Peajes

En Japón, las ciudades están conectadas por una red de autopistas. Esta red consiste de N ciudades y M autopiastas. Cada autopista conecta un par de ciudades distintas. No existen dos autopistas que conecten el mismo par de ciudades. Las ciudades están numeradas de 0 a $N-1$, y las autopistas están numeradas de 0 a $M-1$. Se puede conducir en cualquier autopista en ambas direcciones. Es posible viajar de cualquier ciudad a cualquier ciudad usando las autopistas.

Un cobro es hecho por conducir en cada autopista. El cobro para una autopista depende de la condición del **tráfico** en esta. El tráfico puede ser **ligero** o **pesado**. Cuando el tráfico es ligero, el cobro es A yenes (moneda Japonesa). Cuando el tráfico es pesado, el cobro es B yenes. Está garantizado que $A < B$. Note que usted conoce los valores de A y B.

Tú tienes una máquina que, dadas las condiciones del tráfico de todas las autopistas, calcula el cobro total más pequeño que uno tiene que pagar para viajar entre un par de ciudades S y T ($S \neq T$), bajo condiciones de tráfico específicadas.

Sin embarto, la máquina es solo un prototipo, Los valores de S y T son fijados (i.e., quemados o harcodeados en la máquina) y no son conocidos por tí. Te gustaría determinar S y T. Con el fin de hacerlo, tu plan es pasar varias condiciones de tráfico a la máquina, y usar los valores de los cobros que esta devuelva para deducir S y T. Como pasar 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 ciudades. Para cada i ($0 \leq i \leq M-1$), la autipista i conecta las ciudades `U[i]` y `V[i]`.
- `A`: el cobro para una autopista cuando el tráfico es liviano.
- `B`: el cobro para una autopista cuando el tráfico es pesado.
- Este procedimiento es llamado exactamente una vez por cada caso de prueba.

- Fíjese que el valor de M es el tamaño de los arreglos, y puede ser obtenido como es indicado en las notas de implementación.

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

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

- El tamaño de `w` debe ser M. El arreglo `w` describe las condiciones de tráfico.
- Por cada i ($0 \leq i \leq M-1$), `w[i]` da las condiciones de tráfico en la autopista i. El valor de `w[i]` debe ser 0 o 1.
  - `w[i] = 0` significa que el tráfico de la autopista i es liviano.
  - `w[i] = 1` significa que el tráfico de la autopista i es pesado.
- Esta función devuelve el pago total más pequeño por viajar entre las ciudades S y T, bajo las condiciones de tráfico especificadas por `w`.
- Esta función puede ser llamada a lo mucho 100 veces (por cada caso de prueba).

`find_pair` debería llamar al siguiente procedimiento para reportar la respuesta:

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

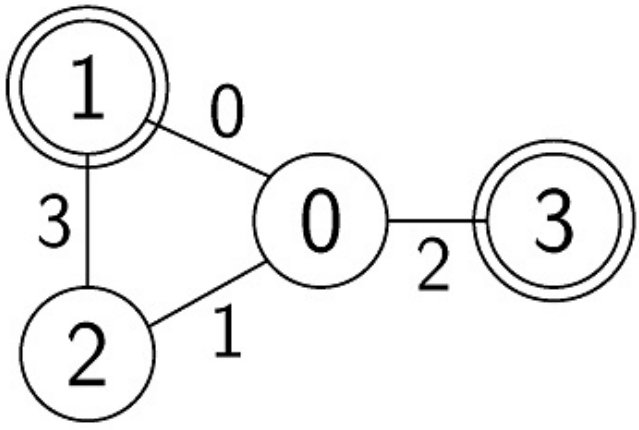
- `s` y `t` deben ser el par S y T (el orden no importa).
- Este procedimiento debe ser llamado exactamente una vez.

Si alguna de las anteriores condiciones no es cumplida, tu programa será juzgado como **`Wrong Answer`**. Por otro lado, tu programa será juzgado como **`Accepted`** y tu puntaje será calculado por el número de llamadas a `ask` (ver Subtareas).

Ejemplo

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

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



En la figura de arriba, el arco con valor de *i* corresponde a la autopista *i*. Algunas posibles llamadas 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 a la función `ask([0, 0, 0, 0])`, el tráfico de cada autopista es liviano y el pago para esta es 1. La ruta más barata de $S = 1$ a $T = 3$ es $1 \to 0 \to 3$. El pago total para este camino es 2. Por tanto, esta función devuelve 2.

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

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

Restricciones

- $2 \le N \le 90\,000$
- $1 \le M \le 130\,000$
- $1 \le A < B \le 1\,000\,000\,000$
- Por cada $0 \le i \le M - 1$
  - $0 \le U[i] \le N - 1$
  - $0 \le V[i] \le N - 1$
  - $U[i] \ne V[i]$
- $(U[i], V[i]) \ne (U[j], V[j])$ and $(U[i], V[i]) \ne (V[j], U[j])$ $(0 \le i < j \le M - 1)$
- Puedes viajar de cualquier ciudad a cualquier ciudad usando las autopistas.
- $0 \le S \le N - 1$
- $0 \le T \le N - 1$
- $S \ne T$

En este problema, el evaluador es NO adaptativo. Esto significa que S y T están fijados al principio de la corrida del evaluador y no dependen de las solicitudes hechas por tu solución.

Subtareas

1. (5 puntos) uno de S o T es 0, $N \le 100$, $M = N - 1$

2. (7 puntos) uno de 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

Asuma que su programa es juzgado como **Accepted**, y realiza X llamadas a `ask`. Luego, tu puntaje P para el caso de prueba, dependiendo del número de subtarea, es calculado como sigue:

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

Observe que su 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 juzgado como **Accepted**, el evaluador de ejemplo imprime `Accepted: q`, con `q` como el número de llamadas a `ask`.

Si tu programa es juzgado como **Wrong Answer**, este imprime `Wrong Answer: MSG`, donde `MSG` es uno de estos:

- `answered not exactly once`: El procedimiento `answer` no fue llamado exactamente una vez.
- `w is invalid`: El tamaño de `w` dado a `ask` no es M o `w[i]` no es 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 des y `t`.