Geometria Computacional

* Cubo:
  + - volumen=lado^3,
    - diagonal=lado\*raiz(3)
* Ortoedro:
  + - volumen=alto\*ancho\*largo ,
    - diagonal=raiz(alto^2+ancho^2+largo^2)
* Prisma:
  + - volumen=areaBase\*altura
* Cono circular:
  + - volumen=(1/3)\*pi\*radio^2\*altura
    - area lateral=pi\*radio\*generatriz
* Esfera:
  + - volumen=(4/3)\*pi\*radio^2
* Octaedro regular:
  + - volumen=(lado^3)/3\*raiz(2)
    - area total=2\*(lado^2)\*raiz(3)
* Tetraedro regular:
  + - volumen=(lado^3)/12\*raiz(2)
    - area total=(lado^2)\*raiz(3)
* Área del cualquier polígono regular
  + - = n \* Apotema^2 \* tan(π/n) n => número de lados
    - = ½ \* n \* Radio^2\* sin(2 × π/n)
    - = ¼ \* n \* Lado^2 / tan(π/n)

Area: **//para calcular el area de un polígono dados sus vértices**

**double** area(){

**double** total = 0.0;

**int** i, j;

**for**(i = 0; i < NumPoints; i++)

{

j = (i+1) % NumPoints;

total += (points[i].x\*points[j].y) –

(points[j].x\*points[i].y);

}

**return** total / 2;}

Radio de la circunferencia circunscrita al triangulo

 s =semiperimetro

Radio de la circunferencia inscrita al triangulo

 s =semiperimetro

Longitud de la mediana

ta = ½ \* Raiz de (2(b^2+c^2)-a^2)

Longitud de la altura

ha = 2/a \* Raiz de (s(s-a)(s-b)(s-c))

Longitud del angulo bisector

la = Raiz de (bc[1-(a/(b+c))^2])

Fibinacci- numeros de Binet

f(n-1) = Raiz de(5)/5 [((1+Raiz de (5))/2)^n - ((1-Raiz de (5))/2)^n ]

Cuantas diagonales hay en un poligono convexo de n lados

= n(n-3)/2

Circunferencia

longitud del arco =radio\*arcoseno del angulo

area del sector circular = (1/2)\*longitud del arco \* radio

area del anillo circular = pi(radio1^2-radio2^2)

Intersectar: **//para saber si dos líneas se cortan**

**boolean** intersect(CLine l1, CLine l2) {

**if** (ccw(l1.p1, l1.p2, l2.p1) == 0)

**return** point\_in\_box(l1.p1, l1.p2, l2.p1);

**if** (ccw(l1.p1, l1.p2, l2.p2) == 0)

**return** point\_in\_box(l1.p1, l1.p2, l2.p2);

**if** (ccw(l2.p1, l2.p2, l1.p1) == 0)

**return** point\_in\_box(l2.p1, l2.p2, l1.p1);

**if** (ccw(l2.p1, l2.p2, l1.p2) == 0)

**return** point\_in\_box(l2.p1, l2.p2, l1.p2);

**return** ((ccw(l1.p1, l1.p2, l2.p1) \* ccw(l1.p1, l1.p2, l2.p2) < 0)

&& (ccw(l2.p1, l2.p2, l1.p1) \* ccw(l2.p1, l2.p2, l1.p2) < 0));

}

**int** ccw(CPoint a, CPoint b, CPoint c) {

**int** dx1 = b.x - a.x;

**int** dx2 = c.x - a.x;

**int** dy1 = b.y - a.y;

**int** dy2 = c.y - a.y;

**if** (dx1 \* dy2 > dy1 \* dx2)

**return** 1; // counter-clockwise order

**if** (dx1 \* dy2 < dy1 \* dx2)

**return** -1; // clockwise order

**return** 0; // collinear

}

Punto en un Cubo:

**boolean** point\_in\_box(CPoint corner1, CPoint corner2, CPoint point) {

**int** x1 = min(corner1.x, corner2.x);

**int** x2 = max(corner1.x, corner2.x);

**int** y1 = min(corner1.y, corner2.y);

**int** y2 = max(corner1.y, corner2.y);

**return** (point.x >= x1 && point.x <= x2 && point.y >= y1 && point.y <= y2);

}

Para saber el punto donde se cortan dos líneas

Primer paso => A = y2-y1 B = x1-x2 C = A\*x1+B\*y1

**double** det = A1\*B2 - A2\*B1

**if**(det == 0){

//Lines are parallel

}**else**{

**double** x = (B2\*C1 - B1\*C2)/det

**double** y = (A1\*C2 - A2\*C1)/det

}

# Points in Figures: Rectangles, Circles, Triangles

To determine whether a point is in figure, use the following rule:  
Note: The problem states that (x,y) will never coincide with a figure border.

For Rectangles:

A point (x,y) is contained in a Rectangle (xUpperLeft,yUpperLeft,xLowerRight,yLowerRight) if and only if **(x>xUpperLeft && x<xLowerRight) &&  (y<yUpperLeft && y>yLowerRight)**

For Circles:

A point (x,y) is contained in a Circle (x0,y0,Radius) if and only if  
**(x-x0)^2 + (y-y0)^2 < Radius^2**

For Triangle:

A point (x,y) is contained in a Triangle (x1,y1,x2,y2,x3,y3) if and only if  
**abs(a1 + a2 + a3 - a4) <= 0.000001**

Note for triangle:

a1,a2,a3 are the area of 3 small triangle (with point (x,y) in one of it's side) inside the original triangle.  
a4 is the area of the original triangle.  
If the point is **inside** the original triangle, then a1+a2+a3 must be **equal** to a4  
If the point is **outside** the original triangle, then a1+a2+a3 will be **larger** than a4  
Try it yourself by drawing a simple diagram.  
I use epsilon to guard against miscalculation. (epsilon=0.000001)  
To find the area, use Matrix theorem to find area (refer to your math books)

**ENCONTRAR LOS LADOS DE UN TRIANGULO CONOCIENDO LAS 3 MEDIANAS**

l1 = (2 \* sqrt(2 \* (pow(b, 2) + pow(c, 2)) - pow(a, 2))) / 3;

l2 = (2 \* sqrt(2 \* (pow(a, 2) + pow(c, 2)) - pow(b, 2))) / 3;

l3 = (2 \* sqrt(2 \* (pow(b, 2) + pow(a, 2)) - pow(c, 2))) / 3;

**RADIO DE LA CIRCUNFERENCIA INSCRITA EN UN TRIANGULO**

//a, b, c son los lados del triangulo

double s = (a + b + c) / 2;

double r = sqrt((s - a) \* (s - b) \* (s - c) / s);

**VOLUMEN DE UN TETRAHEDRO**

double Volumen1() {

double COSB = (WX\*WX+WZ\*WZ-XZ\*XZ)/(2.0\*WX\*WZ);

double COSC = (WX\*WX+WY\*WY-XY\*XY)/(2.0\*WX\*WY);

double COSD = (WZ\*WZ+WY\*WY-YZ\*YZ)/(2.0\*WZ\*WY);

Raiz = pow(1+2\*COSB\*COSC\*COSD-COSB\*COSB-COSC\*COSC-COSD\*COSD,0.5);

return Raiz\*(WX\*WY\*WZ/6.0); }

double VOL(double u, double v, double w, double U, double V, double W) {

return Volumen1(); //Usando Metodo 1...

WX = u, WY = v, WZ = w, YZ = U, XZ = V, XY = W;

return Volumen2(u,v,w,U,V,W); //Usando Metodo 2

//Importante ...

// u, v, y w son 3 lados con un vertice comun...

// U lado opuesto a u (no tienen vertices comunes).

// V lado opuesto a v (no tienen vertices comunes).

// W lado opuesto a w (no tienen vertices comunes). }

**AREA DE INTERSECCION DE 2 CIRCUNFERENCIAS**

//center[i] es el centro de la circunferencia, deben estar en la misma recta

//r es el radio de la circunferencia, beben ser iguales

double d = centers[i] - centers[i - 1];

double ang = Math.acos(d / (2 \* r));

double s = ang \* r \* r;

double t = (d / 2) \* Math.pow(r \* r - (d \* d / 4), 0.5);

double areaInters = 2 \* (s - t);

**PLANTILLA DE GEOMETRIA COMPUTACIONAL**

const double EPS = 1e-8;

const double oo = 1e12;

const double PI = 3.14159265358979323846264;

#define X real()

#define Y imag()

typedef complex<double> P;

typedef vector<P> Pol;

//cross product en 3D

//i = y1\*z2 - y2\*z1;

//j = x2\*z1 - x1\*z2;

//k = x1\*y2 - x2\*y1;

struct circle {

P p;

double r;

circle() {}

circle(P x, double rr) {

p = x, r = rr; } };

struct L: public vector<P> { //Linea

L(P a, P b) {

push\_back(a);

push\_back(b); } };

inline bool operator<(const P a, const P b) {

return a.X != b.X ? a.X < b.X : a.Y < b.Y; }

double cross(P a, P b) { //1

return imag(conj(a) \* b); }

double dot(P a, P b) { //2

return (conj(a) \* b).X; }

double distSqr(P &p1, P &p2){

return (p1.X-p2.X)\*(p1.X-p2.X) + (p1.Y-p2.Y)\*(p1.Y-p2.Y); }

bool contain(circle c,P p){

return distSqr(c.p,p)<= c.r\*c.r; }

**AREA DE UN POLYGONO**

struct point {

int x, y; };

//P es un poligono ordenado anticlockwise.

//Si es clockwise,retorna el area negativa

//Si no esta ordenado no sirve.

//P[0]!=P[n-1]

double PolygonArea(const vector<point>&p) {

double r = 0.0;

for (int i = 0; i < p.size(); ++i) {

int j = (i + 1) % p.size();

r += p[i].x \* p[j].y - p[j].x \* p[i].y; }

return r / 2.0; }

**CENTRO DE CIRCUNFERENCIA DADO 3 PTOS**

Point calcuateCenter(Point p1, Point p2, Point p3) {

double A1 = p2.y - p1.y;

double B1 = -(p1.x - p2.x);

double midpointx1 = (p1.x + p2.x) / 2;

double midpointy1 = (p1.y + p2.y) / 2;

double D1 = B1 \* midpointx1 + A1 \* midpointy1;

double A2 = p3.y - p2.y;

double B2 = -(p2.x - p3.x);

double midpointx2 = (p2.x + p3.x) / 2;

double midpointy2 = (p2.y + p3.y) / 2;

double D2 = B2 \* midpointx2 + A2 \* midpointy2;

double det = B1 \* A2 - B2 \* A1;

if (det == 0)

return null;

double x = (A2 \* D1 - A1 \* D2) / det;

double y = (B1 \* D2 - B2 \* D1) / det;

return new Point(x, y);}

**DETERMINAR SI UN POLYGONO ES CONVEXO**

Returns positive if a-b-c makes a left turn.

Returns negative if a-b-c makes a right turn.

Returns 0.0 if a-b-c are colineal.

double turn(const point& a, const point& b, const point& c) {

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

if (fabs(z) < 1e-9)

return 0.0;

return z; }

Returns true if polygon p is convex.

False if its concave or it can’t be determined

(For example, if all points are colineal we can’t make a choice).

bool isConvexPolygon(const vector<point>& p) {

int mask = 0;

int n = p.size();

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

int j = (i + 1) % n;

int k = (i + 2) % n;

double z = turn(p[i], p[j], p[k]);

if (z < 0.0) {

mask |= 1;

} else if (z > 0.0) {

mask |= 2; }

if (mask == 3)

return false; }

return mask != 0; }

**DETERMINAR SI UN PTO PERTENECE A UN POLYGONO CUALQUIERA**

//los ptos tienen que estar ordenados clockwise o counterclockwise

//manejar el caso de que el pto este en la misma arista del polygono

//en caso de que halla que incluirlo

#define EPS 1e-9

double polarAngle(point p) {

if (fabs(p.x) <= EPS && fabs(p.y) <= EPS)

return -1.0;

if (fabs(p.x) <= EPS)

return (p.y > EPS ? 1.0 : 3.0) \* acos(0);

double theta = atan(1.0 \* p.y / p.x);

if (p.x > EPS)

return (p.y >= -EPS ? theta : (4 \* acos(0) + theta));

return (2 \* acos(0) + theta); }

bool pointInPoly(point p, point poly[], int n) {

double ang = 0.0;

for (int i = n - 1, j = 0; j < n; i = j++) {

point v = (point) {poly[i].x - p.x, poly[i].y - p.y};

point w = (point) {poly[j].x - p.x, poly[j].y - p.y};

double va = polarAngle(v);

double wa = polarAngle(w);

double xx = wa - va;

if (va < -0.5 || wa < -0.5 || fabs(fabs(xx) - 2 \* acos(0)) < EPS) {

//ON THE EDGE

ang += 2 \* acos(0);

continue; }

if (xx < -2 \* acos(0))

ang += xx + 4 \* acos(0);

else if (xx > 2 \* acos(0))

ang += xx - 4 \* acos(0);

else

ang += xx; }

return (ang \* ang > 1.0); }

**DETERMINAR SI UN PTO PERTENECE A UN POLYGONO CONVEXO**

//no pueden existir ptos duplicados y el area no puede ser cero

//los vertices tienen que estar ordenados anti-clockwise

struct pt {

int x, y; };

struct ang {

int a, b; };

bool operator <(const ang & p, const ang & q) {

if (p.b == 0 & &q.b == 0)

return p.a < q.a;

return p.a \* 1ll \* q.b < p.b \* 1ll \* q.a; }

long long sq(pt & a, pt & b, pt & c) {

return a.x \* 1ll \* (b.y - c.y) + b.x \* 1ll \* (c.y - a.y)

+ c.x \* 1ll \* (a.y - b.y); }

int main() {

int n; cin >> n; vector<pt> p(n); int zero\_id = 0;

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

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

if (p[i].x < p[zero\_id].x || (p[i].x == p[zero\_id].x && p[i].y < p[zero\_id].y))

zero\_id = i;}

pt zero = p[zero\_id]; rotate(p.begin(), p.begin() + zero\_id, p.end());

p.erase(p.begin()); --n; vector<ang> a(n);

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

a[i].a = p[i].y - zero.y;

a[i].b = p[i].x - zero.x;

if (a[i].a == 0)

a[i].b = a[i].b < 0 ? -1 : 1;}

for (;;) {

pt q;// leer pto

bool in = false;

if (q.x >= zero.x)

if (q.x == zero.x && q.y == zero.y)

in = true;

else {

ang my = { q.y - zero.y, q.x - zero.x };

if (my.a == 0)

my.b = my.b < 0 ? -1 : 1;

vector<ang>::iterator it = upper\_bound(a.begin(), a.end(), my);

if (it == a.end() & &my.a == a[n - 1].a & &my.b == a[n - 1].b)

it = a.end() - 1;

if (it != a.end() && it != a.begin()) {

int p1 = int(it - a.begin());

if (sq(p[p1], p[p1 - 1], q) <= 0)

in = true; }}

puts(in ? "INSIDE" : "OUTSIDE"); } }

**DISTANCIA DE UN PTO A UNA RECTA**

double dot(double[] A, double[] B, double[] C) {

double[] AB, BC; AB = new double[2]; BC = new double[2];

AB[0] = B[0] - A[0]; AB[1] = B[1] - A[1]; BC[0] = C[0] - B[0];

BC[1] = C[1] - B[1];

double dot = AB[0] \* BC[0] + AB[1] \* BC[1];

return dot;}

double cross(double[] A, double[] B, double[] C) {

double[] AB, AC; AB = new double[2];AC = new double[2];

AB[0] = B[0] - A[0];AB[1] = B[1] - A[1]; AC[0] = C[0] - A[0];

AC[1] = C[1] - A[1];

double cross = AB[0] \* AC[1] - AB[1] \* AC[0];

return cross;}

double distance(double[] A, double[] B) {

double d1 = A[0] - B[0]; double d2 = A[1] - B[1];

return Math.sqrt(d1 \* d1 + d2 \* d2);}

double linePointDist(double[] A, double[] B, double[] C, boolean isSegement) {

double dist = cross(A, B, C) / distance(A, B);

if (isSegement) {

double dot1 = dot(A, B, C);

if (dot1 > 0)

return distance(B, C);

double dot2 = dot(B, A, C);

if (dot2 > 0)

return distance(A, C);}

return Math.abs(dist); }

**DISTANCIA MAXIMA ENTRE 2 PTOS**

double diameter(Pol pt) {//24, 1

int is=0,js=0, n=pt.size();

FAB(i,1,n){

if(pt[i].Y >pt[is].Y) is=i;

if(pt[i].Y <pt[js].Y) js=i; }

double maxd=norm(pt[is]-pt[js]);

int i,maxi,j,maxj;

i = maxi = is; j = maxj = js;

do {

if(cross(pt[(i+1)%n]-pt[i],

pt[(j+1)%n]-pt[j])>=0)

j=(j+1)%n; else i=(i+1)%n;

if (norm(pt[i]-pt[j])>maxd){

maxd =norm(pt[i]-pt[j]);

maxi=i; maxj=j;

} }while(i!=is || j!=js);

return maxd; }

**RADIO DEL MAYOR CIRCULO INSCRITO EN UN POLYGONO CONVEXO**

const double EPS = 1E-9;

#define INF 1000000000

int steps = 60;

struct pt {

double x, y; };

struct line {

double a, b, c; };

double dist(double x, double y, line & l) {

return abs(x \* l.a + y \* l.b + l.c); }

double radius(double x, double y, vector<line> & l) {

int n = (int) l.size();

double res = INF;

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

res = min(res, dist(x, y, l[i]));

return res; }

double y\_radius(double x, vector<pt> & a, vector<line> & l) {

int n = (int) a.size();

double ly = INF, ry = -INF;

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

int x1 = a[i].x, x2 = a[(i + 1) % n].x, y1 = a[i].y, y2 =

a[(i + 1) % n].y;

if (x1 == x2)

continue;

if (x1 > x2)

swap(x1, x2), swap(y1, y2);

if (x1 <= x + EPS & &x - EPS <= x2) {

double y = y1 + (x - x1) \* (y2 - y1) / (x2 - x1);

ly = min(ly, y);

ry = max(ry, y);} }

for (int sy = 0; sy < steps; ++sy) {

double diff = (ry - ly) / 3;

double y1 = ly + diff, y2 = ry - diff;

double f1 = radius(x, y1, l), f2 = radius(x, y2, l);

if (f1 < f2)

ly = y1;

else

ry = y2; }

return radius(x, ly, l); }

int main() {

int n;

vector<pt> a(n);//poligono

vector<line> l(n);

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

l[i].a = a[i].y - a[(i + 1) % n].y;

l[i].b = a[(i + 1) % n].x - a[i].x;

double sq = sqrt(l[i].a \* l[i].a + l[i].b \* l[i].b);

l[i].a /= sq, l[i].b /= sq;

l[i].c = -(l[i].a \* a[i].x + l[i].b \* a[i].y); }

double lx = INF, rx = -INF;

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

lx = min(lx, a[i].x);

rx = max(rx, a[i].x); }

for (int sx = 0; sx < steps; ++sx) {

double diff = (rx - lx) / 3;

double x1 = lx + diff, x2 = rx - diff;

double f1 = y\_radius(x1, a, l), f2 = y\_radius(x2, a, l);

if (f1 < f2)

lx = x1;

else

rx = x2;

}

double ans = y\_radius(lx, a, l);

printf("%.7lf", ans); }

**INTERSECCION DE 2 CIRCULOS**

//supongamos que el circulo 1 está en el origen, de lo contrario solo hay que trasladarlo

//con ambos circulos tenemos el sistema:

x^2+y^2=r1^2

(x-x2)^2+(y-y2)^2=r2^2

//sustraemos la segunda ecuacion de la primera y obtenemos el sistema:

x^2+y^2=r1^2

x(-2x2)+y(-2y2)+(x2^2+y2^2+r1^2-r2^2)=0

//reducir el problema al problema de la interseccion

//de un circulo y una line:

Ax+By+C=0,A=-2x2,B=-2y2,C=x2^2+y2^2+r1^2-r2^2

**INTERSECCION LINEA CIRCULO**

//el centro del circulo debe estar en el origen de coordenadas, sino solo hay que trasladarlo

//a, b, c son los coeficientes de la ecuacion de la recta

double r, a, b, c; // input

#define EPS 1e-9

void line\_circle\_inters() {

double x0 = -a \* c / (a \* a + b \* b), y0 = -b \* c / (a \* a + b \* b);

if (c \* c > r \* r \* (a \* a + b \* b) + EPS)

puts("no points");

else if (abs(c \* c - r \* r \* (a \* a + b \* b)) < EPS) {

puts("one point");

cout << x0 << ' ' << y0 << '\n';

} else {

double d = r \* r - c \* c / (a \* a + b \* b);

double mult = sqrt(d / (a \* a + b \* b));

double ax, ay, bx, by;

ax = x0 + b \* mult;

bx = x0 - b \* mult;

ay = y0 - a \* mult;

by = y0 + a \* mult;

puts("2 points");

cout << ax << ' ' << ay << '\n' << bx << ' ' << by << '\n'; } }

**INTERSECCION LINEA LINEA**

//si lo que tenemos es un segmento delimitado por 2 ptos

//entonces primero tenemos que encontrar A, B, C para cada segmento

A = y2-y1

B = x1-x2

C = A\*x1+B\*y1

//pto interseccion

double det = A1\*B2 - A2\*B1

if(det == 0){

//Lines are parallel

}else{

double x = (B2\*C1 - B1\*C2)/det

double y = (A1\*C2 - A2\*C1)/det }

**PERIMETRO DE LA UNION DE RECTANGULOS**

#define MAXC 1 << 17

#define LEFT( v ) 2 \* ( v ) + 1

#define RIGHT( v ) 2 \* ( v ) + 2

int N, last, size;

long long perimeter;

struct event {

int start, lo, hi, flag;

bool operator < ( const event &e ) const { return start < e.start; } };

vector< event > events[2];

int amount[ 3 \* MAXC ], times[ 3 \* MAXC ];

void update( int node, int lo, int hi, int& start, int& end, int& value ) {

if ( lo > end || hi < start ) return;

if ( start <= lo && hi <= end )

times[node] += value;

else {

int mid = ( lo + hi ) / 2;

update( LEFT( node ), lo, mid, start, end, value );

update( RIGHT( node ), mid + 1, hi, start, end, value ); }

if ( !times[node] )

amount[node] = ( lo == hi ) ? 0 :

amount[ LEFT( node ) ] + amount[ RIGHT( node ) ];

else amount[node] = hi - lo + 1;}

void lineSweep( vector< event >& ls ) {

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

last = 0; size = ls.size();

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

update( 0, 0, MAXC, ls[i].lo, ls[i].hi, ls[i].flag );

perimeter += abs( last - amount[0] );

last = amount[0]; }}

int main() {

scanf( "%d", &N );

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

int x1, x2, y1, y2;

scanf( "%d %d %d %d", &x1, &y1, &x2, &y2 );

if ( x1 > x2 ) swap( x1, x2 );

if ( y1 > y2 ) swap( y1, y2 );

events[0].push\_back( ( event ) { y1, x1, x2 - 1, +1 } );

events[0].push\_back( ( event ) { y2, x1, x2 - 1, -1 } );

events[1].push\_back( ( event ) { x1, y1, y2 - 1, +1 } );

events[1].push\_back( ( event ) { x2, y1, y2 - 1, -1 } ); }

lineSweep( events[0] );

lineSweep( events[1] );

printf( "%lld\n", perimeter ); }