Peajes

En Japón, las ciudades están conectadas por una red de autopistas. Esta red consiste de N ciudades y M autopistas. Cada autopista conecta un par de ciudades distintas. No hay dos autopistas que conecten al mismo par de ciudades. Las ciudades son numeradas del 0 al $N-1$, y las autopistas son numeradas del 0 al $M-1$. Puedes manejar por cualquier autopista en ambas direcciones y viajar de una ciudad a cualquier otra usando las autopistas.

Se cobra un peaje por transitar en cada autopista. Este peaje depende de las condiciones de **tráfico** en la autopista. El tráfico puede ser **ligero** o **pesado**. Cuando el tráfico es ligero, el peaje es A yenes (moneda Japonesa). Cuando el tráfico es pesado, el peaje es B yenes. Se garantiza que $A < B$. Los valores de A y B son conocidos.

Tienes una máquina que, dadas las condiciones de tráfico de todas las autopistas, calcula el peaje menor total que se debe pagar para viajar entre las ciudades S y T ($S \neq T$) bajo las condiciones de tráfico especificadas.

Sin embargo, la máquina es solo un prototipo. Los valores de S y T son fijos (es decir, codificados en la máquina). Tú desconoces estos valores pero te gustaría determinarlos. Para hacerlo, planeas especificar varias condiciones de tráfico a la máquina y usar los valores de peaje que retorna para deducir S y T. Dado que especificar las condiciones de tráfico es costoso, no quieres usar la máquina muchas veces.

Detalles de implementación

Debes implementar el siguiente procedimiento:

```
find_pair(int N, int[] U, int[] V, int A, int B)
```

- `N`: el número de ciudades.
- `U` y `V`: arreglos de tamaño M, donde M es el número de autopistas conectando las ciudades. Para cada i ($0 \leq i \leq M-1$), la autopista i conecta las ciudades `U[i]` y `V[i]`.
- `A`: el valor del peaje para una autopista cuando el tráfico es ligero.
- `B`: el valor del peaje para una autopista cuando el tráfico es pesado.
- Este procedimiento es llamado exactamente una vez para cada caso de prueba.

- Nota que el valor de M es el largo de los arreglos y puede ser obtenido como se indica en la nota de implementación.

El procedimiento `find_pair` puede llamar a la siguiente función:

```
int64 ask(int[] w)
```

- El largo de `w` debe ser M. El arreglo `w` describe las condiciones de tráfico.
- Para cada i ($0 \leq i \leq M-1$), `w[i]` determina la condición de tráfico en la autopista i. El valor de `w[i]` debe ser 0 o 1.
  - `w[i] = 0` significa que el tráfico en la autopista i es ligero.
  - `w[i] = 1` significa que el tráfico en la autopista i es pesado.
- Esta función retorna el peaje menor total para viajar entre las ciudades S y T, bajo las condiciones de tráfico especificadas por `w`.
- Esta función puede ser llamada a la más 100 veces (para cada caso de prueba).

`find_pair` debe de llamar el siguiente procedimiento para reportar la respuesta:

```
answer(int s, int t)
```

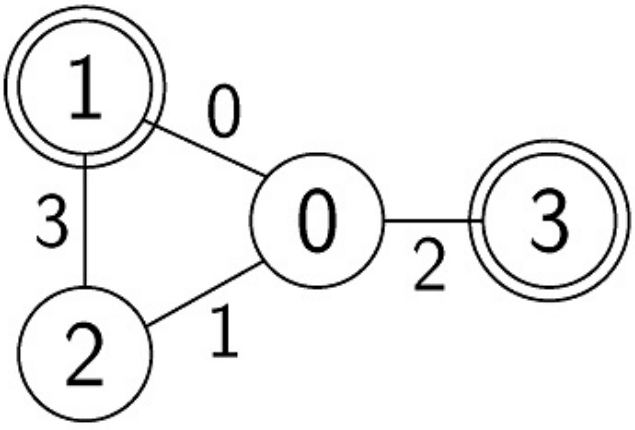
- `s` y `t` debe ser el par S y T (el orden no es importante).
- Este procedimiento debe ser llamado exactamente una vez.

Si alguna de las condiciones anteriores no se cumple, tu programa será evaluado como **`Wrong Answer`**. De lo contrario, tu programa será evaluado como **`Accepted`** y tu puntaje será calculado según el número de llamadas a `ask` (ver Subtareas).

Ejemplo

Sea $N = 4$, $M = 4$, $U = [0, 0, 0, 1]$, $V = [1, 2, 3, 2]$, $A = 1$, $B = 3$, $S = 1$, y $T = 3$.

El evaluador llama a `find_pair(4, [0, 0, 0, 1], [1, 2, 3, 2], 1, 3)`.



En la imagen, la arista con el número i corresponde a la autopista i. Algunas llamadas posibles a `ask` y los valores de retorno correspondientes son listados a continuación:

Llamada	Retorno
`ask([0, 0, 0, 0])`	2
`ask([0, 1, 1, 0])`	4
`ask([1, 0, 1, 0])`	5
`ask([1, 1, 1, 1])`	6

Para la llamada de función `ask([0, 0, 0, 0])`, el tráfico de cada autopista es ligero y el peaje para cada una es 1. La ruta más barata de $S = 1$ a $T = 3$ es $1 \to 0 \to 3$. El peaje total para esta ruta es 2. Por lo tanto, esta función retorna 2.

Para una respuesta correcta, el procedimiento `find_pair` debe llamar `answer(1, 3)` o `answer(3, 1)`.

El archivo `sample-01-in.txt` en el zip adjunto corresponde a este ejemplo. Otros ejemplos de entradas también están disponibles en el zip.

Restricciones

- $2 \leq N \leq 90\,000$
- $1 \leq M \leq 130\,000$
- $1 \leq A < B \leq 1\,000\,000\,000$
- Para cada $0 \leq i \leq M - 1$
  - $0 \leq U[i] \leq N - 1$
  - $0 \leq V[i] \leq N - 1$
  - $U[i] \neq V[i]$
- $(U[i], V[i]) \neq (U[j], V[j])$ y $(U[i], V[i]) \neq (V[j], U[j])$ $(0 \leq i < j \leq M - 1)$
- Puedes viajar de cualquier ciudad a cualquier otra utilizando las autopistas.
- $0 \leq S \leq N - 1$
- $0 \leq T \leq N - 1$
- $S \neq T$

En este problema, el evaluador **NO** es adaptativo. Esto significa que S y T están fijos al inicio de la ejecución del evaluador y no dependen de las consultas realizadas por tu solución.

Subtareas

1. (5 puntos) S o T es 0, $N \leq 100$, $M = N - 1$
2. (7 puntos) S o T es 0, $M = N - 1$

3. (6 puntos) $M = N - 1$, $U[i] = i$, $V[i] = i + 1$ $(0 \leq i \leq M - 1)$
4. (33 puntos) $M = N - 1$
5. (18 puntos) $A = 1$, $B = 2$
6. (31 puntos) Sin restricciones adicionales

Asume que tu programa es evaluado como **`Accepted`**, y realiza X llamadas a `ask`. Entonces tu puntaje P para el caso de prueba, dependiendo del número de la subtarea, se calcula de la siguiente manera:

- Subtarea 1. $P = 5$.
- Subtarea 2. Si $X \leq 60$, $P = 7$. De lo contrario, $P = 0$.
- Subtarea 3. Si $X \leq 60$, $P = 6$. De lo contrario, $P = 0$.
- Subtarea 4. Si $X \leq 60$, $P = 33$. De lo contrario, $P = 0$.
- Subtarea 5. Si $X \leq 52$, $P = 18$. De lo contrario, $P = 0$.
- Subtarea 6.
  - Si $X \leq 50$, $P = 31$.
  - Si $51 \leq X \leq 52$, $P = 21$.
  - Si $53 \leq X$, $P = 0$.

Nota que tu puntaje para cada subtarea es el mínimo de los puntajes para los casos de prueba en la subtarea.

Evaluador de ejemplo

El evaluador de ejemplo lee la entrada en el siguiente formato:

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

Si tu programa es evaluado como **`Accepted`**, el evaluador de ejemplo imprimirá `Accepted: q`, siendo `q` el número de llamadas a `ask`.

Si tu programa es evaluado como **`Wrong Answer`**, el evaluador de ejemplo imprimirá `Wrong Answer: MSG`, donde `MSG` es uno de los siguientes:

- `answered not exactly once`: El procedimiento `answer` no fue llamado exactamente una vez.
- `w is invalid`: El largo de `w` dado a `ask` no es M, o `w[i]` no es ni 0 ni 1 para algún i $(0 \leq i \leq M - 1)$.
- `more than 100 calls to ask`: La función `ask` es llamada más de 100 veces.
- `{s, t} is wrong`: El procedimiento `answer` es llamado con un par incorrecto `s` y `t`.