

```

1  #include <vector>
2  #include <algorithm>
3  #include <cstdlib>
4  #include <complex>
5  #include <iostream>
6  using namespace std;
7
8  typedef complex<long double> point;
9  #define sz(a) ((int)(a).size())
10 #define all(n) (n).begin(),(n).end()
11 #define EPS 1e-9
12 #define O0 1e9
13 #define X real()
14 #define Y imag()
15 #define vec(a,b) ((b)-(a))
16 #define polar(r,t) ((r)*exp(point(0,t)))
17 #define angle(v) (atan2((v).Y,(v).X))
18 #define length(v) ((long double)hypot((v).Y,(v).X))
19 #define lengthSqr(v) (dot(v,v))
20 #define dot(a,b) ((conj(a)*(b)).real())
21 #define cross(a,b) ((conj(a)*(b)).imag())
22 #define rotate(v,t) (polar(v,t))
23 #define rotateabout(v,t,a) (rotate(vec(a,v),t)+(a))
24 #define reflect(p,m) ((conj((p)/(m)))*(m))
25 #define normalize(p) ((p)/length(p))
26 #define same(a,b) (lengthSqr(vec(a,b))<EPS)
27 #define mid(a,b) (((a)+(b))/point(2,0))
28 #define perp(a) (point(-(a).Y,(a).X))
29 #define colliner pointOnLine
30
31 enum STATE {
32     IN, OUT, BOUNDARY
33 };
34
35 bool intersect(const point &a, const point &b, const point &p, const point &q,
36               point &ret) {
37
38     //handle degenerate cases
39
40     double d1 = cross(p - a, b - a);
41     double d2 = cross(q - a, b - a);
42     ret = (d1 * q - d2 * p) / (d1 - d2);
43     if(fabs(d1 - d2) > EPS) return 1;
44     return 0;
45 }
46
47 bool pointOnLine(const point& a, const point& b, const point& p) {
48     if(same(a,b)) return same(a,p);
49     return fabs(cross(vec(a,b),vec(a,p))) < EPS;
50 }
51
52 bool pointOnRay(const point& a, const point& b, const point& p) {
53     if(same(a,b)) return same(a,p);
54     point v1 = normalize(vec(a,b));
55     point v2 = normalize(vec(a,p));
56     return same(v1,v2);
57 }
58
59 bool pointOnSegment(const point& a, const point& b, const point& p) {
60     if (same(a, b))
61         return same(a,p);
62
63     return pointOnRay(a, b, p) && pointOnRay(b, a, p);
64 }
65
66 long double pointLineDist(const point& a, const point& b, const point& p) {
67     if (same(a,b))
68         return hypot(a.X - p.X, a.Y - p.Y);
69

```

```

70         return fabs(cross(vec(a,b),vec(a,p)) / length(vec(a,b)));
71     }
72
73     long double pointSegmentDist(const point& a, const point& b, const point& p) {
74         if (dot(vec(a,b),vec(a,p)) < EPS)
75             return length(vec(a,p));
76         if (dot(vec(b,a),vec(b,p)) < EPS)
77             return length(vec(b,p));
78         return pointLineDist(a, b, p);
79     }
80
81     int lineLatticePointsCount(int x1, int y1, int x2, int y2) {
82         return abs(gcd(x1 - x2, y1 - y2)) + 1;
83     }
84
85     long double triangleAreaBH(long double b, long double h) {
86         return b * h / 2;
87     }
88
89     long double triangleArea2sidesAngle(long double a, long double b, long double
90     t) {
91         return fabs(a * b * sin(t) / 2);
92     }
93
94     long double triangleArea2anglesSide(long double t1, long double t2,
95     long double s) {
96         return fabs(s * s * sin(t1) * sin(t2) / (2 * sin(t1 + t2)));
97     }
98
99     long double triangleArea3sides(long double a, long double b, long double c) {
100         long double s = ((a + b + c) / 2);
101         return sqrt(s * (s - a) * (s - b) * (s - c));
102     }
103
104     long double triangleArea3points(const point& a, const point& b, const point&
105     c) {
106         return fabs(cross(a,b) + cross(b,c) + cross(c,a)) / 2;
107     }
108
109     //count interior
110     int picksTheorm(int a, int b) {
111         return a - b / 2 + 1;
112     }
113
114     //get angle opposite to side a
115     long double cosRule(long double a, long double b, long double c) {
116         // Handle denom = 0
117         long double res = (b * b + c * c - a * a) / (2 * b * c);
118         if (res > 1)
119             res = 1;
120         if (res < -1)
121             res = -1;
122         return acos(res);
123     }
124
125     long double sinRuleAngle(long double s1, long double s2, long double a1) {
126         // Handle denom = 0
127         long double res = s2 * sin(a1) / s1;
128         if (res > 1)
129             res = 1;
130         if (res < -1)
131             res = -1;
132         return asin(res);
133     }
134
135     long double sinRuleSide(long double s1, long double a1, long double a2) {
136         // Handle denom = 0
137         long double res = s1 * sin(a2) / sin(a1);
138         return fabs(res);
139     }

```

```

137 }
138
139 int circleLineIntersection(const point& p0, const point& p1, const point& cen,
140     long double rad, point& r1, point & r2) {
141
142     if (same(p0,p1)){
143         if(fabs(lengthSqr(vec(p0,cen))-(rad*rad)) < EPS)
144         {
145             r1 = r2 = p0;
146             return 1;
147         }
148         return 0;
149     }
150     long double a, b, c, t1, t2;
151     a = dot(p1-p0,p1-p0);
152     b = 2 * dot(p1-p0,p0-cen);
153     c = dot(p0-cen,p0-cen) - rad * rad;
154     double det = b * b - 4 * a * c;
155     int res;
156     if (fabs(det) < EPS)
157         det = 0, res = 1;
158     else if (det < 0)
159         res = 0;
160     else
161         res = 2;
162     det = sqrt(det);
163     t1 = (-b + det) / (2 * a);
164     t2 = (-b - det) / (2 * a);
165     r1 = p0 + t1 * (p1 - p0);
166     r2 = p0 + t2 * (p1 - p0);
167     return res;
168 }
169
170 int circleCircleIntersection(const point &c1, const long double&r1,
171     const point &c2, const long double&r2, point &res1, point
172     &res2) {
173     if (same(c1,c2) && fabs(r1 - r2) < EPS) {
174         res1 = res2 = c1;
175         return fabs(r1) < EPS ? 1 : 00;
176     }
177     long double len = length(vec(c1,c2));
178     if (fabs(len - (r1 + r2)) < EPS || fabs(fabs(r1 - r2) - len) < EPS) {
179         point d, c;
180         long double r;
181         if (r1 > r2)
182             d = vec(c1,c2), c = c1, r = r1;
183         else
184             d = vec(c2,c1), c = c2, r = r2;
185         res1 = res2 = normalize(d) * r + c;
186         return 1;
187     }
188     if (len > r1 + r2 || len < fabs(r1 - r2))
189         return 0;
190     long double a = cosRule(r2, r1, len);
191     point clc2 = normalize(vec(c1,c2)) * r1;
192     res1 = rotate(clc2,a) + c1;
193     res2 = rotate(clc2,-a) + c1;
194     return 2;
195 }
196
197 void circle2(const point& p1, const point& p2, point& cen, long double& r) {
198     cen = mid(p1,p2);
199     r = length(vec(p1,p2)) / 2;
200 }
201
202 bool circle3(const point& p1, const point& p2, const point& p3, point& cen,
203     long double& r) {
204     point m1 = mid(p1,p2);
205     point m2 = mid(p2,p3);

```

```

205     point perp1 = perp(vec(p1,p2));
206     point perp2 = perp(vec(p2,p3));
207     bool res = intersect(m1, m1 + perp1, m2, m2 + perp2, cen);
208     r = length(vec(cen,p1));
209     return res;
210 }
211
212 STATE circlePoint(const point & cen, const long double & r, const point& p) {
213     long double lensqr = lengthSqr(vec(cen,p));
214     if (fabs(lensqr - r * r) < EPS)
215         return BOUNDARY;
216     if (lensqr < r * r)
217         return IN;
218     return OUT;
219 }
220
221 int tangentPoints(const point & cen, const long double & r, const point& p,
222     point &r1, point &r2) {
223     STATE s = circlePoint(cen, r, p);
224     if (s != OUT) {
225         r1 = r2 = p;
226         return s == BOUNDARY;
227     }
228     point cp = vec(cen,p);
229     long double h = length(cp);
230     long double a = acos(r / h);
231     cp = normalize(cp) * r;
232     r1 = rotate(cp,a) + cen;
233     r2 = rotate(cp,-a) + cen;
234     return 2;
235 }
236
237 // minimum enclosing circle
238 //init p array with the points and ps with the number of points
239 //cen and rad are result circle
240 //you must call random_shuffle(p,p+ps); before you call mec
241 #define MAXPOINTS 100000
242 point p[MAXPOINTS], r[3], cen;
243 int ps, rs;
244 long double rad;
245
246 void mec() {
247     if (rs == 3) {
248         circle3(r[0], r[1], r[2], cen, rad);
249         return;
250     }
251     if (rs == 2 && ps == 0) {
252         circle2(r[0], r[1], cen, rad);
253         return;
254     }
255     if (!ps) {
256         cen = r[0];
257         rad = 0;
258         return;
259     }
260     ps--;
261     mec();
262     if (circlePoint(cen, rad, p[ps]) == OUT) {
263         r[rs++] = p[ps];
264         mec();
265         rs--;
266     }
267     ps++;
268 }
269
270
271 //to check if the points are sorted anti-clockwise or clockwise
272 //remove the fabs at the end and it will return -ve value if clockwise
273 long double polygonArea(const vector<point>&p) {

```

```

274         long double res = 0;
275         for (int i = 0; i < sz(p); i++) {
276             int j = (i + 1) % sz(p);
277             res += cross(p[i], p[j]);
278         }
279         return fabs(res) / 2;
280     }
281
282     // return the centroid point of the polygon
283     // The centroid is also known as the "centre of gravity" or the "center of
284     // mass". The position of the centroid
285     // assuming the polygon to be made of a material of uniform density.
286     point polyginCentroid(vector<point> &polygon) {
287         point res(0, 0);
288         long double a = 0;
289         for (int i = 0; i < (int) polygon.size(); i++) {
290             int j = (i + 1) % polygon.size();
291
292             res.X += (polygon[i].X + polygon[j].X) * (polygon[i].X *
293             polygon[j].Y
294                     - polygon[j].X * polygon[i].Y);
295             res.Y += (polygon[i].Y + polygon[j].Y) * (polygon[i].X *
296             polygon[j].Y
297                     - polygon[j].X * polygon[i].Y);
298             a += polygon[i].X * polygon[j].Y - polygon[i].Y * polygon
299             [j].X;
300         }
301         a *= 0.5;
302         res.X /= 6 * a;
303         res.Y /= 6 * a;
304
305         return res;
306     }
307
308     int picksTheorm(vector<point>& p) {
309         long double area = 0;
310         int bound = 0;
311         for (int i = 0; i < sz(p); i++) {
312             int j = (i + 1) % sz(p);
313             area += cross(p[i], p[j]);
314             point v = vec(p[i], p[j]);
315             bound += abs(__gcd((int) v.X, (int) v.Y));
316         }
317         area /= 2;
318         area = fabs(area);
319         return round(area - bound / 2 + 1);
320     }
321
322     void polygonCut(const vector<point>& p, const point&a, const point&b, vector<
323     point>& res) {
324         res.clear();
325         for (int i = 0; i < sz(p); i++) {
326             int j = (i + 1) % sz(p);
327             bool in1 = cross(vec(a, b), vec(a, p[i])) > EPS;
328             bool in2 = cross(vec(a, b), vec(a, p[j])) > EPS;
329             if (in1)
330                 res.push_back(p[i]);
331             if (in1 ^ in2) {
332                 point r;
333                 intersect(a, b, p[i], p[j], r);
334                 res.push_back(r);
335             }
336         }
337     }
338

```

```

339 //assume that both are anti-clockwise
340 void convexPolygonIntersect(const vector<point>& p, const vector<point>& q,
341     vector<point>& res) {
342     res = q;
343     for (int i = 0; i < sz(p); i++) {
344         int j = (i + 1) % sz(p);
345         vector<point> temp;
346         polygonCut(res, p[i], p[j], temp);
347         res = temp;
348         if (res.empty())
349             return;
350     }
351 }
352
353 void voronoi(const vector<point> &pnts, const vector<point>& rect, vector<
354     vector<point> > &res) {
355     res.clear();
356     for (int i = 0; i < sz(pnts); i++) {
357         res.push_back(rect);
358         for (int j = 0; j < sz(pnts); j++) {
359             if (j == i)
360                 continue;
361             point p = perp(vec(pnts[i], pnts[j]));
362             point m = mid(pnts[i], pnts[j]);
363             vector<point> temp;
364             polygonCut(res.back(), m, m + p, temp);
365             res.back() = temp;
366         }
367     }
368 }
369
370 STATE pointInPolygon(const vector<point>& p, const point &pnt) {
371     point p2 = pnt + point(1, 0);
372     int cnt = 0;
373     for (int i = 0; i < sz(p); i++) {
374         int j = (i + 1) % sz(p);
375         if (pointOnSegment(p[i], p[j], pnt))
376             return BOUNDARY;
377         point r;
378         intersect(pnt, p2, p[i], p[j], r);
379         if (!pointOnRay(pnt, p2, r))
380             continue;
381         if (same(r, p[i]) || same(r, p[j]))
382             if (fabs(r.Y - min(p[i].Y, p[j].Y)) < EPS)
383                 continue;
384         if (!pointOnSegment(p[i], p[j], r))
385             continue;
386         cnt++;
387     }
388     return cnt & 1 ? IN : OUT;
389 }
390
391 struct cmp {
392     point about;
393     cmp(point c) {
394         about = c;
395     }
396     bool operator()(const point& p, const point& q) const {
397         double cr = cross(vec(about, p), vec(about, q));
398         if (fabs(cr) < EPS)
399             return make_pair(p.Y, p.X) < make_pair(q.Y, q.X);
400         return cr > 0;
401     }
402 };
403
404 void sortAntiClockWise(vector<point>& pnts){
405     point mn(1 / 0.0, 1 / 0.0);
406     for (int i = 0; i < sz(pnts); i++)
407         if (make_pair(pnts[i].Y, pnts[i].X) < make_pair(mn.Y, mn.X))

```

```

408             mn = pnts[i];
409
410             sort(all(pnts),cmp(mn));
411     }
412
413     void convexHull(vector<point> pnts, vector<point> &convex) {
414         sortAntiClockWise(pnts);
415         convex.clear();
416         convex.push_back(pnts[0]);
417         if (sz(pnts) == 1)
418             return;
419         convex.push_back(pnts[1]);
420         for (int i = 2; i <= sz(pnts); i++) {
421             point c = pnts[i % sz(pnts)];
422             while (sz(convex) > 1) {
423                 point b = convex.back();
424                 point a = convex[sz(convex) - 2];
425                 if (cross(vec(b,a),vec(b,c)) < -EPS)
426                     break;
427                 convex.pop_back();
428             }
429             if (i < sz(pnts))
430                 convex.push_back(pnts[i]);
431         }
432     }

```