

**二维几何**



by tokitsukaze

目录

[1.几何公式 1](#_Toc21379)

[1.1 三角形 1](#_Toc3738)

[1.2 四边形 1](#_Toc31834)

[1.3 正 n 边形 1](#_Toc28155)

[1.4 圆 1](#_Toc24743)

[1.5 棱柱 2](#_Toc2406)

[1.6 棱锥 2](#_Toc18067)

[1.7 棱台 2](#_Toc31684)

[1.8 圆柱 2](#_Toc7479)

[1.9 圆锥 3](#_Toc8246)

[1.10 圆台 3](#_Toc8673)

[1.11 球 3](#_Toc31452)

[1.12 球台 3](#_Toc7207)

[1.13 球扇形 3](#_Toc8449)

[1.14 欧拉四面体公式 3](#_Toc20468)

[2.点与常用函数 4](#_Toc25478)

[2.1 点定义 4](#_Toc31933)

[2.2 + - \* / < == 4](#_Toc21951)

[2.3 常用函数 5](#_Toc15218)

[2.3.1 两个向量夹角 TwoVectorAngle 5](#_Toc23586)

[2.3.2 向量的模 length 5](#_Toc22763)

[2.3.3 点积 dot 5](#_Toc2418)

[2.3.4 叉积 cross 5](#_Toc30953)

[2.3.5 两点距离 dist 5](#_Toc1617)

[2.3.6 线段ab中点 midseg 5](#_Toc28124)

[2.3.7 垂直法向量 Normal 5](#_Toc24163)

[2.3.8 向量翻转 Rotate 5](#_Toc21513)

[2.3.9 三个点求三角形面积 calarea 5](#_Toc10074)

[2.3.10 单位向量 vecunit 5](#_Toc13005)

[2.3.11 向量的极角 VectorAngle 5](#_Toc22958)

[2.3.12 向量叉积意义 5](#_Toc15632)

[3.直线与线段 6](#_Toc12237)

[3.1 直线的定义 6](#_Toc21022)

[3.2 线段 7](#_Toc18432)

[3.2.1 点p在线段上 OnSeg 7](#_Toc10885)

[3.2.2 点p在线段范围内 PointInRange 7](#_Toc26214)

[3.2.3 点到线段距离 DisPointToSeg 7](#_Toc14505)

[3.2.4 两线段最近距离 DisSegToSeg 7](#_Toc24480)

[3.2.5 线段相交判定 JudgeSegInter 8](#_Toc17193)

[3.3 直线 8](#_Toc14899)

[3.3.1 判断直线ab和线段cd是否相交 SegInterLine 8](#_Toc12114)

[3.3.2 点到直线距离 DisPointToLine 8](#_Toc17324)

[3.3.3 判断两条直线位置关系(重合/平行/相交) JudgeLineInter 8](#_Toc25643)

[3.3.4 求两直线交点 PointOfLineInter 9](#_Toc32761)

[3.3.5 点关于直线ax+by+c=0的对称点 PointSymmetryLine 9](#_Toc13023)

[3.3.6 点p在直线ab的投影 GetLineProjection 9](#_Toc19622)

[3.4 半平面交 hpi 10](#_Toc22427)

[4.三角形 11](#_Toc7893)

[4.1 内心(内切圆请见圆部分) InCenter 11](#_Toc32267)

[4.2 外心(外接圆请见圆部分) CircumCircle 11](#_Toc14895)

[4.3 垂心 OrthoCenter 11](#_Toc20257)

[5.多边形 12](#_Toc28703)

[5.1 凸包 graham 12](#_Toc31189)

[5.2 判断点是否在多边形内 JudgePointInPolygon 13](#_Toc12817)

[5.3 多边形重心 (点集逆时针给出) bcenter 13](#_Toc30447)

[5.4 最近点对 14](#_Toc22163)

[5.5 模拟退火求费马点 15](#_Toc2078)

[5.6 最小圆覆盖 16](#_Toc12419)

[5.7 旋转卡壳 17](#_Toc13556)

[5.7.1 平面最远点对(凸包直径) RC\_FarthestPair 18](#_Toc20659)

[5.7.2 点集最大三角形 RC\_Triangle 18](#_Toc23667)

[5.7.3 两凸包最近距离 RC\_Dis 19](#_Toc19881)

[6.圆 20](#_Toc26706)

[6.1 圆的定义 20](#_Toc2663)

[6.2 点P到圆的切线 getTangents 20](#_Toc22312)

[6.3 求a点到b点(逆时针)在的圆上的圆弧长度 ArcLen 21](#_Toc31835)

[6.4 两圆的公切线 CommonTangents 21](#_Toc15635)

[6.5 三角形外接圆 CircumscribedCircle 22](#_Toc12775)

[6.6 三角形内切圆 InscribedCircle 22](#_Toc5798)

[6.7 线段与圆的交点 getSegCircleInter 23](#_Toc9767)

[6.8 直线与圆交点 getLineCircleInter 23](#_Toc26647)

[6.9 两圆交点 getCircleInter 24](#_Toc5082)

[6.10 判断点在圆内 InCircle 25](#_Toc13061)

[7.面积问题 26](#_Toc18585)

[7.1 多边形面积 PolygonArea 26](#_Toc24401)

[7.2 凸多边形面积并 CPIA 26](#_Toc17088)

[7.3 多边形面积并 SPIA 27](#_Toc22494)

[7.4 多边形与圆面积交 PolyCiclrArea 28](#_Toc30649)

[7.5 两圆面积交 CircleCircleArea 30](#_Toc25140)

[8.Pick定理 30](#_Toc19853)

# **1.**几何公式

## **1.1** 三角形

1. 半周长 P=(a+b+c)/2
2. 面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))
3. 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
4. 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
5. 高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)
6. 内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)

=4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)

=Ptan(A/2)tan(B/2)tan(C/2)

1. 外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))

## **1.2** 四边形

D1,D2为对角线,M对角线中点连线,A为对角线夹角

1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
2. S=D1D2sin(A)/2

(以下对圆的内接四边形)

1. ac+bd=D1D2
2. S=sqrt((P-a)(P-b)(P-c)(P-d)),P为半周长

## **1.3** 正 **n** 边形

R为外接圆半径,r为内切圆半径

1. 中心角 A=2PI/n
2. 内角 C=(n-2)PI/n
3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))

## **1.4** 圆

1. 弧长 l=rA
2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)
3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2
4. 扇形面积 S1=rl/2=r^2A/2
5. 弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2

## **1.5** 棱柱

1. 体积 V=Ah,A为底面积,h为高
2. 侧面积 S=lp,l为棱长,p为直截面周长
3. 全面积 T=S+2A

## **1.6** 棱锥

1. 体积 V=Ah/3,A为底面积,h为高

(以下对正棱锥)

1. 侧面积 S=lp/2,l为斜高,p为底面周长
2. 全面积 T=S+A

## **1.7** 棱台

1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2为上下底面积,h为高

(以下为正棱台)

1. 侧面积 S=(p1+p2)l/2,p1.p2为上下底面周长,l为斜高
2. 全面积 T=S+A1+A2

## **1.8** 圆柱

1. 侧面积 S=2PIrh
2. 全面积 T=2PIr(h+r)
3. 体积 V=PIr^2h

## **1.9** 圆锥

1. 母线 l=sqrt(h^2+r^2)
2. 侧面积 S=PIrl
3. 全面积 T=PIr(l+r)
4. 体积 V=PIr^2h/3

## **1.10** 圆台

1. 母线 l=sqrt(h^2+(r1-r2)^2)
2. 侧面积 S=PI(r1+r2)l
3. 全面积 T=PIr1(l+r1)+PIr2(l+r2)
4. 体积 V=PI(r1^2+r2^2+r1r2)h/3

## **1.11** 球

1. 全面积 T=4PIr^2 2. 体积 V=4PIr^3/3

## **1.12** 球台

1. 侧面积 S=2PIrh
2. 全面积 T=PI(2rh+r1^2+r2^2)
3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6

## **1.13** 球扇形

1. 全面积 T=PIr(2h+r0),h为球冠高,r0为球冠底面半径
2. 体积 V=2PIr^2h/3

## **1.14** 欧拉四面体公式

double solve(double a,double b,double c,double d,double e,double f)

{return sqrt((4\*a\*a\*b\*b\*c\*c-a\*a\*(b\*b+c\*c-e\*e)\*(b\*b+c\*c-e\*e)-b\*b\*(c\*c+a\*a-f\*f)\*(c\*c+a\*a-f\*f)-c\*c\*(a\*a+b\*b-d\*d)\*(a\*a+b\*b-d\*d)+(a\*a+b\*b-d\*d)\*(b\*b+c\*c-e\*e)\*(c\*c+a\*a-f\*f)))/12;}

# **2.**点与常用函数

## **2.1** 点定义

int sgn(double x)

{

if(fabs(x)<eps) return 0;

else return x>0?1:-1;

}

//点定义

struct Point

{

double x,y;

Point(){}

Point(double a,double b)

{

x=a;

y=b;

}

void input()

{

scanf("%lf%lf",&x,&y);

}

};

typedef Point Vector;

## **2.2** + - \* / < ==

Vector operator +(Vector a,Vector b){return Vector(a.x+b.x,a.y+b.y);}

Vector operator -(Vector a,Vector b){return Vector(a.x-b.x,a.y-b.y);}

Vector operator \*(Vector a,double p){return Vector(a.x\*p,a.y\*p);}

Vector operator /(Vector a,double p){return Vector(a.x/p,a.y/p);}

bool operator <(Point a,Point b){return a.x<b.x||(a.x==b.x&&a.y<b.y);}

bool operator ==(Point a,Point b){return sgn(a.x-b.x)==0&&sgn(a.y-b.y)==0;}

## **2.3 常用函数**

2.3.1 两个向量夹角 TwoVectorAngle

double TwoVectorAngle(Vector a,Vector b){return acos(dot(a,b)/length(a)/length(b));}

2.3.2 向量的模 length

double length(Vector a){return sqrt(dot(a,a));}

2.3.3 点积 dot

double dot(Vector a,Vector b){return a.x\*b.x+a.y\*b.y;}

2.3.4 叉积 cross

double cross(Vector a,Vector b){return a.x\*b.y-a.y\*b.x;}

2.3.5 两点距离 dist

double dist(Point a,Point b){return sqrt(dot(a-b,a-b));}

2.3.6 线段ab中点 midseg

Point midseg(Point a,Point b){return (a+b)/2;}

2.3.7 垂直法向量 Normal

Vector Normal(Vector x){return Point(-x.y,x.x)/length(x);}

2.3.8 向量翻转 Rotate

Vector Rotate(Vector a,double rad){return Vector(a.x\*cos(rad)-a.y\*sin(rad),a.x\*sin(rad)+a.y\*cos(rad));}

2.3.9 三个点求三角形面积 calarea

double calarea(Point c,Point b,Point a){return cross(b-a,c-a);}

2.3.10 单位向量 vecunit

Vector vecunit(Vector x){return x/length(x);}

2.3.11 向量的极角 VectorAngle

double VectorAngle(Vector v){return atan2(v.y,v.x);}

2.3.12 向量叉积意义

若 PxQ>0,则P在Q的顺时针方向。

若 PxQ<0,则P在Q的逆时针方向。

若 PxQ=0,则P与Q共线,但可能同向也可能反向。

# **3.**直线与线段

## **3.1 直线的定义**

//有向直线,它的左边就是对应的半平面

struct Line

{

Point p;//直线上任意一点

Vector v;

double ang;//极角,即从x正半轴旋转到向量v所需要的角（弧度）

double a,b,c;//直线一般式

Line(){}

Line(Point a,Vector b)//点斜式

{

p=a;

v=b;

ang=atan2(v.y,v.x);

LineGeneralEquation();

}

void twopoint(Point a,Point b)//两点式

{

p=a;

v=b-a;

ang=atan2(v.y,v.x);

LineGeneralEquation();

}

Point getpoint(double a)

{

return p+(v\*a);

}

void LineGeneralEquation()//计算一般式的a,b,c

{

Point p1,p2;

p1=p;

p2=p+v;

a=p2.y-p1.y;

b=p1.x-p2.x;

c=-a\*p1.x-b\*p1.y;

}

};

**3.2** 线段

3.2.1 点p在线段上 OnSeg

//线段包含端点时改成<=

bool OnSeg(Point p,Point p1,Point p2)

{

return sgn(cross(p1-p,p2-p))==0&&sgn(dot(p1-p,p2-p))<0;

}

3.2.2 点p在线段范围内 PointInRange

bool PointInRange(Point p,Point a,Point b)

{

return dot(p-a,b-a)>0&&dot(p-b,a-b)>0;//pab<90 pba<90

}

3.2.3 点到线段距离 DisPointToSeg

double DisPointToSeg(Point p,Point a,Point b)

{

if(a==b) return length(p-a);

Vector v1,v2,v3;

v1=b-a;

v2=p-a;

v3=p-b;

if(sgn(dot(v1,v2))<0) return length(v2);

else if(sgn(dot(v1,v3))>0) return length(v3);

else return fabs(cross(v1,v2))/length(v1);

}

3.2.4 两线段最近距离 DisSegToSeg

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

{

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

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

}

3.2.5 线段相交判定 JudgeSegInter

bool JudgeSegInter(Point a,Point b,Point c,Point d)

{

double t1,t2,t3,t4;

t1=cross(b-a,c-a);

t2=cross(b-a,d-a);

t3=cross(d-c,a-c);

t4=cross(d-c,b-c);

return sgn(t1)\*sgn(t2)<0&&sgn(t3)\*sgn(t4)<0;

}

**3.3** 直线

3.3.1 判断直线ab和线段cd是否相交 SegInterLine

bool SegInterLine(Point a,Point b,Point c,Point d)

{

return sgn(cross((c-b),(a-b)))\*sgn(cross((d-b),(a-b)))<=0;

}

3.3.2 点到直线距离 DisPointToLine

double DisPointToLine(Point p, Point a, Point b)

{

Vector v1,v2;

v1=b-a;

v2=p-a;

return fabs(cross(v1,v2))/length(v1);//如果不取绝对值,得到的是有向距离

}

3.3.3 判断两条直线位置关系(重合/平行/相交) JudgeLineInter

int JudgeLineInter(Line a,Line b)

{

if(sgn(cross(a.v,b.v))==0)

{

if(sgn(cross(a.p-b.p,b.v))==0) return 0;//重合

else return 1;//平行

}

else return 2;//有交点

}

3.3.4 求两直线交点 PointOfLineInter

Point PointOfLineInter(Line a,Line b)

{

Vector u=a.p-b.p;

double t=cross(b.v,u)/cross(a.v,b.v);

return a.p+a.v\*t;

}

3.3.5 点关于直线ax+by+c=0的对称点 PointSymmetryLine

Point PointSymmetryLine(Point p,Line l)

{

Point res;

double k,a,b,c;

a=l.a;

b=l.b;

c=l.c;

k=-2\*(a\*p.x+b\*p.y+c)/(a\*a+b\*b);

res.x=p.x+k\*a;

res.y=p.y+k\*b;

return res;

}

3.3.6 点p在直线ab的投影 GetLineProjection

Point GetLineProjection(Point p,Point a,Point b)

{

Vector v=b-a;

return a+v\*(Dot(v,p-a)/dot(v,v));

}

**3.4** 半平面交 hpi

bool OnLeft(Line l,Point p)

{

return cross(l.v,p-l.p)>0; //点p在有向直线L的左边(线上不算)

}

bool hpicmp(Line a,Line b)

{

return a.ang<b.ang;

}

vector<Point> hpi(vector<Line> l)

{

int n,i;

n=l.size();

sort(l.begin(),l.end(),hpicmp);//按极角排序

int first,last; // 双端队列的第一个元素和最后一个元素的下标

vector<Point> p(n); // p[i]为q[i]和q[i+1]的交点

vector<Line> q(n); // 双端队列

vector<Point> ans; // 结果

q[first=last=0]=l[0]; // 双端队列初始化为只有一个半平面L[0]

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

{

while(first<last&&!OnLeft(l[i],p[last-1])) last--;

while(first<last&&!OnLeft(l[i],p[first])) first++;

q[++last]=l[i];

if(fabs(cross(q[last].v,q[last-1].v))<eps)

{ // 两向量平行且同向，取内侧的一个

last--;

if(OnLeft(q[last],l[i].p)) q[last]=l[i];

}

if(first<last) p[last-1]=PointOfLineInter(q[last-1],q[last]);

}

while(first<last&&!OnLeft(q[first],p[last-1])) last--; //删除无用平面

if(last-first<=1) return ans; //空集

p[last]=PointOfLineInter(q[last],q[first]); //计算首尾两个半平面的交点

for(i=first;i<=last;i++)

{

ans.push\_back(p[i]);// 从deque复制到输出中

}

return ans;

}

# **4.**三角形

**4.1** 内心(内切圆请见圆部分) InCenter

Point InCenter(Point a,Point b,Point c)

{

Line u,v;

double m,n;

u.a=a;

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

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

u.b.x=u.a.x+cos((m+n)/2);

u.b.y=u.a.y+sin((m+n)/2);

v.a=b;

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

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

v.b.x=v.a.x+cos((m+n)/2);

v.b.y=v.a.y+sin((m+n)/2);

return PointOfLineInter(u,v);

}

**4.2** 外心(外接圆请见圆部分) CircumCircle

Point CircumCircle(Point a,Point b,Point c)

{

Point res;

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);

}

**4.3** 垂心 OrthoCenter

Point OrthoCenter(Point a,Point b,Point c)

{

double c1,c2,d;

Point p1,p2;

p1=c-b;

c1=0;

p2=c-a;

c2=dot(b-a,p2);

d=cross(p1,p2);

return Point(a.x+(c1\*p2.y-c2\*p1.y)/d,a.y+(p1.x\*c2-p2.x\*c1)/d);

}

# **5.**多边形

**5.1** 凸包 graham

// 如果不希望在凸包的边上有输入点，把两个 <= 改成 <

// 注意：输入点集会被修改

vector<Point> graham(vector<Point> p)

{

int n,m,k,i;

sort(p.begin(),p.end());//要用到operator <

p.erase(unique(p.begin(),p.end()),p.end());//要用到operator ==

n=p.size();

m=0;

vector<Point> res(n+1);

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

{

while(m>1&&cross(res[m-1]-res[m-2],p[i]-res[m-2])<=0) m--;

res[m++]=p[i];

}

k=m;

for(i=n-2;i>=0;i--)

{

while(m>k&&cross(res[m-1]-res[m-2],p[i]-res[m-2])<=0) m--;

res[m++]=p[i];

}

if(n>1) m--;

res.resize(m);

return res;

}

**5.2** 判断点是否在多边形内 JudgePointInPolygon

int JudgePointInPolygon(Point p,vector<Point> poly)

{

int cnt,n,k,d1,d2;

cnt=0;

n=poly.size();

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

{

if(OnSeg(p,poly[i],poly[(i+1)%n])) return -1;//在边上

k=sgn(cross(poly[(i+1)%n]-poly[i],p-poly[i]));

d1=sgn(poly[i].y-p.y);

d2=sgn(poly[(i+1)%n].y-p.y);

if(k>0&&d1<=0&&d2>0) cnt++;

if(k<0&&d2<=0&&d1>0) cnt--;

}

if(cnt) return 1;//内部

else return 0;//外部

}

**5.3** 多边形重心 (点集逆时针给出) bcenter

Point bcenter(vector<Point> p)

{

int i,j,n;

Point t,ans;

double tp,s,tpx,tpy;

s=t.x=t.y=0;

n=p.size();

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

{

if(i+1==n) j=0;

else j=i+1;

tp=cross(p[i],p[j]);

s+=tp/2;

t=t+(p[i]+p[j])\*tp;

}

ans=t/(6\*s);

return ans;

}

**5.4** 最近点对

//返回距离

Point p1[MAX],temp[MAX];

bool cmpxy(Point a,Point b)

{

if(a.x!=b.x) return a.x<b.x;

return a.y<b.y;

}

bool cmpy(Point a,Point b){return a.y<b.y;}

double ClosestPair(int l,int r)

{

int mid,i,j,k;

double d,d1,d2,d3;

d=INF;

if(l==r) return d;

if(l+1==r) return dist(p1[l],p1[r]);

mid=(l+r)>>1;

d1=ClosestPair(l,mid);

d2=ClosestPair(mid+1,r);

d=min(d1,d2);

k=0;

for(i=l;i<=r;i++)

{

if(fabs(p1[mid].x-p1[i].x)<=d) temp[k++]=p1[i];

}

sort(temp,temp+k,cmpy);

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

{

for(j=i+1;j<k&&temp[j].y-temp[i].y<d;j++)

{

d3=dist(temp[i],temp[j]);

if(d>d3) d=d3;

}

}

return d;

}

int main()

{

//............................

sort(p1,p1+n,cmpxy);

ans=ClosestPair(0,n-1);

//............................

}

**5.5** 模拟退火求费马点

double getres(Point t,Point \*p,int n)

{

int i;

double res=0;

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

{

res+=dist(t,p[i]);

}

return res;

}

pair<Point,double> SA(Point \*p,int n)

{

Point s;

double T=100,t,ds,dz,ans;

int i,j,k,flag,dir[5][2]={0,0,0,1,0,-1,1,0,-1,0};

s=Point(0,0);

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

{

s.x+=p[i].x;

s.y+=p[i].y;

}

s.x/=n;

s.y/=n;

ans=0;

for(k=1;k<=100000;k++)

{

t=T/k;//精度控制

for(i=0;i<5;i++)

{

Point z;

z.x=s.x+dir[i][0]\*t;

z.y=s.y+dir[i][1]\*t;

dz=getres(z,p,n);

if(dz<ans)

{

ans=dz;

s=z;

}

}

}

pair<Point,double> res;

res=make\_pair(s,ans);

return res;

}

**5.6 最小圆覆盖**

double r;

Point center;

void CircumCircle(Point a,Point b,Point c) //三角形外心

{

Point res;

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;

res=Point(a.x+(c1\*b2-c2\*b1)/d,a.y+(a1\*c2-a2\*c1)/d);

center=res;

}

void MinDisWith2Point(Point pi,Point pj,int m,vector<Point> p)

{

int k;

center=MidSeg(pi,pj);

r=dist(pi,pj)/2;

for(k=0;k<m;k++)

{

if(dist(center,p[k])<=r) continue;

double t=cross((pi-pj),(p[k]-pj));

if(t!=0)

{

CircumCircle(pi,pj,p[k]);

r=dist(center,pi);

}

else

{

double d1,d2,d3;

d1=dist(pi,pj);

d2=dist(pi,p[k]);

d3=dist(pj,p[k]);

if(d1>=d2&&d1>=d3)

{

center=MidSeg(pi,pj);

r=dist(pi,pj)/2;

}

else if(d2>=d1&&d2>=d3)

{

center=MidSeg(pi,p[k]);

r=dist(pi,p[k])/2;

}

else

{

center=MidSeg(pj,p[k]);

r=dist(pj,p[k])/2;

}

}

}

}

void MinDisWithPoint(Point pi,int m,vector<Point> p)

{

center=MidSeg(pi,p[0]);

r=dist(pi,p[0])/2;

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

{

if(dist(center, p[j])<=r) continue;

else MinDisWith2Point(pi,p[j],j,p);

}

}

int main()

{

//.................

//n 点的个数

vector<Point> p;

if(n==1)

{

printf("%.2lf %.2lf 0.00\n",p[0].x,p[0].y);

continue;

}

r=dist(p[0],p[1])/2;

center=MidSeg(p[0],p[1]);

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

{

if(dist(center,p[i])<=r) continue;

else MinDisWithPoint(p[i],i,p);

}

printf("%.2lf %.2lf %.2lf\n",center.x,center.y,r);

//...................

}

**5.7** 旋转卡壳

5.7.1 平面最远点对(凸包直径) RC\_FarthestPair

//先求凸包

//最远点对距离的平方

double dist2(Point a,Point b){return dot(a-b,a-b);}//距离的平方

double RC\_FarthestPair(vector<Point> p)

{

int n,i,j,k;

n=p.size();

if(n==1) return 0;

if(n==2) return dist2(p[0],p[1]);

double temp,res=0;

k=1;

for(j=1;j<n;j++)//定点

{

for(i=0;i<n;i++)//旋转

{

while(cross(p[i]-p[(i+j)%n],p[(k+1)%n]-p[k])<0) k=(k+1)%n;

temp=max(dist2(p[j],p[i]),dist2(p[j],p[k]));

res=max(res,temp);

}

}

return res;

}

5.7.2 点集最大三角形 RC\_Triangle

//先求凸包

//返回三角形面积

double RC\_Triangle(vector<Point> p)

{

int n,i,j,k;

n=p.size();

if(n<3) return 0;

double temp,res=0;

k=1;

for(j=1;j<n;j++)//定点

{

for(i=0;i<n;i++)//旋转

{

while(cross(p[i]-p[(i+j)%n],p[(k+1)%n]-p[k])<0) k=(k+1)%n;

temp=max(cross(p[(i+j)%n]-p[i],p[k]-p[i]),cross(p[(i+j)%n]-p[i],p[(k+1)%n]-p[i]));

res=max(res,temp);

}

}

return res/2;

}

5.7.3 两凸包最近距离 RC\_Dis

double RC\_Dis(vector<Point> p,vector<Point> q)

{

int i,j,k,n,m;

n=p.size();

m=q.size();

j=k=0;

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

{

if(p[i].y-p[k].y<-eps) k=i;

}

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

{

if(q[i].y-q[j].y>eps) j=i;

}

double temp,ans=1e9;

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

{

while((temp=cross(p[(k+1)%n]-p[k],q[(j+1)%m]-p[k])-cross(p[(k+1)%n]-p[k],q[j]-p[k]))>eps) j=(j+1)%m;

if(temp<-eps) ans=min(ans,DisPointToSeg(q[j],p[k],p[(k+1)%n]));

else ans=min(ans,DisSegToSeg(p[k],p[(k+1)%n],q[j],q[(j+1)%m]));

k=(k+1)%n;

}

return ans;

}

# **6.**圆

**6.1** 圆的定义

struct Circle

{

Point c;

double r;

Circle(){}

Circle(Point a,double b)

{

c=a;

r=b;

}

Point getpoint(double a) //根据圆心角求点坐标

{

return Point(c.x+cos(a)\*r,c.y+sin(a)\*r);

}

};

**6.2** 点P到圆的切线 getTangents

//切点存在v数组

int getTangents(Point p, Circle c, Vector\* v)

{

Vector u=c.c-p;

double d=length(u);

if(d<c.r) return 0;

else if(sgn(d-c.r)==0)

{

v[0]=Rotate(u,PI/2);

return 1;

}

else

{

double ang=asin(c.r/d);

v[0]=Rotate(u,-ang);

v[1]=Rotate(u,+ang);

return 2;

}

}

**6.3** 求a点到b点(逆时针)在的圆上的圆弧长度 ArcLen

double ArcLen(Circle c,Point a,Point b)

{

double ang1,ang2;

Vector v1,v2;

v1=a-Point(c.c.x,c.c.y);

v2=b-Point(c.c.x,c.c.y);

ang1=atan2(v1.y,v1.x);

ang2=atan2(v2.y,v2.x);

if(ang2<ang1) ang2+=2\*PI;

return c.r\*(ang2-ang1);

}

**6.4** 两圆的公切线 CommonTangents

//两圆的公切线, -1表示无穷条切线

//返回切线条数 切点保存在a b数组

int CommonTangents(Circle c1,Circle c2, Point\* a, Point\* b)

{

int cnt=0;

if(c1.r<c2.r)

{

swap(c1,c2);

swap(a,b);

}

int d=(c1.c.x-c2.c.x)\*(c1.c.x-c2.c.x)+(c1.c.y-c2.c.y)\*(c1.c.y-c2.c.y);

int rd=c1.r-c2.r;

int rs=c1.r+c2.r;

if(d<rd\*rd) return 0; //内含

double base=atan2(c2.c.y-c1.c.y,c2.c.x-c1.c.x);

if(d==0&&sgn(c1.r-c2.r)==0) return -1;//无线多条切线

if(d==rd\*rd)//内切, 1条切线

{

a[cnt]=c1.getpoint(base);

b[cnt]=c2.getpoint(base);

cnt++;

return 1;

}

//有外公切线

double ang=acos((c1.r-c2.r)/sqrt(d));

a[cnt]=c1.getpoint(base+ang);

b[cnt]=c2.getpoint(base+ang);

cnt++;

a[cnt]=c1.getpoint(base-ang);

b[cnt]=c2.getpoint(base-ang);

cnt++;

if(d==rs\*rs)//一条内公切线

{

a[cnt]=c1.getpoint(base);

b[cnt]=c2.getpoint(PI+base);

cnt++;

}

else if(d>rs\*rs)//两条内公切线

{

double ang=acos((c1.r+c2.r)/sqrt(d));

a[cnt]=c1.getpoint(base+ang);

b[cnt]=c2.getpoint(PI+base+ang);

cnt++;

a[cnt]=c1.getpoint(base-ang);

b[cnt]=c2.getpoint(PI+base-ang);

cnt++;

}

return cnt;

}

**6.5** 三角形外接圆 CircumscribedCircle

Circle CircumscribedCircle(Point p1,Point p2,Point p3)

{

double bx=p2.x-p1.x,by=p2.y-p1.y;

double cx=p3.x-p1.x,cy=p3.y-p1.y;

double d=2\*(bx\*cy-by\*cx);

double ex=(cy\*(bx\*bx+by\*by)-by\*(cx\*cx+cy\*cy))/d+p1.x;

double ey=(bx\*(cx\*cx+cy\*cy)-cx\*(bx\*bx+by\*by))/d+p1.y;

Point p=Point(ex,ey);

return Circle(p,length(p1-p));

}

**6.6** 三角形内切圆 InscribedCircle

Circle InscribedCircle(Point p1,Point p2,Point p3)

{

double a=length(p2-p3);

double b=length(p3-p1);

double c=length(p1-p2);

Point p=(p1\*a+p2\*b+p3\*c)/(a+b+c);

return Circle(p,DisPointToLine(p,p1,p2));

}

**6.7** 线段与圆的交点 getSegCircleInter

int getSegCircleInter(Line l,Circle c,Point \*sol)

{

Vector nor=Normal(l.v);

Line l1=Line(c.c,nor);

Point p1=PointOfLineInter(l1,l);

double d=length(p1-c.c);

if(sgn(d-c.r)>0) return 0;

Point p2=vecunit(l.v)\*sqrt(c.r\*c.r-d\*d);

int res=0;

sol[res]=p1+p2;

if(OnSeg(sol[res],l.p,l.getpoint(1))) res++;

sol[res]=p1-p2;

if(OnSeg(sol[res],l.p,l.getpoint(1))) res++;

return res;

}

**6.8** 直线与圆交点 getLineCircleInter

//直线与圆交点 返回个数

int getLineCircleInter(Line l,Circle cc,Point \*sol)

{

double a,b,c,d,e,f,g,delta,t;

a=l.v.x;

b=l.p.x-cc.c.x;

c=l.v.y;

d=l.p.y-cc.c.y;

e=a\*a+c\*c;

f=2\*(a\*b+c\*d);

g=b\*b+d\*d-cc.r\*cc.r;

delta=f\*f-4\*e\*g;

if(sgn(delta)<0) return 0;

if(sgn(delta)==0)

{

t=-f/(2\*e);

sol[0]=l.getpoint(t);

return 1;

}

else

{

t=(-f-sqrt(delta))/(2\*e);

sol[0]=l.getpoint(t);

t=(-f+sqrt(delta))/(2\*e);

sol[1]=l.getpoint(t);

return 2;

}

}

**6.9** 两圆交点 getCircleInter

//两圆交点 返回个数

int getCircleInter(Circle c1,Circle c2,Point \*sol)

{

double r1,r2,x1,y1,x2,y2,d;

r1=c1.r;

r2=c2.r;

x1=c1.c.x;

y1=c1.c.y;

x2=c2.c.x;

y2=c2.c.y;

d=length(c1.c-c2.c);

if(sgn(fabs(r1-r2)-d)>0) return -1;

if(sgn(r1+r2-d)<0) return 0;

double a,b,c,p,q,r;

a=r1\*(x1-x2)\*2;

b=r1\*(y1-y2)\*2;

c=r2\*r2-r1\*r1-d\*d;

p=a\*a+b\*b;

q=-a\*c\*2;

r=c\*c-b\*b;

double cosa,sina,cosb,sinb;

if(sgn(d-(r1+r2))== 0||sgn(d-fabs(r1-r2))==0)

{

cosa=-q/p/2;

sina=sqrt(1-cosa\*cosa);

Point p(x1+c1.r\*cosa,y1+c1.r\*sina);

if(!OnCircle(p,c2)) p.y=y1-c1.r\*sina;

sol[0]=p;

return 1;

}

double delta=sqrt(q\*q-p\*r\*4);

cosa=(delta-q)/p/2;

cosb=(-delta-q)/p/2;

sina=sqrt(1-cosa\*cosa);

sinb=sqrt(1-cosb\*cosb);

Point p1(x1+c1.r\*cosa,y1+c1.r\*sina);

Point p2(x1+c1.r\*cosb,y1+c1.r\*sinb);

if(!OnCircle(p1,c2)) p1.y=y1-c1.r\*sina;

if(!OnCircle(p2,c2)) p2.y=y1-c1.r\*sinb;

if(p1==p2) p1.y=y1-c1.r\*sina;

sol[0]=p1;

sol[1]=p2;

return 2;

}

**6.10 判断点在圆内** InCircle

bool InCircle(Point x,Circle c){return sgn(c.r-length(c.c-x))>=0;}//在圆内(包括圆上)

# **7.面积问题**

**7.1** 多边形面积 PolygonArea

double PolygonArea(Point \*p,int n)

{

double res=0;

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

{

res+=cross(p[i],p[(i+1)%n]);

}

return fabs(res/2);

}

**7.2** 凸多边形面积并 CPIA

//凸多边形面积并 ConvexPolygonInterArea

double CPIA(Point \*a,Point \*b,int na,int nb)

{

Point p[MAX],q[MAX];

int tn,sflag,eflag,i,j;

a[na]=a[0];

b[nb]=b[0];

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

{

p[i]=b[i];

}

for(i=0;i<na&&nb>2;i++)

{

sflag=sgn(cross(a[i+1]-a[i],p[0]-a[i]));

for(j=0,tn=0;j<nb;j++)

{

if(sflag>=0) q[tn++]=p[j];

eflag=sgn(cross(a[i+1]-a[i],p[j+1]-a[i]));

if((sflag^eflag)==-2) q[tn++]=PointOfLineInter(Line(a[i],a[i+1]-a[i]),Line(p[j],p[j+1]-p[j]));

sflag=eflag;

}

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

{

p[j]=q[j];

}

nb=tn;

p[nb]=p[0];

}

if(nb<3) return 0;

return PolygonArea(p,nb);

}

**7.3** 多边形面积并 SPIA

//多边形面积并 SimplePolygonInterArea

double SPIA(Point \*a,Point \*b,int na,int nb)

{

int i,j,temp1,temp2;

Point t1[4],t2[4];

double res=0;

a[na]=t1[0]=a[0];

b[nb]=t2[0]=b[0];

for(i=2;i<na;i++)

{

t1[1]=a[i-1];

t1[2]=a[i];

temp1=sgn(cross(t1[1]-t1[0],t1[2]-t1[0]));

if(temp1<0) swap(t1[1],t1[2]);

for(j=2;j<nb;j++)

{

t2[1]=b[j-1];

t2[2]=b[j];

temp2=sgn(cross(t2[1]-t2[0],t2[2]-t2[0]));

if(temp2<0) swap(t2[1],t2[2]);

res+=CPIA(t1,t2,3,3)\*temp1\*temp2;

}

}

return res;

}

**7.4** 多边形与圆面积交 PolyCiclrArea

//多边形与圆面积交

Point PointOfLineInter(Line a,Line b)//线段交点

{

Vector u=a.p-b.p;

double t=cross(b.v,u)/cross(a.v,b.v);

return a.p+a.v\*t;

}

bool InCircle(Point x,Circle c){return sgn(c.r-length(c.c-x))>=0;}//在圆内(包括圆上)

int getSegCircleInter(Line l,Circle c,Point \*sol)//线段与圆的交点

{

Vector nor=Normal(l.v);

Line l1=Line(c.c,nor);

Point p1=PointOfLineInter(l1,l);

double d=length(p1-c.c);

if(sgn(d-c.r)>0) return 0;

Point p2=vecunit(l.v)\*sqrt(c.r\*c.r-d\*d);

int res=0;

sol[res]=p1+p2;

if(OnSeg(sol[res],l.p,l.getpoint(1))) res++;

sol[res]=p1-p2;

if(OnSeg(sol[res],l.p,l.getpoint(1))) res++;

return res;

}

double VectorAngle(Vector v){return atan2(v.y,v.x);}

double SegCircleArea(Circle c,Point a,Point b) //线段切割圆

{

double a1=VectorAngle(a-c.c);

double a2=VectorAngle(b-c.c);

double da=fabs(a1-a2);

if (sgn(da-PI)>0) da=PI\*2.0-da;

return sgn(cross(b-c.c,a-c.c))\*da\*c.r\*c.r/2.0;

}

double PolyCiclrArea(Circle c,Point \*p,int n)//多边形与圆面积交

{

double res=0;

Point sol[2];

p[n]=p[0];

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

{

double t1,t2;

int cnt=getSegCircleInter(Line(p[i],p[i+1]-p[i]),c,sol);

if(cnt==0)

{

if(!InCircle(p[i],c)||!InCircle(p[i+1],c)) res+=SegCircleArea(c,p[i],p[i+1]);

else res+=cross(p[i+1]-c.c,p[i]-c.c)/2.0;

}

if(cnt==1)

{

if(InCircle(p[i],c)&&!InCircle(p[i+1],c))

{

res+=cross(sol[0]-c.c,p[i]-c.c)/2.0;

res+=SegCircleArea(c,sol[0],p[i+1]);

}

else

{

res+=SegCircleArea(c,p[i],sol[0]);

res+=cross(p[i+1]-c.c,sol[0]-c.c)/2.0;

}

}

if(cnt==2)

{

if((p[i]<p[i+1])^(sol[0]<sol[1])) swap(sol[0],sol[1]);

res+=SegCircleArea(c,p[i],sol[0]);

res+=cross(sol[1]-c.c,sol[0]-c.c)/2.0;

res+=SegCircleArea(c,sol[1],p[i+1]);

}

}

return fabs(res);

}

**7.5** 两圆面积交 CircleCircleArea

double CircleCircleArea(Circle c1,Circle c2)

{

double d=dist(c1.c,c2.c);//计算圆心距

if(sgn(d-(c1.r+c2.r))>=0||sgn(c1.r)==0||sgn(c2.r)==0) return 0;//相离、外切或有一圆半径为0

else if(sgn(d-fabs(c1.r-c2.r))<=0)//内切或内含

{

double r=min(c1.r,c2.r);

return r\*r\*PI;

}

else//相交

{

double a,b,m,n,x,y;

a=acos((d\*d+c1.r\*c1.r-c2.r\*c2.r)/(2\*d\*c1.r));

b=acos((d\*d+c2.r\*c2.r-c1.r\*c1.r)/(2\*d\*c2.r));

m=a\*c1.r\*c1.r;

n=b\*c2.r\*c2.r;

x=c1.r\*c1.r\*sin(a)\*cos(a);

y=c2.r\*c2.r\*sin(b)\*cos(b);

return m+n-(x+y);

}

}

# **8.**Pick定理

Pick定理：一个计算点阵中顶点在格点上的多边形面积公式：S=a+b/2-1，其中a表示多边形内部的点数，b表示多边形边界上的点数，S表示多边形的面积。   
那么a=(2S-b+2)/2   
S可以通过叉积求出   
计算b的方法是：某一条边(x,y)上的整点的数量是gcd(|x|,|y|)+1