Codeforces Beta Round #1

Problem C: Ancient Berland Circus

Prof. Edson Alves

Faculdade UnB Gama

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
```

Solução

- Para encontrar o polígono regular que contém os três pontos dados, é preciso determinar o círculo circunscrito

Solução

- 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

Solução

- 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

Solução

- 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

Solução

- 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

Solução

- 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

```cpp
#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

```
17      Point rotate(double angle) const
18      {
19          auto xv = x*cos(angle) - y*sin(angle);
20          auto yv = x*sin(angle) + y*cos(angle);
21
22          return Point { xv, yv };
23      }
24
25      bool operator==(const Point& P) const
26      {
27          const double EPS { 1e-5 };
28
29          return fabs(x - P.x) < EPS and fabs(y - P.y) < EPS;
30      }
31  };
32
33  struct Triangle {
34      Point A, B, C;
```

Solução AC

```
36      double area() const
37      {
38          auto a = A.distance(B);
39          auto b = B.distance(C);
40          auto c = C.distance(A);
41          auto s = (a + b + c) / 2;
42
43          return sqrt(s*(s - a)*(s - b)*(s - c));
44      }
45
46      double circumradius() const
47      {
48          auto a = A.distance(B);
49          auto b = B.distance(C);
50          auto c = C.distance(A);
51
52          return (a * b * c)/(4 * area());
53      }
```

Solução AC

```
55      Point circumcenter() const
56      {
57          auto d = 2*(A.x*(B.y - C.y) + B.x*(C.y - A.y) + C.x*(A.y - B.y));
58
59          auto A2 = A.x*A.x + A.y*A.y;
60          auto B2 = B.x*B.x + B.y*B.y;
61          auto C2 = C.x*C.x + C.y*C.y;
62
63          auto x = (A2*(B.y - C.y) + B2*(C.y - A.y) + C2*(A.y - B.y))/d;
64          auto y = (A2*(C.x - B.x) + B2*(A.x - C.x) + C2*(B.x - A.x))/d;
65
66          return Point { x, y };
67      }
68  };
69
70  void precomp()
71  {
72      for (int i = 1; i < MAX; ++i)
73          angles[i] = (2.0*PI)/i;
74  }
```

Solução AC

```
76 int sides(const Point& P, const Point& Q, const Point& R)
77 {
78     for (int i = 3; i < 100; ++i)
79     {
80         auto angle = angles[i];
81         int match = 0;
82         Point S { P };
83 
84         for (int j = 0; j < i; ++j)
85         {
86             if (Q == S)
87                 ++match;
88 
89             if (R == S)
90                 ++match;
91 
92             S = S.rotate(angle);
93         }
```

Solução AC

```
95          if (match == 2)
96              return i;
97      }
98
99      return 100;
100 }
101
102 int main()
103 {
104     precomp();
105
106     Point P, Q, R;
107
108     cin >> P.x >> P.y >> Q.x >> Q.y >> R.x >> R.y;
109
110     Triangle t { P, Q, R };
111
112     auto r = t.circumradius();
113     auto C = t.circumcenter();
```

Solução AC

```
115     P = P.translate(Point { -C.x, -C.y } );
116     Q = Q.translate(Point { -C.x, -C.y } );
117     R = R.translate(Point { -C.x, -C.y } );
118
119     int min_sides = sides(P, Q, R);
120
121     auto area = (r * r * min_sides*sin(angles[min_sides]))/2.0;
122
123     cout.precision(6);
124     cout << fixed << area << '\n';
125
126     return 0;
127 }
```