/\*\*\*\*\*\*\*\*\*\*\*三分法求函数极值\*\*\*\*\*\*\*\*\*\*\*\*\*/

void solve()

{

double L, R, m, mm, mv, mmv;

while (L + eps < R)

{

m = (L + R) / 2;

mm = (m + R) / 2;

mv = calc(m);

mmv = calc(mm);

if (mv <= mmv) R = mm; //三分法求最大值时改为mv>=mmv

else L = m;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*基础\*\*\*\*\*\*\*\*\*\*\*/

int dcmp(double x) {

if(fabs(x) < eps) return 0; else return x < 0 ? -1 : 1;

}

struct Point {

double x, y;

Point(double x=0, double y=0):x(x),y(y) { }

};

Point operator + (Point A, Point B) { return Point(A.x+B.x, A.y+B.y); }

Point operator - (Point A, Point B) { return Point(A.x-B.x, A.y-B.y); }

Point operator \* (Point A, double p) { return Point(A.x\*p, A.y\*p); }

Point operator / (Point A, double p) { return Point(A.x/p, A.y/p); }

bool operator < (const Point& a, const Point& b) {

return a.x < b.x || (a.x == b.x && a.y < b.y);

}

bool operator == (const Point& a, const Point &b) {

return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y) == 0;

}

double Dot(Point A, Point B) { return A.x\*B.x + A.y\*B.y; }

double Length(Point A) { return sqrt(Dot(A, A)); }

double Angle(Point A, Point B) { return acos(Dot(A, B) / Length(A) / Length(B)); }

double angle(Point v) { return atan2(v.y, v.x); }

double Cross(Point A, Point B) { return A.x\*B.y - A.y\*B.x; }

/\*

向量叉积

若 P × Q > 0 , 则P在Q的顺时针方向。

若 P × Q < 0 , 则P在Q的逆时针方向。

若 P × Q = 0 , 则P与Q共线，但可能同向也可能反向。

\*/

Point vecunit(Point x){ return x / Length(x);} //单位向量

Point Normal(Point x) { return Point(-x.y, x.x) / Length(x);} //垂直法向量

Point Rotate(Point A, double rad) {

return Point(A.x\*cos(rad)-A.y\*sin(rad), A.x\*sin(rad)+A.y\*cos(rad));

}

double Area2(const Point A, const Point B, const Point C) { return Length(Cross(B-A, C-A)); }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*直线与线段\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//求直线p+tv和q+tw的交点 Cross(v, w) == 0无交点

Point GetLineIntersection(Point p, Point v, Point q, Point w)

{

Point u = p-q;

double t = Cross(w, u) / Cross(v, w);

return p + v\*t;

}

//点p在直线ab的投影

Point GetLineProjection(Point P, Point A, Point B) {

Point v = B-A;

return A+v\*(Dot(v, P-A) / Dot(v, v));

}

//点到直线距离

double DistanceToLine(Point P, Point A, Point B) {

Point v1 = B - A, v2 = P - A;

return fabs(Cross(v1, v2)) / Length(v1); // 如果不取绝对值，得到的是有向距离

}

//点在p线段上

bool OnSegment(Point p, Point a1, Point a2) {

return dcmp(Cross(a1-p, a2-p)) == 0 && dcmp(Dot(a1-p, a2-p)) < 0; //线段包含短点时改成<=

}

// 过两点p1, p2的直线一般方程ax+by+c=0

// (x2-x1)(y-y1) = (y2-y1)(x-x1)

void getLineGeneralEquation(const Point& p1, const Point& p2, double& a, double& b, double &c) {

a = p2.y-p1.y;

b = p1.x-p2.x;

c = -a\*p1.x - b\*p1.y;

}

//点到线段距离

double DistanceToSegment(Point p, Point a, Point b)

{

if(a == b) return Length(p-a);

Point v1 = b-a, v2 = p-a, v3 = p-b;

if(dcmp(Dot(v1, v2)) < 0) return Length(v2);

else if(dcmp(Dot(v1, v3)) > 0) return Length(v3);

else return fabs(Cross(v1, v2)) / Length(v1);

}

//两线段最近距离

double dis\_pair\_seg(Point p1, Point p2, Point p3, Point p4)

{

return min(min(DistanceToSegment(p1, p3, p4), DistanceToSegment(p2, p3, p4)),

min(DistanceToSegment(p3, p1, p2), DistanceToSegment(p4, p1, p2)));

}

//线段相交判定

bool SegmentItersection(Point a1, Point a2, Point b1, Point b2)

{

double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1),

c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);

return dcmp(c1)\*dcmp(c2) < 0 && dcmp(c3)\*dcmp(c4) < 0;

}

// 有向直线。它的左边就是对应的半平面

struct Line {

Point p; // 直线上任意一点

Point v; // 方向向量

double ang; // 极角，即从x正半轴旋转到向量v所需要的角（弧度）

Line() {}

Line(Point P, Point v):p(P),v(v){ ang = atan2(v.y, v.x); }

Point point(double a){

return p+(v\*a);

}

bool operator < (const Line& L) const {

return ang < L.ang;

}

Point point(double a){

Return p+(v\*a);

}

};

//两直线交点

Point GetLineIntersection(Line a, Line b) {

return GetLineIntersection(a.p, a.v, b.p, b.v);

}

// 点p在有向直线L的左边（线上不算）

bool OnLeft(const Line& L, const Point& p) {

return Cross(L.v, p-L.P) >= 0;

}

// 二直线交点，假定交点惟一存在

Point GetLineIntersection(const Line& a, const Line& b) {

Point u = a.P-b.P;

double t = Cross(b.v, u) / Cross(a.v, b.v);

return a.P+a.v\*t;

}

// 半平面交主过程

Point<Point> HalfplaneIntersection(Point<Line> L) {

int n = L.size();

sort(L.begin(), L.end()); // 按极角排序

int first, last; // 双端队列的第一个元素和最后一个元素的下标

Point<Point> p(n); // p[i]为q[i]和q[i+1]的交点

Point<Line> q(n); // 双端队列

Point<Point> ans; // 结果

q[first=last=0] = L[0]; // 双端队列初始化为只有一个半平面L[0]

for(int i = 1; i < n; i++) {

while(first < last && !OnLeft(L[i], p[last-1])) last--;

while(first < last && !OnLeft(L[i], p[first])) first++;

q[++last] = L[i];

if(fabs(Cross(q[last].v, q[last-1].v)) < eps) { // 两向量平行且同向，取内侧的一个

last--;

if(OnLeft(q[last], L[i].P)) q[last] = L[i];

}

if(first < last) p[last-1] = GetLineIntersection(q[last-1], q[last]);

}

while(first < last && !OnLeft(q[first], p[last-1])) last--; // 删除无用平面

if(last - first <= 1) return ans; // 空集

p[last] = GetLineIntersection(q[last], q[first]); // 计算首尾两个半平面的交点

// 从deque复制到输出中

for(int i = first; i <= last; i++) ans.push\_back(p[i]);

return ans;

}

/\*\*\*\*\*\*\*\*\*\*\*多边形\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//多边形有向面积

double PolygonArea(Point<Point> p) {

int n = p.size();

double area = 0;

for(int i = 1; i < n-1; i++)

area += Cross(p[i]-p[0], p[i+1]-p[0]);

return area/2;

}

//多边形重心 点集逆时针给出

Point PolyGravity(Point \*p, int n) {

Point tmp, g = Point(0, 0);

double sumArea = 0, area;

for (int i=2; i<n; ++i) {

area = Cross(p[i-1]-p[0], p[i]-p[0]);

sumArea += area;

tmp.x = p[0].x + p[i-1].x + p[i].x;

tmp.y = p[0].y + p[i-1].y + p[i].y;

g.x += tmp.x \* area;

g.y += tmp.y \* area;

}

g.x /= (sumArea \* 3.0); g.y /= (sumArea \* 3.0);

return g;

}

// 点集凸包

// 如果不希望在凸包的边上有输入点，把两个 <= 改成 <

// 注意：输入点集会被修改

Point<Point> ConvexHull(Point<Point>& p) {

// 预处理，删除重复点

sort(p.begin(), p.end());

p.erase(unique(p.begin(), p.end()), p.end());

int n = p.size();

int m = 0;

Point<Point> ch(n+1);

for(int i = 0; i < n; i++) {

while(m > 1 && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;

ch[m++] = p[i];

}

int k = m;

for(int i = n-2; i >= 0; i--) {

while(m > k && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;

ch[m++] = p[i];

}

if(n > 1) m--;

ch.resize(m);

return ch;

}

//判断点是否在多边形内

int isPointInPolygon(Point p, Polygon poly)

{

int wn = 0;

int n = poly.size();

for (int i = 0; i < n; i++)

{

if (OnSegment(p, poly[i], poly[(i + 1) % n])) return -1; //边界

int k = dcmp(Cross(poly[(i + 1) % n] - poly[i], p - poly[i]));

int d1 = dcmp(poly[i].y - p.y);

int d2 = dcmp(poly[(i + 1) % n].y - p.y);

if (k > 0 && d1 <= 0 && d2 > 0) wn++;

if (k < 0 && d2 <= 0 && d1 > 0) wn--;

}

if (wn != 0) return 1; //内部

return 0; //外部

}

// 凸包直径返回 点集直径的平方

int diameter2(Point<Point>& points) {

Point<Point> p = ConvexHull(points);

int n = p.size();

if(n == 1) return 0;

if(n == 2) return Dist2(p[0], p[1]);

p.push\_back(p[0]); // 免得取模

int ans = 0;

for(int u = 0, v = 1; u < n; u++) {

// 一条直线贴住边p[u]-p[u+1]

for(;;) {

int diff = Cross(p[u+1]-p[u], p[v+1]-p[v]);

if(diff <= 0) {

ans = max(ans, Dist2(p[u], p[v])); // u和v是对踵点

if(diff == 0) ans = max(ans, Dist2(p[u], p[v+1])); // diff == 0时u和v+1也是对踵点

break;

}

v = (v + 1) % n;

}

}

return ans;

}

//两凸包最近距离

double RC\_Distance(Point \*ch1, Point \*ch2, int n, int m)

{

int q=0, p=0;

REP(i, n) if(ch1[i].y-ch1[p].y < -eps) p=i;

REP(i, m) if(ch2[i].y-ch2[q].y > eps) q=i;

ch1[n]=ch1[0]; ch2[m]=ch2[0];

double tmp, ans=1e100;

REP(i, n)

{

while((tmp = Cross(ch1[p+1]-ch1[p], ch2[q+1]-ch1[p]) - Cross(ch1[p+1]-ch1[p], ch2[q]- ch1[p])) > eps)

q=(q+1)%m;

if(tmp < -eps) ans = min(ans,DistanceToSegment(ch2[q],ch1[p],ch1[p+1]));

else ans = min(ans,dis\_pair\_seg(ch1[p],ch1[p+1],ch2[q],ch2[q+1]));

p=(p+1)%n;

}

return ans;

}

//凸包最大内接三角形

double RC\_Triangle(Point\* res,int n)// 凸包最大内接三角形

{

if(n<3) return 0;

double ans=0, tmp;

res[n] = res[0];

int j, k;

REP(i, n)

{

j = (i+1)%n;

k = (j+1)%n;

while((j != k) && (k != i))

{

while(Cross(res[j] - res[i], res[k+1] - res[i]) > Cross(res[j] - res[i], res[k] - res[i])) k= (k+1)%n;

tmp = Cross(res[j] - res[i], res[k] - res[i]);if(tmp > ans) ans = tmp;

j = (j+1)%n;

}

}

return ans;

}

//模拟退火求费马点 保存在ptres中

double fermat\_point(Point \*pt, int n, Point& ptres)

{

Point u, v;

double step = 0.0, curlen, explen, minlen;

int i, j, k, idx;

bool flag;

u.x = u.y = v.x = v.y = 0.0;

REP(i, n)

{

step += fabs(pt[i].x) + fabs(pt[i].y);

u.x += pt[i].x;

u.y += pt[i].y;

}

u.x /= n;

u.y /= n;

flag = 0;

while(step > eps)

{

for(k = 0; k < 10; step /= 2, ++k)

for(i = -1; i <= 1; ++i)

for(j = -1; j <= 1; ++j)

{

v.x = u.x + step\*i;

v.y = u.y + step\*j;

curlen = explen = 0.0;

REP(idx, n)

{

curlen += dist(u, pt[idx]);

explen += dist(v, pt[idx]);

}

if(curlen > explen)

{

u = v;

minlen = explen;

flag = 1;

}

}

}

ptres = u;

return flag ? minlen : curlen;

}

//最近点对

bool cmpxy(const Point& a, const Point& b)

{

if(a.x != b.x)

return a.x < b.x;

return a.y < b.y;

}

bool cmpy(const int& a, const int& b)

{

return point[a].y < point[b].y;

}

double Closest\_Pair(int left, int right)

{

double d = INF;

if(left==right)

return d;

if(left + 1 == right)

return dis(left, right);

int mid = (left+right)>>1;

double d1 = Closest\_Pair(left,mid);

double d2 = Closest\_Pair(mid+1,right);

d = min(d1,d2);

int i,j,k=0;

//分离出宽度为d的区间

for(i = left; i <= right; i++)

{

if(fabs(point[mid].x-point[i].x) <= d)

tmpt[k++] = i;

}

sort(tmpt,tmpt+k,cmpy);

//线性扫描

for(i = 0; i < k; i++)

{

for(j = i+1; j < k && point[tmpt[j]].y-point[tmpt[i]].y<d; j++)

{

double d3 = dis(tmpt[i],tmpt[j]);

if(d > d3)

d = d3;

}

}

return d;

}

/\*\*\*\*\*\*\*\*\*\*\*\*圆\*\*\*\*\*\*\*\*\*\*\*\*/

struct Circle

{

Point c;

double r;

Circle(){}

Circle(Point c, double r):c(c), r(r){}

Point point(double a) //根据圆心角求点坐标

{

return Point(c.x+cos(a)\*r, c.y+sin(a)\*r);

}

};

//求a点到b点(逆时针)在的圆上的圆弧长度

double D(Point a,Point b,int id)

{

double ang1,ang2;

Point v1,v2;

v1=a-Point(C[id].c.x,C[id].c.y);

v2=b-Point(C[id].c.x,C[id].c.y);

ang1=atan2(v1.y,v1.x);

ang2=atan2(v2.y,v2.x);

if(ang2<ang1) ang2+=2\*pi;

return C[id].r\*(ang2-ang1);

}

//直线与圆交点 返回个数

int getLineCircleIntersection(Line L, Circle C, double& t1, double& t2, Point<Point>& sol){

double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;

double e = a\*a + c\*c, f = 2\*(a\*b + c\*d), g = b\*b + d\*d - C.r\*C.r;

double delta = f\*f - 4\*e\*g; // 判别式

if(dcmp(delta) < 0) return 0; // 相离

if(dcmp(delta) == 0) { // 相切

t1 = t2 = -f / (2 \* e); sol.push\_back(L.point(t1));

return 1;

}

// 相交

t1 = (-f - sqrt(delta)) / (2 \* e); sol.push\_back(L.point(t1));

t2 = (-f + sqrt(delta)) / (2 \* e); sol.push\_back(L.point(t2));

return 2;

}

//两圆交点 返回个数

int getCircleCircleIntersection(Circle C1, Circle C2, Point<Point>& sol) {

double d = Length(C1.c - C2.c);

if(dcmp(d) == 0) {

if(dcmp(C1.r - C2.r) == 0) return -1; // 重合，无穷多交点

return 0;

}

if(dcmp(C1.r + C2.r - d) < 0) return 0;

if(dcmp(fabs(C1.r-C2.r) - d) > 0) return 0;

double a = angle(C2.c - C1.c);

double da = acos((C1.r\*C1.r + d\*d - C2.r\*C2.r) / (2\*C1.r\*d));

Point p1 = C1.point(a-da), p2 = C1.point(a+da);

sol.push\_back(p1);

if(p1 == p2) return 1;

sol.push\_back(p2);

return 2;

}

//P到圆的切线

//v[i]是第i条切线的向量, 返回切线数

int getTangents(Point p, Circle C, Point\* v)

{

Point u = C.c - p;

double dist = Length(u);

if (dist < C.r) return 0;

else if (dcmp(dist - C.r) == 0)

{

//P在圆上,只有一条切线

v[0] = Rotate(u, PI / 2);

return 1;

}

else

{

double ang = asin(C.r / dist);

v[0] = Rotate(u, -ang);

v[1] = Rotate(u, +ang);

return 2;

}

}

//两圆的公切线, -1表示无穷条切线

int getTangents(Circle A, Circle B, Point\* a, Point\* b)

{

int cnt = 0;

if (A.r < B.r) swap(A, B), swap(a, b);

///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int 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);

int rdiff = A.r - B.r;

int rsum = A.r + B.r;

if (d2 < rdiff \* rdiff) return 0; //内含

///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);

if (d2 == 0 && A.r == B.r) return -1; //无线多条切线

if (d2 == rdiff \* rdiff) //内切, 1条切线

{

///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

a[cnt] = A.point(base); b[cnt] = B.point(base); cnt++;

return 1;

}

//有外公切线

double ang = acos((A.r - B.r) / sqrt(d2));

a[cnt] = A.point(base + ang); b[cnt] = B.point(base + ang); cnt++;

a[cnt] = A.point(base - ang); b[cnt] = B.point(base - ang); cnt++;

if (d2 == rsum \* rsum) //一条内公切线

{

a[cnt] = A.point(base); b[cnt] = B.point(PI + base); cnt++;

}

else if (d2 > rsum \* rsum) //两条内公切线

{

double ang = acos((A.r + B.r) / sqrt(d2));

a[cnt] = A.point(base + ang); b[cnt] = B.point(PI + base + ang); cnt++;

a[cnt] = A.point(base - ang); b[cnt] = B.point(PI + base - ang); cnt++;

}

return cnt;

}

//三角形外接圆

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, Length(p1-p));

}

//三角形内切圆

Circle InscribedCircle(Point p1, Point p2, Point p3) {

double a = Length(p2-p3);

double b = Length(p3-p1);

double c = Length(p1-p2);

Point p = (p1\*a+p2\*b+p3\*c)/(a+b+c);

return Circle(p, DistanceToLine(p, p1, p2));

}

// 过点p到圆C的切线。v[i]是第i条切线的向量。返回切线条数

int getTangents(Point p, Circle C, Point\* v) {

Point u = C.c - p;

double dist = Length(u);

if(dist < C.r) return 0;

else if(dcmp(dist - C.r) == 0) { // p在圆上，只有一条切线

v[0] = Rotate(u, PI/2);

return 1;

} else {

double ang = asin(C.r / dist);

v[0] = Rotate(u, -ang);

v[1] = Rotate(u, +ang);

return 2;

}

}

//所有经过点p 半径为r 且与直线L相切的圆心

Point<Point> CircleThroughPointTangentToLineGivenRadius(Point p, Line L, double r) {

Point<Point> ans;

double t1, t2;

getLineCircleIntersection(L.move(-r), Circle(p, r), t1, t2, ans);

getLineCircleIntersection(L.move(r), Circle(p, r), t1, t2, ans);

return ans;

}

//半径为r 与a b两直线相切的圆心

Point<Point> CircleTangentToLinesGivenRadius(Line a, Line b, double r) {

Point<Point> ans;

Line L1 = a.move(-r), L2 = a.move(r);

Line L3 = b.move(-r), L4 = b.move(r);

ans.push\_back(GetLineIntersection(L1, L3));

ans.push\_back(GetLineIntersection(L1, L4));

ans.push\_back(GetLineIntersection(L2, L3));

ans.push\_back(GetLineIntersection(L2, L4));

return ans;

}

//与两圆相切 半径为r的所有圆心

Point<Point> CircleTangentToTwoDisjointCirclesWithRadius(Circle c1, Circle c2, double r) {

Point<Point> ans;

Point v = c2.c - c1.c;

double dist = Length(v);

int d = dcmp(dist - c1.r -c2.r - r\*2);

if(d > 0) return ans;

getCircleCircleIntersection(Circle(c1.c, c1.r+r), Circle(c2.c, c2.r+r), ans);

return ans;

}

//多边形与圆相交面积

Point GetIntersection(Line a, Line b) //线段交点

{

Point u = a.p-b.p;

double t = Cross(b.v, u) / Cross(a.v, b.v);

return a.p + a.v\*t;

}

bool InCircle(Point x, Circle c) { return dcmp(c.r - Length(c.c - x)) >= 0;}

bool OnCircle(Point x, Circle c) { return dcmp(c.r - Length(c.c - x)) == 0;}

//线段与圆的交点

int getSegCircleIntersection(Line L, Circle C, Point\* sol)

{

Point nor = normal(L.v);

Line pl = Line(C.c, nor);

Point ip = GetIntersection(pl, L);

double dis = Length(ip - C.c);

if (dcmp(dis - C.r) > 0) return 0;

Point dxy = vecunit(L.v) \* sqrt(sqr(C.r) - sqr(dis));

int ret = 0;

sol[ret] = ip + dxy;

if (OnSegment(sol[ret], L.p, L.point(1))) ret++;

sol[ret] = ip - dxy;

if (OnSegment(sol[ret], L.p, L.point(1))) ret++;

return ret;

}

double SegCircleArea(Circle C, Point a, Point b) //线段切割圆

{

double a1 = angle(a - C.c);

double a2 = angle(b - C.c);

double da = fabs(a1 - a2);

if (da > PI) da = PI \* 2.0 - da;

return dcmp(Cross(b - C.c, a - C.c)) \* da \* sqr(C.r) / 2.0;

}

double PolyCircleArea(Circle C,Point \*p,int n){

double ret=0;

Point sol[2];

p[n]=p[0];

for(int i=0;i<n;i++){

double t1,t2;

int cnt=getSegCircleIntersection(Line(p[i],p[i+1]-p[i]),C,sol); //判断线段与圆有几个交点，

if(cnt==0){ //0个交点，判断线段在多边形内部还是外部。

if(!InCircle(p[i],C)||!InCircle(p[i+1],C))ret+=SegCircleArea(C,p[i],p[i+1]); //外部直接计算圆弧面积

else ret+=Cross(p[i+1]-C.c,p[i]-C.c)/2; //内部计算三角形面积。

}

if(cnt==1){

if(InCircle(p[i],C)&&(!InCircle(p[i+1],C)||OnCircle(p[i+1],C)))ret+=Cross(sol[0]-C.c,p[i]-C.c)/2,ret+=SegCircleArea(C,sol[0],p[i+1]);//,cout<<"jj-1"<<endl;

else ret+=SegCircleArea(C,p[i],sol[0]),ret+=Cross(p[i+1]-C.c,sol[0]-C.c)/2;//,cout<<"jj-2"<<endl;

}

if(cnt==2){ //两个交点

if((p[i]<p[i+1])^(sol[0]<sol[1]))swap(sol[0],sol[1]);

ret+=SegCircleArea(C,p[i],sol[0]);

ret+=Cross(sol[1]-C.c,sol[0]-C.c)/2;

ret+=SegCircleArea(C,sol[1],p[i+1]);

}

// cout<<ret<<endl;

}

return fabs(ret);

}

/\*\*\*\*\*\*\*\*\*其他模板\*\*\*\*\*\*\*\*\*/

//以下模板来自网上，都未使用过

//pick定理

LL x\_mult(cpoint a,cpoint b,cpoint p)

{

return (a.x-p.x)\*(b.y-p.y)-(a.y-p.y)\*(b.x-p.x);

}

LL pick()

{

LL s =0, e = 0;

for(int i=0;i<n;i++)

{

s += x\_mult(re[i],re[i+1],re[0]);

e += gcd(abs(re[i].y-re[i+1].y),abs(re[i].x-re[i+1].x));

}e/=2;s/=2;

return Abs(s)+1-e;

}

//快速判断点是否在凸包内

struct POINT{

double x,y;

POINT(double \_x = 0, double \_y = 0):x(\_x),y(\_y){};

void show(){

cout<<x<<" "<<y<<endl;

}

};

POINT p[MAXN],wp[MAXN];

double multiply(POINT sp,POINT ep,POINT op){ //叉积

return (sp.x-op.x) \* (ep.y-op.y) - (ep.x-op.x) \* (sp.y-op.y);

}

bool onseg(POINT a,POINT s,POINT e){ // 判断点是否在线段上

if(multiply(a,s,e) == 0 && a.x <= max(s.x,e.x) && a.x >= min(s.x,e.x)

&& a.y <= max(s.y,e.y) && a.y >= min(s.y,e.y))

return true;

return false;

}

bool inside(POINT pp,POINT sp,POINT ep,POINT op){ //判断点pp是否在三角形中(极角序)

if(onseg(pp,sp,ep) || onseg(pp,sp,op) || onseg(pp,ep,op)) //如果在三角形上

return true;

if(multiply(sp,ep,pp) > 0 && multiply(ep,op,pp) > 0

&& multiply(sp,op,pp) < 0) //如果在三角形内

return true;

return false;

}

bool bsearch(POINT a,int len)

{ //二分所构造的三角形

int l = 1,r = len,m;

while(l < r){

m = (l + r) / 2;

if(inside(a,p[0],p[m],p[m+1]) == true) return true;

if(multiply(p[0],p[m],a) >= 0 && multiply(p[0],p[m+1],a) <= 0

&& multiply(p[m],p[m+1],a) < 0) return false;

if(multiply(p[0],p[m],a) > 0 && multiply(p[0],p[m+1],a) > 0)

l = m + 1;

else r = m;

}

return false;

}

int main()

{

int n,m,k,tmp = 0,cnt = 0;

scanf("%d%d%d",&n,&m,&k);

for(int i = 0 ; i < n ; i++)

scanf("%lf%lf",&p[i].x,&p[i].y);

p[n] = p[0];

for(int i = 0 ; i < m ; i++){

scanf("%lf%lf",&wp[i].x,&wp[i].y);

if(bsearch(wp[i],n-1) == true) cnt++;

}

if(cnt >= k) printf("YES\n");

else printf("NO\n");

return 0;

}

//圆的面积并

//圆的面积并

//

#include<cstdio>

#include<cstring>

#include<cmath>

#include<cstdlib>

#include<algorithm>

#define sqr(x) ((x)\*(x))

using namespace std;

const int N = 1010;

const double eps = 1e-8;

const double pi = acos(-1.0);

double area[N];

int n;

int dcmp(double x) {

if (x < -eps) return -1; else return x > eps;

}

struct cp {

double x, y, r, angle;

int d;

cp(){}

cp(double xx, double yy, double ang = 0, int t = 0) {

x = xx; y = yy; angle = ang; d = t;

}

void get() {

scanf("%lf%lf%lf", &x, &y, &r);

d = 1;

}

}cir[N], tp[N \* 2];

double dis(cp a, cp b) {

return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));

}

double cross(cp p0, cp p1, cp p2) {

return (p1.x - p0.x) \* (p2.y - p0.y) - (p1.y - p0.y) \* (p2.x - p0.x);

}

int CirCrossCir(cp p1, double r1, cp p2, double r2, cp &cp1, cp &cp2) {

double mx = p2.x - p1.x, sx = p2.x + p1.x, mx2 = mx \* mx;

double my = p2.y - p1.y, sy = p2.y + p1.y, my2 = my \* my;

double sq = mx2 + my2, d = -(sq - sqr(r1 - r2)) \* (sq - sqr(r1 + r2));

if (d + eps < 0) return 0; if (d < eps) d = 0; else d = sqrt(d);

double x = mx \* ((r1 + r2) \* (r1 - r2) + mx \* sx) + sx \* my2;

double y = my \* ((r1 + r2) \* (r1 - r2) + my \* sy) + sy \* mx2;

double dx = mx \* d, dy = my \* d; sq \*= 2;

cp1.x = (x - dy) / sq; cp1.y = (y + dx) / sq;

cp2.x = (x + dy) / sq; cp2.y = (y - dx) / sq;

if (d > eps) return 2; else return 1;

}

bool circmp(const cp& u, const cp& v) {

return dcmp(u.r - v.r) < 0;

}

bool cmp(const cp& u, const cp& v) {

if (dcmp(u.angle - v.angle)) return u.angle < v.angle;

return u.d > v.d;

}

double calc(cp cir, cp cp1, cp cp2) {

double ans = (cp2.angle - cp1.angle) \* sqr(cir.r)

- cross(cir, cp1, cp2) + cross(cp(0, 0), cp1, cp2);

return ans / 2;

}

void CirUnion(cp cir[], int n) {

cp cp1, cp2;

sort(cir, cir + n, circmp);

for (int i = 0; i < n; ++i)

for (int j = i + 1; j < n; ++j)

if (dcmp(dis(cir[i], cir[j]) + cir[i].r - cir[j].r) <= 0)

cir[i].d++;

for (int i = 0; i < n; ++i) {

int tn = 0, cnt = 0;

for (int j = 0; j < n; ++j) {

if (i == j) continue;

if (CirCrossCir(cir[i], cir[i].r, cir[j], cir[j].r,

cp2, cp1) < 2) continue;

cp1.angle = atan2(cp1.y - cir[i].y, cp1.x - cir[i].x);

cp2.angle = atan2(cp2.y - cir[i].y, cp2.x - cir[i].x);

cp1.d = 1; tp[tn++] = cp1;

cp2.d = -1; tp[tn++] = cp2;

if (dcmp(cp1.angle - cp2.angle) > 0) cnt++;

}

tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, pi, -cnt);

tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, -pi, cnt);

sort(tp, tp + tn, cmp);

int p, s = cir[i].d + tp[0].d;

for (int j = 1; j < tn; ++j) {

p = s; s += tp[j].d;

area[p] += calc(cir[i], tp[j - 1], tp[j]);

}

}

}

void solve()

{

scanf("%d", &n);

for (int i = 0; i < n; ++i)

cir[i].get();

memset(area, 0, sizeof(area));

CirUnion(cir, n);

//去掉重复计算的

for (int i = 1; i <= n; ++i) {

area[i] -= area[i + 1];

}

//area[i]为重叠了i次的面积

//tot 为总面积

double tot = 0;

for(int i=1; i<=n; i++) tot += area[i];

printf("%f\n", tot);

}

//多边形面积并

#define PDI pair<double,int>

#define point pair<double,double>

#define mp make\_pair

#define pb push\_back

#define x first

#define y second

#define zero 1e-8

#define maxN 502

#define maxp 5

point operator +(point a,point b) { return mp(a.x+b.x,a.y+b.y); }

point operator -(point a,point b) { return mp(a.x-b.x,a.y-b.y); }

double operator \*(point a,point b) { return a.x\*b.y-b.x\*a.y; }

double operator ^(point a,point b) { return a.x\*b.x+a.y\*b.y; }

inline double cross(point o,point a,point b) { return (a-o)\*(b-o); }

inline int cmp(double x) { if (fabs(x)<zero) return 0; return x>0? 1:-1; }

class Polygon

{

private: int i; double s;

public: int n; point p[maxp];

point& operator[] (int idx) { return p[idx]; }

void input() { for (i=0;i<n;i++) scanf("%lf %lf",&p[i].x,&p[i].y); p[n]=p[0]; }

double Area() { for (s=0,i=0;i<n;i++) s+=p[i]\*p[i+1]; return s/2; }

};

PDI s[maxN\*maxp\*2];

Polygon P[maxN];

double S,ts;

int N;

inline double seg(point o,point a,point b)

{

if (cmp(b.x-a.x)==0) return (o.y-a.y)/(b.y-a.y);

return (o.x-a.x)/(b.x-a.x);

}

double PolygonUnion()

{

int M,c1,c2; double s1,s2,ret=0;

for (int i=0;i<N;i++)

for (int ii=0;ii<P[i].n;ii++)

{

M=0;

s[M++]=mp(0.00,0);

s[M++]=mp(1.00,0);

for (int j=0;j<N;j++) if (i!=j)

for (int jj=0;jj<P[j].n;jj++)

{

c1=cmp(cross(P[i][ii],P[i][ii+1],P[j][jj]));

c2=cmp(cross(P[i][ii],P[i][ii+1],P[j][jj+1]));

if (c1==0 && c2==0)

{

if (((P[i][ii+1]-P[i][ii])^(P[j][jj+1]-P[j][jj]))>0 && i>j)

{

s[M++]=mp(seg(P[j][jj],P[i][ii],P[i][ii+1]),1);

s[M++]=mp(seg(P[j][jj+1],P[i][ii],P[i][ii+1]),-1);

}

}

else

{

s1=cross(P[j][jj],P[j][jj+1],P[i][ii]);

s2=cross(P[j][jj],P[j][jj+1],P[i][ii+1]);

if (c1>=0 && c2<0) s[M++]=mp(s1/(s1-s2),1);

else if (c1<0 && c2>=0) s[M++]=mp(s1/(s1-s2),-1);

}

}

sort(s,s+M);

double pre=min(max(s[0].x,0.0),1.0),now;

double sum=0;

int cov=s[0].y;

for (int j=1;j<M;j++)

{

now=min(max(s[j].x,0.0),1.0);

if (!cov) sum+=now-pre;

cov+=s[j].y;

pre=now;

}

ret+=P[i][ii]\*P[i][ii+1]\*sum;

}

return ret/2;

}

int main()

{

scanf("%d\n",&N);

for (int i=0;i<N;i++)

{

P[i].n=4;

P[i].input();

ts=P[i].Area();

if (cmp(ts<0))

{

reverse(P[i].p,P[i].p+P[i].n);

P[i][P[i].n]=P[i][0];

ts=-ts;

}

S+=ts;

}

printf("%.9lf\n",S/PolygonUnion());

}

//二维平面一个n个节点的简单多边形，多边形内有一个灯泡，求照明面积。

const double eps = 1e-8;

const double pi = acos(-1.);

using namespace std;

int dblcmp( double x )

{

if( fabs(x) < eps ) return 0;

return x > 0 ? 1 : -1;

}

double nowAng;

struct point

{

double x, y, a;

point(){}

point( double \_x, double \_y ) : x(\_x), y(\_y)

{

a = atan2(y, x);

}

bool operator<( const point p ) const

{

if( dblcmp(a-p.a) == 0 )

return x\*x+y\*y < p.x\*p.x+p.y\*p.y;

return a < p.a;

}

} p[60000], O;

inline double dis( point a, point b )

{

double dx = a.x-b.x;

double dy = a.y-b.y;

return sqrt(dx\*dx+dy\*dy);

}

inline double cross( point k, point a, point b )

{

return (a.x-k.x)\*(b.y-k.y) - (a.y-k.y)\*(b.x-k.x);

}

point inter( point a1, point a2, double ang )

{

point b1 = point(0, 0), b2 = point(cos(ang), sin(ang));

double u = cross(a1, a2, b1), v = cross(a2, a1, b2);

return point((b1.x\*v+b2.x\*u)/(v+u), (b1.y\*v+b2.y\*u)/(v+u));

}

struct line

{

point a, b;

line(){};

line( point \_a, point \_b ) : a(\_a), b(\_b){};

bool operator<( const line p ) const

{

if( fabs(a.x-p.a.x) < eps && fabs(a.y-p.a.y) < eps )

return cross(a, b, p.b) < 0;

point d1 = inter(a, b, nowAng);

point d2 = inter(p.a, p.b, nowAng);

return d1.x\*d1.x+d1.y\*d1.y < d2.x\*d2.x+d2.y\*d2.y;

}

};

struct Event

{

double ang;

int id, st;

line L;

bool operator<( const Event p ) const

{

if( dblcmp(ang-p.ang) == 0 )

return st > p.st;

return ang < p.ang;

}

} E[200000];

int c;

void add( point a, point b, int k )

{

if( b < a )

swap(a, b);

E[c].ang = a.a, E[c].st = 1, E[c].L = line(a, b), E[c++].id = k;

E[c].ang = b.a, E[c].st = 0, E[c++].id = k;

}

multiset<line> S;

multiset<line>::iterator itArr[100000];

inline double cal( line L, double d1, double d2 )

{

point a = inter(L.a, L.b, d1);

point b = inter(L.a, L.b, d2);

return fabs(0.5\*cross(O, a, b));

}

int main()

{

int i, j, k, n;

double d, t, ans, pre;

while( scanf("%lf %lf", &O.x, &O.y) != EOF )

{

c = 0;

scanf("%d", &n);

for( i = 0; i < n; ++i )

{

scanf("%lf %lf", &p[i].x, &p[i].y);

p[i].x -= O.x, p[i].y -= O.y;

p[i].a = atan2(p[i].y, p[i].x);

}

O.x = O.y = 0;

p[n] = p[0];

for( i = k = 0; i < n; ++i )

{

d = fabs(p[i+1].a-p[i].a);

if( d < pi )

add(p[i], p[i+1], k++);

else

{

point tmp = inter(p[i], p[i+1], pi);

tmp.a = pi\*dblcmp(p[i].a);

add(p[i], tmp, k++);

tmp.a = pi\*dblcmp(p[i+1].a);

add(p[i+1], tmp, k++);

}

}

sort(E, E+c);

S.clear();

ans = 0; pre = -pi;

for( i = 0; i < c; ++i )

{

nowAng = E[i].ang;

if( E[i].st )

{

if( S.size() > 0 )

ans += cal(\*S.begin(), pre, E[i].ang);

itArr[E[i].id] = S.insert(E[i].L);

}

else

{

ans += cal(\*S.begin(), pre, E[i].ang);

S.erase(itArr[E[i].id]);

}

pre = E[i].ang;

}

printf("%.10lf\n", ans);

}

return 0;

}

//二维平面有n(0 < n <= 50000)条线段，要求判断n条线段是否存在交点，如果有，输出相交线段的编号。

#define MP make\_pair

#define PI pair

#define FI first

#define SE second

#define PB push\_back

#define SZ size()

const double eps = 1e-10;

const double pi = acos(-1.);

const int mod = 1000000007;

const int maxn = 50100;

const int INF = 99999999;

struct point

{

int x, y;

bool operator<( const point p ) const

{

if( x == p.x ) return y < p.y;

return x < p.x;

}

} L[maxn][2];

int tim;

struct eve

{

int x, id, st;

eve(){}

eve( int \_x, int \_id, int \_st ) : x(\_x), id(\_id), st(\_st) {}

bool operator<( const eve p ) const

{

if( p.x == x ) return st > p.st;

return x < p.x;

}

} E[maxn\*2];

void getLine( point x, point y, double& a, double& b, double& c )

{

a = y.y - x.y;

b = x.x - y.x;

c = y.x\*x.y - x.x\*y.y;

}

struct ele

{

int id;

double k, c;

ele( int \_id )

{

id = \_id;

if( L[id][0].x == L[id][1].x )

k = 0, c = L[id][0].y;

else

{

double A, B, C;

getLine(L[id][0], L[id][1], A, B, C);

k = -A/B, c = -C/B;

}

}

bool operator<( const ele p ) const

{

return tim\*k+c < tim\*p.k+p.c;

}

};

set<ele> S;

set<ele>::iterator itArr[maxn];

inline set<ele>::iterator preIt( set<ele>::iterator it )

{

return it == S.begin() ? S.end() : --it;

}

inline set<ele>::iterator nxtIt( set<ele>::iterator it )

{

return it == S.end() ? S.end() : ++it;

}

int cross( point& k, point& a, point& b )

{

return (a.x-k.x)\*(b.y-k.y) - (a.y-k.y)\*(b.x-k.x);

}

int dot( point& k, point& a, point& b )

{

return (a.x-k.x)\*(b.x-k.x) + (a.y-k.y)\*(b.y-k.y);

}

inline int sgn( int x )

{

if( x > 0 ) return 1;

if( x < 0 ) return -1;

return x;

}

bool inter( int a, int b )

{

int d1 = sgn(cross(L[a][0], L[a][1], L[b][0]));

int d2 = sgn(cross(L[a][0], L[a][1], L[b][1]));

int d3 = sgn(cross(L[b][0], L[b][1], L[a][0]));

int d4 = sgn(cross(L[b][0], L[b][1], L[a][1]));

if( (d1^d2)==-2 && (d3^d4)==-2 ) return 1;

if( d1 == 0 && dot(L[b][0], L[a][0], L[a][1]) <= 0 ) return 1;

if( d2 == 0 && dot(L[b][1], L[a][0], L[a][1]) <= 0 ) return 1;

if( d3 == 0 && dot(L[a][0], L[b][0], L[b][1]) <= 0 ) return 1;

if( d4 == 0 && dot(L[a][1], L[b][0], L[b][1]) <= 0 ) return 1;

return 0;

}

bool solve( int n )

{

sort(E, E+n);

S.clear();

for( int i = 0; i < n; ++i )

{

tim = E[i].x;

int id = E[i].id;

if( E[i].st == 1 )

{

ele t = ele(id);

set<ele>::iterator nxt = S.lower\_bound(t), pre = preIt(nxt);

if( nxt != S.end() && inter((\*nxt).id, id) )

{

printf("YES\n%d %d\n", (\*nxt).id, id);

return 1;

}

if( pre != S.end() && inter((\*pre).id, id) )

{

printf("YES\n%d %d\n", (\*pre).id, id);

return 1;

}

itArr[id] = S.insert(nxt, t);

}

else

{

set<ele>::iterator pre = preIt( itArr[id] ), nxt = nxtIt( itArr[id] );

if( pre != S.end() && nxt != S.end() && inter((\*pre).id, (\*nxt).id) )

{

printf("YES\n%d %d\n", (\*pre).id, (\*nxt).id);

return 1;

}

S.erase(itArr[id]);

}

}

return 0;

}

int main()

{

srand(4);

int T, cases = 1;

int i, j, k, e;

double A, B, C;

int N, M;

scanf("%d", &N);

for( i = 1; i <= N; ++i )

{

scanf("%d %d %d %d", &L[i][0].x, &L[i][0].y, &L[i][1].x, &L[i][1].y);

if( L[i][1] < L[i][0] )

swap(L[i][0], L[i][1]);

}

for( i = 1, e = 0; i <= N; ++i )

{

E[e++] = eve(L[i][0].x, i, +1);

E[e++] = eve(L[i][1].x, i, -1);

}

if( !solve(e) )

puts("NO");

return 0;

}

/\*\*\*\*\*\*\*三维几何\*\*\*\*\*\*\*/

#include <iostream>

#include <cstdio>

#include <cmath>

#define zero( x ) ( ( ( x ) > 0 ? ( x ) : -( x ) ) < eps )

const double eps = 1e-8;

struct point3

{

double x, y, z;

};

struct line3

{

point3 a, b;

};

struct plane3

{

point3 a, b, c;

};

//计算 Cross Product U \* V

point3 xmult( point3 u, point3 v )

{

point3 ret;

ret.x = u.y \* v.x - v.y \* u.z;

ret.y = u.z \* v.x - u.x \* v.z;

ret.z = u.x \* v.y - u.y \* v.x;

return ret;

}

//计算 Dot Product U ? V

double dmult( point3 u, point3 v )

{

return u.x \* v.x + u.y \* v.y + u.z \* v.z;

}

//矢量差 U - V

point3 subt( point3 u, point3 v )

{

point3 ret;

ret.x = u.x - v.x;

ret.y = u.y - v.y;

ret.z = u.z - v.z;

return ret;

}

//取平面法向量

point3 pvec( plane3 s )

{

return xmult( subt( s.a, s.b ), subt( s.b, s.c ) );

}

point3 pvec( point3 s1, point3 s2, point3 s3 )

{

return xmult( subt( s1, s2 ), subt( s2, s3 ) );

}

//两点距离，单参数取向量大小

double distance( point3 p1, point3 p2 )

{

return sqrt( ( p1.x - p2.x ) \* ( p1.x - p2.x ) + ( p1.y - p2.x ) \* ( p1.y - p2.y ) + ( p1.z - p2.z ) \* ( p1.z - p2.z ) );

}

//向量大小

double vlen( point3 p )

{

return sqrt( p.x \* p.x + p.y \* p.y + p.z \* p.z );

}

//判定三点共线

int dots\_inline( point3 p1, point3 p2, point3 p3 )

{

return vlen( xmult( subt( p1, p2 ), subt( p2, p3 ) ) ) < eps;

}

//判定四点共面

int dots\_onplane( point3 a, point3 b, point3 c, point3 d )

{

return zero( dmult( pvec( a, b, c ), subt( d, a ) ) );

}

//判定点是否在线段上，包括端点和共线

int dot\_online\_in( point3 p, line3 l )

{

return zero( vlen( xmult( subt( p, l.a ), subt( p, l.b ) ) ) ) && ( l.a.x - p.x ) \* ( l.b.x - p.x ) < eps && ( l.a.y - p.y ) \* ( l.b.y - p.y ) < eps && ( l.a.z - p.z ) \* ( l.b.z - p.z ) < eps;

}

int dot\_online\_in( point3 p, point3 l1, point3 l2 )

{

return zero( vlen( xmult( subt( p, l1 ), subt( p, l2 ) ) ) ) && ( l1.x - p.x ) \* ( l2.x - p.x ) < eps && ( l1.y - p.y ) \* ( l2.y - p.y ) < eps && ( l1.z - p.z ) \* ( l2.z - p.z ) < eps;

}

//判定点是否在线段上，不包括端点

int dot\_online\_ex( point3 p, line3 l )

{

return dot\_online\_in( p, l ) && ( !zero( p.x - l.a.x ) || ! zero( p.y - l.a.y ) || !zero( p.z - l.a.z ) ) && ( !zero( p.x - l.b.x ) || !zero( p.y - l.b.y ) || !zero( p.z - l.b.z ) );

}

int dot\_online\_ex( point3 p, point3 l1, point3 l2 )

{

return dot\_online\_in( p, l1, l2 ) && ( !zero( p.x - l1.x ) || !zero( p.y - l1.y ) || !zero( p.z - l1.z ) ) && ( !zero( p.x - l2.x ) || !zero( p.y - l2.y ) || !zero( p.z - l2.z ) );

}

//判定点是否在空间三角形上，包括边界，三点共线无意义

int dot\_inplane\_in( point3 p, plane3 s )

{

return zero( vlen( xmult( subt( s.a, s.b ), subt( s.a, s.c ) ) ) - vlen( xmult( subt( p, s.a ), subt( p, s.b ) ) ) - vlen( xmult( subt( p, s.b ), subt( p, s.c ) ) ) - vlen( xmult( subt( p, s.c ), subt( p, s.a ) ) ) );

}

int dot\_inplane\_in( point3 p, point3 s1, point3 s2, point3 s3 )

{

return zero( vlen( xmult( subt( s1, s2 ), subt( s1, s3 ) ) ) - vlen( xmult( subt( p, s1 ), subt( p, s2 ) ) ) - vlen( xmult( subt( p, s2 ), subt( p, s3 ) ) ) - vlen( xmult( subt( p, s3 ), subt( p, s1 ) ) ) );

}

//判定点是否在空间三角形上，不包括边界，三点共线无意义

int dot\_inplane\_ex( point3 p, plane3 s )

{

return dot\_inplane\_in( p, s ) && vlen( xmult( subt( p, s.a ), subt( p, s.b ) ) ) > eps && vlen( xmult( subt( p, s.b ), subt( p, s.c ) ) ) > eps && vlen( xmult( subt( p, s.c ), subt( p, s.a ) ) ) > eps;

}

int dot\_inplane\_ex( point3 p, point3 s1, point3 s2, point3 s3 )

{

return dot\_inplane\_in( p, s1, s2, s3 ) && vlen( xmult( subt( p, s1 ), subt( p, s2 ) ) ) > eps && vlen( xmult( subt( p, s2 ), subt( p, s3 ) ) ) > eps && vlen( xmult( subt( p, s3 ), subt( p, s1 ) ) ) > eps;

}

//判定两点在线段同侧，点在线段上返回0，不共面无意义

int same\_side( point3 p1, point3 p2, line3 l )

{

return dmult( xmult( subt( l.a, l.b ), subt( p1, l.b ) ), xmult( subt( l.a, l.b ), subt( p2, l.b ) ) ) > eps;

}

int same\_side( point3 p1, point3 p2, point3 l1, point3 l2 )

{

return dmult( xmult( subt( l1, l2 ), subt( p1, l2 ) ), xmult( subt( l1, l2 ), subt( p2, l2 ) ) ) > eps;

}

//判定两点在线段异侧，点在平面上返回0

int opposite\_side( point3 p1, point3 p2, line3 l )

{

return dmult( xmult( subt( l.a, l.b ), subt( p1, l.b ) ), xmult( subt( l.a, l.b ), subt( p2, l.b ) ) ) < -eps;

} int opposite\_side( point3 p1, point3 p2, point3 l1, point3 l2 )

{

return dmult( xmult( subt( l1, l2 ), subt( p1, l2 ) ), xmult( subt( l1, l2 ), subt( p2, l2 ) ) ) < -eps;

}

//判定两点在平面同侧，点在平面上返回0

int same\_side( point3 p1, point3 p2, plane3 s )

{

return dmult( pvec( s ), subt( p1, s.a ) ) \* dmult( pvec( s ), subt( p2, s.a ) ) > eps;

}

int same\_side( point3 p1, point3 p2, point3 s1, point3 s2, point3 s3 )

{

return dmult( pvec( s1, s2, s3 ), subt( p1, s1 ) ) \* dmult( pvec( s1, s2, s3 ), subt( p2, s1 ) ) > eps;

}

//判定两点在平面异侧，点在平面上返回0

int opposite\_side( point3 p1, point3 p2, plane3 s )

{

return dmult( pvec( s ), subt( p1, s.a ) ) \* dmult( pvec( s ), subt( p2, s.a ) ) < -eps;

}

int opposite\_side( point3 p1, point3 p2, point3 s1, point3 s2, point3 s3 )

{

return dmult( pvec( s1, s2, s3 ), subt( p1, s1 ) ) \* dmult( pvec( s1, s2, s3 ), subt( p2, s1 ) ) < eps;

}

//判定两直线平行

int parallel( line3 u, line3 v )

{

return vlen( xmult( subt( u.a, u.b ), subt( v.a, v.b ) ) ) < eps;

}

int parallel( point3 u1, point3 u2, point3 u3, point3 v1, point3 v2, point3 v3 )

{

return vlen( xmult( pvec( u1, u2, u3 ), pvec( v1, v2, v3 ) ) ) < eps;

}

//判定直线与平面平行

int parallel( line3 l, plane3 s )

{

return zero( dmult( subt( l.a, l.b ), pvec( s ) ) );

}

int parallel( point3 l1, point3 l2, point3 s1, point3 s2, point3 s3 )

{

return zero( dmult( subt( l1, l2 ), pvec( s1, s2, s3 ) ) );

}

//判定两直线垂直

int perpendicular( line3 u, line3 v )

{

return zero( dmult( subt( u.a, u.b ), subt( v.a, v.b ) ) );

}

int perpendicular( point3 u1, point3 u2, point3 v1, point3 v2 )

{

return zero( dmult( subt( u1, u2 ), subt( v1, v2 ) ) );

}

//判定两平面垂直

int perpendicular( plane3 u, plane3 v )

{

return zero( dmult( pvec( u ), pvec( v ) ) );

}

int perpendicular( point3 u1, point3 u2, point3 u3, point3 v1, point3 v2, point3 v3 )

{

return zero( dmult( pvec( u1, u2, u3 ), pvec( v1, v2, v3 ) ) );

}

//判定直线与平面平行

int perpendicular( line l, plane3 s )

{

return vlen( xmult( subt( l.a, l.b ), pvec( s ) ) ) < eps;

}

int perpendicular( point3 l1, point3 l2, point3 s1, point3 s2, point3 s3 )

{

return vlen( xmult( subt( l1, l2 ), pvec( s1, s2, s3 ) ) ) < eps;

}

//判定两直线相交，包括端点和部分重合

int intersect\_in( line3 u, line3 v )

{

if( !dots\_onplane( u.a, u.b, v.a, v.b ) )

return 0;

if( !dotsinline( u.a, u.b, v.a ) || !dots\_inline( u.a, u.b, v.b ) )

return !same\_side( u.a, u.b, v ) && !same\_side( v.a, v.b, u );

return dot\_online\_in( u.a, v ) || dot\_online\_in( u.b, v ) || dot\_online\_in( v.a, u ) || dot\_online\_in( v.b, u );

}

int intersect\_in( point3 u1, point3 u2, opint3 v1, point3 v2 )

{

if( !dots\_onplane( u1, u2, v1, v2 ) )

return 0;

if( !dots\_inline( u1, u2, v1 ) || !dots\_inline( u1, u2, v2 ) )

return !same\_side( u1, u2, v1, v2 ) && !same\_side( v1, v2, u1, u2 );

return dot\_online\_in( u1, v1, v2 ) || dot\_online\_in( u2, v1, v2 ) || dot\_online\_in( v1, u1, u2 ) || dot\_online\_in( v2, u1, u2 );

}

//判定两线段相交，不包括端点和部分重合

int intersect\_ex( line3 u, line3 v )

{

return dots\_onplane( u.a, u.b, v.a, v.b ) && opposite\_side( u.a, u.b, v ) && opposite\_side( v.a, v.b, u );

}

int intersect\_ex( point3 u1, point3 u2, point3 v1, point3 v2 )

{

return dots\_onplane( u1, u2, v1, v2 ) && opposite\_side( u1, u2, v1, v2 ) && opposite\_side( v1, v2, u1, u2 );

}

//判定线段与空间三角形相交，包括交于边界和（部分）包含

int intersect\_in( line3 l, plane3 s )

{

return !same\_side( l.a, l.b, s ) && !same\_side( s.a, s.b, l.a, l.b, s.c ) && !same\_side( s.b, s.c, l.a, l.b, s.a ) && !same\_side( s.c, s.a, l.a, l.b, s.b );

}

int intersect\_in( point3 l1, point3 l2, point3 s1, point3 s2, point3 s3 )

{

return !same\_side( l1, l2, s1, s2, s3 ) && !same\_side( s1, s2, l1, l2, s3 ) && !same\_side( s2, s3, l1, l2, s1 ) && !same\_side( s3, s1, l1, l2, s2 );

}

//判定线段与空间三角形相交，不包括交于边界和（部分）包含

int intersect\_ex( line3 l, plane3 s )

{

return opposite\_side( l.a, l.b, s ) && opposite\_side( s.a, s.b, l.a, l.b, s.c ) && opposite\_side( s.b, s.c, l.a, l.b, s.a ) && opposite\_side( s.c, s.a, l.a, l.b, s.b );

}

int intersect\_ex( point3 l1, point3 l2, point3 s1, point3 s2, point3 s3 )

{

return opposite\_side( l1, l2, s1, s2, s3 ) && opposite\_side( s1, s2, l1, l2, s3 ) && opposite\_side( s2, s3, l1, l2, s1 ) && opposite\_side( s3, s1, l1, l2, s2 );

}

//计算两直线交点，注意事先判断直线是否共面和平行

//线段交点另外判断线段相交（同时还是要判断是否平行）

point3 intersection( line3 u, line3 v )

{

point3 ret = u.a;

double t = ( ( u.a.x - v.a.x ) \* ( v.a.y - v.b.y ) - ( u.a.y - v.a.y ) \* ( v.a.x - v.b.x ) ) / ( ( u.a.x - u.b.x ) \* ( v.a.y - v.b.y ) - ( u.a.y - u.b.y ) \* ( v.a.x - v.b.x ) );

ret.x += ( u.b.x - u.a.x ) \* t;

ret.y += ( u.b.y - u.a.y ) \* t;

ret.z += ( u.b.z - u.a.z ) \* t;

return ret;

}

point3 intersection( point3 u1, point3 u2, point3 v1, point3 v2 )

{

point3 ret = u1;

double t = ( ( u1.x - v1.x ) \* ( v1.y - v2.y ) - ( u1.y - v1.y ) \* ( v1.x - v2.x ) ) / ( ( u1.x - u2.x ) \* ( v1.y - v2.y ) - ( u1.y - u2.y ) \* ( v1.x - v2.x ) );

ret.x += ( u2.x - u1.x ) \* t;

ret.y += ( u2.y - u1.y ) \* t;

ret.z += ( u2.z - u1.z ) \* t;

return ret;

}

//计算直线与平面交点，注意事先判断是否平行，并保证三点不共线

//线段和空间三角形交点另外判断

point3 intersection( line3 l, plane3 s )

{

point3 ret = pvec( s );

double t = ( ret.x \* ( s.a.x - l.a.x ) + ret.y \* ( s.a.y - l.a.y ) + ret.z \* ( s.a.z - l.a.z ) ) / ( ret.x \* ( l.b.x - l.a.x ) + ret.y \* ( l.b.y - l.a.y ) + ret.z \* ( l.b.z - l.a.z ) );

ret.x = l.a.x + ( l.b.x - l.a.x ) \* t;

ret.y = l.a.y + ( l.b.y - l.a.y ) \* t;

ret.z = l.a.z + ( l.b.z - l.a.z ) \* t;

return ret;

}

point3 intersection( point3 l1, point3 l2, point3 s1, point3 s2, point3 s3 )

{

point3 ret = pvec( s1, s2, s3 );

double t = ( ret.x \* ( s1.x - l1.x ) + ret.y \* ( s1.y - l1.y ) + ret.z \* ( s1.z - l1.z ) ) / ( ret.x \* ( l2.x - l1.x ) + ret.y \* ( l2.y - l1.y ) + ret.z \* ( l2.z - l1.z ) );

ret.x = l1.x + ( l2.x - l1.x ) \* t;

ret.y = l1.y + ( l2.y - l1.y ) \* t;

ret.z = l1.z + ( l2.z - l1.z ) \* t;

return ret;

}

//计算两平面交线，注意事先判断是否平行，并保证三点不共线

line3 intersection( plane3 u, plane3 v )

{

line3 ret;

ret.a = parallel( v.a, v.b, u.a, u.b, u.c ) ? intersection( v.b, v.c, u.a, u.b, u.c ) : intersection( v.a, v.b, u.a, u.b, u.c );

ret.b = parallel( v.c, v.a, u.a, u.b, u.c ) ? intersection( v.b, v.c, u.a, u.b, u.c ) : intersection( v.c, v.a, u.a, u.b, u.c );

return ret;

}

line3 intersection( point3 u1, point3 u2, point3 u3, point3 v1, point3 v2, point3 v3 )

{

line3 ret;

ret.a = parallel( v1, v2, u1, u2, u3 ) ? intersection( v2, v3, u1, u2, u3 ) : intersection( v1, v2, u1, u2, u3 );

ret.b = parallel( v3, v1, u1, u2, u3 ) ? intersection( v2, v3, u1, u2, u3 ) : intersection( v3, v1, u1, u2, u3 );

return ret;

}

//点到直线距离

double ptoline( point3 p, line3 l )

{

return vlen( xmult( subt( p, l.a ), subt( l.b, l.a ) ) ) / distance( l.a, l.b );

}

double ptoline( point3 p, opint3 l1, point3 l2 )

{

return vlen( xmult( subt( p, l1 ), subt( l2, l1 ) ) ) / distance( l1, l2 );

}

//点到平面距离

double ptoplane( point3 p, plane3 s )

{

return fabs( dmult( pvec( s ), subt( p, s.a ) ) ) / vlen( pvec( s ) );

}

double ptoplane( point3 p, point3 s1, point3 s2, point3 s3 )

{

return fabs( dmult( pvec( s1, s2, s3 ), subt( p, s1 ) ) ) / vlen( pvec( s1, s2, s3 ) );

}

//直线到直线距离

double linetoline( lien3 u, line3 v )

{

point3 n = xmult( subt( u.a, u.b ), subt( v.a, v.b ) );

return fabs( dmult( subt( u.a, v.a ), n ) ) / vlen( n );

}

double linetoline( point3 u1, point3 u2, point3 v1, point3 v2 )

{

point3 n = xmult( subt( u1, u2 ), subt( v1, v2 ) );

return fabs( dmult( subt( u1, v1 ), n ) ) / vlen( n );

}

//两直线夹角 cos 值

double angle\_cos( line3 u, line3 v )

{

return dmult( subt( u.a, u.b ), subt( v.a, v.b ) ) / vlen( subt( u.a, u.b ) ) / vlen( subt( v.a, v.b ) );

}

double angle\_cos( point3 u1, point3 u2, point3 v1, point3 v2 )

{

return dmult( subt( u1, u2 ), subt( v1, v2 ) ) / vlen( sut( u1, u2 ) ) / vlen( subt( v1, v2 ) );

}

//两平面夹角 cos 值

double angle\_cos( plane3 u, plane3 v )

{

return dmult( pvec( u ), pvec( v ) ) / vlen( pvec( u ) ) / vlen( pvec( v ) );

}

double angle\_cos( point3 u1, point3 u2, point3 v1, point3 v2, point3 v3 )

{

return dmult( pvec( u1, u2, u3 ), pvec( v1, v2, v3 ) ) / vlen( pvec( u1, u2, u3 ) ) / vlen( pvec( v1, v2, v3 ) );

}

//直线平面夹角 sin 值

double angle\_sin( line3 l, plane3 s )

{

return dmult( subt( l.a, l.b ), pvec( s ) ) / vlen( subt( l.a, l.b ) ) / vlen( pvec( s ) );

}

double angle\_sin( point3 l1, point3 l2, point3 s1, point3 s2, point3 s3 )

{

return dmult( subt( l1, l2 ), pvc( s1, s2, s3 ) ) / vlen( subt( l1, l2 ) ) / vlen( pvec( s1, s2, s3 ) );

}

下面的来自戴神：

#include<vector>

#include<list>

#include<map>

#include<set>

#include<deque>

#include<queue>

#include<stack>

#include<bitset>

#include<algorithm>

#include<functional>

#include<numeric>

#include<utility>

#include<iostream>

#include<sstream>

#include<iomanip>

#include<cstdio>

#include<cmath>

#include<cstdlib>

#include<cctype>

#include<string>

#include<cstring>

#include<cstdio>

#include<cmath>

#include<cstdlib>

#include<ctime>

#include<climits>

#include<complex>

#define mp make\_pair

#define pb push\_back

using namespace std;

const double eps=1e-8;

const double inf=1e20;

const double pi=acos(-1.0);

const int maxp=11;

int dblcmp(double d)

{

if (fabs(d)<eps)return 0;

return d>eps?1:-1;

}

inline double sqr(double x){return x\*x;}

struct point

{

double x,y;

point(){}

point(double \_x,double \_y):

x(\_x),y(\_y){};

void input()

{

scanf("%lf%lf",&x,&y);

}

void output()

{

printf("%.2f %.2f\n",x,y);

}

bool operator==(point a)const

{

return dblcmp(a.x-x)==0&&dblcmp(a.y-y)==0;

}

bool operator<(point a)const

{

return dblcmp(a.x-x)==0?dblcmp(y-a.y)<0:x<a.x;

}

double len()

{

return hypot(x,y);

}

double len2()

{

return x\*x+y\*y;

}

double distance(point p)

{

return hypot(x-p.x,y-p.y);

}

point add(point p)

{

return point(x+p.x,y+p.y);

}

point sub(point p)

{

return point(x-p.x,y-p.y);

}

point mul(double b)

{

return point(x\*b,y\*b);

}

point div(double b)

{

return point(x/b,y/b);

}

double dot(point p)

{

return x\*p.x+y\*p.y;

}

double det(point p)

{

return x\*p.y-y\*p.x;

}

double rad(point a,point b)

{

point p=\*this;

return fabs(atan2(fabs(a.sub(p).det(b.sub(p))),a.sub(p).dot(b.sub(p))));

}

point trunc(double r)

{

double l=len();

if (!dblcmp(l))return \*this;

r/=l;

return point(x\*r,y\*r);

}

point rotleft()

{

return point(-y,x);

}

point rotright()

{

return point(y,-x);

}

point rotate(point p,double angle)//绕点p逆时针旋转angle角度

{

point v=this->sub(p);

double c=cos(angle),s=sin(angle);

return point(p.x+v.x\*c-v.y\*s,p.y+v.x\*s+v.y\*c);

}

};

struct line

{

point a,b;

line(){}

line(point \_a,point \_b)

{

a=\_a;

b=\_b;

}

bool operator==(line v)

{

return (a==v.a)&&(b==v.b);

}

//倾斜角angle

line(point p,double angle)

{

a=p;

if (dblcmp(angle-pi/2)==0)

{

b=a.add(point(0,1));

}

else

{

b=a.add(point(1,tan(angle)));

}

}

//ax+by+c=0

line(double \_a,double \_b,double \_c)

{

if (dblcmp(\_a)==0)

{

a=point(0,-\_c/\_b);

b=point(1,-\_c/\_b);

}

else if (dblcmp(\_b)==0)

{

a=point(-\_c/\_a,0);

b=point(-\_c/\_a,1);

}

else

{

a=point(0,-\_c/\_b);

b=point(1,(-\_c-\_a)/\_b);

}

}

void input()

{

a.input();

b.input();

}

void adjust()

{

if (b<a)swap(a,b);

}

double length()

{

return a.distance(b);

}

double angle()//直线倾斜角 0<=angle<180

{

double k=atan2(b.y-a.y,b.x-a.x);

if (dblcmp(k)<0)k+=pi;

if (dblcmp(k-pi)==0)k-=pi;

return k;

}

//点和线段关系

//1 在逆时针

//2 在顺时针

//3 平行

int relation(point p)

{

int c=dblcmp(p.sub(a).det(b.sub(a)));

if (c<0)return 1;

if (c>0)return 2;

return 3;

}

bool pointonseg(point p)

{

return dblcmp(p.sub(a).det(b.sub(a)))==0&&dblcmp(p.sub(a).dot(p.sub(b)))<=0;

}

bool parallel(line v)

{

return dblcmp(b.sub(a).det(v.b.sub(v.a)))==0;

}

//2 规范相交

//1 非规范相交

//0 不相交

int segcrossseg(line v)

{

int d1=dblcmp(b.sub(a).det(v.a.sub(a)));

int d2=dblcmp(b.sub(a).det(v.b.sub(a)));

int d3=dblcmp(v.b.sub(v.a).det(a.sub(v.a)));

int d4=dblcmp(v.b.sub(v.a).det(b.sub(v.a)));

if ((d1^d2)==-2&&(d3^d4)==-2)return 2;

return (d1==0&&dblcmp(v.a.sub(a).dot(v.a.sub(b)))<=0||

d2==0&&dblcmp(v.b.sub(a).dot(v.b.sub(b)))<=0||

d3==0&&dblcmp(a.sub(v.a).dot(a.sub(v.b)))<=0||

d4==0&&dblcmp(b.sub(v.a).dot(b.sub(v.b)))<=0);

}

int linecrossseg(line v)//\*this seg v line

{

int d1=dblcmp(b.sub(a).det(v.a.sub(a)));

int d2=dblcmp(b.sub(a).det(v.b.sub(a)));

if ((d1^d2)==-2)return 2;

return (d1==0||d2==0);

}

int linecrossline(line v)

{

if ((\*this).parallel(v))

{

return v.relation(a)==3;

}

return 2;

}

point crosspoint(line v)

{

double a1=v.b.sub(v.a).det(a.sub(v.a));

double a2=v.b.sub(v.a).det(b.sub(v.a));

return point((a.x\*a2-b.x\*a1)/(a2-a1),(a.y\*a2-b.y\*a1)/(a2-a1));

}

double dispointtoline(point p)

{

return fabs(p.sub(a).det(b.sub(a)))/length();

}

double dispointtoseg(point p)

{

if (dblcmp(p.sub(b).dot(a.sub(b)))<0||dblcmp(p.sub(a).dot(b.sub(a)))<0)

{

return min(p.distance(a),p.distance(b));

}

return dispointtoline(p);

}

point lineprog(point p)

{

return a.add(b.sub(a).mul(b.sub(a).dot(p.sub(a))/b.sub(a).len2()));

}

point symmetrypoint(point p)

{

point q=lineprog(p);

return point(2\*q.x-p.x,2\*q.y-p.y);

}

};

struct circle

{

point p;

double r;

circle(){}

circle(point \_p,double \_r):

p(\_p),r(\_r){};

circle(double x,double y,double \_r):

p(point(x,y)),r(\_r){};

circle(point a,point b,point c)//三角形的外接圆

{

p=line(a.add(b).div(2),a.add(b).div(2).add(b.sub(a).rotleft())).crosspoint(line(c.add(b).div(2),c.add(b).div(2).add(b.sub(c).rotleft())));

r=p.distance(a);

}

circle(point a,point b,point c,bool t)//三角形的内切圆

{

line u,v;

double m=atan2(b.y-a.y,b.x-a.x),n=atan2(c.y-a.y,c.x-a.x);

u.a=a;

u.b=u.a.add(point(cos((n+m)/2),sin((n+m)/2)));

v.a=b;

m=atan2(a.y-b.y,a.x-b.x),n=atan2(c.y-b.y,c.x-b.x);

v.b=v.a.add(point(cos((n+m)/2),sin((n+m)/2)));

p=u.crosspoint(v);

r=line(a,b).dispointtoseg(p);

}

void input()

{

p.input();

scanf("%lf",&r);

}

void output()

{

printf("%.2lf %.2lf %.2lf\n",p.x,p.y,r);

}

bool operator==(circle v)

{

return ((p==v.p)&&dblcmp(r-v.r)==0);

}

bool operator<(circle v)const

{

return ((p<v.p)||(p==v.p)&&dblcmp(r-v.r)<0);

}

double area()

{

return pi\*sqr(r);

}

double circumference()

{

return 2\*pi\*r;

}

//0 圆外

//1 圆上

//2 圆内

int relation(point b)

{

double dst=b.distance(p);

if (dblcmp(dst-r)<0)return 2;

if (dblcmp(dst-r)==0)return 1;

return 0;

}

int relationseg(line v)

{

double dst=v.dispointtoseg(p);

if (dblcmp(dst-r)<0)return 2;

if (dblcmp(dst-r)==0)return 1;

return 0;

}

int relationline(line v)

{

double dst=v.dispointtoline(p);

if (dblcmp(dst-r)<0)return 2;

if (dblcmp(dst-r)==0)return 1;

return 0;

}

//过a b两点 半径r的两个圆

int getcircle(point a,point b,double r,circle&c1,circle&c2)

{

circle x(a,r),y(b,r);

int t=x.pointcrosscircle(y,c1.p,c2.p);

if (!t)return 0;

c1.r=c2.r=r;

return t;

}

//与直线u相切 过点q 半径r1的圆

int getcircle(line u,point q,double r1,circle &c1,circle &c2)

{

double dis=u.dispointtoline(q);

if (dblcmp(dis-r1\*2)>0)return 0;

if (dblcmp(dis)==0)

{

c1.p=q.add(u.b.sub(u.a).rotleft().trunc(r1));

c2.p=q.add(u.b.sub(u.a).rotright().trunc(r1));

c1.r=c2.r=r1;

return 2;

}

line u1=line(u.a.add(u.b.sub(u.a).rotleft().trunc(r1)),u.b.add(u.b.sub(u.a).rotleft().trunc(r1)));

line u2=line(u.a.add(u.b.sub(u.a).rotright().trunc(r1)),u.b.add(u.b.sub(u.a).rotright().trunc(r1)));

circle cc=circle(q,r1);

point p1,p2;

if (!cc.pointcrossline(u1,p1,p2))cc.pointcrossline(u2,p1,p2);

c1=circle(p1,r1);

if (p1==p2)

{

c2=c1;return 1;

}

c2=circle(p2,r1);

return 2;

}

//同时与直线u,v相切 半径r1的圆

int getcircle(line u,line v,double r1,circle &c1,circle &c2,circle &c3,circle &c4)

{

if (u.parallel(v))return 0;

line u1=line(u.a.add(u.b.sub(u.a).rotleft().trunc(r1)),u.b.add(u.b.sub(u.a).rotleft().trunc(r1)));

line u2=line(u.a.add(u.b.sub(u.a).rotright().trunc(r1)),u.b.add(u.b.sub(u.a).rotright().trunc(r1)));

line v1=line(v.a.add(v.b.sub(v.a).rotleft().trunc(r1)),v.b.add(v.b.sub(v.a).rotleft().trunc(r1)));

line v2=line(v.a.add(v.b.sub(v.a).rotright().trunc(r1)),v.b.add(v.b.sub(v.a).rotright().trunc(r1)));

c1.r=c2.r=c3.r=c4.r=r1;

c1.p=u1.crosspoint(v1);

c2.p=u1.crosspoint(v2);

c3.p=u2.crosspoint(v1);

c4.p=u2.crosspoint(v2);

return 4;

}

//同时与不相交圆cx,cy相切 半径为r1的圆

int getcircle(circle cx,circle cy,double r1,circle&c1,circle&c2)

{

circle x(cx.p,r1+cx.r),y(cy.p,r1+cy.r);

int t=x.pointcrosscircle(y,c1.p,c2.p);

if (!t)return 0;

c1.r=c2.r=r1;

return t;

}

int pointcrossline(line v,point &p1,point &p2)//求与线段交要先判断relationseg

{

if (!(\*this).relationline(v))return 0;

point a=v.lineprog(p);

double d=v.dispointtoline(p);

d=sqrt(r\*r-d\*d);

if (dblcmp(d)==0)

{

p1=a;

p2=a;

return 1;

}

p1=a.sub(v.b.sub(v.a).trunc(d));

p2=a.add(v.b.sub(v.a).trunc(d));

return 2;

}

//5 相离

//4 外切

//3 相交

//2 内切

//1 内含

int relationcircle(circle v)

{

double d=p.distance(v.p);

if (dblcmp(d-r-v.r)>0)return 5;

if (dblcmp(d-r-v.r)==0)return 4;

double l=fabs(r-v.r);

if (dblcmp(d-r-v.r)<0&&dblcmp(d-l)>0)return 3;

if (dblcmp(d-l)==0)return 2;

if (dblcmp(d-l)<0)return 1;

}

int pointcrosscircle(circle v,point &p1,point &p2)

{

int rel=relationcircle(v);

if (rel==1||rel==5)return 0;

double d=p.distance(v.p);

double l=(d+(sqr(r)-sqr(v.r))/d)/2;

double h=sqrt(sqr(r)-sqr(l));

p1=p.add(v.p.sub(p).trunc(l).add(v.p.sub(p).rotleft().trunc(h)));

p2=p.add(v.p.sub(p).trunc(l).add(v.p.sub(p).rotright().trunc(h)));

if (rel==2||rel==4)

{

return 1;

}

return 2;

}

//过一点做圆的切线 (先判断点和圆关系)

int tangentline(point q,line &u,line &v)

{

int x=relation(q);

if (x==2)return 0;

if (x==1)

{

u=line(q,q.add(q.sub(p).rotleft()));

v=u;

return 1;

}

double d=p.distance(q);

double l=sqr(r)/d;

double h=sqrt(sqr(r)-sqr(l));

u=line(q,p.add(q.sub(p).trunc(l).add(q.sub(p).rotleft().trunc(h))));

v=line(q,p.add(q.sub(p).trunc(l).add(q.sub(p).rotright().trunc(h))));

return 2;

}

double areacircle(circle v)

{

int rel=relationcircle(v);

if (rel>=4)return 0.0;

if (rel<=2)return min(area(),v.area());

double d=p.distance(v.p);

double hf=(r+v.r+d)/2.0;

double ss=2\*sqrt(hf\*(hf-r)\*(hf-v.r)\*(hf-d));

double a1=acos((r\*r+d\*d-v.r\*v.r)/(2.0\*r\*d));

a1=a1\*r\*r;

double a2=acos((v.r\*v.r+d\*d-r\*r)/(2.0\*v.r\*d));

a2=a2\*v.r\*v.r;

return a1+a2-ss;

}

double areatriangle(point a,point b)

{

if (dblcmp(p.sub(a).det(p.sub(b))==0))return 0.0;

point q[5];

int len=0;

q[len++]=a;

line l(a,b);

point p1,p2;

if (pointcrossline(l,q[1],q[2])==2)

{

if (dblcmp(a.sub(q[1]).dot(b.sub(q[1])))<0)q[len++]=q[1];

if (dblcmp(a.sub(q[2]).dot(b.sub(q[2])))<0)q[len++]=q[2];

}

q[len++]=b;

if (len==4&&(dblcmp(q[0].sub(q[1]).dot(q[2].sub(q[1])))>0))swap(q[1],q[2]);

double res=0;

int i;

for (i=0;i<len-1;i++)

{

if (relation(q[i])==0||relation(q[i+1])==0)

{

double arg=p.rad(q[i],q[i+1]);

res+=r\*r\*arg/2.0;

}

else

{

res+=fabs(q[i].sub(p).det(q[i+1].sub(p))/2.0);

}

}

return res;

}

};

struct polygon

{

int n;

point p[maxp];

line l[maxp];

void input()

{

for (int i=0;i<n;i++)

{

p[i].input();

}

}

void add(point q)

{

p[n++]=q;

}

void getline()

{

for (int i=0;i<n;i++)

{

l[i]=line(p[i],p[(i+1)%n]);

}

}

struct cmp

{

point p;

cmp(const point &p0){p=p0;}

bool operator()(const point &aa,const point &bb)

{

point a=aa,b=bb;

int d=dblcmp(a.sub(p).det(b.sub(p)));

if (d==0)

{

return dblcmp(a.distance(p)-b.distance(p))<0;

}

return d>0;

}

};

void norm()

{

point mi=p[0];

for (int i=1;i<n;i++)mi=min(mi,p[i]);

sort(p,p+n,cmp(mi));

}

void getconvex(polygon &convex)

{

int i,j,k;

sort(p,p+n);

convex.n=n;

for (i=0;i<min(n,2);i++)

{

convex.p[i]=p[i];

}

if (n<=2)return;

int &top=convex.n;

top=1;

for (i=2;i<n;i++)

{

while (top&&convex.p[top].sub(p[i]).det(convex.p[top-1].sub(p[i]))<=0)

top--;

convex.p[++top]=p[i];

}

int temp=top;

convex.p[++top]=p[n-2];

for (i=n-3;i>=0;i--)

{

while (top!=temp&&convex.p[top].sub(p[i]).det(convex.p[top-1].sub(p[i]))<=0)

top--;

convex.p[++top]=p[i];

}

}

bool isconvex()

{

bool s[3];

memset(s,0,sizeof(s));

int i,j,k;

for (i=0;i<n;i++)

{

j=(i+1)%n;

k=(j+1)%n;

s[dblcmp(p[j].sub(p[i]).det(p[k].sub(p[i])))+1]=1;

if (s[0]&&s[2])return 0;

}

return 1;

}

//3 点上

//2 边上

//1 内部

//0 外部

int relationpoint(point q)

{

int i,j;

for (i=0;i<n;i++)

{

if (p[i]==q)return 3;

}

getline();

for (i=0;i<n;i++)

{

if (l[i].pointonseg(q))return 2;

}

int cnt=0;

for (i=0;i<n;i++)

{

j=(i+1)%n;

int k=dblcmp(q.sub(p[j]).det(p[i].sub(p[j])));

int u=dblcmp(p[i].y-q.y);

int v=dblcmp(p[j].y-q.y);

if (k>0&&u<0&&v>=0)cnt++;

if (k<0&&v<0&&u>=0)cnt--;

}

return cnt!=0;

}

//1 在多边形内长度为正

//2 相交或与边平行

//0 无任何交点

int relationline(line u)

{

int i,j,k=0;

getline();

for (i=0;i<n;i++)

{

if (l[i].segcrossseg(u)==2)return 1;

if (l[i].segcrossseg(u)==1)k=1;

}

if (!k)return 0;

vector<point>vp;

for (i=0;i<n;i++)

{

if (l[i].segcrossseg(u))

{

if (l[i].parallel(u))

{

vp.pb(u.a);

vp.pb(u.b);

vp.pb(l[i].a);

vp.pb(l[i].b);

continue;

}

vp.pb(l[i].crosspoint(u));

}

}

sort(vp.begin(),vp.end());

int sz=vp.size();

for (i=0;i<sz-1;i++)

{

point mid=vp[i].add(vp[i+1]).div(2);

if (relationpoint(mid)==1)return 1;

}

return 2;

}

//直线u切割凸多边形左侧

//注意直线方向

void convexcut(line u,polygon &po)

{

int i,j,k;

int &top=po.n;

top=0;

for (i=0;i<n;i++)

{

int d1=dblcmp(p[i].sub(u.a).det(u.b.sub(u.a)));

int d2=dblcmp(p[(i+1)%n].sub(u.a).det(u.b.sub(u.a)));

if (d1>=0)po.p[top++]=p[i];

if (d1\*d2<0)po.p[top++]=u.crosspoint(line(p[i],p[(i+1)%n]));

}

}

double getcircumference()

{

double sum=0;

int i;

for (i=0;i<n;i++)

{

sum+=p[i].distance(p[(i+1)%n]);

}

return sum;

}

double getarea()

{

double sum=0;

int i;

for (i=0;i<n;i++)

{

sum+=p[i].det(p[(i+1)%n]);

}

return fabs(sum)/2;

}

bool getdir()//1代表逆时针 0代表顺时针

{

double sum=0;

int i;

for (i=0;i<n;i++)

{

sum+=p[i].det(p[(i+1)%n]);

}

if (dblcmp(sum)>0)return 1;

return 0;

}

point getbarycentre()

{

point ret(0,0);

double area=0;

int i;

for (i=1;i<n-1;i++)

{

double tmp=p[i].sub(p[0]).det(p[i+1].sub(p[0]));

if (dblcmp(tmp)==0)continue;

area+=tmp;

ret.x+=(p[0].x+p[i].x+p[i+1].x)/3\*tmp;

ret.y+=(p[0].y+p[i].y+p[i+1].y)/3\*tmp;

}

if (dblcmp(area))ret=ret.div(area);

return ret;

}

double areaintersection(polygon po)

{

}

double areaunion(polygon po)

{

return getarea()+po.getarea()-areaintersection(po);

}

double areacircle(circle c)

{

int i,j,k,l,m;

double ans=0;

for (i=0;i<n;i++)

{

int j=(i+1)%n;

if (dblcmp(p[j].sub(c.p).det(p[i].sub(c.p)))>=0)

{

ans+=c.areatriangle(p[i],p[j]);

}

else

{

ans-=c.areatriangle(p[i],p[j]);

}

}

return fabs(ans);

}

//多边形和圆关系

//0 一部分在圆外

//1 与圆某条边相切

//2 完全在圆内

int relationcircle(circle c)

{

getline();

int i,x=2;

if (relationpoint(c.p)!=1)return 0;

for (i=0;i<n;i++)

{

if (c.relationseg(l[i])==2)return 0;

if (c.relationseg(l[i])==1)x=1;

}

return x;

}

void find(int st,point tri[],circle &c)

{

if (!st)

{

c=circle(point(0,0),-2);

}

if (st==1)

{

c=circle(tri[0],0);

}

if (st==2)

{

c=circle(tri[0].add(tri[1]).div(2),tri[0].distance(tri[1])/2.0);

}

if (st==3)

{

c=circle(tri[0],tri[1],tri[2]);

}

}

void solve(int cur,int st,point tri[],circle &c)

{

find(st,tri,c);

if (st==3)return;

int i;

for (i=0;i<cur;i++)

{

if (dblcmp(p[i].distance(c.p)-c.r)>0)

{

tri[st]=p[i];

solve(i,st+1,tri,c);

}

}

}

circle mincircle()//点集最小圆覆盖

{

random\_shuffle(p,p+n);

point tri[4];

circle c;

solve(n,0,tri,c);

return c;

}

int circlecover(double r)//单位圆覆盖

{

int ans=0,i,j;

vector<pair<double,int> >v;

for (i=0;i<n;i++)

{

v.clear();

for (j=0;j<n;j++)if (i!=j)

{

point q=p[i].sub(p[j]);

double d=q.len();

if (dblcmp(d-2\*r)<=0)

{

double arg=atan2(q.y,q.x);

if (dblcmp(arg)<0)arg+=2\*pi;

double t=acos(d/(2\*r));

v.push\_back(make\_pair(arg-t+2\*pi,-1));

v.push\_back(make\_pair(arg+t+2\*pi,1));

}

}

sort(v.begin(),v.end());

int cur=0;

for (j=0;j<v.size();j++)

{

if (v[j].second==-1)++cur;

else --cur;

ans=max(ans,cur);

}

}

return ans+1;

}

int pointinpolygon(point q)//点在凸多边形内部的判定

{

if (getdir())reverse(p,p+n);

if (dblcmp(q.sub(p[0]).det(p[n-1].sub(p[0])))==0)

{

if (line(p[n-1],p[0]).pointonseg(q))return n-1;

return -1;

}

int low=1,high=n-2,mid;

while (low<=high)

{

mid=(low+high)>>1;

if (dblcmp(q.sub(p[0]).det(p[mid].sub(p[0])))>=0&&dblcmp(q.sub(p[0]).det(p[mid+1].sub(p[0])))<0)

{

polygon c;

c.p[0]=p[mid];

c.p[1]=p[mid+1];

c.p[2]=p[0];

c.n=3;

if (c.relationpoint(q))return mid;

return -1;

}

if (dblcmp(q.sub(p[0]).det(p[mid].sub(p[0])))>0)

{

low=mid+1;

}

else

{

high=mid-1;

}

}

return -1;

}

};

struct polygons

{

vector<polygon>p;

polygons()

{

p.clear();

}

void clear()

{

p.clear();

}

void push(polygon q)

{

if (dblcmp(q.getarea()))p.pb(q);

}

vector<pair<double,int> >e;

void ins(point s,point t,point X,int i)

{

double r=fabs(t.x-s.x)>eps?(X.x-s.x)/(t.x-s.x):(X.y-s.y)/(t.y-s.y);

r=min(r,1.0);r=max(r,0.0);

e.pb(mp(r,i));

}

double polyareaunion()

{

double ans=0.0;

int c0,c1,c2,i,j,k,w;

for (i=0;i<p.size();i++)

{

if (p[i].getdir()==0)reverse(p[i].p,p[i].p+p[i].n);

}

for (i=0;i<p.size();i++)

{

for (k=0;k<p[i].n;k++)

{

point &s=p[i].p[k],&t=p[i].p[(k+1)%p[i].n];

if (!dblcmp(s.det(t)))continue;

e.clear();

e.pb(mp(0.0,1));

e.pb(mp(1.0,-1));

for (j=0;j<p.size();j++)if (i!=j)

{

for (w=0;w<p[j].n;w++)

{

point a=p[j].p[w],b=p[j].p[(w+1)%p[j].n],c=p[j].p[(w-1+p[j].n)%p[j].n];

c0=dblcmp(t.sub(s).det(c.sub(s)));

c1=dblcmp(t.sub(s).det(a.sub(s)));

c2=dblcmp(t.sub(s).det(b.sub(s)));

if (c1\*c2<0)ins(s,t,line(s,t).crosspoint(line(a,b)),-c2);

else if (!c1&&c0\*c2<0)ins(s,t,a,-c2);

else if (!c1&&!c2)

{

int c3=dblcmp(t.sub(s).det(p[j].p[(w+2)%p[j].n].sub(s)));

int dp=dblcmp(t.sub(s).dot(b.sub(a)));

if (dp&&c0)ins(s,t,a,dp>0?c0\*((j>i)^(c0<0)):-(c0<0));

if (dp&&c3)ins(s,t,b,dp>0?-c3\*((j>i)^(c3<0)):c3<0);

}

}

}

sort(e.begin(),e.end());

int ct=0;

double tot=0.0,last;

for (j=0;j<e.size();j++)

{

if (ct==1)tot+=e[j].first-last;

ct+=e[j].second;

last=e[j].first;

}

ans+=s.det(t)\*tot;

}

}

return fabs(ans)\*0.5;

}

};

const int maxn=500;

struct circles

{

circle c[maxn];

double ans[maxn];//ans[i]表示被覆盖了i次的面积

double pre[maxn];

int n;

circles(){}

void add(circle cc)

{

c[n++]=cc;

}

bool inner(circle x,circle y)

{

if (x.relationcircle(y)!=1)return 0;

return dblcmp(x.r-y.r)<=0?1:0;

}

void init\_or()//圆的面积并去掉内含的圆

{

int i,j,k=0;

bool mark[maxn]={0};

for (i=0;i<n;i++)

{

for (j=0;j<n;j++)if (i!=j&&!mark[j])

{

if ((c[i]==c[j])||inner(c[i],c[j]))break;

}

if (j<n)mark[i]=1;

}

for (i=0;i<n;i++)if (!mark[i])c[k++]=c[i];

n=k;

}

void init\_and()//圆的面积交去掉内含的圆

{

int i,j,k=0;

bool mark[maxn]={0};

for (i=0;i<n;i++)

{

for (j=0;j<n;j++)if (i!=j&&!mark[j])

{

if ((c[i]==c[j])||inner(c[j],c[i]))break;

}

if (j<n)mark[i]=1;

}

for (i=0;i<n;i++)if (!mark[i])c[k++]=c[i];

n=k;

}

double areaarc(double th,double r)

{

return 0.5\*sqr(r)\*(th-sin(th));

}

void getarea()

{

int i,j,k;

memset(ans,0,sizeof(ans));

vector<pair<double,int> >v;

for (i=0;i<n;i++)

{

v.clear();

v.push\_back(make\_pair(-pi,1));

v.push\_back(make\_pair(pi,-1));

for (j=0;j<n;j++)if (i!=j)

{

point q=c[j].p.sub(c[i].p);

double ab=q.len(),ac=c[i].r,bc=c[j].r;

if (dblcmp(ab+ac-bc)<=0)

{

v.push\_back(make\_pair(-pi,1));

v.push\_back(make\_pair(pi,-1));

continue;

}

if (dblcmp(ab+bc-ac)<=0)continue;

if (dblcmp(ab-ac-bc)>0) continue;

double th=atan2(q.y,q.x),fai=acos((ac\*ac+ab\*ab-bc\*bc)/(2.0\*ac\*ab));

double a0=th-fai;

if (dblcmp(a0+pi)<0)a0+=2\*pi;

double a1=th+fai;

if (dblcmp(a1-pi)>0)a1-=2\*pi;

if (dblcmp(a0-a1)>0)

{

v.push\_back(make\_pair(a0,1));

v.push\_back(make\_pair(pi,-1));

v.push\_back(make\_pair(-pi,1));

v.push\_back(make\_pair(a1,-1));

}

else

{

v.push\_back(make\_pair(a0,1));

v.push\_back(make\_pair(a1,-1));

}

}

sort(v.begin(),v.end());

int cur=0;

for (j=0;j<v.size();j++)

{

if (cur&&dblcmp(v[j].first-pre[cur]))

{

ans[cur]+=areaarc(v[j].first-pre[cur],c[i].r);

ans[cur]+=0.5\*point(c[i].p.x+c[i].r\*cos(pre[cur]),c[i].p.y+c[i].r\*sin(pre[cur])).det(point(c[i].p.x+c[i].r\*cos(v[j].first),c[i].p.y+c[i].r\*sin(v[j].first)));

}

cur+=v[j].second;

pre[cur]=v[j].first;

}

}

for (i=1;i<=n;i++)

{

ans[i]-=ans[i+1];

}

}

};

struct halfplane:public line

{

double angle;

halfplane(){}

halfplane(point \_a,point \_b)

{

a=\_a;

b=\_b;

}

halfplane(line v)

{

a=v.a;

b=v.b;

}

void calcangle()

{

angle=atan2(b.y-a.y,b.x-a.x);

}

bool operator<(const halfplane &b)const

{

return angle<b.angle;

}

};

struct halfplanes

{

int n;

halfplane hp[maxp];

point p[maxp];

int que[maxp];

int st,ed;

void push(halfplane tmp)

{

hp[n++]=tmp;

}

void unique()

{

int m=1,i;

for (i=1;i<n;i++)

{

if (dblcmp(hp[i].angle-hp[i-1].angle))hp[m++]=hp[i];

else if (dblcmp(hp[m-1].b.sub(hp[m-1].a).det(hp[i].a.sub(hp[m-1].a))>0))hp[m-1]=hp[i];

}

n=m;

}

bool halfplaneinsert()

{

int i;

for (i=0;i<n;i++)hp[i].calcangle();

sort(hp,hp+n);

unique();

que[st=0]=0;

que[ed=1]=1;

p[1]=hp[0].crosspoint(hp[1]);

for (i=2;i<n;i++)

{

while (st<ed&&dblcmp((hp[i].b.sub(hp[i].a).det(p[ed].sub(hp[i].a))))<0)ed--;

while (st<ed&&dblcmp((hp[i].b.sub(hp[i].a).det(p[st+1].sub(hp[i].a))))<0)st++;

que[++ed]=i;

if (hp[i].parallel(hp[que[ed-1]]))return false;

p[ed]=hp[i].crosspoint(hp[que[ed-1]]);

}

while (st<ed&&dblcmp(hp[que[st]].b.sub(hp[que[st]].a).det(p[ed].sub(hp[que[st]].a)))<0)ed--;

while (st<ed&&dblcmp(hp[que[ed]].b.sub(hp[que[ed]].a).det(p[st+1].sub(hp[que[ed]].a)))<0)st++;

if (st+1>=ed)return false;

return true;

}

void getconvex(polygon &con)

{

p[st]=hp[que[st]].crosspoint(hp[que[ed]]);

con.n=ed-st+1;

int j=st,i=0;

for (;j<=ed;i++,j++)

{

con.p[i]=p[j];

}

}

};

struct point3

{

double x,y,z;

point3(){}

point3(double \_x,double \_y,double \_z):

x(\_x),y(\_y),z(\_z){};

void input()

{

scanf("%lf%lf%lf",&x,&y,&z);

}

void output()

{

printf("%.2lf %.2lf %.2lf",x,y,z);

}

bool operator==(point3 a)

{

return dblcmp(a.x-x)==0&&dblcmp(a.y-y)==0&&dblcmp(a.z-z)==0;

}

bool operator<(point3 a)const

{

return dblcmp(a.x-x)==0?dblcmp(y-a.y)==0?dblcmp(z-a.z)<0:y<a.y:x<a.x;

}

double len()

{

return sqrt(len2());

}

double len2()

{

return x\*x+y\*y+z\*z;

}

double distance(point3 p)

{

return sqrt((p.x-x)\*(p.x-x)+(p.y-y)\*(p.y-y)+(p.z-z)\*(p.z-z));

}

point3 add(point3 p)

{

return point3(x+p.x,y+p.y,z+p.z);

}

point3 sub(point3 p)

{

return point3(x-p.x,y-p.y,z-p.z);

}

point3 mul(double d)

{

return point3(x\*d,y\*d,z\*d);

}

point3 div(double d)

{

return point3(x/d,y/d,z/d);

}

double dot(point3 p)

{

return x\*p.x+y\*p.y+z\*p.z;

}

point3 det(point3 p)

{

return point3(y\*p.z-p.y\*z,p.x\*z-x\*p.z,x\*p.y-p.x\*y);

}

double rad(point3 a,point3 b)

{

point3 p=(\*this);

return acos(a.sub(p).dot(b.sub(p))/(a.distance(p)\*b.distance(p)));

}

point3 trunc(double r)

{

r/=len();

return point3(x\*r,y\*r,z\*r);

}

point3 rotate(point3 o,double r)

{

}

};

struct line3

{

point3 a,b;

line3(){}

line3(point3 \_a,point3 \_b)

{

a=\_a;

b=\_b;

}

bool operator==(line3 v)

{

return (a==v.a)&&(b==v.b);

}

void input()

{

a.input();

b.input();

}

double length()

{

return a.distance(b);

}

double dispointtoline(point3 p)

{

return b.sub(a).det(p.sub(a)).len()/a.distance(b);

}

double dispointtoseg(point3 p)

{

if (dblcmp(p.sub(b).dot(a.sub(b)))<0||dblcmp(p.sub(a).dot(b.sub(a)))<0)

{

return min(p.distance(a),p.distance(b));

}

return dispointtoline(p);

}

point3 lineprog(point3 p)

{

return a.add(b.sub(a).trunc(b.sub(a).dot(p.sub(a))/b.distance(a)));

}

point3 rotate(point3 p,double ang)//p绕此向量逆时针arg角度

{

if (dblcmp((p.sub(a).det(p.sub(b)).len()))==0)return p;

point3 f1=b.sub(a).det(p.sub(a));

point3 f2=b.sub(a).det(f1);

double len=fabs(a.sub(p).det(b.sub(p)).len()/a.distance(b));

f1=f1.trunc(len);f2=f2.trunc(len);

point3 h=p.add(f2);

point3 pp=h.add(f1);

return h.add((p.sub(h)).mul(cos(ang\*1.0))).add((pp.sub(h)).mul(sin(ang\*1.0)));

}

};

struct plane

{

point3 a,b,c,o;

plane(){}

plane(point3 \_a,point3 \_b,point3 \_c)

{

a=\_a;

b=\_b;

c=\_c;

o=pvec();

}

plane(double \_a,double \_b,double \_c,double \_d)

{

//ax+by+cz+d=0

o=point3(\_a,\_b,\_c);

if (dblcmp(\_a)!=0)

{

a=point3((-\_d-\_c-\_b)/\_a,1,1);

}

else if (dblcmp(\_b)!=0)

{

a=point3(1,(-\_d-\_c-\_a)/\_b,1);

}

else if (dblcmp(\_c)!=0)

{

a=point3(1,1,(-\_d-\_a-\_b)/\_c);

}

}

void input()

{

a.input();

b.input();

c.input();

o=pvec();

}

point3 pvec()

{

return b.sub(a).det(c.sub(a));

}

bool pointonplane(point3 p)//点是否在平面上

{

return dblcmp(p.sub(a).dot(o))==0;

}

double angleplane(plane f)//两平面夹角

{

return acos(o.dot(f.o)/(o.len()\*f.o.len()));

}

double dispoint(point3 p)//点到平面距离

{

return fabs(p.sub(a).dot(o)/o.len());

}

point3 pttoplane(point3 p)//点到平面最近点

{

line3 u=line3(p,p.add(o));

crossline(u,p);

return p;

}

int crossline(line3 u,point3 &p)//平面和直线的交点

{

double x=o.dot(u.b.sub(a));

double y=o.dot(u.a.sub(a));

double d=x-y;

if (dblcmp(fabs(d))==0)return 0;

p=u.a.mul(x).sub(u.b.mul(y)).div(d);

return 1;

}

int crossplane(plane f,line3 &u)//平面和平面的交线

{

point3 oo=o.det(f.o);

point3 v=o.det(oo);

double d=fabs(f.o.dot(v));

if (dblcmp(d)==0)return 0;

point3 q=a.add(v.mul(f.o.dot(f.a.sub(a))/d));

u=line3(q,q.add(oo));

return 1;

}

};

polygons ps;

int main()

{

int i,j,k;

int n;

scanf("%d",&n);

double ans=0;

for (i=0;i<n;i++)

{

polygon pl;

pl.n=4;

pl.input();

ans+=pl.getarea();

ps.push(pl);

}

printf("%.12lf\n",ans/ps.polyareaunion());

return 0;

}