Geometria Computacional

Sweep line: problemas resolvidos

Prof. Edson Alves

2019

Faculdade UnB Gama

Sumário

UVA 10245 – The Closest Pair Problem

Problema

Given a set of points in a two dimensional space, you will have to find the distance between the closest two points.

Entrada e saída

Input

The input file contains several sets of input. Each set of input starts with an integer N $(0 \leq N \leq 10000)$, which denotes the number of points in this set. The next N line contains the coordinates of N twodimensional points. The first of the two numbers denotes the X-coordinate and the latter denotes the Y-coordinate. The input is terminated by a set whose $N = 0$. This set should not be processed. The value of the coordinates will be less than 40000 and non-negative.

Output

For each set of input produce a single line of output containing a floating point number (with four digits after the decimal point) which denotes the distance between the closest two points. If there is no such two points in the input whose distance is less than 10000, print the line '`INFINITY`'.

Exemplo de entradas e saídas

Sample Input

```
3
0 0
10000 10000
20000 20000
5
0 2
6 67
43 71
39 107
189 140
0
```

Sample Output

```
INFINITY
36.2215
```

Solução $O(TN \log N)$

- Este é o problema clássico de par de pontos mais próximos
- O algoritmo de *sweep line* para este problema pode ser utiliza, porém é preciso ter cuidado com algumas restrições do problema
- Em primeiro lugar, embora não fique claro no texto, a entrada admite coordenadas em ponto flutuante para os pontos
- Além disso, há o caso especial em que a entrada contém somente um ponto
- Neste caso, a resposta deve ser `INFINITY`
- Isto feito, cada caso de teste pode ser resolvido em $O(N \log N)$

Solução com complexidade $O(TN \log N)$

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

using namespace std;
using ii = pair<double, double>;
using point = pair<double, double>;

#define x first
#define y second

double dist(const point& P, const point& Q)
{
    return hypot(P.x - Q.x, P.y - Q.y);
}

double solve(int N, vector<point>& ps)
{
    sort(ps.begin(), ps.end());

    if (N == 1)
        return -1;

```

Solução com complexidade $O(TN \log N)$

```
    auto d = dist(ps[0], ps[1]);

    set<ii> S;
    S.insert(ii(ps[0].y, ps[0].x));
    S.insert(ii(ps[1].y, ps[1].x));

    for (int i = 2; i < N; ++i)
    {
        auto P = ps[i];
        auto it = S.lower_bound(point(P.y - d, 0));

        while (it != S.end())
        {
            auto Q = point(it->second, it->first);

            if (Q.x < P.x - d)
            {
                it = S.erase(it);
                continue;
            }

```

Solução com complexidade $O(TN \log N)$

```
            if (Q.y > P.y + d)
                break;

            auto t = dist(P, Q);

            if (t < d)
                d = t;

            ++it;
        }

        S.insert(ii(P.y, P.x));
    }

    return d < 10000 ? d : -1;
}

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

Solução com complexidade $O(TN \log N)$

```

    while (cin >> N, N)
    {
        vector<point> ps(N);

        for (int i = 0; i < N; ++i)
            cin >> ps[i].x >> ps[i].y;

        auto ans = solve(N, ps);

        if (ans < 0)
            cout << "INFINITY\n";
        else
        {
            cout.precision(4);
            cout << fixed << ans << '\n';
        }
    }

    return 0;
}
```

Codeforces Round #329 (Div. 2) – Problem B: Anton and Lines

Problema

The teacher gave Anton a large geometry homework, but he didn't do it (as usual) as he participated in a regular round on Codeforces. In the task he was given a set of n lines defined by the equations $y = k_i x + b_i$. It was necessary to determine whether there is at least one point of intersection of two of these lines, that lays strictly inside the strip between $x_1 < x_2$. In other words, is it true that there are $1 \leq i < j \leq n$ and x', y', such that:

- $y' = k_i x' + b_i$, that is, point (x', y') belongs to the line number i;
- $y' = k_j x' + b_j$, that is, point (x', y') belongs to the line number j;
- $x_1 < x' < x_2$, that is, point (x', y') lies inside the strip bounded by $x_1 < x_2$.

You can't leave Anton in trouble, can you? Write a program that solves the given task.

Entrada e saída

Input

The first line of the input contains an integer n $(2 \leq n \leq 100000)$ – the number of lines in the task given to Anton. The second line contains integers x_1 and x_2 $(-1000000 \leq x_1 < x_2 \leq 1000000)$ defining the strip inside which you need to find a point of intersection of at least two lines.

The following n lines contain integers k_i, b_i $(-1000000 \leq k_i, b_i \leq 1000000)$ – the descriptions of the lines. It is guaranteed that all lines are pairwise distinct, that is, for any two $i \neq j$ it is true that either $k_i \neq k_j$, or $b_i \neq b_j$.

Output

Print "Yes"(without quotes), if there is at least one intersection of two distinct lines, located strictly inside the strip. Otherwise print "No"(without quotes).

Exemplo de entradas e saídas

Sample Input

```
4
1 2
1 2
1 0
0 1
0 2

2
1 3
1 0
-1 3
```

Sample Output

```
YES


NO
```

Solução com complexidade $O(N \log N)$

- A busca completa, que verifica todos os pares de segmentos de reta, tem complexidade $O(N^2)$, o que leva ao TLE, pois $N \leq 10^5$
- Contudo, é possível determinar as possíveis interseções no intervalo (x_1, x_2) com um algoritmo *sweep line*
- Os segmentos devem ser ordenados, em ordem crescente, pelo valor da coordenada y do segmento no ponto x_1
- Em caso de empate, deve-se ordenar pela coordena y no ponto x_2
- Cada segmento deve ser processado uma única vez, nesta ordem
- Deve-se manter o registro da maior coordenada y em x_2 já encontrada (inicialmente, este valor deve ser igual a $-\infty$
- Se a coordenada y em x_2 do segmento a ser processado for menor do que a maior já encontrada, significa que houve uma interseção com algum dos segmentos já processados

Solução com complexidade $O(N \log N)$

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

using namespace std;
using ll = long long;

const ll oo { 1000000000000000000LL };

struct Line
{
    ll k, b;

    ll eval(ll x) const { return k*x + b; }
};

bool solve(ll x1, ll x2, vector<Line>& lines)
{
    // Ordenação por coordenada y em x1, depois em x2
    sort(lines.begin(), lines.end(), [&](const Line& r, const Line& s) {
        if (r.eval(x1) != s.eval(x1))
            return r.eval(x1) < s.eval(x1);

```

Solução com complexidade $O(N \log N)$

```cpp
            return r.eval(x2) < s.eval(x2);
    });

    auto max_y = -oo;

    for (const auto& r : lines)
    {
        auto y = r.eval(x2);

        if (y < max_y)
            return true;

        max_y = max(y, max_y);
    }

    return false;
}

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

Solução com complexidade $O(N \log N)$

```cpp

    int n;
    cin >> n;

    int x1, x2;
    cin >> x1 >> x2;

    vector<Line> lines(n);

    for (int i = 0; i < n; ++i)
        cin >> lines[i].k >> lines[i].b;

    auto ans = solve(x1, x2, lines);

    cout << (ans ? "Yes" : "No") << '\n';

    return 0;
}
```

Referências

1. UVA 10245 – The Closest Pair Problem
2. Codeforces Round #329 (Div. 2) – Problem B: Anton and Lines