#include <cstdio>  
#include <cstring>  
#include <cmath>  
#include <vector>  
#include <complex>  
#include <algorithm>  
  
using namespace std;  
typedef pair<int,int> Pii;  
const double pi = acos(-1.);  
const double eps = 1e-10;  
  
inline int sgn(double x) { return x < -eps ? -1 : x > eps ? 1 : 0;}  
inline double getDistance(double x, double y) { return sqrt(x \* x + y \* y); }  
inline double torad(double deg) { return deg / 180 \* pi; }  
  
struct Point {  
 double x, y;  
 Point (double x = 0, double y = 0): x(x), y(y) {}  
 void read () { scanf("%lf%lf", &x, &y); }  
 void write () { printf("(%f, %f)\n", x, y); }  
  
 bool operator == (const Point& u) const { return sgn(x - u.x) == 0 && sgn(y - u.y) == 0; }  
 bool operator != (const Point& u) const { return !(\*this == u); }  
 bool operator < (const Point& u) const { return sgn(x - u.x) < 0 || (sgn(x-u.x)==0 && sgn(y-u.y) < 0); }  
 bool operator > (const Point& u) const { return u < \*this; }  
 bool operator <= (const Point& u) const { return \*this < u || \*this == u; }  
 bool operator >= (const Point& u) const { return \*this > u || \*this == u; }  
 Point operator + (const Point& u) { return Point(x + u.x, y + u.y); }  
 Point operator - (const Point& u) { return Point(x - u.x, y - u.y); }  
 Point operator \* (const double u) { return Point(x \* u, y \* u); }  
 Point operator / (const double u) { return Point(x / u, y / u); }  
 double operator ^ (const Point u) { return x \* u.y - y \* u.x; }  
};  
typedef Point Vector;  
typedef vector<Point> Polygon;  
  
struct Line {  
 double a, b, c;  
 Line (double a = 0, double b = 0, double c = 0): a(a), b(b), c(c) {}  
};  
  
struct DirLine {  
 Point p; Vector v; double ang;  
 DirLine () {}  
 DirLine (Point p, Vector v): p(p), v(v) { ang = atan2(v.y, v.x); }  
 bool operator < (const DirLine& u) const { return ang < u.ang; }  
};  
  
struct Circle {  
 Point o; double r;  
 Circle () {}  
 Circle (Point o, double r = 0): o(o), r(r) {}  
 void read () { o.read(), scanf("%lf", &r); }  
 Point point(double rad) { return Point(o.x + cos(rad)\*r, o.y + sin(rad)\*r); }  
 double getArea (double rad) { return rad \* r \* r / 2; }  
};  
  
namespace Punctual {  
 double getDistance(Point a, Point b) { double x=a.x-b.x, y=a.y-b.y; return sqrt(x\*x + y\*y); }  
};  
  
namespace Vectorial {  
 /\* 点积: 两向量长度的乘积再乘上它们夹角的余弦, 夹角大于90度时点积为负 \*/  
 double getDot(Vector a, Vector b) { return a.x \* b.x + a.y \* b.y; }  
 /\* 叉积: 叉积等于两向量组成的三角形有向面积的两倍, cross(v, w) = -cross(w, v) \*/  
 double getCross(Vector a, Vector b) { return a.x \* b.y - a.y \* b.x; }  
  
 double getLength(Vector a) { return sqrt(getDot(a, a)); }  
 double getPLength(Vector a) { return getDot(a, a); }  
 double getAngle(Vector u) { return atan2(u.y, u.x); }  
 double getAngle(Vector a, Vector b) { return acos(getDot(a, b) / getLength(a) / getLength(b)); }  
 Vector rotate(Vector a, double rad) { return Vector(a.x\*cos(rad)-a.y\*sin(rad), a.x\*sin(rad)+a.y\*cos(rad)); }  
 /\* 单位法线 \*/  
 Vector getNormal(Vector a) { double l = getLength(a); return Vector(-a.y/l, a.x/l); }  
};  
  
namespace ComplexVector {  
 typedef complex<double> Point;  
 typedef Point Vector;  
  
 double getDot(Vector a, Vector b) { return real(conj(a)\*b); }  
 double getCross(Vector a, Vector b) { return imag(conj(a)\*b); }  
 Vector rotate(Vector a, double rad) { return a\*exp(Point(0, rad)); }  
};  
  
namespace Linear {  
 using namespace Vectorial;  
  
 Line getLine(double x1, double y1, double x2, double y2) { return Line(y2-y1, x1-x2, y1\*x2-x1\*y2); }  
 Line getLine(double a, double b, Point u) { return Line(a, -b, u.y \* b - u.x \* a); }  
  
 bool getIntersection (Line p, Line q, Point& o) {  
 if (fabs(p.a \* q.b - q.a \* p.b) < eps) return false;  
 o.x = (q.c \* p.b - p.c \* q.b) / (p.a \* q.b - q.a \* p.b);  
 o.y = (q.c \* p.a - p.c \* q.a) / (p.b \* q.a - q.b \* p.a);  
 return true;  
 }  
  
 /\* 直线pv和直线qw的交点 \*/  
 bool getIntersection (Point p, Vector v, Point q, Vector w, Point& o) {  
 if (sgn(getCross(v, w)) == 0) return false;  
 Vector u = p - q;  
 double k = getCross(w, u) / getCross(v, w);  
 o = p + v \* k;  
 return true;  
 }  
  
 /\* 点p到直线ab的距离 \*/  
 double getDistanceToLine (Point p, Point a, Point b) { return fabs(getCross(b-a, p-a) / getLength(b-a)); }  
  
 double getDistanceToSegment (Point p, Point a, Point b) {  
 if (a == b) return getLength(p-a);  
 Vector v1 = b - a, v2 = p - a, v3 = p - b;  
 if (sgn(getDot(v1, v2)) < 0) return getLength(v2);  
 else if (sgn(getDot(v1, v3)) > 0) return getLength(v3);  
 else return fabs(getCross(v1, v2) / getLength(v1));  
 }  
  
 /\* 点p在直线ab上的投影 \*/  
 Point getPointToLine (Point p, Point a, Point b) { Vector v = b-a; return a+v\*(getDot(v, p-a) / getDot(v,v)); }  
  
 /\* 判断线段是否存在交点 \*/  
 bool haveIntersection (Point a1, Point a2, Point b1, Point b2) {  
 double c1=getCross(a2-a1, b1-a1), c2=getCross(a2-a1, b2-a1), c3=getCross(b2-b1, a1-b1), c4=getCross(b2-b1,a2-b1);  
 return sgn(c1)\*sgn(c2) < 0 && sgn(c3)\*sgn(c4) < 0;  
 }  
  
 /\* 判断点是否在线段上 \*/  
 bool onSegment (Point p, Point a, Point b) { return sgn(getCross(a-p, b-p)) == 0 && sgn(getDot(a-p, b-p)) < 0; }  
 bool onLeft(DirLine l, Point p) { return sgn(l.v ^ (p-l.p)) >= 0; }  
}  
  
namespace Triangular {  
 using namespace Vectorial;  
  
 double getAngle(double a, double b, double c) { return acos((a\*a+b\*b-c\*c) / (2\*a\*b)); }  
 double getArea(double a, double b, double c) { double s =(a+b+c)/2; return sqrt(s\*(s-a)\*(s-b)\*(s-c)); }  
 double getArea(double a, double h) { return a \* h / 2; }  
 double getArea(Point a, Point b, Point c) { return fabs(getCross(b - a, c - a)) / 2; }  
 double getDirArea(Point a, Point b, Point c) { return getCross(b - a, c - a) / 2; }  
};  
  
namespace Polygonal {  
 using namespace Vectorial;  
 using namespace Linear;  
  
 double getArea(Point\* p, int n) {  
 double ret = 0;  
 for (int i = 0; i < n - 1; i++)  
 ret += (p[i] - p[0]) ^ (p[i+1] - p[0]);  
 return fabs(ret / 2);  
 }  
  
 /\* 凸包 \*/  
 int getConvexHull (Point\* ps, int n, Point\* ch) {  
 /\* 可共线删去两个 =，需要先去除重点! \*/  
 sort(ps, ps + n);  
 int k = 0;  
 for (int i = 0; i < n; ++i) {  
 while (k > 1 && sgn(getCross(ch[k - 1] - ch[k - 2], ps[i] - ch[k - 1])) <= 0) k--;  
 ch[k++] = ps[i];  
 }  
 for (int i = n - 2, t = k; i >= 0; --i) {  
 while (k > t && sgn(getCross(ch[k - 1] - ch[k - 2], ps[i] - ch[k - 1])) <= 0) k--;  
 ch[k++] = ps[i];  
 }  
 if (n > 1) k--;  
 return k;  
 }  
  
 int isPointInPolygon(Point o, Point\* p, int n) {  
 int wn = 0;  
 for (int i = 0; i < n; i++) {  
 int j = (i + 1) % n;  
 if (onSegment(o, p[i], p[j]) || o == p[i]) return 0; // 边界上  
 int k = sgn(getCross(p[j] - p[i], o-p[i]));  
 int d1 = sgn(p[i].y - o.y);  
 int d2 = sgn(p[j].y - o.y);  
 if (k > 0 && d1 <= 0 && d2 > 0) wn++;  
 if (k < 0 && d2 <= 0 && d1 > 0) wn--;  
 }  
 return wn ? -1 : 1;  
 }  
  
 /\* 旋转卡壳 \*/  
 void rotatingCalipers(Point \*p, int n, vector<Pii>& sol) {  
 sol.clear();  
 int j = 1; p[n] = p[0];  
 for (int i = 0; i < n; i++) {  
 while (getCross(p[j+1]-p[i+1], p[i]-p[i+1]) > getCross(p[j]-p[i+1], p[i]-p[i+1]))  
 j = (j + 1) % n;  
 sol.push\_back(make\_pair(i, j));  
 sol.push\_back(make\_pair(i + 1, j + 1));  
 }  
 }  
  
 void rotatingCalipersGetRectangle(Point \*p, int n, double& area, double& perimeter) {  
 p[n] = p[0];  
 int l = 1, r = 1, j = 1;  
 area = perimeter = 1e20;  
  
 for (int i = 0; i < n; i++) {  
 Vector v = (p[i+1]-p[i]) / getLength(p[i+1]-p[i]);  
 while (sgn(getDot(v, p[r%n]-p[i]) - getDot(v, p[(r+1)%n]-p[i])) < 0) r++;  
 while (j < r || sgn(getCross(v, p[j%n]-p[i]) - getCross(v,p[(j+1)%n]-p[i])) < 0) j++;  
 while (l < j || sgn(getDot(v, p[l%n]-p[i]) - getDot(v, p[(l+1)%n]-p[i])) > 0) l++;  
 double w = getDot(v, p[r%n]-p[i])-getDot(v, p[l%n]-p[i]);  
 double h = getDistanceToLine(p[j%n], p[i], p[i+1]);  
 area = min(area, w \* h);  
 perimeter = min(perimeter, 2 \* w + 2 \* h);  
 }  
 }  
  
 /\* 计算半平面相交可以用增量法，o(n^2)，初始设置4条无穷大的半平面 \*/  
 /\* 用有向直线A->B切割多边形u，返回左侧。可能退化成单点或线段 \*/  
 Polygon cutPolygon(Polygon u, Point a, Point b) {  
 Polygon ret;  
 int n = u.size();  
 for (int i = 0; i < n; i++) {  
 Point c = u[i], d = u[(i+1)%n];  
 if (sgn((b-a)^(c-a)) >= 0) ret.push\_back(c);  
 if (sgn((b-a)^(c-d)) != 0) {  
 Point t;  
 getIntersection(a, b-a, c, d-c, t);  
 if (onSegment(t, c, d))  
 ret.push\_back(t);  
 }  
 }  
 return ret;  
 }  
  
 /\* 半平面相交 \*/  
 int halfPlaneIntersection(DirLine\* li, int n, Point\* poly) {  
 sort(li, li + n);  
  
 int first, last;  
 Point\* p = new Point[n];  
 DirLine\* q = new DirLine[n];  
 q[first=last=0] = li[0];  
  
 for (int i = 1; i < n; i++) {  
 while (first < last && !onLeft(li[i], p[last-1])) last--;  
 while (first < last && !onLeft(li[i], p[first])) first++;  
 q[++last] = li[i];  
  
 if (sgn(q[last].v ^ q[last-1].v) == 0) {  
 last--;  
 if (onLeft(q[last], li[i].p)) q[last] = li[i];  
 }  
  
 if (first < last)  
 getIntersection(q[last-1].p, q[last-1].v, q[last].p, q[last].v, p[last-1]);  
 }  
  
 while (first < last && !onLeft(q[first], p[last-1])) last--;  
 if (last - first <= 1) { delete [] p; delete [] q; return 0; }  
 getIntersection(q[last].p, q[last].v, q[first].p, q[first].v, p[last]);  
  
 int m = 0;  
 for (int i = first; i <= last; i++) poly[m++] = p[i];  
 delete [] p; delete [] q;  
 return m;  
 }  
  
 /\* 去除多边形共线点 \*/  
 Polygon simplify(const Polygon& poly) {  
 Polygon ret;  
 int n = poly.size();  
 for (int i = 0; i < n; i++) {  
 Point a = poly[i];  
 Point b = poly[(i+1)%n];  
 Point c = poly[(i+2)%n];  
 if (sgn((b-a)^(c-b)) != 0 && (ret.size() == 0 || b != ret[ret.size()-1]))  
 ret.push\_back(b);  
 }  
 return ret;  
 }  
};  
  
namespace Circular {  
 using namespace Linear;  
 using namespace Vectorial;  
 using namespace Triangular;  
  
 /\* 直线和圆的交点 \*/  
 int getLineCircleIntersection (Point p, Point q, Circle O, double& t1, double& t2, vector<Point>& sol) {  
 Vector v = q - p;  
 /\* 使用前需清空sol \*/  
 //sol.clear();  
 double a = v.x, b = p.x - O.o.x, c = v.y, d = p.y - O.o.y;  
 double e = a\*a+c\*c, f = 2\*(a\*b+c\*d), g = b\*b+d\*d-O.r\*O.r;  
 double delta = f\*f - 4\*e\*g;  
 if (sgn(delta) < 0) return 0;  
 if (sgn(delta) == 0) {  
 t1 = t2 = -f / (2 \* e);  
 sol.push\_back(p + v \* t1);  
 return 1;  
 }  
  
 t1 = (-f - sqrt(delta)) / (2 \* e); sol.push\_back(p + v \* t1);  
 t2 = (-f + sqrt(delta)) / (2 \* e); sol.push\_back(p + v \* t2);  
 return 2;  
 }  
  
 /\* 圆和圆的交点 \*/  
 int getCircleCircleIntersection (Circle o1, Circle o2, vector<Point>& sol) {  
 double d = getLength(o1.o - o2.o);  
  
 if (sgn(d) == 0) {  
 if (sgn(o1.r - o2.r) == 0) return -1;  
 return 0;  
 }  
  
 if (sgn(o1.r + o2.r - d) < 0) return 0;  
 if (sgn(fabs(o1.r-o2.r) - d) > 0) return 0;  
  
 double a = getAngle(o2.o - o1.o);  
 double da = acos((o1.r\*o1.r + d\*d - o2.r\*o2.r) / (2\*o1.r\*d));  
  
 Point p1 = o1.point(a-da), p2 = o1.point(a+da);  
  
 sol.push\_back(p1);  
 if (p1 == p2) return 1;  
 sol.push\_back(p2);  
 return 2;  
 }  
  
 /\* 过定点作圆的切线 \*/  
 int getTangents (Point p, Circle o, Vector\* v) {  
 Vector u = o.o - p;  
 double d = getLength(u);  
 if (d < o.r) return 0;  
 else if (sgn(d - o.r) == 0) {  
 v[0] = rotate(u, pi / 2);  
 return 1;  
 } else {  
 double ang = asin(o.r / d);  
 v[0] = rotate(u, -ang);  
 v[1] = rotate(u, ang);  
 return 2;  
 }  
 }  
  
 /\* a[i] 和 b[i] 分别是第i条切线在O1和O2上的切点 \*/  
 /\* have some problems \*/  
 int getTangents(Circle o1, Circle o2, Point\* a, Point\* b) {  
 int cnt = 0;  
 if (sgn(o1.r - o2.r) < 0) { swap(o1, o2); swap(a, b); }  
 double d2 = getPLength(o1.o - o2.o);  
 double rdif = o1.r - o2.r, rsum = o1.r + o2.r;  
 if (sgn(d2 - rdif \* rdif) < 0) return 0;  
 if (sgn(d2) == 0 && sgn(o1.r - o2.r) == 0) return -1;  
  
 double base = getAngle(o2.o - o1.o);  
 if (sgn(d2 - rdif \* rdif) == 0) {  
 a[cnt] = o1.point(base); b[cnt] = o2.point(base); cnt++;  
 return cnt;  
 }  
  
 double ang = acos( rdif / sqrt(d2) );  
 a[cnt] = o1.point(base+ang); b[cnt] = o2.point(base+ang); cnt++;  
 a[cnt] = o1.point(base-ang); b[cnt] = o2.point(base-ang); cnt++;  
  
 if (sgn(d2 - rsum \* rsum) == 0) {  
 a[cnt] = o1.point(base); b[cnt] = o2.point(base); cnt++;  
 } else if (sgn(d2 - rsum \* rsum) > 0) {  
 double ang = acos( rsum / sqrt(d2) );  
 a[cnt] = o1.point(base+ang); b[cnt] = o2.point(pi+base+ang); cnt++;  
 a[cnt] = o1.point(base-ang); b[cnt] = o2.point(pi+base-ang); cnt++;  
 }  
 return cnt;  
 }  
  
 /\* 三点确定外切圆 \*/  
 Circle CircumscribedCircle(Point p1, Point p2, Point p3) {  
 double Bx = p2.x - p1.x, By = p2.y - p1.y;  
 double Cx = p3.x - p1.x, Cy = p3.y - p1.y;  
 double D = 2 \* (Bx \* Cy - By \* Cx);  
 double cx = (Cy \* (Bx \* Bx + By \* By) - By \* (Cx \* Cx + Cy \* Cy)) / D + p1.x;  
 double cy = (Bx \* (Cx \* Cx + Cy \* Cy) - Cx \* (Bx \* Bx + By \* By)) / D + p1.y;  
 Point p = Point(cx, cy);  
 return Circle(p, getLength(p1 - p));  
 }  
  
 /\* 三点确定内切圆 \*/  
 Circle InscribedCircle(Point p1, Point p2, Point p3) {  
 double a = getLength(p2 - p3);  
 double b = getLength(p3 - p1);  
 double c = getLength(p1 - p2);  
 Point p = (p1 \* a + p2 \* b + p3 \* c) / (a + b + c);  
 return Circle(p, getDistanceToLine(p, p1, p2));  
 }  
  
 /\* 三角形一顶点为圆心 \*/  
 double getPublicAreaToTriangle(Circle O, Point a, Point b) {  
 if (sgn((a-O.o)^(b-O.o)) == 0) return 0;  
 int sig = 1;  
 double da = getLength(O.o-a), db = getLength(O.o-b);  
 if (sgn(da-db) > 0) {  
 swap(da, db); swap(a, b); sig = -1;  
 }  
  
 double t1, t2;  
 vector<Point> sol;  
 int n = getLineCircleIntersection(a, b, O, t1, t2, sol);  
  
 if (sgn(da-O.r) <= 0) {  
 if (sgn(db-O.r) <= 0) return getDirArea(O.o, a, b) \* sig;  
  
 int k = 0;  
 if (n == 2 && getPLength(sol[0]-b) > getPLength(sol[1]-b)) k = 1;  
  
 double ret = getArea(O.o, a, sol[k]) + O.getArea(getAngle(sol[k]-O.o, b-O.o));  
 double tmp = (a-O.o)^(b-O.o);  
 return ret \* sig \* sgn(tmp);  
 }  
  
 double d = getDistanceToSegment(O.o, a, b);  
 if (sgn(d-O.r) >= 0) {  
 double ret = O.getArea(getAngle(a-O.o, b-O.o));  
 double tmp = (a-O.o)^(b-O.o);  
 return ret \* sig \* sgn(tmp);  
 }  
  
 double ret1 = O.getArea(getAngle(a-O.o, b-O.o));  
 double ret2 = O.getArea(getAngle(sol[0]-O.o, sol[1]-O.o)) - getArea(O.o, sol[0], sol[1]);  
 double ret = (ret1 - ret2), tmp = (a-O.o)^(b-O.o);  
 return ret \* sig \* sgn(tmp);  
 }  
  
 double getPublicAreaToPolygon (Circle O, Point\* p, int n) {  
 if (sgn(O.r) == 0) return 0;  
 double area = 0;  
 for (int i = 0; i < n; i++) {  
 int u = (i + 1) % n;  
 area += getPublicAreaToTriangle(O, p[i], p[u]);  
 }  
 return fabs(area);  
 }  
};