# Loup-garou (Werewolf)

Il y a $N$ villes et $M$ routes dans la préfecture japonaise d'Ibaraki. Les villes sont numérotées de $0$ à $N-1$ par ordre croissant de leur population. Chaque route connecte une paire de villes distinctes, et peut être parcourue dans les deux sens. Vous pouvez voyager de n'importe quelle ville à n'importe quelle autre en utilisant une ou plusieurs routes.

Vous avez planifié $Q$ excursions, numérotées de $0$ à $Q-1$. L'excursion $i$ $(0 \leq i \leq Q-1)$ vous fait voyager de la ville $S_i$ à la ville $E_i$.

Vous êtes un loup-garou. Vous avez deux formes : une **forme humaine** et une **forme de loup**. Au début de chacune de vos excursions, vous êtes en forme humaine. À la fin de chacune de vos excursions, vous devez être en forme de loup. Au cours de votre excursion, vous devez vous **transformer** (changer de la forme humaine à la forme de loup) exactement une fois et cette transformation doit avoir lieu alors que vous êtes dans l'une des villes (possiblement $S_i$ ou $E_i$).

La vie d'un loup-garou n'est pas facile. Par expérience, vous savez qu'il vaut mieux éviter les villes peu peuplées quand vous êtes sous forme humaine et les villes fortement peuplées quand vous êtes en forme de loup. Plus précisément, pour chaque excursion $i$ $(0 \leq i \leq Q-1)$, vous avez choisi deux entiers $L_i$ et $R_i$ satisfaisant $0 \leq L_i \leq R_i \leq N-1$ qui indiquent quelles villes éviter. Pour l'excursion $i$, vous devez éviter les villes $0, 1, \ldots, L_i - 1$ en forme humaine et éviter les villes $R_i + 1, R_i + 2, \ldots, N-1$ quand vous êtes en forme de loup. En particulier, cela veut dire que vous aurez à vous transformer dans l'une des villes $L_i, L_i + 1, \ldots, R_i$ lors de l'excursion $i$.

Votre tâche est de déterminer, pour chaque excursion, s'il vous est possible de voyager de la ville $S_i$ à la ville $E_i$ en respectant les contraintes précisées ci-dessus. La longueur du chemin que vous empruntez n'importe pas.

## Détails d'implémentation

Votre code doit définir la fonction suivante :

```
int[] check_validity(int N, int[] X, int[] Y, int[] S, int[] E, int[]
 L, int[] R)
```

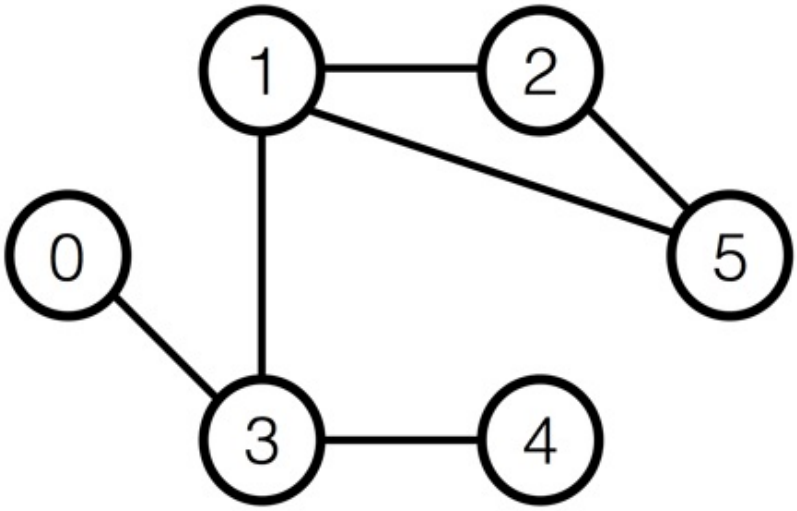
- `N`: le nombre de villes.
- `X` et `Y` : des tableaux de longueur $M$. Pour chaque $j$ ($0 \leq j \leq M-1$), la ville `X[j]` est connectée directement à la ville `Y[j]` par une route.
- `S`, `E`, `L`, et `R` : des tableaux de longueur $Q$, représentant les excursions.

La fonction `check_validity` est appelée exactement une fois pour chaque test. Cette fonction doit renvoyer un tableau $A$ d'entiers de longueur $Q$. La valeur d'un $A_i$ ( $0 \leq i \leq Q-1$) doit être $1$ si une excursion respectant les conditions énoncées plus-haut est possible. Sinon, ça doit être $0$.

## Exemple

Soient $N = 6$, $M = 6$, $Q = 3$, $X = [5, 1, 1, 3, 3, 5]$, $Y = [1, 2, 3, 4, 0, 2]$, $S = [4, 4, 5]$, $E = [2, 2, 4]$, $L = [1, 2, 3]$, et $R = [2, 2, 4]$.

L'évaluateur appelle `check_validity(6, [5, 1, 1, 3, 3, 5], [1, 2, 3, 4, 0, 2], [4, 4, 5], [2, 2, 4], [1, 2, 3], [2, 2, 4])`.




Pour l'excursion $0$, vous pouvez voyager de la ville $4$ à la ville $2$ de la façon suivante :

- Commencer à la ville $4$ (vous êtes en forme humaine)
- Se déplacer à la ville $3$ (vous êtes en forme humaine)
- Se déplacer à la ville $1$ (vous êtes en forme humaine)
- Vous transformer en loup (vous êtes en forme de loup)
- Se déplacer à la ville $2$ (vous êtes en forme de loup)

Pour les excursions $1$ et $2$ vous ne pouvez pas voyager entre les deux villes.

Votre programme doit donc renvoyer $[1, 0, 0]$.

Les fichiers `sample-01-in.txt` et `sample-01-out.txt` dans l'archive zip du sujet correspondent à cet exemple. D'autres exemples d'entrées/sorties sont disponibles dans l'archive.

## Contraintes

- $2 \leq N \leq 200\,000$
- $N - 1 \leq M \leq 400\,000$
- $1 \leq Q \leq 200\,000$
- For each $0 \leq j \leq M - 1$
  - $0 \leq X_j \leq N - 1$
  - $0 \leq Y_j \leq N - 1$
  - $X_j \neq Y_j$
- Vous pouvez voyager de toute ville à toute autre en utilisant les routes.
- Chaque paire de ville n'est connectée directement par au plus une route. Autrement dit, pour tout $0 \leq j < k \leq M - 1$, $(X_j, Y_j) \neq (X_k, Y_k)$ et $(Y_j, X_j) \neq (X_k, Y_k)$.
- Pour chaque $0 \leq i \leq Q - 1$
  - $0 \leq L_i \leq S_i \leq N - 1$
  - $0 \leq E_i \leq R_i \leq N - 1$
  - $S_i \neq E_i$
  - $L_i \leq R_i$

## Sous-tâches

1. (7 points) $N \leq 100$, $M \leq 200$, $Q \leq 100$
2. (8 points) $N \leq 3\,000$, $M \leq 6\,000$, $Q \leq 3\,000$
3. (34 points) $M = N - 1$ et aucune ville n'est connectée à plus de 2 autres villes (les villes sont connectées en une ligne)
4. (51 points) Aucune contrainte additionnelle

## Évaluateur d'exemple

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

- ligne 1: $N$ $M$ $Q$
- linge $2 + j$ ($0 \leq j \leq M - 1$) : $X_j$ $Y_j$
- ligne $2 + M + i$ ($0 \leq i \leq Q - 1$) : $S_i$ $E_i$ $L_i$ $R_i$

L'évaluateur d'exemple affiche la valeur de retour de `check_validity` dans le format suivant :

- ligne $1 + i$ ($0 \leq i \leq Q - 1$) : $A_i$