C++

#include <iostream>

#include <cstdlib>

#include <cstdio>

#include <cstring>

#include <vector>

#include <cmath>

#include <algorithm>

using namespace std;

const int MAXN = 10000;

const double PI = atan(1.0) \* 4;

const double EPS = 1e-10;

class Point {

public:

double x, y;

Point() {}

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

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

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

Point &operator += (const Point &r) { x += r.x; y += r.y; return \*this; }

Point &operator \*= (double m) { x \*= m; y \*= m; return \*this; }

Point pOfRotate(double angle) const {

double cosA = cos(angle);

double sinA = sin(angle);

return Point(cosA\*x-sinA\*y, sinA\*x+cosA\*y);

}

Point pOfRotate90() const { return Point(-y, x); }

double length() const { return sqrt(x\*x+y\*y); }

Point pOfNormal() const {

double len = length();

return Point(x/len, y/len);

}

double angle() const { return atan2(y, x); }

};

ostream & operator <<(ostream &os, const Point &v)

{

os << "(" << v.x << "," << v.y << ")";

return os;

}

class Segment;

class Circle;

class Seg {

public:

virtual double getLeft() const = 0;

virtual double getRight() const = 0;

virtual double getY(double x) const = 0;

virtual double getLength(double x1, double x2) const = 0;

virtual void intersect(Seg \*r) const = 0;

virtual void intersect(const Segment &v) const = 0;

virtual void intersect(const Circle &v) const = 0;

bool contains(double x) const { return x>=getLeft() && x<=getRight(); }

virtual void acceptPrint(ostream &os) const = 0;

};

ostream & operator <<(ostream &os, const Seg &v)

{

v.acceptPrint(os);

return os;

}

Point intersectRet[4];

int tIntersectRet;

class Segment : public Seg {

public:

Point a, b;

Segment &moveLeft(double dis)

{

Point tmp = ((b-a).pOfRotate90().pOfNormal() \*= dis);

a += tmp;

b += tmp;

return \*this;

}

virtual double getLeft() const { return a.x; }

virtual double getRight() const { return b.x; }

virtual double getY(double x) const {

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

}

virtual double getLength(double x1, double x2) const {

return (x2-x1) \* (b-a).length() / (b.x-a.x);

}

virtual void intersect(Seg \*r) const {

r->intersect(\*this);

}

virtual void intersect(const Segment &v) const {

tIntersectRet = 0;

double ang = (b-a).angle();

Point c = (v.a-a).pOfRotate(-ang);

Point d = (v.b-a).pOfRotate(-ang);

// Bug

//double di = b.length();

double di = (b-a).length();

if (!((c.y>0&&d.y<0) || (c.y<0&&d.y>0)))

return ;

double x = (d.x-c.x) \* (-c.y) / (d.y-c.y) + c.x;

if (x<0 || x>di)

return ;

Point ret = Point(x,0).pOfRotate(ang)+a;

intersectRet[tIntersectRet++] = ret;

}

virtual void intersect(const Circle &v) const;

virtual void acceptPrint(ostream &os) const {

os << a << "-" << b;

}

};

class Circle : public Seg {

public:

Point c;

double r;

virtual double getLeft() const { return c.x - r; }

virtual double getRight() const { return c.x + r; }

virtual double getY(double x) const {

double y2 = r \* r - (c.x - x) \* (c.x - x);

if (y2<0) y2 = 0;

return c.y + sqrt(y2);

}

virtual double getLength(double x1, double x2) const {

x1 -= c.x; x2 -= c.x;

double a1 = Point(x1, sqrt(abs(r\*r-x1\*x1))).angle(), a2 = Point(x2, sqrt(abs(r\*r-x2\*x2))).angle();

return (a1-a2) \* r;

}

virtual void intersect(Seg \*r) const {

r->intersect(\*this);

}

virtual void intersect(const Segment &v) const {

tIntersectRet = 0;

Point a = v.a - c;

Point b = v.b - c;

double ang = (b-a).angle();

Point nA = a.pOfRotate(-ang);

Point nB = b.pOfRotate(-ang);

double y = nA.y;

if (y>r || y<-r)

return ;

double x = sqrt(r\*r - y\*y);

if (x>=nA.x && x<=nB.x)

intersectRet[tIntersectRet++] = Point(x, y).pOfRotate(ang) + c;

if (-x>=nA.x && -x<=nB.x)

intersectRet[tIntersectRet++] = Point(-x, y).pOfRotate(ang) + c;

}

virtual void intersect(const Circle &v) const {

tIntersectRet = 0;

Point p = v.c - c;

double d = p.length();

if (d > r + v.r || d==0)

return ;

double x = (r\*r - v.r\*v.r + d\*d) / (2\*d);

if (x <= r)

{

double y = sqrt(abs(r\*r - x\*x));

double ang = p.angle();

intersectRet[tIntersectRet++] = Point(x,y).pOfRotate(ang) + c;

intersectRet[tIntersectRet++] = Point(x,-y).pOfRotate(ang) + c;

}

}

virtual void acceptPrint(ostream &os) const {

os << c << "," << r;

}

};

void Segment::intersect(const Circle &v) const {

v.intersect(\*this);

}

int n;

Point inps[MAXN];

vector<Seg \*> segs;

vector<double> spes;

double radius = 1;

void input()

{

scanf("%d%lf", &n, &radius);

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

{

double x, y;

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

inps[i] = Point(x, y);

}

}

void process()

{

segs.clear();

spes.clear();

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

{

Circle \*tmp = new Circle;

tmp->c = inps[i];

tmp->r = radius;

segs.push\_back(tmp);

}

for (int i = 0; i + 1 < n; ++i)

{

Segment \*tmp = new Segment;

tmp->a = inps[i];

tmp->b = inps[i+1];

tmp->moveLeft(radius);

segs.push\_back(tmp);

}

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

{

spes.push\_back(segs[i]->getLeft());

spes.push\_back(segs[i]->getRight());

}

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

{

for (int j = i+1; j < (int)segs.size(); ++j)

{

segs[i]->intersect(segs[j]);

if (tIntersectRet > 0)

{

for (int id = 0; id < tIntersectRet; ++id)

{

//cout << \*segs[i] << " " << \*segs[j] << " : " << intersectRet[id] << endl;

spes.push\_back(intersectRet[id].x);

}

}

}

}

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

double pre = spes[0];

const double NONE = 1e30;

double preEnd = NONE;

double totalLen = 0;

for (int i = 1; i < (int)spes.size(); ++i)

{

if (spes[i]-pre < EPS)

continue;

double cur = (pre+spes[i]) / 2;

//cout << "Processing " << cur << " from " << pre << " to " << spes[i] << endl;

if (cur>=inps[0].x && cur<=inps[n-1].x)

{

double MY = -NONE;

int who;

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

{

if (!segs[j]->contains(cur))

continue;

double y = segs[j]->getY(cur);

if (y > MY)

{

MY = y;

who = j;

}

}

if (preEnd != NONE)

{

double LY = segs[who]->getY(pre);

//cout << "Drop info " << \*segs[who] << " " << "[" << pre << "]" << endl;

totalLen += abs(preEnd-LY);

//cout << "Pre drop = " << abs(preEnd-LY) << " from " << preEnd << " to " << LY << endl;

}

double len = segs[who]->getLength(pre, spes[i]);

if (len < 0)

printf("Error!\n");

//cout << "Curlen = " << len << " from " << pre << " to " << spes[i] << endl;

totalLen += len;

preEnd = segs[who]->getY(spes[i]);

}

pre = spes[i];

}

printf("%0.2lf\n", totalLen);

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

delete segs[i];

segs.clear();

}

int main()

{

input();

process();

return 0;

}