

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 #include <map>
5 #include <math.h>
6 #include <complex>
7 using namespace std;
8 #define rep(i,n) for (int i=0; i < (n); i++)
9
10 // 主に以下の資料を参考に作成した。
11 // - http://www.prefield.com/algorithm
12 // - http://www.deqnotes.net/acmicpc/2d\_geometry/
13 // - https://github.com/infnty/acm/tree/master/lib/geometry
14 // - サークルの先輩が作ったライブラリ
15
16 /* 基本要素 */
17
18 typedef double D; // 座標値の型。doubleかlong doubleを想定
19 typedef complex<D> P; // Point
20 typedef pair<P, P> L; // Line
21 typedef vector<P> VP;
22 const D EPS = 1e-9; // 許容誤差。問題によって変える
23 #define X real()
24 #define Y imag()
25 #define LE(n,m) ((n) < (m) + EPS)
26 #define GE(n,m) ((n) + EPS > (m))
27 #define EQ(n,m) (abs((n)-(m)) < EPS)
28
29 // 内積 dot(a,b) = |a||b|cosθ
30 D dot(P a, P b) {
31     return (conj(a)*b).X;
32 }
33 // 外積 cross(a,b) = |a||b|sinθ
34 D cross(P a, P b) {
35     return (conj(a)*b).Y;
36 }
37
38 // 点の進行方向
39 int ccw(P a, P b, P c) {
40     b -= a; c -= a;
41     if (cross(b,c) > EPS) return +1; // counter clockwise
42     if (cross(b,c) < -EPS) return -1; // clockwise
43     if (dot(b,c) < -EPS) return +2; // c--a--b on line
44     if (norm(b) < norm(c)) return -2; // a--b--c on line or a==b
45     return 0; // a--c--b on line or a==c or b==c
46 }
47
48 /* 交差判定 直線・線分は縮退してはならない。接する場合は交差するとみなす。isecはintersectの略 */
49
50 // 直線と点
51 bool isecLP(P a1, P a2, P b) {
52     return abs(ccw(a1, a2, b)) != 1; // return EQ(cross(a2-a1, b-a1), 0); と等価
53 }
54
55 // 直線と直線
56 bool isecLL(P a1, P a2, P b1, P b2) {
57     return !isecLP(a2-a1, b2-b1, 0) || isecLP(a1, b1, b2);
58 }
59
60 // 直線と線分
61 bool isecLS(P a1, P a2, P b1, P b2) {
62     return cross(a2-a1, b1-a1) * cross(a2-a1, b2-a1) < EPS;
63 }
64
65 // 線分と線分
66 bool isecSS(P a1, P a2, P b1, P b2) {
67     return ccw(a1, a2, b1)*ccw(a1, a2, b2) <= 0 &&
68         ccw(b1, b2, a1)*ccw(b1, b2, a2) <= 0;
69 }
70
71 // 線分と点
72 bool isecSP(P a1, P a2, P b) {

```

```

73 return !ccw(a1, a2, b);
74 }
75
76
77 /* 距離 各直線・線分は縮退してはならない */
78
79 // 点pの直線aへの射影点を返す
80 P proj(P a1, P a2, P p) {
81     return a1 + dot(a2-a1, p-a1)/norm(a2-a1) * (a2-a1);
82 }
83
84 // 点pの直線aへの反射点を返す
85 P reflection(P a1, P a2, P p) {
86     return 2.0*proj(a1, a2, p) - p;
87 }
88
89 D distLP(P a1, P a2, P p) {
90     return abs(proj(a1, a2, p) - p);
91 }
92
93 D distLL(P a1, P a2, P b1, P b2) {
94     return isecLL(a1, a2, b1, b2) ? 0 : distLP(a1, a2, b1);
95 }
96
97 D distLS(P a1, P a2, P b1, P b2) {
98     return isecLS(a1, a2, b1, b2) ? 0 : min(distLP(a1, a2, b1), distLP(a1, a2, b2));
99 }
100
101 D distSP(P a1, P a2, P p) {
102     P r = proj(a1, a2, p);
103     if (isecSP(a1, a2, r)) return abs(r-p);
104     return min(abs(a1-p), abs(a2-p));
105 }
106
107 D distSS(P a1, P a2, P b1, P b2) {
108     if (isecSS(a1, a2, b1, b2)) return 0;
109     return min(min(distSP(a1, a2, b1), distSP(a1, a2, b2)),
110               min(distSP(b1, b2, a1), distSP(b1, b2, a2)));
111 }
112
113 // 2直線の交点
114 P crosspointLL(P a1, P a2, P b1, P b2) {
115     D d1 = cross(b2-b1, b1-a1);
116     D d2 = cross(b2-b1, a2-a1);
117     if (EQ(d1, 0) && EQ(d2, 0)) return a1; // same line
118     if (EQ(d2, 0)) throw "kouten ga nai"; // 交点がない
119     return a1 + d1/d2 * (a2-a1);
120 }
121
122
123 /* 円 */
124
125 D distLC(P a1, P a2, P c, D r) {
126     return max(distLP(a1, a2, c) - r, 0.0);
127 }
128
129 D distSC(P a1, P a2, P c, D r) {
130     D dSqr1 = norm(c-a1), dSqr2 = norm(c-a2);
131     if (dSqr1 < r*r ^ dSqr2 < r*r) return 0; // 円が線分を包含するとき距離0ならここをORに変える
132     if (dSqr1 < r*r & dSqr2 < r*r) return r - sqrt(max(dSqr1, dSqr2));
133     return max(distSP(a1, a2, c) - r, 0.0);
134 }
135
136 VP crosspointLC(P a1, P a2, P c, D r) {
137     VP ps;
138     P ft = proj(a1, a2, c);
139     if (!GE(r*r, norm(ft-c))) return ps;
140
141     P dir = sqrt(max(r*r - norm(ft-c), 0.0)) / abs(a2-a1) * (a2-a1);
142     ps.push_back(ft + dir);
143     if (!EQ(r*r, norm(ft-c))) ps.push_back(ft - dir);
144     return ps;
145 }
146

```

```

147 D distCC(P a, D ar, P b, D br) {
148   D d = abs(a-b);
149   return GE(d, abs(ar-br)) ? max(d-ar-br, 0.0) : abs(ar-br) - d;
150 }
151
152 // 2円の交点
153 VP crosspointCC(P a, D ar, P b, D br) {
154   VP ps;
155   P ab = b-a;
156   D d = abs(ab);
157   D crL = (norm(ab) + ar*ar - br*br) / (2*d);
158   if (EQ(d, 0) || ar < abs(crL)) return ps;
159
160   P abN = ab * P(0, sqrt(ar*ar - crL*crL) / d);
161   P cp = a + crL/d * ab;
162   ps.push_back(cp + abN);
163   if (!EQ(norm(abN), 0)) ps.push_back(cp - abN);
164   return ps;
165 }
166
167 // 点pから円aへの接線の接点
168 VP tangentPoints(P a, D ar, P p) {
169   VP ps;
170   D sin = ar / abs(p-a);
171   if (!LE(sin, 1)) return ps; // ここでNaNも弾かれる
172   D t = M_PI_2 - asin(min(sin, 1.0));
173   ps.push_back(a + (p-a)*polar(sin, t));
174   if (!EQ(sin, 1)) ps.push_back(a + (p-a)*polar(sin, -t));
175   return ps;
176 }
177
178 // 2円の共通接線。返される各直線に含まれる頂点は円との接点となる
179 vector<L> tangentLines(P a, D ar, P b, D br) {
180   vector<L> ls;
181   D d = abs(b-a);
182   rep(i,2) {
183     D sin = (ar - (1-i*2)*br) / d;
184     if (!LE(sin*sin, 1)) break;
185     D cos = sqrt(max(1 - sin*sin, 0.0));
186     rep(j,2) {
187       P n = (b-a) * P(sin, (1-j*2)*cos) / d;
188       ls.push_back(L(a + ar*n, b + (1-i*2)*br*n));
189       if (cos < EPS) break; // 重複する接線を見逃す (重複していいならこの行不要)
190     }
191   }
192   return ls;
193 }
194
195 // 三角形の外心。点a,b,cは同一線線上にあってはならない
196 P circumcenter(P a, P b, P c) {
197   a = (a-c)*0.5;
198   b = (b-c)*0.5;
199   return c + crosspointLL(a, a*P(1,1), b, b*P(1,1));
200 }
201
202 // 点aと点bを通り、半径がrの円の中心を返す
203 VP circlesPointsRadius(P a, P b, D r) {
204   VP cs;
205   P abH = (b-a)*0.5;
206   D d = abs(abH);
207   if (d == 0 || d > r) return cs; // 必要なら !LE(d,r) として円1つになる側へ丸める
208   D dN = sqrt(r*r - d*d); // 必要なら max(r*r - d*d, 0) とする
209   P n = abH * P(0,1) * (dN / d);
210   cs.push_back(a + abH + n);
211   if (dN > 0) cs.push_back(a + abH - n);
212   return cs;
213 }
214
215 // 点aと点bを通り、直線lに接する円の中心
216 VP circlesPointsTangent(P a, P b, P l1, P l2) {
217   P n = (l2-l1) * P(0,1);
218   P m = (b-a) * P(0,0.5);
219   D rC = dot((a+b)*0.5-l1, n);
220   D qa = norm(n)*norm(m) - dot(n,m)*dot(n,m);

```

```

221 D qb = -rC * dot(n,m);
222 D qc = norm(n)*norm(m) - rC*rC;
223 D qd = qb*qb - qa*qc; // qa*k^2 + 2*qb*k + qc = 0
224
225 VP cs;
226 if (qd < -EPS) return cs;
227 if (EQ(qa, 0)) {
228     if (!EQ(qb, 0)) cs.push_back((a+b)*0.5 - m * (qc/qb/2));
229     return cs;
230 }
231 D t = -qb/qa;
232 cs.push_back((a+b)*0.5 + m * (t + sqrt(max(qd, 0.0))/qa));
233 if (qd > EPS) cs.push_back((a+b)*0.5 + m * (t - sqrt(max(qd, 0.0))/qa));
234 return cs;
235 }
236
237 // 点集合を含む最小の円の中心
238 P minEnclosingCircle(const VP& ps) {
239     P c;
240     double move = 0.5;
241     rep(i,39) { // 2^(-39-1) \approx 0.9e-12
242         rep(t,50) {
243             D max = 0;
244             int k = 0;
245             rep(j, ps.size()) if (max < norm(ps[j]-c)) {
246                 max = norm(ps[j]-c);
247                 k = j;
248             }
249             c += (ps[k]-c) * move;
250         }
251         move /= 2;
252     }
253     return c;
254 }
255
256
257 /* 多角形 */
258
259 // 頂点の順序 (sortやmax_elementに必要)
260 namespace std {
261     bool operator<(const P a, const P b) {
262         return a.X != b.X ? a.X < b.X : a.Y < b.Y;
263     }
264 }
265
266 // 凸包
267 VP convexHull(VP ps) { // 元の点集合がソートされていいならVP&に
268     int n = ps.size(), k = 0;
269     sort(ps.begin(), ps.end());
270     VP ch(2*n);
271     for (int i = 0; i < n; ch[k++] = ps[i++]) // lower-hull
272         while (k >= 2 && ccw(ch[k-2], ch[k-1], ps[i]) <= 0) --k; // 余計な点も含むなら == -1 とする
273     for (int i = n-2, t = k+1; i >= 0; ch[k++] = ps[i--]) // upper-hull
274         while (k >= t && ccw(ch[k-2], ch[k-1], ps[i]) <= 0) --k;
275     ch.resize(k-1);
276     return ch;
277 }
278
279 // 凸判定。縮退を認めないならccwの判定部分を != 1 とする
280 bool isCcwConvex(const VP& ps) {
281     int n = ps.size();
282     rep(i, n) if (ccw(ps[i], ps[(i+1) % n], ps[(i+2) % n]) == -1) return false;
283     return true;
284 }
285
286 // 凸多角形の内部判定 O(n)
287 // 点が領域内部なら1、境界上なら2、外部なら0を返す
288 int inConvex(P p, const VP& ps) {
289     int n = ps.size();
290     int dir = ccw(ps[0], ps[1], p);
291     rep(i, n) {
292         int ccwc = ccw(ps[i], ps[(i+1) % n], p);
293         if (!ccwc) return 2; // 線分上に存在
294         if (ccwc != dir) return 0;

```

```

295 }
296 return 1;
297 }
298
299 // 凸多角形の内部判定 O(logn)
300 // 点が領域内部なら1、境界上なら2、外部なら0を返す
301 int inCcwConvex(const VP& ps, P p) {
302     int n = ps.size();
303     P g = (ps[0] + ps[n / 3] + ps[n*2 / 3]) / 3.0;
304     if (g == p) return 1;
305     P gp = p - g;
306
307     int l = 0, r = n;
308     while (l + 1 < r) {
309         int mid = (l + r) / 2;
310         P gl = ps[l] - g;
311         P gm = ps[mid] - g;
312         if (cross(gl, gm) > 0) {
313             if (cross(gl, gp) >= 0 && cross(gm, gp) <= 0) r = mid;
314             else l = mid;
315         }
316         else {
317             if (cross(gl, gp) <= 0 && cross(gm, gp) >= 0) l = mid;
318             else r = mid;
319         }
320     }
321     r %= n;
322     D cr = cross(ps[l] - p, ps[r] - p);
323     return EQ(cr, 0) ? 2 : cr < 0 ? 0 : 1;
324 }
325
326 // 多角形の内部判定
327 // 点が領域内部なら1、境界上なら2、外部なら0を返す
328 int inPolygon(const VP& ps, P p) {
329     int n = ps.size();
330     bool in = false;
331     rep (i, n) {
332         P a = ps[i] - p;
333         P b = ps[(i + 1) % n] - p;
334         if (EQ(cross(a,b), 0) && LE(dot(a,b), 0)) return 2;
335         if (a.Y > b.Y) swap(a,b);
336         if ((a.Y*b.Y < 0 || (a.Y*b.Y < EPS && b.Y > EPS)) && LE(cross(a, b), 0)) in = !in;
337     }
338     return in;
339 }
340
341 // 凸多角形クリッピング
342 VP convexCut(const VP& ps, P a1, P a2) {
343     int n = ps.size();
344     VP ret;
345     rep(i,n) {
346         int ccwc = ccw(a1, a2, ps[i]);
347         if (ccwc != -1) ret.push_back(ps[i]);
348         int ccwn = ccw(a1, a2, ps[(i + 1) % n]);
349         if (ccwc * ccwn == -1) ret.push_back(crosspointLL(a1, a2, ps[i], ps[(i + 1) % n]));
350     }
351     return ret;
352 }
353
354 // 凸多角形の直径 (最遠点对)
355 pair<int, int> convexDiameter(const VP& ps) {
356     int n = ps.size();
357     int i = min_element(ps.begin(), ps.end()) - ps.begin();
358     int j = max_element(ps.begin(), ps.end()) - ps.begin();
359     int maxI, maxJ;
360     D maxD = 0;
361     rep(_, 2*n) {
362         if (maxD < norm(ps[i]-ps[j])) {
363             maxD = norm(ps[i]-ps[j]);
364             maxI = i;
365             maxJ = j;
366         }
367         if (cross(ps[i]-ps[(i+1) % n], ps[(j+1) % n]-ps[j]) <= 0) j = (j+1) % n;
368         else i = (i+1) % n;
369     }

```

```

369 }
370 return make_pair(maxI, maxJ);
371 }
372
373 // 多角形の符号付面積
374 D area(const VP& ps) {
375     D a = 0;
376     rep (i, ps.size()) a += cross(ps[i], ps[(i+1) % ps.size()]);
377     return a / 2;
378 }
379
380 // 多角形の幾何学的重心
381 P centroid(const VP& ps) {
382     int n = ps.size();
383     D aSum = 0;
384     P c;
385     rep (i, n) {
386         D a = cross(ps[i], ps[(i+1) % n]);
387         aSum += a;
388         c += (ps[i] + ps[(i+1) % n]) * a;
389     }
390     return 1 / aSum / 3 * c;
391 }
392
393 // ボロノイ領域
394 VP voronoiCell(P p, const VP& ps, const VP& outer) {
395     VP cl = outer;
396     rep (i, ps.size()) {
397         if (EQ(norm(ps[i]-p), 0)) continue;
398         P h = (p+ps[i])*0.5;
399         cl = convexCut(cl, h, h + (ps[i]-h)*P(0,1));
400     }
401     return cl;
402 }
403
404 /* 幾何グラフ */
405
406 struct Edge {
407     int from, to;
408     D cost;
409     Edge(int from, int to, D cost) : from(from), to(to), cost(cost) {}
410 };
411 struct Graph {
412     int n;
413     vector<vector<Edge>> > edges;
414     Graph(int n) : n(n), edges(n) {}
415     void addEdge(Edge e) {
416         edges[e.from].push_back(e);
417         edges[e.to].push_back(Edge(e.to, e.from, e.cost));
418     }
419 };
420
421 // 線分アレンジメント (線分の位置関係からグラフを作成)
422 Graph segmentArrangement(const vector<L>& segs, VP& ps) {
423     int n = segs.size();
424     rep (i, n) {
425         ps.push_back(segs[i].first);
426         ps.push_back(segs[i].second);
427         rep (j, i) {
428             if (isecSS(segs[i].first, segs[i].second, segs[j].first, segs[j].second))
429                 ps.push_back(crosspointLL(segs[i].first, segs[i].second, segs[j].first, segs[j].second));
430         }
431     }
432     sort(ps.begin(), ps.end());
433     ps.erase(unique(ps.begin(), ps.end()), ps.end());
434
435     int m = ps.size();
436     Graph gr(m);
437     vector<pair<D, int>> list;
438     rep (i, n) {
439         list.clear();
440         rep (j, m) {
441             if (isecSP(segs[i].first, segs[i].second, ps[j]))
442                 list.push_back(make_pair(norm(segs[i].first-ps[j]), j));

```

```

443     }
444     sort(list.begin(), list.end());
445     rep (j, list.size() - 1) {
446         int a = list[j].second;
447         int b = list[j+1].second;
448         gr.addEdge(Edge(a, b, abs(ps[a]-ps[b])));
449     }
450 }
451 return gr;
452 }
453
454 // 可視グラフ (点集合から見える位置へ辺を張ったグラフ)
455 Graph visibilityGraph(const VP& ps, const vector<VP>& objs) {
456     int n = ps.size();
457     Graph gr(n);
458     rep (i,n) rep (j,i) {
459         P a = ps[i], b = ps[j];
460         if (!EQ(norm(a-b), 0)) rep (k, objs.size()) {
461             const VP& obj = objs[k];
462             int inStA = inConvex(a, obj);
463             int inStB = inConvex(b, obj);
464             if ((inStA ^ inStB) % 2 || inStA * inStB != 1 && inConvex((a+b)*0.5, obj) == 1) goto skip;
465             rep (l, obj.size()) {
466                 P cur = obj[l];
467                 P next = obj[(l + 1) % obj.size()];
468                 if (isecSS(a, b, cur, next) && !isecSP(cur, next, a) && !isecSP(cur, next, b)) goto skip;
469             }
470         }
471         gr.addEdge( Edge(i, j, abs(a-b)) );
472         skip: {}
473     }
474     return gr;
475 }
476
477
478 /* その他 */
479
480 // 重複する線分を併合する
481 vector<L> mergeSegments(vector<L> segs) {
482     int n = segs.size();
483     rep (i,n) if (segs[i].second < segs[i].first) swap(segs[i].second, segs[i].first);
484
485     rep (i,n) rep (j,i) {
486         L &l1 = segs[i], &l2 = segs[j];
487         if (EQ(cross(l1.second-l1.first, l2.second-l2.first), 0)
488             && isecLP(l1.first, l1.second, l2.first)
489             && ccw (l1.first, l1.second, l2.second) != 2
490             && ccw (l2.first, l2.second, l1.second) != 2) {
491             segs[j] = L(min(l1.first, l2.first), max(l1.second, l2.second));
492             segs[i--] = segs[--n];
493             break;
494         }
495     }
496     segs.resize(n);
497     return segs;
498 }
499
500
501 // この辺にコードを載せるほどでもないが重要な定理とか図とか書いておくとよい気がします
502
503 // 余弦定理
504 // △ABC において、a = BC, b = CA, c = AB としたとき
505 //  $a^2 = b^2 + c^2 - 2bc \cos \angle CAB$ 
506
507 // ヘロンの公式
508 // 3辺の長さがa,b,cである三角形の面積T
509 //  $T = \sqrt{s(s-a)(s-b)(s-c)}$ ,  $s = (a+b+c)/2$ 
510
511 // ピックの定理
512 // 多角形の頂点が全て格子点上にあり、内部に穴がないとき
513 //  $S = i + b/2 - 1$  (S: 多角形の面積, i: 多角形の内部にある格子点の数, b: 辺上の格子点の数)

```