OJ 10687

Monitoring the Amazon

Prof. Edson Alves

Faculdade UnB Gama

A network of autonomous, battery-powered, data acquisition stations has been installed to monitor the climate in the region of Amazon. An order-dispatch station can initiate transmission of instructions to the control stations so that they change their current parameters. To avoid overloading the battery, each station (including the order-dispatch station) can only transmit to two other stations. The destinataries of a station are the two closest stations. In case of draw, the first criterion is to chose the westernmost (leftmost on the map), and the second criterion is to chose the southernmost (lowest on the map).

You are commissioned by Amazon State Government to write a program that decides if, given the localization of each station, messages can reach all stations.

Uma rede de estações de aquisição de dados autônomas, alimentadas por baterias, foram instaladas para monitorar o clima na região da Amazônia. Uma estação de controle pode iniciar a transição de instruções para as demais estações para que elas alterem seus parâmetros atuais. Para evitar a sobrecarga da bateria, cada estação (incluindo a estação de controle) podem transmitir informações para apenas duas outras estações. Os destinatários de uma estação são as duas estações mais próximas. Em caso de empate, o primeiro critério é escolher a estação mais à oeste (esquerda no mapa), e o segundo critério é escolher a estação mais ao sul (parte inferior do mapa).

Você foi contratado pelo Governo do Estado do Amazonas para escrever um programa que decide se, dadas as localizações de cada estação, mensagens podem alcançar todas as estações.

Input

The input consists of an integer N, followed by N pairs of integers X_i, Y_i, indicating the localization coordinates of each station. The first pair of coordinates determines the position of the order-dispatch station, while the remaining $N-1$ pairs are the coordinates of the other stations. The following constraints are imposed: $-20 \leq X_i, Y_i \leq 20$, and $1 \leq N \leq 1000$. The input is terminated with $N = 0$.

Output

For each given expression, the output will echo a line with the indicating if all stations can be reached or not (see sample output for the exact format).

Entrada

A entrada consiste em um inteiro N, seguido por N pares de inteiros X_i, Y_i, indicando as coordenadas de localização de cada estação. O primeiro par de coordenadas determina a posição da estação de controle, enquaneo que os $N-1$ pares restantes são as coordenadas das outras estações. São impostas as seguintes restrições: $-20 \leq X_i, Y_i \leq 20$ e $1 \leq N \leq 1000$. A entrada terminada com $N = 0$.

Saída

Para cada expressão dada a saída imprima uma linha indicando se todas as estações podem ser atingidas ou não (veja o exemplo para o formato exato da saída).

Exemplo de entrada e saída

Exemplo de entrada e saída

```
4
```

Exemplo de entrada e saída

4 ⟵ *# de estações*

Exemplo de entrada e saída

```
4
1 0 0 1 -1 0 0 -1
```

Exemplo de entrada e saída

```
4
1 0 0 1 -1 0 0 -1
```

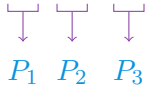
P_1

Exemplo de entrada e saída

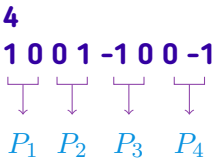
```
4
1 0 0 1 -1 0 0 -1
```

Exemplo de entrada e saída

```
4
1 0 0 1 -1 0 0 -1
```

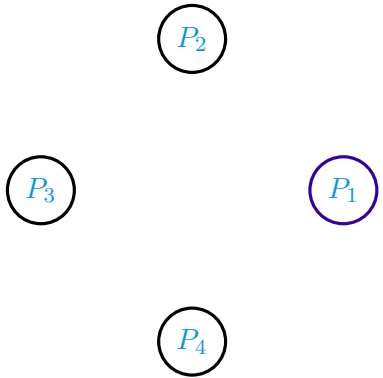


Exemplo de entrada e saída



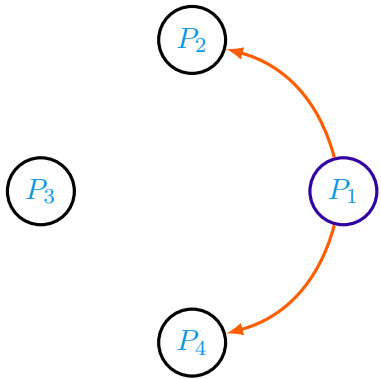
Exemplo de entrada e saída

```
4
1 0 0 1 -1 0 0 -1
```



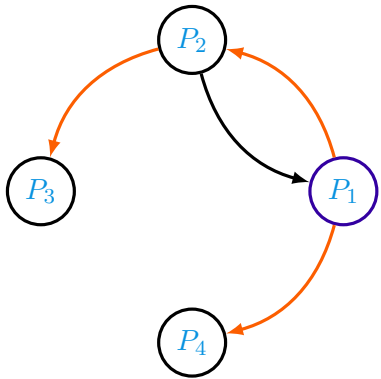
Exemplo de entrada e saída

```
4
1 0 0 1 -1 0 0 -1
```



Exemplo de entrada e saída

```
4
1 0 0 1 -1 0 0 -1
```

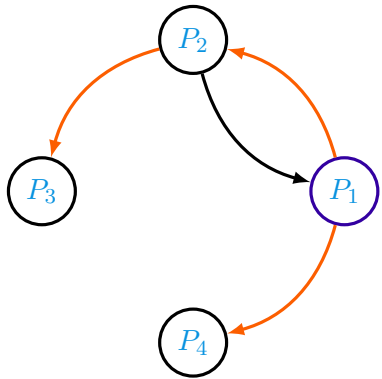


Exemplo de entrada e saída

```
4
1 0 0 1 -1 0 0 -1
```

↓

Ok



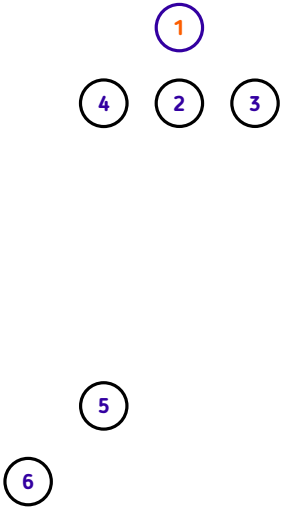
Exemplo de entrada e saída

Exemplo de entrada e saída

```
6
0 3 0 4 1 3 -1 3 -1 -4 -2 -5
```

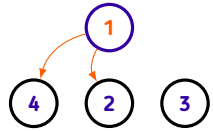
Exemplo de entrada e saída

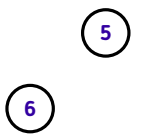
```
6
0 3 0 4 1 3 -1 3 -1 -4 -2 -5
```



Exemplo de entrada e saída

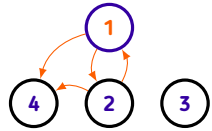
6
0 3 0 4 1 3 -1 3 -1 -4 -2 -5





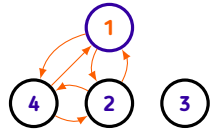
Exemplo de entrada e saída

6
0 3 0 4 1 3 -1 3 -1 -4 -2 -5



Exemplo de entrada e saída

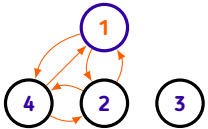
6
0 3 0 4 1 3 -1 3 -1 -4 -2 -5

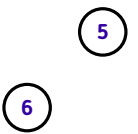


Exemplo de entrada e saída

6
0 3 0 4 1 3 -1 3 -1 -4 -2 -5

x





Solução

Solução

- ⋆ **Adjacência de u:** (d^2, x, y, v)

Solução

- ⋆ **Adjacência de u:** (d^2, x, y, v)
- ⋆ **Critério do problema = ordenação lexicográfica**

Solução

- ⋆ **Adjacência de u:** (d^2, x, y, v)
- ⋆ **Critério do problema = ordenação lexicográfica**
- ⋆ **BFS pode ser usada para identificar os vértices atingíveis**

Solução

- ⋆ **Adjacência de u:** (d^2, x, y, v)
- ⋆ **Critério do problema = ordenação lexicográfica**
- ⋆ **BFS pode ser usada para identificar os vértices atingíveis**
- ⋆ **Complexidade:** $O(N^2)$ **por caso de teste**

```
void bfs(int s)
{
    queue<int> q;

    q.push(s);
    visited[s] = true;

    while (not q.empty())
    {
        auto u = q.front(); q.pop();

        for (auto [d2, x, y, v] : adj[u])
        {
            if (not visited[v]) {
                visited[v] = true;
                q.push(v);
            }
        }
    }
}
```

```
bool solve(int N, const vector<ii>& ps)
{
    for (int p = 1; p <= N; ++p)
        adj[p].clear();

    for (int p = 1; p <= N; ++p)
    {
        auto [px, py] = ps[p];

        for (int q = p + 1; q <= N; ++q)
        {
            auto [qx, qy] = ps[q];
            auto d2 = (px - qx)*(px - qx) + (py - qy)*(py - qy);

            adj[p].insert(edge { d2, qx, qy, q });
            adj[q].insert(edge { d2, px, py, p });
```

```
            if (adj[p].size() > 2)
                adj[p].erase(prev(adj[p].end()));

            if (adj[q].size() > 2)
                adj[q].erase(prev(adj[q].end()));
        }
    }

    visited.reset();
    bfs(1);

    return (int) visited.count() == N;
}
```