Permutation Game

Alice et Bob sont des amis d'enfance qui aiment jouer à des jeux éducatifs. Aujourd'hui, Ils jouent à un nouveau jeu sur les graphes.

Le jeu contient un graphe **connexe** avec m sommets, numérotés de 0 à $m-1$, et e arêtes, numérotées de 0 à $e-1$. L'arête i connecte les sommets $u[i]$ et $v[i]$.

Le jeu contient aussi une permutation $p[0], p[1], \ldots, p[n-1]$ de longueur n, où $m \leq n$. Une permutation est un tableau où chaque nombre de 0 à $n-1$ apparaît exactement une fois, dans un ordre quelconque. Le **score** d'une permutation p est le nombre d'indices i tels que $p[i] = i$.

Le jeu va durer au plus 10^{100} tours. A chaque tour :

1. Si Alice décide de finir le jeu, le jeu s'arrête.
2. Sinon, Alice choisit **des indices distincts** $t[0], t[1], \ldots, t[m-1]$, où $0 \leq t[i] < n$. Notez que, le jeu **ne nécessite PAS** que $t[0] < t[1] < \ldots < t[m-1]$.
3. Bob choisit un indice $0 \leq j < e$ des arêtes du graphe et échange $p[t[u[j]]]$ et $p[t[v[j]]]$.

Alice veut maximiser le score final de la permutation tandis que Bob veut minimiser le score final de la permutation.

Votre tâche est d'aider Alice et de jouer contre Bob, tandis que les mouvements de Bob sont simulés par le juge.

On définit le score optimal comme le score final de la permutation si Alice et Bob jouent optimalement.

Vous devez déterminer le score optimal de la permutation, et ensuite jouer au jeu avec Bob pour **au moins** obtenir ce score là au bout d'un nombre fini de tours.

Notez que la stratégie d'Alice doit fonctionner peut importe ce que Bob fait, même s'il n'effectue pas les mouvements optimaux.

Détails d'implémentation

Vous devez implémenter la fonction suivante:

```
int Alice(int m, int e, std::vector<int> u, std::vector<int> v,
          int n, std::vector<int> p)
```

- m: le nombre de sommets dans le graphe.
- e: le nombre d'arêtes dans le graphe.
- u et v: tableaux de taille e décrivant les arêtes du graphe.
- n: la longueur de la permutation.
- p: un tableau de taille n décrivant la permutation.
- La fonction est appelée exactement une fois.
- La fonction doit retourner un entier – le score optimal du jeu.

Dans cette fonction, vous pouvez appeler la fonction suivante:

```
int Bob(std::vector<int> t)
```

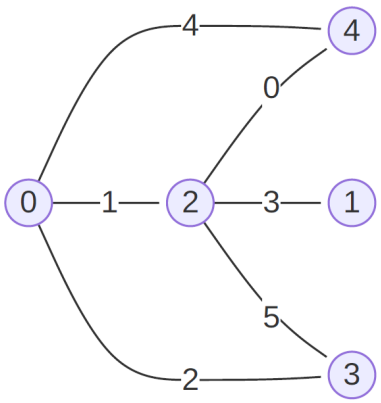
- t: un tableau de taille m, contenant des indices distincts, où $0 \leq t[i] < n$ et $t[i] \neq t[j]$ pour tous $i \neq j$.
- La fonction retourne un entier j satisfaisant $0 \leq j < e$.
- Cette fonction peut être appelée plusieurs fois.

Exemple

Considérez l'appel suivant:

```
Alice(5, 6, [4, 0, 3, 1, 4, 2], [2, 2, 0, 2, 0, 3],
      10, [8, 2, 7, 6, 1, 5, 0, 9, 3, 4])
```

Le graphe est comme suit:



et p est initialement $[8, 2, 7, 6, 1, 5, 0, 9, 3, 4]$.

Au vu des contraintes précédentes, on peut démontrer que le score optimal de la permutation est 1 .

Supposez que, Alice effectue les 4 mouvements:

Argument de t à `Bob`	Valeur retournée par `Bob`	Indices correspondants de p	p après l'échange de Bob
$[3,1,5,2,0]$	5	$5,2$	$[8,2,5,6,1,7,0,9,3,4]$
$[9,3,7,2,1]$	0	$1,7$	$[8,9,5,6,1,7,0,2,3,4]$
$[5,6,7,8,9]$	1	$5,7$	$[8,9,5,6,1,2,0,7,3,4]$
$[7,5,2,3,6]$	3	$5,2$	$[8,9,2,6,1,5,0,7,3,4]$

Notez que les coups d'Alice et de Bob ci-dessus ne sont pas nécessairement les coups optimaux. Les coups montrés sont purement à titre de démonstration. Notez aussi que si Alice peut finir le jeu directement, vu que le score initial est déjà 1.

Après qu'Alice ait effectué tous les coups ci-dessus, le score actuel de la permutation est 3 ($p[2]=2, p[5]=5, p[7]=7$).

La fonction `Alice` retournera finalement 1, le score optimal de la permutation.

Notez que même si Alice a obtenu un score de 3 en jouant avec Bob, vous obtiendriez 0 points si la valeur de retour de `Alice()` était 3 au lieu de 1.

Constraints

- $2 \le m \le 400$
- $m-1 \le e \le 400$
- $0 \le u[i], v[i] < m$
- $m \le n \le 400$
- $0 \le p[i] < n$
- Le graphe est connexe, ne contient pas de boucles sur un même sommet ni d'arêtes multiples.
- p est une permutation, c'est-à-dire $p[i] \ne p[j]$ pour tout $i \ne j$.

Sous-tâches

1. (6 points) $m=2$
2. (6 points) $e>m$
3. (10 points) $e=m-1$
4. (24 points) $e=m=3$
5. (24 points) $e=m=4$
6. (30 points) $e=m$

Pour chaque sous-tâche, vous pouvez obtenir un score partiel. Soit r le ratio maximum de $\frac{k}{n}$ parmi tous les cas de test d'une sous-tâche, où k est le nombre de tours (c'est-à-dire le nombre d'appels à `Bob()`). Alors, votre score pour cette sous-tâche est multiplié par le nombre suivant :

Condition	Coefficient
$12 \leq r$	0
$3 < r < 12$	$1 - \log_{10}(r-2)$
$r \leq 3$	1

En particulier, Si le problème est résolu en $3n$ tours, vous obtenez la note complète pour la sous-tâche. Utiliser plus de $12n$ tours aura pour conséquence un 0 pour la sous-tâche (affiché comme `Output isn't correct`).

Evaluateur d'exemple

L'évaluateur d'exemple lit l'entrée au format suivant:

- ligne 1: $m\ e$
- ligne $2+i$ $(0 \leq i \leq e-1)$: $u[i]\ v[i]$
- ligne $2+e$: n
- ligne $3+e$: $p[0]\ p[1]\ \ldots\ p[n-1]$

L'évaluateur d'exemple affiche la sortie au format suivant:

- ligne 1: permutation finale p
- ligne 2: valeur retournée par `Alice()`
- ligne 3: score actuel de la permutation finale
- ligne 4: le nombre de tours