前置代码

```
namespace pre{
  using namespace std;
  #define io ios::sync_with_stdio(false);cin.tie(0);
  typedef long long ll;
  typedef long double ld;
  const int inf=(1<<30);
  const long double eps=1e-8;
  const long double pi=acos((long double)-1.0);
  typedef long long ll;
  const ll mod=1e9+7;
  const ll mod2=998244353;
  const ll mod3=1004535809;
  const int maxn=1e6+5;
  const ll linf=1LL<<62;
  inline bool _zero(double x){return (((x)>0?(x):-(x))<eps);}
  inline double _sq(double x){return x*x;}
  inline ll sqr(ll a){return a*a;}
  inline int _sign(double x){return ((x)>eps?1:((x)<-eps?2:0));}
}
```

点

note：旋转的函数需要注意精度问题

类定义

```
using namespace pre;
  struct Point{//point(x,y)
      double x,y;
      Point():x(0),y(0){}
      Point(double x,double y): x(x),y(y) {}
      friend istream& operator >>(istream &in,Point &p){
      in>>p.x>>p.y;
      return in;
    }
    friend ostream& operator <<(ostream &out,const Point &p){
      out<<p.x<<" "<<p.y;
      return out;
    }
      Point operator + (Point b) const {return Point(x+b.x,y+b.y);}
      Point operator - (Point b) const {return Point(x-b.x,y-b.y);}
      Point operator * (double b) const {return Point(b*x,b*y);}
      double operator * (Point b) const {return x*b.y-y*b.x;} //叉乘
      double operator & (Point b) const {return x*b.x+y*b.y;} //点乘
```

```
      double distance() const {return sqrt(_sq(x)+_sq(y));}
      Point unit() const {return Point(x/distance(),y/distance());}
      bool operator < (const Point &b) const{//极角排序，[-pi,pi)开始逆时针排序
      return (*this)*b>eps || _zero((*this)*b) && distance()<b.distance();
    }
    static bool comp(const Point &a,const Point &b){//坐标排序，下到上，左到右
      return a.y<b.y || a.y==b.y && a.x<b.x;
    }
      double distance(Point p) const {
      return (*(this)-p).distance();
    }
      double distance(Point p1,Point p2) const {
      return (p2-p1).distance();
    }
      double xmult(Point p1,Point p2) const {//向量this->p1叉乘this->p2
      return (p1-(*this))*(p2-(*this));
    }
      double dmult(Point p1,Point p2) const {//向量this->p1点乘this->p2
      return (p1-(*this))&(p2-(*this));
    }
      bool dots_inline(Point p1,Point p2) const{//三点共线
          return _zero(xmult(p1,p2));
    }
      Point rotate(Point p,double angle,double scale){ //p绕逆时针旋转angle，放大或缩小至
scale
          Point ret=p;
          x-=p.x,y-=p.y;
          p.x=scale*cos(angle);
          p.y=scale*sin(angle);
          ret.x+=x*p.x-y*p.y;
          ret.y+=x*p.y+y*p.x;
          return ret;
      }
      double rag(Point a,Point b)const{//a->this->b夹角
      return acos((_sq(distance(a))+_sq(distance(b))-
_sq(a.distance(b)))/(2*distance(a)*distance(b)));
    }
    Point pedal() const{  //顺时针转90度
      return Point(y,-x);
    }
  };
```

极角定义

```
struct Polar//polar(radius,theta)
  {
      double radius,theta;
      Polar():radius(0),theta(0){}
```

```
      Polar(double theta):radius(1),theta(theta){}
      Polar(double radius,double theta):radius(radius),theta(theta){}
      Polar(Point p):radius(p.distance()),theta(atan2(p.y,p.x)){}
      friend istream& operator >>(istream &in,Polar &p){
      in>>p.radius>>p.theta;
      return in;
    }
    friend ostream& operator <<(ostream &out,const Polar &p){
      out<<p.radius<<" "<<p.theta;
      return out;
    }
      bool operator < (const Polar &b)const{return theta<b.theta || theta==b.theta &&
radius<b.radius;}
      Point topoint(){  //和x正半轴夹角，(0,pi)在y>0区域，(-pi,0)在y<0区域
      return Point(radius*cos(theta),radius*sin(theta));
    }
      Polar unit(){ //unit circle
      return Polar(theta);
    }
  };
```

线

类定义

```
struct Line{//line
      Point a,b;
      Line(): a(Point()),b(Point()){}
      Line(Point a,Point b): a(a),b(b) {}
      friend istream& operator >>(istream &in,Line &l){
      in>>l.a>>l.b;
      return in;
    }
    friend ostream& operator <<(ostream &out,const Line &l){
      out<<l.a<<" "<<l.b;
      return out;
    }
      Point midpoint() const {//mid point
      return (a+b)*0.5;
    }
      double distance() const {//length
      return a.distance(b);
    }
      bool dot_inline(Point p) {//if point in line
      return _zero(p.xmult(a,b));
    }
```

```
      bool dot_online(Point p) {//if point in segment
      return _zero(p.xmult(a,b))&&(a.x-p.x)*(b.x-p.x)<eps&&(a.y-p.y)*(b.y-p.y)<eps;
    }
      bool dot_online_in(Point p){//if point in segment except two point
      return dot_online(p)&&(!_zero(p.x-a.x)||!_zero(p.y-a.y))&&(!_zero(p.x-
b.x)||!_zero(p.y-b.y));
    }

  //提前判断点在线上
      bool same_side(Point p1,Point p2){//if two points on same side except line
      return a.xmult(b,p1)*a.xmult(b,p2)>eps;
    }
      bool opposite_side(Point p1,Point p2){//if two points on opposite side except
line
      return a.xmult(b,p1)*a.xmult(b,p2)<-eps;
    }

      bool parallel(Line v){//if two lines parallel
      return _zero(Point().xmult(a-b,v.a-v.b));
    }
    bool parallel_ex(Line v){//if two lines parallel and not coincidence 不重合
      return _zero(Point().xmult(a-b,v.a-v.b))&&!dot_inline(v.a);
    }
      bool perpendicular(Line v){//if two lines perpendicular 两线垂直
      return _zero(Point().dmult(a-b,v.a-v.b));
    }
      bool intersect(Line v){//if two segments have same points 线段是否相交
          return !same_side(v.a,v.b)&&!v.same_side(a,b);
      }
      bool intersect_ex(Line v){//if two segments have only same point except end
points
      return opposite_side(v.a,v.b)&&v.opposite_side(a,b);
    }
      Point intersection(Line v){//calculate the intersection of two lines,please
notice they are not parallel
          Point ret=a;
          double t=((a.x-v.a.x)*(v.a.y-v.b.y)-(a.y-v.a.y)*(v.a.x-v.b.x))
                  /((a.x-b.x)*(v.a.y-v.b.y)-(a.y-b.y)*(v.a.x-v.b.x));
          return a+(b-a)*t;
      }
      Point pedal(Point p){//calculate the nearest point to the line,pedal  直线到点最短距
离
          Point t=p;
          t.x+=a.y-b.y,t.y+=b.x-a.x;
          if(dot_inline(p)) return p;
          return intersection(Line(p,t));
      }
      double distance(Point p){//calculate the distance from point to line
      return fabs(b.xmult(p,a))/distance();
```

```
    }
      Point ptoseg(Point p){//calculate the nearest point to the segment    线段到点最短距离
          Point t=p;
          t.x+=a.y-b.y,t.y+=b.x-a.x;
          if (p.xmult(a,t)*p.xmult(b,t)>eps)
              return a.distance(p)<b.distance(p)?a:b;
          return pedal(p);
      }
      double disptoseg(Point p){//calculate the distance from point to segment  点到线段距离
          Point t=p;
          t.x+=a.y-b.y,t.y+=b.x-a.x;
          if (p.xmult(a,t)*p.xmult(b,t)>eps)
              return Line(p,a).distance()<Line(p,b).distance()?Line(p,a).distance():Line(p,b).distance();
          return fabs(b.xmult(p,a))/distance();
      }
      Line midperpendicular(){    //中垂线
        return Line(midpoint(),midpoint()+(b-a).pedal());
    }
    Point definite_point(double rate){  //a往b方向rate倍的点
      Point v=b-a;
      return a+v*rate;
    }
  };
```

一般式表示

```
struct General_line// ax+by+c=0 for general line expression exp
  {
      double a,b,c;
      General_line():a(0),b(0),c(0){}
      General_line(double a,double b,double c):a(a),b(b),c(c){}
      General_line(Line l):a((l.a-l.b).y),b(-(l.a-l.b).x),c(-a*l.a.x-b*l.a.y){}
      friend istream& operator >>(istream &in,General_line &l){
      in>>l.a>>l.b>>l.c;
      return in;
    }
    friend ostream& operator <<(ostream &out,const General_line &l){
      out<<l.a<<" "<<l.b<<" "<<l.c;
      return out;
    }
  //点到直线对称点
      Point symmetricalPointofLine(Point p){//calculate symmetric point of the line
          double d=_sq(a)+_sq(b);
          return Point((_sq(b)*p.x-_sq(a)*p.x-2*a*b*p.y-2*a*c)/d,(_sq(a)*p.y-_sq(b)*p.y-2*a*b*p.x-2*b*c)/d);
```

```
    }
};
```

圆

类定义

```
struct Circle//circle(point,radius)
  {
      Point center;
      double radius;
      Circle():center(Point()),radius(1){}
      Circle(Point center,double radius):center(center),radius(radius){}
      Circle(Line l): center(l.midpoint()),radius(l.distance()/2){}//diameter for a
circle
      friend istream& operator >>(istream &in,Circle &c){
      in>>c.center>>c.radius;
      return in;
    }
    friend ostream& operator <<(ostream &out,const Circle &c){
      out<<c.center<<" "<<c.radius;
      return out;
    }
      Circle(Point p1,Point p2,Point p3){//three points for a circle
        Line u,v;
          u=Line(p1,p2).midperpendicular();
          v=Line(p1,p3).midperpendicular();
      center=u.intersection(v);
      radius=center.distance(p1);
    }
      double area() const{//the area

      return pi*_sq(radius);
    }
      double perimeter(){//the perimeter  周长
        return 2.0*pi*radius;
    }
      bool inside_circle(Point p){//if point in circle
      return center.distance(p)<radius+eps;
    }
      bool intersect(Line l){//if line and circle have intersection
      return l.distance(center)<radius+eps;
    }
      bool intersect_ex(Line l){//if segment and circle have intersection
          double t1=center.distance(l.a)-radius,t2=center.distance(l.b)-radius;
          Point t=center;
          if (t1<eps||t2<eps)
              return t1>-eps||t2>-eps;
```

```
            t.x+=l.a.y-l.b.y;
            t.y+=l.b.x-l.a.x;
            return t.xmult(l.a,center)*t.xmult(l.b,center)<eps&&l.distance(center)-
radius<eps;
        }
        bool intersect(Circle c2){//if two circles have intersection
        return center.distance(c2.center)
<radius+c2.radius+eps&&center.distance(c2.center)>fabs(radius-c2.radius)-eps;
    }
        int circleposition(Circle c2){  //relative location of two circles
            double d=center.distance(c2.center),rs=radius+c2.radius,rd=radius-c2.radius;
            if (_zero(d)&&_zero(rd)) return -2;//coincidence  重合
            if (_zero(rs-d)) return 4;//circumscribed   外切
            if (_zero(rd-d)) return 3;//inscribe and c2 in c    两种内切
            if (_zero(rd+d)) return -3;//inscribe and c in c2
            if (d>rs) return 0;//separation of two circles from outside 外离
            if (intersect(c2)) return 2;//circle intersection   相交
            if (rd>0) return 1;//separation of two circles and c2 in c  两种内离
            return -1; //separation of two circles and c in c2
        }
    double intersection_area(Circle c2){    //圆交面积
        double dis=center.distance(c2.center);
        if(radius+c2.radius<=dis)
              return 0.0;
          if(radius-c2.radius>=dis)
              return c2.area();
          if(c2.radius-radius>=dis)
              return area();
          double angle1 =acos((_sq(radius)+_sq(dis)-_sq(c2.radius))/(2*dis*radius));
          double angle2 =acos((_sq(c2.radius)+_sq(dis)-_sq(radius))/(2*dis*c2.radius));
          return _sq(radius)*angle1+_sq(c2.radius)*angle2-sin(angle1)*radius*dis;
    }
    Point intersection(Circle c2){//use if have intersection and return one of it
        double dis=center.distance(c2.center);
        Point p1=(center+c2.center)*0.5,p2=(c2.center-center)*0.5;
        double i1=(_sq(radius)-_sq(c2.radius))/(_sq(dis)),i2=sqrt(2*
(_sq(radius)+_sq(c2.radius))/(_sq(dis))-_sq(i1)-1);
        //另外一点为p1+p2*i1-Point(p2.y,p2.x)*i2, 可以配合circlepostion使用
        return p1+p2*i1+Point(p2.y,p2.x)*i2;
    }
    double distance(Point p){
        double dis=p.distance(center)-radius;
        if(dis<0) return 0.0;
        else return dis;
    }
    double distance(Line l){
        double dis=l.distance(center)-radius;
        if(dis<0) return 0.0;
        else return dis;
```

```
    }
    double disptoseg(Line l){
      double dis=l.disptoseg(center)-radius;
      if(dis<0) return 0.0;
      else return dis;
    }
  };
```

三角形

类定义

```
struct Triangle{//triangle
      vector<Point> p;
      Triangle(){p.resize(3);}
      Triangle(Point a,Point b,Point c){p.resize(3);p[0]=a;p[1]=b;p[2]=c;}
      double area()//the area
      {
          return fabs(p[0].xmult(p[1],p[2]))/2;
      }
      Point perpencenter()//the perpencenter,orthocenter of a triangle  垂心，三条高的交点
      {
          Line u,v;
          u.a=p[2];
          u.b=Line(p[0],p[1]).pedal(p[2]);
          v.a=p[1];
          v.b=Line(p[0],p[2]).pedal(p[1]);
          return u.intersection(v);
      }
    //重心，三条中线交点
      Point barycenter()//the barycenter of triangle,it's the point of lowest sum of
square of distance to three points,it's the point of largest product of square of
distance to three points in triangle
      {
          Line u,v;
          u.a=Line(p[0],p[1]).midpoint();
          u.b=p[2];
          v.a=Line(p[0],p[2]).midpoint();
          v.b=p[1];
          return u.intersection(v);
      }

      Point circumcenter()//excenter of a triangle 外心，三条中垂线的交点，外接圆圆心
      {
          Point tmp;
      tmp.x=p[0].x+(0.5*(_sq(p[1].x-p[0].x)+_sq(p[1].y-p[0].y))*(p[2].y-p[0].y)-0.5*
(_sq(p[2].x-p[0].x)+_sq(p[2].y-p[0].y))*(p[1].y-p[0].y))/((p[1].x-p[0].x)*(p[2].y-
p[0].y)-(p[2].x-p[0].x)*(p[1].y-p[0].y));
```

```
      tmp.y=p[0].y+(0.5*(_sq(p[2].x-p[0].x)+_sq(p[2].y-p[0].y))*(p[1].x-p[0].x)-0.5*
(_sq(p[1].x-p[0].x)+_sq(p[1].y-p[0].y))*(p[2].x-p[0].x))/((p[1].x-p[0].x)*(p[2].y-
p[0].y)-(p[2].x-p[0].x)*(p[1].y-p[0].y));
          return tmp;
      }
      Point incenter()//incenter of a triangle  内心，内接圆圆心
      {
          Line u,v;
          double m,n;
          u.a=p[0];
          m=atan2(p[1].y-p[0].y,p[1].x-p[0].x);
          n=atan2(p[2].y-p[0].y,p[2].x-p[0].x);
          u.b.x=u.a.x+cos((m+n)/2);
          u.b.y=u.a.y+sin((m+n)/2);
          v.a=p[1];
          m=atan2(p[0].y-p[1].y,p[0].x-p[1].x);
          n=atan2(p[2].y-p[1].y,p[2].x-p[1].x);
          v.b.x=v.a.x+cos((m+n)/2);
          v.b.y=v.a.y+sin((m+n)/2);
          return u.intersection(u);
      }
      Point fermetpoint(){//fermet point,the point of lowest sum of distance to three
points  费马点

          Point u,v;
          double
step=fabs(p[0].x)+fabs(p[0].y)+fabs(p[1].x)+fabs(p[1].y)+fabs(p[2].x)+fabs(p[2].y);
          int i,j,k;
          u.x=(p[0].x+p[1].x+p[2].x)/3;
          u.y=(p[0].y+p[1].y+p[2].y)/3;
          while (step>eps)
              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
(u.distance(p[0])+u.distance(p[1])+u.distance(p[2])>v.distance(p[0])+v.distance(p[1])+v
.distance(p[2]))
                              u=v;
                      }
          return u;
      }
      Circle circumcircle(){//the circumcircle of the triangle  外接圆

          Circle tmp;
      tmp.center.x=p[0].x+(0.5*(_sq(p[1].x-p[0].x)+_sq(p[1].y-p[0].y))*(p[2].y-
p[0].y)-0.5*(_sq(p[2].x-p[0].x)+_sq(p[2].y-p[0].y))*(p[1].y-p[0].y))/((p[1].x-p[0].x)*
(p[2].y-p[0].y)-(p[2].x-p[0].x)*(p[1].y-p[0].y));
```

```
        tmp.center.y=p[0].y+(0.5*(_sq(p[2].x-p[0].x)+_sq(p[2].y-p[0].y))*(p[1].x-
p[0].x)-0.5*(_sq(p[1].x-p[0].x)+_sq(p[1].y-p[0].y))*(p[2].x-p[0].x))/((p[1].x-p[0].x)*
(p[2].y-p[0].y)-(p[2].x-p[0].x)*(p[1].y-p[0].y));
            tmp.radius=tmp.center.distance(p[0]);
        return tmp;
        }
    };
```

多边形

类定义

注意多边形退化为线

```
struct Polygon{ //polygon
        int n;
        vector<Point> p;
        Polygon(): n(0){}
        Polygon(int n,Point* p0): n(n) {for (int i=0;i<n;i++) p.push_back(p0[i]);}
        Polygon(int n,vector<Point> p0): n(n) {p.assign(p0.begin(),p0.end());}
        Polygon(vector<Point> p0): n(p0.size()) {p.assign(p0.begin(),p0.end());}
        friend istream& operator >>(istream &in,Polygon &x){
        in>>x.n;
        x.p.resize(x.n);
        for(int i=0;i<x.n;i++)
          in>>x.p[i];
        return in;
    }
    friend ostream& operator <<(ostream &out,const Polygon &x){
        for(int i=0;i<x.p.size();i++) cout<<x.p[i]<<endl;
        return out;
    }
        bool is_convex(){//if this is convex,edge collinear //判断非严格凸包
            int i,s[3]={1,1,1};
            for (i=0;i<n&&s[1]|s[2];i++)
                s[_sign(p[i].xmult(p[(i+1)%n],p[(i+2)%n]))]=0;
            return s[1]|s[2];
        }
        bool is_convex_ex(){//if this is convex,no edge collinear //判断严格凸包
            int i,s[3]={1,1,1};
            for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
                s[_sign(p[i].xmult(p[(i+1)%n],p[(i+2)%n]))]=0;
            return s[0]&&s[1]|s[2];
        }
        bool inside_convex(Point q){ //if point in convex
            int i,s[3]={1,1,1};
            for (i=0;i<n&&s[1]|s[2];i++)
                s[_sign(p[i].xmult(p[(i+1)%n],q))]=0;
```

```
        return s[1]|s[2];
    }
    bool inside_convex_ex(Point q){//if point in convex,except edge
        int i,s[3]={1,1,1};
        for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
            s[_sign(p[i].xmult(p[(i+1)%n],q))]=0;
        return s[0]&&s[1]|s[2];
    }
    int inside_polygon(Point q,int on_edge=1){//if point in polygon,if on edge return
on_edge
        Point q2;
        int i=0,count;
        while (i<n)
            for (count=i=0,q2.x=rand()+10000,q2.y=rand()+10000;i<n;i++)
                if (_zero(p[(i+1)%n].xmult(q,p[i]))&&(p[i].x-q.x)*(p[(i+1)%n].x-q.x)
<eps&&(p[i].y-q.y)*(p[(i+1)%n].y-q.y)<eps)
                    return on_edge;
                else if (_zero(p[i].xmult(q,q2)))
                    break;
                else if (q2.xmult(q,p[i])*q2.xmult(q,p[(i+1)%n])<-
eps&&p[(i+1)%n].xmult(p[i],q)*p[(i+1)%n].xmult(p[i],q2)<-eps)
                    count++;
        return count&1;
    }
    int inside_polygon(Line l)////if segments in convex include edge
    {
        Point l1=l.a,l2=l.b,tt;
        vector<Point> t;
        int i,j;
        if (!inside_polygon(l1)||!inside_polygon(l2))
            return 0;
        for (i=0;i<n;i++)
            if
(Line(p[i],p[(i+1)%n]).opposite_side(l1,l2)&&l.opposite_side(p[i],p[(i+1)%n]))
                return 0;
            else if (Line(p[i],p[(i+1)%n]).dot_online_in(l1))
                t.push_back(l1);
            else if (Line(p[i],p[(i+1)%n]).dot_online_in(l2))
                t.push_back(l2);
            else if (Line(l1,l2).dot_online_in(p[i]))
                t.push_back(p[i]);
        for (i=0;i<t.size();i++)
            for (j=i+1;j<t.size();j++){
                tt.x=(t[i].x+t[j].x)/2;
                tt.y=(t[i].y+t[j].y)/2;
                if (!inside_polygon(tt))
                    return 0;
            }
        return 1;
```

```
    }
    Point barycenter(){//the barycenter of the polygon  重心
        Point ret,t;
        double t1=0,t2;
        int i;
        ret.x=ret.y=0;
        for (i=1;i<n-1;i++)
            if (fabs(t2=p[i+1].xmult(p[0],p[i]))>eps){
                t=Triangle(p[0],p[i],p[i+1]).barycenter();
                ret.x+=t.x*t2;
                ret.y+=t.y*t2;
                t1+=t2;
            }
        if (fabs(t1)>eps)
            ret.x/=t1,ret.y/=t1;
        return ret;
    }
    double area(){//the area of the polygon
        double s1=0,s2=0;
        for(int 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;
    }
    double perimeter(){//the perimeter of the polygon 周长
    double pm=0;
    if(n>1) pm=p[n-1].distance(p[0]);
    for(int i=0;i<n-1;i++)
      pm+=p[i].distance(p[i+1]);
    return pm;
  }
    Polygon graham(){//Calculate the convex of polygon or points  凸包
        Polygon g=(*this);
        if(g.n<3) return g;
        sort(g.p.begin(),g.p.end(),Point().comp);
        Point bp=g.p[0];
        for(int i=0;i<g.n;i++) g.p[i]=g.p[i]-bp;
        sort(g.p.begin(),g.p.end());
        Polygon gra;
        gra.p.push_back(g.p[0]);
        gra.p.push_back(g.p[1]);
    int i=2;
    for(i=2;i<n;i++)
    {
      while(gra.p.size()>1&&gra.p[gra.p.size()-2].xmult(gra.p[gra.p.size()-1],g.p[i])
<eps) //<eps æ¯∂'Õπ∞¸ <-eps ¥¯180Ω«Õπ∞¸
        gra.p.pop_back();
      gra.p.push_back(g.p[i]);
    }
    gra.n=gra.p.size();
```

```
      for(int i=0;i<gra.n;i++) gra.p[i]=gra.p[i]+bp;
          return gra;
      }
    Circle mincircle(){//calculate the min circle to cover polygon or points  最小覆盖圆
      Polygon g=*this;
        random_shuffle(g.p.begin(),g.p.end());
        Circle ans;
        ans=Circle(g.p[0],0);
      Point &c=ans.center;
      double &r=ans.radius;
        for(int i=1;i<n;i++)
      {
            if(c.distance(g.p[i])>r+eps)
        {
                ans=Circle(g.p[i],0);
                for(int j=0;j<i;j++)
          {
                    if(c.distance(g.p[j])>r+eps)
            {

 ans=Circle(Line(g.p[i],g.p[j]).midpoint(),g.p[i].distance(g.p[j])/2);
                        for(int k=0;k<j;k++) {
                            if(c.distance(g.p[k])>r+eps)
                {
                                ans=Triangle(g.p[i],g.p[j],g.p[k]).circumcircle();
                            }
                        }
                    }
                }
            }
        }
        return ans;
    }
    double rotate_calipers(){ //求凸多边形宽度，凸多边形的宽度定义为平行切线间的最小距离
      Polygon convex=graham();
      if(convex.n<2) return 0;
      double ans=convex.p[0].distance(convex.p[1]);
      int q=1;
      for(int i=0;i<convex.n;i++)
        {
          while(convex.p[(i+1)%convex.n].xmult(convex.p[q],convex.p[i])
<convex.p[(i+1)%convex.n].xmult(convex.p[(q+1)%convex.n],convex.p[i]))
                q=(q+1)%convex.n;

 ans=max(ans,max(convex.p[q].distance(convex.p[i]),convex.p[(q+1)%convex.n].distance(co
nvex.p[(i+1)%convex.n])));
        }
        return ans;
    }
```

```
};
```