OJ 10000

Longest Paths

Prof. Edson Alves

Faculdade UnB Gama

It is a well known fact that some people do not have their social abilities completely enabled. One example is the lack of talent for calculating distances and intervals of time. This causes some people to always choose the longest way to go from one place to another, with the consequence that they are late to whatever appointments they have, including weddings and programming contests. This can be highly annoying for their friends.

César has this kind of problem. When he has to go from one point to another he realizes that he has to visit many people, and thus always chooses the longest path. One of César's friends, Felipe, has understood the nature of the problem. Felipe thinks that with the help of a computer he might be able to calculate the time that César is going to need to arrive to his destination. That way he could spend his time in something more enjoyable than waiting for César.

É bem conhecido o fato que algumas pessoas não tem suas habilidades sociais plenamente desenvolvidas. Um exemplo é a falta de talento para calcular distâncias e intervalos de tempo. Isto faz com que algumas pessoas sempre escolham o caminho mais longo de um lugar a outro, e assim elas chegam atrasadas em quaisquer compromissos que elas tenham, incluindo casamentos e maratonas de programação. Este comportamento pode ser irritante para seus amigos.

César tem este tipo de problema. Quando ele precisa ir de um ponto ao outro ele entende que ele ainda tem que visitar muitas pessoas, e assim ele escolhe o caminho mais longo. Um dos amigos de César, Felipe, entendeu a natureza do problema. Felipe imagina que, com a a ajuda de um computador, ele será capaz de calcular o tempo que César levará para chegar ao seu destino. Deste modo ele poderá passar seu tempo fazendo algo mais divertido do que ficar esperando por César.

Your goal is to help Felipe developing a program that computes the length of the longest path that can be constructed in a given graph from a given starting point (César's residence). You can assume that there is at least one path from this starting point to any destination. Also, the graph has no cycles (there is no path from any node to itself), so César will reach his destination in a finite time. In the same line of reasoning, nodes are not considered directly connected to themselves.

Seu objetivo é ajudar Felipe, escrevendo um programa que calcula o comprimento do maior caminho em um grafo a partir de um dado ponto de partida (a casa de César). Você pode assumir que há no mínimo um caminho do ponto de partida a qualquer destino e também que o grafo não tem ciclos (não há um caminho de qualquer nó a si mesmo), de modo que César chegará ao seu destino em um espaço de tempo finito. Na mesma linha de raciocínio, considere que um nó não esteja conectado a si mesmo.

Input

The input consists of a number of cases. The first line on each case contains a positive number n $(1 < n \leq 100)$ that specifies the number of points that César might visit (i.e., the number of nodes in the graph).

A value of $n = 0$ indicates the end of the input.

After this, a second number s is provided, indicating the starting point in César's journey $(1 \leq s \leq n)$. Then, you are given a list of pairs of places p and q, one pair per line, with the places on each line separated by white-space. The pair "p q" indicates that César can visit q after p.

Entrada

A entrada é composta por uma série de casos de teste. A primeira linha de cada caso contém um inteiro positivo n $(1 < n \leq 100)$ que indica o número de pontos que César pode visitar (isto é, o número de nós no grafo).

O valor $n = 0$ indica o fim da entrada.

A linha seguinte contém um inteiro s, indicando o ponto de partida da jornada de César $(1 \leq s \leq n)$. Então, há uma lista de pares de localidades p e q, um par por linha, com as localizações separadas por um espaço em branco. O par "p q" indica que César pode visitar q após p.

Input

A pair of zeros (‘`0 0`’) indicates the end of the case.

As mentioned before, you can assume that the graphs provided will not be cyclic and that every place will be reachable from the starting place.

Output

For each test case you have to find the length of the longest path that begins at the starting place. You also have to print the number of the final place of such longest path. If there are several paths of maximum length, print the final place with smallest number.

Print a new line after each test case.

Entrada

Um par de zeros (‘0 0’) indica o fim do caso de teste.

Como dito antes, você pode assumir que o grafo em questão não tem ciclos e que qualquer localidade pode ser alcançada saindo do ponto de partida.

Saída

Para cada caso de teste você deve determinar o comprimento do maior caminho que inicia no ponto de partida. Você também deve imprimir o número da localidade que finaliza tal caminho. Se há vários caminhos de comprimento máximo, imprima a localização final de menor identificador.

Imprima uma nova linha após cada caso de teste.

Exemplo de entrada e saída

Exemplo de entrada e saída

```
2
```

Exemplo de entrada e saída

2 ⟵ *# de localidades*

Exemplo de entrada e saída

2 ⟵ *# de localidades*

Exemplo de entrada e saída

2

Exemplo de entrada e saída

```
2
1
```

Exemplo de entrada e saída

2
1 ⟵ *ponto de partida*

Exemplo de entrada e saída

```
2
1 ⟵ ponto de partida
```

Exemplo de entrada e saída

```
2
1
```

Exemplo de entrada e saída

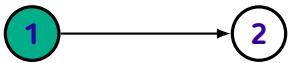
```
2
1
1 2
```

Exemplo de entrada e saída

```
2
1
1 2 ⟵ conexão entre localidades
```

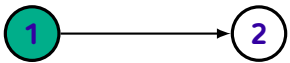
Exemplo de entrada e saída

```
2
1
1 2 ⟵ conexão entre localidades
```



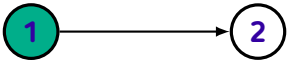
Exemplo de entrada e saída

```
2
1
1 2
```



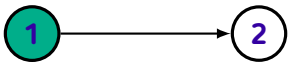
Exemplo de entrada e saída

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



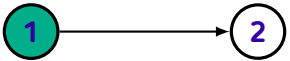
Exemplo de entrada e saída

```
2
1
1 2
0 0 ⟵ fim do caso de teste
```



Exemplo de entrada e saída

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



Exemplo de entrada e saída

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

Exemplo de entrada e saída

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

1, 2

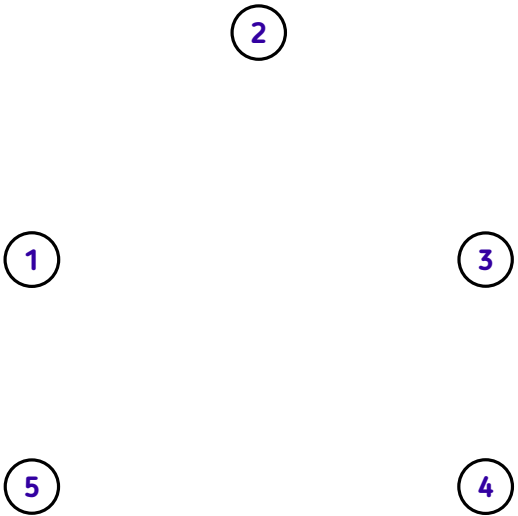
Exemplo de entrada e saída

Exemplo de entrada e saída

```
5
```

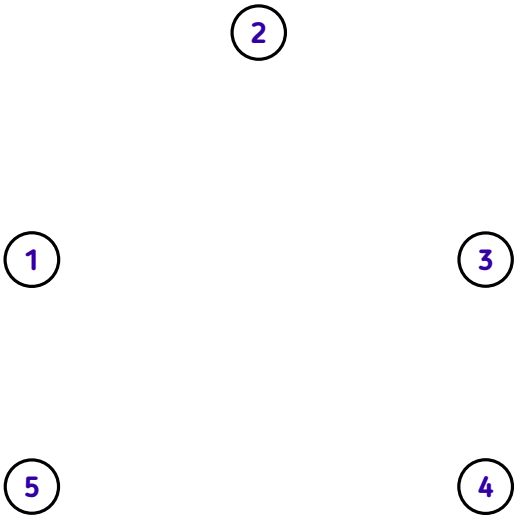
Exemplo de entrada e saída

5



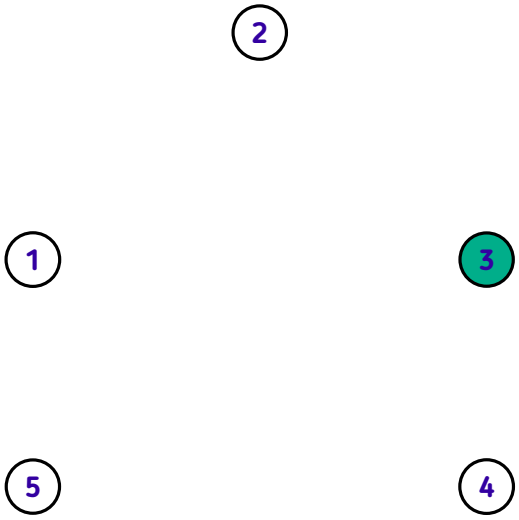
Exemplo de entrada e saída

```
5
3
```



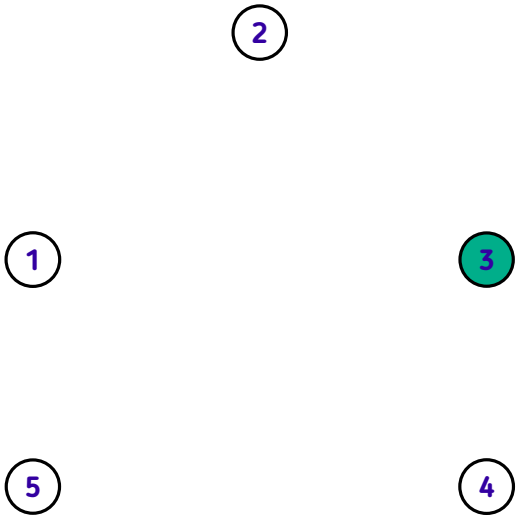
Exemplo de entrada e saída

```
5
3
```



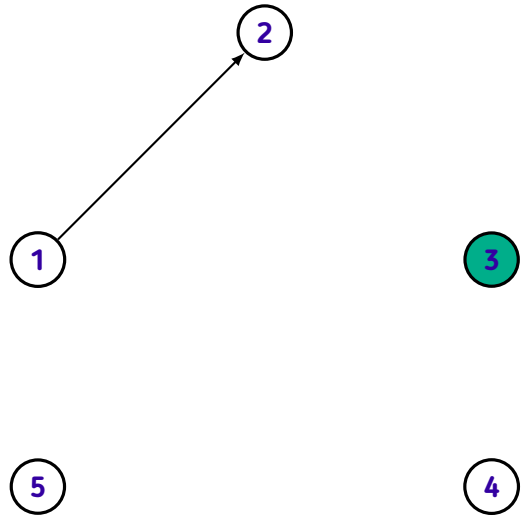
Exemplo de entrada e saída

```
5
3
1 2
```



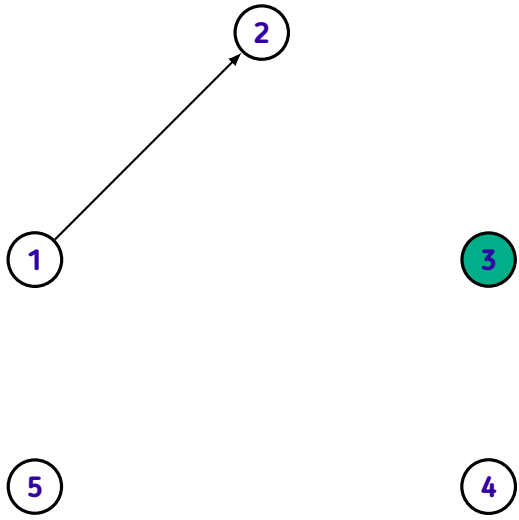
Exemplo de entrada e saída

5
3
1 2



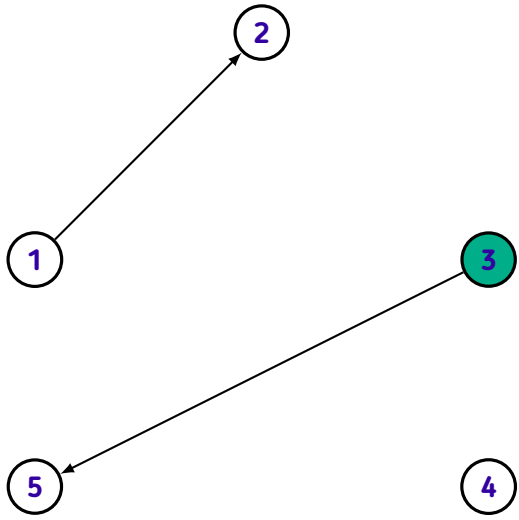
Exemplo de entrada e saída

5
3
1 2
3 5



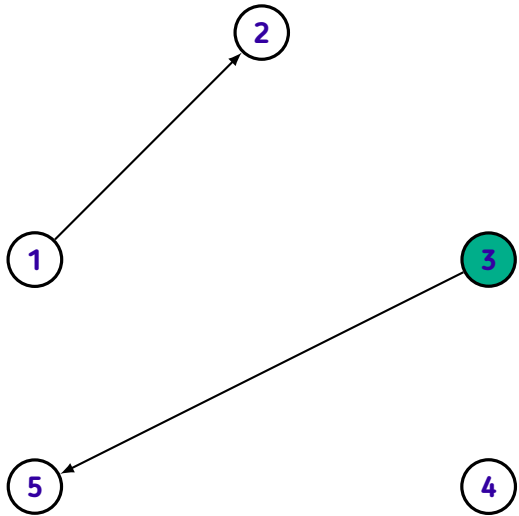
Exemplo de entrada e saída

5
3
1 2
3 5



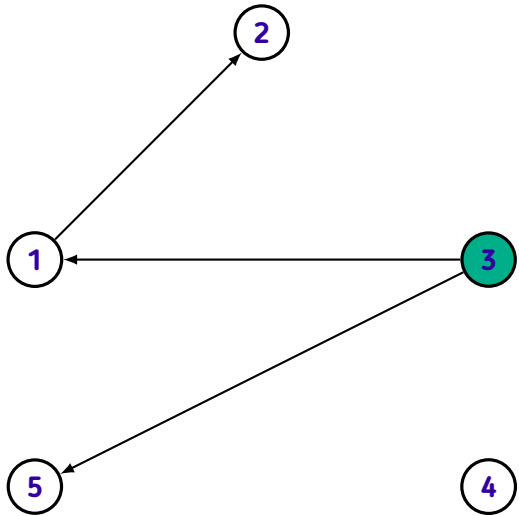
Exemplo de entrada e saída

```
5
3
1 2
3 5
3 1
```



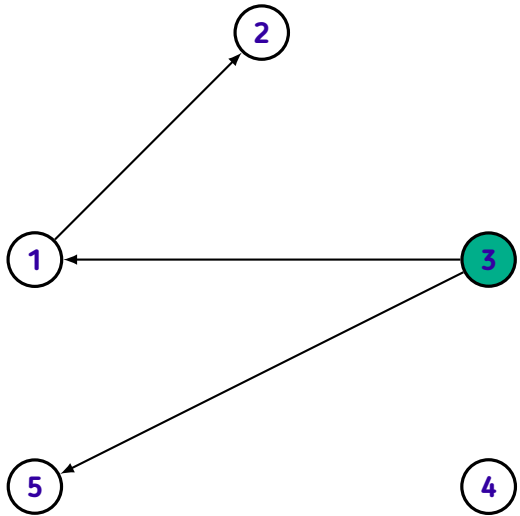
Exemplo de entrada e saída

```
5
3
1 2
3 5
3 1
```



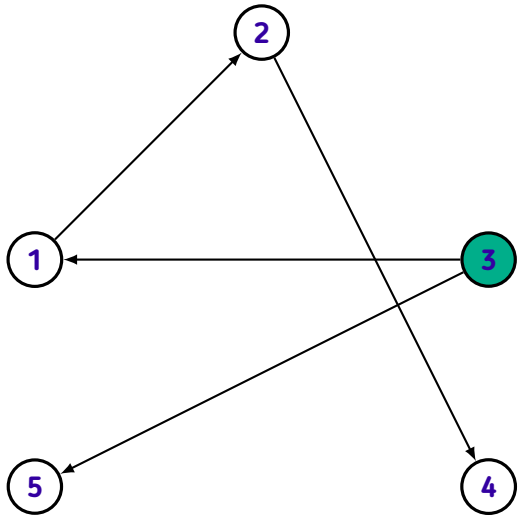
Exemplo de entrada e saída

```
5
3
1 2
3 5
3 1
2 4
```



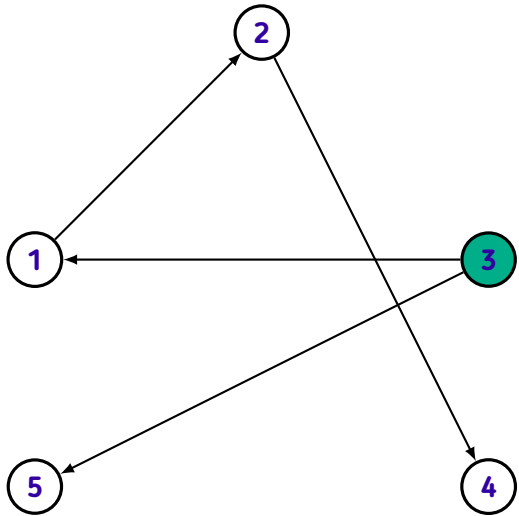
Exemplo de entrada e saída

```
5
3
1 2
3 5
3 1
2 4
```



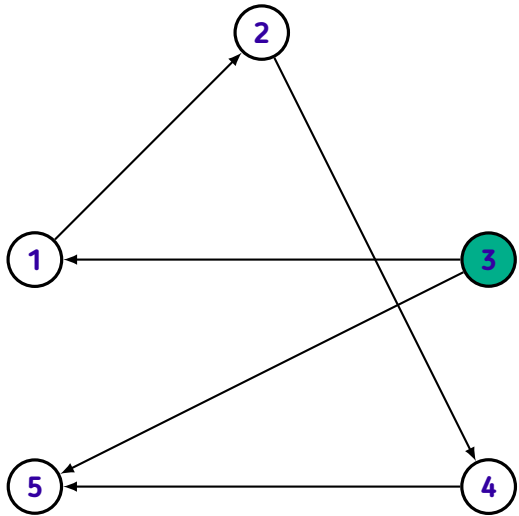
Exemplo de entrada e saída

```
5
3
1 2
3 5
3 1
2 4
4 5
```



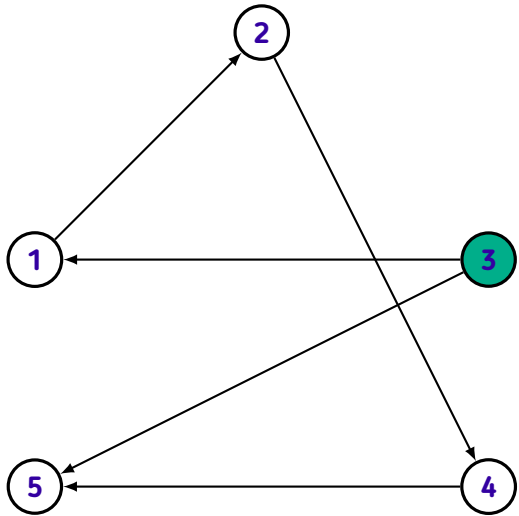
Exemplo de entrada e saída

```
5
3
1 2
3 5
3 1
2 4
4 5
```



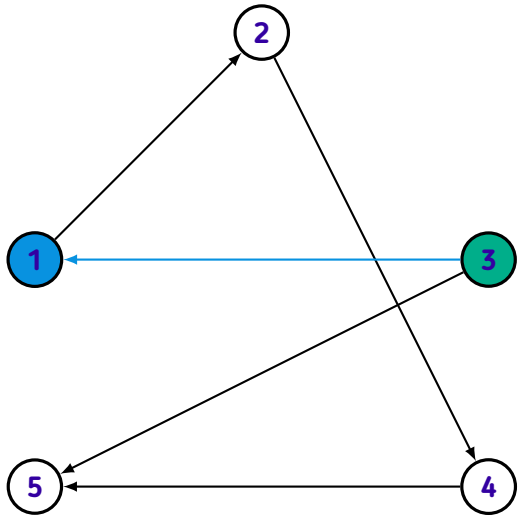
Exemplo de entrada e saída

```
5
3
1 2
3 5
3 1
2 4
4 5
0 0
```



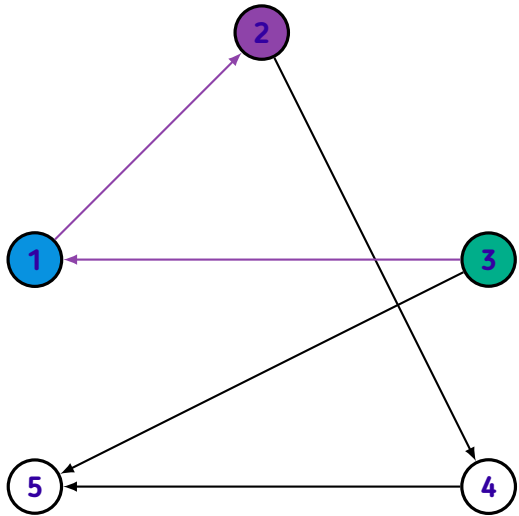
Exemplo de entrada e saída

```
5
3
1 2
3 5
3 1
2 4
4 5
0 0
```



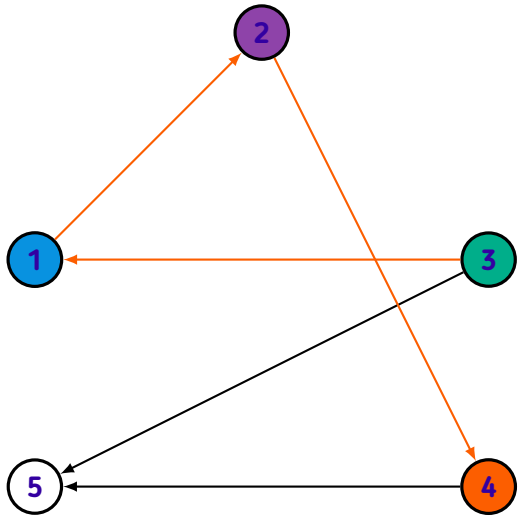
Exemplo de entrada e saída

```
5
3
1 2
3 5
3 1
2 4
4 5
0 0
```



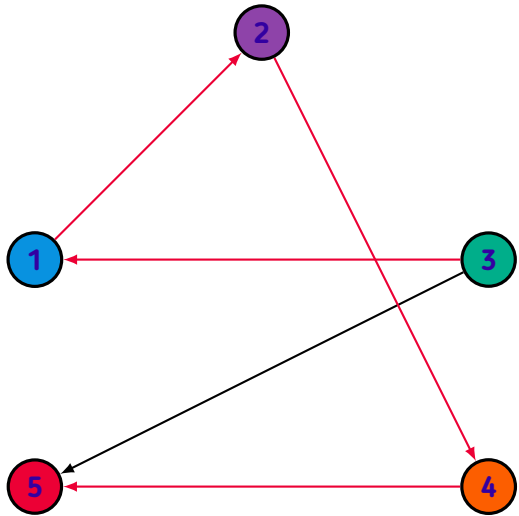
Exemplo de entrada e saída

```
5
3
1 2
3 5
3 1
2 4
4 5
0 0
```



Exemplo de entrada e saída

```
5
3
1 2
3 5
3 1
2 4
4 5
0 0
```

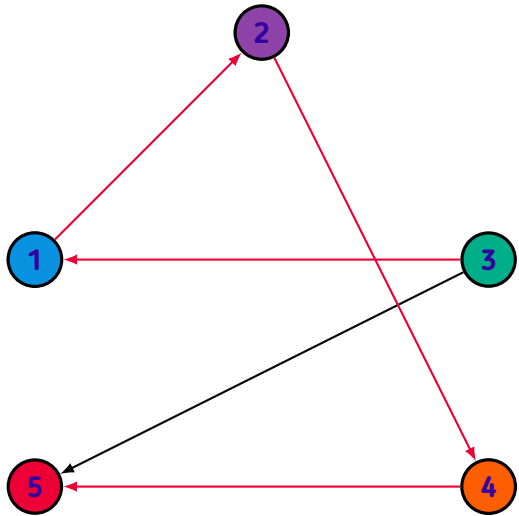


Exemplo de entrada e saída

```
5
3
1 2
3 5
3 1
2 4
4 5
0 0
```

↓

4, 5



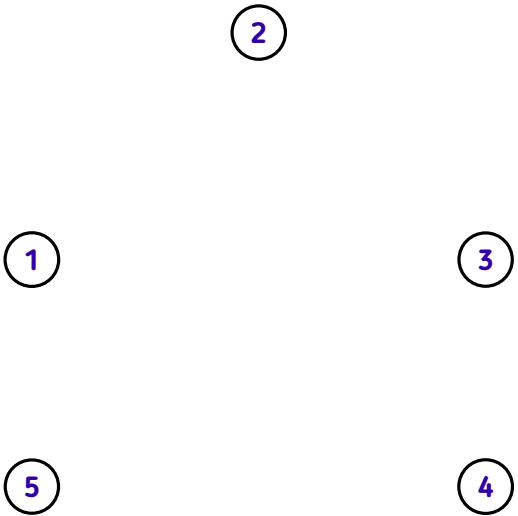
Exemplo de entrada e saída

Exemplo de entrada e saída

```
5
```

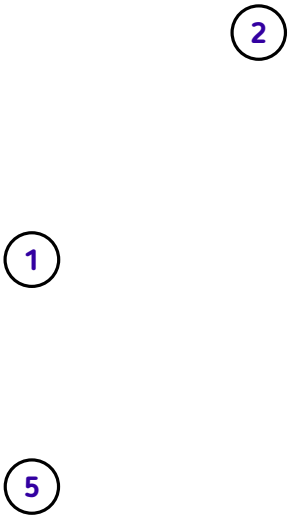
Exemplo de entrada e saída

5



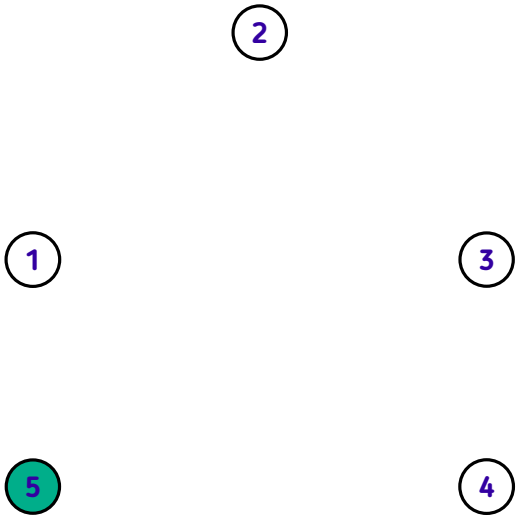
Exemplo de entrada e saída

```
5
5
```



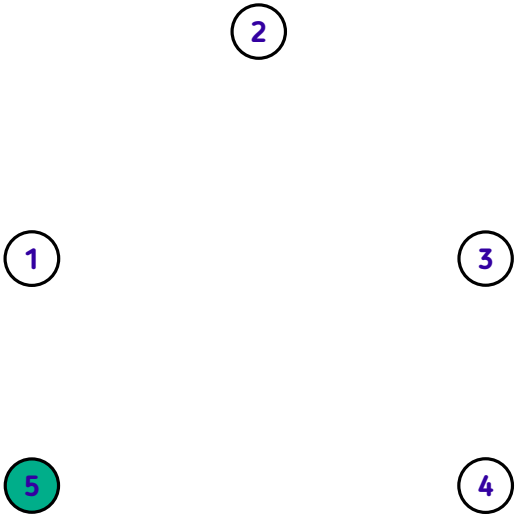
Exemplo de entrada e saída

5
5



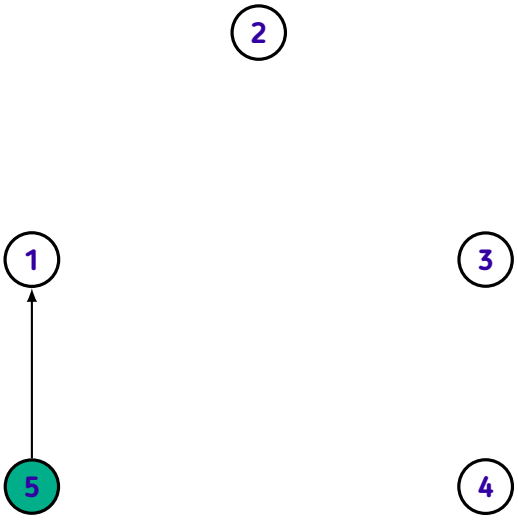
Exemplo de entrada e saída

```
5
5
5 1
```



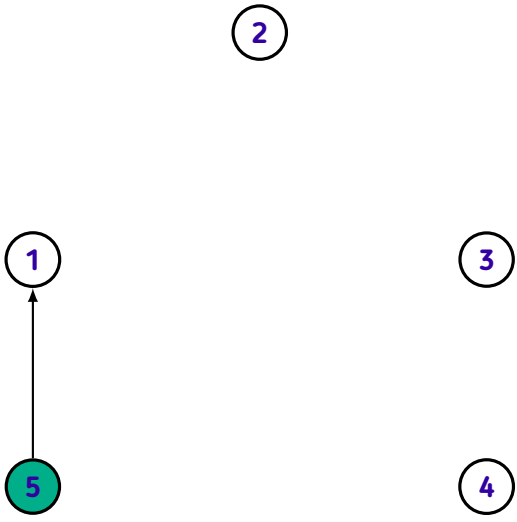
Exemplo de entrada e saída

5
5
5 1



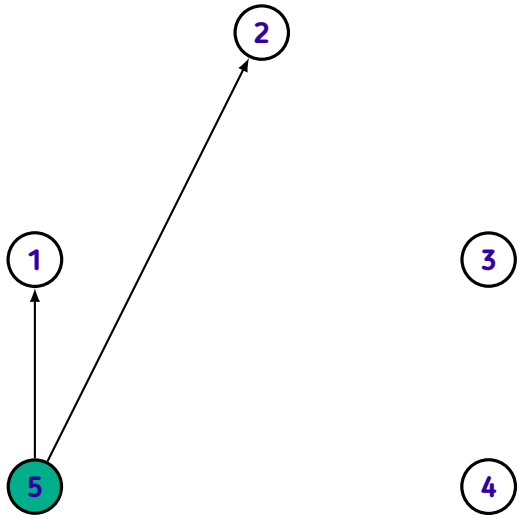
Exemplo de entrada e saída

```
5
5
5 1
5 2
```



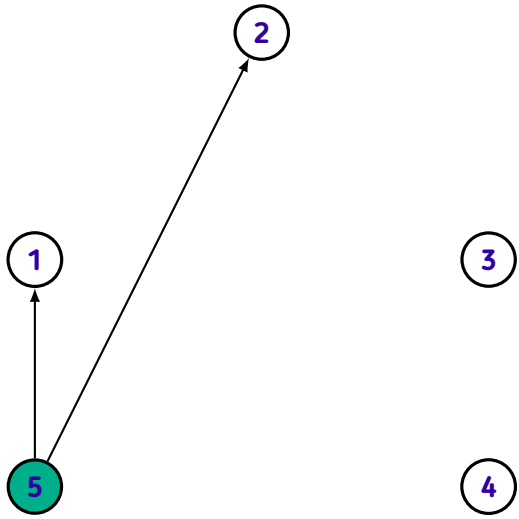
Exemplo de entrada e saída

5
5
5 1
5 2



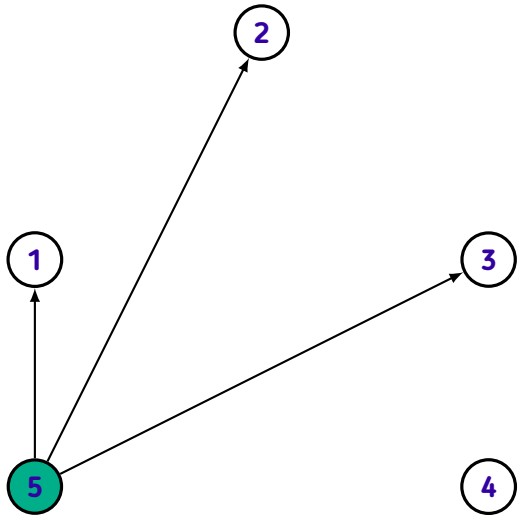
Exemplo de entrada e saída

5
5
5 1
5 2
5 3



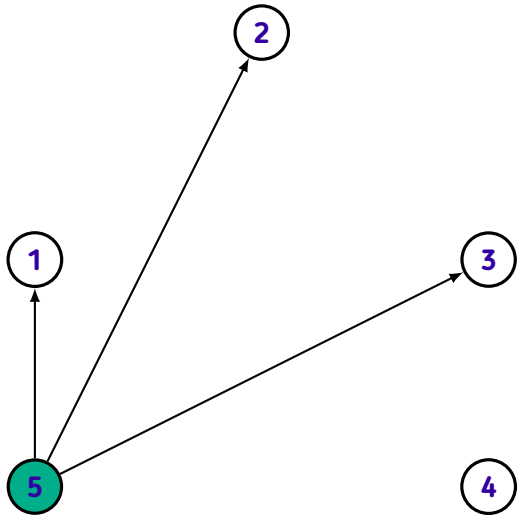
Exemplo de entrada e saída

5
5
5 1
5 2
5 3



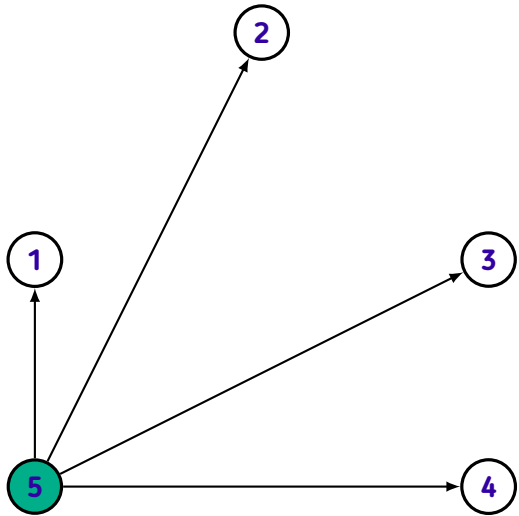
Exemplo de entrada e saída

5
5
5 1
5 2
5 3
5 4



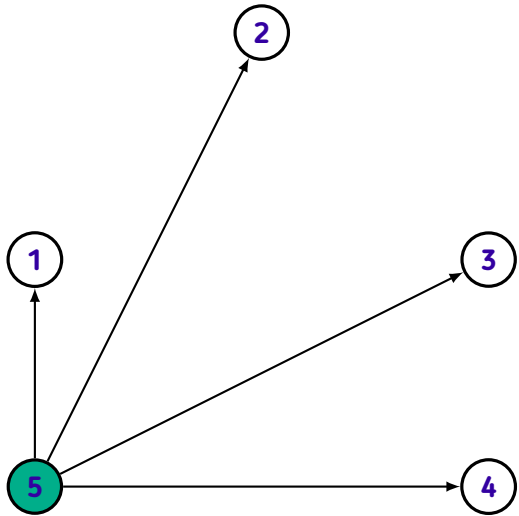
Exemplo de entrada e saída

5
5
5 1
5 2
5 3
5 4



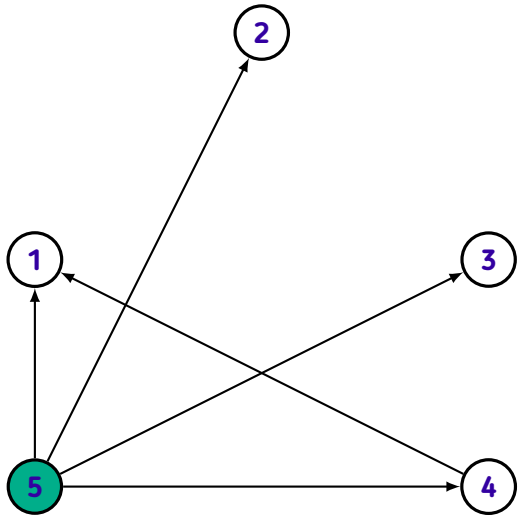
Exemplo de entrada e saída

```
5
5
5 1
5 2
5 3
5 4
4 1
```



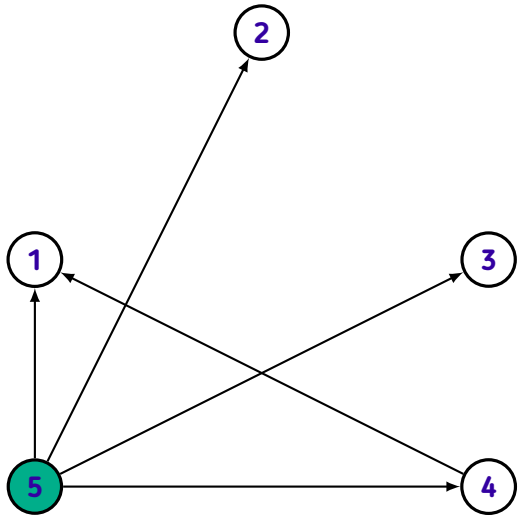
Exemplo de entrada e saída

5
5
5 1
5 2
5 3
5 4
4 1



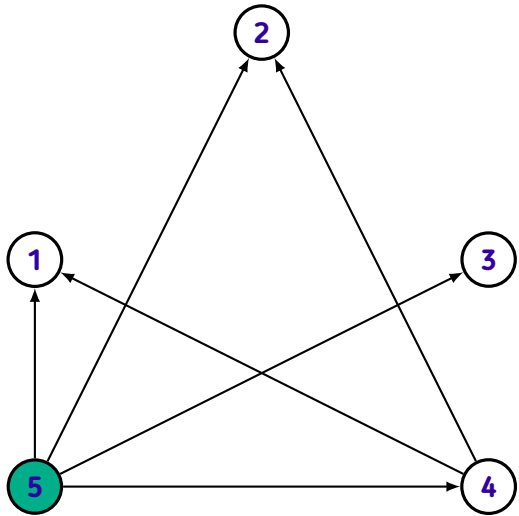
Exemplo de entrada e saída

```
5
5
5 1
5 2
5 3
5 4
4 1
4 2
```



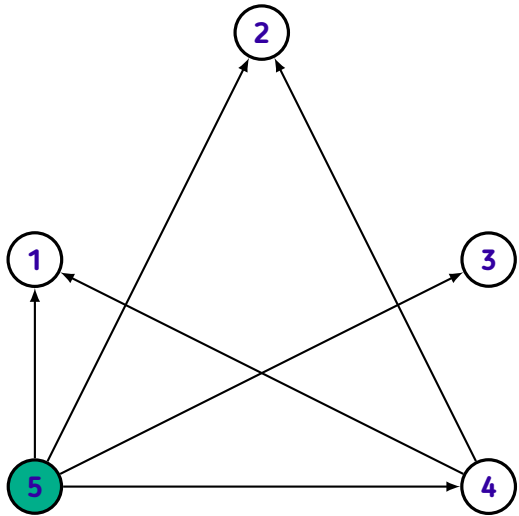
Exemplo de entrada e saída

5
5
5 1
5 2
5 3
5 4
4 1
4 2



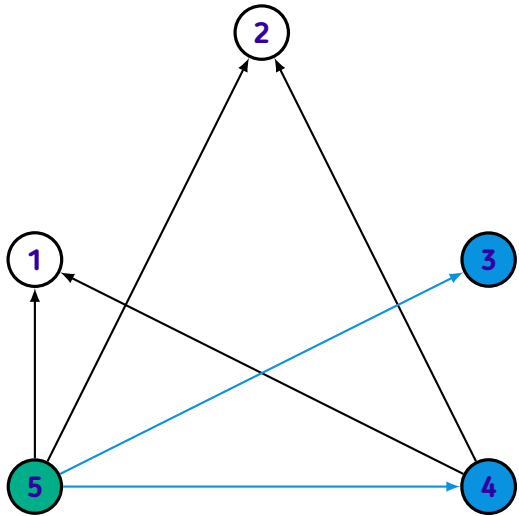
Exemplo de entrada e saída

```
5
5
5 1
5 2
5 3
5 4
4 1
4 2
0 0
```



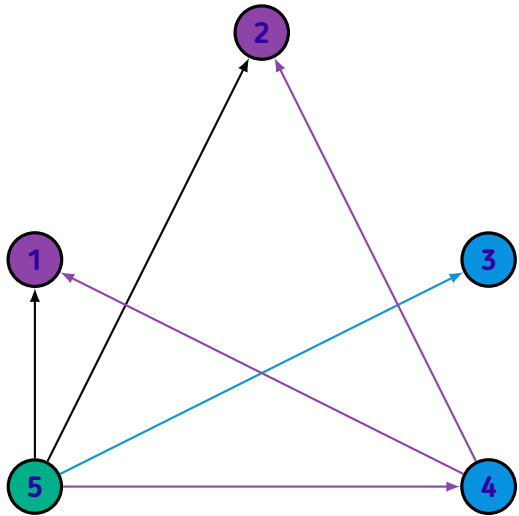
Exemplo de entrada e saída

```
5
5
5 1
5 2
5 3
5 4
4 1
4 2
0 0
```



Exemplo de entrada e saída

```
5
5
5 1
5 2
5 3
5 4
4 1
4 2
0 0
```

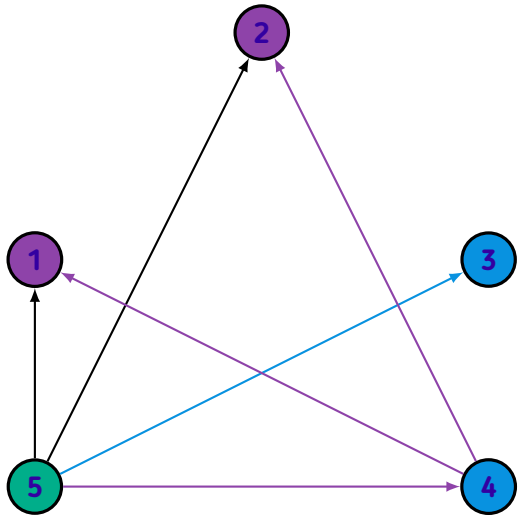


Exemplo de entrada e saída

```
5
5
5 1
5 2
5 3
5 4
4 1
4 2
0 0
```

↓

2, 1



Solução

Solução

- **A BFS pode ser modificada para computar as distâncias máximas**

Solução

- A BFS pode ser modificada para computar as distâncias máximas
- Esta modificação, contudo, altera a complexidade da travessia

Solução

- ⋆ **A BFS pode ser modificada para computar as distâncias máximas**
- ⋆ **Esta modificação, contudo, altera a complexidade da travessia**
- ⋆ **Isto porque um mesmo nó pode entrar até $N - 1$ vezes na fila**

Solução

- ⋆ **A BFS pode ser modificada para computar as distâncias máximas**
- ⋆ **Esta modificação, contudo, altera a complexidade da travessia**
- ⋆ **Isto porque um mesmo nó pode entrar até $N - 1$ vezes na fila**
- ⋆ **Assim complexidade no pior caso será $O(N^2)$**

Solução

- ⋆ **A BFS pode ser modificada para computar as distâncias máximas**
- ⋆ **Esta modificação, contudo, altera a complexidade da travessia**
- ⋆ **Isto porque um mesmo nó pode entrar até $N - 1$ vezes na fila**
- ⋆ **Assim complexidade no pior caso será $O(N^2)$**
- ⋆ **Se o grafo tiver ciclos, esta travessia modificada entrará em um laço infinito!**

```cpp
vector<int> max_dist(int s, int N, int w = 1) {
    vector<int> dist(N + 1, -1);
    queue<int> q;

    dist[s] = 0; q.push(s);

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

        for (auto v : adj[u]) {
            if (dist[v] < dist[u] + w) {
                dist[v] = dist[u] + w;
                q.push(v);
            }
        }
    }

    return dist;
}
```

```
pair<int, int> solve(int s, int N)
{
    auto dist = max_dist(s, N);

    auto d = 0, v = s;

    for (int u = 1; u <= N; ++u)
        if (dist[u] > d)
        {
            d = dist[u];
            v = u;
        }

    return { d, v };
}
```