Mil Ilhas

Mil ilhas é um grupo de bonitas ilhas localizado no Mar de Java. Consiste em N ilhas, numeradas de 0 a $N-1$.

Existem M canoas, numeradas de 0 a $M-1$, que pode ser usadas para navegar entre ilhas. Para cada i tal que $0 \leq i \leq M-1$, a canoa i pode ser ancorada na ilha $U[i]$ ou na ilha $V[i]$, e pode ser usada para navegar entre as ilhas $U[i]$ e $V[i]$. Especificamente, quando a canoa está ancorada na ilha $U[i]$, ela pode ser usada para navegar da ilha $U[i]$ para a ilha $V[i]$, passando a ficar ancorada na ilha $V[i]$. Similarmente, quando a canoa está ancorada na ilha $V[i]$, ela pode ser usada para navegar da ilha $V[i]$ para a ilha $U[i]$, passando a ficar ancorada na ilha $U[i]$. Inicialmente, a canoa está ancorada na ilha $U[i]$. É possível que múltiplas canoas possas ser usadas para navegar entre o mesmo par de ilhas. É também possível que múltiplas canoas estejam ancoradas na mesma ilha.

Por razões de segurança, uma canoa necessita de receber manutenção depois de cada vez que navega, o que impede que a mesma canoa duas vezes seguidas. Isto é, depois de usar uma canoa i, outra canoa deve ser usada antes da canoa i poder ser usada novamente.

A Bu Dengklek quer planear uma viagem entre algumas das ilhas. A sua viagem é "**válida** se e só se as seguintes condições forem satisfeitas.

- Ela começa e termina a sua viagem na ilha 0.
- Ela visita pelo menos uma ilha além da ilha 0.
- Depois da viagem terminar, cada canoa está ancorada na mesma ilha em que estava antes da viagem, ou seja, a canoa i, para cada i tal que $0 \leq i \leq M-1$, deve estar ancorada na ilha $U[i]$.

Ajuda a Bu Dengklek a descobrir uma viagem válida envolvendo navegar um máximo de $2\,000\,000$ vezes, ou determina que nenhuma viagem válida existe. Pode ser provada que com as restrições deste problema (ver secção de Restrições), se uma viagem válida existir, também existe uma viagem que não envolve navegar mais do que $2\,000\,000$ vezes.

Detalhes de Implementação

Deves implementar a seguinte função:

```
union(bool, int[]) find_journey(int N, int M, int[] U, int[] V)
```

- N: o número de ilhas.
- M: o número de canoas.
- U, V: arrays de tamanho M descrevendo o número de canoas.
- Este procedimento deverá devolver ou um booleano ou um array de inteiros.
    - Se nenhuma viagem válida existir, a função deve devolver `false`.
    - Se uma viagem válida existe, tens duas opções:
        - Para receberes pontuação máxima, a função deve devolver um array de no máximo $2\,000\,000$ inteiros representando uma viagem válida. Mais precisamente, os elementos deste array devem ser os números das canoas que são usadas na viagem (na ordem em que são usadas).
        - Para receberes pontuação parcial, a função deve devolver `true`, um array de mais do que $2\,000\,000$ inteiros ou um array de inteiros que não descreve uma viagem válida. (ver a secção de Subtarefas para mais detalhes.)
- Esta função é chamada exatamente uma vez.

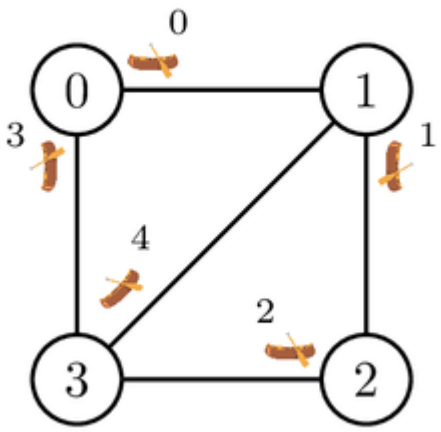
Exemplos

Exemplo 1

Considera a seguinte chamada:

```
find_journey(4, 5, [0, 1, 2, 0, 3], [1, 2, 3, 3, 1])
```

As ilhas e as canoas são mostradas na figura abaixo.



Uma possível viagem válida é a seguinte. Bu Dengklek primeiro navega as canoas 0, 1, 2, e 4 nessa ordem. Como resultado, ela está na ilha 1. Depois disso, Bu Dengklek pode navegar a canoa 0 outra vez já que está atualmente ancorada na ilha 1 e a última canoa que ela usou não foi a canoa 0. Depois de navegar a canoa 0 outra vez, a Bu Dengklek está agora na ilha 0. No entanto, as canoas 1, 2 e 4 não estão ancoradas nas mesmas ilhas que estavam antes da viagem. A Bu

Dengklek continua então a sua viagem ao navegar as canoas 3, 2, 1, 4, e 3 outra vez. A Bu Dengklek está de volta na ilha 0 e todas as canoas estão ancoradas nas mesmas ilhas que estavam antes da viagem.

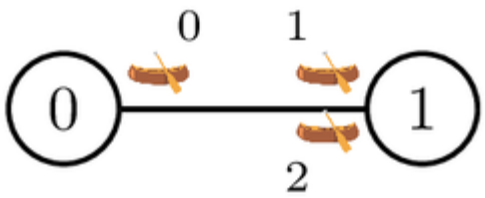
Portanto, o valor devolvido $[0, 1, 2, 4, 0, 3, 2, 1, 4, 3]$ representa uma viagem válida.

Exemplo 2

Considera a seguinte chamada:

```
find_journey(2, 3, [0, 1, 1], [1, 0, 0])
```

As ilhas e as canoas são mostradas na figura abaixo.



A Bu Dengklek apenas pode começar a navegar a canoa 0, sendo que depois pode navegar a canoa 1 ou 2. Nota que ela não pode navegar a canoa 0 duas vezes seguidas. Em ambos os casos, a está de volta à ilha 0. Contudo, as canoas não estão ancoradas nas mesmas ilhas onde estavam no início da viagem, e a Bu Dengklek não pode navegar nenhuma canoa depois, uma vez que a única canoa ancorada na ilha 0 é a que ela acabou de usar. Como não existe nenhuma viagem válida, a função deve devolver `false`.

Restrições

- $2 \leq N \leq 100\,000$
- $1 \leq M \leq 200\,000$
- $0 \leq U[i] \leq N - 1$ e $0 \leq V[i] \leq N - 1$ (para cada i tal que $0 \leq i \leq M - 1$)
- $U[i] \neq V[i]$ (para cada i tal que $0 \leq i \leq M - 1$)

Subtarefas

1. (5 pontos) $N = 2$
2. (5 pontos) $N \leq 400$. Para cada par de ilhas distintas x e y ($0 \leq x < y \leq N - 1$), existem exatamente duas canoas que podem ser usadas para navegar entre elas. Uma delas está ancorada na ilha x, e a outra está ancorada na ilha y.

3. (21 pontos) $N \leq 1000$, M é par e para cada i **par** tal que $0 \leq i \leq M-1$, as canoas i e $i+1$ podem ambas ser usadas para navegar entre as ilhas $U[i]$ e $V[i]$. A canoa i está inicialmente ancorada na ilha $U[i]$ e a canoa $i+1$ está inicialmente ancorada na ilha $V[i]$. Formalmente, $U[i] = V[i+1]$ e $V[i] = U[i+1]$.
4. (24 pontos) $N \leq 1000$, M é par e para cada i **par** tal que $0 \leq i \leq M-1$, as i e $i+1$ podem ambas ser usadas para navegar entre as ilhas $U[i]$ e $V[i]$. Ambas as canoas estão inicialmente ancoradas na ilha $U[i]$. Formalmente, $U[i] = U[i+1]$ e $V[i] = V[i+1]$.
5. (45 pontos) Sem restrições adicionais.

Para cada caso de teste onde existe uma viagem válida existe, a tua solução:

- recebe pontuação máxima se devolver uma viagem válida,
- recebe 35% dos pontos se devolver `true`, um array de mais de $2\,000\,000$ inteiros, ou um array que não descreva uma viagem válida,
- recebe 0 pontos em qualquer outro caso.

Para cada caso de teste em que uma viagem válida não existe, a tua solução:

- recebe pontuação máxima se devolver `false`,
- recebe 0 pontos em qualquer outro caso.

Nota que a pontuação final de cada subtarefa é o mínimo de pontos para os casos de teste dessa subtarefa.

Avaliador Exemplo

O avaliado exemplo lê o input no seguinte formato:

- linha 1: N M
- linha $2+i$ ($0 \leq i \leq M-1$): $U[i]$ $V[i]$

O avaliador exemplo imprimir as tuas respostas no seguinte formato:

- Se `find_journey` devolver um `bool`:
  - linha 1: 0
  - linha 2: 0 se `find_journey` devolver `false` ou 1 caso contrário.
- Se `find_journey` devolver um `int[]`, sejam os elementos deste array $c[0], c[1], \ldots c[k-1]$. O avaliador exemplo imprime:
  - linha 1: 1
  - linha 2: k
  - linha 3: $c[0]$ $c[1]$ $\ldots$ $c[k-1]$