Geometria Computacional

Polígonos: problemas resolvidos

Prof. Edson Alves

2019

Faculdade UnB Gama

Sumário

Codeforces Beta Round #1 – Problem C: Ancient Berland Circus

Problema

Nowadays all circuses in Berland have a round arena with diameter 13 meters, but in the past things were different.

In Ancient Berland arenas in circuses were shaped as a regular (equiangular) polygon, the size and the number of angles could vary from one circus to another. In each corner of the arena there was a special pillar, and the rope strung between the pillars marked the arena edges.

Recently the scientists from Berland have discovered the remains of the ancient circus arena. They found only three pillars, the others were destroyed by the time.

You are given the coordinates of these three pillars. Find out what is the smallest area that the arena could have.

Entrada e saída

Input

The input file consists of three lines, each of them contains a pair of numbers – coordinates of the pillar. Any coordinate doesn't exceed 1000 by absolute value, and is given with at most six digits after decimal point.

Output

Output the smallest possible area of the ancient arena. This number should be accurate to at least 6 digits after the decimal point. It's guaranteed that the number of angles in the optimal polygon is not larger than 100.

Exemplo de entradas e saídas

Sample Input

```
0.000000 0.000000
1.000000 1.000000
0.000000 1.000000
```

Sample Output

```
1.00000000
```

Observações sobre o problema

- Para encontrar o polígono regular que contém os três pontos dados, é preciso determinar o círculo circunscrito
- Observe que este círculo será o mesmo que circunscreve o triângulo formado pelos pontos dados
- O número mínimo de lados pode ser determinado por força bruta, uma vez que o número máximo de lados é igual a 100
- Para facilitar o processo de rotação, os três pontos devem ser transladados de modo que o centro do círculo circunscrito fique na origem
- Para um número de lados n, rotacione um ponto P escolhido em todos os ângulos possíveis e veja se os outros dois pontos foram encontrados
- A rotina de comparação de pontos flutuantes não deve usar um valor ε muito agressivo, pois pode levar ao WA ($\varepsilon = 10^{-5}$ é suficiente, $\varepsilon = 10^{-6}$ gera WA no oitavo caso de teste)

Solução AC

```
#include <bits/stdc++.h>

using namespace std;

const double PI { acos(-1.0) };
const int MAX { 110 };

double angles[MAX];

struct Point {
    double x, y;

    double distance(const Point& P) const
    {
        return hypot(x - P.x, y - P.y);
    }

    Point translate(const Point& P) const
    {
        return Point { x + P.x, y + P.y };
    }
```

Solução AC

```cpp

    Point rotate(double angle) const
    {
        auto xv = x*cos(angle) - y*sin(angle);
        auto yv = x*sin(angle) + y*cos(angle);

        return Point { xv, yv };
    }

    bool operator==(const Point& P) const
    {
        const double EPS { 1e-5 };

        return fabs(x - P.x) < EPS and fabs(y - P.y) < EPS;
    }
};

struct Triangle {
    Point A, B, C;

```

Solução AC

```
    double area() const
    {
        auto a = A.distance(B);
        auto b = B.distance(C);
        auto c = C.distance(A);
        auto s = (a + b + c) / 2;

        return sqrt(s*(s - a)*(s - b)*(s - c));
    }

    double circumradius() const
    {
        auto a = A.distance(B);
        auto b = B.distance(C);
        auto c = C.distance(A);

        return (a * b * c)/(4 * area());
    }

```

Solução AC

```
    Point circumcenter() const
    {
        auto d = 2*(A.x*(B.y - C.y) + B.x*(C.y - A.y) + C.x*(A.y - B.y));

        auto A2 = A.x*A.x + A.y*A.y;
        auto B2 = B.x*B.x + B.y*B.y;
        auto C2 = C.x*C.x + C.y*C.y;

        auto x = (A2*(B.y - C.y) + B2*(C.y - A.y) + C2*(A.y - B.y))/d;
        auto y = (A2*(C.x - B.x) + B2*(A.x - C.x) + C2*(B.x - A.x))/d;

        return Point { x, y };
    }
};

void precomp()
{
    for (int i = 1; i < MAX; ++i)
        angles[i] = (2.0*PI)/i;
}
```

Solução AC

```
int sides(const Point& P, const Point& Q, const Point& R)
{
    for (int i = 3; i < 100; ++i)
    {
        auto angle = angles[i];
        int match = 0;
        Point S { P };

        for (int j = 0; j < i; ++j)
        {
            if (Q == S)
                ++match;

            if (R == S)
                ++match;

            S = S.rotate(angle);
        }

        if (match == 2)
            return i;
```

Solução AC

```cpp
    }

    return 100;
}

int main()
{
    precomp();

    Point P, Q, R;

    cin >> P.x >> P.y >> Q.x >> Q.y >> R.x >> R.y;

    Triangle t { P, Q, R };

    auto r = t.circumradius();
    auto C = t.circumcenter();

    P = P.translate(Point { -C.x, -C.y } );
    Q = Q.translate(Point { -C.x, -C.y } );
    R = R.translate(Point { -C.x, -C.y } );
```

Solução AC

```

    int min_sides = sides(P, Q, R);

    auto area = (r * r * min_sides*sin(angles[min_sides]))/2.0;

    cout.precision(6);
    cout << fixed << area << '\n';

    return 0;
}
```

UVA 11265 – The Sultan's Problem

Problema

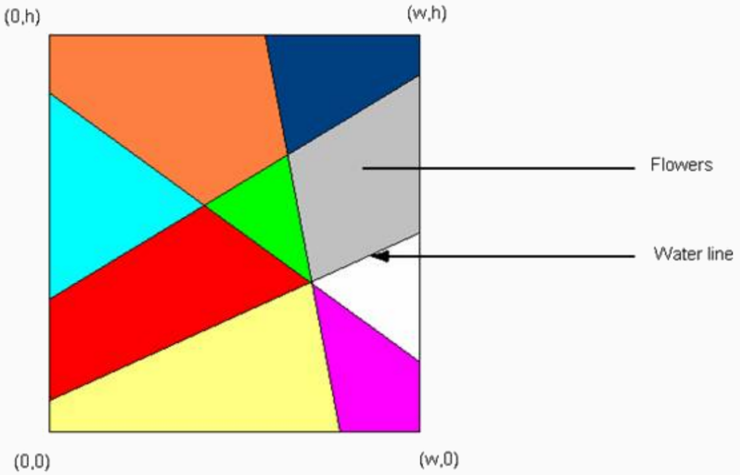
Once upon a time, there lived a great sultan, who was very much fond of his wife. He wanted to build a Tajmahal for his wife (ya, our sultan idolized Mughal emperor Shahjahan). But alas, due to budget cuts, loans, dues and many many things, he had no fund to build something so big. So, he decided to build a beautiful garden, inside his palace.

The garden is a rectangular piece of land. All the palaces water lines run through the garden, thus, dividing it into pieces of varying shapes. He wanted to cover each piece of the land with flowers of same kind.

The figure above shows a sample flower garden. All the lines are straight lines, with two end points on two different edge of the rectangle. Each piece of the garden is covered with the same kind of flowers.

The garden has a small fountain, located at position (x, y). You can assume that, it is not on any of the lines. He wants to fill that piece with her favourite flower. So, he asks you to find the area of that piece.

Problema



Entrada e saída

Input

Input contains around 500 test cases. Each case starts with 5 integers, N, W, H, x, y, the number of lines, width and height of the garden and the location of the fountain. Next N lines each contain 4 integers, x_1 y_1 x_2 y_2, the two end points of the line. The end points are always on the boundary of the garden. You can assume that, the fountain is not located on any line.

Constraints

- $1 \leq W, H \leq 200,000$
- $1 \leq N \leq 500$

Output

For each test case, output the case number, with the area of the piece covered with her favourite flower. Print 3 digits after the decimal point.

Exemplo de entradas e saídas

Sample Input

```
3 100 100 20 30
0 0 100 100
100 0 0 100
0 40 40 100
```

Sample Output

```
Case #1: 1780.000
```

Solução

- A área que contém a fonte F deve ser obtida através de sucessivos cortes do polígono P que representa o jardim
- Inicialmente P é um retângulo
- Cada nova reta fará um corte em P, separando a área que contém F do restante
- A orientação do corte é importante: se F está à esquerda dos pontos A e B da reta, o corte é feito na orientação AB
- Se estiver à direita, a orientação é feita no sentido oposto BA
- Cada corte tem complexidade $O(n)$, onde n é o número de vértices do polígono
- No pior caso, a solução tem complexidade $O(N^2)$, onde N é o número retas

Solução AC

```cpp
#include <bits/stdc++.h>

using namespace std;

struct Point {
    double x, y;

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

    double distance(const Point& P) const
    {
        return hypot(x - P.x, y - P.y);
    }

    bool operator==(const Point& P) const
    {
        const double EPS { 1e-6 };
        return fabs(x - P.x) < EPS and fabs(y - P.y) < EPS;
    }
};

```

Solução AC

```
    vector<Point> vs;
    int n;

    Polygon(const vector<Point>& vs) : vs(vs), n(vs.size())
    {
        vs.push_back(vs[0]);
    }

    double area() const {
        double a = 0;

        for (int i = 0; i < n; ++i)
        {
            a += vs[i].x * vs[i+1].y;
            a -= vs[i+1].x * vs[i].y;
        }

        return 0.5 * fabs(a);
    }
};
```

Solução AC

```cpp
Point intersection(const Point& P, const Point& Q,
    const Point& A, const Point& B)
{
    auto a = B.y - A.y;
    auto b = A.x - B.x;
    auto c = B.x * A.y - A.x * B.y;
    auto u = fabs(a * P.x + b * P.y + c);
    auto v = fabs(a * Q.x + b * Q.y + c);

    return Point((P.x*v + Q.x*u)/(u + v), (P.y*v + Q.y*u)/(u + v));
}

double D(const Point& P, const Point& Q, const Point& R)
{
    return (P.x * Q.y + P.y * R.x + Q.x * R.y)
        - (R.x * Q.y + R.y * P.x + Q.x * P.y);
}

```

Solução AC

```
Polygon cut_polygon(const Polygon& P, const Point& A, const Point& B)
{
    vector<Point> points;

    for (int i = 0; i < P.n; ++i)
    {
        auto d1 = D(A, B, P.vs[i]);
        auto d2 = D(A, B, P.vs[i + 1]);

        if (d1 > -EPS)
            points.push_back(P.vs[i]);

        if (d1 * d2 < -EPS)
            points.push_back(intersection(P.vs[i], P.vs[i+1], A, B));
    }

    return Polygon(points);
}

```

Solução AC

```cpp
int main() {
    int N, W, H, x, y, test = 0;

    while (cin >> N >> W >> x >> y) {
        Polygon p({ Point(0, 0), Point(W, 0), Point(W, H), Point(0, H) });
        Point F(x, y);

        while (N--) {
            Point Q, R;
            cin >> Q.x >> Q.y >> R.x >> R.y;

            if (D(Q, R, F) > 0)
                p = cut_polygon(p, Q, R);
            else
                p = cut_polygon(p, R, Q);
        }

        printf("Case #%d: %.3f\n", ++test, p.area());
    }
    return 0;
}
```

Referências

1. Codeforces Beta Round #1: Ancient Berland Circus
2. UVA 11265 – The Sultan's Problem