计算几何

1.基础知识

向量的点积叉积。在涉及二维平面的题目上，叉积的方向只有z轴向上以及向下。故可以省略地用有符号数字来表示。

模板

```
#define eps 1e-8
#define MAXN 1e9
#define defAng auto si=sin(angle);\
    auto co=cos(angle);
class vec
{
public:
    vec(double xx=0,double yy=0):x(xx),y(yy)
    {
    }
    double x,y;
    double len()
    {
        return sqrt(x*x+y*y);
    }
    bool operator ==(const vec &a)
    {
        return a.x==x&&a.y==y;
    }
    vec unit()//单位向量
    {
        auto l=len();
        return vec(x/l,y/l);
     }
    vec normal()//单位法向量
    {
        return vec(-y,x).unit();
    }
    void debug()
    {
        printf("%lf %lf\n",x,y);
    }
 };
typedef vec point;
int dcmp(double x)
{
    if (fabs(x)<eps) return 0;
    else return x<0?-1:1;
```

```
}
double sqr(double x)
{
    return x*x;
}
vec operator +(vec a,vec b)
{
    return vec(a.x+b.x,a.y+b.y);
}
vec operator -(vec a,vec b)
{
    return vec(a.x-b.x,a.y-b.y);
}
vec operator *(vec a,double b)
{
    return vec(a.x*b,a.y*b);
}
vec operator /(vec a,double b)
{
    return vec(a.x/b,a.y/b);
}
double dot(vec a,vec b)
{
    return a.x*b.x+a.y*b.y;
}
double cross(vec a,vec b)
{
    return a.x*b.y-a.y*b.x;
}
double angle(vec a,vec b)
{
    return acos(dot(a,b)/a.len()/b.len());
}
vec rotate(vec a,double angle)//逆时针旋转
{
    defAng
    return vec(co,si)*a.x+(vec(-si,co))*a.y;
}
```

2.基本操作

点到直线的距离

利用叉积求平行四边形的面积然后初以底即可

点到线段的距离

点到直线的距离以及点到线段两点距离的最小值

判断两线段是否相交(快速排斥实验和跨立实验)

快速排斥实验：分别以两线段为对角线构成的两个矩形假如不相交，则两条线段肯定不相交

跨立实验：分别计算其中一条直线的两个端点与另一条直线两端点的叉积的乘积，假如小于等于0，则两直线相交（充要条件）

两直线的交点

暴力解方程即可

多边形的面积

可以计算多边形两两顶点与某一点所围成的三角形面积和。由于叉积的方向性，所选的点任意，故直接选取远点即可。

点是否在多边形内

充要条件是引射线与多边形的交点为奇数，为了方便，可以做一条指向很远横坐标的线段。

三角形内心

解析几何方法解方程计算出坐标即可

凸包□andrew算法）

把所有点以横坐标为第一关键字，以纵坐标为第二关键字排序，然后根据逆时针遍历凸包上的边只能左转的规律（两边向量叉积小于零），先正序扫描一遍求出下半凸壳，再逆序扫描一遍求出上半凸壳。时间复杂度 $O(n\log n)$

最小圆覆盖

依次遍历各点，假如当前点 i 不在最小圆内，则更新后的最小圆必经过该点，然后遍历从 1 到 $i-1$ 的点，假如有点 j 不在最小圆内，则更新后的最小圆也必经过该点，然后再从 1 到 $j-1$ 遍历各点，重复上述操作，由于三点确定圆，所以得到的就是当前最小圆。

时间复杂度 $O(n)$ □因为第二三次遍历的均摊时间复杂度均为 $O(\frac{3}{n})$ □可以忽略不记。

平面最近点对

利用分治的办法，先把平面分为两半，假设此时求得的最小距离为 h □再考虑两平面中横坐标相距小于 h 的情况。

3.相关代码

```
double distl(point p,point a,point b)//点到直线距离
{
    vec line=b-a;
    return fabs(cross(p-a,line))/line.len();
}
double distr(point p,point a,point b)//点到线段距离
{
    if (a==b)
    return (p-a).len();
    vec l1=p-a;
    vec l2=p-b;
    vec l=b-a;
    if (dcmp(dot(l1,l)<0))
    return l1.len();
    if (dcmp(dot(l2,l)<0))
    return l2.len();
    return fabs(cross(l1,l))/l.len();
 }
double iscrossr(point a,point b,point c,point d)//判断线段是否相交(快速排斥实验和跨立实验)
{
    if
(!(min(a.x,b.x)<max(c.x,d.x)&&min(a.y,b.y)<max(c.y,d.y)&&min(c.y,d.y)<max(a.y,b.y)&&min(c.x,d.x)<max(a.x,b.x)))
        return false;
    return dcmp(cross(b-a,c-a)*dcmp(cross(b-a,d-a)))<=0&&dcmp(cross(d-c,a-c))*dcmp(cross(d-c,b-c))<=0;
 }
vec getcross(point a,point b,point c,point d)//两直线的交点，直接解方程即可
{
    double c1=cross(a,b);
    double c2=cross(c,d);
    return vec(c1*(c.x-d.x)-c2*(a.x-b.x),c1*(c.y-d.y)-c2*(a.y-b.y))/((a.x-b.x)*(c.y-d.y)-(a.y-b.y)*(c.x-d.x));
}
double polyS(vector<point> p)//多边形面积
{
    auto size=p.size();
    double ans=0;
    for (int i=1;i<size-1;i++)
    ans+=cross(p[i]-p[0],p[i+1]-p[0]);
    return ans;
 }
bool inpoly(point a,vector<point> &p)  //点是否在多边形内（引射线与多边形相交的点数为奇数）
{
    auto size=p.size();
```

```
    int cnt=0;
    for (int i=0;i<size;i++)
    {
        auto &p1=p[i];
        auto &p2=p[(i+1)%size];
        if (iscrossr(p1,p2,0,MAXN))
            cnt++;
    }
    return cnt%2;
}
vector<point> andrew(vector<point> &p)
{
    sort(p.begin(),p.end(),[](point &a,point &b){
        return a.x<b.x||(a.x==b.x&&a.y<b.y);
    });
    auto size=p.size();
    vector<point> ans(2*size);
    int k=0;
    for (int i=0;i<size;i++)
    {
        while (k>=2&&dcmp(cross(ans[k-1]-ans[k-2],p[i]-ans[k-2])<0))
        k--;
        ans[k++]=p[i];
    }
    for (int i=size-2,t=k+1;i>=0;i--)
    {
        while (k>=t&&dcmp(cross(ans[k-1]-ans[k-2],p[i]-ans[k-2])<0))
        k--;
        ans[k++]=p[i];
    }
    ans.resize(k-1);
    return ans;
}
point geto(point a, point b, point c) {
  double a1, a2, b1, b2, c1, c2;
  point ans;
  a1 = 2 * (b.x - a.x), b1 = 2 * (b.y - a.y),
  c1 = sqr(b.x) - sqr(a.x) + sqr(b.y) - sqr(a.y);
  a2 = 2 * (c.x - a.x), b2 = 2 * (c.y - a.y),
  c2 = sqr(c.x) - sqr(a.x) + sqr(c.y) - sqr(a.y);
  if (dcmp(a1)==0) {
    ans.y = c1 / b1;
    ans.x = (c2 - ans.y * b2) / a2;
  } else if (dcmp(b1)==0) {
    ans.x = c1 / a1;
    ans.y = (c2 - ans.x * a2) / b2;
  } else {
    ans.x = (c2 * b1 - c1 * b2) / (a2 * b1 - a1 * b2);
    ans.y = (c2 * a1 - c1 * a2) / (b2 * a1 - b1 * a2);
  }
  return ans;
```

```
}
```

4.例题

最小圆覆盖

```
#include <iostream>
#include <stdio.h>
#include <algorithm>
#include <functional>
#include <stack>
#include <vector>
#include <cstring>
#include <queue>
#include <cmath>
#include <map>
using namespace std;
#define eps 1e-8
#define MAXN 1e9
#define defAng auto si=sin(angle);\
    auto co=cos(angle);
class vec
{
public:
    vec(double xx=0,double yy=0):x(xx),y(yy)
    {
    }
    double x,y;
    double len()
    {
        return sqrt(x*x+y*y);
    }
    bool operator ==(const vec &a)
    {
        return a.x==x&&a.y==y;
    }
    vec unit()//单位向量
    {
        auto l=len();
        return vec(x/l,y/l);
     }
    vec normal()//单位法向量
    {
        return vec(-y,x).unit();
    }
    void debug()
```

```
    {
        printf("%lf %lf\n",x,y);
    }
 };
typedef vec point;
int dcmp(double x)
{
    if (fabs(x)<eps) return 0;
    else return x<0?-1:1;
}
double sqr(double x)
{
    return x*x;
}
vec operator +(vec a,vec b)
{
    return vec(a.x+b.x,a.y+b.y);
}
vec operator -(vec a,vec b)
{
    return vec(a.x-b.x,a.y-b.y);
}
vec operator *(vec a,double b)
{
    return vec(a.x*b,a.y*b);
}
vec operator /(vec a,double b)
{
    return vec(a.x/b,a.y/b);
}
double dot(vec a,vec b)
{
    return a.x*b.x+a.y*b.y;
}
double cross(vec a,vec b)
{
    return a.x*b.y-a.y*b.x;
}
double angle(vec a,vec b)
{
    return acos(dot(a,b)/a.len()/b.len());
}
vec rotate(vec a,double angle)//逆时针旋转
{
    defAng
    return vec(co,si)*a.x+(vec(-si,co))*a.y;
}
double distl(point p,point a,point b)//点到直线距离
{
    vec line=b-a;
    return fabs(cross(p-a,line))/line.len();
```

```
}
double distr(point p,point a,point b)//点到线段距离
{
    if (a==b)
    return (p-a).len();
    vec l1=p-a;
    vec l2=p-b;
    vec l=b-a;
    if (dcmp(dot(l1,l)<0))
    return l1.len();
    if (dcmp(dot(l2,l)<0))
    return l2.len();
    return fabs(cross(l1,l))/l.len();
 }
double iscrossr(point a,point b,point c,point d)//判断线段是否相交(快速排斥实验和跨立实验)
{
    if
(!(min(a.x,b.x)<max(c.x,d.x)&&min(a.y,b.y)<max(c.y,d.y)&&min(c.y,d.y)<max(a.y,b.y)&&min(c.x,d.x)<max(a.x,b.x)))
        return false;
    return dcmp(cross(b-a,c-a)*dcmp(cross(b-a,d-a)))<=0&&dcmp(cross(d-c,a-c))*dcmp(cross(d-c,b-c))<=0;
 }
vec getcross(point a,point b,point c,point d)//两直线的交点，直接解方程即可
{
    double c1=cross(a,b);
    double c2=cross(c,d);
    return vec(c1*(c.x-d.x)-c2*(a.x-b.x),c1*(c.y-d.y)-c2*(a.y-b.y))/((a.x-b.x)*(c.y-d.y)-(a.y-b.y)*(c.x-d.x));
}
point geto(point a, point b, point c) {
  double a1, a2, b1, b2, c1, c2;
  point ans;
  a1 = 2 * (b.x - a.x), b1 = 2 * (b.y - a.y),
  c1 = sqr(b.x) - sqr(a.x) + sqr(b.y) - sqr(a.y);
  a2 = 2 * (c.x - a.x), b2 = 2 * (c.y - a.y),
  c2 = sqr(c.x) - sqr(a.x) + sqr(c.y) - sqr(a.y);
  if (dcmp(a1)==0) {
    ans.y = c1 / b1;
    ans.x = (c2 - ans.y * b2) / a2;
  } else if (dcmp(b1)==0) {
    ans.x = c1 / a1;
    ans.y = (c2 - ans.x * a2) / b2;
  } else {
    ans.x = (c2 * b1 - c1 * b2) / (a2 * b1 - a1 * b2);
    ans.y = (c2 * a1 - c1 * a2) / (b2 * a1 - b1 * a2);
  }
  return ans;
}
```

```
int main()
{
    vector<point> p;
    int n;
    double r=0;
    scanf("%d",&n);
    for (int i=1;i<=n;i++)
    {
        double x,y;
        scanf("%lf%lf",&x,&y);
        p.emplace_back(point(x,y));
    }
    random_shuffle(p.begin(),p.end());
    auto o=p[0];
    for (int i=0;i<n;i++)
    {
        if (dcmp((o-p[i]).len()-r)<=0)
        continue;
        o.x=(p[0].x+p[i].x)/2;
        o.y=(p[0].y+p[i].y)/2;
        r=(p[i]-p[0]).len()/2;
        for (int j=1;j<i;j++)
        {
            if (dcmp((o-p[j]).len()-r)<=0)
            continue;
            o.x=(p[i].x+p[j].x)/2;
            o.y=(p[i].y+p[j].y)/2;
            r=(p[j]-p[i]).len()/2;
            for (int k=0;k<j;k++)
            {
                if (dcmp((o-p[k]).len()-r)<=0)
                continue;
                o = geto(p[i], p[j], p[k]);
                r = (o-p[i]).len();
            }
        }
    }
     printf("%.10lf\n%.10lf %.10lf", r, o.x, o.y);
}
```