Geometria Computacional

Círculos – Algoritmos: problemas resolvidos

Prof. Edson Alves

2018

Faculdade UnB Gama

Sumário

Educational Codeforces Round #2 – Problem D: Area of Two Circle’s Intersection

Problema

You are given two circles. Find the area of their intersection.

Entrada e saída

Input

The first line contains three integers x_1, y_1, r_1 $(-10^9 \leq x_1, y_1 \leq 10^9, 1 \leq r_1 \leq 10^9)$ - the position of the center and the radius of the first circle.

The second line contains three integers x_2, y_2, r_2 $(-10^9 \leq x_2, y_2 \leq 10^9, 1 \leq r_2 \leq 10^9)$ - the position of the center and the radius of the second circle.

Output

Print the area of the intersection of the circles. The answer will be considered correct if the absolute or relative error doesn't exceed 10^{-6}.

Exemplo de entradas e saídas

Sample Input	Sample Output
0 0 4 6 0 4	7.2529880636417560l379
0 0 5 11 0 5	0.0000000000000000000

Solução com complexidade $O(1)$

- É preciso tratar 3 casos especiais para a solução correta deste problema
- No primeiro caso os círculos não tem interseção, isto é, a distância d entre os pontos $P = (x_1, y_1)$ e $Q = (x_2, y_2)$ é maior ou igual ao dobro da soma dos raios
- Assim, a área de interseção será igual a zero
- No segundo caso, um círculo contém o outro, isto é, $d \leq r_1 + r2$
- Logo a interseção terá a mesma área do círculo de menor raio
- No terceiro e último caso os dois círculos se tocam em exatamente dois pontos
- A área será a soma dos segmentos definidos pelos triângulos formados pelos centros e dos pontos de interseção

Solução AC com complexidade $O(1)$

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

using namespace std;
using ll = long long;

constexpr long double PI = acosl(-1.0);

long double intersection_area(long double r, long double R, long double d)
{
    auto angle = acosl((r*r + d*d - R*R)/(2*r*d));  // Lei dos cossenos
    auto sector = angle * r * r;                    // Setor de 2*angle
    auto T = r * r * cosl(angle) * sinl(angle);     // Aŕea do triângulo

    return sector - T;
}

void solve(ll X1, ll Y1, ll R1, ll X2, ll Y2, ll R2)
{
    // Não se interceptam ou se tocam em um único ponto
    auto dist2 = (X1 - X2)*(X1 - X2) + (Y1 - Y2)*(Y1 - Y2);

```

Solução AC com complexidade $O(1)$

```
    if (dist2 >= (R1 + R2)*(R1 + R2))
    {
        cout << "0.00000000000000000000\n";
        return;
    }

    // O primeiro círculo será o de maior raio
    if (R2 > R1)
    {
        swap(X1, X2);
        swap(Y1, Y2);
        swap(R1, R2);
    }

    // O menor está contido no maior: a resposta é a área do menor
    if (dist2 <= (R1 - R2)*(R1 - R2))
    {
        cout.precision(20);
        cout << PI*R2*R2 << '\n';
        return;
    }
```

Solução AC com complexidade $O(1)$

```cpp

    // Dois pontos de interseção
    auto d = sqrtl(dist2);
    auto A1 = intersection_area(R1, R2, d);
    auto A2 = intersection_area(R2, R1, d);

    cout.precision(20);
    cout << A1 + A2 << endl;
}

int main()
{
    ios::sync_with_stdio(false);

    int X1, Y1, R1, X2, Y2, R2;
    cin >> X1 >> Y1 >> R1 >> X2 >> Y2 >> R2;

    solve(X1, Y1, R1, X2, Y2, R2);

    return 0;
}
```

Codeforces Beta Round #2 – Problem C: Commentator Problem

Problema

The Olympic Games in Bercouver are in full swing now. Here everyone has their own objectives: sportsmen compete for medals, and sport commentators compete for more convenient positions to give a running commentary. Today the main sport events take place at three round stadiums, and the commentator's objective is to choose the best point of observation, that is to say the point from where all the three stadiums can be observed. As all the sport competitions are of the same importance, the stadiums should be observed at the same angle. If the number of points meeting the conditions is more than one, the point with the maximum angle of observation is prefered.

Would you, please, help the famous Berland commentator G. Berniev to find the best point of observation. It should be noted, that the stadiums do not hide each other, the commentator can easily see one stadium through the other.

Entrada e saída

Input

The input data consists of three lines, each of them describes the position of one stadium. The lines have the format x, y, r, where (x, y) are the coordinates of the stadium's center $(-10^3 \leq x, y \leq 10^3)$, and r $(1 \leq r \leq 10^3)$ is its radius. All the numbers in the input data are integer, stadiums do not have common points, and their centers are not on the same line.

Output

Print the coordinates of the required point with five digits after the decimal point. If there is no answer meeting the conditions, the program shouldn't print anything. The output data should be left blank.

Exemplo de entradas e saídas

Sample Input

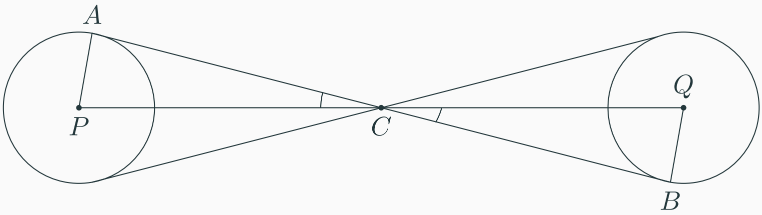
```
0 0 10
60 0 10
30 30 10
```

Sample Output

```
30.00000 0.00000
```

Solução

- Uma estratégia útil para solucionar um problema sofisticado é trabalhar com casos mais simples, que permitam a identificação de relações e propriedades das variáveis do problema
- Considere o caso de apenas dois estádios circulares com centros P e Q, e que ambos tenham o mesmo raio r



Solução

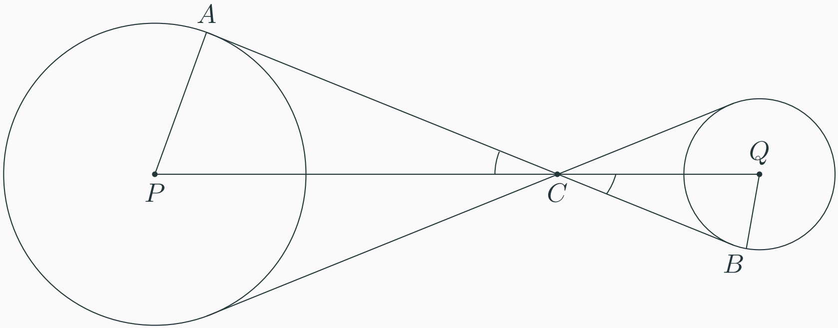
- Os segmentos AP e BQ tem comprimento r
- Os triângulos PAC e QBC são retângulos e congruentes
- Logo, a distância do ponto ideal C aos centros P e Q deve ser a mesma
- O conjunto de pontos que satisfaz a condição $d(P, C) = d(Q, C)$ é a mediatriz do segmento PQ
- O parâmetros da mediatriz são dados por

$$\begin{aligned} 0 &= d^2(P, C) - d^2(Q, C) \\ &= (x - x_P)^2 + (y - y_P)^2 - (x - x_Q)^2 - (y - y_Q)^2 \\ &= -2(x_P - x_Q)x - 2(y_P - y_Q)y + (x_P^2 + y_P^2 - x_Q^2 - y_Q^2) \end{aligned}$$

- Se os raios dos três estádios são idênticos, a solução será a interseção de duas das mediatrizes, se existir

Solução

- Considere agora o caso onde os raios r_P e r_Q são distintos



Solução

- Neste caso, os triângulos PAC e QBC são semelhantes, mas não congruentes
- Da congruência segue que

$$\frac{d(P,C)}{r_P} = \frac{d(Q,C)}{r_Q}$$

- Daí

$$\begin{aligned}
0 &= r_Q^2 d^2(P,C) - r_P^2 d^2(Q,C) \\
&= r_Q^2 \left[(x - x_P)^2 + (y - y_P)^2 \right] - r_P^2 \left[(x - x_Q)^2 + (y - y_Q)^2 \right] \\
&= \left[(r_Q^2 - r_P^2) x^2 - 2x(r_Q^2 x_P - r_P^2 x_Q) \right] \\
&+ \left[(r_Q^2 - r_P^2) y^2 - 2y(r_Q^2 y_P - r_P^2 y_Q) \right] + \left[x_P^2 + y_P^2 - x_Q^2 - y_Q^2 \right]
\end{aligned}$$

Solução

- Seja
$$x_0 = \frac{r_Q^2 x_P - r_P^2 x_Q}{r_Q^2 - r_P^2} \quad \text{e} \quad y_0 = \frac{r_Q^2 y_P - r_P^2 y_Q}{r_Q^2 - r_P^2}$$
- Completando o quadrado em x e em y segue que
$$(x - x_0)^2 + (y - y_0)^2 = R^2,$$
onde
$$R = \frac{r_Q^2(x_P^2 + y_P^2) - r_P^2(x_Q^2 - y_Q^2)}{r_Q^2 - r_P^2} - x_0^2 - y_0^2$$
- Neste caso, a solução será, dentre as duas possíveis interseções entre os círculos correspondentes à dois pares de estádios com raios distintos, a que produz o maior ângulo
- Basta observar que, quando mais próximo o ponto do centro do círculo, maior será o ângulo de observação

Solução AC com complexidade $O(1)$

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

using namespace std;

template<typename T>
struct Point
{
    T x, y;

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

template<typename T>
struct Line
{
    T a, b, c;
};

```

Solução AC com complexidade $O(1)$

```cpp
template<typename T>
struct Circle
{
    Point<T> C;
    T r;
};

template<typename T> vector<Point<T>>
intersection(const Circle<T>& c1, const Circle<T>& c2)
{
    vector<Point<double>> ps;
    double d = hypot(c1.C.x - c2.C.x, c1.C.y - c2.C.y);

    if (d > c1.r + c2.r or d < fabs(c1.r - c2.r))
        return ps;

    // Caso d == 0 ignorado por conta das restrições da entrada
    auto a = (c1.r * c1.r - c2.r * c2.r + d * d)/(2 * d);
    auto h = sqrt(c1.r * c1.r - a * a);
    auto x = c1.C.x + (a/d)*(c2.C.x - c1.C.x);
    auto y = c1.C.y + (a/d)*(c2.C.y - c1.C.y);
```

Solução AC com complexidade $O(1)$

```cpp

    auto P = Point<double> { x, y };

    x = P.x + (h/d)*(c2.C.y - c1.C.y);
    y = P.y - (h/d)*(c2.C.x - c1.C.x);

    ps.push_back( { x, y } );

    x = P.x - (h/d)*(c2.C.y - c1.C.y);
    y = P.y + (h/d)*(c2.C.x - c1.C.x);

    ps.push_back( { x, y } );

    return ps;
}

Circle<double> best_circle(const Circle<int>& P, const Circle<int>& Q)
{
    auto rP2 = (double) P.r * P.r;
    auto rQ2 = (double) Q.r * Q.r;

```

Solução AC com complexidade $O(1)$

```cpp
    auto x = (rQ2*P.C.x - rP2*Q.C.x)/(rQ2 - rP2);
    auto y = (rQ2*P.C.y - rP2*Q.C.y)/(rQ2 - rP2);
    auto K = (rQ2*P.C.x*P.C.x - rP2*Q.C.x*Q.C.x
            + rQ2*P.C.y*P.C.y - rP2*Q.C.y*Q.C.y)/(rQ2 - rP2);

    auto r = sqrt(x*x + y*y - K);

    return { x, y, r };
}


template<typename T>
Point<double> intersection(const Line<T>& r, const Line<T>& s)
{
    auto det = r.a * s.b - r.b * s.a;

    // Caso det == 0 ignorado por conta das condições da entrada

    double x = (double) (-r.c * s.b + s.c * r.b) / det;
    double y = (double) (-s.c * r.a + r.c * s.a) / det;

```

Solução AC com complexidade $O(1)$

```
    return { x, y };
}

template<typename T>
Line<T> perpendicular_bisector(const Point<T>& P, const  Point<T>& Q)
{
    auto a = 2*(Q.x - P.x);
    auto b = 2*(Q.y - P.y);
    auto c = (P.x * P.x + P.y * P.y) - (Q.x * Q.x + Q.y * Q.y);
    return { a, b, c };
}

vector<Point<double>> solve(const vector<Circle<int>>& ps)
{
    vector<Point<double>> ans;
    enum { P, Q, R };

    if (ps[P].r == ps[Q].r and ps[Q].r == ps[R].r) {
        auto r = perpendicular_bisector(ps[P].C, ps[Q].C);
        auto s = perpendicular_bisector(ps[Q].C, ps[R].C);
        ans.push_back(intersection(r, s));
```

Solução AC com complexidade $O(1)$

```
    } else
    {
        vector<Circle<double>> cs;

        if (ps[P].r != ps[Q].r)
            cs.push_back(best_circle(ps[P], ps[Q]));

        if (ps[P].r != ps[R].r)
            cs.push_back(best_circle(ps[P], ps[R]));

        if (ps[Q].r != ps[R].r)
            cs.push_back(best_circle(ps[Q], ps[R]));

        auto qs = intersection(cs[0], cs[1]);

        if (not qs.empty())
        {
            auto A = qs.front();
            auto B = qs.back();
            auto distA = 1e9, distB = 1e9;

```

Solução AC com complexidade $O(1)$

```
            for (int i = 0; i < 3; ++i)
            {
                Point<double> X { (double) ps[i].C.x, (double) ps[i].C.x };

                distA = min(distA, A.distance(X));
                distB = min(distB, B.distance(X));
            }

            distA < distB ? ans.push_back(A) : ans.push_back(B);
        }
    }


    return ans;
}

int main()
{
    vector<Circle<int>> ps;

```

Solução AC com complexidade $O(1)$

```cpp
    for (int i = 0; i < 3; ++i)
    {
        int x, y, r;
        cin >> x >> y >> r;

        ps.push_back(Circle<int> { x, y, r });
    }

    auto ans = solve(ps);

    if (not ans.empty())
        printf("%.5f %.5f\n", ans[0].x, ans[0].y);

    return 0;
}
```

Referências

1. Educational Codeforces Round 2 – Problem D: Area of Two Circles' Intersection
2. Codeforces Beta Round #2 – Problem C: Commentator Problem