Islas Milenarias

Las Islas Milenarias son un grupo de hermosas islas situadas en el Mar de Java. El grupo consta de N islas enumeradas del 0 a $N-1$.

Hay M canoas enumeradas del 0 a $M-1$ que pueden ser usadas para navegar entre islas. Para cada i tal que $0 \leq i \leq M-1$, la i-ésima canoa puede estar atracada en la isla $U[i]$ o $V[i]$, y puede utilizarse para navegar entre las islas $U[i]$ y $V[i]$. En concreto, cuando la canoa está atracada en la isla $U[i]$, puede utilizarse para navegar desde la isla $U[i]$ hasta la isla $V[i]$, tras lo cual la canoa pasa a estar atracada en la isla $V[i]$. Del mismo modo, cuando la canoa está atracada en la isla $V[i]$, puede utilizarse para navegar desde la isla $V[i]$ hasta la isla $U[i]$, tras lo cual la canoa pasa a estar atracada en la isla $U[i]$. Inicialmente, la canoa está atracada en la isla $U[i]$. Es posible que hayan varias canoas usadas para navegar entre el mismo par de islas. También es posible que varias canoas estén atracadas en la misma isla.

Por razones de seguridad, cada vez que se usa una canoa, la canoa necesita recibir mantenimiento después, lo que impide que la misma canoa se use dos veces seguidas. Es decir, después de utilizar la i-ésima canoa, hay que utilizar otra canoa antes de poder volver a utilizar la i-ésima canoa.

Bu Dengklek quiere planificar un viaje por algunas islas. Su viaje es **válido** si y sólo si se cumplen las siguientes condiciones.

- Empieza y termina su viaje en la isla 0.
- Visita al menos una isla distinta de la isla 0.
- Una vez finalizado el viaje, cada canoa está atracada en la misma isla en la que estaba antes del viaje. Es decir, para cada i tal que $0 \leq i \leq M-1$, la canoa i debe estar atracada en la isla $U[i]$.

Ayuda a Bu Dengklek a encontrar cualquier viaje válido en el que navegue entre islas como máximo $2\ 000\ 000$ veces, o determina que no existe tal viaje válido. Se puede demostrar que bajo las restricciones especificadas en esta tarea (ver sección Restricciones), si existe un viaje válido, también existe un viaje válido en el que no se navega más de $2\ 000\ 000$ veces.

Detalles de Implementación

Su tarea es implementar el siguiente procedimiento:

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

- N: El número de islas.
- M: El número de canoas.
- U, V: Arreglos de longitud M que describen las canoas.
- El procedimiento debe devolver un booleano o un arreglo de enteros.
  - Si no existe un viaje válido, la función debe devolver `false`.
  - Si existe un viaje válido, tiene dos opciones:
    - Para recibir la puntuación completa, el procedimiento debe devolver un array de a lo mucho $2\,000\,000$ enteros que representen un viaje válido. Es decir, los elementos de este arreglo deben ser los números de las canoas que se utilizan en el viaje (en el orden en que se utilizan).
    - Para recibir una puntuación parcial, el procedimiento debe devolver `true`, un arreglo de más de $2\,000\,000$ enteros, o un arreglo de enteros que no describen un viaje válido. (Ver la sección Subtareas para más detalles.)
- El procedimiento se llama exactamente una vez.

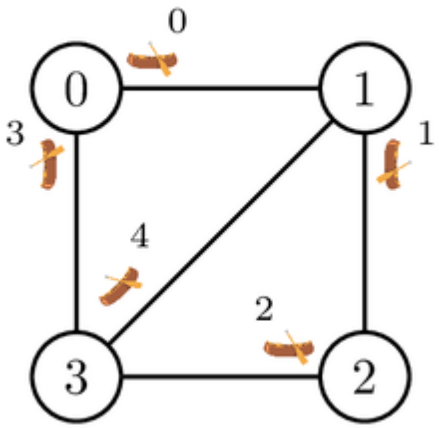
Ejemplos

Ejemplo 1

Considere la siguiente llamada:

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

Las islas y las canoas se muestran en la siguiente imagen.



Un posible viaje válido es el siguiente. Bu Dengklek navega primero en las canoas 0, 1, 2 y 4 en ese orden. Como resultado, se encuentra en la isla 1. Después, Bu Dengklek puede volver a navegar en la canoa 0, ya que está atracada en la isla 1 y la última canoa que utilizó no es la canoa 0. Después de volver a navegar en la canoa 0, Bu Dengklek se encuentra ahora en la isla 0. Sin embargo, las canoas 1, 2 y 4 no están atracadas en las mismas islas que antes del viaje. Bu Dengklek continúa

entonces su viaje navegando en las canoas 3, 2, 1, 4 y 3 de nuevo. Bu Dengklek vuelve a la isla 0 y todas las canoas están atracadas en las mismas islas que antes del viaje.

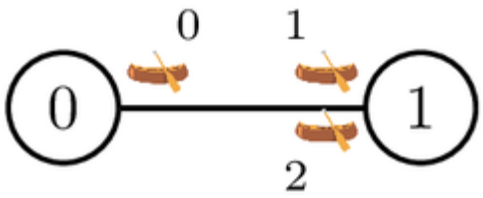
Por tanto, el valor devuelto $[0, 1, 2, 4, 0, 3, 2, 1, 4, 3]$ representa un viaje válido.

Ejemplo 2

Considere la siguiente llamada:

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

Las islas y las canoas se muestran en la siguiente imagen.



Bu Dengklek sólo puede empezar a navegar en la canoa 0, tras lo cual puede navegar en la canoa 1 o 2. Nótese que no puede navegar en la canoa 0 dos veces seguidas. En ambos casos, Bu Dengklek vuelve a la isla 0. Sin embargo, las canoas no están atracadas en las mismas islas que antes del viaje, y Bu Dengklek no puede navegar en ninguna canoa después, ya que la única canoa atracada en la isla 0 es la que acaba de utilizar. Como no hay ningún viaje válido, el procedimiento debe devolver `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) $N \leq 400$. Para 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 atracada en la isla x, y la otra en la isla y.

3. (21 puntos) $N \leq 1000$, M es par, y para cada i **par** tal que $0 \leq i \leq M-1$, las canoas i y $i+1$ pueden utilizarse para navegar entre las islas $U[i]$ y $V[i]$. La canoa i está inicialmente atracada en la isla $U[i]$ y la canoa $i+1$ está inicialmente atracada 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 para cada i **par** tal que $0 \leq i \leq M-1$, las canoas i y $i+1$ pueden navegar entre las islas $U[i]$ y $V[i]$. Ambas canoas están inicialmente atracadas en la isla $U[i]$. Formalmente, $U[i] = U[i+1]$ y $V[i] = V[i+1]$.
5. (45 puntos) Sin restricciones adicionales.

Para cada caso de prueba en el que exista un viaje válido, su solución:

- obtiene todos los puntos si devuelve un viaje válido,
- obtiene un 35% de los puntos si devuelve `true`, un array de más de $2\ 000\ 000$ enteros, o un array que no describe un viaje válido,
- obtiene 0 puntos en cualquier otro caso.

Para cada caso de prueba en el que no exista un viaje válido, su solución

- obtiene todos los puntos si devuelve `false`,
- obtiene 0 puntos en cualquier otro caso.

Tenga en cuenta que la puntuación final para cada subtarea es el mínimo de los puntos de todos los casos de prueba en la misma.

Evaluador de prueba

El evaluador de prueba 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 de prueba imprime sus respuestas con el siguiente formato:

- Si `find_journey` devuelve un `bool`:
  - línea 1: 0
  - línea 2: 0 si `find_journey` devuelve `false`, o 1 en caso contrario.
- Si `find_journey` devuelve un `int[]`, denótese los elementos de este array por $c[0], c[1], \ldots c[k-1]$. El evaluador de prueba imprime:
  - línea 1: 1
  - línea 2: k
  - línea 3: $c[0]\ c[1]\ \ldots\ c[k-1]$