Highway Tolls

Las ciudades en Japón están conectadas por una red de carreteras. La red consiste de N ciudades y M carreteras. Cada carretera conecta un par de ciudades distintas. No existen 2 carreteras que conecten el mismo par de ciudades. Las ciudades están numeradas del 0 a $N-1$ y las carreteras del 0 al $M-1$. Todas las carreteras viajan en ambas direcciones. Se asegura que puedes viajar de cualquier ciudad a cualquier otra usando las carreteras.

Se cobra una cuota por viajar en cada carretera. La cuota por usar una carretera depende del **tráfico** en la carretera. El tráfico puede ser **ligero** o **pesado**. Cuando el tráfico es ligero, la cuota es A yenes (moneda japonesa). Cuando el tráfico es pesado la cuota es B yenes. Se asegura que $A < B$. Conoces los valores de A y B.

Tienes una máquina que, dado el tráfico de todas las carreteras, calcula la cuota mínima que se debe pagar para viajar entre las ciudades S y T ($S \neq T$).

Como esta máquina es un prototipo, las ciudades S y T son fijas y no las conoces. Tu tarea es encontrar S y T. Para lograrlo debes usar la maquina con diferentes condiciones de tráfico y, con las cuotas que devuelve, deducir S y T. Usar la máquina es muy caro por lo que quieres usarla pocas veces.

Detalles de implementación

Implementa el siguiente siguiente procedimiento:

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

- `N`: la cantidad de ciudades.
- `U` y `V`: arreglos de longitud M, donde M es la cantidad de carreteras. Para cada i ($0 \leq i \leq M-1$), la carretera i conecta las ciudades `U[i]` y `V[i]`.
- `A`: la cuota con tráfico ligero.
- `B`: la cuota con tráfico pesado.
- Este procedimiento se manda a llamar solo una vez para cada caso de prueba.
- El valor M es la longitud del arreglo y se puede obtener como se indica en las notas de implementación.

El procedimiento `find_pair` puede llamar la siguiente función:

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

- El arreglo `w` describe las condiciones del tráfico. La longitud de `w` debe ser M.
- Para cada i ($0 \leq i \leq M-1$), `w[i]` indica el tráfico en la carretera i. El valor de `w[i]` debe ser 0 o 1.
  - `w[i] = 0` indica que el tráfico en la carretera i es ligero.
  - `w[i] = 1` indica que el tráfico en la carretera i es pesado.
- Esta función devuelve la cuota más barata al viajar entre las ciudades S y T, con las condiciones de tráfico indicadas en `w`.
- Esta función se puede llamar a lo más 100 veces (para cada caso de prueba).

`find_pair` debe llamar la siguiente función para reportar la respuesta:

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

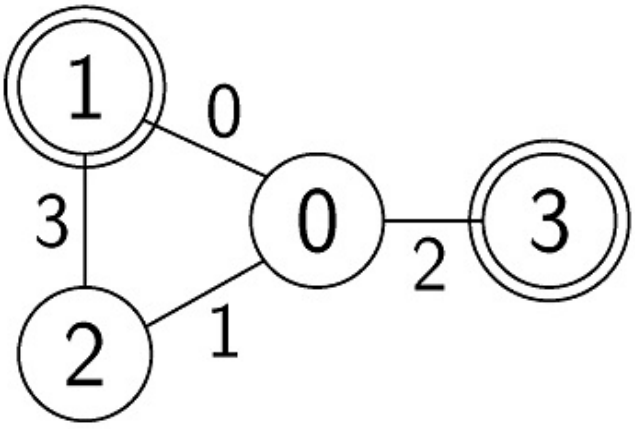
- `s` y `t` el par de ciudades S y T (el orden no importa).
- Esta función se debe llamar exactamente 1 vez.

Si alguna de las condiciones anteriores no se cumplen tu programa se evaluará como **`Wrong Answer`**. De lo contrario será **`Accepted`** y tu puntaje se calcula dependiendo del 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 `find_pair(4, [0, 0, 0, 1], [1, 2, 3, 2], 1, 3)`.



En la figura anterior, el arista con el numero i corresponde a la carretera i. Algunas posibles llamadas a `ask` y los valores devueltos se muetran a continuación:

Llamada	Valor devuelto
`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 `ask([0, 0, 0, 0])`, todas las carreteras tienen tráfico ligero y la cuota de cada carretera es 1. La ruta más barata desde $S = 1$ hasta $T = 3$ es $1 \to 0 \to 3$. La cuota total de esta ruta es 2, por lo que la función devuelve 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. Otras entradas de ejemplo se encuentran en el paquete.

Constraints

- $2 \leq N \leq 90\,000$
- $1 \leq M \leq 130\,000$
- $1 \leq A < B \leq 1\,000\,000\,000$
- 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 ciudad utlizando las carreteras.
- $0 \leq S \leq N - 1$
- $0 \leq T \leq N - 1$
- $S \neq T$

El evaluador de este problema no es adaptativo. Esto signigica que S y T son fijos desde el inicio de la ejecución y no dependen de las consultas que haga 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

Asumiendo que tu programa es evaluado como **Accepted**, y que haces X llamadas a `ask`. El puntaje P para el caso de prueba, dependiendo 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 $53 \leq X$, $P = 0$.

Tu puntaje para cada subtarea es el puntaje mínimo de los casos de prueba de esa 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 imprime `Accepted: q`, siendo `q` la cantidad de llamadas a `ask`.

Si tu programa es evaluado como **Wrong Answer**, imprime `Wrong Answer: MSG`, donde `MSG` puede ser:

- `answered not exactly once`: El procedimiento `answer` no fue llamado exactamente una vez.
- `w is invalid`: La longitud de `w` al llamar `ask` no es M o `w[i]` no es 0 ó 1 para alguna i ($0 \leq i \leq M - 1$).
- `more than 100 calls to ask`: La función `ask` fue llamada más de 100 veces.
- `{s, t} is wrong`: mandaste llamar `answer` con valores incorrectos para `s` y `t`.