¿Dónde está la raíz?

`Este problema es interactivo`

Te dan un árbol de n vértices. El árbol es un grafo que tiene exactamente un camino simple entre cualquier par de vértices. **Está garantizado que al menos un nodo tenga grado de al menos 3.** Uno de los vértices es la raíz y tu tarea es encontrarlo.

Puedes hacer consultas de la siguiente manera:

- Para un conjunto de vértices $a_1, a_2, \ldots, a_m$, pregunta si el menor ancestro común es parte del conjunto.

Necesitas averiguar la raíz del árbol.

Como un recordatorio, el menor ancestro común de un conjunto de vértices es el vértice v más alejado de la raíz tal que v es parte del camino desde cualquier vértice del conjunto a la raíz.

Interacción

Comienza la interacción leyendo un único entero n ($4 \le n \le 500$) - el número de vértices.

Luego lee las siguientes $n-1$ líneas. La i-ésima línea contiene dos enteros a_i, b_i ($1 \le a_i, b_i \le n$), indicando que hay una arista entre los nodos a_i, b_i en el árbol.

Está garantizado que estas $n-1$ aristas forman el árbol y que al menos hay un vértice con grado de al menos 3.

Para hacer una consulta, primero imprime "`?`", luego el entero m, y luego m enteros distintos $a_1, a_2, \ldots, a_m$ ($1 \le m \le n$, $1 \le a_i \le n$, todos los a_i son distintos) - los vértices, para quienes quieres preguntar si su mínimo ancestro común está entre ellos.

Como respuesta, el problema imprimirá "`YES`" si el mínimo ancestro común está en $a_1, a_2, \ldots, a_m$, y "`NO`" de otra manera.

Puedes preguntar a lo mucho 1000 veces. Imprimir la respuesta no cuenta como hacer una pregunta.

Cuando has identificado a la raíz, imprime el símbolo "`!`" y luego otro entero v ($1 \le v \le n$) - la raíz. Luego termina el programa.

Luego de imprimir la consulta, no olvides de imprimir un fin de línea (std::endl o "\n") y hacer "flush" del la salida. Para hacer eso, haga:

- `fflush(stdout)` o `cout.flush()` en C++;
- `stdout.flush()` en Python;

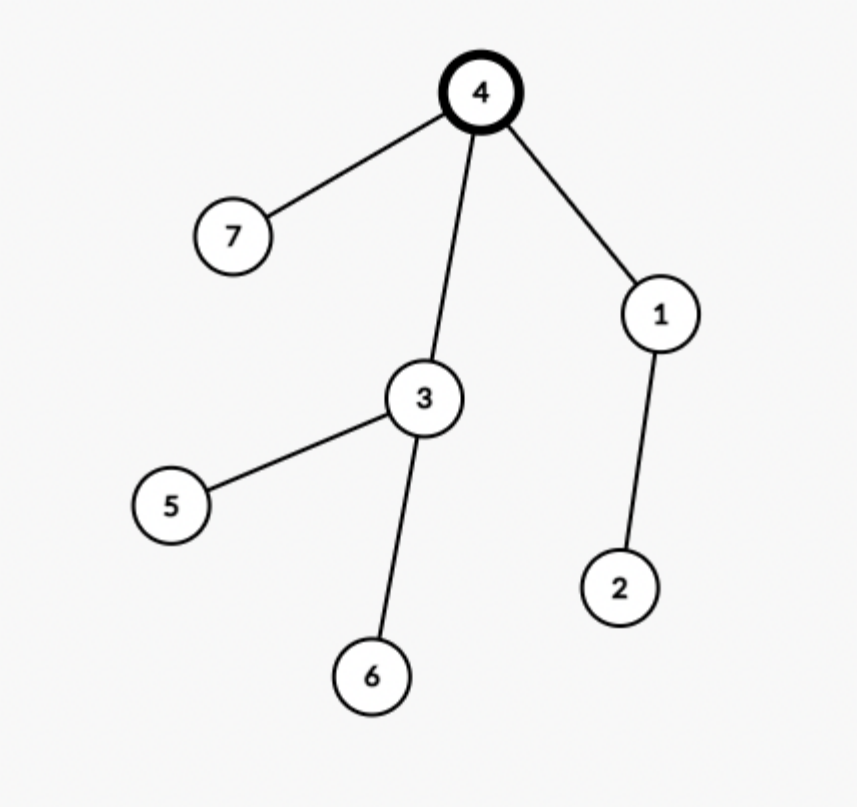
Está garantizado que para cada caso de prueba el árbol y su raíz son fijados antes de comenzar la interacción. En otras palabras, **el interactuador no es adaptativo**.

No olvides de hacer "flush" luego de cada consulta.

Ejemplo

```
Entrada:
7
4 1
1 2
4 3
3 5
3 6
4 7
Salida:
? 2 5 6
Entrada:
NO
Salida:
? 3 6 3 5
Entrada:
YES
Salida:
? 2 1 7
Entrada:
NO
Salida:
? 2 4 6
Entrada:
YES
Salida:
! 4
```

Notas



La raíz escondida es el vértice 4.

En la primera pregunta, el mínimo ancestro común de los vértices 5 y 6 es el vértice 3, que no es parte del conjunto de consulta, así que la respuesta es "`NO`".

En la segunda pregunta, el mínimo ancestro común de los vértices 3, 5 y 6 es el vértice 3, así que la respuesta es "`YES`".

En la tercera pregunta, el mínimo ancestro común de los vértices 1 y 7 es el vértice 4 así que la respuesta es "`NO`".

En la cuarta pregunta, el mínimo ancestro común de los vértices 4 y 6 es el vértice 4 así que la respuesta es "`YES`".

Luego de eso, puedes adivinar que la raíz es el vértice 4, que es la respuesta correcta.

Puntuación

Sea k el máximo número de consultas que puedes hacer para una subtarea. Puedes hacer a lo mucho 1000 consultas, así que $k \leq 1000$.

1. (7 puntos): $n \leq 9$

2. (10 puntos): $n \leq 30$
3. (hasta 83 puntos): $n \leq 500$

En la tercera subtarea, si $k \leq 9$, obtendrás 83 puntos. De otra manera, obtendrás $\lfloor \max(10, 83 \cdot (1 - \frac{\ln(k-6)}{7})) \rfloor$ puntos.

Por consiguiente, para obtener todos los puntos, tu solución debe hacer a lo mucho 9 consultas en cada caso de prueba de la subtarea.

El código de C++ que calcula el número de puntos:

```
((k <= 9) ? 83: max(10, int(83 * (1 - log(k - 6.0) / 7))))
```