

```

1  // 幾何ライブラリ 2次元ベクトル内積/外積 線分距離 直線交点 凸包 円
2  #include <bits/stdc++.h>
3
4  using namespace std;
5
6  using ll = long long;
7  using vi = vector<int>;
8  using vll = vector<ll>;
9  using vvi = vector<vector<int>>;
10 using vvll = vector<vector<ll>>;
11
12 double eps = 0.0000001;
13 // asin(1.) * 2
14 double pi = 3.14159265358979323846264338327950288;
15 using p2 = complex<double>;
16 // x: real
17 // y: imag
18
19 double det(p2 v1, p2 v2) {
20     return v1.real() * v2.imag() - v1.imag() * v2.real();
21 }
22
23 double dot(p2 v1, p2 v2) {
24     return v1.real() * v2.real() + v1.imag() * v2.imag();
25 }
26
27 double dist2(p2 v) {
28     return dot(v, v);
29 }
30
31 double dist(p2 v) {
32     return sqrt(dist2(v));
33 }
34
35 bool same(double x, double y) { return abs(x - y) < eps; }
36
37 double dist2(p2 l1, p2 l2) { return dot(l1 - l2, l1 - l2); }
38
39 double dist(p2 l1, p2 l2) {
40     return sqrt(dist2(l1, l2));
41 }
42
43 int ccw(p2 a, p2 b, p2 c) {
44     b -= a;
45     c -= a;
46     if (det(b, c) > eps) return 1;
47     if (det(b, c) < -eps) return -1;
48     if (dot(b, c) < -eps) return 2;
49     if (dist2(b) < dist2(c)) return -2;
50     return 0;

```

```

51 }
52
53 auto p2comp = [](const p2 &l, const p2 &r) {
54     if (abs(l.real() - r.real()) > eps)
55         return l.real() < r.real();
56     return l.imag() < r.imag();
57 };
58
59 struct Line {
60     p2 st, ed;
61
62     Line(p2 st, p2 ed) : st(st), ed(ed) {}
63
64     Line(double x1, double y1, double x2, double y2)
65         : st(p2(x1, y1)), ed(p2(x2, y2)) {}
66
67     Line(p2 st, double x, double y) : st(st), ed(p2(x, y)) {}
68
69     Line(double x, double y, p2 ed) : st(p2(x, y)), ed(ed) {}
70
71     double dist() { return sqrt(dist2(st, ed)); }
72
73     bool isPalla(Line l) { return abs(det(ed - st, l.ed - l.st)) < eps; }
74
75     double x() { return ed.real() - st.real(); }
76
77     double y() { return ed.imag() - st.imag(); }
78
79     p2 v() { return ed - st; }
80 };
81
82 // l1.st + (l1.st - l1.ed) * r.first = l2.st + (l2.st - l2.ed) * r.second
83 // 方程式を満たす(r.first, r.second)を返す
84 // l1.isPalla(l2) => (nan, nan)
85 pair<double, double> interP(Line l1, Line l2) {
86     double a = l1.x();
87     double b = -l2.x();
88     double c = l1.y();
89     double d = -l2.y();
90     double inv = 1. / (a * d - c * b);
91     double e1 = -l1.st.real() + l2.st.real();
92     double e2 = -l1.st.imag() + l2.st.imag();
93     return make_pair((d * e1 - b * e2) * inv, (-c * e1 + a * e2) * inv);
94 }
95
96 bool intersec(Line l1, Line l2) {
97     if (!l1.isPalla(l2))
98         return false;
99     auto r = interP(l1, l2);
100     return eps < r.first && r.first < 1. - eps && eps < r.second &&

```

```

101     r.second < 1. - eps;
102 }
103
104 double inter_r(Line l, p2 c) {
105     p2 a = l.st, b = l.ed;
106     return dot(a - c, l.v()) / dist2(l.v());
107 }
108
109 double dist(Line l, p2 c) {
110     double r = inter_r(l, c);
111     if (r < -eps) return dist(l.st, c);
112     if (1. + eps < r) return dist(l.ed, c);
113     return dist(l.st + l.v() * r, c);
114 }
115
116 p2 nearest(Line l, p2 c) {
117     double r = inter_r(l, c);
118     if (r < -eps) return l.st;
119     if (1. + eps < r) return l.ed;
120     return l.st + l.v() * r;
121 }
122
123 double dist(Line l1, Line l2) {
124     return min(min(dist(l1, l2.st), dist(l1, l2.ed)), min(dist(l2, l1.st), dist(l2, l1.ed)));
125 }
126
127 struct Poly {
128     vector<p2> ps;
129     double d;
130
131     Poly(vector<p2> ps) : ps(ps) {
132         d = 0;
133         for (int i = 0; i < ps.size(); i++) d += dist(ps[i], ps[(i + 1) % ps.size()]);
134     }
135
136     // 頂点上/辺上は微妙
137     bool include(p2 p) {
138         // 半直線
139         Line l(p, p2(-10000, -1));
140         int c = 0;
141         for (int i = 0; i < ps.size(); i++) {
142             if (intersec(l, Line(ps[i], ps[(i + 1) % ps.size()]))) c++;
143         }
144         return c % 2 == 1;
145     }
146
147     bool include(p2 p, bool on_vert, bool on_edge) {
148         for (auto &q : ps) if (dist(p, q) < eps) return on_vert;
149         for (int i = 0; i < ps.size(); i++) {
150             if (ccw(ps[i], ps[(i + 1) % ps.size()], p) == 0) return on_edge;

```

```

151     }
152     return include(p);
153 }
154
155 bool intersec(Line l) {
156     for (int i = 0; i < ps.size(); i++) {
157         if (intersec(l, Line(ps[i], ps[(i + 1) % ps.size()])))
158             return true;
159     }
160     return false;
161 }
162 };
163
164 struct Circle {
165     p2 p;
166     double r;
167
168     Circle(p2 p, double r) : p(p), r(r) {}
169
170     bool include(p2 l) { return dist2(p, l) < r * r + eps; }
171
172     // 円同士の交点
173     // 存在すれば2つ
174     vector<p2> intersec(Circle c) {
175         p2 diff = c.p - p;
176         double dist = dot(diff, diff);
177         double a = (dist + r * r - c.r * c.r) / 2.;
178         double D = dist * r * r - a * a;
179         if (D < eps)
180             return vector<p2>();
181         double Dsqrt = sqrt(D);
182         vector<p2> ps;
183         ps.emplace_back((a * diff.real() + diff.imag() * Dsqrt) / dist + p.real(),
184                         (a * diff.imag() - diff.real() * Dsqrt) / dist + p.imag());
185         ps.emplace_back((a * diff.real() - diff.imag() * Dsqrt) / dist + p.real(),
186                         (a * diff.imag() + diff.real() * Dsqrt) / dist + p.imag());
187         return ps;
188     }
189 };
190
191 // 半時計回り
192 struct ConX {
193     vector<p2> ps;
194
195     // graham scan
196     // ref: プログラミングコンテストチャレンジブック p233
197     ConX(vector<p2> v) {
198         sort(v.begin(), v.end(), p2comp);
199
200         int k = 0, n = v.size();

```

```

201     ps.resize(n * 2);
202     for (int i = 0; i < n; i++) {
203         while (k > 1 && det(ps[k - 1] - ps[k - 2], v[i] - ps[k - 1]) < eps)
204             k--;
205         ps[k++] = v[i];
206     }
207     for (int i = n - 2, t = k; i >= 0; i--) {
208         while (k > t && det(ps[k - 1] - ps[k - 2], v[i] - ps[k - 1]) < eps)
209             k--;
210         ps[k++] = v[i];
211     }
212     ps.resize(k - 1);
213 }
214
215 Poly toPoly() {
216     return Poly(ps);
217 }
218
219 size_t size() { return ps.size(); }
220 };
221
222 int n;
223 vector<double> rs;
224 vector<p2> ps;
225 vector<Circle> cs;
226
227 bool f(double l) {
228     cs.clear();
229     for (int i = 0; i < n; i++) {
230         double rr = rs[i] * rs[i] - l * l;
231         if (rr < eps)
232             return false;
233         cs.emplace_back(ps[i], sqrt(rr));
234     }
235     vector<p2> may;
236     for (int i = 0; i < n; i++) {
237         may.push_back(cs[i].p);
238         for (int j = i + 1; j < n; j++) {
239             auto v = cs[i].intersec(cs[j]);
240             if (v.size() == 0)
241                 continue;
242             may.push_back(v[0]);
243             may.push_back(v[1]);
244         }
245     }
246     for (auto &p : may) {
247         bool ok = true;
248         for (auto &c : cs) {
249             if (!c.include(p)) {
250                 ok = false;

```

```
251     break;
252 }
253 }
254 if (ok)
255     return true;
256 }
257 return false;
258 }
259
260 int main() {
261     printf("%.20lf\n", asin(1.) * 2);
262     cin.tie(nullptr);
263     ios::sync_with_stdio(false);
264     Poly p(vector<p2>({p2(0, 0), p2(1, 0.5), p2(2, 0), p2(2, 2), p2(1, 1.5), p2(0, 2)}))
265 );
266     vector<double> xy({-1, 0, 1, 2, 3});
267     for (auto &x : xy)
268         for (auto &y : xy) {
269             cerr << x << " " << y << " " << (p.include(p2(x, y), true, true) ? "YES" : "NO")
270             << endl;
271         }
272     return 0;
273 }
```