Simurgh

Dans les anciennes légendes persanes de Shahnameh, Zal, le légendaire héros persan, est follement amoureux de Rudaba, la princesse de Kabul. Quand Zal a demandé la main de Rudaba, son père lui a proposé un défi.

Il y a n villes en Perse, numérotées de 0 à $n-1$, et m routes à double sens, numérotées de 0 à $m-1$. Chaque route relie deux villes distinctes. Chaque paire de villes est reliée par une route au plus. Certaines des routes sont des *routes royales* et sont utilisées pour les voyages du roi. La tâche de Zal est de déterminer lesquelles des routes sont des routes royales.

Zal dispose d'une carte de toutes les villes et routes de Perse. Il ne sait pas lesquelles des routes sont royales, mais il peut demander l'aide de Simurgh, l'oiseau mythique bienveillant, protecteur de Zal. Cependant, Simurgh ne veut pas directement révéler l'ensemble des routes royales. Au lieu de cela, elle informe Zal que l'ensemble des routes royales est un *ensemble d'or*. Un ensemble de routes est un ensemble d'or si et seulement si :

- Il comprend *exactement* $n-1$ routes.
- Pour chaque paire de villes, il est possible de se déplacer de l'une vers l'autre en utilisant seulement les routes de cet ensemble.

En outre, Zal peut poser des questions à Simurgh. Pour chaque question :

1. Zal choisit un *ensemble d'or* de routes,
2. Simurgh révèle à Zal le nombre de routes royales parmi l'ensemble d'or choisi.

Votre programme doit aider Zal à trouver l'ensemble des routes royales en posant à Simurgh au plus q questions. L'évaluateur jouera le rôle de Simurgh.

Détails d'implémentation

Vous devez implémenter les fonctions suivantes :

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

- n : nombre de villes,
- u et v : tableaux de taille m. Pour tout $0 \leq i \leq m-1$, $u[i]$ et $v[i]$ sont les villes reliées par la route i.
- Cette fonction doit renvoyer un tableau de taille $n-1$ contenant les indices des routes royales (dans un ordre arbitraire).

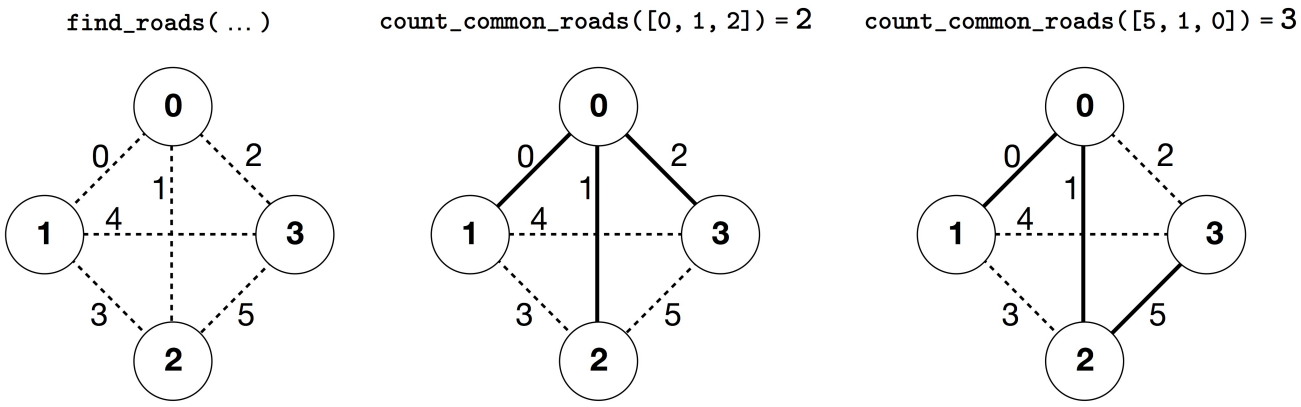
Votre solution peut faire au plus q appels à la fonction suivante de l'évaluateur :

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

- r : tableau de taille $n-1$ contenant les indices des routes d'un ensemble d'or (dans un ordre arbitraire).
- Cette fonction renvoie le nombre de routes royales dans l'ensemble r.
- Remarque : le mot "common" dans le nom de la fonction réfère au fait qu'on compte le nombre de routes en commun entre l'ensemble des routes royales et l'ensemble r.

Exemple

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



Dans cet exemple, il y a 4 villes et 6 routes. On désigne par (a, b) la route qui relie les villes a et b. Les routes sont numérotées de 0 à 5 dans cet ordre: $(0, 1)$, $(0, 2)$, $(0, 3)$, $(1, 2)$, $(1, 3)$ et $(2, 3)$. Tout ensemble d'or doit avoir $n - 1 = 3$ routes.

Supposons que les routes royales soient les routes d'indices 0, 1, et 5, c'est-à-dire les routes $(0, 1)$, $(0, 2)$, et $(2, 3)$ et que le programme fasse les appels suivants:

- `count_common_roads([0, 1, 2])` renvoie 2. Cette requête porte sur les routes numérotées $0, 1$ et 2, autrement dit les routes $(0, 1)$, $(0, 2)$ et $(0, 3)$. Deux d'entre elles sont des routes royales.
- `count_common_roads([5, 1, 0])` renvoie 3. Cette requête porte sur l'ensemble de toutes les routes royales.

La fonction `find_roads` doit renvoyer `[5, 1, 0]` ou tout autre tableau de taille 3 contenant ces trois éléments.

Notez bien que les appels suivants ne sont pas permis :

- `count_common_roads([0, 1])`: la taille de r n'est pas 3.
- `count_common_roads([0, 1, 3])`: r ne décrit pas un ensemble d'or, en effet, il est

impossible de passer de la ville 0 à la ville 3 en utilisant seulement les routes $(0, 1)$, $(0, 2)$, $(1, 2)$.

Contraintes

- $2 \leq n \leq 500$
- $n - 1 \leq m \leq n(n-1)/2$
- $0 \leq u[i], v[i] \leq n - 1$ (pour tout $0 \leq i \leq m - 1$)
- Pour tout $0 \leq i \leq m - 1$, la route i relie deux villes distinctes (i.e., $u[i] \neq v[i]$).
- Il y a au plus une route entre chaque paire de villes.
- Il existe un chemin entre toute paire de villes.
- L'ensemble de toutes les routes royales est un ensemble d'or.
- `find_roads` doit appeler `count_common_roads` au plus q fois. Pour chaque appel, l'ensemble de routes décrit par r doit être un ensemble d'or.

Sous-tâches:

1. (13 points) $n \leq 7$, $q = 30\,000$
2. (17 points) $n \leq 50$, $q = 30\,000$
3. (21 points) $n \leq 240$, $q = 30\,000$
4. (19 points) $q = 12000$, de plus il y a une route entre chaque paire de villes.
5. (30 points) $q = 8000$

Évaluateur d'exemple

L'évaluateur d'exemple lit l'entrée dans le format suivant:

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

Ici, $s[0], s[1], \ldots, s[n-2]$ sont les indices des routes royales.

L'évaluateur d'exemple affiche `YES`, si `find_roads` appelle `count_common_roads` au plus $30\,000$ fois, et renvoie le bon ensemble de routes royales. Sinon, il affiche `NO`.

Noter que la fonction `count_common_roads` dans l'évaluateur d'exemple ne vérifie pas si r respecte toutes les proprietés d'un ensemble d'or. Au lieu de cela, elle calcule et renvoie le nombre de routes royales dont les indices sont dans r. Cependant, si le programme que vous soumettez appelle `count_common_roads` avec des indices de routes ne formant pas un ensemble d'or, le verdict de l'évaluateur sur CMS sera 'Wrong Answer'.

Note Technique

La fonction `count_common_roads` en C++ et Pascal utilise la méthode *pass by reference* pour des raisons d'efficacité. Vous pouvez toujours appeler la fonction de la manière habituelle. On garantit que l'évaluateur ne va pas changer la valeur de r.