



Problème Navigation

 3 s  512 Mo

On a un **graphe cactus simple, connexe et non-orienté**¹ avec $N \leq 1000$ nœuds et M arêtes. Ses nœuds ont des couleurs (notées avec des entiers positifs entre 0 et 1499). Initialement, tous les nœuds ont la couleur 0. Un **robot déterministe sans mémoire**² explore le graphe en se déplaçant de nœud en nœud. Il doit visiter tous les nœuds au moins une fois puis s'arrêter.

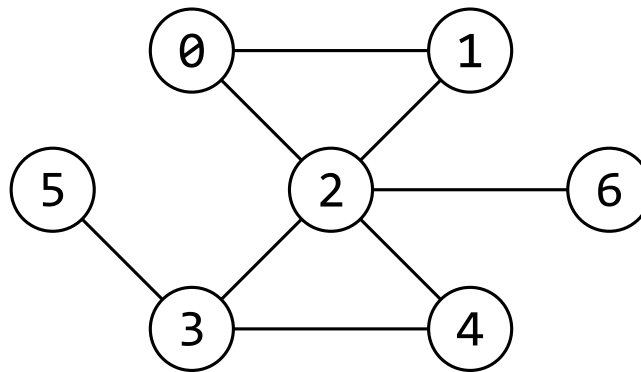
Le robot commence sur un nœud quelconque du graphe. À chaque étape, il voit la couleur du nœud actuel et les couleurs de tous les nœuds adjacents **dans un ordre qui est fixé pour le nœud actuel** (c'est-à-dire que revisiter le nœud donnera au robot la même séquence de nœuds adjacents, même si leurs couleurs ont changé depuis). Le robot effectue l'une des deux actions suivantes :

1. il décide de s'arrêter.
2. il choisit une nouvelle couleur (qui peut être la couleur actuelle) pour le nœud actuel et vers quel nœud adjacent se déplacer. Le nœud adjacent est identifié par un indice entre 0 et $D - 1$, où D est le nombre de nœuds adjacents.

Dans le deuxième cas, le nœud actuel est recoloré (ou éventuellement conserve sa couleur) et le robot se déplace vers le nœud adjacent choisi. Cela se répète jusqu'à ce que le robot s'arrête ou jusqu'à ce qu'il atteigne la limite d'itérations. Le robot gagne s'il visite tous les nœuds, puis s'arrête en respectant la limite de $L = 3000$ itérations (sinon il perd).

Vous devez mettre en place une stratégie pour le robot qui résolve le problème sur n'importe quel graphe cactus tel que décrit. De plus, vous devez essayer de minimiser le nombre de couleurs distinctes que votre solution utilise. Ici, la couleur 0 est toujours comptée comme utilisée.

¹Un *graphe cactus simple, connexe et non-orienté* est un *graphe simple, connexe et non-orienté* (chaque nœud est atteignable depuis chaque autre nœud ; les arêtes sont bidirectionnelles ; il n'y a pas de boucle, c'est-à-dire une arête d'un nœud vers lui-même, ni d'arêtes multiples reliant une même paire de nœuds) dans lequel chaque arête appartient à un seul cycle simple (un cycle simple est un cycle qui ne contient chaque nœud qu'une fois). L'image ci-dessous est un exemple.



²Un robot est déterministe sans mémoire, si son action ne dépend que de ses entrées (c'est-à-dire qu'il ne stocke pas de données d'une étape à l'autre), et qu'il choisit toujours la même action quand on lui donne les mêmes entrées.



Détails d'implémentation

La stratégie du robot doit être implémentée via la fonction suivante :

```
std::pair<int, int> navigate(int currColor, std::vector<int> adjColors)
```

Elle reçoit en paramètre la couleur du nœud actuel et les couleurs de tous les nœuds adjacents (dans l'ordre). Elle doit renvoyer une paire dont le premier élément est la nouvelle couleur du nœud actuel et dont le deuxième élément est l'indice du nœud adjacent auquel le robot doit se déplacer. Sinon, si le robot doit s'arrêter, la fonction doit renvoyer la paire $(-1, -1)$.

Cette fonction sera appelée de manière répétée de manière à choisir les actions du robot. Puisqu'elle est déterministe, si `navigate` a déjà été appelée avec certains paramètres, elle ne sera jamais rappelée avec ces paramètres à nouveau ; on va plutôt réutiliser la valeur de retour précédente. De plus, chaque test peut contenir $T \leq 5$ sous-tests (des graphes et/ou des positions de départ distincts) et ils peuvent être lancés en parallèle (c'est-à-dire que votre programme peut recevoir en alternance des appels pour des sous-tests différents). Enfin, les appels à `navigate` peuvent se produire dans **des exécutions séparées** de votre programme (mais ils pourraient aussi parfois se produire dans la même exécution). Le nombre total d'exécutions de votre programme est $P = 100$. Pour toutes ces raisons, votre programme ne devrait pas essayer de passer de l'information entre les différents appels.



Contraintes

- $3 \leq N \leq 1000$
- $0 \leq \text{Couleur} < 1500$
- $L = 3000$
- $T \leq 5$
- $P = 100$



Notation

La fraction S des points que vous recevrez pour une sous-tâche dépend de C , le nombre maximum de couleurs distinctes que votre solution utilise (dont la couleur 0) sur chacun des tests de cette sous-tâche ou n'importe laquelle des sous-tâches requises :

- Si votre solution échoue sur n'importe quel sous-test, alors $S = 0$.
- Si $C \leq 4$, alors $S = 1.0$.
- Si $4 < C \leq 8$, alors $S = 1.0 - 0.6 \frac{C-4}{4}$.
- Si $8 < C \leq 21$, alors $S = 0.4 \frac{8}{C}$.
- Si $C > 21$, alors $S = 0.15$.



Sous-tâches

Sous-tâche	Points	Sous-tâches requises	N	Contraintes supplémentaires
0	0	—	≤ 300	L'exemple.
1	6	—	≤ 300	Le graphe est un cycle. ¹
2	7	—	≤ 300	Le graphe est une étoile. ²
3	9	—	≤ 300	Le graphe est un chemin. ³
4	16	2 – 3	≤ 300	Le graphe est un arbre. ⁴
5	27	—	≤ 300	Tous les nœuds ont au plus 3 nœuds adjacents et le nœud où le robot commence a un seul nœud adjacent.
6	28	0 – 5	≤ 300	—
7	7	0 – 6	—	—

¹Un cycle a les arêtes : $(i, (i + 1) \bmod N)$ pour chaque $0 \leq i < N$.

²Une étoile a les arêtes : $(0, i)$ pour chaque $1 \leq i < N$.

³Un chemin a les arêtes : $(i, i + 1)$ pour chaque $0 \leq i < N - 1$.

⁴Un arbre est un graphe sans cycles.



Exemple

On considère le graphe de l'image ci-dessus dans l'énoncé, qui a $N = 7$, $M = 8$ et les arêtes $(0, 1)$, $(1, 2)$, $(2, 0)$, $(2, 3)$, $(3, 4)$, $(4, 2)$, $(3, 5)$ et $(2, 6)$. De plus, puisque les ordres des éléments dans les listes d'adjacence des nœuds sont importants, ils sont donnés dans ce tableau :

Nœud	Nœuds adjacents
0	2, 1
1	2, 0
2	0, 3, 4, 6, 1
3	4, 5, 2
4	2, 3
5	3
6	2

On suppose que le robot commence au nœud 5. Voici une séquence possible (perdante) d'interactions :

#	Couleurs	Nœud	Appel à navigater	Valeur de retour
1	0, 0, 0, 0, 0, 0, 0	5	navigate(0, {0})	{1, 0}
2	0, 0, 0, 0, 0, 1, 0	3	navigate(0, {0, 1, 0})	{4, 2}
3	0, 0, 0, 4, 0, 1, 0	2	navigate(0, {0, 4, 0, 0, 0})	{0, 3}
4	0, 0, 0, 4, 0, 1, 0	6	¹ navigate(0, {0})	{1, 0}
5	0, 0, 0, 4, 0, 1, 1	2	navigate(0, {0, 4, 0, 1, 0})	{8, 0}
6	0, 0, 8, 4, 0, 1, 1	0	navigate(0, {8, 0})	{3, 0}
7	3, 0, 8, 4, 0, 1, 1	2	navigate(8, {3, 4, 0, 1, 0})	{2, 2}
8	3, 0, 2, 4, 0, 1, 1	4	navigate(0, {2, 4})	{1, 1}
9	3, 0, 2, 4, 1, 1, 1	3	navigate(4, {1, 1, 2})	{-1, -1}

Ici, le robot a utilisé un total de 6 couleurs distinctes: 0, 1, 2, 3, 4 et 8 (remarquez que 0 aurait compté comme utilisée même si le robot n'avait jamais renvoyé la couleur 0 puisque tous les nœuds commencent avec la couleur 0). Le robot s'est exécuté pour 9 itérations avant de s'arrêter. Toutefois, il a échoué puisqu'il s'est arrêté sans avoir visité le nœud 1.



¹Remarquez que l'appel à `navigate` à l'itération 4 ne se produirait pas vraiment. En effet, il est équivalent à celui de l'appel à l'itération 1, donc l'évaluateur réutiliserait simplement la valeur renvoyée par votre fonction sur ce premier appel. Néanmoins, cela compte tout de même comme une itération du robot.



Évaluateur d'exemple

L'évaluateur d'exemple ne lance pas plusieurs exécutions de votre programme, donc tous les appels à `navigate` seront dans la même exécution de votre programme.

Le format d'entrée est le suivant : Tout d'abord, on lit T , le nombre de sous-tests. Puis pour chaque sous-test :

- ligne 1 : les deux entiers N et M ;
- ligne $2 + i$ (pour chaque $0 \leq i < M$) : les deux entiers A_i et B_i , qui sont les deux nœuds que l'arête i relie ($0 \leq A_i, B_i < N$).

L'évaluateur d'exemple affichera ensuite le nombre de couleurs distinctes que votre solution a utilisé et le nombre d'itérations dont elle a eu besoin avant de s'arrêter. Autrement, elle affichera un message d'erreur si votre solution a échoué.

Par défaut, l'évaluateur d'exemple affiche des informations détaillées sur ce que le robot voit et fait à chaque itération. Vous pouvez désactiver ceci en changeant la valeur de `DEBUG` de `true` à `false`.