这里是计算几何全家桶

笛卡尔坐标系下的点定义

```
//definition
struct V{
  double x,y;
  V():x(0),y(0){}
  V(double a,double b):x(a),y(b){}
};
V ans[10];//declared for other functions
int tot;
inline void input(V &a){a.x=read();a.y=read();}
void print(const V &a,int op=1){printf("%.10lf %.10lf",a.x,a.y);putchar(op?10:32);}
//op:endl or space

```

向量基本运算

加减乘除点积叉积等等

```
//basic operation
inline V operator + (const V &a,const V &b){return (V){a.x+b.x,a.y+b.y};}
inline V operator - (const V &a,const V &b){return (V){a.x-b.x,a.y-b.y};}
inline V operator * (const double &x,const V &a){return (V){a.x*x,a.y*x};}
inline V operator * (const V &a,const double &x){return (V){a.x*x,a.y*x};}
inline V operator / (const V &a,const double x){return (V){a.x/x,a.y/x};}
inline bool operator == (const V &a,const V &b){return abs(a.x-b.x)<eps&&abs(a.y-b.y)<eps;}
inline bool operator != (const V &a,const V &b){return !(a==b);}
inline double operator * (const V &a,const V &b){return a.x*b.x+a.y*b.y;}
inline double operator ^ (const V &a,const V &b){return a.x*b.y-a.y*b.x;}
inline double len(const V &a){return sqrt(a.x*a.x+a.y*a.y);}
inline V mid(const V &a,const V &b){return (V){(a.x+b.x)/2,(a.y+b.y)/2};}
inline V cui(const V &a){return (V){a.y,-a.x};}//not take direction into account
inline V danwei(const V &a){return a/len(a);}
inline double tri_S(const V &a,const V &b,const V &c){return abs((b-a)^(c-a))/2;}//always be non-negative
inline bool operator < (const V &a,const V &b){
  return a.x<b.x-eps||(abs(a.x-b.x)<eps&&a.y<b.y-eps);
}
```

角度相关

```
inline double angle(const V &a,const V &b) { return acos(a * b / len(a) / len(b)); }
inline bool dun(const V &a,const V &b,const V &c){return ((b-a)*(c-a))<-eps;}//angle:BAC
inline bool rui(const V &a,const V &b,const V &c){return ((b-a)*(c-a))>eps;}
inline bool zhi(const V &a,const V &b,const V &c){return abs(((b-a)*(c-a)))<eps;}
```

```
inline V rotate(const V &o,double t){
  double s=sin(t),c=cos(t);
  return (V){o.x*c-o.y*s,o.x*s+o.y*c};
}
```

直线与线段相关

```
//definition
struct line{
  V d,a,b;
};
inline line trans(double a,double b,double c,double d){//given coordinates
  V dd(c-a,d-b),x(a,b),y(c,d);
  dd=dd/len(dd);
  return (line){dd,x,y};
}
inline line trans(const V &a,const V &b){//given points
  V dd(b.x-a.x,b.y-a.y);
  dd=dd/len(dd);
  return (line){dd,a,b};
}
inline void input(line &l){
  double a=read(),b=read(),c=read(),d=read();l=trans(a,b,c,d);return;
}
```

点线结合

```
inline V cui(const line &l,const V &o){//directed
  return l.a+((o-l.a)*l.d)*l.d;
}
```

```
inline V duichen(const line &l,const V &o){
    V u=cui(l,o);
    return (V){2*u.x-o.x,2*u.y-o.y};
}
```

点线关系

```
inline bool on_line(const line &l,const V &o){return abs((l.d^(o-l.a)))
<eps;}
inline bool on_seg(const line &l,const V &o){
  return abs(len(o-l.a)+len(o-l.b)-len(l.a-l.b))<eps;
}
inline int pos(const line &l,const V &o){
  if(!on_line(l,o)){
    if((l.d^(o-l.a))>eps) return 1;//counter clockwise
    else return 2;//clockwise
  }
  else{
    if(((o-l.a)*(o-l.b))<eps) return 5;//on segment
    else if(((o-l.a)*l.d)<-eps) return 3;//online back
    else return 4;//online front
  }
}
```

点线距离

```
inline double dis(const line &l,const V &o,int op=0){//op=0:dis on line,op=1:dis on segment
  if(op&&(dun(l.a,o,l.b)||dun(l.b,o,l.a))) return min(len(o-l.a),len(o-l.b));
  else return abs((o-l.a)^(o-l.b))/len(l.a-l.b);
}
```

线线问题

```
inline bool gongxian(const line &a,const line &b){return abs(a.d^b.d)<eps;}
inline bool cuizhi(const line &a,const line &b){return abs(a.d*b.d)<eps;}
```

是否相交

```
inline bool jiao(const line &u,const line &v){
  if(min(u.a.x,u.b.x)>max(v.a.x,v.b.x)+eps||max(u.a.x,u.b.x)<min(v.a.x,v.b.x)-eps||
  min(u.a.y,u.b.y)>max(v.a.y,v.b.y)+eps||max(u.a.y,u.b.y)<min(v.a.y,v.b.y)-eps) return false;//rapid rejection test
  return ((u.a-v.a)^v.d)*((u.b-v.a)^v.d)<eps&&((v.a-u.a)^u.d)*((v.b-u.a)^u.d)<eps;//straddle test
}
```

线线交点

```
inline V jiaodian(line u,line v){
  double k=((v.a-u.a)^v.d)/(u.d^v.d);
  return u.a+(k*u.d);
}
```

角平分线

```
inline line pingfen(const V &a,const V &b,const V &c){//angle:BAC
  V d1=(b-a)/len(b-a),d2=(c-a)/len(c-a),d=(d1+d2)/len(d1+d2);
  return (line){d,a,a+d};
}
```

多边形

```
inline double S(const V  *a,const int n){
  double res(0);
  for(int i=1;i<=n;i++) res+=(a[i]^a[i%n+1]);
  return res/2;
}
```

多边形的凹凸性

```
bool is_convex(const V *a,const int n){
  a[0]=a[n];a[n+1]=a[1];
  int op=0;
  for(int i=1;i<=n;i++){
    double o=(a[i+1]-a[i])^(a[i]-a[i-1]);
    if(abs(o)<eps) continue;//three neighbouring points on the same line
    int nop=o>0?1:-1;
    if(!op) op=nop;
    else if(op!=nop) return false;
  }
  return true;
}
```

点在多边形内

```
int in_poly(const V *a,const int n,const V o){
  int res(0);
  for(int i=1;i<=n;i++){
    V u=a[i],v=a[i+1];
    if(on_seg(trans(u,v),o)) return 1;//on the edge
    if(abs(u.y-v.y)<eps) continue;//ignore horizontal
    if(max(u.y,v.y)<o.y-eps) continue;//equal will not continue
    if(min(u.y,v.y)>o.y-eps) continue;//equal will continue
    double x=u.x+(o.y-u.y)/(v.y-u.y)*(v.x-u.x);
    if(x<o.x) res^=1;
  }
  return res?2:0;//odd:in even:out
}
```

点在凸包内

```
bool cmp2(V a,V b){
    return (a^b)>0;
}
bool in(const V *c,const V &o){
    if(((c[n]^o)>eps)||((o^c[2])>eps)) return 0;
    int pl=lower_bound(c+1,c+1+n,o,cmp2)-c-1;
    return ((c[pl+1]-c[pl])^(o-c[pl]))>-eps;
}
```

圆

表示法

```
struct cir{
  V o;
  double r;
  cir(double a=0,double b=0,double c=0):o(a,b),r(c){}
};
inline void input(cir &cc){cc.o.x=read();cc.o.y=read();cc.r=read();}
```

位置关系

点圆位置关系

```
inline bool incir(const cir &c,const V &p){return len(p-c.o)<c.r-eps;}
inline bool oncir(const cir &c,const V &p){return (len(p-c.o)-c.r)<eps;}
inline bool outcir(const cir &c,const V &p){return len(p-c.o)>c.r+eps;}
```

线圆位置关系

```
inline double dis(const cir &c,const line &l){return dis(l,c.o);}
inline int pos(const cir &c,const line &l){
    double d=dis(c,l);
    if(d<c.r-eps) return 1;//intersect
    else if(abs(d-c.r)<eps) return 2;//tangent
    else if(d>c.r+eps) return 3;//disjoint
}
```

圆圆位置关系

```
inline double dis(const cir &c1,const cir &c2){return len(c1.o-c2.o);}
inline int pos(const cir &c1,const cir &c2){
  double d=dis(c1,c2);
  if(d>c1.r+c2.r+eps) return 4;//do not cross
  else if(abs(d-c1.r-c2.r)<eps) return 3;//circumscribed
  else if(max(c1.r,c2.r)<min(c1.r,c2.r)+d -eps) return 2;//intersect
  else if(abs(max(c1.r,c2.r)-min(c1.r,c2.r)-d)<eps) return 1;//inscribed
  else return 0;//include
}
```

三角形和圆问题

三角形和内切圆

```
inline cir incir(const V &a,const V &b,const V &c){
  V x(jiaodian(pingfen(a,b,c),pingfen(b,a,c)));
  return cir(x,dis(trans(a,b),x));
}
```

三角形和外接圆

```
inline cir outcir(const V &a,const V &b,const V &c){
  V x(jiaodian(zhongcui(a,b),zhongcui(a,c)));
  return cir(x,len(x-a));
}
inline void input(cir &cc){in(cc.o);cc.r=read();}
```

交点相关

圆与直线交点

```
inline double calc(double xie,double zhi){return sqrt(xie*xie-zhi*zhi);}//Pythagorean Theorem
inline void cross_line_cir(const cir &c,const line &l){
  tot=0;
  V x=chui(l,c.o);
  double dd=len(x-c.o);
  if(c.r<dd-eps) return;//none cross point
  if(abs(c.r-dd)<eps){//one cross point
    ans[++tot]=x;return;
  }
  double dis=calc(c.r,dd);
  V a=x+dis*l.d,b=x-dis*l.d;//two cross points
  ans[++tot]=a;ans[++tot]=b;
  return;
}
```

圆圆交点

```
inline void cross_cir_cir(const cir &c1,const cir c2){
  int op=pos(c1,c2);
  if(op==4||op==0) return;//none cross point
  V L=c2.o-c1.o;
  double a=c2.r,b=c1.r,c=len(L);
  double t=acos((b*b+c*c-a*a)/(2*b*c));//find the angle
  V x=c1.o+c1.r*rotate((L)/len(L),t),y=duichen(trans(c1.o,c2.o),x);
  if(x<y) print(x,0),print(y,1);
  else print(y,0),print(x,1);
}
```

切点/线相关

点到圆的切点

```
inline void qiedian(const cir &c,const V &p){
  tot=0;
  line L=trans(c.o,p);
  double t=acos(c.r/len(c.o-p));
  V a=c.o+(c.r*rotate(L.d,t)),b=duichen(L,a);
  ans[++tot]=a;
  if(a!=b) ans[++tot]=b;
  return;
}
```

圆圆公切线

```
inline void common_qie(const cir &c1,const cir &c2){
  tot=0;
  int op=pos(c1,c2);
  if(op){//outside tangent lines
    double d=c1.r-c2.r,t=acos(d/len(c1.o-c2.o));
```

```
    V u=c1.o+(c1.r*danwei(rotate(c2.o-c1.o,t))),v=c1.o+
(c1.r*danwei(rotate(c2.o-c1.o,-t)));
    ans[++tot]=u;
    if(u!=v) ans[++tot]=v;
  }
  if(op>2){//inside tangent lines
    double d=len(c2.o-c1.o)/(c1.r+c2.r)*c1.r,t=acos(c1.r/d);
    V u=c1.o+(c1.r*danwei(rotate(c2.o-c1.o,t))),v=c1.o+
(c1.r*danwei(rotate(c2.o-c1.o,-t)));
    ans[++tot]=u;
    if(u!=v) ans[++tot]=v;
  }
}
```

圆的面积相关

圆、扇形、弓形

```
inline double cir_S(const cir &c){return pi*c.r*c.r;}
inline double shan(const cir &c,const V &a,const V &b){//S of sector
  double t=ang(a-c.o,b-c.o);
  return t/2*c.r*c.r;
}
inline double gong(const cir &c,const V &a,const V &b){//S of bow
    return shan(c,a,b)-tri_S(c.o,a,b);
}
```

圆与多边形交

```
inline double O_tri(const cir &c,V a,V b){
//directed S of Intersection between circle O and triangle OAB
  if(on(trans(a,b),c.o)) return 0.0;
  int f=(((a-c.o)^(b-c.o))>0)?1:-1;//direction
  line l=trans(a,b);
  int f1=incir(c,a),f2=incir(c,b);
  if(f1&&f2)  return f*tri_S(a,b,c.o);//both incircle:the S of triangle
  else if(!f1&&!f2){//both outcircle:a sector(possibly minus a bow)
    V u=(c.o+(c.r*danwei(a-c.o))),v=(c.o+(c.r*danwei(b-c.o)));
    double S=shan(c,u,v);
    if(dis(l,c.o,1)<c.r-eps){
      cross_line_cir(c,l);
      assert(tot==2);
      S-=gong(c,ans[1],ans[2]);
    }
    return f*S;
  }
  else{//one in and one out:a traingle and a sector
    if(!f1) swap(a,b);
    double sa=Sin(b-a,c.o-a),su=sa/c.r*len(c.o-a),A=asin(sa),U=asin(su);
    if((b-a)*(c.o-a)<-eps) A=pi-A;
    double t=pi-A-U,dis=sin(t)/sa*c.r;
    V u=a+(dis*danwei(b-a)),v=c.r*danwei(b-c.o);
    return f*(tri_S(c.o,a,u)+shan(c,u,v));
  }
}
double common_cir_poly(const V *a,const cir &c,int n){
```

```
  double res(0);
  for(int i=1;i<=n;i++) res+=O_tri(c,a[i],a[i+1]);
  return res;
}
```

圆与圆面积交

```
inline double common_cir_cir(const cir &c1,const cir &c2){
  int op=pos(c1,c2);
  if(op>=3) return 0;//common area=0
  else if(op<=1) return min(cir_S(c1),cir_S(c2));//completly include
  else{//partly include
    double L=len(c1.o-c2.o);
    double t1=2*acos((L*L+c1.r*c1.r-c2.r*c2.r)/(2*L*c1.r));
    double t2=2*acos((L*L+c2.r*c2.r-c1.r*c1.r)/(2*L*c2.r));
    return c1.r*c1.r*t1/2-c1.r*c1.r*sin(t1)/2+c2.r*c2.r*t2/2-
c2.r*c2.r*sin(t2)/2;
  }
}
```

其他

皮克定理

条件：整点任意多边形。
内容：**面积=整点数+1/2*边点数-1**。
（为什么减一？可以考虑极限情况，多边形退化成单格点时，面积是0）

> 拓展：对于平行四边形的格点依然成立；对于三角形的格点，面积要乘二，即2*面积=整点数+1/2*边点数-1，可以理解成变成三角形后面积减少了一半所以要乘二才能相等。

距离

> 规定：以下的 d 表示维度，$x_{i,d}$ 表示点 i 在第 d 维的坐标。

欧几里得距离：$\sqrt{\sum_d (x_{i,d}-x_{j,d})^2}$
曼哈顿距离：$\sum_d |x_{i,d}-x_{j,d}|$
切比雪夫距离：$\max_d |x_{i,d}-x_{j,d}|$

曼哈顿距离和切比雪夫距离的转化

曼哈顿距离下的点 (x,y) 等价于切比雪夫距离下的点 $(x+y,x-y)$。
反过来，切比雪夫下的点 (x,y) 等价于切比雪夫距离下的点 $(\frac{x+y}{2},\frac{x-y}{2})$。
证明：暴力拆分曼哈顿的定义或者结合单位正方形。

极坐标问题

叉积极角排序

```
struct point{
    double x,y;
    point(double x=0, double y=0):x(x), y(y){}
    point operator - (const point &t)const{
        return point(x-t.x, y-t.y);
    }//a - b
    double operator *(const point &t)const{
```

```
        return x*t.x + y*t.y;
    }//a * b
    double operator ^(const point &t)const{
        return x*t.y - y*t.x;
    }//a X b
};
double compare(point a,point b,point c)//计算极角 ab × ac
{
    return (b-a)^(c-a);
}
bool cmp(point a,point b)
{
    double f=compare(p[pos],a,b);
    if(f==0) return a.x-p[pos].x<b.x-p[pos].x;
    else if(f>0) return true;
    else return false;
}
```

```
int Quadrant(point a)//象限排序，注意包含四个坐标轴
{
    if(a.x>0&&a.y>=0)  return 1;
    if(a.x<=0&&a.y>0)  return 2;
    if(a.x<0&&a.y<=0)  return 3;
    if(a.x>=0&&a.y<0)  return 4;
}


bool cmp2(point a,point b)//先象限后极角
{
    if(Quadrant(a)==Quadrant(b))//返回值就是象限
        return cmp(a,b);
    else Quadrant(a)<Quadrant(b);
}
```