二维计算几何集合

2D Computational Geometry

注意事项

1. 输出陷阱：小心是否会输出 `-0.00`，此时应该加上 eps 后再输出；
2. 函数越界：使用 `asin(a),acos(a),sqrt(a)` 等函数时，应先校准 a（避免 `asin(1.000001)` 情形）；
3. 所有比较用 `cmp` 和 `sgn` 函数！
4. 如果数据较小，可以调低 `eps`；
5. 重载的叉乘运算符 `^` 优先级较低，注意加括号。

模板

```cpp
#include<algorithm>
#include<cstring>
#include<vector>
#include<cstdio>
#include<cmath>

using namespace std;

const double eps = 1e-8;
const double PI = 4 * atan2(1, 1);
const double INF = 1e16;
const int N = 100005;
inline int sgn(double x){
    if(fabs(x) < eps)    return 0;
    else if(x > 0)  return 1;
    else    return -1;
}
inline int cmp(double x, double y){ return sgn(x-y); }
double rand01(){ return rand() / (double)RAND_MAX; }
double randeps(){ return (rand01() - 0.5) * eps; }

//------------------------------------ Vector & Point ------------------------------------//
struct Vector{
    double x, y;
    Vector() {}
    Vector(double x, double y):x(x), y(y){}
    void read(){ scanf("%lf%lf", &x, &y); }
};
typedef Vector Point;
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 * (double k, Vector A){ return Vector(k * A.x, k * A.y); }
Vector operator * (Vector A, double k){ return k * A; }
Vector operator / (Vector A, double k){ return Vector(A.x / k, A.y / k); }
bool operator < (const Vector &A, const Vector &B){
    return cmp(A.x, B.x) == 0 ? cmp(A.y, B.y) < 0 : cmp(A.x, B.x) < 0;
}
bool operator > (const Vector &A, const Vector &B){ return B < A; }
bool operator == (const Vector &A, const Vector &B){ return (cmp(A.x, B.x) == 0) && (cmp(A.y, B.y) == 0); }
bool operator != (const Vector &A, const Vector &B){ return !(A == B); }
// dot product
double operator * (Vector A, Vector B){ return A.x * B.x + A.y * B.y; }
// cross product
double operator ^ (Vector A, Vector B){ return A.x * B.y - A.y * B.x; }
double Length(Vector A){ return sqrt(A * A); }
// polar angle of vector A, in (-PI,PI]
double Angle(Vector A){ return atan2(A.y, A.x); }
// angle between two vectors, in [0,PI]
double Angle(Vector A, Vector B){ return atan2(fabs(A ^ B), A * B); }
// angle between two vectors, in (-PI, PI]
double signedAngle(Vector A, Vector B){
    double ang = Angle(A, B); if(sgn(A ^ B) < 0) ang *= -1; return ang;
}
// check which half plane is vector A in (up / down)
```

```
bool quad(Vector A){ return sgn(A.y) == 1 || (sgn(A.y) == 0 && sgn(A.x) <= 0); }
// cmpAngle() for sort/lower_bound by polar angle
bool cmpAngle(const Vector &A, const Vector &B){
    if(quad(A) != quad(B))  return quad(A) < quad(B);
    return sgn(A ^ B) > 0 || (sgn(A ^ B) == 0 && Length(A) < Length(B));
}
// the signed area of the parallelogram formed by vector(AB) and vector(AC)
double ParallelogramArea(Point A, Point B, Point C){ return (B - A) ^ (C - A); }
// the signed area of the parallelogram formed by vector v1 and v2
double ParallelogramArea(Vector v1, Vector v2){ return v1 ^ v2; }
// the signed area of the triangle ABC
double TriangleArea(Point A, Point B, Point C){ return ((B - A) ^ (C - A)) / 2; }
// rotate rad counterclockwise
Vector Rotate(Vector A, double rad){
    double co = cos(rad), si = sin(rad);
    return Vector(A.x * co - A.y * si, A.x * si + A.y * co);
}
// get the normal vector of A
Vector Normal(Vector A){ double L = Length(A); return Vector(-A.y/L, A.x/L); }
// get the symmetry vector of A about B
Vector Symmetry(Vector A, Vector B){ return 2 * B * (A * B / (B * B)) - A; }
// test if vector(bc) is to the left of (ab)
bool ToTheLeft(Point A, Point B, Point C){ return sgn((B - A) ^ (C - B)) > 0; }
// test if vector B is to the left of vector A
bool ToTheLeft(Vector A, Vector B){ return sgn(A ^ B) > 0; }
double DistancePointToPoint(Point A, Point B){ return Length(A-B); }
//------------------------------------------------------------------------------//

//------------------------------------ Line ------------------------------------//
struct Line{
    Point p;
    Vector v;
    double ang; // angle of inclination (-PI, PI]
    Line() {}
    Line(Point p, Vector v):p(p), v(v){ ang = atan2(v.y, v.x); }
    Line(double a, double b, double c){ // ax + by + c = 0
        if(sgn(a) == 0)         p = Point(0, -c/b), v = Vector(1, 0);
        else if(sgn(b) == 0)    p = Point(-c/a, 0), v = Vector(0, 1);
        else                    p = Point(0, -c/b), v = Vector(-b, a);
    }
    Point getPoint(double t){ return p + v * t; }
    bool operator < (const Line &L) const{ return ang < L.ang; }
};
bool PointOnLine(Point p, Line l){ return sgn(l.v ^ (p-l.p)) == 0; }
bool PointOnRight(Point p, Line l){ return sgn(l.v ^ (p-l.p)) < 0; }
bool LineParallel(Line l1, Line l2){ return sgn(l1.v ^ l2.v) == 0; }
bool LineSame(Line l1, Line l2){ return LineParallel(l1, l2) && sgn((l1.p-l2.p) ^ l1.v) == 0; }
Point GetLineIntersection(Line l1, Line l2){
    Vector u = l1.p - l2.p;
    double t = (l2.v ^ u) / (l1.v ^ l2.v);
    return l1.p + l1.v * t;
}
double DistancePointToLine(Point p, Line l){ return fabs(((p - l.p) ^ l.v) / Length(l.v)); }
double DistancePointToLine(Point p, Point A, Point B){ return fabs(((B - A) ^ (p - A)) / Length(B - A)); }
double DistancePointToSegment(Point p, Point A, Point B){
    if(A == B)  return DistancePointToPoint(p, A);
    Vector v1 = p - A, v2 = p - B, v3 = A - B; // v1:vector(Ap), v2:vector(Bp), v3:vector(BA)
    if(sgn(v1 * v3) > 0)    return DistancePointToPoint(p, A);
    if(sgn(v2 * v3) < 0)    return DistancePointToPoint(p, B);
    return DistancePointToLine(p, A, B);
}
Point PointLineProjection(Point p, Line l){ return l.p + l.v * ((l.v * (p - l.p)) / (l.v * l.v)); }
bool PointOnSegment(Point p, Point A, Point B){
    return sgn((p - A) * (p - B)) <= 0 && sgn((p - A) ^ (p - B)) == 0;
}
bool PointOnSegmentEndExcluded(Point p, Point A, Point B){
    return sgn((p - A) * (p - B)) < 0 && sgn((p - A) ^ (p - B)) == 0;
}
bool SegmentIntersectedEndExcluded(Point A1, Point A2, Point B1, Point B2){
    return (sgn((A1 - B1) ^ (B1 - B2)) * sgn((A2 - B1) ^ (B1 - B2)) < 0)
        && (sgn((B1 - A1) ^ (A1 - A2)) * sgn((B2 - A1) ^ (A1 - A2)) < 0);
}
bool SegmentIntersected(Point A1, Point A2, Point B1, Point B2){
    if(SegmentIntersectedEndExcluded(A1, A2, B1, B2))   return true;
    return PointOnSegment(A1, B1, B2) || PointOnSegment(A2, B1, B2) ||
           PointOnSegment(B1, A1, A2) || PointOnSegment(B2, A1, A2) ? true : false;
}
bool LineSegmentIntersected(Line L, Point A, Point B){
```

```cpp
    Point p_1 = L.p, p_2 = L.getPoint(1);
    return sgn(((p_2 - p_1) ^ (A - p_1))) * sgn(((p_2 - p_1) ^ (B - p_1))) <= 0;
}
bool LineSegmentIntersectedEndExcluded(Line L, Point A, Point B){
    Point p_1 = L.p, p_2 = L.getPoint(1);
    return sgn(((p_2 - p_1) ^ (A - p_1))) * sgn(((p_2 - p_1) ^ (B - p_1))) < 0;
}
//-------------------------------------------------------------------------------//

//------------------------------------ Polygon ------------------------------------//
typedef vector<Point> Polygon;
double PolygonArea(int n, Point p[]){
    double S = 0;
    for(int i = 2; i < n; i++)
        S += ((p[i] - p[1]) ^ (p[i+1] - p[1])) / 2;
    return S;
}
double PolygonArea(Polygon poly){
    double S = 0;
    for(int i = 1; i < poly.size() - 1; i++)
        S += ((poly[i] - poly[0]) ^ (poly[i+1] - poly[0])) / 2;
    return S;
}
int PointInPolygon(Point A, int n, Point p[]){ // 0: outside; 1: inside; -1: on edge
    int wn = 0; // winding number
    for(int i = 1; i <= n; i++){
        int nxt = i + 1 > n ? 1 : i + 1;
        if(PointOnSegment(A, p[i], p[nxt])) return -1;
        int k = sgn((p[nxt] - p[i]) ^ (A - p[i]));
        int d1 = sgn(p[i].y - A.y);
        int d2 = sgn(p[nxt].y - A.y);
        if(k > 0 && d1 <= 0 && d2 > 0)  wn++;
        if(k < 0 && d2 <= 0 && d1 > 0)  wn--;
    }
    if(wn != 0) return 1;
    return 0;
}
int PointInPolygon(Point A, Polygon poly){ // 0: outside; 1: inside; -1: on edge
    int wn = 0; // winding number
    int n = poly.size();
    for(int i = 0; i < n; i++){
        int nxt = (i + 1) % n;
        if(PointOnSegment(A, poly[i], poly[nxt]))   return -1;
        int k = sgn((poly[nxt] - poly[i]) ^ (A - poly[i]));
        int d1 = sgn(poly[i].y - A.y);
        int d2 = sgn(poly[nxt].y - A.y);
        if(k > 0 && d1 <= 0 && d2 > 0)  wn++;
        if(k < 0 && d2 <= 0 && d1 > 0)  wn--;
    }
    if(wn != 0) return 1;
    return 0;
}
Point getPolygonCenter(int n, Point p[]){
    Point res(0, 0);
    double S = 0;
    for(int i = 2; i < n; i++){
        double area = ((p[i] - p[1]) ^ (p[i+1] - p[1]));
        res = res + (p[1] + p[i] + p[i+1]) * area;
        S += area;
    }
    return res / S / 3;
}
// the left part of l and poly form a new polygon
Polygon cutPolygon(Line l, Polygon poly){
    Polygon newpoly;
    int n = poly.size();
    for(int i = 0; i < n; i++){
        Point C = poly[i], D = poly[(i+1)%n];
        if(sgn(l.v ^ (C - l.p)) >= 0)   newpoly.push_back(C);
        if(sgn(l.v ^ (C - D)) != 0){
            Point q = GetLineIntersection(l, Line(C, C - D));
            if(PointOnSegmentEndExcluded(q, C, D))  newpoly.push_back(q);
        }
    }
    return newpoly;
}
//-------------------------------------------------------------------------------//

```

```
//------------------------------------ Circle -----------------------------------//
struct Circle{
    Point p;
    double r;
    Circle() {}
    Circle(Point p, double r):p(p), r(r) {}
    Point getPoint(double alpha){
        return Point(p.x + cos(alpha) * r, p.y + sin(alpha) * r);
    }
};
void getLineCircleIntersection(Line L, Circle C, Point res[], int &resn){
    // resn is the number of intersecton points
    // intersection points are stored in res[]
    resn = 0;
    Point q = PointLineProjection(C.p, L);
    double d = DistancePointToPoint(C.p, q);
    if(cmp(d, C.r) > 0)   return;                          // separated
    else if(cmp(d, C.r) == 0){ res[++resn] = q; return; }   // tangent
    Vector u = L.v / Length(L.v);
    double dd = sqrt(C.r * C.r - d * d);
    res[++resn] = q - dd * u, res[++resn] = q + dd * u;     // intersected
}
void getCircleCircleIntersection(Circle C1, Circle C2, Point res[], int &resn){
    // resn is the number of intersection points (-1 if two circles coincide)
    // intersection points are stored in res[]
    resn = 0;
    double d = DistancePointToPoint(C1.p, C2.p);
    if(sgn(d) == 0) {
        if (cmp(C1.r, C2.r) == 0)   resn = -1;          // two circles are the same
        return;                                         // or concentric
    }
    if(cmp(C1.r + C2.r, d) < 0) return;                 // separated
    if(cmp(fabs(C1.r - C2.r), d) > 0)   return;         // contained
    double a = Angle(C2.p - C1.p);
    double da = acos((d * d + C1.r * C1.r - C2.r * C2.r) / (2 * d * C1.r));
    Point p1 = C1.getPoint(a - da), p2 = C1.getPoint(a + da);
    res[++resn] = p1;
    if(p1 != p2)    res[++resn] = p2;                   // tangent or intersected
}
void getTangents(Point p, Circle C, Line L[], int &lid){
    // lid is the number of tangent lines
    // tangent lines are stored in L[]
    lid = 0;
    Vector u = C.p - p;
    double d = Length(u);
    if(cmp(d, C.r) < 0) return;
    else if(cmp(d, C.r) == 0)   L[++lid] = Line(p, Rotate(u, PI / 2));
    else if(cmp(d, C.r) > 0){
        double ang = asin(C.r / d);
        L[++lid] = Line(p, Rotate(u, -ang));
        L[++lid] = Line(p, Rotate(u, ang));
    }
}
void getTangents(Point p, Circle C, Point P[], int &pid){
    // pid is the number of tangent points
    // tangent points are stored in P[]
    pid = 0;
    Vector u = p - C.p;
    double d = Length(u);
    if(cmp(d, C.r) < 0) return;
    else if(cmp(d, C.r) == 0)   P[++pid] = p;
    else if(cmp(d, C.r) > 0){
        double ang = acos(C.r / d);
        P[++pid] = C.p + Rotate(u, -ang) / d * C.r;
        P[++pid] = C.p + Rotate(u, ang) / d * C.r;
    }
}
void getTangents(Circle C1, Circle C2, Point c1[], Point c2[], int &resn){
    // resn is the number of tangent lines
    // c1[] and c2[] are relevant points on C1 and C2
    resn = 0;
    if(cmp(C1.r, C2.r) < 0)   swap(C1, C2), swap(c1, c2);
    double d = DistancePointToPoint(C1.p, C2.p);
    if(sgn(d) == 0 && cmp(C1.r, C2.r) == 0){ resn = -1; return; }   // two circles are the same
    if(cmp(C1.r - C2.r, d) > 0) return;                             // contained
    double base = Angle(C2.p - C1.p);
    if(cmp(C1.r - C2.r, d) == 0){                                   // internally tangent
        c1[++resn] = C1.getPoint(base), c2[resn] = C2.getPoint(base);
```

```
        return;
    }
    double ang = acos((C1.r - C2.r) / d);
    c1[++resn] = C1.getPoint(base - ang), c2[resn] = C2.getPoint(base - ang);
    c1[++resn] = C1.getPoint(base + ang), c2[resn] = C2.getPoint(base + ang);
    if(cmp(C1.r + C2.r, d) == 0)                                    // externally tangent
        c1[++resn] = C1.getPoint(base), c2[resn] = C2.getPoint(base + PI);
    else if(cmp(C1.r + C2.r, d) < 0){                               // separated
        ang = acos((C1.r + C2.r) / d);
        c1[++resn] = C1.getPoint(base - ang), c2[resn] = C2.getPoint(base - ang + PI);
        c1[++resn] = C1.getPoint(base + ang), c2[resn] = C2.getPoint(base + ang + PI);
    }
}
// get inversion from a circle to a circle
// ensure that point O is not on circle A beforehand
Circle getInversionC2C(Point O, double R, Circle A){
    double OA = Length(A.p - O);
    double rB = R * R / 2 * (1 / (OA - A.r) - 1 / (OA + A.r));
    double xB = O.x + rB / A.r * (A.p.x - O.x);
    double yB = O.y + rB / A.r * (A.p.y - O.y);
    return Circle(Point(xB, yB), rB);
}
// get inversion from a line to a circle
// ensure that point O is not on line L beforehand
// point O is on the result circle
Circle getInversionL2C(Point O, double R, Line L){
    Point P = PointLineProjection(O, L);
    double d = Length(P - O);
    double r = R * R / d / 2;
    Vector v = (P - O) / Length(P - O) * r;
    return Circle(O + v, r);
}
// get inversion from a circle to a line
// ensure that point O is on circle A
Line getInversionC2L(Point O, double R, Circle A){
    Point P = (A.p - O) / Length(A.p - O) * R * R / A.r / 2;
    Vector v = Normal(A.p - O);
    return Line(P, v);
}
//-------------------------------------------------------------------------------------//
//#####################################################################################//

//------------------------------------- Convex Hull -----------------------------------//
void ConvexHull(int n, Point p[], Point sta[], int &staid){
    // there're n points stored in p[], the points on convex hull will be saved in sta[]
    sort(p+1, p+n+1);
    n = unique(p+1, p+n+1) - (p+1);
    staid = 0;
    for(int i = 1; i <= n; i++){
        // points on edge
        // while(staid > 1 && sgn((sta[staid]-sta[staid-1]) ^ (p[i]-sta[staid-1])) < 0) staid--;
        // no points on edge
        while(staid > 1 && sgn((sta[staid]-sta[staid-1]) ^ (p[i]-sta[staid-1])) <= 0) staid--;
        sta[++staid] = p[i];
    }
    int k = staid;
    for(int i = n-1; i >= 1; i--){
        // points on edge
        // while(staid > k && sgn((sta[staid]-sta[staid-1]) ^ (p[i]-sta[staid-1])) < 0) staid--;
        // no points on edge
        while(staid > k && sgn((sta[staid]-sta[staid-1]) ^ (p[i]-sta[staid-1])) <= 0) staid--;
        sta[++staid] = p[i];
    }
    if(n > 1)   staid--;
}

// check if point A is in ConvexHull p[]
// note that p[1] must be the original point
bool PointInConvexHull(Point A, int n, Point p[]){
    if(sgn(A ^ p[2]) > 0 || sgn(A ^ p[n]) < 0)  return false;
    int pos = lower_bound(p + 1, p + n + 1, A, cmpAngle) - p - 1;
    return sgn((A - p[pos]) ^ (p[pos%n+1] - p[pos])) <= 0;
}

void Minkowski(int n1, Point p1[], int n2, Point p2[], Point tmp[], Point res[], int &resn){
    // tmp[] is a auxiliary array
    // p1[] is a convex hull consist of n1 points
    // p2[] is a convex hull consist of n2 points
```

```
    // res[] is the Minkowski tmp of these two convex hull consist of resn points
    p1[n1+1] = p1[1], p2[n2+1] = p2[1];
    vector<Vector> v1, v2;
    for(int i = 1; i <= n1; i++)    v1.emplace_back(p1[i+1] - p1[i]);
    for(int i = 1; i <= n2; i++)    v2.emplace_back(p2[i+1] - p2[i]);
    int pt1 = 0, pt2 = 0, tid = 1;
    tmp[1] = p1[1] + p2[1];
    while(pt1 < n1 && pt2 < n2){
        tid++;
        if(sgn(v1[pt1] ^ v2[pt2]) >= 0) tmp[tid] = tmp[tid-1] + v1[pt1++];
        else                            tmp[tid] = tmp[tid-1] + v2[pt2++];
    }
    while(pt1 < n1) tid++, tmp[tid] = tmp[tid-1] + v1[pt1++];
    while(pt2 < n2) tid++, tmp[tid] = tmp[tid-1] + v2[pt2++];
    ConvexHull(tid, tmp, res, resn);
}

//----------------------------------------------------------------------------------------------//

//------------------------------------ Rotating Calipers -----------------------------------//
void RotatingCalipers(int m, Point p[]){
    // p[] = sta[], m = staid in ConvexHull()
    if(m == 2){
        // do something
        return;
    }
    p[m+1] = p[1];
    int ver = 2;
    for(int i = 1; i <= m; i++){ // enumerate edge: p[i] ~ p[i+1]
        while(TriangleArea(p[i], p[i+1], p[ver]) < TriangleArea(p[i], p[i+1], p[ver+1])){
            ver++;
            if(ver == m+1)    ver = 1; // find the corresponding point: ver
            // do something
        }
    }
}
// calculate the diameter of a convex hull
double DiameterConvexHull(int m, Point p[]){
    // p[] = sta[], m = staid in ConvexHull()
    double ans = 0;
    if(m == 2){
        ans = (p[1] - p[2]) * (p[1] - p[2]);
        return sqrt(ans);
    }
    p[m+1] = p[1];
    int ver = 2;
    for(int i = 1; i <= m; i++){
        while(TriangleArea(p[i], p[i+1], p[ver]) < TriangleArea(p[i], p[i+1], p[ver+1])){
            ver++;
            if(ver == m+1)    ver = 1;
            ans = max(ans, max((p[ver] - p[i]) * (p[ver] - p[i]), (p[ver] - p[i+1]) * (p[ver] - p[i+1])));
        }
    }
    return sqrt(ans);
}

// solve min-area-rectangle cover OR min-perimeter-rectangle cover problem
struct MinRectangleCover{

    double minArea, minPeri;
    Point minAreaPoints[10], minPeriPoints[10];

    void cal(int i, int nxti, int ver, int j, int k, Point p[]){
        Point t[4];
        Vector v = p[nxti] - p[i], u = Normal(v);
        t[0] = GetLineIntersection(Line(p[i], v), Line(p[j], u));
        t[1] = GetLineIntersection(Line(p[j], u), Line(p[ver], v));
        t[2] = GetLineIntersection(Line(p[ver], v), Line(p[k], u));
        t[3] = GetLineIntersection(Line(p[k], u), Line(p[i], v));
        double area = fabs((t[1] - t[0]) ^ (t[0] - t[3]));
        if(cmp(area, minArea) < 0){
            minArea = area;
            minAreaPoints[0] = t[0], minAreaPoints[1] = t[1];
            minAreaPoints[2] = t[2], minAreaPoints[3] = t[3];
        }
        double peri = Length(t[1]-t[0]) + Length(t[0]-t[3]); peri *= 2;
        if(cmp(peri, minPeri) < 0){
            minPeri = peri;
```

```cpp
            minPeriPoints[0] = t[0], minPeriPoints[1] = t[1];
            minPeriPoints[2] = t[2], minPeriPoints[3] = t[3];
        }
    }
    inline void Norm(int &x, int m){ ((x %= m) += m) %= m; if(x == 0)  x = m; }
    inline double func(int mid, int i, int nxti, Point p[], int m, int kind){
        Norm(mid, m);
        if(kind == 1)
            return (p[nxti]-p[i]) * (p[mid]-p[i]) / Length(p[nxti]-p[i]);
        else
            return (p[i]-p[nxti]) * (p[mid]-p[nxti]) / Length(p[i]-p[nxti]);
    }
    int tripartition(int l, int r, int i, int nxti, Point p[], int m, int kind){
        while(r < l)   r += m;
        int mid1 = l, mid2 = r;
        while(mid1 < mid2){
            mid1 = l + (r - l) / 3;
            mid2 = r - (r - l) / 3;
            // func(x) is a unimodal function
            if(func(mid1, i, nxti, p, m, kind) < func(mid2, i, nxti, p, m, kind))
                l = mid1 + 1;
            else    r = mid2 - 1;
        }
        return l;
    }
    // minimum rectangle covering the points p[]
    void solve(int m, Point p[]){
        minArea = minPeri = INF;
        int ver = 2;
        for(int i = 1; i <= m; i++){
            int nxti = i + 1; Norm(nxti, m);
            while(TriangleArea(p[i], p[nxti], p[ver]) < TriangleArea(p[i], p[nxti], p[ver+1]))
                ver++, Norm(ver, m);
            int l = nxti, r = ver;
            int j = tripartition(l, r, i, nxti, p, m, 1);
            l = ver, r = i;
            int k = tripartition(l, r, i, nxti, p, m, 2);
            Norm(k, m), Norm(j, m);
            cal(i, nxti, ver, j, k, p);
        }
    }

};

//------------------------------------------------------------------------------------------------//

//------------------------------------ HalfplaneIntersection ------------------------------------//
struct Halfplane{
    Point P[N]; // P[i] is the intersection of line Q[i] and Q[i+1]
    Line Q[N];  // deque
    void HalfplaneIntersection(Line L[], int n, Point res[], int &m){
        // L[] are the lines, n is the number of lines, res[] stores the result of the intersection (a polygon)
        // m is the number of points of the intersection (which is a polygon)
        sort(L + 1, L + n + 1);
        int head, tail;
        Q[head = tail = 0] = L[1];
        for(int i = 2; i <= n; i++){
            while(head < tail && PointOnRight(P[tail - 1], L[i]))   tail--;
            while(head < tail && PointOnRight(P[head], L[i]))   head++;
            Q[++tail] = L[i];
            if(sgn(Q[tail].v ^ Q[tail - 1].v) == 0){ // parallel, the inner one remains
                tail--;
                if(!PointOnRight(L[i].p, Q[tail]))    Q[tail] = L[i];
            }
            if(head < tail) P[tail - 1] = GetLineIntersection(Q[tail-1], Q[tail]);
        }
        while(head < tail && PointOnRight(P[tail - 1], Q[head])) tail--; // delete useless plane
        P[tail] = GetLineIntersection(Q[tail], Q[head]);

        m = 0;
        for(int i = head; i <= tail; i++)   res[++m] = P[i];
    }
};
//------------------------------------------------------------------------------------------------//

int main(){
    ;
}
```