Navegando islas

El archipiélago de Thousands Islands es un grupo de hermosas islas situadas en el Mar de Java. Contiene N islas, numeradas desde 0 hasta $N-1$.

Existen M canoas, numeradas desde 0 hasta $M-1$, que pueden utilizarse para navegar entre las islas. Para cada i tal que $0 \leq i \leq M-1$, la canoa i se puede estacionar tanto en la isla $U[i]$ como en la $V[i]$, y puede utilizarse para navegar entre las islas $U[i]$ y $V[i]$. Específicamente, cuando la canoa está estacionada en la isla $U[i]$, puede ser utilizada para navegar desde la isla $U[i]$ hasta la isla $V[i]$, luego de lo cual la canoa queda estacionada en la isla $V[i]$. De manera similar, cuando la canoa está estacionada en la isla $V[i]$, se puede utilizar para navegar desde la isla $V[i]$ hasta la isla $U[i]$, luego de lo cual la canoa queda estacionada en la isla $U[i]$. Al comienzo, la canoa está estacionada en la isla $U[i]$. Es posible que existan múltiples canoas que puedan ser utilizadas para navegar entre el mismo par de islas. También es posible que múltiples canoas estén estacionadas en la misma isla.

Por cuestiones de seguridad, es necesario hacerle mantenimiento a una canoa luego de utilizarla para navegar, por lo cual está prohibido navegar con la misma canoa dos veces seguidas. Es decir, luego de utilizar una canoa i, se debe utilizar alguna otra canoa antes de poder volver a utilizar la canoa i.

Bu Dengklek desea programar un paseo por algunas de las islas. Su paseo es **válido** si y solo si se cumplen las siguientes condiciones.

- Comienza y finaliza su paseo en la isla 0.
- Visita al menos una isla que no sea la isla 0.
- Al finalizar el paseo, cada canoa está estacionada en la misma isla en la que estaba al comenzar el viaje. En otras palabras, la canoa i, para cada i tal que $0 \leq i \leq M-1$, debe estar estacionada en la isla $U[i]$.

Debes ayudar a Bu Dengklek a encontrar cualquier paseo válido, en el cual se navegue a lo sumo $2\ 000\ 000$ veces, o bien determinar que no existe tal paseo válido. Es posible demostrar que con las restricciones del enunciado (ver la sección Restricciones), si existe algún paseo válido, entonces también existe un paseo válido que no requiere navegar más de $2\ 000\ 000$ veces.

Detalles de implementación

Debes implementar la siguiente función:

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

- N: la cantidad de islas.
- M: la cantidad de canoas.
- U, V: arreglos de tamaño M que describen las canoas.
- La función debe retornar o bien un booleano o bien un arreglo de enteros.
    - Si no existe paseo válido, la función debe retornar `false`.
    - Si existe paseo válido, tienes dos opciones:
        - Para obtener el puntaje total, la función debe retornar un arreglo de a lo sumo $2\,000\,000$ enteros, que represente un paseo válido. Más precisamente, los elementos de este arreglo deben indicar los números de las canoas que se utilizan en el paseo (en el orden en que se utilizan).
        - Para obtener puntaje parcial, la función debe retornar `true`, un arreglo de más de $2\,000\,000$ enteros, o un arreglo de enteros que no representa un paseo válido (ver la sección Subtareas para más detalles).
- Esta función se llama exactamente una vez.

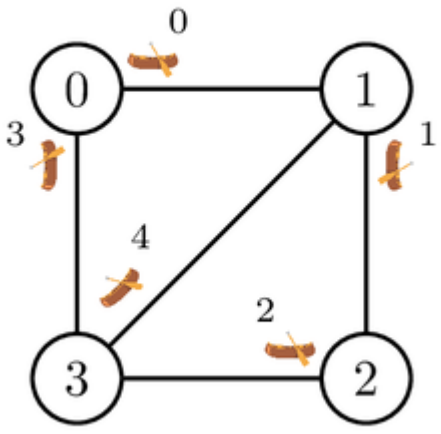
Ejemplos

Ejemplo 1

Se realiza la siguiente llamada:

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

En la siguiente figura se muestran las islas y canoas.



Un posible paseo válido es el siguiente. Bu Dengklek navega primero en las canoas 0, 1, 2, y 4, en ese orden. Como resultado de esta navegación, está en la isla 1. Luego de eso, Bu Dengklek puede navegar en la canoa 0 nuevamente, ya que en ese momento se encuentra estacionada en la isla 1

y la canoa 0 no fue la última canoa utilizada. Luego de navegar nuevamente en la canoa 0, Bu Dengklek ahora está en la isla 0. Sin embargo, las canoas 1, 2 y 4 no están estacionadas en las mismas islas que al comenzar el paseo. Bu Dengklek continúa entonces su paseo navegando en las canoas 3, 2, 1, 4, y 3 nuevamente. Bu Dengklek ha vuelto a la isla 0 y todas las canoas están estacionadas en las mismas islas que al comenzar el viaje.

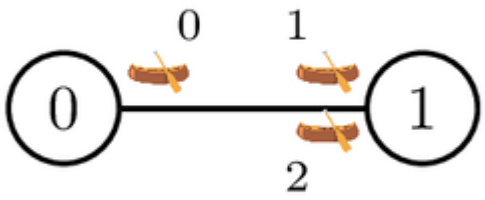
Por lo tanto, si la función retorna $[0, 1, 2, 4, 0, 3, 2, 1, 4, 3]$, este arreglo representa un paseo válido.

Ejemplo 2

Se realiza la siguiente llamada:

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

En la siguiente figura se muestran las islas y canoas.



Bu Dengklek solamente puede comenzar navegando en la canoa 0, luego de lo cual puede navegar tanto la canoa 1 como la 2. Notar que no tiene permitido navegar en la canoa 0 dos veces seguidas. En ambos casos, Bu Dengklek vuelve a la isla 0. Sin embargo, las canoas no están estacionadas en las mismas islas que al comenzar el paseo, y Bu Dengklek ya no puede navegar en ninguna canoa, ya que la única canoa estacionada en la isla 0 es la que acaba de utilizar. Como no hay paseo válido, la función debe retornar `false`.

Restricciones

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

Subtareas

1. (5 puntos) $N = 2$

2. (5 puntos) $N \leq 400$. Por cada par de islas distintas x e y ($0 \leq x < y \leq N-1$), hay exactamente dos canoas que pueden utilizarse para navegar entre ellas. Una de ellas está estacionada en la isla x, y la otra está estacionada en la isla y.
3. (21 puntos) $N \leq 1000$, M es par, y por cada i **par** tal que $0 \leq i \leq M-1$, las canoas i e $i+1$ pueden ambas ser utilizadas para viajar entre las islas $U[i]$ y $V[i]$. La canoa i inicialmente está estacionada en la isla $U[i]$ y la canoa $i+1$ inicialmente está estacionada en la isla $V[i]$. Formalmente, $U[i] = V[i+1]$ y $V[i] = U[i+1]$.
4. (24 puntos) $N \leq 1000$, M es par, y por cada i **par** tal que $0 \leq i \leq M-1$, las canoas i e $i+1$ pueden ambas ser utilizadas para viajar entre las islas $U[i]$ y $V[i]$. Ambas canoas se encuentran inicialmente estacionadas en la isla $U[i]$. Formalmente, $U[i] = U[i+1]$ y $V[i] = V[i+1]$.
5. (45 puntos) Sin más restricción.

Por cada caso de prueba en el que exista un paseo válido, una solución:

- obtiene el puntaje completo si retorna un paseo válido,
- obtiene 35% del puntaje si retorna `true`, un arreglo de más de $2\,000\,000$ enteros, o un arreglo que no representa un paseo válido,
- obtiene 0 puntos en cualquier otro caso.

Por cada caso de prueba en el que no exista un paseo válido, una solución:

- obtiene el puntaje completo si retorna `false`,
- obtiene 0 puntos en cualquier otro caso.

Notar que el puntaje final de cada subtarea será el puntaje mínimo obtenido entre los casos de prueba de esa subtarea.

Evaluador local

El evaluador local lee la entrada con el siguiente formato:

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

El evaluador local escribe la salida con el siguiente formato:

- Si `find_journey` retorna un `bool`:
  - línea 1: 0
  - línea 2: 0 si `find_journey` retorna `false`, o 1 si no.
- Si `find_journey` retorna un `int[]`, llamamos $c[0], c[1], \ldots c[k-1]$ a los elementos de este arreglo. El evaluador local escribe:
  - línea 1: 1
  - línea 2: k
  - línea 3: $c[0]$ $c[1]$ $\ldots$ $c[k-1]$