Simurgh

De acuerdo a leyendas antiguas persas en Shahnameh, Zal, el legengario héreo persa, estába perdidamente enamorado de Rudaba, la princesa de Kabul. Cuando Zal pidió la mano de Rudaba para casarse, su padre le puso un reto.

Existen n ciudades en Persia, numeradas de 0 a $n-1$, y m caminos bidireccionales, numerados de 0 a $m-1$. Cada camino conecta un par de ciudades distintas. Cada par de ciudades están conectadas por a lo más un camino. Algunos de los caminos son *caminos reales* usados solamente por la gente de la realeza. La tarea de Zal es determinar cuáles de los caminos son *caminos reales*.

Zal tiene un mapa que contiene todas las ciudades y caminos de Persia. Él no conoce cuáles de los caminos son reales pero puede pedir ayuda a Simurgh, la benevolente ave protectora mítica de Zal. Sin embargo, el ave Simurgh no quiere revelarle directamente a Zal el conjunto de caminos que son *caminos reales*. Simurgh primero le revela a Zal que el conjunto de *caminos reales* es un *conjunto dorado*. Un conjunto de caminos es un *conjunto dorado* si y sólo si:

- Tiene *exactamente* $n-1$ caminos, y
- para cada par de ciudades, es posible llegar de una ciudad a otra viajando sólo por los caminos del conjunto.

Además, Zal puede hacer preguntas a Simurgh. Para cada pregunta:

1. Zal escoge un *conjunto dorado* de caminos, después
2. Simurgh le responde a Zal cuántos caminos del *conjunto dorado* son *caminos reales*.

Tu programa debe ayudar a Zal a encontrar el conjunto de *caminos reales* en a lo más q preguntas. El evaluador tomará el papel de Simurgh.

Detalles de la implementación

Debes implementar la siguiente funcion:

```
int[] find_roads(int n, int[] u, int[] v)
```

- n: número de ciudades,
- u y v: arreglos de longitud m. Para todo $0 \leq i \leq m-1$, $u[i]$ y $v[i]$ son ciudades conectadas por el camino i.
- Esta funcion debe retornar un arreglo de longitud $n-1$ que contiene los índices de los *caminos reales* (en cualquier orden).

Tu solución puede hacer a lo más q llamadas de la siguiente funcion del evaluador:

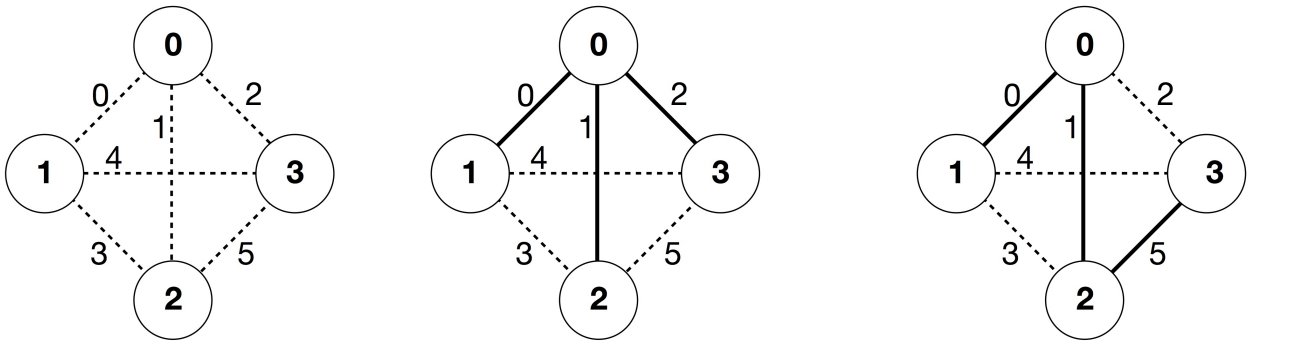
```
int count_common_roads(int[] r)
```

- r: arreglo de longitud $n-1$ que contiene los índices de un *conjunto dorado* (en cualquier orden).
- Esta funcion retorna el número de *caminos reales* que se encuentran en el arreglo r.

Ejemplo

```
find_roads(4, [0, 0, 0, 1, 1, 2], [1, 2, 3, 2, 3, 3])
```

`find_roads( ... )` `count_common_roads([0, 1, 2]) = 2` `count_common_roads([5, 1, 0]) = 3`



En este ejemplo hay 4 ciudades y 6 caminos. Denotamos (a, b) como un camino que conecta las ciudades a y b. Los caminos están numerados de 0 a 5 en el siguiente orden: $(0, 1)$, $(0, 2)$, $(0, 3)$, $(1, 2)$, $(1, 3)$, and $(2, 3)$. Cada *conjunto dorado* tiene $n - 1 = 3$ caminos.

Asuma que los *caminos reales* son los caminos con índices 0, 1 y 5, es decir, los caminos $(0, 1)$, $(0, 2)$ y $(2, 3)$. El programa de ejemplo hace las siguientes llamadas:

- `count_common_roads([0, 1, 2])` retorna 2. Esta llamada pregunta por los índices $0, 1$, y 2, es decir, los caminos $(0, 1)$, $(0, 2)$ y $(0, 3)$. Dos de ellos son *caminos reales*.
- `count_common_roads([5, 1, 0])` retorna 3. Esta llamada pregunta acerca de todos los *caminos reales*.

La funcion `find_roads` debe retornar `[5, 1, 0]` o cualquier otro arreglo de longitud 3 que contenga esos 3 elementos.

Nota que las siguientes llamadas no están permitidas:

- `count_common_roads([0, 1])`: aquí la longitud de r no es 3.
- `count_common_roads([0, 1, 3])`: aquí r no describe un *conjunto dorado*, pues es imposible viajar de la ciudad 0 a la ciudad 3 usando sólo los caminos $(0, 1)$, $(0, 2)$ y $(1, 2)$.

Restricciones

- $2 \leq n \leq 500$
- $n-1 \leq m \leq n(n-1)/2$
- $0 \leq u[i], v[i] \leq n-1$ (para todo $0 \leq i \leq m-1$)
- Para todo $0 \leq i \leq m-1$, el camino con índice i conecta dos ciudades distintas (i.e., $u[i] \neq v[i]$).
- Para cada par de ciudades, a lo más hay un camino que las conecta.
- Es posible viajar entre cada par de ciudades usando los caminos.
- El conjunto de *caminos reales* es un *conjunto dorado*.
- `find_roads` debe llamar `count_common_roads` a lo más q veces. En cada llamada, el conjunto de caminos especificados en r deben formar un *conjunto dorado*.

Subtarea

1. (13 puntos) $n \leq 7$, $q = 30,000$
2. (17 puntos) $n \leq 50$, $q = 30,000$
3. (21 puntos) $n \leq 240$, $q = 30,000$
4. (19 puntos) $q = 12,000$ y existe un camino entre cada par de ciudades.
5. (30 puntos) $q = 8,000$

Evaluador de ejemplo

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

- línea 1: n m
- línea $2+i$ (para todo $0 \leq i \leq m-1$): $u[i]$ $v[i]$
- línea $2+m$: $s[0]$ $s[1]$ $\ldots$ $s[n-2]$

Aquí, $s[0], s[1], \ldots, s[n-2]$ son los índices de los *caminos reales*.

El evaluador de ejemplo imprime `YES`, si `find_roads` manda llamar `count_common_roads` a lo más $30,000$ veces y retorna el conjunto correcto de *caminos reales*. De lo contrario, imprime `NO`.

Ten en cuenta que la funcion `count_common_roads` en el evaluador de ejemplo no revisa si r tiene todas las propiedades de un *conjunto dorado*. El evaluador de ejemplo cuenta y regresa el número de *caminos reales* del arreglo r. Sin embargo, si el programa que envias llama a `count_common_roads` con un conjunto de caminos que no describen un *conjunto dorado*, el veredicto del evaluador será 'Wrong Answer''.

Notas técnicas

La funcion `count_common_roads` en C++ y Pascal pasa el *método por referencia* por razones de eficiencia. Sin embargo, puedes mandar llamar la funcion en la forma usual. Se garantiza que el evaluador no cambiará el valor del arreglo r.