

Sig函数

点到直线距离

double disptoline(Point p ,Point l1,Point l2)

两点距离

double dis(struct Point p1,struct Point p2)

直线和圆交点

void intersection\_line\_circle(Point l1,Point l2,Point& p1,Point& p2)

共线三点位置关系

int point\_on\_line(Point a,Point b,Point c)

\*/

int cala(struct Point a,struct Point b,Point o,double r)

{

double d = disptoline(o,a,b);

double da = dis(a,o);

double db = dis(b,o);

if (sig(d - r) < 0)

{

if (sig(da - r) <= 0 && sig(db - r) <= 0)//线段都在圆内

{

return -1;

}else

{

struct Point p1,p2;//直线与圆交点

intersection\_line\_circle(a,b,p1,p2);

double on\_p1 = point\_on\_line(p1,a,b);

double on\_p2 = point\_on\_line(p2,a,b);

if (sig(on\_p1) <= 0 && sig(on\_p2) > 0)

{//p1在ab上，p2不在ab上，说明线段与圆有一个交点p1

return 1;

}else if (sig(on\_p2) <= 0 && sig(on\_p1) > 0)

{//p2在ab上，p1不在ab上，说明线段与圆有一个交点p2

return 2;

}else if (sig(on\_p1) <= 0 && sig(on\_p2) <= 0)

{//p1,p2都在ab上，说明线段与圆有两个交点

return 3;

}else if (sig(on\_p1) > 0 && sig(on\_p2) > 0)

{//p1,p2都不在ab上，说明线段与圆没有交点

return 0;

}

}

}else //线段所在直线全在圆外

return 0;

}

#### 两圆交点

//要求两圆已经相交（相切）

void c2Point(Point p1,double r1,Point p2,double r2,Point &rp1,Point &rp2)

{

double a,b,r;

a = p2.x - p1.x;

b = p2.y - p1.y;

r = (a\*a + b\*b + r1\*r1 - r2\*r2)/2;

if(a == 0 && b != 0)

{

rp1.y = rp2.y = r/b;

rp1.x = sqrt(r1\*r1 - rp1.y\*rp1.y);

rp2.x = -rp1.x;

}else if(a != 0 && b == 0)

{

rp1.x = rp2.x = r/a;

rp1.y = sqrt(r1\*r1 - rp1.x\*rp2.x);

rp2.y = -rp1.y;

}else if(a != 0 && b != 0)

{

double delta;

delta = b\*b\*r\*r - (a\*a + b\*b)\*(r\*r - r1\*r1\*a\*a);

rp1.y = (b\*r + sqrt(delta)) / (a\*a + b\*b);

rp2.y = (b\*r - sqrt(delta)) / (a\*a + b\*b);

rp1.x = (r - b\*rp1.y) / a;

rp2.x = (r - b\*rp2.y) / a;

}

rp1.x += p1.x;

rp1.y += p1.y;

rp2.x += p1.x;

rp2.y += p1.y;

}

#### 两圆公共面积

/\*

需要函数：

两点距dis();

double tang(double a,double b,double c)

{

double s = (a+b+c)/2;

return sqrt(s\*(s-a)\*(s-b)\*(s-c));

}

\*/

double Inter\_Cir(Point o1,double r1,Point o2,double r2)

{

double d = dis(o1,o2);

if (d >= r1 + r2)

return 0;

if (d <= r1 - r2)

return PI\*r2\*r2;

if (d <= r2 - r1)

return PI\*r1\*r1;

double s = (acos((d\*d + r1\*r1 - r2\*r2)/(2\*d\*r1))\*r1\*r1)

+ (acos((d\*d + r2\*r2 - r1\*r1)/(2\*d\*r2))\*r2\*r2);

s -= tang(d,r1,r2)\*2;

return s;

}

#### 过圆外一点切线

//p---圆心坐标， r---圆半径， sp---圆外一点， rp1,rp2---切点坐标

/\*需要函数:两圆交点

c2Point(Point p1,double r1,Point p2,double r2,Point &rp1,Point &rp2)

\*/

void cutPoint(Point p,double r,Point sp,Point &rp1,Point &rp2)

{

Point p2;

p2.x = (p.x + sp.x)/2;

p2.y = (p.y + sp.y)/2;

double dx2,dy2,r2;

dx2 = p2.x - p.x;

dy2 = p2.y - p.y;

r2 = sqrt(dx2 \* dx2 + dy2 \* dy2);

c2Point(p,r,p2,r2,rp1,rp2);

}

#### 平面最小覆盖圆

/\*

所求半径r，圆心O

需要函数double dis(Point a, Point b);

期望O(n)，但好像木有。。

\*/

int n;

Point O,p[150];

double r;

void calc(double a,double b,double c,double d,double e,double f)

{

ax+by+c=0,dx+ey+f=0

//注意到三角形里两条中垂线不可能平行，所以不会产生除0错误

O.y=(c\*d-f\*a)/(b\*d-e\*a);

O.x=(c\*e-f\*b)/(a\*e-b\*d);

}

void cal()

{

O=p[1];r=0; //初始C1

for (int i = 2; i <= n; ++i) //A

if (dis(O,p[i]) > r + EPS)

{

O = p[i];

r = 0;

for (int j = 1; j <= i - 1; ++j) //B

if (dis(O,p[j]) > r + EPS)

{

O.x = (p[i].x + p[j].x) / 2;

O.y = (p[i].y + p[j].y) / 2;

r = dis(O,p[j]);

for (int k = 1; k <= j - 1; ++k) //C

if (dis(O,p[k]) > r + EPS)

{

calc(p[j].x - p[i].x, //六个参数

p[j].y - p[i].y,

(p[j].x \* p[j].x + p[j].y \* p[j].y

- p[i].x \* p[i].x - p[i].y \* p[i].y) / 2,

p[k].x - p[i].x,

p[k].y - p[i].y,

(p[k].x \* p[k].x + p[k].y \* p[k].y

- p[i].x \* p[i].x - p[i].y \* p[i].y) / 2);

r=dis(O,p[k]);

}

}

}

}

## 三维几何

### 体积

#### 四面体体积

//给定double AB, AC, AD, BC, BD, CD六个边长

double tetra\_V()

{

double ta,tb,tc;

ta = acos(chg((AB\*AB + AC\*AC - BC\*BC)/(2 \* AB \* AC)));

tb = acos(chg((AB\*AB + AD\*AD - BD\*BD)/(2 \* AB \* AD)));

tc = acos(chg((AD\*AD + AC\*AC - CD\*CD)/(2 \* AD \* AC)));

double w = (ta + tb + tc)/2;

return AB\*AC\*AD\*sqrt(sin(w)\*sin(w - ta)\*sin(w - tb)\*sin(w - tc))/3;

}

### 三维凸包

#include<stdio.h>

#include<string.h>

#include<math.h>

#include<algorithm>

using namespace std;

#define PR 1e-8

#define N 510

struct TPoint

{

double x,y,z;

TPoint(){}

TPoint(double \_x,double \_y,double \_z):x(\_x),y(\_y),z(\_z){}

TPoint operator-(const TPoint p) {return TPoint(x-p.x,y-p.y,z-p.z);}

TPoint operator\*(const TPoint p) {return TPoint(y\*p.z-z\*p.y,z\*p.x-x\*p.z,x\*p.y-y\*p.x);}//叉积

double operator^(const TPoint p) {return x\*p.x+y\*p.y+z\*p.z;}//点积

};

struct fac//

{

int a,b,c;//凸包一个面上的三个点的编号

bool ok;//该面是否是最终凸包中的面

};

struct T3dhull

{

int n;//初始点数

TPoint ply[N];//初始点

int trianglecnt;//凸包上三角形数

fac tri[N];//凸包三角形

int vis[N][N];//点i到点j是属于哪个面

double dist(TPoint a){

return sqrt(a.x\*a.x+a.y\*a.y+a.z\*a.z);}//两点长度

double area(TPoint a,TPoint b,TPoint c){

return dist((b-a)\*(c-a));}//三角形面积\*2

double volume(TPoint a,TPoint b,TPoint c,TPoint d){

return (b-a)\*(c-a)^(d-a);}//四面体有向体积\*6

double ptoplane(TPoint &p,fac &f)//正：点在面同向

{

TPoint m=ply[f.b]-ply[f.a],n=ply[f.c]-ply[f.a],t=p-ply[f.a];

return (m\*n)^t;

}

void deal(int p,int a,int b)

{

int f=vis[a][b];

fac add;

if(tri[f].ok)

{

if((ptoplane(ply[p],tri[f]))>PR) dfs(p,f);

else

{

add.a=b,add.b=a,add.c=p,add.ok=1;

vis[p][b]=vis[a][p]=vis[b][a]=trianglecnt;

tri[trianglecnt++]=add;

}

}

}

void dfs(int p,int cnt)//维护凸包，如果点p在凸包外更新凸包

{

tri[cnt].ok=0;

deal(p,tri[cnt].b,tri[cnt].a);

deal(p,tri[cnt].c,tri[cnt].b);

deal(p,tri[cnt].a,tri[cnt].c);

}

bool same(int s,int e)//判断两个面是否为同一面

{

TPoint a=ply[tri[s].a],b=ply[tri[s].b],c=ply[tri[s].c];

return fabs(volume(a,b,c,ply[tri[e].a]))<PR

&&fabs(volume(a,b,c,ply[tri[e].b]))<PR

&&fabs(volume(a,b,c,ply[tri[e].c]))<PR;

}

void construct()//构建凸包

{

int i,j;

trianglecnt=0;

if(n<4) return ;

bool tmp=true;

for(i=1;i<n;i++)//前两点不共点

{

if((dist(ply[0]-ply[i]))>PR)

{

swap(ply[1],ply[i]); tmp=false; break;

}

}

if(tmp) return;

tmp=true;

for(i=2;i<n;i++)//前三点不共线

{

if((dist((ply[0]-ply[1])\*(ply[1]-ply[i])))>PR)

{

swap(ply[2],ply[i]); tmp=false; break;

}

}

if(tmp) return ;

tmp=true;

for(i=3;i<n;i++)//前四点不共面

{

if(fabs((ply[0]-ply[1])\*(ply[1]-ply[2])^(ply[0]-ply[i]))>PR)

{

swap(ply[3],ply[i]); tmp=false; break;

}

}

if(tmp) return ;

fac add;

for(i=0;i<4;i++)//构建初始四面体

{

add.a=(i+1)%4,add.b=(i+2)%4,add.c=(i+3)%4,add.ok=1;

if((ptoplane(ply[i],add))>0)

swap(add.b,add.c);

vis[add.a][add.b]=vis[add.b][add.c]

=vis[add.c][add.a]=trianglecnt;

tri[trianglecnt++]=add;

}

for(i=4;i<n;i++)//构建更新凸包

{

for(j=0;j<trianglecnt;j++)

{

if(tri[j].ok&&(ptoplane(ply[i],tri[j]))>PR)

{

dfs(i,j); break;

}

}

}

int cnt=trianglecnt;

trianglecnt=0;

for(i=0;i<cnt;i++)

{

if(tri[i].ok)

tri[trianglecnt++]=tri[i];

}

}

double area()//表面积

{

double ret=0;

for(int i=0;i<trianglecnt;i++)

ret+=area(ply[tri[i].a],ply[tri[i].b],ply[tri[i].c]);

return ret/2.0;

}

double volume()//体积

{

TPoint p(0,0,0);

double ret=0;

for(int i=0;i<trianglecnt;i++)

ret+=volume(p,ply[tri[i].a],ply[tri[i].b],ply[tri[i].c]);

return fabs(ret/6);

}

int facetri() {return trianglecnt;}//表面三角形数

int facepolygon()//表面多边形数

{

int ans=0,i,j,k;

for(i=0;i<trianglecnt;i++)

{

for(j=0,k=1;j<i;j++)

{

if(same(i,j)) {k=0;break;}

}

ans+=k;

}

return ans;

}

}hull;

int main()

{

while(scanf("%d",&hull.n) != EOF)

{

int i;

for(i=0;i<hull.n;i++)

scanf("%lf%lf%lf",&hull.ply[i].x,&hull.ply[i].y,&hull.ply[i].z);

hull.construct();

printf("%.3f\n",hull.area());

}

return 0;

}

### 最小包围球

/\*1\*/

#include <cstdio>

#include <climits>

#include <iostream>

#include <cmath>

#include <ctime>

#include <cstdlib>

#define sqr(x) ((x)\*(x))

using namespace std;

const int N = 110, nt = 1, L = 30;

const double inf = 1e100;

const double eps = 1e-10;

int n;

double delta, maxv, minv;

struct vpoint{

double x, y, z, d;

int id;

}vp[N], test[N];

double dis(vpoint a, vpoint b) {

return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y) + sqr(a.z - b.z) + eps);

}

void calc(vpoint &p) {

p.d = 0;

for (int i = 0; i < n; ++i) {

double t = dis(p, vp[i]);

if (p.d < t) {

p.d = t;

p.id = i;

}

}

}

void updata(int id) {

double d = test[id].d;

test[id].x += (vp[test[id].id].x - test[id].x) / d \* delta;

test[id].y += (vp[test[id].id].y - test[id].y) / d \* delta;

test[id].z += (vp[test[id].id].z - test[id].z) / d \* delta;

calc(test[id]);

}

void solve() {

maxv = -inf; minv = inf;

for (int i = 0; i < n; ++i) {

scanf("%lf%lf%lf", &vp[i].x, &vp[i].y, &vp[i].z);

maxv = max(maxv, vp[i].x);

maxv = max(maxv, vp[i].y);

minv = min(minv, vp[i].x);

minv = min(minv, vp[i].y);

}

for (int i = 0; i < nt; ++i) {

test[i].x = minv + (rand() % (int(maxv - minv + 1) \* 1000)) / 1000.0;

test[i].y = minv + (rand() % (int(maxv - minv + 1) \* 1000)) / 1000.0;

test[i].z = minv + (rand() % (int(maxv - minv + 1) \* 1000)) / 1000.0;

calc(test[i]);

}

double r = 0.98;

for (delta = 30000; delta > eps; delta \*= r) {

for (int i = 0; i < nt; ++i)

for (int j = 0; j < L; ++j)

updata(i);

}

double res = inf;

int x = 0;

for (int i = 0; i < nt; ++i)

if (test[i].d < res)

{

res = test[i].d;

x = i;

}

printf("%.10lf %.10lf %.10lf\n", test[x].x, test[x].y, test[x].z);

}

int main() {

while (~scanf("%d", &n)) {

solve();

}

return 0;

}

/\*2\*/

#include <cstdio>

#include <climits>

#include <iostream>

#include <cmath>

#include <ctime>

#include <cstdlib>

#define sqr(x) ((x)\*(x))

using namespace std;

const int N = 110, nt = 1, L = 30;

const double inf = 1e100;

const double eps = 1e-10;

int n;

double delta, maxv, minv;

struct vpoint{

double x, y, z, d;

int id;

}vp[N], test[N];

double dis(vpoint a, vpoint b) {

return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y) + sqr(a.z - b.z) + eps);

}

void calc(vpoint &p) {

p.d = 0;

for (int i = 0; i < n; ++i) {

double t = dis(p, vp[i]);

if (p.d < t) {

p.d = t;

p.id = i;

}

}

}

void updata(int id) {

double d = test[id].d;

test[id].x += (vp[test[id].id].x - test[id].x) / d \* delta;

test[id].y += (vp[test[id].id].y - test[id].y) / d \* delta;

test[id].z += (vp[test[id].id].z - test[id].z) / d \* delta;

calc(test[id]);

}

void solve() {

maxv = -inf; minv = inf;

for (int i = 0; i < n; ++i) {

scanf("%lf%lf%lf", &vp[i].x, &vp[i].y, &vp[i].z);

maxv = max(maxv, vp[i].x);

maxv = max(maxv, vp[i].y);

minv = min(minv, vp[i].x);

minv = min(minv, vp[i].y);

}

for (int i = 0; i < nt; ++i) {

test[i].x = minv + (rand() % (int(maxv - minv + 1) \* 1000)) / 1000.0;

test[i].y = minv + (rand() % (int(maxv - minv + 1) \* 1000)) / 1000.0;

test[i].z = minv + (rand() % (int(maxv - minv + 1) \* 1000)) / 1000.0;

calc(test[i]);

}

double r = 0.98;

for (delta = 30000; delta > eps; delta \*= r) {

for (int i = 0; i < nt; ++i)

for (int j = 0; j < L; ++j)

updata(i);

}

double res = inf;

int x = 0;

for (int i = 0; i < nt; ++i)

if (test[i].d < res)

{

res = test[i].d;

x = i;

}

printf("%.10lf %.10lf %.10lf\n", test[x].x, test[x].y, test[x].z);

}

int main() {

while (~scanf("%d", &n)) {

solve();

}

return 0;

}

## 旋转变换

将向量(x,y)逆时针旋转d弧度：

X = x \* cos(d) - y \* sin(d);

Y = x \* sin(d) + y \* cos(d);

将3D点绕Z轴旋转的公式

X’=X\*cos(φ)-Y\*sin(φ)

Y’=X\*sin(φ)+Y\*cos(φ)

Z’=Z

// 三维点a绕向量p0p1旋转后坐标(旋转按照右手定则)

vpoint rotate(vpoint a, vpoint p0, vpoint p1, double angle) {

double d, s, c, x, y, z, x1, y1, z1;

d = dis(p0, p1); s = sin(angle); c = cos(angle);

x = (p1.x - p0.x) / d; x1 = a.x - p0.x;

y = (p1.y - p0.y) / d; y1 = a.y - p0.y;

z = (p1.z - p0.z) / d; z1 = a.z - p0.z;

p0.x += x1\*(c+x\*x\*(1-c)) + y1\*(x\*y\*(1-c)-z\*s) + z1\*(x\*z\*(1-c)+y\*s);

p0.y += y1\*(c+y\*y\*(1-c)) + z1\*(y\*z\*(1-c)-x\*s) + x1\*(y\*x\*(1-c)+z\*s);

p0.z += z1\*(c+z\*z\*(1-c)) + x1\*(z\*x\*(1-c)-y\*s) + y1\*(z\*y\*(1-c)+x\*s);

return p0;

}

# 其它成题：

### N边形费马点(退火)

#include <iostream>

#include <cstdio>

#include <cstring>

#include <cmath>

#define EPS 1e-8

#define N 128

using namespace std;

struct Point

{

double x, y;

Point(double \_x, double \_y)

{

x = \_x, y = \_y;

}

Point(){}

double toPoint(Point a)

{

return sqrt((a.x - x) \* (a.x - x) + (a.y - y) \* (a.y - y));

}

}p[N];

int n = 4;

double cost(Point v)

{

double sum = 0;

for (int i = 1; i <= n; ++i)

sum += v.toPoint(p[i]);

return sum;

}

double delta, nows, temps;

Point a, b;

double step[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

bool change;

bool init()

{

a.x = a.y = 0;

for (int i = 1; i <= n; ++i)

{

scanf("%lf%lf", &p[i].x, &p[i].y);

if ((p[i].x == -1) && (p[i].y == -1)) return false;

a.x += p[i].x, a.y += p[i].y;

}

a.x /= (double)n;

a.y /= (double)n;

return true;

}

int main()

{

while (init())

{

delta = 5000;

nows = cost(a);

while (delta > EPS)

{

do

{

change = false;

for (int k = 0; k < 4; ++k)

{

b = Point(a.x + step[k][0] \* delta, a.y + step[k][1] \* delta);

temps = cost(b);

if (temps < nows)

{

a = b;

nows = temps;

change = true;

break;

}

}

}while (change);

delta \*= 0.8;

}

printf("%.4f\n", nows);

}

return 0;

}

### 简单多边形与圆的公共面积

struct Point

{

double x,y;

}v[MAXN],o;

int n;

double r,ans;

//r为圆半径，o为圆心，v[]给出简单多边形点坐标,顺逆时针均可

int sig(double a)

{

if (fabs(a) < EPS)

return 0;

else if (a > 0)

return 1;

else return -1;

}

double cross(Point p0,Point p1,Point p2)

{

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

double dis(struct Point p1,struct Point p2)

{

return sqrt((p1.x - p2.x)\*(p1.x - p2.x) + (p1.y - p2.y)\*(p1.y - p2.y));

}

double disptoline(Point p,Point l1,Point l2)

{

return fabs(cross(p,l1,l2))/dis(l1,l2);

}

double dot(Point p1,Point p2,Point p0)

{

return (p1.x-p0.x)\*(p2.x-p0.x)+(p1.y-p0.y)\*(p2.y-p0.y);

}

double point\_on\_line(Point a,Point b,Point c) //求a点是不是在线段bc上，>0不在，=0与端点重合，<0在。

{

return (b.x-a.x)\*(c.x-a.x) + (b.y-a.y)\*(c.y-a.y);

}

Point intersection(Point u1,Point u2,Point v1,Point v2)

{

Point ret = u1;

double t = ( (u1.x - v1.x) \* (v1.y - v2.y) - (u1.y - v1.y) \* (v1.x - v2.x) )

/ ( (u1.x - u2.x) \* (v1.y-v2.y) - (u1.y - u2.y) \* (v1.x - v2.x) );

ret.x += (u2.x - u1.x) \* t;

ret.y += (u2.y - u1.y) \* t;

return ret;

}

void intersection\_line\_circle(Point l1,Point l2,Point& p1,Point& p2)

{

Point p = o;

double t;

p.x += l1.y - l2.y;

p.y += l2.x - l1.x;

p = intersection(p,o,l1,l2);

t = sqrt(r \* r - dis(p,o) \* dis(p,o)) / dis(l1,l2);

p1.x = p.x + (l2.x - l1.x) \* t;

p1.y = p.y + (l2.y - l1.y) \* t;

p2.x = p.x - (l2.x - l1.x) \* t;

p2.y = p.y - (l2.y - l1.y) \* t;

}

int dots\_inline(Point p1,Point p2,Point p3)

{

return fabs(cross(p1,p2,p3)) < EPS;

}

void cala(struct Point a,struct Point b)

{

if (dots\_inline(o,a,b))//考虑反三角函数精度

return;

double d = disptoline(o,a,b);

double da = dis(a,o);

double db = dis(b,o);

if (sig(d - r) < 0)

{

if (sig(da - r) <= 0 && sig(db - r) <= 0)//线段都在圆内

{

ans += cross(o,a,b)/2;

}else

{

double ta = 0;

double A;

struct Point p1,p2;//交点

intersection\_line\_circle(a,b,p1,p2);

double on\_p1 = point\_on\_line(p1,a,b);

double on\_p2 = point\_on\_line(p2,a,b);

if (sig(on\_p1) <= 0 && sig(on\_p2) > 0)//p1在ab上

{

if (sig(db - r) > 0 && sig(da - r) <= 0)//点b在圆外,a在圆内

{

A = acos(dot(p1,b,o)/(r\*db));

ta = A\*r\*r/2;

ta += fabs(cross(o,a,p1)/2);

}else

{

A = acos(dot(p1,a,o)/(r\*da));

ta = A\*r\*r/2;

ta += fabs(cross(o,b,p1)/2);

}

}else if (sig(on\_p2) <= 0 && sig(on\_p1) > 0)

{

double A;

if (sig(db - r) > 0 && sig(da - r) <= 0)//点b在圆外,a在圆内

{

A = acos(dot(p2,b,o)/(r\*db));

ta = A\*r\*r/2;

ta += fabs(cross(o,a,p2)/2);

}else

{

A = acos(dot(p2,a,o)/(r\*da));

ta = A\*r\*r/2;

ta += fabs(cross(o,b,p2)/2);

}

}else if (sig(on\_p1) <= 0 && sig(on\_p2) <= 0)

{

A = acos(dot(p1,a,o)/(r\*da));

ta = A\*r\*r/2;

A = acos(dot(p1,p2,o)/(r\*r));

double tmp = A\*r\*r/2;

if (sig(ta - tmp) >= 0)

{

A = acos(dot(p2,a,o)/(r\*da));

ta = A\*r\*r/2;

A = acos(dot(p1,b,o)/(r\*db));

ta += A\*r\*r/2;

}else

{

A = acos(dot(p2,b,o)/(r\*db));

ta += A\*r\*r/2;

}

ta += fabs(cross(o,p1,p2)/2);

}else if (sig(on\_p1) > 0 && sig(on\_p2) > 0)

{

A = acos(dot(a,b,o)/(db\*da));

ta = A\*r\*r/2;

}

if (sig(cross(o,a,b)) >= 0)

ans += ta;

else ans -= ta;

}

}else //线段在圆外

{

double A = acos(dot(a,b,o)/(da\*db));

if (sig(cross(o,a,b)) > 0)

ans += A\*r\*r/2;

else ans -= A\*r\*r/2;

}

}

/\*计算

初始化ans = 0.0;

for (int i = 0; i < n; ++i)

cala(v[i],v[(i + 1)%n]);

printf("%.2f\n",fabs(ans));

\*/

### 面积最大的三角形

#include<stdio.h>

#include<string.h>

#include<algorithm>

#include<math.h>

using namespace std;

#define EPS 1e-8

#define N 50015

struct TPoint

{

double x,y;

TPoint(){}

TPoint(double \_x,double \_y):x(\_x),y(\_y){}

TPoint operator-(const TPoint p) {return TPoint(x-p.x,y-p.y);}

}ply[N];

int n;

double MAX(double a,double b) {return a>b?a:b;}

double dist(TPoint a,TPoint b)//两点距

{

TPoint c(b-a);

return sqrt(c.x\*c.x+c.y\*c.y);

}

double cross(TPoint a,TPoint b,TPoint c)//叉积

{

TPoint s(b-a),t(c-a);

return s.x\*t.y-s.y\*t.x;

}

int dblcmp(double a)

{

if(fabs(a)<EPS) return 0;

return a>0?1:-1;

}

bool cmpx(TPoint a,TPoint b) //x、y排序

{

if(fabs(a.x-b.x)<EPS) return a.y<b.y;

return a.x<b.x;

}

bool cmp(TPoint a,TPoint b)//叉积内排序

{

int d1=dblcmp(cross(ply[0],a,b));

return d1>0||(d1==0 && dist(ply[0],a) < dist(ply[0],b));

}

double diss(TPoint a,TPoint b)

{

return dist(a,b)\*dist(a,b);

}

int main()

{

while(~scanf("%d",&n))

{

int i,top=2,j,p,q,r;

for(i = 0; i < n; i++)

scanf("%lf%lf",&ply[i].x,&ply[i].y);//输入点

sort(ply,ply + n,cmpx);//按x排序

sort(ply + 1,ply + n,cmp);

for(i = 2;i < n; i++)//凸包

{

while(top > 1 && (dblcmp(

cross(ply[top-2],ply[i],ply[top-1]))) >= 0)

top--;

ply[top++] = ply[i];

}

double max1=-1e20,tmp;

n = top; ply[n] = ply[0];

ply[n+1]=ply[1]; ply[n+2]=ply[2];

p=0; q=1; r=2;

while(1)//求最大三角形距离

{

int pp = p,qq = q,rr = r;

while(cross(ply[p],ply[q],ply[r+1]) - (tmp=cross(ply[p],ply[q],ply[r]))>EPS)

r = (r+1)%n;

max1 = MAX(max1,fabs(tmp)\*0.5);

while(cross(ply[p],ply[q+1],ply[r])-(tmp=cross(ply[p],ply[q],ply[r]))>EPS)

q = (q+1)%n;

max1 = MAX(max1,fabs(tmp)\*0.5);

while(cross(ply[p+1],ply[q],ply[r])-(tmp=cross(ply[p],ply[q],ply[r]))>EPS)

p = (p+1)%n;

max1 = MAX(max1,fabs(tmp)\*0.5);

if(pp == p && qq == q && rr == r)

r = (r+1)%n;

if(r == 0) break;

}

if(n<3) max1=0;

printf("%.2lf\n",max1);

}

return 0;

}

### 判定多边形是否对称

#include <iostream>

#include <cstdio>

#include <cstring>

#include <cmath>

#define EPS 0.00000001

using namespace std;

int n,num;

struct Point

{

int x,y;

}v[20005],s,t;

bool is\_polygon()

{

\_\_int64 s1=0,s2=0;

int i;

for (i=0;i<n;i++)

s1+=v[(i+1)%n].y\*v[i].x,s2+=v[(i+1)%n].y\*v[(i+2)%n].x;

return (s1 != s2);

}

int cross(Point p0,Point p1,Point p2)

{

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

int dots\_inline(Point p1,Point p2,Point p3)

{

return cross(p1,p2,p3);

}

int dot(int x1,int y1,int x2,int y2)

{

return (x1\*x2 + y1\*y2);

}

bool ok\_v(int x)//对称轴过x号顶点

{

Point t1,t2,mid;

for (int i = 1; i <= num + 1; ++i)

{

t1 = v[(x + n - i) % n];

t2 = v[(x + i) % n];

mid.x = (t1.x + t2.x)/2;

mid.y = (t1.y + t2.y)/2;

if (dots\_inline(mid,s,t) != 0 || dot(t1.x - t2.x ,t1.y - t2.y , s.x - t.x,s.y - t.y) != 0)

return false;

}

return true;

}

bool ok\_e(int x)//对称轴过x和x + 1边的中点

{

Point t1,t2,mid;

for (int i = 0; i <= num + 1; ++i)

{

t1 = v[(x + n - i) % n];

t2 = v[(x + 1 + i) % n];

mid.x = (t1.x + t2.x)/2;

mid.y = (t1.y + t2.y)/2;

if (dots\_inline(mid,s,t) != 0 || dot(t1.x - t2.x ,t1.y - t2.y , s.x - t.x,s.y - t.y) != 0)

{

return false;

}

}

return true;

}

bool cal()

{

if (n % 2 == 0)//偶数个点

{

//点-点

num = (n - 2)/2;

for (int i = 0; i < n; ++i)

{

s = v[i];

t = v[(i + num + 1)%n];

if (ok\_v(i))

return true;

}

//边-边

num = (n - 4)/2;

for (int i = 0; i < n; ++i)

{

s.x = (v[i].x + v[(i + 1) % n].x)/2;

s.y = (v[i].y + v[(i + 1) % n].y)/2;

t.x = (v[(i + n - num - 1) % n].x + v[(i + num + 2) % n].x)/2;

t.y = (v[(i + n - num - 1) % n].y + v[(i + num + 2) % n].y)/2;

if (ok\_e(i))

return true;

}

return false;

}else

{

for (int i = 0; i < n; ++i)

{

//点-边

num = (n - 3)/2;

s = v[i];

t.x = (v[(i + n - num - 1) % n].x + v[(i + num + 1) % n].x)/2;

t.y = (v[(i + n - num - 1) % n].y + v[(i + num + 1) % n].y)/2;

if (ok\_v(i))

{

return true;

}

}

return false;

}

}

int main()

{

while(scanf("%d",&n) != EOF)

{

for (int i = 0; i < n; ++i)

{

scanf("%d%d",&v[i].x,&v[i].y);

v[i].x \*= 2;

v[i].y \*= 2;

}

if (!is\_polygon())

{

printf("YES\n");

continue;

}

if (cal())

printf("YES\n");

else printf("NO\n");

}

return 0;

}

### 贴海报

//POJ2528 贴海报,每个点代表一单位线段，注意离散化方法

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#define N 10010

using namespace std;

struct PAP

{

int l,r;

}p[N];

int rem[2\*N];

int n,cnt;

int hash[10000010];

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

struct CNode

{

int L,R;

bool covered;

}v[100\*N];

void Build(int p,int s,int t)

{

v[p].L = s;

v[p].R = t;

v[p].covered = false;

int mid = (s + t)/2;

if (s != t)

{

Build(p\*2,s,mid);

Build(p\*2 + 1,mid + 1,t);

}

}

bool Insert(int p,int s,int t)

{

if(v[p].covered)

return false;

if (v[p].L >= s && v[p].R <= t)

{

v[p].covered = true;

return true;

}

bool res;

int mid = (v[p].L + v[p].R)/2;

if (t <= mid)

res = Insert(p\*2,s,t);

else if (s >= mid + 1)

res = Insert(p\*2 + 1,s,t);

else

{//这里直接||会截断的

res = Insert(p\*2,s,mid);

res = Insert(p\*2 + 1,mid + 1,t) || res;

}

//优化：更新父结点

if (v[p\*2].covered && v[p\*2 + 1].covered)

v[p].covered = true;

return res;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main()

{

int T;

scanf("%d",&T);

while(T--)

{

scanf("%d",&n);

cnt = 0;

for (int i = 1; i <= n; ++i)

{

scanf("%d%d",&p[i].l,&p[i].r);

rem[++cnt] = p[i].l;

rem[++cnt] = p[i].r;

}

sort(rem + 1,rem + cnt + 1);

cnt = unique(rem + 1,rem + cnt + 1) - rem;

int ncnt = 1;

hash[rem[1]] = 1;

for (int i = 2; i < cnt; ++i)

{

if (rem[i] - rem[i - 1] == 1)//相邻

{

hash[rem[i]] = ++ncnt;

}else

{//把不相邻的空留出来

ncnt += 2;

hash[rem[i]] = ncnt;

}

}

Build(1,1,ncnt);

int ans = 0;

for (int i = n; i >= 1; --i)

{

if(Insert(1,hash[p[i].l],hash[p[i].r]))

ans++;

}

printf("%d\n",ans);

}

return 0;

}

### 矩形面积并

//POJ 1151 矩形切割的面积

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#define N 105

using namespace std;

struct REC

{

double x1,x2,y1,y2;

}r[N];

Double ry[2\*N];//按y坐标离散化

int ny;

double ans;

int findx(double x)

{

int l = 1,r = ny;

int mid;

while(l < r)

{

mid = (l + r)/2;

if (x == ry[mid])

return mid;

else if (x < ry[mid])

r = mid - 1;

else l = mid + 1;

}

return l;

}

struct EDGE

{

double x;

bool f;

double y1,y2;

}e[2\*N];//竖边从左到右一个个插入

int ne;

int cmp(const EDGE &a,const EDGE &b)

{

return a.x < b.x;

}

//区间线段树

struct NODE

{

int L,R;

double len;

int cov;

}v[5\*N];

void Build(int p,int s,int t)

{

v[p].L = s;v[p].R = t;

v[p].len = 0.0;

v[p].cov = 0;

int mid = (s + t)/2;

if (s + 1 != t) //表示区间，叶结点L，R差1

{

Build(p\*2,s,mid);

Build(p\*2 + 1,mid,t);

}

}

void Insert(int p,int s,int t)

{

if (s <= v[p].L && v[p].R <= t)

{

v[p].len = ry[v[p].R] - ry[v[p].L];

v[p].cov++;

return;

}

int mid = (v[p].L + v[p].R)/2;

if (t <= mid)

Insert(p\*2,s,t);//

else if (s >= mid)

Insert(p\*2 + 1,s,t);//

else

{

Insert(p\*2,s,mid);

Insert(p\*2 + 1,mid,t);

}

if (v[p].cov == 0)

{

v[p].len = v[p\*2].len + v[p\*2 + 1].len;

}

}

void Delete(int p,int s,int t)

{

if (s <= v[p].L && v[p].R <= t)

{

if(v[p].cov != 0)

v[p].cov--;

if (v[p].cov == 0)

{

if (v[p].L + 1 == v[p].R)

v[p].len = 0;

else

v[p].len = v[p\*2].len + v[p\*2 + 1].len;

}

return;

}

int mid = (v[p].L + v[p].R)/2;

if (t <= mid)

Delete(p\*2,s,t);//介种地方要注意啊！

else if (s >= mid)

Delete(p\*2 + 1,s,t);//

else

{

Delete(p\*2,s,mid);

Delete(p\*2 + 1,mid,t);

}

if (v[p].cov == 0)

{

v[p].len = v[p\*2].len + v[p\*2 + 1].len;

}

}

int main()

{

int ca = 1;

int n;

while(scanf("%d",&n) && n)

{

double a,b,c,d;

ny = 0;ne = 0;

for (int i = 1; i <= n; ++i)

{

scanf("%lf%lf%lf%lf",&a,&b,&c,&d);

r[i].x1 = min(a,c);

r[i].x2 = max(a,c);

r[i].y1 = min(b,d);

r[i].y2 = max(b,d);

ry[++ny] = b;

ry[++ny] = d;

e[++ne].x = a;

e[ne].f = true;

e[ne].y1 = r[i].y1;

e[ne].y2 = r[i].y2;

e[++ne].x = c;

e[ne].f = false;

e[ne].y1 = r[i].y1;

e[ne].y2 = r[i].y2;

}

sort(ry + 1,ry + ny + 1);

ny = unique(ry + 1, ry + ny + 1) - (ry + 1);

sort (e + 1,e + ne + 1,cmp);

Build(1,1,ny );

ans = 0;

for (int i = 1; i < ne; ++i)

{

int s,t;

s = findx(e[i].y1);

t = findx(e[i].y2);

if (e[i].f)

Insert(1,s,t);

else Delete(1,s,t);

ans += v[1].len\*(e[i + 1].x - e[i].x);

}

printf("Test case #%d\n",ca++);

printf("Total explored area: %.2f\n\n",ans);

}

return 0;

}

### 半平面交求多边形的核

//半平面交求多边形的核

#include <cstdio>

#include <algorithm>

#include <cmath>

#include <iostream>

#define MAXN 105

#define eps 1e-8

using namespace std;

struct Point

{

double x,y;

Point(){}

Point(double a,double b)

{x = a,y = b;}

}point[MAXN];

int sig(double k)

{

return (k < -eps) ? -1 : (k > eps);

}

void getline(const Point &x,const Point &y,double &a,double &b,double &c)

{

a = y.y - x.y;

b = x.x - y.x;

c = y.x \* x.y - x.x \* y.y;

}

Point p[MAXN],q[MAXN];

int sz,s;

Point interect(const Point &x,const Point &y,double &a,double &b,double &c)

{

double u = fabs(a \* x.x + b \* x.y + c);

double v = fabs(a \* y.x + b \* y.y + c);

return Point( (x.x \* v + y.x \* u) / (u + v) , (x.y \* v + y.y \* u) / (u + v) );

}

void init(int n)

{//初始化边界

for(int i = 1 ; i <= n ; i ++)

p[i] = point[i];

p[n + 1] = p[1]; p[0] = p[n];

sz = n;

}

//利用半平面切割

void cut(double a,double b,double c)

{

s = 0;

for(int i = 1 ; i <= sz ; i ++)

{

if(sig(a \* p[i].x + b \* p[i].y + c) >= 0)

{//因为线段是顺时针给出的，如果是逆时针就是<=0

q[++s] = p[i];

}else

{

if(sig(a \* p[i - 1].x + b \* p[i - 1].y + c) > 0)//逆时针就是<0

q[++s] = interect(p[i - 1],p[i],a,b,c);

if(sig(a \* p[i + 1].x + b \* p[i + 1].y + c) > 0)//逆时针就是<0

q[++s] = interect(p[i + 1],p[i],a,b,c);

}

}

//最后的p数组存放半平面的点集合,可能重复

for(int i = 1; i <= s ; i ++)

p[i] = q[i];

p[s + 1] = p[1] , p[0] = p[s];

sz = s;

}

void solve(int n)

{

init(n);//初始化边界

double a,b,c;

for(int i = 1 ; i <= n ; i ++)

{

getline(point[i],point[i + 1],a,b,c);

cut(a,b,c);

}

/\*

double area = 0;//面积

for( int i = 1 ; i <= sz ; i ++)

area += p[i].x \* p[i + 1].y - p[i + 1].x \* p[i].y;

area = fabs(area / 2.0);

\*/

if(!sz) puts("Surveillance is impossible.");

//这里如果有一个点，或者一条线段都可以，所以判断sz是不是等于0就行了，不用判断面积

else puts("Surveillance is possible.");

puts("");

}

int main()

{

int n;int ca = 1;

while(scanf("%d",&n),n)

{

for(int i = 1; i <= n ; i ++)

scanf("%lf%lf",&point[i].x,&point[i].y);

point[n + 1] = point[1];//注意这一步，方便以后运算

printf("Floor #%d\n",ca++);

solve(n);

}

}

### 半平面交求多条直线所围区域面积

#include <cstdio>

#include <cstdlib>

#include <iostream>

#include <algorithm>

#include <cstring>

#include <string>

#include <cmath>

using namespace std;

const double eps = 1e-8;

const double pi = acos(-1.0);

const int N = 20010;

const double maxl = 10000;

struct Point {

double x, y;

Point(double xx = 0, double yy = 0): x(xx), y(yy) {};

};

int dcmp(double x) {

if (x < -eps) return -1; else return x > eps;

}

double cross(Point p0, Point p1, Point p2) { // p0p1 与 p0p2 叉积

return (p1.x - p0.x) \* (p2.y - p0.y) - (p2.x - p0.x) \* (p1.y - p0.y);

}

bool EqualPoint(Point a, Point b) {

return dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) == 0;

}

//==============半平面交==================//

struct cvector {

Point s, e;

double ang, d;

};

void setline(double x1, double y1, double x2, double y2, cvector &v)

{

v.s.x = x1; v.s.y = y1;

v.e.x = x2; v.e.y = y2;

v.ang = atan2(y2 - y1, x2 - x1);

if (dcmp(x1 - x2))

v.d = (x1 \* y2 - x2 \* y1) / fabs(x1 - x2);

else

v.d = (x1 \* y2 - x2 \* y1) / fabs(y1 - y2);

}

//判向量平行

bool parallel(const cvector &a, const cvector &b)

{

double u = (a.e.x - a.s.x) \* (b.e.y - b.s.y)

- (a.e.y - a.s.y) \* (b.e.x - b.s.x);

return dcmp(u) == 0;

}

//求两向量(直线)交点 (两向量不能平行或重合)

Point CrossPoint(const cvector &a, const cvector &b)

{

Point res;

double u = cross(a.s, a.e, b.s), v = cross(a.e, a.s, b.e);

res.x = (b.s.x \* v + b.e.x \* u) / (u + v);

res.y = (b.s.y \* v + b.e.y \* u) / (u + v);

return res;

}

//半平面交排序函数[优先顺序: 1.极角 2.前面的直线在后面的左边]

static bool VecCmp(const cvector &l, const cvector &r) {

if (dcmp(l.ang - r.ang)) return l.ang < r.ang;

return l.d < r.d;

}

cvector deq[N]; //用于计算的双端队列

//获取半平面交的多边形（多边形的核）

//注意:1.半平面在向量左边, 2.函数会改变vec[]中的值vec大小不可小于2.

//函数运行后如果n[即返回多边形的点数量]为0则

//不存在半平面交的多边形（不存在区域或区域面积无穷大）

void HalfPanelCross(cvector vec[], int n, Point cp[], int &m)

{

int i, tn; m = 0;

sort(vec, vec + n, VecCmp);

for(i = tn = 1; i < n; ++i) //平面在向量左边的筛选

if(dcmp(vec[i].ang - vec[i - 1].ang) != 0)

vec[tn++] = vec[i];

n = tn;

int bot = 0, top = 1;

deq[0] = vec[0];

deq[1] = vec[1]; // vec[]大小不可小于2

for (i = 2; i < n; ++i) {

if (parallel(deq[top], deq[top - 1]) ||

parallel(deq[bot], deq[bot + 1]) ) return ;

while ( bot < top && dcmp( cross(vec[i].s, vec[i].e,

CrossPoint(deq[top], deq[top - 1])) ) < 0 ) top--;

while ( bot < top && dcmp( cross(vec[i].s, vec[i].e,

CrossPoint(deq[bot], deq[bot + 1])) ) < 0 ) bot++;

deq[++top] = vec[i];

}

while ( bot < top && dcmp( cross(deq[bot].s, deq[bot].e,

CrossPoint(deq[top], deq[top - 1])) ) < 0 ) top--;

while ( bot < top && dcmp( cross(deq[top].s, deq[top].e,

CrossPoint(deq[bot], deq[bot + 1])) ) < 0 ) bot++;

if (top <= bot + 1) return ; // 两条或两条以下的直线，面积无穷大

for (i = bot; i < top; i ++)

cp[m++] = CrossPoint(deq[i], deq[i + 1]);

if (bot < top + 1)

cp[m++] = CrossPoint(deq[bot], deq[top]);

m = unique(cp, cp + m, EqualPoint) - cp;

for (i = 0; i < m; ++i) {

if (dcmp(cp[i].x) == 0) cp[i].x = 0;

if (dcmp(cp[i].y) == 0) cp[i].y = 0;

}

}

double PolygonArea(Point p[], int n)

{

if (n < 3) return 0;

double s = p[0].y \* (p[n - 1].x - p[1].x);

for (int i = 1; i < n; ++i)

s += p[i].y \* (p[i - 1].x - p[(i + 1) % n].x);

return fabs(s / 2); // 顺时针方向s为负

}

int n, m;

cvector v[N];

Point cp[N];

void solve()

{

setline(0, 0, maxl, 0, v[0]);

setline(maxl, 0, maxl, maxl, v[1]);

setline(maxl, maxl, 0, maxl, v[2]);

setline(0, maxl, 0, 0, v[3]);

n += 4;

double x1, x2, y1, y2;

for (int i = 4; i < n; ++i)

{

scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);

setline(x1, y1, x2, y2, v[i]);

}

HalfPanelCross(v, n, cp, m);

if (m < 3)

printf("0.0\n");

else

printf("%.1f\n", PolygonArea(cp, m));

}

int main()

{

while (scanf("%d", &n) != EOF)

{

solve();

}

return 0;

}

### 平面图欧拉定理

//给平面图求分出的面数，V:不重复交点数，E：被点分出的边数

//V + F - E = 1 本题F = E - V + 2;

#include <iostream>

#include <cstring>

#include <cstdio>

#include <cmath>

#include <algorithm>

#define EPS 0.00000001

using namespace std;

struct Point

{

double x,y;

}v[310],rem[90005];

int n,E,V,num;

int sig(double x)

{

if (fabs(x) < EPS)

return 0;

else if(x > 0) return 1;

else return -1;

}

int cmp(const Point &a,const Point &b)

{

if ((a.x - b.x) > EPS)

return 1;

else if (fabs(a.x - b.x) < EPS && (a.y - b.y) > EPS)

return 1;

else return 0;

}

int cmp2(const Point &a,const Point &b)

{

if (sig(a.x - b.x) == 0 && sig(a.y - b.y) == 0)

return 1;

else return 0;

}

double cross(Point p0,Point p1,Point p2)

{

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

double dot(double x1,double y1,double x2,double y2) //点积

{

return x1\*x2+y1\*y2;

}

int Point\_on\_line(Point a,Point b,Point c) //求a点是不是在线段bc上，>0不在，=0与端点重合，<0在。

{

return sig(dot(b.x-a.x,b.y-a.y,c.x-a.x,c.y-a.y));

}

int Share\_Line(Point a,Point b,Point c,Point d)

{

if (sig((a.x - b.x)\*(c.y - d.y) - (c.x - d.x)\*(a.y - b.y)) == 0 &&

sig((a.x - b.x)\*(c.x - d.x) - (c.y - d.y)\*(a.y - b.y)) == 0)

return 1;

return 0;

}

void ab\_cross\_cd(Point a,Point b,Point c,Point d)

{

double 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(s3=cross(c,d,a));

d4 = sig(s4=cross(c,d,b));

Point p;

//（要考虑到共线但没有公共点的情况）

//如果规范相交则求交点

if (d1\*d2 < 0 && d3\*d4 < 0) //整型

{

p.x=(c.x\*s2-d.x\*s1)\*1.0/(s2-s1);

p.y=(c.y\*s2-d.y\*s1)\*1.0/(s2-s1);

rem[num++] = p;

}

if (d1==0 && Point\_on\_line(c,a,b)<=0)

{

p.x = c.x;p.y = c.y;

rem[num++] = p;

}

if (d2==0 && Point\_on\_line(d,a,b)<=0)

{

p.x = d.x;p.y = d.y;

rem[num++] = p;

}

if (d3==0 && Point\_on\_line(a,c,d)<=0)

{

p.x = d.x;p.y = d.y;

rem[num++] = p;

}

if (d4==0 && Point\_on\_line(b,c,d)<=0)

{

p.x = d.x;p.y = d.y;

rem[num++] = p;

}

}

int main()

{

int ca = 1;

while (scanf("%d",&n) && n != 0)

{

E = 0;V = 0;num = 0;

for (int i = 0; i < n; ++i)

scanf("%lf%lf",&v[i].x,&v[i].y);

int cnt;

n--;

for (int i = 1; i <= n; ++i)

{

for (int j = i + 1; j <= n; ++j)

{

if (!Share\_Line(v[i - 1],v[i],v[j - 1],v[j]))

{

ab\_cross\_cd(v[i - 1],v[i],v[j - 1],v[j]);

}

}

}

sort(rem,rem + num,cmp);

num = unique(rem,rem + num,cmp2) - rem;

V = num;

for (int i = 0; i < num; ++i)

{

for (int j = 1; j <= n; ++j)

if(sig(cross(rem[i],v[j - 1],v[j])) == 0 && Point\_on\_line(rem[i],v[j - 1],v[j]) < 0)

{

E++;

}

}

printf("Case %d: There are %d pieces.\n",ca++,E + n - V + 2);

}

return 0;

}

### 光线遇圆反射

//POJ 1263，输出遇到的前10个圆编号，也适用三维

#include <iostream>

#include <cmath>

using namespace std;

#define out(x) (cout << #x << ": " << x << endl)

const int maxint = 0x7FFFFFFF;

template <class T> void get\_max(T &a, const T &b) {b > a ? a = b : 1;}

template <class T> void get\_min(T &a, const T &b) {b < a ? a = b : 1;}

const double eps = 1e-9;

const double pi = acos(-1);

struct point {

double x, y, z;

point() {}

point(double \_x, double \_y, double \_z) : x(\_x), y(\_y), z(\_z) {}

};

struct vec {

double x, y, z;

vec() {}

vec(double \_x, double \_y, double \_z) : x(\_x), y(\_y), z(\_z) {}

vec(point p) : x(p.x), y(p.y), z(p.z) {}

vec(point s, point e) : x(e.x - s.x), y(e.y - s.y), z(e.z - s.z) {}

};

int sgn(double d) {

return (d > eps) - (d < -eps);

}

vec cmul(const vec &a, const vec &b) {

return vec(a.y \* b.z - b.y \* a.z, a.z \* b.x - a.x \* b.z, a.x \* b.y - a.y \* b.x);

}

double dmul(const vec &a, const vec &b) {

return a.x \* b.x + a.y \* b.y + a.z \* b.z;

}

double sqr(const double &x) {

return x \* x;

}

double len(const vec &a) {

return sqrt(sqr(a.x) + sqr(a.y) + sqr(a.z));

}

double len2(const vec &a) {

return sqr(a.x) + sqr(a.y) + sqr(a.z);

}

double dist2\_point\_point(const point &a, const point &b) {

return sqr(a.x - b.x) + sqr(a.y - b.y) + sqr(a.z - b.z);

}

double dist2\_point\_line(const point &p, const point &a, const point &b) {

return abs(len2(cmul(vec(p, a), vec(p, b)))) / len2(vec(a, b));

}

int inter\_circle\_line(const point &c, double r, const point &p0, const vec &v0, point &p1, vec &v1) {

double h2 = dist2\_point\_line(c, p0, point(p0.x + v0.x, p0.y + v0.y, p0.z + v0.z));

int flag = sgn(h2 - sqr(r));

if (flag > 0) return 0; /\* 不相交 \*/

double sh = dmul(v0, vec(p0, c));

if (sh <= 0) return 0; /\* 光线延长后与圆心相交，无法反射 \*/

double l2 = dist2\_point\_point(p0, c);

double t = sqrt(l2 - h2) - sqrt(sqr(r) - h2);

double lv0 = len(v0);

vec vt(v0.x \* t / lv0, v0.y \* t / lv0, v0.z \* t / lv0);

p1 = point(p0.x + vt.x, p0.y + vt.y, p0.z + vt.z);

double t2 = 2 \* (dmul(vt, vec(p1, c)) / r);

vec vtt((p1.x - c.x) \* t2 / r, (p1.y - c.y) \* t2 / r, (p1.z - c.z) \* t2 / r);

v1 = vec(vt.x + vtt.x, vt.y + vtt.y, vt.z + vtt.z);

return 1;

}

int n;

point c[100];

double r[100];

point p;

vec v;

int main() {

int T = 1;

while (scanf("%d", &n), n != 0) {

printf("Scene %dn", T++);

for (int i = 0; i < n; i++) {

scanf("%lf%lf%lf", &c[i].x, &c[i].y, &r[i]);

c[i].z = 0;

}

point tp;

scanf("%lf%lf%lf%lf", &p.x, &p.y, &v.x, &v.y);

p.z = 0;

v.z = 0;

int cnt = 0;

while (cnt <= 10) {

double mind = 1e100;

int mini = -1;

point mp;

vec mv;

for (int i = 0; i < n; i++) {

point np;

vec nv;

if (inter\_circle\_line(c[i], r[i], p, v, np, nv)) {

double td = len2(vec(p, np));

if (td < mind) {

mind = td;

mini = i;

mp = np;

mv = nv;

}

}

}

if (mini == -1) {

printf("inf");

break;

}

if (cnt == 10) {

printf("...");

break;

}

printf("%d ", mini + 1);

cnt++;

p = mp;

v = mv;

}

printf("nn");

}

return 0;

}

### 其它几何模板

//三维点向一个平面投影后，求覆盖圆

//其中很多函数可能有用

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <algorithm>

#include <math.h>

using namespace std;

const double eps = 1e-8, inf = 1e+9, pi = acos(-1.0);

inline int sign(double x) {

return x < -eps ? -1 : x > eps;

}

inline double sqr(double x) {

return x \* x;

}

/\* ---------- points ---------- \*/

struct point {

double x, y;

point(double x = 0, double y = 0) : x(x), y(y) {}

bool operator<(const point &p) const {

return sign(x - p.x) \* 2 + sign(y - p.y) < 0;

}

bool operator==(const point &p) const {

return !sign(x - p.x) && !sign(y - p.y);

}

};

void read(point \*p) {

scanf("%lf %lf", &p->x, &p->y);

}

void print(point p) {

printf("%.3lf %.3lf\n", p.x, p.y);

}

point operator-(point a, point b) {

a.x -= b.x;

a.y -= b.y;

return a;

}

double cross(point a, point b, point c) {

return (b.x - a.x) \* (c.y - b.y) - (b.y - a.y) \* (c.x - b.x);

}

double dist(point a, point b) {

return hypot(a.x - b.x, a.y - b.y);

}

/\* mid point \*/

point mp(point a, point b) {

return point((a.x + b.x) / 2, (a.y + b.y) / 2);

}

/\* next point \*/

point np(point a, double alpha, double d) {

return point(a.x + d \* cos(alpha), a.y + d \* sin(alpha));

}

/\* PA \* PB \*/

double dot(point p, point a, point b) {

return (a.x - p.x) \* (b.x - p.x) + (a.y - p.y) \* (b.y - p.y);

}

/\* delta angle of ABC \*/

inline double delta\_angle(point a, point b, point c) {

return acos(dot(b, a, c) / (dist(a, b) \* dist(b, c)));

}

/\* if ABC a sharp triangle \*/

int sharp(point a, point b, point c) {

return sign(dot(a, b, c)) > 0 && sign(dot(b, a, c)) > 0 && sign(dot(c, a, b)) > 0;

}

double fix(double a, double b = 0) {

a -= b;

if (sign(a) < 0) a += 2 \* pi;

if (sign(a - 2 \* pi) >= 0) a -= 2 \* pi;

return a;

}

double angle(point a, point b) {

return fix(atan2(b.y - a.y, b.x - a.x));

}

point rotate(point a, point b, double alpha) {

double s, c, x, y;

a.x -= b.x;

a.y -= b.y;

c = cos(alpha);

s = sin(alpha);

x = a.x \* c - a.y \* s;

y = a.x \* s + a.y \* c;

a.x = x + b.x;

a.y = y + b.y;

return a;

}

double area\_heron(double a, double b, double c) {

double s = (a + b + c) / 2.0;

if (a > s || b > s || c > s) return -1;

return sqrt(s \* (s - a) \* (s - b) \* (s - c));

}

double area\_triangle(point a, point b, point c) {

return fabs(a.x \* b.y + b.x \* c.y + c.x \* a.y - a.x \* c.y - b.x \* a.y - c.x \* b.y) / 2;

}

double area\_polygon(point a[], int n) {

double area = 0;

for (int j = 0, i = n - 1; j < n; i = j++)

area += a[i].x \* a[j].y - a[i].y \* a[j].x;

return fabs(area) / 2;

}

point centroid(point a[], int n) {

double area = 0;

point c;

for (int i = 0; i < n; i++) {

int j = (i + 1) % n;

area += a[i].x \* a[j].y - a[i].y \* a[j].x;

c.x += (a[i].x + a[j].x) \* (a[i].x \* a[j].y - a[i].y \* a[j].x);

c.y += (a[i].y + a[j].y) \* (a[i].x \* a[j].y - a[i].y \* a[j].x);

}

area = fabs(area) / 2;

c.x /= 6 \* area;

c.y /= 6 \* area;

return c;

}

/\* ---------- convex hull ---------- \*/

point \_\_o;

bool acmp(point a, point b) {

int c = sign(cross(\_\_o, a, b));

return c > 0 || !c && dist(a, \_\_o) < dist(b, \_\_o);

}

/\* find convex hull of p[n] in place

\* return # of points of resulting convex hull \*/

int find\_convex(point p[], int n) {

\_\_o = \*min\_element(p, p + n);

sort(p, p + n, acmp);

int top = 0;

point \*stack = (point \*) malloc(n \* sizeof(point)); // XXX malloc here!

for (int i = 0; i < n; i++) {

while (top >= 2 && sign(cross(stack[top - 2], stack[top - 1], p[i])) <= 0) top--;

stack[top++] = p[i];

}

copy(stack, stack + top, p);

free(stack);

return top;

}

/\* ---------- rotate calipers ---------- \*/

double shadow\_length(double alpha, point a, point b) {

double dx = a.x - b.x;

double dy = a.y - b.y;

double c = cos(alpha);

double s = sin(alpha);

return fabs(dx \* c + dy \* s);

}

/\* min area & min peri rectangle covering, using rotate calipers \*/

void rotate\_calipers(point ps[], int n, double &area, double &peri) {

area = peri = inf;

n = find\_convex(ps, n);

ps[n] = ps[0];

point \*q[4] = { NULL, NULL, NULL, NULL };

for (int i = 0; i < n; i++) {

point \*p = &ps[i];

if (!q[0] || q[0]->y > p->y || q[0]->y == p->y && q[0]->x > p->x) q[0] = p;

if (!q[1] || q[1]->x < p->x || q[1]->x == p->x && q[1]->y > p->y) q[1] = p;

if (!q[2] || q[2]->y < p->y || q[2]->y == p->y && q[2]->x < p->x) q[2] = p;

if (!q[3] || q[3]->x > p->x || q[3]->x == p->x && q[3]->y < p->y) q[3] = p;

}

double alpha = 0;

for (int k = 0; k < n + 5; k++) {

int bi = -1;

double gap\_min = inf;

for (int i = 0; i < 4; i++) {

double gap = fix(angle(q[i][0], q[i][1]), alpha + i \* pi / 2);

if (gap < gap\_min) {

gap\_min = gap;

bi = i;

}

}

if (++q[bi] == ps + n) q[bi] = ps + 0;

alpha = fix(alpha + gap\_min);

double a = shadow\_length(alpha + pi / 2, \*q[0], \*q[2]);

double b = shadow\_length(alpha, \*q[1], \*q[3]);

area = min(area, a \* b);

peri = min(peri, a + a + b + b);

}

}

/\* ---------- lines ---------- \*/

struct line {

point p, q;

line() {}

line(point p, point q) : p(p), q(q) {}

};

int parallel(line u, line v) {

return !sign((u.p.x - u.q.x) \* (v.p.y - v.q.y) - (v.p.x - v.q.x) \* (u.p.y - u.q.y));

}

/\* same side: 1; at least one of a, b touches l: 0; otherwise -1 \*/

int side(line m, point p, point q) {

return sign(cross(m.p, m.q, p)) \* sign(cross(m.p, m.q, q));

}

int on\_line(line l, point p) {

return !sign(cross(l.p, l.q, p));

}

/\* u, v: line \*/

int coinside(line u, line v) {

return on\_line(u, v.p) && on\_line(u, v.q);

}

/\* u, v: line segment, inclusive \*/

int intersected(line u, line v) {

return !parallel(u, v) && side(u, v.p, v.q) <= 0 && side(v, u.p, u.q) <= 0;

}

/\* u, v: line segment, exclusive \*/

int intersected\_exclusive(line u, line v) {

return !parallel(u, v) && side(u, v.p, v.q) < 0 && side(v, u.p, u.q) < 0;

}

/\* intersection point \*/

point ip(line u, line v) {

double n = (u.p.y - v.p.y) \* (v.q.x - v.p.x) - (u.p.x - v.p.x) \* (v.q.y - v.p.y);

double d = (u.q.x - u.p.x) \* (v.q.y - v.p.y) - (u.q.y - u.p.y) \* (v.q.x - v.p.x);

double r = n / d;

return point(u.p.x + r \* (u.q.x - u.p.x), u.p.y + r \* (u.q.y - u.p.y));

}

/\* if P on the line segment l, inclusive \*/

int on\_lineseg(line l, point P) {

return on\_line(l, P) && sign(dot(P, l.p, l.q)) <= 0;

}

/\* if P on the line segment l, exclusive \*/

int on\_lineseg\_exclusive(line l, point P) {

return on\_line(l, P) && sign(dot(P, l.p, l.q)) < 0;

}

double dist\_line\_point(line l, point a) {

return fabs(cross(l.p, l.q, a)) / dist(l.p, l.q);

}

double dist\_lineseg\_point(line l, point a) {

if (on\_lineseg(l, a)) return 0;

if (on\_line(l, a) || !sharp(l.p, a, l.q))

return min(dist(l.p, a), dist(l.q, a));

return dist\_line\_point(l, a);

}

/\* u: line segment, ab: ray, p is the resulting intersection point \*/

int intersected\_lineseg\_ray(line u, line v, point &p) {

if (parallel(u, v)) return 0;

p = ip(u, v);

return on\_lineseg(u, p) && (on\_lineseg(v, p) || on\_lineseg(line(v.p, p), v.q));

}

/\* if point a inside polygon p[n] \*/

int inside\_polygon(point p[], int n, point a) {

double sum = 0;

for (int i = 0; i < n; i++) {

int j = (i + 1) % n;

if (on\_lineseg(line(p[i], p[j]), a)) return 0;

double angle = acos(dot(a, p[i], p[j]) / dist(a, p[i]) / dist(a, p[j]));

sum += sign(cross(a, p[i], p[j])) \* angle;

}

return sign(sum);

}

/\* if lineseg l strickly inside polygon p[n] \*/

int lineseg\_inside\_polygon(point p[], int n, line l) {

for (int i = 0; i < n; i++) {

int j = (i + 1) % n;

line l1(p[i], p[j]);

if (on\_lineseg\_exclusive(l, p[i])) return 0;

if (intersected\_exclusive(l, l1)) return 0;

}

return inside\_polygon(p, n, mp(l.p, l.q));

}

/\* if lineseg l intersect convex polygon p[n] \*/

int intersect\_convex\_lineseg(point p[], int n, line l) {

if (n < 3) return 0;

point q[4]; int k = 0;

q[k++] = l.p;

q[k++] = l.q;

for (int i = 0; i < n; i++) {

if (on\_lineseg(l, p[i])) {

q[k++] = p[i];

} else {

int j = (i + 1) % n;

line a(p[i], p[j]);

point X = ip(a, l);

if (on\_lineseg(l, X) && on\_lineseg(a, X)) q[k++] = X;

}

}

sort(q, q + k);

for (int i = 0; i + 1 < k; i++) {

if (inside\_polygon(p, n, mp(q[i], q[i + 1]))) return 1;

}

return 0;

}

line perpendicular(line l, point a) {

return line(a, point(a.x + l.p.y - l.q.y, a.y + l.q.x - l.p.x));

}

point pedal(line l, point a) {

return ip(l, perpendicular(l, a));

}

point mirror(line l, point a) {

point p = pedal(l, a);

return point(p.x \* 2 - a.x, p.y \* 2 - a.y);

}

point perpencenter(point a, point b, point c) {

line u = perpendicular(line(b, c), a);

line v = perpendicular(line(a, c), b);

return ip(u, v);

}

/\* Inscribed circle center \*/

point icc(point A, point B, point C) {

double a = dist(B, C), b = dist(C, A), c = dist(A, B),

p = (a + b + c) / 2,

s = sqrt(p \* (p - a) \* (p - b) \* (p - c));

point cp;

cp.x = (a \* A.x + b \* B.x + c \* C.x) / (a + b + c);

cp.y = (a \* A.y + b \* B.y + c \* C.y) / (a + b + c);

return cp;

}

/\* Perpendicular bisector \*/

line pb(point a, point b) {

return perpendicular(line(a, b), mp(a, b));

}

/\* circumcicle center \*/

point ccc(point A, point B, point C) {

double a1 = B.x - A.x, b1 = B.y - A.y, c1 = (sqr(a1) + sqr(b1)) / 2;

double a2 = C.x - A.x, b2 = C.y - A.y, c2 = (sqr(a2) + sqr(b2)) / 2;

double d = a1 \* b2 - a2 \* b1;

point cp;

cp.x = A.x + (c1 \* b2 - c2 \* b1) / d;

cp.y = A.y + (a1 \* c2 - a2 \* c1) / d;

return cp;

}

/\* translate l with distance e and direction s \*/

line translate(line l, double e, int s) {

double d = dist(l.p, l.q);

double x = l.p.y - l.q.y;

double y = l.q.x - l.p.x;

x \*= s \* e / d;

y \*= s \* e / d;

l.p.x += x; l.p.y += y;

l.q.x += x; l.q.y += y;

return l;

}

/\* ---------- intersection points convex hull ---------- \*/

bool lcmp(line u, line v) {

int c = sign((u.p.x - u.q.x) \* (v.p.y - v.q.y) - (v.p.x - v.q.x) \* (u.p.y - u.q.y));

return c < 0 || !c && sign(cross(u.p, u.q, v.p)) < 0;

}

/\* XXX sizeof(p) MUST be as large as n \* 2

\* return # of points of resulting convex hull \*/

int ip\_convex(line l[], int n, point p[]) {

for (int i = 0; i < n; i++)

if (l[i].q < l[i].p) swap(l[i].p, l[i].q);

sort(l, l + n, lcmp);

int n1 = 0;

for (int i = 0, j = 0; i < n; i = j) {

while (j < n && parallel(l[i], l[j])) j++;

if (j - i == 1) {

l[n1++] = l[i];

} else {

l[n1++] = l[i];

l[n1++] = l[j - 1];

}

}

n = n1;

l[n + 0] = l[0];

l[n + 1] = l[1];

int m = 0;

for (int i = 0, j = 0; i < n; i++) {

while (j < n + 2 && parallel(l[i], l[j])) j++;

for (int k = j; k < n + 2 && parallel(l[j], l[k]); k++)

p[m++] = ip(l[i], l[k]);

}

return find\_convex(p, m);

}

/\* ---------- circles ---------- \*/

struct circle {

point o;

double r;

circle(point o = point(), double r = 1) : o(o), r(r) {}

circle(double x, double y, double r = 1) : o(x, y), r(r) {}

};

int intersected\_circle\_line(circle c, line l) {

return sign(dist\_line\_point(l, c.o) - c.r) < 0;

}

int ip\_circle\_line(circle c, line l, point &p1, point &p2) {

point a = l.p, b = l.q;

double dx = b.x - a.x, dy = b.y - a.y;

double sdr = sqr(dx) + sqr(dy), dr = sqrt(sdr);

double d, disc, x, y;

a.x -= c.o.x; a.y -= c.o.y;

b.x -= c.o.x; b.y -= c.o.y;

d = a.x \* b.y - b.x \* a.y;

disc = sqr(c.r) \* sdr - sqr(d);

if (disc < -eps) return 0;

if (disc < +eps) disc = 0; else disc = sqrt(disc);

x = disc \* dx \* (dy > 0 ? 1 : -1);

y = disc \* fabs(dy);

p1.x = (+d \* dy + x) / sdr + c.o.x;

p2.x = (+d \* dy - x) / sdr + c.o.x;

p1.y = (-d \* dx + y) / sdr + c.o.y;

p2.y = (-d \* dx - y) / sdr + c.o.y;

return disc > eps ? 2 : 1;

}

int ip\_circle\_circle(const circle &c1, const circle &c2, point &p1, point &p2) {

double mx = c2.o.x - c1.o.x, sx = c2.o.x + c1.o.x, mx2 = sqr(mx);

double my = c2.o.y - c1.o.y, sy = c2.o.y + c1.o.y, my2 = sqr(my);

double sq = mx2 + my2, d = -(sq - sqr(c1.r - c2.r)) \* (sq - sqr(c1.r + c2.r));

if (!sign(sq)) return 0;

if (d + eps < 0) return 0;

if (d < eps) d = 0; else d = sqrt(d);

double x = mx \* ((c1.r + c2.r) \* (c1.r - c2.r) + mx \* sx) + sx \* my2;

double y = my \* ((c1.r + c2.r) \* (c1.r - c2.r) + my \* sy) + sy \* mx2;

double dx = mx \* d, dy = my \* d; sq \*= 2;

p1.x = (x + dy) / sq; p1.y = (y - dx) / sq;

p2.x = (x - dy) / sq; p2.y = (y + dx) / sq;

return d > eps ? 2 : 1;

}

double circle\_circle\_intersection\_area(circle A, circle B) {

double d, dA, dB, tx, ty;

d = hypot(B.o.x - A.o.x, B.o.y - A.o.y);

if ((d < eps) || (d + A.r <= B.r) || (d + B.r <= A.r))

return sqr((B.r < A.r) ? B.r : A.r) \* pi;

if (d >= A.r + B.r)

return 0;

dA = tx = (sqr(d) + sqr(A.r) - sqr(B.r)) / d / 2;

ty = sqrt(sqr(A.r) - sqr(tx));

dB = d - dA;

return sqr(A.r) \* acos(dA / A.r) - dA \* sqrt(sqr(A.r) - sqr(dA)) +

sqr(B.r) \* acos(dB / B.r) - dB \* sqrt(sqr(B.r) - sqr(dB));

}

/\* return 2 points of tangency of c and p \*/

void circle\_tangents(circle c, point p, point &a, point &b) {

double d = sqr(c.o.x - p.x) + sqr(c.o.y - p.y);

double para = sqr(c.r) / d;

double perp = c.r \* sqrt(d - sqr(c.r)) / d;

a.x = c.o.x + (p.x - c.o.x) \* para - (p.y - c.o.y) \* perp;

a.y = c.o.y + (p.y - c.o.y) \* para + (p.x - c.o.x) \* perp;

b.x = c.o.x + (p.x - c.o.x) \* para + (p.y - c.o.y) \* perp;

b.y = c.o.y + (p.y - c.o.y) \* para - (p.x - c.o.x) \* perp;

}

/\* 0: oncircle; 1: inside circle; -1: outside circle\*/

int on\_circle(circle c, point a) {

return sign(c.r - dist(a, c.o));

}

/\* minimum circle that covers 2 points \*/

circle cc2(point a, point b) {

return circle(mp(a, b), dist(a, b) / 2);

}

/\* minimum circle that covers 3 points \*/

circle cc3(point p, point q, point r) {

circle c;

if (on\_circle(c = cc2(p, q), r) >= 0) return c;

if (on\_circle(c = cc2(p, r), q) >= 0) return c;

if (on\_circle(c = cc2(q, r), p) >= 0) return c;

c.o = ccc(p, q, r);

c.r = dist(c.o, p);

return c;

}

/\* minimum circle that covers n points \*/

circle min\_circle\_cover(point p[], int n) {

if (n == 1) return circle(p[0], 0);

if (n == 2) return cc2(p[0], p[1]);

random\_shuffle(p, p + n);

point \*ps[4] = { &p[0], &p[1], &p[2], &p[3] };

circle c = cc3(\*ps[0], \*ps[1], \*ps[2]);

while (1) {

point \*b = p;

for (int i = 1; i < n; i++)

if (dist(p[i], c.o) > dist(\*b, c.o)) b = &p[i];

if (on\_circle(c, \*b) >= 0) return c;

ps[3] = b;

for (int i = 0; i < 3; i++) {

swap(ps[i], ps[3]);

if (on\_circle(c = cc3(\*ps[0], \*ps[1], \*ps[2]), \*ps[3]) >= 0) break;

}

}

}

typedef struct pt {

double x, y, z;

} vt;

inline vt operator+(pt a, pt b) {

a.x += b.x;

a.y += b.y;

a.z += b.z;

return a;

}

inline vt operator-(pt a, pt b) {

a.x -= b.x;

a.y -= b.y;

a.z -= b.z;

return a;

}

inline vt operator\*(double t, vt a) {

a.x \*= t;

a.y \*= t;

a.z \*= t;

return a;

}

inline vt operator\*(vt a, double t) {

a.x \*= t;

a.y \*= t;

a.z \*= t;

return a;

}

inline vt operator/(vt a, double t) {

a.x /= t;

a.y /= t;

a.z /= t;

return a;

}

inline vt operator\*(vt a, vt b) {

vt c;

c.x = a.y \* b.z - a.z \* b.y;

c.y = a.z \* b.x - a.x \* b.z;

c.z = a.x \* b.y - a.y \* b.x;

return c;

}

inline double operator^(vt a, vt b) {

return a.x \* b.x + a.y \* b.y + a.z \* b.z;

}

inline double len(vt a) {

return sqrt(a ^ a);

}

inline int zero(vt a) {

return !sign(a.x) && !sign(a.y) && !sign(a.z);

}

inline double frand() {

return rand() / (RAND\_MAX + 1.0);

}

/\* returns a vt that perps to u \*/

vt perp\_vt(vt u) {

vt v, n;

while (1) {

v.x = frand();

v.y = frand();

v.z = frand();

if (!zero(n = u \* v)) return n;

}

}

const int N = 1024;

int n, m;

pt ps[N];

point qs[N];

int main() {

scanf("%d %d", &n, &m);

for (int i = 0; i < n; i++)

scanf("%lf %lf %lf", &ps[i].x, &ps[i].y, &ps[i].z);

while (m--) {

vt norm, xdir, ydir;

scanf("%lf %lf %lf", &norm.x, &norm.y, &norm.z);

norm = norm / len(norm);

xdir = perp\_vt(norm);

xdir = xdir / len(xdir);

ydir = norm \* xdir;

for (int i = 0; i < n; i++) {

qs[i].x = ps[i] ^ xdir;

qs[i].y = ps[i] ^ ydir;

}

sort(qs, qs + n);

int n1 = unique(qs, qs + n) - qs;

printf("%.10lf\n", min\_circle\_cover(qs, n1).r);

}

return 0;

}

### 一些可能用到的性质

扇形的重心在角平分线上距圆心Xc = 2\*R\*sinA/3/A 的位置，A为扇形圆心角的一半

设地球上某点的经度为lambda,纬度为phi， 则这点的空间坐标是

x=cos(phi)\*cos(lambda)

y=cos(phi)\*sin(lambda)

z=sin(phi)

设地球上两点的空间坐标分别为(x1,y1,z1),(x2,y2,z2)

直线距离即为R\*sqrt((x2-x1)\*(x2-x1)+(y2-y1)\*(y2-y1)+(z2-z1)\*(z2-z1)),

则它们的夹角为 A = acos(x1 \* x2 + y1 \* y2 + z1 \* z2)，

则两地距离为 A \* R，其中R为地球平均半径6371

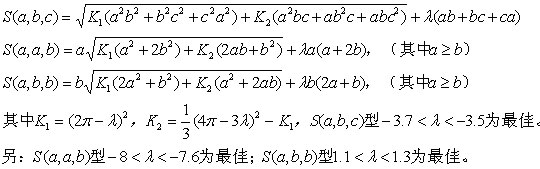
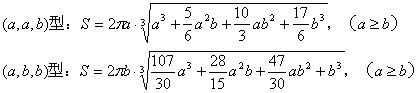
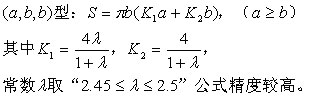
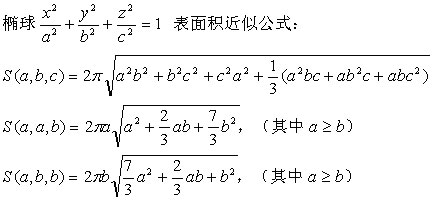
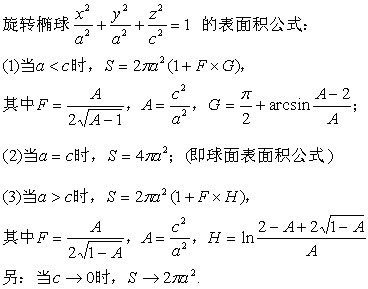
椭圆上的点（x，y）与两焦点围成的三角形面积 S=b^2\*tan(α/2) α为点(x，y)与两焦点连线的夹角

椭圆周长近似值：PI\*q(1 + 3\*h/(10 + sqrt(4-3\*h))\*(1 + m\*n)（精度很好）

(q = a+b，h=((a-b)/(a+b))^2， m=22/7\*PI - 1，n = ((a-b)/a)^33.697)

椭球体积V=4/3\*PI\*A\*B\*C。其中A,B,C分别是X，Y，Z 轴方向的半径。

椭球表面积公式：



费马点即到个顶点距离和最小的点：

对于三角形：

1.三个角都小于120度，则费马点在三角形内部与任意两定点的连线构成的角都为120度

2.若存在一个角大于等于120度，则费马点为此角顶点

对于平面四边形（说话要注意，哈哈）：

1.若为凸四边形，则费马点为两对角线交点

2.对于凹四边形，为其凹顶点

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

圆内接三角形周长最大的是等边三角形

/\*

到三边距离平方和最小的点，三角形重心的等角共扼点

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AH·HD=BH·HE=CH·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/AP·tanB+ AC/AQ·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，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=90°+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=p·AI·BI·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.外心有如下一系列优美性质：

性质1 三角形的外心是三角形三条边垂直平分线的交点；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=360°-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/AP ·sin2B+ AC/AQ·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QD·QE·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。

\*/

Turan定理:

空间内n个点 若他们之间的连线条数大于等于[(n^2+1)/4](取整) 则必存在一个以这些点为顶点的三角形