Chapter 4 Geometry



Keshuai

2015-11-07

Copyright ：1.0

目录

一．二维几何

（一）基础

1、点/向量

2、线/线段

3、圆

4、多边形

（二）面积交

1、圆与圆/多边形

2、扫描法：求多个图形的面积交X

例题：hdu 4629（多个三角形面积并）

二．三维几何

（一）基础

1、点/向量

2、线/线段

3、三角形/四面体

4、球

三．几何算法

（一）凸包

1、二维凸包

（1）点集面积和：计算给出点集组成的任意多边形面积和

例题：Zoj 3871（给定点集）

例题：Hdu 5448（给定凸包）

2、三维凸包

（二）旋转卡壳X

（三）半平面交

（四）仿射矩阵：旋转问题X

（五）运动规划X

例题：uva 1017（轮廓线）

一、二维几何

（一）基础

1、点/向量：

|  |  |
| --- | --- |
| getDot() | 向量点积 |
| getCroess() | 向量叉积 |
| getLength() | 向量长度 |
| getPowLength() | 向量长度的平方 |
| getAngle() | 向量角度，返回弧度制 |
| rotate() | 旋转角度，传入弧度制 |
| getNormal() | 获得向量的单位法向量 |
| torad() | 角度转换成弧度 |

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Point2.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const double pi = 4 \* atan(1);

const double eps = 1e-8;

inline int dcmp (double x) { if (fabs(x) < eps) return 0; else return x < 0 ? -1 : 1; }

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("%lf %lf", x, y); }

bool operator == (const Point& u) const { return dcmp(x - u.x) == 0 && dcmp(y - u.y) == 0; }

bool operator != (const Point& u) const { return !(\*this == u); }

bool operator < (const Point& u) const { return dcmp(x - u.x) < 0 || (dcmp(x-u.x)==0 && dcmp(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; } // 叉积

double operator & (const Point& u) { return x \* u.x + y \* u.y; } // 点积

};

typedef Point Vector;

/\* 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 getPowLength (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); }

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

2、线/线段：

|  |  |
| --- | --- |
| getLine() | 获得直线方程式 |
| getIntersection() | 获得两条直线交点 |
| getDistanceToLine() | 点到直线距离 |
| getDistanceToSegment() | 点到线段距离 |
| getPointToLine() | 点在直线上的投影 |
| haveIntersection() | 判断两线段是否有交点 |
| onSegment() | 判断点是否在线段上 |

需要Point2.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Line2.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct Line {

double a, b, c;

Line (double a = 0, double b = 0, double c = 0): a(a), b(b), c(c) {}

};

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 (dcmp(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)); }

/\* 点p到线段ab的距离 \*/

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 (dcmp(getDot(v1, v2)) < 0) return getLength(v2);

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

}

/\* 判断点是否在线段上 \*/

bool onSegment (Point p, Point a, Point b) { return dcmp(getCross(a-p, b-p)) == 0 && dcmp(getDot(a-p, b-p)) < 0; }

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

3、圆：

|  |  |
| --- | --- |
| getLineCircleIntersection() | 求直线与圆的交点 |
| getCircleCircleIntersection() | 求圆与圆的交点 |
| getInscribedCircle() | 三点确定内切圆（需要getDistanceToLine函数） |
| getCircumscribedCircle() | 三点确定外切圆 |
| getTangentsPointToCircle() | 过定点做圆的切线 |
| getTangentsCircleToCircle() | 求圆与圆的切线 |
| getTangPointsPointToCircle() | 过定点做圆切线求切点 |
| getTangPointsCircleToCircle() | 圆与圆切线求切点 |

需要Point2.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Circle.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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 area (double rad) { return rad \* r \* r / 2; }

};

/\* 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 (dcmp(delta) < 0) return 0;

if (dcmp(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 (dcmp(d) == 0) {

if (dcmp(o1.r - o2.r) == 0) return -1;

return 0;

}

if (dcmp(o1.r + o2.r - d) < 0) return 0;

if (dcmp(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;

}

/\* 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));

}

/\* 过定点作圆的切线 \*/

int getTangentsPointToCircle (Point p, Circle o, Vector\* v) {

Vector u = o.o - p;

double d = getLength(u);

if (d < o.r) return 0;

else if (dcmp(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 getTangentsCircleToCircle (Circle o1, Circle o2, Point\* a, Point\* b) {

int cnt = 0;

if (o1.r < o2.r) { swap(o1, o2); swap(a, b); }

double d2 = getLength(o1.o - o2.o); d2 = d2 \* d2;

double rdif = o1.r - o2.r, rsum = o1.r + o2.r;

if (d2 < rdif \* rdif) return 0;

if (dcmp(d2) == 0 && dcmp(o1.r - o2.r) == 0) return -1;

double base = getAngle(o2.o - o1.o);

if (dcmp(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 (dcmp(d2 - rsum \* rsum) == 0) {

a[cnt] = o1.point(base); b[cnt] = o2.point(base); cnt++;

} else if (d2 > rsum \* rsum) {

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;

}

/\* 获得点与圆的切点 \*/

void getTangPointsPointToCircle(Point p, Circle c, Point& a, Point& b) {

double k = 2 \* (c.r\*c.r + c.o.x\*p.x + c.o.y\*p.y - c.o.x\*c.o.x - c.o.y\*c.o.y);

double s = 2 \* (p.x-c.o.x), t = 2 \* (p.y - c.o.y);

if (dcmp(t) == 0) {

a.x = b.x = k / s;

double tmp = sqrt(c.r \* c.r - (c.o.x-a.x)\*(c.o.x-a.x));

a.y = c.o.y + tmp;

b.y = c.o.y - tmp;

} else {

double x = c.o.y - k/t;

double A = 1 + (s/t) \* (s/t);

double B = 2 \* (x\*s/t - c.o.x);

double C = c.o.x \* c.o.x + x \* x - c.r \* c.r;

double tmp = sqrt(B \* B - 4 \* A \* C);

a.x = (tmp - B) / 2 / A;

b.x = (-tmp - B) / 2 / A;

a.y = (k - s \* a.x) / t;

b.y = (k - s \* b.x) / t;

}

}

/\* 圆与圆的切点 \*/

void getTangPointsCircleToCircle (Circle c1, Circle c2, Point& a, Point& b) {

double x0 = c2.o.x - c1.o.x, y0 = c2.o.y - c1.o.y, cs, sn, rdis = c1.r - c2.r;

if (dcmp(y0) == 0) {

cs = rdis / x0;

sn = sqrt(1 - cs \* cs);

a = Point(c1.r \* cs + c1.o.x, c1.r \* sn + c1.o.y);

b = Point(c1.r \* cs + c1.o.x, c1.r \* (-sn) + c1.o.y);

} else {

double A = (x0/y0)\*(x0/y0) + 1;

double B = -2 \* x0 \* rdis / y0 / y0;

double C = (rdis / y0) \* (rdis / y0) - 1;

double delta = sqrt(B \* B - 4 \* A \* C);

cs = (-B + delta) / 2 / A;

sn = (rdis - x0 \* cs) / y0;

a = Point(c1.r \* cs + c1.o.x, c1.r \* sn + c1.o.y);

cs = (-B - delta) / 2 / A;

sn = (rdis - x0 \* cs) / y0;

b = Point(c1.r \* cs + c1.o.x, c1.r \* sn + c1.o.y);

}

}

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

4、多边形：

|  |  |
| --- | --- |
| getAngle() | 三边确定角度：余弦定理 |
| getArea() | 获得三角形的面积 |
| getDirArea() | 获得三角形的有向面积 |
| getPolygonArea() | 获得多边形的面积 |
| isPointInPolygon() | 判断点是否在多边形内 |
| simplify() | 删除多边形的共线点 |
| cutPolygon() | 切割多边形 |

需要Point2.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Polygon.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 余弦定理: 三边确定一角 \*/

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; }

typedef vector<Point> Polygon;

double getPolygonalgetArea (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 ret/2;

}

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 = dcmp(getCross(p[j] - p[i], o-p[i]));

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

int d2 = dcmp(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;

}

/\* 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 (dcmp((b-a)^(c-b)) != 0 && (ret.size() == 0 || b != ret[ret.size()-1]))

ret.push\_back(b);

}

return ret;

}

/\* 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 (dcmp((b-a)^(c-a)) >= 0) ret.push\_back(c);

if (dcmp((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;

}

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

（二）面积交

1、圆与圆/多边形：

|  |  |
| --- | --- |
| getPublicAreaCircleToCircle() | 圆与圆的面积交 |
| getPublicAreaCircleToTriangle() | 圆与三角形的面积交，其中三角形一顶点为圆心 |
| getPublicAreaCircleToPolugon() | 圆与多边形的面积交 |

需要Point2.cpp、Line2.cpp、Circle.cpp、Polygon.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PublicArea.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 圆与圆面积交 \*/

double getPublicAreaCircleToCircle(Circle a, Circle b) {

double dis = getLength(a.o - b.o);

if (dcmp(dis-a.r-b.r) >= 0) return 0;

if (dis <= fabs(a.r - b.r)) { return min(a.getArea(2\*pi), b.getArea(2\*pi)); }

double ang1 = getAngle(a.r, dis, b.r);

double ang2 = getAngle(b.r, dis, a.r);

double ret = ang1 \* a.r \* a.r + ang2 \* b.r \* b.r - dis \* a.r \* sin(ang1);

return ret;

}

/\* 三角形一顶点为圆心 \*/

double getPublicAreaCircleToTriangle (Circle O, Point a, Point b) {

if (dcmp((a-O.o)^(b-O.o)) == 0) return 0;

int sig = 1;

double da = getPowLength(O.o-a), db = getPowLength(O.o-b);

if (dcmp(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 (dcmp(da-O.r\*O.r) <= 0) {

if (dcmp(db-O.r\*O.r) <= 0) return getDirArea(O.o, a, b) \* sig;

int k = 0;

if (getPowLength(sol[0]-b) > getPowLength(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 \* dcmp(tmp);

}

double d = getDistanceToSegment(O.o, a, b);

if (dcmp(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 \* dcmp(tmp);

}

double k1 = O.r / getLength(a - O.o), k2 = O.r / getLength(b - O.o);

Point p = O.o + (a - O.o) \* k1, q = O.o + (b - O.o) \* k2;

double ret1 = O.getArea(getAngle(p-O.o, q-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 \* dcmp(tmp);

}

/\* 多边形和圆的面积交 \*/

double getPublicAreaCircleToPolygon (Circle O, Point\* p, int n) {

if (dcmp(O.r) == 0) return 0;

double area = 0;

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

int u = (i + 1) % n;

area += getPublicAreaCircleToTriangle(O, p[i], p[u]);

}

return fabs(area);

}

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

2、扫描法：

例题：hdu 4629（多个三角形面积并）

题目大意：给定n个三角形，求覆盖k次的面积

解题思路：离散化

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu4629.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

const double eps = 1e-10;

inline int dcmp(double e) { if (fabs(e) < eps) return 0; return e < 0 ? -1 : 1; }

struct Point {

double x, y;

void read() { scanf("%lf%lf", &x, &y); }

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

Point operator < (const Point& u) const { return x < u.x || (dcmp(x - u.x) == 0 && y < u.y); }

Point operator + (const Point& u) const { return Point(x + u.x, y + u.y); }

Point operator - (const Point& u) const { return Point(x - u.x, y - u.y); }

Point operator \* (const double& k) const { return Point(x \* k, y \* k); }

Point operator / (const double& k) const { return Point(x / k, y / k); }

};

typedef Point Vector;

double getDot(Vector a, Vector b) { return a.x \* b.x + a.y \* b.y; }

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

double getLength(Vector a) { return sqrt(getDot(a, a)); }

Vector rotate(Vector v, double rad) { return Vector(v.x \* cos(rad) - v.y \* sin(rad), v.x \* sin(rad) + v.y \* cos(rad)); }

double getArea(Point a, Point b, Point c) { return getCross(b - a, c - a); }

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

}

/\* 直线pv和直线qw的交点 \*/

bool getIntersection (Point p, Vector v, Point q, Vector w, Point& o) {

if (dcmp(getCross(v, w)) == 0) return false;

Vector u = p - q;

double k = getCross(w, u) / getCross(v, w);

o = p + v \* k;

return true;

}

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

const int maxn = 55;

const int maxm = 1e5 + 5;

typedef pair<double,int> pdi;

int N, M;

double ans[maxn], len[maxn], event[maxm];

Point P[maxn][4];

pdi T[maxn<<1];

bool checkVertical() {

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

for (int j = 0; j < 3; j++)

if (dcmp(P[i][j].x - P[i][j+1].x) == 0) return true;

}

return false;

}

void adjust() {

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

for (int j = 0; j < 4; j++)

P[i][j] = rotate(P[i][j], 0.001);

}

}

void init () {

int n = N = M = 0;

scanf("%d", &n);

while (n--) {

for (int i = 0; i < 3; i++) P[N][i].read();

P[N][3] = P[N][0];

if (dcmp(getArea(P[N][0], P[N][1], P[N][2]))) N++;

}

while (checkVertical()) adjust();

}

inline bool onSegment(Point p, Point a, Point b) { return dcmp(getDot(a - p, b - p)) <= 0;}

bool verticalPos(Point l, Point r, int id, double& in, double& ot) {

int c = 0;

double y[3];

Point tmp;

//printf("%lf %lf %lf %lf\n", l.x, l.y, r.x, r.y);

for (int i = 0; i < 3; i++) {

getIntersection(l, r-l, P[id][i], P[id][i+1]-P[id][i], tmp);

if (onSegment(tmp, P[id][i], P[id][i+1]))

y[c++] = tmp.y;

}

sort(y, y + c);

if (c <= 1) return false;

in = y[0], ot = y[c-1];

return true;

}

void calculate (double x) {

int c = 0;

double in, ot;

Point a = Point(x, 0), b = Point(x, 1);

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

if (verticalPos(a, b, i, in, ot)) {

T[c++] = make\_pair(in, 1);

T[c++] = make\_pair(ot, -1);

}

}

sort(T, T + c);

int mv = 0;

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

if (mv > 0) len[mv] += T[i].first - T[i-1].first;

mv += T[i].second;

}

}

void getPosition() {

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

for (int ki = 0; ki < 3; ki++) {

Point a = P[i][ki], b = P[i][ki+1];

event[M++] = a.x;

for (int j = i + 1; j < N; j++) {

for (int kj = 0; kj < 3; kj++) {

Point c = P[j][kj], d = P[j][kj+1], e;

if (haveIntersection(a, b, c, d)) {

getIntersection(a, b-a, c, d-c, e);

event[M++] = e.x;

}

}

}

}

}

sort(event, event + M);

M = unique(event, event + M) - event;

}

void solve() {

getPosition();

memset(ans, 0, sizeof(ans));

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

memset(len, 0, sizeof(len));

calculate(event[i-1] - eps);

calculate(event[i] + eps);

double x = event[i] - event[i-1];

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

ans[j] += len[j] \* x / 2;

}

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

printf("%.4lf\n", ans[i]);

}

int main () {

int cas;

scanf("%d", &cas);

while (cas--) {

init();

solve();

}

return 0;

}

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

二、三维几何

（一）基础

1、点/向量：

|  |  |
| --- | --- |
| getDot() | 向量点积 |
| getCroess() | 向量叉积 |
| getLength() | 向量长度 |
| getAngle() | 向量角度，返回弧度制 |
| getNormal() | 获得向量的单位法向量 |
| getDistancePointToPlane() | 点到平面的距离 |
| getPlaneProjection() | 点在平面上的投影 |

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Point3.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

inline int dcmp (double x) { if (fabs(x) < eps) return 0; else return x < 0 ? -1 : 1; }

struct Point3 {

double x, y, z;

Point3 (double x = 0, double y = 0, double z = 0): x(x), y(y), z(z) {}

void read () { scanf("%lf%lf%lf", &x, &y, &z); }

bool operator < (const Point3& u) const { return dcmp(x-u.x)<0 || (dcmp(x-u.x)==0 && dcmp(y-u.y)<0) || (dcmp(x-u.x)==0 && dcmp(y-u.y)==0 && dcmp(z-u.z) < 0); }

bool operator > (const Point3& u) const { return u < (\*this); }

bool operator == (const Point3& u) const { return !(u < (\*this) || (\*this) < u); }

bool operator != (const Point3& u) const { return !((\*this) == u); }

bool operator <= (const Point3& u) const { return \*this < u || \*this == u; }

bool operator >= (const Point3& u) const { return \*this > u || \*this == u; }

Point3 operator + (const Point3& u) const { return Point3(x+u.x, y+u.y, z+u.z); }

Point3 operator - (const Point3& u) const { return Point3(x-u.x, y-u.y, z-u.z); }

Point3 operator \* (const double u) const { return Point3(x\*u, y\*u, z\*u); }

Point3 operator / (const double u) const { return Point3(x/u, y/u, z/u); }

};

typedef Point3 Vector3;

double getDot(Vector3 a, Vector3 b) { return a.x\*b.x + a.y\*b.y + a.z\*b.z; }

Vector3 getCross (Vector3 a, Vector3 b) { return Vector3(a.y\*b.z-a.z\*b.y, a.z\*b.x-a.x\*b.z, a.x\*b.y-a.y\*b.x); }

double getLength(Vector3 a) { return sqrt(getDot(a, a)); }

double getAngle(Vector3 a, Vector3 b) { return acos(getDot(a, b) / getLength(a) / getLength(b)); }

Vector3 getNormal(Point3 a, Point3 b, Point3 c) {

Vector3 u = a-b, v = b-c;

Vector3 k = getCross(u, v);

return k / getLength(k);

}

double getDistancePointToPlane (Point3 p, Point3 p0, Vector3 v) { return fabs(getDot(p-p0, v)); }

Point3 getPlaneProjection (Point3 p, Point3 p0, Vector3 v) { return p - v \* getDot(p-p0, v); }

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

2、线/线段：

|  |  |
| --- | --- |
| getDistancePointToLine() | 空间点到直线距离 |
| getDistancePointToSegment() | 空间点到线段距离 |
| getPointLineToLine() | 异面直线公垂线 |
| getDistanceLineToLine() | 异面直线距离 |
| getDistanceSegmentToSegment() | 异面线段距离 |

需要Point3.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Line3.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 空间点到直线的距离 \*/

double getDistancePointToLine(Point3 p, Point3 a, Point3 b) {

Vector3 v1 = b-a, v2 = p-a;

return getLength(getCross(v1,v2)) / getLength(v1);

}

/\* 空间点到线段的距离 \*/

double getDistancePointToSegment(Point3 p, Point3 a, Point3 b) {

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

Vector3 v1 = b-a, v2 = p-a, v3 = p-b;

if (dcmp(getDot(v1, v2)) < 0) return getLength(v2);

else if (dcmp(getDot(v1, v3)) > 0) return getLength(v3);

else return getLength(getCross(v1, v2)) / getLength(v1);

}

/\* 异面直线公垂线 \*/

bool getPointLineToLine (Point3 a, Vector3 u, Point3 b, Vector3 v, double& s) {

double p = getDot(u, u) \* getDot(v, v) - getDot(u, v) \* getDot(u, v);

if (dcmp(p) == 0) return false;

double q = getDot(u, v) \* getDot(v, a-b) - getDot(v, v) \* getDot(u, a-b);

s = p/q;

return true;

}

/\* 异面直线的距离 \*/

double getDistanceLineToLine (Point3 a, Vector3 u, Point3 b, Vector3 v) {

Vector3 p = getCross(u, v);

return fabs(getDot(p, (a-b)) / getLength(p));

}

/\* 异面线段距离 \*/

double getDistanceSegmentToSegment(Point3 a, Point3 b, Point3 c, Point3 d) {

double s, t;

bool flag1 = getPointLineToLine(a, b-a, c, d-c, s);

bool flag2 = getPointLineToLine(c, d-c, a, b-a, t);

if (flag1 && flag2 && dcmp(s) > 0 && dcmp(s - 1) < 0 && dcmp(t) > 0 && dcmp(t-1) < 0) {

Vector3 u = b-a, v = d-c;

Point3 p = a + u \* s, q = b + v \* t;

return getLength(p-q);

} else {

double ans = 1e20;

ans = min(ans, getDistancePointToSegment(a, c, d));

ans = min(ans, getDistancePointToSegment(b, c, d));

ans = min(ans, getDistancePointToSegment(c, a, b));

ans = min(ans, getDistancePointToSegment(d, a, b));

return ans;

}

}

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

3、三角形/四面体：

|  |  |
| --- | --- |
| getArea() | 获得三角形面积 |
| onTriangle() | 判断点是否在三角形内 |
| haveIntersectionTirSeg() | 线段与三角形交点 |
| getVolume() | 四面体体积（6倍） |

需要Point3.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Triangle3.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 三角形面积 \*/

double getArea (Point3 a, Point3 b, Point3 c) { return getLength(getCross(b-a, c-a)); }

/\* 判断点是否在三角形内 \*/

bool onTriangle (Point3 p, Point3 a, Point3 b, Point3 c) {

double area1 = getArea(p, a, b);

double area2 = getArea(p, b, c);

double area3 = getArea(p, c, a);

return dcmp(area1 + area2 + area3 - getArea(a, b, c)) == 0;

}

/\* 三角形与线段交点 \*/

bool haveIntersectionTriSeg (Point3 p0, Point3 p1, Point3 p2, Point3 a, Point3 b, Point3& p) {

Vector3 v = getCross(p1-p0, p2-p0);

if (dcmp(getDot(v, b-a)) == 0) return false;

else {

double t = getDot(v, p0 - a) / getDot(v, b-a);

if (dcmp(t) < 0 || dcmp(t-2) > 0) return false;

p = a + (b-a) \* t;

return onTriangle(p, p0, p1, p2);

}

}

/\* 有向体积，是4边形的6倍 \*/

double getVolume (Point3 a, Point3 b, Point3 c, Point3 d) { return fabs(getDot(d-a, getCross(b-a, c-a))/6); } /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

4、球：

|  |  |
| --- | --- |
| getBallVolume() | 球体积 |
| getBallArea() | 球表面积 |
| getBallLength() | 球上两点的弧长 |

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Ball.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double getBallVolume (double r) { return pi \* 4 \* pow(r, 3) / 3; }

double getBallArea (double r) { return pi \* 4 \* pow(r, 2); }

double getBallLength (double lat1, double lot1, double lat2, double lot2, double r) {

double dlot = lot2 - lot1;

double dlat = lat2 - lat1;

double a = pow(sin(dlat/2), 2) + cos(lat1)\*cos(lat2)\*pow(sin(dlot/2), 2);

double c = 2 \* atan2(sqrt(a), sqrt(1-a));

return c \* r;

}

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

三、几何算法

（一）凸包

1、二维凸包：

需要Point2.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ConvexHull2.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 凸包 \*/

int getConvexHull (Point\* p, int n, Point\* ch) {

sort(p, p + n);

int m = 0;

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

/\* 可共线 \*/

//while (m > 1 && dcmp(getCross(ch[m-1]-ch[m-2], p[i]-ch[m-1])) < 0) m--;

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

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

}

int k = m;

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

/\* 可共线 \*/

//while (m > k && dcmp(getCross(ch[m-1]-ch[m-2], p[i]-ch[m-2])) < 0) m--;

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

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

}

if (n > 1) m--;

return m;

}

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

1. 点集面积和：计算给出点集组成的任意多边形面积和

例题：zoj 3871（给定点集）

题目大意：给定一个点集，计算任意点组成的多形的面积和

解题思路： 枚举每条边，计算该条边的贡献值。因为给定集合非凸包，所以对于每个点，要预处理其它点与它的极角，并按照极角排序，然后枚举另一点，则这条边即为当前枚举边，计算有多少个点在其右侧，即可计算。算法复杂度为o(n^2)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu4760.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int mod = 998244353;

int N, pow2[maxn];

Point P[maxn], T[maxn \* 2];

inline bool cmp(const Point& a, const Point& b) { return a.ang < b.ang; }

int main () {

pow2[0] = 1;

for (int i = 1; i < maxn; i++) pow2[i] = pow2[i-1] \* 2 % mod;

for (int i = 0; i < maxn; i++) pow2[i] = (pow2[i] - 1 + mod) % mod;

int cas;

scanf("%d", &cas);

while (cas--) {

scanf("%d", &N);

for (int i = 0; i < N; i++) P[i].read();

ll ans = 0;

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

int sz = 0;

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

T[sz] = P[j];

T[sz++].ang = atan2(P[j].y-P[i].y, P[j].x-P[i].x);

}

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

T[j+sz] = T[j];

T[j+sz].ang += pi \* 2;

}

sort(T, T + sz \* 2, cmp);

int r = 0;

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

while (T[r+1].ang - T[j].ang < pi) r++;

ans = (ans + getArea(Point(0, 0), P[i], T[j]) % mod \* pow2[r-j]) % mod;

}

}

printf("%lld\n", ans);

}

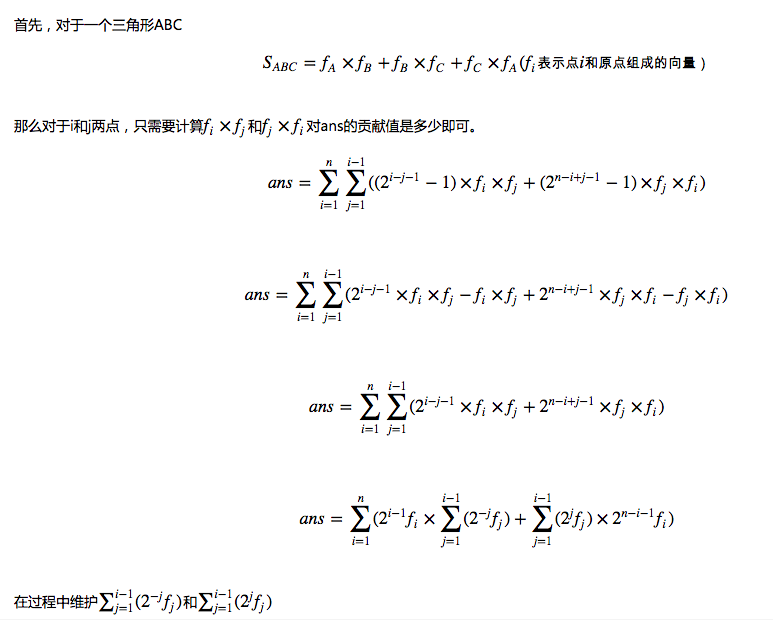
return 0;

}

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

例题：hdu 5448（给定凸包）

题目大意：给定一个点集，点集为凸包，计算任意点组成的多形的面积和

解题思路：算法复杂度o(n) 

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu5448.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef long long ll;

const int maxn = 100005;

const int mod = 1e9 + 7;

struct Point {

int x, y;

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

void read () { scanf("%d%d", &x, &y); }

int operator \* (const Point& u) { return (1LL \* x \* u.y % mod - 1LL \* y \* u.x % mod + mod) % mod; }

Point operator \* (const int u) { return Point(1LL \* u \* x % mod, 1LL \* u \* y % mod); }

Point operator + (const Point& u) { return Point((x+u.x)%mod, (y+u.y)%mod); }

};

int N, mul[maxn], inv[maxn];

int mul\_mod(ll x, int n, int mod) {

int ret = 1;

while (n) {

if (n&1) ret = x \* ret % mod;

x = x \* x % mod;

n >>= 1;

}

return ret;

}

int main () {

mul[0] = inv[0] = 1;

for (int i = 1; i < maxn; i++) mul[i] = 2 \* mul[i-1] % mod;

ll inv2 = mul\_mod(2, mod-2, mod);

for (int i = 1; i < maxn; i++) inv[i] = inv2 \* inv[i-1] % mod;

int cas;

scanf("%d", &cas);

while (cas--) {

int ans = 0;

scanf("%d", &N);

Point fi, fg, fh;

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

fi.read();

ans = (ans + 1LL \* mul[i-1] \* (fi \* fh) % mod) % mod;

ans = (ans + 1LL \* (i==N?inv2:mul[N-i-1]) \* (fg \* fi) % mod) % mod;

fg = fg + fi \* mul[i];

fh = fh + fi \* inv[i];

}

printf("%d\n", ans);

}

return 0;

}

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

2、三维凸包：

需要Point3.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ConvexHull3.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct Face {

int v[3];

Face (int a = 0, int b = 0, int c = 0) { v[0] = a, v[1] = b, v[2] = c;}

Vector3 normal (Point3 \*p) const { return getCross(p[v[1]] - p[v[0]], p[v[2]]-p[v[0]]); }

int cansee (Point3 \*p, int i) const {

return getDot(p[i]-p[v[0]], normal(p)) > 0 ? 1 : 0;

}

};

int vis[1005][1005];

double rand01() { return rand() / (double) RAND\_MAX; }

double randeps() { return (rand01() - 0.5) \* eps; }

Point3 addNoise(Point3 p) { return Point3(p.x+randeps(), p.y+randeps(), p.z+randeps()); }

vector<Face> CH3D (Point3 \*o, int n, Point3\* p) {

for (int i = 0; i < n; i++) p[i] = addNoise(o[i]);

memset(vis, -1, sizeof(vis));

vector<Face> cur;

cur.push\_back(Face(0, 1, 2));

cur.push\_back(Face(2, 1, 0));

for (int i = 3; i < n; i++) {

vector<Face> net;

for (int j = 0; j < cur.size(); j++) {

Face& f = cur[j];

int res = f.cansee(p, i);

if (!res) net.push\_back(f);

for (int k = 0; k < 3; k++) vis[f.v[k]][f.v[(k+1)%3]] = res;

}

for (int j = 0; j < cur.size(); j++) {

for (int k = 0; k < 3; k++) {

int a = cur[j].v[k], b = cur[j].v[(k+1)%3];

if (vis[a][b] != vis[b][a] && vis[a][b])

net.push\_back(Face(a, b, i));

}

}

cur = net;

}

return cur;

}

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

（二）旋转卡壳

需要Point2.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*rotatingCalipers.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 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));

}

}

/\* 求包含凸包点集的最小矩形，分面积和周长 \*/

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 (dcmp(getDot(v, p[r%n]-p[i]) - getDot(v, p[(r+1)%n]-p[i])) < 0) r++;

while (j < r || dcmp(getCross(v, p[j%n]-p[i]) - getCross(v,p[(j+1)%n]-p[i])) < 0) j++;

while (l < j || dcmp(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);

}

}

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

（三）半平面交：线性规划

需要Point2.cpp 和Line3.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*HalfPlane.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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; }

};

bool onLeft(DirLine l, Point p) { return dcmp(l.v ^ (p-l.p)) >= 0; }

/\* 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 (dcmp(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;

}

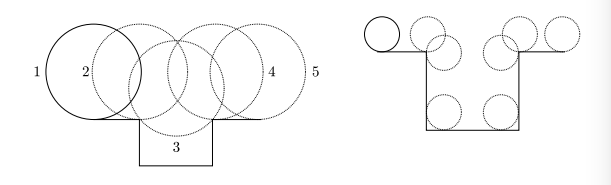
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（四）放射矩阵

（五）运动规划

例题：uva 1017（轮廓线）

题目大意：求圆心经过的路线，给定折线的点的方式为与前一个点的相对关系



解题思路：将所有点依次连接起来形成一条曲线，圆心移动的轨迹其实就是一条时刻与它距离为r的曲线。

对于线段就是平移，对于点就是一个圆。要求的轨迹其实就是所有线段和圆的轮廓，所以从起始位置开始，每次暴力出下一要移动到的点，距离就是最终的和。

对于从圆上的点A移动到线段或是圆上的点B，角AOB（O为圆心）要尽量小。

对于从线段上的点A移动到线段或是圆上的点B，AB的距离要尽量小。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*uva1017.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <vector>

#include <algorithm>

using namespace std;

const double pi = 4 \* atan(1);

const double eps = 1e-8;

inline 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) {}

void read () { scanf("%lf%lf", &x, &y); }

bool operator == (const Point& u) const { return dcmp(x - u.x) == 0 && dcmp(y - u.y) == 0; }

bool operator != (const Point& u) const { return !(\*this == u); }

bool operator < (const Point& u) const { return x < u.x || (x == u.x && y < u.y); }

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;

double getDot (Vector a, Vector b) { return a.x \* b.x + a.y \* b.y; }

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 getAngle (Vector u) { return atan2(u.y, u.x); }

double getAngle (Vector a, Vector b) { return acos(getDot(a, b) / getLength(a) / getLength(b)); }

/\* 直线pv和直线qw的交点 \*/

bool getIntersection (Point p, Vector v, Point q, Vector w, Point& o) {

if (dcmp(getCross(v, w)) == 0) return false;

Vector u = p - q;

double k = getCross(w, u) / getCross(v, w);

o = p + v \* k;

return true;

}

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

/\* 加等号为可为端点 \*/

}

bool onSegment (Point p, Point a, Point b) {

/\* 可否在两端 \*/

if (p == a || p == b) return true;

return dcmp(getCross(a-p, b-p)) == 0 && dcmp(getDot(a-p, b-p)) < 0;

}

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; }

};

/\* 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 (dcmp(delta) < 0) return 0;

if (dcmp(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 (dcmp(d) == 0) {

if (dcmp(o1.r - o2.r) == 0) return -1;

return 0;

}

if (dcmp(o1.r + o2.r - d) < 0) return 0;

if (dcmp(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;

}

const int maxn = 205;

const double inf = 0x3f3f3f3f3f3f3f;

struct Segment {

Point s, e;

Segment () {};

Segment (Point s, Point e): s(s), e(e) {}

};

double R;

int N, M, idx[maxn], nS, nC;

Point Q[maxn], P[maxn];

Segment Gseg[maxn];

Circle Gcir[maxn];

void init () {

M = 0;

nS = nC = 1;

for (int i = 0; i < N; i++) Q[i].read();

int n = 0;

P[n++] = Point(0, 0);

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

if (dcmp(Q[i].x))

P[n] = Point(P[n-1].x + Q[i].x, P[n-1].y), n++;

if (dcmp(Q[i].y))

P[n] = Point(P[n-1].x, P[n-1].y + Q[i].y), n++;

}

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

if (i) {

if (dcmp(P[i-1].x - P[i].x) == 0) {

if (dcmp(P[i-1].y - P[i].y) < 0)

Gseg[nS] = Segment(Point(P[i-1].x - R, P[i-1].y), Point(P[i].x - R, P[i].y));

else

Gseg[nS] = Segment(Point(P[i-1].x + R, P[i-1].y), Point(P[i].x + R, P[i].y));

} else

Gseg[nS] = Segment(Point(P[i-1].x, P[i-1].y + R), Point(P[i].x, P[i].y + R));

idx[M++] = -nS, nS++;

}

Gcir[nC] = Circle(P[i], R);

idx[M++] = nC++;

}

}

void handle (int u, int v, Point& p, int& idx, int cur, Point o) {

Point t = Point(inf , inf);

double k1, k2;

vector<Point> sol;

if (u < 0 && v < 0) {

u = -u, v = -v;

if (haveIntersection(Gseg[u].s, Gseg[u].e, Gseg[v].s, Gseg[v].e)) {

getIntersection(Gseg[u].s, Gseg[u].e-Gseg[u].s, Gseg[v].s, Gseg[v].e-Gseg[v].s, t);

double k1 = getLength(t - Gseg[u].s) / getLength(Gseg[u].e-Gseg[u].s);

double k2 = getLength(p - Gseg[u].s) / getLength(Gseg[u].e-Gseg[u].s);

if (idx == 0 || dcmp(k1 - k2) < 0 || (dcmp(k1-k2) == 0 && idx < cur))

idx = cur, p = t;

}

} else if (u < 0 && v > 0) {

u = -u;

int n = getLineCircleIntersection(Gseg[u].s, Gseg[u].e, Gcir[v], k1, k2, sol);

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

if (onSegment(sol[i], Gseg[u].s, Gseg[u].e)) {

double k1 = getLength(sol[i] - Gseg[u].s) / getLength(Gseg[u].e-Gseg[u].s);

double k2 = getLength(t - Gseg[u].s) / getLength(Gseg[u].e-Gseg[u].s);

if (dcmp(k1 - k2) < 0) t = sol[i];

}

}

double k1 = getLength(t - Gseg[u].s) / getLength(Gseg[u].e-Gseg[u].s);

double k2 = getLength(p - Gseg[u].s) / getLength(Gseg[u].e-Gseg[u].s);

if (idx == 0 || dcmp(k1 - k2) < 0 || (dcmp(k1-k2) == 0 && idx < cur))

idx = cur, p = t;

} else if (u > 0 && v < 0) {

v = -v;

double rad = inf;

int n = getLineCircleIntersection(Gseg[v].s, Gseg[v].e, Gcir[u], k1, k2, sol);

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

if (onSegment(sol[i], Gseg[v].s, Gseg[v].e)) {

double tmp = (o == sol[i] ? 0 : getAngle(o-Gcir[u].o, sol[i]-Gcir[u].o));

if (dcmp((o-Gcir[u].o) \* (sol[i]-Gcir[u].o)) > 0) tmp = 2 \* pi - tmp;

if (dcmp(rad - tmp) >= 0)

rad = tmp, t = sol[i];

}

}

double k = (o == p ? 0 : getAngle(o-Gcir[u].o, p-Gcir[u].o));

if (dcmp((o-Gcir[u].o) \* (p-Gcir[u].o)) > 0) k = 2 \* pi - k;

if (idx == 0 || dcmp(rad - k) < 0 || (dcmp(rad-k) == 0 && idx < cur))

idx = cur, p = t;

} else if (u > 0 && v > 0) {

double rad = inf;

int n = getCircleCircleIntersection (Gcir[u], Gcir[v], sol);

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

double tmp = (o == sol[i] ? 0 : getAngle(o-Gcir[u].o, sol[i]-Gcir[u].o));

if (dcmp((o-Gcir[u].o) \* (sol[i]-Gcir[u].o)) > 0) tmp = 2 \* pi - tmp;

if (dcmp(rad - tmp) >= 0)

rad = tmp, t = sol[i];

}

double k = (o == p ? 0 : getAngle(o-Gcir[u].o, p-Gcir[u].o));

if (dcmp((o-Gcir[u].o) \* (p-Gcir[u].o)) > 0) k = 2 \* pi - k;

if (idx == 0 || dcmp(rad - k) < 0 || (dcmp(rad-k) == 0 && idx < cur))

idx = cur, p = t;

}

}

double solve () {

int mv = 0;

double ans = 0, rad = 0;

Point s = P[0] + Point(0, R);

while (mv + 1 < M) {

int re = 0;

Point e;

for (int i = mv + 1; i < M; i++)

handle(idx[mv], idx[i], e, re, i, s);

if (idx[mv] > 0) {

int u = idx[mv];

double tmp = getAngle(s-Gcir[u].o, e-Gcir[u].o);

if (dcmp((s-Gcir[u].o) \* (e-Gcir[u].o)) > 0) tmp = 2 \* pi - tmp;

rad += tmp;

} else if (dcmp(s.x - e.x) == 0)

ans += fabs(e.y - s.y);

else if (dcmp(s.y - e.y) == 0)

ans += fabs(e.x - s.x);

s = e, mv = re;

}

return ans + rad \* R;

}

int main () {

int cas = 1;

while (scanf("%lf%d", &R, &N) == 2) {

if (R == 0 && N == 0) break;

init ();

printf("Case %d: Distance = %.3lf\n\n", cas++, solve());

}

return 0;

}

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