OJ 10245

The Closest Pair Problem

Prof. Edson Alves

Faculdade UnB Gama

OJ 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

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

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

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

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`

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 $O(TN \log N)$

```
11 double dist(const Point& P, const Point& Q)
12 {
13     return hypot(P.x - Q.x, P.y - Q.y);
14 }
15
16 double solve(int N, vector<Point>& ps)
17 {
18     sort(ps.begin(), ps.end());
19
20     if (N == 1)
21         return -1;
22
23     auto d = dist(ps[0], ps[1]);
24
25     set<ii> S;
26     S.insert(ii(ps[0].y, ps[0].x));
27     S.insert(ii(ps[1].y, ps[1].x));
```

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

```
29      for (int i = 2; i < N; ++i)
30      {
31          auto P = ps[i];
32          auto it = S.lower_bound(Point(P.y - d, 0));
33
34          while (it != S.end())
35          {
36              auto Q = Point(it->second, it->first);
37
38              if (Q.x < P.x - d)
39              {
40                  it = S.erase(it);
41                  continue;
42              }
43
44              if (Q.y > P.y + d)
45                  break;
46
47              auto t = dist(P, Q);
```

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

```
49              if (t < d)
50                  d = t;
51
52              ++it;
53          }
54
55          S.insert(ii(P.y, P.x));
56      }
57
58      return d < 10000 ? d : -1;
59  }
```