

```

1 #include <bits/stdc++.h>
2 #define maxn 800005
3 #define int long long
4 using namespace std;
5
6
7 // +-----+
8 // |          |
9 // | Geometry Template BASIC(Sgn) |
10 // |          |
11 // +-----+
12
13
14 //const long long eps = 0;
15 const long double eps = 1e-8;
16
17 //long double情况下使用的sgn函数
18 int sgn(long double x){
19     if(fabs(x) <= eps)    return 0;
20     return x > 0 ? 1 : -1;
21 }
22 /*
23 //long long情况下使用的sgn函数
24 int sgnLL(long long x){
25     if(abs(x) == eps)    return 0;
26     return x > 0 ? 1 : -1;
27 }
28 */
29
30 // +-----+
31 // |          |
32 // | Geometry Template Struct |
33 // |          |
34 // +-----+
35
36 /* *** 点（基础模板） *** */
37 template<typename T> struct TP{
38     T x, y;
39     TP(){}
40     TP(T _x, T _y){ x = _x; y = _y; }
41     TP operator -() const {
42         return {-x, -y};
43     }
44     friend TP operator +(const TP &a, const TP &b){
45         return {a.x + b.x, a.y + b.y};
46     }
47     friend TP operator -(const TP &a, const TP &b){
48         return {a.x - b.x, a.y - b.y};
49     }
50     friend T operator *(const TP &a, const TP &b){
51         return a.x * b.x + a.y * b.y;
52     }
53     friend T operator ^(const TP &a, const TP &b){
54         return a.x * b.y - a.y * b.x;
55     }
56     friend bool operator ==(const TP &a, const TP &b){
57         return sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0;
58     }
59     friend bool operator <(const TP &a, const TP &b){
60         if(sgn(a.x - b.x) == 0)    return sgn(a.y - b.y) < 0;
61         return sgn(a.x - b.x) < 0;

```

```

62     }
63     TP operator *(const long double &k) const{
64         return {x * k, y * k};
65     }
66     TP operator /(const long double &k) const{
67         return {x / k, y / k};
68     }
69     //a在逆时针方向: 1, 顺时针方向: -1, 其他: 0
70     int toleft(const TP &a) const {
71         auto t = (*this) ^ a;
72         return (t > eps) - (t < -eps);
73     }
74     //返回极角, (-PI, PI]
75     long double angle(){
76         return atan2(y, x);
77     }
78     //返回长度
79     long double len() const {
80         return sqrt(len2());
81     }
82     //返回两点间距离
83     long double dis(const TP &a) const {
84         return sqrt(dis2(a));
85     }
86     //返回长度的平方
87     T len2() const {
88         return (*this) * (*this);
89     }
90     //返回两点间距离的平方
91     T dis2(const TP &a) const {
92         return TP(x - a.x, y - a.y).len2();
93     }
94     //返回两向量的夹角
95     long double ang(const TP &a) const {
96         return acos(max(-1.0, min(1.0, ((*this) * a) / (len() * a.Len()))));
97     }
98     //返回逆时针旋转rad后的结果
99     TP rot(const long double rad) const {
100         return {x * cos(rad) - y * sin(rad), x * sin(rad) + y * cos(rad)};
101     }
102     //旋转 (指定cos和sin)
103     TP rot(const long double cosr, const long double sinr) const {
104         return {x * cosr - y * sinr, x * sinr + y * cosr};
105     }
106     //返回长度为R的向量
107     TP trunc(long double r){
108         long double l = len();
109         if(!sgn(l)) return (*this);
110         r /= l;
111         return {x * r, y * r};
112     }
113     void input(){
114         cin >> x >> y;
115     }
116     void print(){
117         cout << "[Point]\n";
118         cout << x << " " << y << '\n';
119     }
120 }; using Point = TP<long double>;
121
122 /* *** 线 (基础模板) *** */
123 template<typename T> struct TL{

```

```

124     TP<T> s, e;
125     TL(){}
126     TL(TP<T> _s, TP<T> _e){ s = _s; e = _e; }
127     friend T operator *(const TL &la, const TL &lb){
128         return (la.e - la.s) * (lb.e - lb.s);
129     }
130     friend T operator ^(const TL &la, const TL &lb){
131         return (la.e - la.s) ^ (lb.e - lb.s);
132     }
133     friend bool operator ==(const TL &la, const TL &lb){
134         return la.parallel(lb) && la.isOnSeg(lb.s);
135     }
136     //点到直线的距离
137     long double length(){
138         return (e - s).Len();
139     }
140     //点到直线的距离
141     long double disLine(const TP<T> &p) const {
142         return fabs((p - s) ^ (e - s)) / length();
143     }
144     //点到线段的距离
145     long double disSeg(const TP<T> &p) const{
146         if(sgn((p - s) * (e - s)) < 0 || sgn((p - e) * (s - e)) < 0){
147             return min(p.dis(s), p.dis(e));
148         }
149         return disLine(p);
150     }
151     //点到直线的投影
152     TP<T> proj(const TP<T> &p) const {
153         return s + (((e - s) * ((e - s) * (p - s))) / ((e - s).Len2()));
154     }
155     //关于直线的对称点
156     TP<T> symmetryPoint(TP<T> p){
157         Point q = proj(p);
158         return Point(2 * q.x - p.x, 2 * q.y - p.y);
159     }
160     //两直线平行
161     bool parallel(TL v){
162         return sgn((e - s) ^ (v.e - v.s)) == 0;
163     }
164     //两直线交点
165     TP<T> crosspoint(TL v){
166         auto a1 = (v.e - v.s) ^ (s - v.s);
167         auto a2 = (v.e - v.s) ^ (e - v.s);
168         return {(s.x * a2 - e.x * a1) / (a2 - a1), (s.y * a2 - e.y * a1) / (a2 - a1)};
169     }
170     //点在线段上, 端点: -1, 线段内: 1, 其他: 0
171     int isOnSeg(const TP<T> &p){
172         if(p == s || p == e) return -1;
173         return sgn((p - s) ^ (e - s)) == 0 && sgn((p - s) * (p - e)) <= 0;
174     }
175     //2 -> 规范相交, 1 -> 非规范相交, 0 -> 不相交
176     int segCrossSeg(TL v){
177         int d1 = sgn((e - s) ^ (v.s - s));
178         int d2 = sgn((e - s) ^ (v.e - s));
179         int d3 = sgn((v.e - v.s) ^ (s - v.s));
180         int d4 = sgn((v.e - v.s) ^ (e - v.s));
181         if((d1 ^ d2) == -2 && (d3 ^ d4) == -2) return 2;
182         return (d1 == 0 && sgn((v.s - s) * (v.s - e)) <= 0) ||
183             (d2 == 0 && sgn((v.e - s) * (v.e - e)) <= 0) ||
184             (d3 == 0 && sgn((s - v.s) * (s - v.e)) <= 0) ||
185             (d4 == 0 && sgn((e - v.s) * (e - v.e)) <= 0);

```

```

186 }
187 // this -> Line, v -> Seg, 2, 规范相交, 1, 非规范相交, 0, 不相交
188 int lineCrossSeg(TL v){
189     int d1 = sgn((e - s) ^ (v.s - s));
190     int d2 = sgn((e - s) ^ (v.e - s));
191     if((d1 ^ d2) == -2) return 2;
192     return (d1 == 0 || d2 == 0);
193 }
194 //两直线关系, 0, 平行, 1, 重合, 2, 相交
195 int lineRelation(TL v){
196     if((*this).parallel(v)){
197         return v.toleft(s) == 0;
198     }
199     return 2;
200 }
201 //a在直线的, 逆时针方向: 1, 顺时针方向: -1, 其他: 0
202 int toleft(const TP<T> &p) const {
203     int c = sgn((p - s) ^ (e - s));
204     if(c < 0) return 1;
205     else if(c > 0) return -1;
206     else return 0;
207 }
208 void print(){
209     cout << "[Line]\n";
210     cout << s.x << " " << s.y << " " << e.x << " " << e.y << '\n';
211 }
212 }; using Line = TL<long double>;
213
214 /* *** 圆 (基础模板) *** */
215 const long double PI = acos(-1.0);
216 template<typename T> struct TC{
217     TP<T> c;
218     long double r;
219     TC(){ }
220     TC(TP<T> _c, long double _r){ c = _c; r = _r; }
221     //根据极角返回圆上一点
222     TP<T> point(long double a) {
223         return TP<T>(c.x + cos(a) * r, c.y + sin(a) * r);
224     }
225     long double area(){
226         return PI * r * r;
227     }
228     //5 -> 相离, 4 -> 外切, 3 -> 相交, 2 -> 内切, 1 -> 内含
229     int relationCircle(TC v){
230         long double d = c.dis(v.c);
231         if(sgn((d - r - v.r)) > 0) return 5;
232         if(sgn((d - r - v.r)) == 0) return 4;
233         long double l = fabs(r - v.r);
234         if(sgn((d - r - v.r)) < 0 && sgn(d - l) > 0) return 3;
235         if(sgn(d - l) == 0) return 2;
236         if(sgn(d - l) < 0) return 1;
237     }
238     //【已测试: ZOJ1597】
239     //两圆相交得到的面积
240     long double areaCircle(TC v){
241         int rel = relationCircle(v);
242         if(rel >= 4) return 0.0;
243         if(rel <= 2) return min(area(), v.area());
244         long double d = c.dis(v.c);
245         long double hf = (r + v.r + d) / 2.0;
246         long double ss = 2 * sqrt(hf * (hf - r) * (hf - v.r) * (hf - d));
247         long double a1 = acos((r * r + d * d - v.r * v.r) / (2.0 * r * d));

```

```

248     a1 = a1 * r * r;
249     long double a2 = acos((v.r * v.r + d * d - r * r) / (2.0 * v.r * d));
250     a2 = a2 * v.r * v.r;
251     return a1 + a2 - ss;
252 }
253 }; using Circle = TC<long double>;
254
255 /* *** 多边形 (基础模板) *** */
256 template<typename T> struct TG{
257     vector<TP<T>> p;
258     size_t nxt(const size_t i) const {return i == p.size() - 1 ? 0 : i + 1;}
259     size_t pre(const size_t i) const {return i == 0 ? p.size() - 1 : i - 1;}
260     //求面积的二倍 (逆时针存点则为正)
261     T getArea2(){
262         int siz = p.size();
263         T sum = 0;
264         for(int i = 0; i < siz; i++){
265             sum += (p[i] ^ p[(i + 1) % siz]);
266         }
267         return sum;
268     }
269     //Winding, 判断点与多边形关系, {true, 0} -> 点在边上, {false, cnt} -> 回转数为0 -> 外部, 其他 -> 内部
270     pair<bool, int> winding(const Point &a) {
271         int cnt = 0;
272         for(int i = 0; i < p.size(); i++){
273             Point u = p[i], v = p[nxt(i)];
274             if(sgn((a - u) ^ (a - v)) == 0 && sgn((a - u)*(a - v)) <= 0) return {true, 0};
275             if(sgn(u.y - v.y) == 0) continue;
276             Line uv = {u, v - u};
277             if(u.y < v.y - eps && uv.toleft(a) <= 0) continue;
278             if(u.y > v.y + eps && uv.toleft(a) >= 0) continue;
279             if(u.y < a.y - eps && v.y >= a.y - eps) cnt++;
280             if(u.y >= a.y - eps && v.y < a.y - eps) cnt--;
281         }
282         return {false, cnt};
283     }
284     void print(){
285         cout << "[Polygon]\n";
286         for(int i = 0; i < p.size(); i++){
287             cout << i << " " << p[i].x << " " << p[i].y << '\n';
288         }
289     }
290 }; using Polygon = TG<long double>;
291
292
293 // +-----+
294 // |                                     |
295 // | Geometry Template Function       |
296 // |                                     |
297 // +-----+
298
299
300 //凸包点集调整 -> 起点变为下凸壳最左侧的点:
301 void adjustConvexHull(vector<Point> &P, vector<Point> &tmp){
302     int n = P.size(); tmp.resize(n);
303     int pos = -1;
304     long double minX = 1e60, maxY = -1e60;
305     for(int i = 0; i < n; i++){
306         if(P[i].x < minX || (fabs(P[i].x - minX) <= eps && P[i].y > maxY)){
307             pos = i;
308             minX = P[i].x; maxY = P[i].y;
309         }

```

```

310     }
311     int cnt = 0;
312     for(int i = pos; i < n; i++)    tmp[cnt++] = P[i];
313     for(int i = 0; i < pos; i++)    tmp[cnt++] = P[i];
314     for(int i = 0; i < n; i++)    P[i] = tmp[i];
315 }
316 //求解点集p的凸包（Andrew算法），逆时针存于点集ans中。
317 #define bk1(x) (x.back())
318 #define bk2(x) (*(x.rbegin() + 1))
319 void findConvexHull(vector<Point> p, vector<Point> &ans){
320     vector<Point> st;
321     sort(p.begin(), p.end(), [&](const Point &A, const Point &B){ return sgn(A.x - B.x) ? A.x < B.x : A.y <
B.y; });
322     for(Point u : p){
323         while(st.size() > 1 && ((bk1(st) - bk2(st)).toLeft(u - bk2(st))) <= 0){
324             st.pop_back();
325         }
326         st.push_back(u);
327     }
328     int k = st.size();
329     p.pop_back(); reverse(p.begin(), p.end());
330     for(Point u : p){
331         while(st.size() > k && ((bk1(st) - bk2(st)).toLeft(u - bk2(st))) <= 0){
332             st.pop_back();
333         }
334         st.push_back(u);
335     }
336     st.pop_back();
337     ans.clear();
338     for(auto x : st)    ans.push_back(x);
339 }
340
341 //叉积排序函数
342 bool argcmpC(const Point &a, const Point &b){
343     auto Quad = [](const Point &a){
344         if(a.y < -eps)    return 1;
345         if(a.y > +eps)    return 4;
346         if(a.x < -eps)    return 5;
347         if(a.x > +eps)    return 3;
348         return 2;
349     };
350     int qa = Quad(a), qb = Quad(b);
351     if(qa != qb)    return qa < qb;
352     auto cross = (a ^ b);
353     return cross > eps;
354 }
355
356 //极角序，自动去重，返回<方向，个数>的集合，需依赖上方的argcmpC函数
357 vector<pair<Point, int>> polarUniqueTrans(vector<Point> &p){
358     map<Point, int, decltype(&argcmpC)> uni{&argcmpC};
359     vector<pair<Point, int>> res;
360     for(auto x : p)    uni[x]++;
361     for(auto x : uni)    res.push_back(x);
362     return res;
363 }
364 //【已测试：P4525 「模板」自适应辛普森法 1】
365 //自适应simp积分，近似求int[a, b]
366 long double functionVal(long double x){
367     //returns f(x)
368     return 0;
369 }
370 long double simp(long double l, long double r){

```

```

371     long double mid = (L + r) / 2.0;
372     return (r - L) * (functionVal(L) + 4 * functionVal(mid) + functionVal(r)) / 6.0;
373 }
374 //需注意eps精度问题
375 long double asr(long double L, long double r, long double ans){
376     long double mid = (L + r) / 2.0;
377     long double vL = simp(L, mid), vR = simp(mid, r), tmp = vL + vR - ans;
378     if(fabs(tmp) <= eps) return ans;
379     else return asr(L, mid, vL) + asr(mid, r, vR);
380 }
381 //求解两圆公切线: 返回切线的条数, -1表示无穷多条切线, a -> A上的切点, b -> B上的切点
382 int getCircleTangents(Circle A, Circle B, vector<Point> &a, vector<Point> &b){
383     int cnt = 0;
384     if (A.r < B.r) {
385         swap(A, B);
386         swap(a, b);
387     }
388     double d2 = (A.c.x - B.c.x) * (A.c.x - B.c.x) + (A.c.y - B.c.y) * (A.c.y - B.c.y);
389     double rdiff = A.r - B.r;
390     double rsum = A.r + B.r;
391     if (sgn(d2 - rdiff * rdiff) < 0) return 0; // 内含
392     double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
393     //无限多条切线
394     if (sgn(d2) == 0 && sgn(A.r - B.r) == 0) return -1;
395     //内切, 一条切线
396     if (sgn(d2 - rdiff * rdiff) == 0) {
397         a.push_back(A.point(base));
398         b.push_back(B.point(base));
399         cnt++;
400         return cnt;
401     }
402     //有外公切线
403     double ang = acos(rdiff / sqrt(d2));
404     a.push_back(A.point(base + ang));
405     b.push_back(B.point(base + ang));
406     a.push_back(A.point(base - ang));
407     b.push_back(B.point(base - ang));
408     cnt += 2;
409     if(sgn(d2 - rsum * rsum) == 0){ // 一条内公切线
410         a.push_back(A.point(base));
411         b.push_back(B.point(PI + base));
412         cnt++;
413     } else if(sgn(d2 - rsum * rsum) > 0){ // 两条内公切线
414         double ang = acos(rsum / sqrt(d2));
415         a.push_back(A.point(base + ang));
416         b.push_back(B.point(PI + base + ang));
417         a.push_back(A.point(base - ang));
418         b.push_back(B.point(PI + base - ang));
419         cnt += 2;
420     }
421     return cnt;
422 }
423
424 /* 圆的反演: C2C, C2L, L2C */
425 /*
426 * 反演变换
427 * 适用于题目中存在多个圆/直线之间的相切关系的情况。
428 * 1. 圆O外的点的反演点在圆O内, 反之亦然, 圆O上的点的反演点为其自身。
429 * 2. 不过点O的圆, 其反演图形也是不过点O的圆。
430 * 3. 过点O的圆, 其反演图形是不过点O的直线。
431 * 4. 两个图形相切, 则他们的反演图形也相切。 (*)
432 * 5. 两个不经过反演点的外切的圆, 反演之后的图形为相交的两条直线。

```

```

433 * 如果其中一个圆经过反演点，那么反演之后的图形为一个圆和它的一条切线并且反演点和反演后的圆的圆心在切线的【同一
    侧】。
434 * 内切的话反演中心和反演圆的圆心在【异侧】。
435 */
436 //【已测试：HDU4773：Problem of Apollonius】
437 //点O在圆A外，求圆A的反演圆B，R是反演半径
438 Circle inversionC2C(Point O, long double R, Circle A){
439     long double OA = (A.c - O).Len();
440     long double RB = 0.5 * ((1 / (OA - A.r)) - (1 / (OA + A.r))) * R * R;
441     long double OB = OA * RB / A.r;
442     long double Bx = O.x + (A.c.x - O.x) * OB / OA;
443     long double By = O.y + (A.c.y - O.y) * OB / OA;
444     return Circle(Point(Bx, By), RB);
445 }
446 //【已测试：HDU4773：Problem of Apollonius】
447 //直线反演为过O点的圆B，R是反演半径
448 Circle inversionL2C(Point O, long double R, Point A, Point B){
449     Point P = Line(A, B).proj(O);
450     long double d = (O - P).Len();
451     long double RB = R * R / (2 * d);
452     Point VB = (P - O) / d * RB;
453     return Circle(O + VB, RB);
454 }
455 //【已测试：HDU4773：Problem of Apollonius】
456 //圆A经过反演中心O，反演得到直线L
457 Line inversionC2L(Point O, long double R, Circle A){
458     long double angle = (O - A.c).angle();
459     if(sgn(angle) < 0) angle += 2 * PI;
460     long double angleL = angle + PI / 2;
461     long double angleR = angle - PI / 2;
462     if(angleL < 0) angleL += 2 * PI;
463     if(angleR < 0) angleR += 2 * PI;
464     Point PL = A.point(angleL), PR = A.point(angleR), dirL = PL - O, dirR = PR - O;
465     long double disL = O.dis(PL), disrL = R * R / disL, disR = O.dis(PR), disrR = R * R / disR;
466     return Line(O + dirL.trunc(disrL), O + dirR.trunc(disrR));
467 }
468 /*
469 * 根据两个圆的位置关系来确定情况：
470 * (1) 两个圆内含，没有公共点，没有公切线
471 * (2) 两圆内切，有一个条公切线
472 * (3) 两圆完全重合，有无数条公切线
473 * (4) 两圆相交。有2条公切线
474 * (5) 两圆外切，有3条公切线
475 * (6) 两圆相离，有4条公切线
476 */
477
478
479 // +-----+
480 // | |
481 // | Geometry Template ExStruct |
482 // | |
483 // +-----+
484
485
486 /* 拓展自Polygon：求解点是否在凸包内，以及凸包外一点对该凸包的切线 */
487 struct Convex : Polygon{
488     //闵可夫斯基和，对应凸包
489     //【已测试：BZOJ2564. 集合的面积】
490     Convex operator +(const Convex &c){
491         const auto &p = this->p;
492         vector<Line> e1(p.size()), e2(c.p.size()), edge(p.size() + c.p.size());
493         Convex res;

```

```

494     res.p.reserve(p.size() + c.p.size());
495     for(int i = 0; i < p.size(); i++){
496         e1[i] = {p[i], p[this -> nxt(i)]};
497     }
498     for(int i = 0; i < c.p.size(); i++){
499         e2[i] = {c.p[i], c.p[c.nxt(i)]};
500     }
501     const auto cmp = [](const Line &u, const Line &v) { return argcmpC(u.e - u.s, v.e - v.s); };
502     rotate(e1.begin(), min_element(e1.begin(), e1.end(), cmp), e1.end());
503     rotate(e2.begin(), min_element(e2.begin(), e2.end(), cmp), e2.end());
504     merge(e1.begin(), e1.end(), e2.begin(), e2.end(), edge.begin(), cmp);
505     const auto check = [](const vector<Point> &p, const Point &u){
506         const auto back1 = p.back(), back2 = *prev(p.end(), 2);
507         return (back1 - back2).toLeft(u - back1) == 0 && (back1 - back2) * (u - back1) >= -eps;
508     };
509     auto u = e1[0].s + e2[0].s;
510     for(const auto &v : edge){
511         while(res.p.size() > 1 && check(res.p, u)){
512             res.p.pop_back();
513         }
514         res.p.push_back(u);
515         u = u + v.e - v.s;
516     }
517     if(res.p.size() > 1 && check(res.p, res.p[0]))    res.p.pop_back();
518     return res;
519 }
520 // 【已测试: Enclosure】
521 //O(logN)判断点是否在凸包内, 1: 在凸包内, 0: 在凸包外, -1: 在凸包上
522 int inConvex(const Point &a){
523     auto &p = this->p;
524     int l = 1, r = (int)(p.size()) - 2;
525     while(l <= r){
526         auto mid = (l + r) / 2;
527         auto t1 = (p[mid] - p[0]).toLeft(a - p[0]);
528         auto t2 = (p[mid + 1] - p[0]).toLeft(a - p[0]);
529         if(t1 >= 0 && t2 <= 0){
530             if(mid == 1 && Line(p[0], p[mid]).isOnSeg(a))    return -1;
531             if(mid + 1 == (int)(p.size()) - 1 && Line(p[0], p[mid + 1]).isOnSeg(a))    return -1;
532             if(Line(p[mid], p[mid + 1]).isOnSeg(a))    return -1;
533             return (p[mid + 1] - p[mid]).toLeft(a - p[mid]) > 0;
534         }
535         if(t1 < 0)    r = mid - 1;
536         else    l = mid + 1;
537     }
538     return false;
539 }
540 // 【已测试: USAC003FALL - Beauty Contest G】
541 //旋转卡壳, 求解内容取决于传入的函数F
542 template<typename F> void rotcaliper(const F &func) {
543     const auto &p = this->p;
544     const auto area = [](const Point &u, const Point &v, const Point &w){ return fabs((w - u) ^ (w - v)); };
545 };
546     for(int i = 0, j = 1; i < p.size(); i++){
547         const auto nexti = this -> nxt(i);
548         func(p[i], p[nexti], p[j]);
549         while(area(p[this -> nxt(j)], p[i], p[nexti]) >= area(p[j], p[i], p[nexti])){
550             j = this -> nxt(j);
551             func(p[i], p[nexti], p[j]);
552         }
553     }
554 // 【已测试: USAC003FALL - Beauty Contest G】

```

```

555 //旋转卡壳，求凸包直径（平方），需根据选定类型确定返回值（long long double/long long）
556 long double diameter2(){
557     const auto &p = this -> p;
558     if(p.size() == 1) return 0;
559     if(p.size() == 2) return p[0].dis2(p[1]);
560     long double ans = 0;
561     auto func = [&](const Point &u, const Point &v, const Point &w){
562         ans = max(ans, max(w.dis2(u), w.dis2(v)));
563     };
564     rotcaliper(func);
565     return ans;
566 }
567 //【已测试: Enclosure】
568 //O(logN)求解凸包外一点切线（返回其中一个切点），配合tangent使用
569 template<typename F> int extreme(const F &dir){
570     auto &p = this -> p;
571     auto check = [&](const int i){
572         return dir(p[i]).toLeft(p[this->nxt(i)] - p[i]) >= 0;
573     };
574     auto dir0 = dir(p[0]);
575     auto check0 = check(0);
576     if(check0 == 0 && check((int)(p.size()) - 1)) return 0;
577     int l = 0, r = p.size() - 1;
578     while(l < r){
579         auto mid = (l + r) / 2;
580         auto checkm = check(mid);
581         if(checkm == check0){
582             auto t = dir0.toLeft(p[mid] - p[0]);
583             if((check0 == 0 && t <= 0) || (check0 && t < 0)) checkm ^= 1;
584         }
585         if(checkm) l = mid + 1;
586         else r = mid;
587     }
588     return r;
589 }
590 //【已测试: Enclosure】
591 //凸包外一点切点（返回两个切点下标）
592 pair<int, int> tangent(const Point &a){
593     int i = extreme([&](const Point &u){ return u - a; });
594     int j = extreme([&](const Point &u){ return a - u; });
595     return {i, j};
596 }
597 //【已测试: A highway and the seven dwarfs】
598 //求直线与凸包上点的关系
599 pair<int, int> tangent(const Line &l, const Point &dir){
600     int i = extreme([&](...){ return dir; });
601     int j = extreme([&](...){ return -dir; });
602     return {i, j};
603 }
604 //O(logN)求直线是否穿过凸包
605 bool isLineCrossConvex(const Line &l, const Point &dir){
606     if(p.size() <= 1) return true;
607     if(p.size() == 2) return l.toLeft(p[0]) == l.toLeft(p[1]);
608     auto t = tangent(l, dir);
609     return l.toLeft(p[t.first]) == l.toLeft(p[t.second]);
610 }
611 };
612
613 //【已测试: Enclosure】
614 /* 拓展自Convex: 利用前缀和求解凸包中若干【连续】点构成的小凸包的面积 */
615 struct sumConvex : Convex{
616     vector<long double> sum;

```

```

617 void init(){
618     getSum();
619 }
620 void getSum(){
621     auto &p = this->p;
622     vector<long double> a(p.size());
623     for(int i = 0; i < p.size(); i++){
624         a[i] = p[this->pre(i)] ^ p[i];
625     }
626     sum.resize(p.size());
627     partial_sum(a.begin(), a.end(), sum.begin());
628 }
629 long double queryTangentSum(const Point &a){
630     auto &p = this->p;
631     pair<int, int> result = this->tangent(a);
632     int l = result.second, r = result.first;
633     return querySum(l, r);
634 }
635 long double querySum(){
636     return sum.back();
637 }
638 long double querySum(int l, int r){
639     if(l <= r) return sum[r] - sum[l] + (p[r] ^ p[l]);
640     return sum[p.size() - 1] - sum[l] + sum[r] + (p[r] ^ p[l]);
641 }
642 };
643 // 【已测试: [HNOI2012]射箭】
644 /* 半平面 (单个) */
645 struct Halfplane : Line{
646     Halfplane(){}
647     Halfplane(Point _s, Point _e){s = _s; e = _e;}
648     //叉积排序, 减少精度损失
649     bool operator <(const Halfplane &b){
650         Point A = e - s, B = b.e - b.s;
651         return argcmpC(A, B);
652     }
653 };
654
655 /* 半平面交 (集合) */
656 struct Halfplanes{
657     int n, st, ed, que[maxn];
658     Point p[maxn]; Halfplane hp[maxn];
659     //去重 & 便于判断非法情况
660     void unique(){
661         int m = 1;
662         for(int i = 1; i < n; i++){
663             if(!(sgn(hp[i] ^ hp[i - 1]) == 0 && sgn(hp[i] * hp[i - 1]) >= 0)){
664                 hp[m++] = hp[i];
665             }
666             else if(sgn((hp[m - 1].e - hp[m - 1].s) ^ (hp[i].s - hp[m - 1].s)) > 0){
667                 hp[m - 1] = hp[i];
668             }
669         }
670         n = m;
671     }
672     //True -> 存在半平面交
673     bool Halfplaneinsert(){
674         sort(hp, hp + n); unique();
675         que[st = 0] = 0; que[ed = 1] = 1;
676         p[1] = hp[0].crosspoint(hp[1]);
677         for(int i = 2; i < n; i++){
678             while(st < ed && sgn((hp[i].e - hp[i].s) ^ (p[ed] - hp[i].s)) < 0) ed--;

```

```

679         while(st < ed && sgn((hp[i].e - hp[i].s) ^ (p[st + 1] - hp[i].s)) < 0)    st++;
680         que[++ed] = i;
681         if(hp[i].parallel(hp[que[ed - 1]]))    return false;
682         p[ed] = hp[i].crosspoint(hp[que[ed - 1]]);
683     }
684     while(st < ed && sgn((hp[que[st]].e - hp[que[st]].s) ^ (p[ed] - hp[que[st]].s)) < 0)    ed--;
685     while(st < ed && sgn((hp[que[ed]].e - hp[que[ed]].s) ^ (p[st + 1] - hp[que[ed]].s)) < 0)    st++;
686     if(st + 1 >= ed)    return false;
687     return true;
688 }
689 //Halfplaneinsert True -> 得到半平面交对应的凸包
690 void getConvex(Polygon &con){
691     p[st] = hp[que[st]].crosspoint(hp[que[ed]]);
692     for(int j = st, i = 0; j <= ed; i++, j++){
693         con.p.push_back(p[j]);
694     }
695 }
696 //压入新的半平面
697 void push(Halfplane tmp){
698     hp[n++] = tmp;
699 }
700 };
701
702 //【已测试: 70D - Professor's task】
703 /* 动态凸包 (set维护): 需选用前三个点确定BASIC点, 而后进行维护, 不支持删除 */
704 Point DBASIC;
705 bool argcmpB(const Point &A, const Point &B){
706     Point p1 = A - DBASIC, p2 = B - DBASIC;
707     //此处可以换用叉积版本维护
708     long double len1 = p1.Len2(), len2 = p2.Len2();
709     long double ang1 = p1.angle(), ang2 = p2.angle();
710     if(sgn(ang1 - ang2) == 0)    return len1 < len2;
711     return ang1 < ang2;
712 }
713 struct DConvexHull{
714     set<Point, decltype(&argcmpB)> Set{&argcmpB};
715     void init(const Point &A, const Point &B, const Point &C){
716         //A,B,C为任意三点, 处理完毕后直接调用Insert即可
717         DBASIC = {(A.x + B.x + C.x) / 3,
718                 (A.y + B.y + C.y) / 3};
719         Set.insert(A); Set.insert(B); Set.insert(C);
720     }
721     set<Point>::iterator Pre(set<Point>::iterator it){
722         if(it == Set.begin())    it = Set.end();
723         return --it;
724     }
725     set<Point>::iterator Nxt(set<Point>::iterator it){
726         ++it;
727         return it == Set.end() ? Set.begin() : it;
728     }
729     //询问点是否在凸包内
730     bool Query(Point v){
731         auto it = Set.lower_bound(v);
732         if(it == Set.end())    it = Set.begin();
733         Point v1 = v - (*Pre(it));
734         Point v2 = (*it) - (*Pre(it));
735         return sgn(v1 ^ v2) <= 0;
736     }
737     //往凸包中插入一个新的点
738     void Insert(Point v){
739         if(Query(v))    return;
740         Set.insert(v);

```

```
741     auto it = Nxt(Set.find(v));
742     while(Set.size() > 3 && sgn((v - (*Nxt(it))) ^ ((*it) - (*Nxt(it)))) <= 0){
743         Set.erase(it);
744         it = Nxt(Set.find(v));
745     }
746     it = Pre(Set.find(v));
747     while(Set.size() > 3 && sgn((v - (*it)) ^ ((*it) - (*Pre(it)))) >= 0){
748         Set.erase(it);
749         it = Pre(Set.find(v));
750     }
751 }
752 };
753
754 signed main(void){
755     cout << "Helloworld!\n";
756 }
```

---