Educational Codeforces Round 1

Problem C: Nearest vectors

Prof. Edson Alves

Faculdade UnB Gama

Educational Codeforces Round 1 – Problem C: Nearest vectors

Problema

É dado um conjunto de vetores no plano, cada um deles partindo da origem. Sua tarefa é encontrar um par de vetores com o ângulo não-orientado mínimo entre eles.

Um ângulo não-orientado é um valor não-negativo, mínimo entre os ângulos nos sentidos horário e anti-horário. Um ângulo não-orientado está sempre entre 0 e π. For exemplo, vetores com direções opostas tem ângulo igual a π.

Entrada e saída

Entrada

A primeira linha da entrada contém um único inteiro n ($2 \le n \le 100000$) – o número de vetores.

A i-ésima das n linhas seguintes contém dois inteiros x_i e y_i ($|x|, |y| \le 10000, x^2 + y^2 > 0$) – as coordenadas do i-ésimo vetor. Os vetores são numerados de 1 a n na ordem em que aparecem na entreada. É garantido que nenhum par de vetores compartilha a mesma direção (mas eles ainda podem ter direções opostas).

Saída

Imrpima dois números a e b ($a \neq b$) – um par de índices de vetores com ângulo não-orientado mínimo. Você pode imprimir os números em qualquer ordem. Se há múltiplas soluções possíveis, imprima qualquer uma delas.

Exemplo de entradas e saídas

Entrada	Saída
4 -1 0 0 -1 1 0 1 1 6 -1 0 0 -1 1 0 1 1 -4 -5 -4 -6	3 4 6 5

Observações sobre o problema

- O ângulo que um vetor faz com o eixo-x positivo pode ser computado com a função `atan2()` da biblioteca de matemática do C/C++

Observações sobre o problema

- O ângulo que um vetor faz com o eixo-x positivo pode ser computado com a função `atan2()` da biblioteca de matemática do C/C++
- É preciso observa que o retorno da função está no intervalo $[-\pi, \pi]$

Observações sobre o problema

- O ângulo que um vetor faz com o eixo-x positivo pode ser computado com a função `atan2()` da biblioteca de matemática do C/C++
- É preciso observa que o retorno da função está no intervalo $[-\pi, \pi]$
- Uma vez ordenados os vetores por ângulo, basta computar o ângulo entre dois vetores consecutivos usando a diferença

Observações sobre o problema

- O ângulo que um vetor faz com o eixo-x positivo pode ser computado com a função `atan2()` da biblioteca de matemática do C/C++
- É preciso observa que o retorno da função está no intervalo $[-\pi, \pi]$
- Uma vez ordenados os vetores por ângulo, basta computar o ângulo entre dois vetores consecutivos usando a diferença
- Por conta do retorno da função `atan2()`, esta diferença pode ser negativa

Observações sobre o problema

- O ângulo que um vetor faz com o eixo-x positivo pode ser computado com a função `atan2()` da biblioteca de matemática do C/C++
- É preciso observa que o retorno da função está no intervalo $[-\pi, \pi]$
- Uma vez ordenados os vetores por ângulo, basta computar o ângulo entre dois vetores consecutivos usando a diferença
- Por conta do retorno da função `atan2()`, esta diferença pode ser negativa
- Caso isto aconteça, basta somar 2π ao resultado

Observações sobre o problema

- O ângulo que um vetor faz com o eixo-x positivo pode ser computado com a função `atan2()` da biblioteca de matemática do C/C++
- É preciso observa que o retorno da função está no intervalo $[-\pi, \pi]$
- Uma vez ordenados os vetores por ângulo, basta computar o ângulo entre dois vetores consecutivos usando a diferença
- Por conta do retorno da função `atan2()`, esta diferença pode ser negativa
- Caso isto aconteça, basta somar 2π ao resultado
- O valor de π pode ser computado através da expressão `acos(-1.0)`

Observações sobre o problema

- O ângulo que um vetor faz com o eixo-x positivo pode ser computado com a função `atan2()` da biblioteca de matemática do C/C++
- É preciso observa que o retorno da função está no intervalo $[-\pi, \pi]$
- Uma vez ordenados os vetores por ângulo, basta computar o ângulo entre dois vetores consecutivos usando a diferença
- Por conta do retorno da função `atan2()`, esta diferença pode ser negativa
- Caso isto aconteça, basta somar 2π ao resultado
- O valor de π pode ser computado através da expressão `acos(-1.0)`
- Por fim, é preciso usar o tipo **`long double`**, caso contrário o veredito será WA, por conta da precisão

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

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

using ii = std::pair<int, int>;

struct Vector
{
    int x, y, idx;
    long double angle;

    Vector(int xv, int yv, int i)
        : x(xv), y(yv), idx(i), angle(atan2l(y, x)) {}

    bool operator<(const Vector& v) const
    {
        return angle < v.angle;
    }
};
```

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

```
ii solve(std::vector<Vector>& vs, int N) {
    sort(vs.begin(), vs.end());
    vs.push_back(vs.front());

    long double min_angle = 10.0;
    ii ans;

    for (int i = 0; i < N; ++i) {
        auto angle = vs[i + 1].angle - vs[i].angle;

        if (angle < 0) angle += 2*acosl(-1.0);

        if (angle < min_angle) {
            min_angle = angle;
            ans = ii(vs[i].idx, vs[i+1].idx);
        }
    }

    return ans;
}
```

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

```
40 int main() {
41     std::ios::sync_with_stdio(false);
42
43     int N;
44     std::cin >> N;
45
46     std::vector<Vector> vs;
47
48     for (int i = 1; i <= N; ++i) {
49         int x, y;
50         std::cin >> x >> y;
51         vs.emplace_back(x, y, i);
52     }
53
54     auto [i, j] = solve(vs, N);
55     std::cout << i << ' ' << j << '\n';
56
57     return 0;
58 }
```

Solução AC usando aritmética inteira estendida

```
1  from math import *
2
3
4  class Vector:
5
6      def __init__(self, x, y, label):
7          self.x = x
8          self.y = y
9          self.label = label
10
11         if x >= 0 and y > 0:
12             self.quad = 1;
13         elif x < 0 and y >= 0:
14             self.quad = 2;
15         elif x <= 0 and y < 0:
16             self.quad = 3;
17         else:
18             self.quad = 4;
19
```

Solução AC usando aritmética inteira estendida

```
20      def __lt__(self, v):
21          if self.quad != v.quad:
22              return self.quad < v.quad
23
24          return self.y * v.x < self.x * v.y
25
26
27  def non_oriented_angle(a, b):
28
29      num = a.x * b.x + a.y * b.y
30      signal = 1
31
32      if num < 0:
33          signal = -1;
34
35      d1 = a.x * a.x + a.y * a.y
36      d2 = b.x * b.x + b.y * b.y
37
38      return (signal * num * num, d1 * d2)
```

Solução AC usando aritmética inteira estendida

```
41 def solve(vs, N):
42
43     vs.sort()
44     vs.append(vs[0])
45
46     min_angle = (-10, 1)
47     a = -1
48     b = -1
49
50     for i in xrange(N):
51         angle = non_oriented_angle(vs[i], vs[i + 1])
52
53         if angle[0] * min_angle[1] > angle[1] * min_angle[0]:
54             min_angle = angle
55             a = vs[i].label
56             b = vs[i + 1].label
57
58     print '{} {}'.format(a, b)
```

Solução AC usando aritmética inteira estendida

```
61 if __name__ == '__main__':
62
63     N = int(raw_input())
64
65     vs = []
66
67     for i in xrange(N):
68         x, y = [int(k) for k in raw_input().split()]
69         vs.append(Vector(x, y, i + 1))
70
71     solve(vs, N)
```