Geometria Computacional

Pontos: problemas resolvidos

Prof. Edson Alves

2018

Faculdade UnB Gama

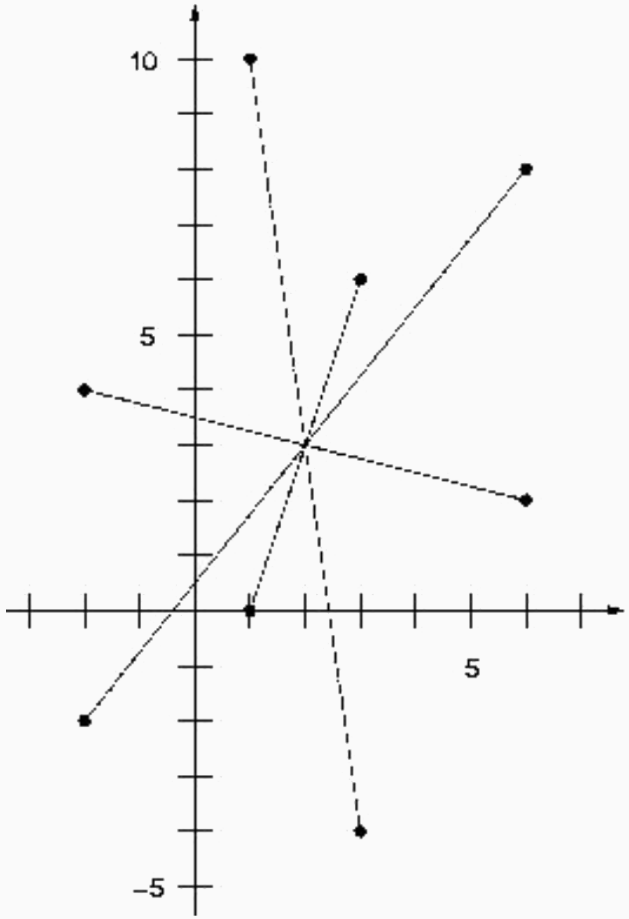
Sumário

UVA 10585 – Center of symmetry

Problema

Given is a set of n points with integer coordinates. Your task is to decide whether the set has a center of symmetry.

A set of points S has the center of symmetry if there exists a point s (not necessarily in S) such that for every point p in S there exists a point q in S such that $p - s = s - q$.



Entrada e saída

Input

The first line of input contains a number c giving the number of cases that follow. The first line of data for a single case contains number $1 \leq n \leq 10000$. The subsequent n lines contain two integer numbers each which are the x and y coordinates of a point. Every point is unique and we have that $-10000000 \leq x, y \leq 10000000$.

Output

For each set of input data print 'yes' if the set of points has a center of symmetry and 'no' otherwise.

Exemplo de entradas e saídas

Sample Input

```
1
8
1 10
3 6
6 8
6 2
3 -4
1 0
-2 -2
-2 4
```

Sample Output

```
yes
```

Solução por força bruta

- Manipulando a expressão $p - s = s - q$ obtemos

$$s = \frac{p+q}{2},$$

  ou seja, o centro de simetria é o ponto médio de p e q
- O uso de ponto flutuante pode ser evitado se usarmos a expressão

$$2s = p + q,$$

  e trabalharmos com o dobro do centro de simetria
- A solução de força bruta consiste em fixar um ponto A em S e, para todos os pontos B em S, computar $2s$
- Agora, para todos os pontos p em S, calculamos $q = 2s - p$ e verificamos se q pertence a S ou não
- Usando uma estrutura `set`, que permite verificar se q pertence a S em $O(\log n)$, temos uma solução $O(n^2 \log n)$
- Como $n \leq 10^4$, esta solução pode dar TLE ou AC, a depender da velocidade do servidor

Solução AC/TLE com complexidade $O(n^2 \log n)$

```cpp
#include <iostream>
#include <set>

struct Point {
    int x, y;

    bool operator<(const Point& p) const
    {
        return x == p.x ? y < p.y : x < p.x;
    }
};

bool has_center_of_symmetry(const std::set<Point>& S)
{
    auto A = *S.begin();        // Ponto qualquer de S fixo

    for (auto& B : S)
    {
        // Candidato à centro de simetria
        auto _2s = Point { A.x + B.x, A.y + B.y };
        bool ok = true;
```

Solução AC/TLE com complexidade $O(n^2 \log n)$

```
22
23          // Verifica se o candidato atende todos os pontos de S
24          for (auto& p : S)
25          {
26              auto q = Point { _2s.x - p.x, _2s.y - p.y };
27
28              if (S.count(q) == 0)
29              {
30                  ok = false;
31                  break;
32              }
33          }
34
35          if (ok)
36              return true;
37      }
38
39      return false;
40  }
41
```

Solução AC/TLE com complexidade $O(n^2 \log n)$

```
42 int main()
43 {
44     int c, n;
45     std::cin >> c;
46 
47     while (c--) {
48         std::cin >> n;
49 
50         std::set<Point> S;
51 
52         while (n--) {
53             int x, y;
54             std::cin >> x >> y;
55             S.insert(Point { x, y });
56         }
57 
58         std::cout << (has_center_of_symmetry(S) ? "yes" : "no") << '\n';
59     }
60 
61     return 0;
62 }
```

Solução mais eficiente

- Para reduzir a complexidade assintótica da solução, é preciso investigar as propriedades do problema
- Suponha que o centro de simetria s pertença ao conjunto S. Então fazendo $p = s$ na expressão $p - s = s - q$ temos que $q = s$, ou seja, o ponto de simetria fica pareado consigo mesmo
- Como todos os pontos são distintos, então se $p \neq s$ então $q \neq s$, isto é, os pontos são pareados dois a dois
- Deste modo, se existir, s só estará em S se n for ímpar
- Por um breve momento, vamos pensar no caso especial onde todos os pontos tem coordenada y igual a zero
- Considere agora p o ponto com menor coordenada x
- Neste cenário, podemos observar que q deve ser, obrigatoriamente, o ponto com maior coordenada x

Solução mais eficiente

- De fato, seja $r \neq q$ o ponto de maior coordenada x. Então r deve parear com um ponto t com coordenada maior do que x (pois os pontos são todos distintos), de modo que teremos
$$\frac{x_r + x_t}{2} > \frac{x_p + x_q}{2},$$
pois $x_p < x_t$ e $x_q < x_r$, o que impossibilita a existência de um centro de simetria
- Assim, se os pontos estiverem ordenados, o primeiro deve parear com o último, de modo que é necessário verificar apenas um único candidato
- Porém o fato acima foi deduzido para pontos sobre o eixo-x
- Contudo, é fácil estender o resultado para duas dimensões: uma vez ordenados por coordenada x, o ponto com menor coordenada x e menor coordenada y deve parear com o ponto com maior coordenada x e maior coordenada y, pelo mesmo motivo já apresentado
- Assim, a solução passa a ter complexidade $O(n \log n)$

Solução AC com complexidade $O(n \log n)$

```cpp
#include <iostream>
#include <set>

struct Point {
    int x, y;

    bool operator<(const Point& p) const
    {
        return x == p.x ? y < p.y : x < p.x;
    }
};

```

Solução AC com complexidade $O(n \log n)$

```cpp
bool has_center_of_symmetry(const std::set<Point>& S)
{
    auto A = *S.begin();    // Primeiro ponto, após ordenados
    auto B = *S.rbegin();   // Último ponto, após ordenados

    // Candidato à centro de simetria
    auto _2s = Point { A.x + B.x, A.y + B.y };

    // Verifica se o candidato atende todos os pontos de S
    for (auto& p : S)
    {
        auto q = Point { _2s.x - p.x, _2s.y - p.y };

        if (S.count(q) == 0)
            return false;
    }

    return true;
}

```

Solução AC com complexidade $O(n \log n)$

```
33 int main()
34 {
35     int c, n;
36     std::cin >> c;
37 
38     while (c--) {
39         std::cin >> n;
40 
41         std::set<Point> S;
42 
43         while (n--) {
44             int x, y;
45             std::cin >> x >> y;
46             S.insert(Point { x, y });
47         }
48 
49         std::cout << (has_center_of_symmetry(S) ? "yes" : "no") << '\n';
50     }
51 
52     return 0;
53 }
```

Referências

1. UVA 10585 – *Center of Symmetry*