计算几何 0921 补

目录

[旋转卡壳\_平面点集最大三角形 1](#_Toc304385373)

[旋转卡壳\_最小面积覆盖矩形 4](#_Toc304385374)

[旋转卡壳\_凸多边形内公切线 10](#_Toc304385375)

[圆与多边形面积交 17](#_Toc304385376)

[角的扫描线\_简单多边形点光源照亮面积 NlogN 20](#_Toc304385377)

[点光源线遮挡可见线数 25](#_Toc304385378)

旋转卡壳\_平面点集最大三角形

**const** **double** eps = 1e-8;

**int** sig(**double** t)

{

**return** (t > eps) - (t < -eps);

}

**const** **int** maxn = 50010;

**struct** Point

{

**double** x, y;

**int** id;

Point()

{

}

Point(**double** p, **double** q) :

x(p), y(q)

{

}

**double** cross(**const** Point& p) **const**

{

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

}

**double** dist(**const** Point& p) **const**

{

**double** a = x - p.x;

**double** b = y - p.y;

**return** sqrt(a \* a + b \* b);

}

Point **operator**-(**const** Point& p) **const**

{

**return** Point(x - p.x, y - p.y);

}

**bool** **operator**<(**const** Point& p) **const**

{

**const** Point zero(0, 0);

**double** t = **this**->cross(p);

**if** (sig(t))

**return** sig(t) > 0;

**return** sig(**this**->dist(zero) - p.dist(zero)) > 0;

}

};

Point point[maxn], stack[2 \* maxn];

**int** N, top;

**void** convexHull()

{

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

{

**if** (sig(point[1].y - point[i].y) > 0 || (sig(point[1].y - point[i].y)

== 0 && sig(point[1].x - point[i].x) > 0))

swap(point[1], point[i]);

}

**for** (**int** i = 2; i <= N; i++)

point[i] = point[i] - point[1];

sort(point + 2, point + 1 + N);

**int** n = N;

N = 1;

**const** Point zero(0, 0);

point[1] = zero;

**for** (**int** i = 2; i <= n;)

{

Point current = point[i];

point[++N] = current;

**for** (; i <= n && !sig(current.cross(point[i])); i++)

;

}

top = 0;

stack[top++] = point[1];

stack[top++] = point[2];

**for** (**int** i = 3; i <= N; i++)

{

**while** (top >= 2 && sig((stack[top - 1] - stack[top - 2]).cross(point[i]

- stack[top - 2])) < 0)

top--;

stack[top++] = point[i];

}

}

**double** S(**const** Point& p, **const** Point& q, **const** Point& r)

{

**return** fabs(p.cross(q) + q.cross(r) + r.cross(p)) / 2;

}

**double** S(**int** p, **int** q, **int** r)

{

**return** S(stack[p], stack[q], stack[r]);

}

**void** init()

{

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

{

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

}

convexHull();

**for** (**int** i = 0; i < top; i++)

{

stack[i + top] = stack[i];

}

}

**void** work()

{

**double** result = 0;

**int** j, k;

**for** (**int** i = 0; i < top; i++)

{

**for** (**int** j = i + 1, k = i + 2; j < i + top - 1; j++)

{

**for** (; k != i + top && sig(S(i, j, k) - S(i, j, k + 1)) <= 0; k++)

;

result = max(S(i, j, k), result);

}

}

printf("%.2lf\n", result);

}

**int** main()

{

scanf("%d", &N);

**while** (N != -1)

{

init();

work();

scanf("%d", &N);

}

**return** 0;

}

旋转卡壳\_最小面积覆盖矩形

**const** **double** eps = 1e-8;

**const** **int** maxn = 1010;

**const** **double** PI = acos(-1.0);

**int** sig(**double** t)

{

**return** (t > eps) - (t < -eps);

}

**struct** Point

{

**double** x, y;

Point(){}

Point(**double** p, **double** q) : x(p), y(q) {}

**double** cross(**const** Point& p) **const**

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

**double** dot(**const** Point& p) **const**

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

**double** dist(**const** Point& p) **const**

{

**double** a = x - p.x;

**double** b = y - p.y;

**return** sqrt(a \* a + b \* b);

}

Point **operator**-(**const** Point& p) **const**

{**return** Point(x - p.x, y - p.y);}

Point **operator**+(**const** Point& p) **const**

{**return** Point(x + p.x, y + p.y);}

**int** id;

**bool** **operator**<(**const** Point& p) **const**

{

**const** Point zero(0, 0);

**double** t = **this**->cross(p);

**if** (sig(t))

**return** sig(t) > 0;

**return** sig(**this**->dist(zero) - p.dist(zero)) > 0;

}

Point trans() **const**

{

**return** Point(-y, x);

}

};

Point getSol(Point a, Point b, Point c, Point d)

{

**double** s = (d.y - b.y) \* (b.x - a.x) \* (d.x - c.x) + (b.y - a.y) \* b.x

\* (d.x - c.x) - (d.y - c.y) \* (b.x - a.x) \* d.x;

**double** t = (b.y - a.y) \* (d.x - c.x) - (d.y - c.y) \* (b.x - a.x);

**double** x = s / t;

**double** s1 = (d.x - b.x) \* (b.y - a.y) \* (d.y - c.y) + (b.x - a.x) \* b.y

\* (d.y - c.y) - (d.x - c.x) \* (b.y - a.y) \* d.y;

**double** t1 = (b.x - a.x) \* (d.y - c.y) - (d.x - c.x) \* (b.y - a.y);

**double** y = s1 / t1;

Point res(x, y);

**return** res;

}

**int** N;

**struct** Poly

{

Point point[maxn], stack[2 \* maxn];

**int** N, top;

**void** init()

{

N = 0;

}

**void** add(**const** Point& p)

{

point[++N] = p;

}

**void** convexHull()

{

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

{

**if** (sig(point[1].y - point[i].y) > 0 || (sig(point[1].y

- point[i].y) == 0 && sig(point[1].x - point[i].x) > 0))

swap(point[1], point[i]);

}

**for** (**int** i = 2; i <= N; i++)

point[i] = point[i] - point[1];

sort(point + 2, point + 1 + N);

**int** n = N;

N = 1;

**const** Point zero(0, 0);

point[1] = zero;

**for** (**int** i = 2; i <= n;)

{

Point current = point[i];

point[++N] = current;

**for** (; i <= n && !sig(current.cross(point[i])); i++)

;

}

top = 0;

stack[top++] = point[1];

stack[top++] = point[2];

**for** (**int** i = 3; i <= N; i++)

{

**while** (top >= 2 && sig((stack[top - 1] - stack[top - 2]).cross(

point[i] - stack[top - 2])) < 0)

top--;

stack[top++] = point[i];

}

}

**void** finish()

{

**for** (**int** i = 0; i < top; i++)

{

stack[i + top] = stack[i];

}

}

};

Poly poly;

**const** Point zero(0, 0);

**struct** Line

{

Point a, b;

**int** current;

**double** cosValue() **const**

{

Point p = poly.stack[current + 1] - poly.stack[current];

Point q = b - a;

**return** p.dot(q) / p.dist(zero) / q.dist(zero);

}

**void** adjust()

{

**if** (sig((b - a).cross(poly.stack[current + 1] - poly.stack[current]))

== 0)

current++;

}

};

**void** init()

{

poly.init();

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

{

Point tmp;

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

poly.add(tmp);

}

poly.convexHull();

poly.finish();

}

Line line[4];

**double** S()

{

Point point[5];

**for** (**int** i = 0; i < 4; i++)

{

**int** next = (i + 1) % 4;

Point p1 = poly.stack[line[i].current];

Point p2 = p1 + line[i].b - line[i].a;

Point p3 = poly.stack[line[next].current];

Point p4 = p3 + line[next].b - line[next].a;

point[i] = getSol(p1, p2, p3, p4);

}

point[4] = point[0];

**double** res = 0;

**for** (**int** i = 0; i < 4; i++)

{

res += point[i].cross(point[i + 1]);

}

res = fabs(res);

**return** res;

}

**void** buildLine()

{

//bottom ymin

line[0].a = poly.stack[0];

**for** (**int** i = 0; i < poly.top; i++)

{

**if** (line[0].a.y >= poly.stack[i].y)

{

line[0].a = poly.stack[i];

line[0].current = i;

}

}

line[0].b = line[0].a;

line[0].b.x += 1;

//right xmax

line[1].a = poly.stack[0];

**for** (**int** i = 0; i < poly.top; i++)

{

**if** (line[1].a.x <= poly.stack[i].x)

{

line[1].a = poly.stack[i];

line[1].current = i;

}

}

line[1].b = line[1].a;

line[1].b.y += 1;

//up ymax

line[2].a = poly.stack[0];

**for** (**int** i = 0; i < poly.top; i++)

{

**if** (line[2].a.y <= poly.stack[i].y)

{

line[2].a = poly.stack[i];

line[2].current = i;

}

}

line[2].b = line[2].a;

line[2].b.x -= 1;

//left xmin

line[3].a = poly.stack[0];

**for** (**int** i = 0; i < poly.top; i++)

{

**if** (line[3].a.x >= poly.stack[i].x)

{

line[3].a = poly.stack[i];

line[3].current = i;

}

}

line[3].b = line[3].a;

line[3].b.y -= 1;

}

**void** work()

{

**if** (poly.top <= 2)

{

puts("0.0000");

**return**;

}

**double** result = 1e15;

buildLine();

**do**

{

result = min(result, S());

**double** arc[4];

**for** (**int** i = 0; i < 4; i++)

{

arc[i] = line[i].cosValue();

}

**double** mx = arc[0];

**int** idx = 0;

**for** (**int** i = 1; i < 4; i++)

{

**if** (mx < arc[i])

{

mx = arc[i];

idx = i;

}

}

Point vec = poly.stack[line[idx].current + 1]

- poly.stack[line[idx].current];

**for** (**int** i = idx, j = 0; j < 4; i = (i + 1) % 4, j++)

{

line[i].b = line[i].a + vec;

line[i].adjust();

vec = vec.trans();

}

} **while** (sig(atan2(line[0].b.y - line[0].a.y, line[0].b.x - line[0].a.x)

- PI / 2) <= 0);

printf("%.4lf\n", result / 2);

}

旋转卡壳\_凸多边形内公切线

**const** **int** maxn = 10010;

**const** **double** eps = 1e-8;

**const** **double** PI = acos(-1.0);

**int** sig(**double** t)

{

**return** (t > eps) - (t < -eps);

}

//-----------------------------此处点类同上----------------------------

**const** Point zero(0, 0);

Point getSol(Point a, Point b, Point c, Point d)

{

**double** s = (d.y - b.y) \* (b.x - a.x) \* (d.x - c.x) + (b.y - a.y) \* b.x

\* (d.x - c.x) - (d.y - c.y) \* (b.x - a.x) \* d.x;

**double** t = (b.y - a.y) \* (d.x - c.x) - (d.y - c.y) \* (b.x - a.x);

**double** x = s / t;

**double** s1 = (d.x - b.x) \* (b.y - a.y) \* (d.y - c.y) + (b.x - a.x) \* b.y

\* (d.y - c.y) - (d.x - c.x) \* (b.y - a.y) \* d.y;

**double** t1 = (b.x - a.x) \* (d.y - c.y) - (d.x - c.x) \* (b.y - a.y);

**double** y = s1 / t1;

Point res(x, y);

**return** res;

}

**struct** Poly

{

Point point[2 \* maxn];

**int** N;

**void** init()

{

N = 0;

}

**void** add(**const** Point& p)

{

point[N++] = p;

}

**void** finish()

{

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

{

point[i + N] = point[i];

}

}

};

**struct** Line

{

Point a, b;

Point \*arr;

**int** current;

**double** rad, dis;

**double** cosValue()

{

Point t = arr[current + 1] - arr[current];

Point s = b - a;

**return** t.dot(s) / t.dist(zero) / s.dist(zero);

}

**void** build()

{

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

**if** (sig(rad + PI) <= 0)

rad = PI;

dis = a.dist(b);

}

**bool** **operator**<(**const** Line& p) **const**

{

**if** (sig(rad - p.rad))

**return** sig(rad - p.rad) < 0;

**return** sig(dis - p.dis) > 0;

}

};

Poly P, Q;

Line lp, lq;

**int** N, M;

**void** init()

{

P.init();

Q.init();

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

{

Point tmp;

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

P.add(tmp);

}

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

{

Point tmp;

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

Q.add(tmp);

}

P.finish();

Q.finish();

lp.arr = P.point;

lq.arr = Q.point;

}

**void** buildLine()

{

//ymin

lp.a = lp.arr[0];

lp.current = 0;

**for** (**int** i = 0; i < P.N; i++)

{

**if** (lp.a.y > P.point[i].y || (lp.a.y == P.point[i].y && lp.a.x

> P.point[i].x))

{

lp.a = P.point[i];

lp.current = i;

}

}

lp.b = lp.a;

lp.b.x -= 1;

//ymax

lq.a = lq.arr[0];

lq.current = 0;

**for** (**int** i = 0; i < Q.N; i++)

{

**if** (lq.a.y < Q.point[i].y || (lq.a.y == Q.point[i].y && lq.a.x

< Q.point[i].x))

{

lq.a = Q.point[i];

lq.current = i;

}

}

lq.b = lq.a;

lq.b.x += 1;

}

**int** same(**const** Point& a, **const** Point& b, **const** Line& l)

{

Point t = l.b - l.a;

Point s = b - l.a;

Point v = a - l.a;

**int** p = sig(t.cross(s));

**int** q = sig(t.cross(v));

**if** (p \* q >= 0)

{

**if** (p == 0 && q == 0)

**return** 0;

**if** (p)

**return** p;

**return** q;

}

**return** 0;

}

Line result[maxn \* 6];

**int** R;

Point tmp[maxn \* 2];

**int** C;

**void** record(**int** pidx, **int** qidx)

{

Line tmp;

tmp.a = P.point[pidx];

tmp.b = Q.point[qidx];

**int** pre = (pidx - 1 + P.N) % P.N;

**int** next = (pidx + 1) % P.N;

**int** t = same(P.point[pre], P.point[next], tmp);

**if** (!t)

**return**;

pre = (qidx - 1 + Q.N) % Q.N;

next = (qidx + 1) % Q.N;

**int** s = same(Q.point[pre], Q.point[next], tmp);

**if** (!s)

**return**;

**if** (t == s)

**return**;

result[R].a = P.point[pidx];

result[R].b = Q.point[qidx];

result[R].current = pidx;

R++;

}

**int** isRight()

{

**for** (**int** i = 0; i < C; i++)

{

**int** pre = (i - 1 + C) % C;

**int** next = (i + 1) % C;

Point t = tmp[i] - tmp[pre];

Point s = tmp[next] - tmp[i];

**if** (sig(t.cross(s)) > 0)

**return** 0;

}

**return** 1;

}

**void** work()

{

R = 0;

buildLine();

**int** initP = lp.current;

**int** initQ = lq.current;

//record(lp.arr[lp.current],lq.arr[lq.current]);

**do**

{

**double** p = lp.cosValue();

**double** q = lq.cosValue();

**int** t = sig(p - q);

**if** (t > 0)

{

Point x = lp.arr[(lp.current + 1) % P.N] - lp.arr[lp.current];

lp.b = lp.a + x;

lp.current = (lp.current + 1) % P.N;

lq.b = lq.a - x;

record(lp.current, lq.current);

}

**else** **if** (t < 0)

{

Point x = lq.arr[(lq.current + 1) % Q.N] - lq.arr[lq.current];

lq.b = lq.a + x;

lq.current = (lq.current + 1) % Q.N;

lp.b = lp.a - x;

record(lp.current, lq.current);

}

**else**

{

Point x = lq.arr[(lq.current + 1) % Q.N] - lq.arr[lq.current];

lq.b = lq.a + x;

lq.current = (lq.current + 1) % Q.N;

lp.b = lp.a - x;

lp.current = (lp.current + 1) % P.N;

record(lp.current, lq.current);

record((lp.current - 1 + P.N) % P.N, lq.current);

record(lp.current, (lq.current - 1 + Q.N) % Q.N);

}

} **while** (lp.current != initP || lq.current != initQ);

**for** (**int** i = 0; i < R; i++)

result[i].build();

sort(result, result + R);

**int** l;

**for** (l = 0; l < R;)

{

**double** current = result[l].rad;

**for** (; l < R && sig(current - result[l].rad) == 0; l++)

;

**break**;

}

swap(result[1], result[l]);

C = 0;

tmp[C++] = getSol(result[0].a, result[0].b, result[1].a, result[1].b);

tmp[C++] = result[0].a;

**for** (**int** i = (result[0].current + 1) % P.N; i != result[1].current; i = (i

+ 1) % P.N)

{

tmp[C++] = lp.arr[i];

}

tmp[C++] = lp.arr[result[1].current];

**if** (!isRight())

swap(result[0], result[1]);

//first slope && first

**if** (sig(result[0].a.x - result[0].b.x) == 0)

puts("VERTICAL");

**else**

{

**double** res = (result[0].b.y - result[0].a.y) / (result[0].b.x

- result[0].a.x);

**char** o[32];

sprintf(o, "%.3lf", res);

**if** (strcmp(o, "-0.000") == 0)

puts("0.000");

**else**

printf("%.3lf\n", res);

}

printf("%.0lf %.0lf\n", lp.arr[result[0].current].x,

lp.arr[result[0].current].y);

**for** (**int** i = (result[0].current + 1) % P.N; i != result[1].current; i = (i

+ 1) % P.N)

{

printf("%.0lf %.0lf\n", lp.arr[i].x, lp.arr[i].y);

}

//last && last slope

**if** (result[0].current != result[1].current)

printf("%.0lf %.0lf\n", lp.arr[result[1].current].x,

lp.arr[result[1].current].y);

**if** (sig(result[1].a.x - result[1].b.x) == 0)

puts("VERTICAL");

**else**

{

**double** res = (result[1].b.y - result[1].a.y) / (result[1].b.x

- result[1].a.x);

**char** o[32];

sprintf(o, "%.3lf", res);

**if** (strcmp(o, "-0.000") == 0)

puts("0.000");

**else**

printf("%.3lf\n", res);

//printf("%.3lf\n",res+0.000);

}

}

圆与多边形面积交

**struct** Point

{

**double** x, y;

**Point**() {}

**Point**(**double** p, **double** q) : x(p), y(q){}

**double** **cross**(**const** Point& p) **const**

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

**double** **dot**(**const** Point& p) **const**

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

Point **operator+**(**const** Point& p) **const**

{**return** Point(x + p.x, y + p.y);}

Point **operator-**(**const** Point& p) **const**

{**return** Point(x - p.x, y - p.y);}

**double** **dist**(**const** Point& p) **const**

{

**double** a = x - p.x;

**double** b = y - p.y;

**return** **sqrt**(a \* a + b \* b);

}

**bool** **operator==**(**const** Point& p) **const**

{**return** sig(x - p.x) == 0 && sig(y - p.y) == 0;}

**double** **dist**(**const** Point& p, **const** Point& q)

{

**double** s = **this**->cross(p) + p.cross(q) + q.cross(\***this**);

s = **fabs**(s);

**return** s / p.dist(q);

}

};

**const** Point zero(0, 0);

**struct** Poly

{

Point point[5010];

**int** cnt;

**void** **init**()

{

cnt = 0;

}

**void** **addPoint**(**const** Point& p)

{

point[cnt++] = p;

}

**void** **finish**()

{

point[cnt] = point[0];

}

};

complex<**double**> **ei**(**double** theta)

{

complex<**double**> r(0, 1);

**return** exp(theta \* r);

}

Point **trans**(**const** Point& p, **double** theta, **double** fac)

{

complex<**double**> r(p.x, p.y);

r = r \* ei(theta) \* fac;

**return** Point(r.real(), r.imag());

}

Poly poly;

**struct** Circle

{

Point center;

**double** r;

**int** **intersec**(**const** Point& p, **const** Point& q)

{

**return** sig(center.dist(p, q) - r);

}

**void** **intersec**(**const** Point& p, **const** Point& q, Point &r1, Point &r2)

{

**double** d = center.dist(p, q);

**double** dis = (zero - p).dot(q - p) / p.dist(q);

Point D = trans(q - p, 0, dis / p.dist(q)) + p;

Point t = trans(q - p, 0, **sqrt**(r \* r - d \* d) / p.dist(q));

r1 = D + t;

r2 = D - t;

}

**int** **intLine**(**const** Point& p, **const** Point& q)

{

**double** t1 = (q - p).dot(center - p);

**double** t2 = (p - q).dot(center - q);

**return** sig(center.dist(p, q) - r) < 0 && sig(t1) >= 0 && sig(t2) >= 0;

}

};

Circle circle;

**int** **isin**(**const** Point& p)

{

**return** sig(circle.center.dist(p) - circle.r) <= 0;

}

**int** **onSeg**(**const** Point& p, **const** Point& a, **const** Point& b)

{

**return** sig((a - p).dot(b - p)) < 0;

}

**double** **getS**(**const** Point& from, **const** Point& to)

{

**if** (!sig(from.cross(to)))

**return** 0.0;

Point t1, t2;

circle.intersec(from, to, t1, t2);

Point pnt[10];

**int** cnt = 0;

pnt[cnt++] = from;

**if** (onSeg(t1, from, to))

pnt[cnt++] = t1;

**if** (onSeg(t2, from, to))

pnt[cnt++] = t2;

pnt[cnt++] = to;

**if** (cnt == 4 && sig((pnt[2] - pnt[1]).dot(pnt[1] - pnt[0])) < 0)

swap(pnt[1], pnt[2]);

**double** res = 0;

**for** (**int** i = 0; i < cnt - 1; i++)

{

Point a = pnt[i];

Point b = pnt[i + 1];

**double** theta = fixacos(a.dot(b) / a.dist(zero) / b.dist(zero));

**if** (!isin(pnt[i]) || !isin(pnt[i + 1]))

{

res += circle.r \* circle.r \* theta;

}

**else**

res += **fabs**(pnt[i].cross(pnt[i + 1]));

}

**return** res;

}

**double** **interS**()

{

**double** s = 0;

**for** (**int** i = 0; i < poly.cnt; i++)

{

**int** sign = sig(poly.point[i].cross(poly.point[i + 1]));

s += getS(poly.point[i], poly.point[i + 1]) \* sign;

}

s = **fabs**(s);

**return** s / 2;

}

角的扫描线\_简单多边形点光源照亮面积 NlogN

**const** **double** eps = 1e-8;

**const** **int** maxn = 50010;

**int** **sig**(**double** t)

{

**return** (t > eps) - (t < -eps);

}

**struct** Point

{

**double** x, y;

**Point**() {}

**Point**(**double** p, **double** q) : x(p), y(q) {}

**double** **dot**(**const** Point& p) **const**

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

**double** **cross**(**const** Point& p) **const**

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

**double** **dist**(**const** Point& p) **const**

{

**double** a = x - p.x;

**double** b = y - p.y;

**return** a \* a + b \* b;

}

Point **operator-**(**const** Point& p) **const**

{**return** Point(x - p.x, y - p.y);}

Point **operator+**(**const** Point& p) **const**

{**return** Point(x + p.x, y + p.y);}

Point **operator/**(**double** p) **const**

{**return** Point(x / p, y / p);}

**bool** **operator==**(**const** Point& p) **const**

{**return** sig(x - p.x) == 0 && sig(y - p.y) == 0;}

};

Point **getSol**(Point a, Point b, Point c, Point d)

{

**double** s = (d.y - b.y) \* (b.x - a.x) \* (d.x - c.x) + (b.y - a.y) \* b.x

\* (d.x - c.x) - (d.y - c.y) \* (b.x - a.x) \* d.x;

**double** t = (b.y - a.y) \* (d.x - c.x) - (d.y - c.y) \* (b.x - a.x);

**double** x = s / t;

**double** s1 = (d.x - b.x) \* (b.y - a.y) \* (d.y - c.y) + (b.x - a.x) \* b.y

\* (d.y - c.y) - (d.x - c.x) \* (b.y - a.y) \* d.y;

**double** t1 = (b.x - a.x) \* (d.y - c.y) - (d.x - c.x) \* (b.y - a.y);

**double** y = s1 / t1;

Point res(x, y);

**return** res;

}

Point lastVisit, visit, center;

**struct** Segment

{

Point A, B;

**int** index;

**void** **adjust**()

{

**if** (sig(A.cross(B)) < 0)

swap(A, B);

}

**double** **getValue**() **const**

{

Point pnt = getSol(center, visit, A, B);

**double** p = (A - pnt).dot(B - pnt);

**double** q = (pnt - center).dot(visit - center);

//if (sig(p)>0 || sig(q)<0) return 1e15;

**return** center.dist(pnt);

}

**bool** **operator<**(**const** Segment& p) **const**

{

**double** d0 = getValue();

**double** d1 = p.getValue();

**return** sig(d0 - d1) < 0;

}

};

**struct** Event

{

**double** value;

Point point;

**int** in;

**int** belong;

**bool** **operator<**(**const** Event& p) **const**

{

**return** sig(value - p.value) < 0;

}

};

Segment segment[maxn];

Point point[maxn + 1];

Event event[3 \* maxn];

vector<Event> eventGroup[3 \* maxn];

set<Segment> S;

**int** N, E;

**double** **getS**(**const** Point& p, **const** Point& q)

{

**return** **fabs**(p.cross(q));

}

**void** **init**()

{

**scanf**("%lf%lf", &center.x, &center.y);

**scanf**("%d", &N);

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

{

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

}

point[N + 1] = point[1];

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

{

segment[i].A = point[i];

segment[i].B = point[i + 1];

segment[i].A = segment[i].A - center;

segment[i].B = segment[i].B - center;

segment[i].adjust();

}

center = Point(0, 0);

**int** n = N;

N = 0;

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

{

**if** (sig(segment[i].A.cross(segment[i].B)) == 0)

**continue**;

segment[++N] = segment[i];

segment[N].index = N;

}

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

{

event[E].value = **atan2**(segment[i].A.y, segment[i].A.x);

event[E].point = segment[i].A;

event[E].in = 1;

event[E++].belong = i;

event[E].value = **atan2**(segment[i].B.y, segment[i].B.x);

event[E].point = segment[i].B;

event[E].in = 0;

event[E++].belong = i;

}

sort(event, event + E);

**int** e = E;

E = 0;

**for** (**int** i = 0; i < e;)

{

Event current = event[i];

**for** (; i < e && sig(current.value - event[i].value) == 0; i++)

{

eventGroup[E].push\_back(event[i]);

}

E++;

}

eventGroup[E++] = eventGroup[0];

eventGroup[E] = eventGroup[1];

}

**void** **print**()

{

**for** (set<Segment>::iterator ite = S.begin(); ite != S.end(); ite++)

{

**printf**("%d\n", ite->index);

}

}

**void** **work**()

{

**double** result = 0;

visit = (eventGroup[0][0].point + eventGroup[E - 2][0].point) / 2;

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

{

**double** p = segment[i].A.cross(visit);

**double** q = visit.cross(segment[i].B);

**if** (sig(p) > 0 && sig(q) > 0)

{

S.insert(segment[i]);

}

}

**for** (**int** i = 0; i < E; i++)

{

visit = (eventGroup[i][0].point

+ eventGroup[i == 0 ? (E - 2) : (i - 1)][0].point) / 2;

set<Segment>::iterator ite = S.begin();

Point t = getSol(center, eventGroup[i][0].point, ite->A, ite->B);

**if** (i)

result += getS(lastVisit, t);

**for** (vector<Event>::iterator ite = eventGroup[i].begin(); ite

!= eventGroup[i].end(); ite++)

{

**if** (!ite->in)

{

set<Segment>::iterator iter = S.find(segment[ite->belong]);

**if** (iter == S.end())

**continue**;

S.erase(iter);

}

}

visit = (eventGroup[i][0].point + eventGroup[i + 1][0].point) / 2;

**for** (vector<Event>::iterator ite = eventGroup[i].begin(); ite

!= eventGroup[i].end(); ite++)

{

**if** (ite->in)

{

S.insert(segment[ite->belong]);

}

}

ite = S.begin();

lastVisit = getSol(center, eventGroup[i][0].point, ite->A, ite->B);

}

result /= 2;

**printf**("%.2lf\n", result);

}

点光源线遮挡可见线数

**const** **double** eps = 1e-8;

**const** **int** maxn = 10010;

**int** **sig**(**double** t)

{

**return** (t > eps) - (t < -eps);

}

**//--------------------------以下通用点类-------------------------------**

Point **getSol**(Point a, Point b, Point c, Point d)

{

**double** s = (d.y - b.y) \* (b.x - a.x) \* (d.x - c.x) + (b.y - a.y) \* b.x

\* (d.x - c.x) - (d.y - c.y) \* (b.x - a.x) \* d.x;

**double** t = (b.y - a.y) \* (d.x - c.x) - (d.y - c.y) \* (b.x - a.x);

**double** x = s / t;

**double** s1 = (d.x - b.x) \* (b.y - a.y) \* (d.y - c.y) + (b.x - a.x) \* b.y

\* (d.y - c.y) - (d.x - c.x) \* (b.y - a.y) \* d.y;

**double** t1 = (b.x - a.x) \* (d.y - c.y) - (d.x - c.x) \* (b.y - a.y);

**double** y = s1 / t1;

Point res(x, y);

**return** res;

}

Point center, visit;

**struct** Segment

{

Point A, B;

**int** index;

**void** **adjust**()

{

**if** (sig(A.cross(B)) < 0)

{

swap(A, B);

}

}

**bool** **operator<**(**const** Segment& p) **const**

{

Point pnt0 = getSol(center, visit, A, B);

Point pnt1 = getSol(center, visit, p.A, p.B);

**double** d0 = pnt0.dist(center);

**double** d1 = pnt1.dist(center);

**return** sig(d0 - d1) < 0;

}

};

**struct** Event

{

**double** value;

Point point;

**int** belong;

**int** in;

**bool** **operator<**(**const** Event& p) **const**

{

**return** sig(value - p.value) < 0;

}

};

Segment segment[maxn];

Event event[5 \* maxn];

**int** N, E;

set<Segment> S;

**void** **init**()

{

**scanf**("%lf%lf", ¢er.x, ¢er.y);

E = 0;

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

{

**scanf**("%lf%lf%lf%lf", &segment[i].A.x, &segment[i].A.y,

&segment[i].B.x, &segment[i].B.y);

segment[i].A = segment[i].A - center;

segment[i].B = segment[i].B - center;

segment[i].index = i;

}

center = Point(0, 0);

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

{

segment[i].adjust();

event[E].value = **atan2**(segment[i].A.y, segment[i].A.x);

event[E].point = segment[i].A;

event[E].in = 1;

event[E++].belong = i;

event[E].value = **atan2**(segment[i].B.y, segment[i].B.x);

event[E].point = segment[i].B;

event[E].in = 0;

event[E++].belong = i;

}

sort(event, event + E);

S.clear();

}

**int** result[maxn];

**void** **work**()

{

set<Segment>::iterator ite;

**memset**(result, 0, **sizeof**(result));

visit = event[0].point;

**if** (!event[0].in)

S.insert(segment[event[0].belong]);

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

{

**if** (i == event[0].belong)

**continue**;

**double** p = segment[i].A.cross(event[0].point);

**double** q = event[0].point.cross(segment[i].B);

**if** (sig(p) > 0 && sig(q) > 0)

{

S.insert(segment[i]);

}

}

**for** (**int** i = 0; i < E; i++)

{

visit = event[i].point;

**if** (event[i].in)

{

S.insert(segment[event[i].belong]);

ite = S.begin();

result[ite->index] = 1;

}

**else**

{

ite = S.find(segment[event[i].belong]);

S.erase(ite);

**if** (!S.empty())

{

ite = S.begin();

result[ite->index] = 1;

}

}

}

**int** count = 0;

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

count += result[i];

**printf**("%d\n", count);

}