Geometry

实数比较

```
const db EPS = 1e-9;

inline int sign(db a) { return a < -EPS ? -1 : a > EPS; }

inline int cmp(db a, db b){ return sign(a-b); }
```

点

```
struct P {
    db x, y;
    P() {}
    P(db _x, db _y) : x(_x), y(_y) {}
    P operator+(P p) { return {x + p.x, y + p.y}; }
    P operator-(P p) { return {x - p.x, y - p.y}; }
    P operator*(db d) { return {x * d, y * d}; }
    P operator/(db d) { return {x / d, y / d}; }

    bool operator<(P p) const {
        int c = cmp(x, p.x);
        if (c) return c == -1;
        return cmp(y, p.y) == -1;
    }

    bool operator==(P o) const{
        return cmp(x,o.x) == 0 && cmp(y,o.y) == 0;
    }
```

其他函数

```
db distTo(P p) { return (*this-p).abs(); }
db alpha() { return atan2(y, x); }
void read() { cin>>x>>y; }
void write() {cout<<"("<<x<<","<<y<<")"<<endl;}
db abs() { return sqrt(abs2());}
db abs2() { return x * x + y * y; }
P rot90() { return P(-y,x);}
P unit() { return *this/abs(); }
int quad() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) >= 0); }
```

点积/叉积

```
    db dot(P p) { return x * p.x + y * p.y; }
    db det(P p) { return x * p.y - y * p.x; }
```

如何极角排序呢?

- 小思考：如何快速判是否有三点共线?

例题

- 有n个点，求锐角三角形的个数/总面积(Beijing Regional 18)。
- n<=2000，可能三点共线。

旋转

复数或矩阵?

```
P rot(db an){ return {x*cos(an)-y*sin(an),x*sin(an) + y*cos(an)}; }
```

小例题

- 有n个点，你要支持将[l,r]之间的点平移/绕某点旋转，查询某个点现在的位置。

- 有一个点，经过一系列平移/旋转操作之后回到了原位，求这个点的位置。

crossop

```
#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))
```

直线平行

```
bool chkLL(P p1, P p2, P q1, P q2) {
    db a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
    return sign(a1+a2) != 0;
}
```

直线交点

```
P isLL(P p1, P p2, P q1, P q2) {
    db a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
    return (p1 * a2 + p2 * a1) / (a1 + a2);
}
```

线段相交

```
bool intersect(db l1,db r1,db l2,db r2){
    if(l1>r1) swap(l1,r1); if(l2>r2) swap(l2,r2);
    return !( cmp(r1,l2) == -1 || cmp(r2,l1) == -1 );
}

bool isSS(P p1, P p2, P q1, P q2){
    return intersect(p1.x,p2.x,q1.x,q2.x) && intersect(p1.y,p2.y,q1.y,q2.y) &&
    crossOp(p1,p2,q1) * crossOp(p1,p2,q2) <= 0 && crossOp(q1,q2,p1)
            * crossOp(q1,q2,p2) <= 0;
}

bool isSS_strict(P p1, P p2, P q1, P q2){
    return crossOp(p1,p2,q1) * crossOp(p1,p2,q2) < 0 && crossOp(q1,q2,p1)
            * crossOp(q1,q2,p2) < 0;
}
```

点在线段上判定

```
bool isMiddle(db a, db m, db b) {
    return sign(a - m) == 0 || sign(b - m) == 0 || (a < m != b < m);
}

bool isMiddle(P a, P m, P b) {
    return isMiddle(a.x, m.x, b.x) && isMiddle(a.y, m.y, b.y);
}

bool onSeg(P p1, P p2, P q){
    return crossOp(p1,p2,q) == 0 && isMiddle(p1, q, p2);
}

bool onSeg_strict(P p1, P p2, P q){
    return crossOp(p1,p2,q) == 0 && sign((q-p1).dot(p1-p2)) * sign((q-p2).dot(p1-p2)) < 0;
}
```

投影/反射/最近点

```
P proj(P p1, P p2, P q) {
    P dir = p2 - p1;
    return p1 + dir * (dir.dot(q - p1) / dir.abs2());
}

P reflect(P p1, P p2, P q){
    return proj(p1,p2,q) * 2 - q;
}

db nearest(P p1,P p2,P q){
    P h = proj(p1,p2,q);
    if(isMiddle(p1,h,p2))
        return q.distTo(h);
    return min(p1.distTo(q),p2.distTo(q));
}
```

线段距离

```
db disSS(P p1, P p2, P q1, P q2){
    if(isSS(p1,p2,q1,q2)) return 0;
    return min(min(nearest(p1,p2,q1),nearest(p1,p2,q2)), min(nearest(q1,q2,p1),nearest(q1,q2,p2)
}
```

夹角

```
db rad(P p1,P p2){
    return atan2l(p1.det(p2),p1.dot(p2));
}
```

多边形面积

```
db area(vector<P> ps){
    db ret = 0; rep(i,0,ps.size()) ret += ps[i].det(ps[(i+1)%ps.size()]);
    return ret/2;
}
```

点包含

```
int contain(vector<P> ps, P p){ //2:inside,1:on_seg,0:outside
    int n = ps.size(), ret = 0;
    rep(i,0,n){
        P u=ps[i],v=ps[(i+1)%n];
        if(onSeg(u,v,p)) return 1;
        if(cmp(u.y,v.y)<=0) swap(u,v);
        if(cmp(p.y,u.y) >0 || cmp(p.y,v.y) <= 0) continue;
        ret ^= crossOp(p,u,v) > 0;
    }
    return ret*2;
}
```

WF2017 A

- 给出一个 n 个点的简单多边形
- 问能放进去的最长线段长度
- $n \leq 200$

凸包

```
vector<P> convexHull(vector<P> ps) {
    int n = ps.size(); if(n <= 1) return ps;
    sort(ps.begin(), ps.end());
    vector<P> qs(n * 2); int k = 0;
    for (int i = 0; i < n; qs[k++] = ps[i++])
        while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0) --k;
    for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
        while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0) --k;
    qs.resize(k - 1);
    return qs;
}
```

```
vector<P> convexHullNonStrict(vector<P> ps) {
    //caution: need to unique the Ps first
    int n = ps.size(); if(n <= 1) return ps;
    sort(ps.begin(), ps.end());
    vector<P> qs(n * 2); int k = 0;
    for (int i = 0; i < n; qs[k++] = ps[i++])
        while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0) --k;
    for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
        while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0) --k;
    qs.resize(k - 1);
    return qs;
}
```

点集直径

```
db convexDiameter(vector<P> ps){
    int n = ps.size(); if(n <= 1) return 0;
    int is = 0, js = 0; rep(k,1,n) is = ps[k]<ps[is]?k:is, js = ps[js] < ps[k]?k:js;
    int i = is, j = js;
    db ret = ps[i].distTo(ps[j]);
    do{
        if((ps[(i+1)%n]-ps[i]).det(ps[(j+1)%n]-ps[j]) >= 0)
            (++j)%=n;
        else
            (++i)%=n;
        ret = max(ret,ps[i].distTo(ps[j]));
    }while(i!=is || j!=js);
    return ret;
}
```

convexcut

```
vector<P> convexCut(const vector<P>&ps, P q1, P q2) {
    vector<P> qs;
    int n = ps.size();
    rep(i,0,n){
        P p1 = ps[i], p2 = ps[(i+1)%n];
        int d1 = crossOp(q1,q2,p1), d2 = crossOp(q1,q2,p2);
        if(d1 >= 0) qs.pb(p1);
        if(d1 * d2 < 0) qs.pb(isLL(p1,p2,q1,q2));
    }
    return qs;
}
```

Xuzhou 2018 M的几何部分

- 给你一个凸多边形，和多边形外一个点，问能照到哪些边。

WF2012 A

- 三维空间中 n 个点，每个点有个速度向量
- 问在运动过程中，最小生成树变化了多少次（只有连续作为最小生成树超过 1e-6 的时间才会被计入）
- $n \le 50$
- 数据保证在任意长度大于等于 1e-6 的时间范围内，都存在一个时刻最小生成树是唯一的

CF 503 D

- n个点，问是否存在一个三角形，面积恰好为S。
- n<=2000

CF 505 F

- n个点，问有多少种方案选出两个不相交的三角形。
- 假设没有三点共线。
- n<=2000

JAG2016 I

- 一个矩形的城市，中间有一条河
- 河是两条从上边界到下边界且互相不交的折线
- 给定起点终点，求第一关键字水里距离，第二关键字陆地距离的最短路
- 折线点数不超过 20

小例题

- 有n个向量，你要选出其中k个/某个子集，使得这里面的向量加起来模长最大。

圆与圆的关系

```
int type(P o1,db r1,P o2,db r2){
    db d = o1.distTo(o2);
    if(cmp(d,r1+r2) == 1) return 4;
    if(cmp(d,r1+r2) == 0) return 3;
    if(cmp(d,abs(r1-r2)) == 1) return 2;
    if(cmp(d,abs(r1-r2)) == 0) return 1;
    return 0;
}
```

圆与线的交

```
vector<P> isCL(P o,db r,P p1,P p2){
    if (cmp(abs((o-p1).det(p2-p1)/p1.distTo(p2)),r)>0) return {};
    db x = (p1-o).dot(p2-p1), y = (p2-p1).abs2(), d = x * x - y * ((p1-o).abs2() - r*r);
    d = max(d,0.0); P m = p1 - (p2-p1)*(x/y), dr = (p2-p1)*(sqrt(d)/y);
    return {m-dr,m+dr}; //along dir: p1->p2
}
```

圆交

```
vector<P> isCC(P o1, db r1, P o2, db r2) { //need to check whether two circles are the same
    db d = o1.distTo(o2);
    if (cmp(d, r1 + r2) == 1) return {};
    if (cmp(d,abs(r1-r2))==-1) return {};
    d = min(d, r1 + r2);
    db y = (r1 * r1 + d * d - r2 * r2) / (2 * d), x = sqrt(r1 * r1 - y * y);
    P dr = (o2 - o1).unit();
    P q1 = o1 + dr * y, q2 = dr.rot90() * x;
    return {q1-q2,q1+q2};//along circle 1
}
```

切线

```
vector<P> tanCP(P o, db r, P p) {
    db x = (p - o).abs2(), d = x - r * r;
    if (sign(d) <= 0) return {}; // on circle => no tangent
    P q1 = o + (p - o) * (r * r / x);
    P q2 = (p - o).rot90() * (r * sqrt(d) / x);
    return {q1-q2,q1+q2}; //counter clock-wise
}
```

外切线

```
vector<L> extanCC(P o1, db r1, P o2, db r2) {
    vector<L> ret;
    if (cmp(r1, r2) == 0) {
        P dr = (o2 - o1).unit().rot90() * r1;
        ret.pb({o1 + dr, o2 + dr}), ret.pb({o1 - dr, o2 - dr});
    } else {
        P p = (o2 * r1 - o1 * r2) / (r1 - r2);
        vector<P> ps = tanCP(o1, r1, p), qs = tanCP(o2, r2, p);
        rep(i,0,min(ps.size(),qs.size())) ret.pb({ps[i], qs[i]}); //c1 counter-clock wise
    }
    return ret;
}
```

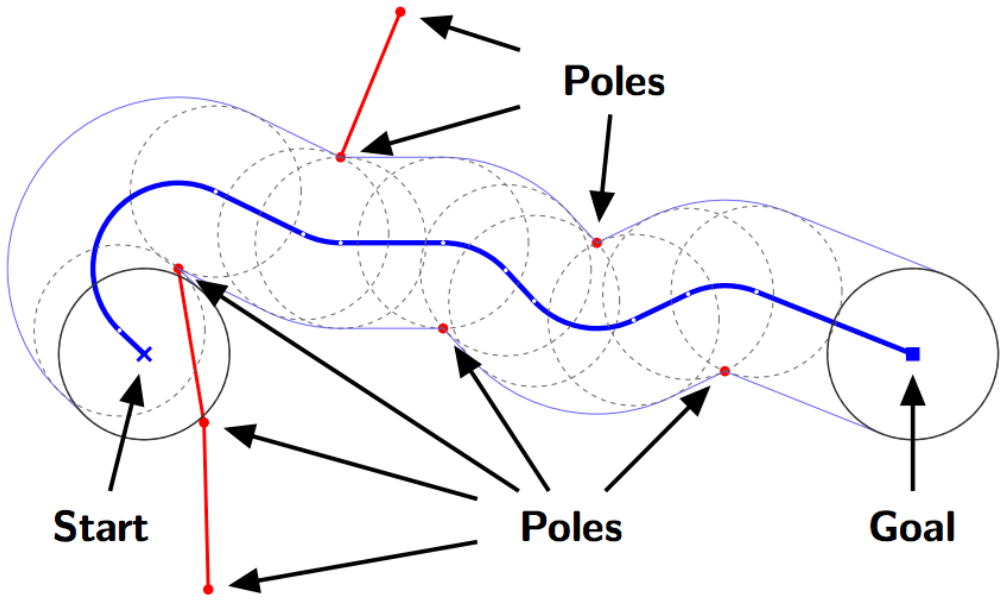
内切线

```
vector<L> intanCC(P o1, db r1, P o2, db r2) {
    vector<L> ret;
    P p = (o1 * r2 + o2 * r1) / (r1 + r2);
    vector<P> ps = tanCP(o1,r1,p), qs = tanCP(o2,r2,p);
    rep(i,0,min(ps.size(),qs.size())) ret.pb({ps[i], qs[i]}); //c1 counter-clock wise
    return ret;
}
```

Cornering at Poles (Tokyo Regional 2014)

给定半径为R的圆的初始位置和目标位置，以及平面上一些障碍点，求最短路距离。

障碍点的个数<=8



Cornering at Poles (Tokyo Regional 2014)

首先将圆缩成点，障碍点扩成圆。

考虑点可能走过的路径

- 圆与圆的公切线
- 点到圆的切线
- 圆弧

建立关键点

- 圆上的切点

建立边

自适应辛普森

```
double F(double x) {
}
double simpson(double a,double b) {
    double c=a+(b-a)/2;
    return (F(a)+F(b)+4*F(c))*(b-a)/6;
}
double rsimpson(double a,double b,double A){
    double c=a+(b-a)/2;
    double L=simpson(a,c),R=simpson(c,b);
    if (fabs(A-L-R)<=eps) return L+R+(A-L-R)/15;
    return rsimpson(a,c,L)+rsimpson(c,b,R);
}
```

圆与三角形的交

```
db areaCT(db r, P p1, P p2){
    vector<P> is = isCL(P(0,0),r,p1,p2);
    if(is.empty()) return r*r*rad(p1,p2)/2;
    bool b1 = cmp(p1.abs2(),r*r) == 1, b2 = cmp(p2.abs2(), r*r) == 1;
    if(b1 && b2){
        if(sign((p1-is[0]).dot(p2-is[0])) <= 0 &&
            sign((p1-is[0]).dot(p2-is[0])) <= 0)
        return r*r*(rad(p1,is[0]) + rad(is[1],p2))/2 + is[0].det(is[1])/2;
        else return r*r*rad(p1,p2)/2;
    }
    if(b1) return (r*r*rad(p1,is[0]) + is[0].det(p2))/2;
    if(b2) return (p1.det(is[1]) + r*r*rad(is[1],p2))/2;
    return p1.det(p2)/2;
}
```

求多边形与圆的交

圆并/交

```
void work(){
    vector<int> cand = {};
    rep(i,0,m){
        bool ok = 1;
        rep(j,0,m) if(i!=j){
            if(rs[j] > rs[i] + EPS && rs[i] + cs[i].distTo(cs[j]) <= rs[j] + EPS){
                ok = 0; break;
            }
            if(cs[i] == cs[j] && cmp(rs[i],rs[j]) == 0 && j < i){
                ok = 0; break;
            }
        }
        if(ok) cand.pb(i);
    }

    rep(i,0,cand.size()) cs[i] = cs[cand[i]], rs[i] = rs[cand[i]];
    m = cand.size();

    db area = 0;
    P wc = 0;
```

```
    //work
    rep(i,0,m){
        vector<pair<db,int> > ev = {{0,0},{2*PI,0}};

        int cur = 0;

        rep(j,0,m) if(j!=i){
            auto ret = isCC(cs[i],rs[i],cs[j],rs[j]);
            if(!ret.empty()){
                db l = (ret[0] - cs[i]).alpha();
                db r = (ret[1] - cs[i]).alpha();
                l = norm(l); r = norm(r);
                ev.pb({l,1});ev.pb({r,-1});
                if(l > r) ++cur;
            }
        }

        sort(ev.begin(), ev.end());
        rep(j,0,ev.size() - 1){
            cur += ev[j].se;
            if(cur == 0){
                area += calc_area_circle(cs[i],rs[i],ev[j].fi,ev[j+1].fi);
                wc = wc + calc_wc_circle(cs[i],rs[i],ev[j].fi,ev[j+1].fi);
            }
        }
    }
```

多校2018 9E

- 给出一个 n 个点的简单多边形和半径 R
- m 次询问，每一次给出一个半径为 R 的圆，问把这个圆给完全移到多边形里的最短距离
- $n,m \le 200$，坐标范围 1e6
- 保证 R 变化 0.1 答案不变

CF 462 C

- 平面上给你n个圆，问将平面分成了多少块。

EC Final 2018 F

- 空间中有个球是障碍，求从s到t的最小距离。

平行/同向/极角序

```
bool parallel(L l0, L l1) { return sign( l0.dir().det( l1.dir() ) ) == 0; }

bool sameDir(L l0, L l1) { return parallel(l0, l1) && sign(l0.dir().dot(l1.dir()) ) == 1; }

bool cmp (P a,  P b) {
    if (a.quad() != b.quad()) {
        return a.quad() < b.quad();
    } else {
        return sign( a.det(b) ) > 0;
    }
}
```

```
bool operator < (L l0, L l1) {
    if (sameDir(l0, l1)) {
        return l1.include(l0[0]);
    } else {
        return cmp( l0.dir(), l1.dir() );
    }
}

bool check(L u, L v, L w) {
    return w.include(isLL(u,v));
}

vector<P> halfPlaneIS(vector<L> &l) {
    sort(l.begin(), l.end());
    deque<L> q;
    for (int i = 0; i < (int)l.size(); ++i) {
        if (i && sameDir(l[i], l[i - 1])) continue;
        while (q.size() > 1 && !check(q[q.size() - 2], q[q.size() - 1], l[i])) q.pop_back();
        while (q.size() > 1 && !check(q[1], q[0], l[i])) q.pop_front();
        q.push_back(l[i]);
    }
    while (q.size() > 2 && !check(q[q.size() - 2], q[q.size() - 1], q[0])) q.pop_back();
    while (q.size() > 2 && !check(q[1], q[0], q[q.size() - 1])) q.pop_front();
    vector<P> ret;
    for (int i = 0; i < (int)q.size(); ++i) ret.push_back(isLL(q[i], q[(i + 1) % q.size()]));
    return ret;
}
```

求两个凸多边形并的周长

最小圆覆盖

```
pair<P,db> min_circle(vector<P> ps){
    random_shuffle(ps.begin(), ps.end());
    int n = ps.size();
    P o = ps[0]; db r = 0;
    rep(i,1,n) if(o.distTo(ps[i]) > r + EPS){
        o = ps[i], r = 0;
        rep(j,0,i) if(o.distTo(ps[j]) > r + EPS){
            o = (ps[i] + ps[j]) / 2; r = o.distTo(ps[i]);
            rep(k,0,j) if(o.distTo(ps[k]) > r + EPS){
                o = circumCenter(ps[i],ps[j],ps[k]);
                r = o.distTo(ps[i]);
            }
        }
    }
    return {o,r};
}
```

Nanjing 18的D

- 给n个点，求最小球覆盖。

多边形重心

WF2018 G

- 给出一个 n 个点的简单多边形
- 找到最小的 R
- 使得多边形内任何一点到最近顶点的距离小于等于 R
- $n \leq 2000$