Papa et bébé grenouille

Fichier d'entrée	Fichier de sortie	Limite de temps	Limite de mémoire
entrée standard	sortie standard	1 seconde	256 MiB

Papa et bébé grenouille vivent sur une ligne de n rochers **régulièrement espacés**, numérotés de 1 à n de gauche à droite. Le rocher i a une hauteur entière strictement positive h_i.

Les grenouilles peuvent sauter de pierre en pierre, en suivant les règles suivantes :

- En un saut, papa grenouille se déplace toujours vers le rocher le **plus proche** dont la hauteur est **strictement supérieure** à la hauteur du rocher actuel. Si il existe plusieurs possibilités, il choisit le rocher **le plus à droite**.
- En un saut, bébé grenouille se déplace toujours vers le rocher le **plus proche** dont la hauteur est **strictement inférieure** à la hauteur du rocher actuel. Si il existe plusieurs possibilités, il choisit le rocher le **plus à droite**.

Papa grenouille aimerait choisir un rocher sur lequel construire une *maison*, et un rocher **différent** sur lequel construire une *école*. Il veut que :

- Papa grenouille puisse se rendre de la maison à l'école, en faisant au plus k sauts.
- Bébé grenouille puisse se rendre de l'école à la maison, en faisant au plus k sauts.

Un rocher est une *maison potentielle* si il existe **au moins** un autre rocher sur lequel on peut construire l'école, si la maison est construite sur ce premier rocher.

Votre objectif est de déterminer quels rochers sont des maisons potentielles.

Sous-tâches et Contraintes

Pour chaque sous-tâche, on vous garantit que :

- $n \geq 2$
- $1 \leq k \leq n$
- Pour chaque rocher i, $1 \leq h_i \leq 1\,000\,000\,000$

Les contraintes supplémentaires de chaque sous-tâche sont données ci-dessous.

Sous-tâche	Points	n	k
1	10	$n \leq 2000$	$k = 1$
2	10	$n \leq 2000$	$k \leq 5$
3	20	$n \leq 2000$	$k \leq 100$
4	20	$n \leq 2000$	$k \leq n$
5	20	$n \leq 100\,000$	$k \leq 5$
6	10	$n \leq 100\,000$	$k = n$
7	10	$n \leq 100\,000$	$k \leq n$

Entrée

La première ligne de l'entrée contient deux entiers, n et k.

La seconde ligne de l'entrée contient n entiers $h_1, \ldots, h_n$, indiquant les hauteurs des rochers.

Sortie

La sortie doit contenir une chaîne de n caractères sur une seule ligne. Le i-ème caractère doit être `1` si le rocher i est une maison potentielle, ou `0` sinon.

Entrée d'exemple 1

```
4 1
4 1 3 2
```

Sortie d'exemple 1

```
0001
```

Entrée d'exemple 2

```
7 2
3 1 2 1 2 2 3
```

Sortie d'exemple 2

```
0101010
```

Entrée d'exemple 3

```
10 3
10 5 6 4 7 3 8 2 9 1
```

Sortie d'exemple 3

```
0000010001
```

Explications

Dans le premier exemple, la maison ne peut être construite que sur le rocher 4, avec l'école construite sur le rocher 3. Par conséquent, la seule maison potentielle est le rocher 4.

Dans le deuxième exemple :

- La maison peut être construite sur le rocher 2, avec l'école sur le rocher 1. Papa grenouille saute alors sur les rochers $2 \to 3 \to 1$, et bébé grenouille saute directement du rocher 1 au rocher 2.

- La maison peut être construite sur le rocher 4, avec l'école sur le rocher 5. Papa et bébé grenouille peuvent alors sauter directement de la maison à l'école, et réciproquement.
- La maison peut être construite sur le rocher 6, avec l'école sur le rocher 7. Papa et bébé grenouille peuvent alors sauter directement de la maison à l'école, et réciproquement.

Par conséquent, les rochers 2, 4 et 6 sont des maisons potentielles.

Dans le troisième exemple :

- La maison peut être construite sur le rocher 6, avec l'école sur le rocher 1. Papa grenouille saute alors sur les rochers $6 \to 7 \to 9 \to 1$, et bébé grenouille saute sur les rochers $1 \to 2 \to 4 \to 6$.
- La maison peut être construite sur le rocher 10, avec l'école sur le rocher 9. Papa et bébé grenouille peuvent alors sauter directement de la maison à l'école, et réciproquement.

Par conséquent, les rochers 6 et 10 sont des maisons potentielles.

Chemin coloré

Fichier d'entrée	Fichier de sortie	Limite de temps	Limite de mémoire
entrée standard	sortie standard	2 secondes	1024 MiB

Une exposition d'art vient d'être inaugurée dans les allées d'un centre commercial. Il s'agit d'un long chemin de n carreaux, numérotés de 1 à n de gauche à droite. Chaque carreau est coloré avec une des c couleurs possibles, numérotées de 1 à c. Le i-ème carreau est de la couleur x_i.

Vous commencez sur le carreau le plus à gauche, et devez vous rendre sur le carreau le plus à droite. Vous ne pouvez vous déplacer qu'en faisant des sauts vers la droite. Grâce à vos jambes puissantes, vous êtes capable de sauter par dessus un nombre arbitraire de carreaux à chaque saut.

Mais les choses ne sont pas si simples. L'artiste insiste sur le fait que vous ne pouvez sauter d'un carreau coloré à un autre carreau coloré que si les couleurs sont *compatibles* ! Vous avez une liste de p paires de couleurs compatibles (une couleur peut être ou peut ne pas être compatible avec elle même). La j-ème paire indique que les couleurs a_j et b_j sont compatibles

Les couleurs sont particulièrement jolies, vous avez donc envie de compter le nombre de manières possibles pour vous rendre du carreau le plus à gauche au carreau le plus à droite. Deux possibilités sont différentes si il existe un carreau par lequel vous passez dans une mais pas dans l'autre. La réponse pouvant être grande, donnez votre réponse modulo $1\,000\,000\,007$.

Comme ce nombre peut être grand, vous pouvez avoir envie d'utiliser des entiers sur 64 bits (comme les `long long` en C++) dans votre programme.

Sous-tâches et Contraintes

Dans toutes les sous-tâches :

- $2 \leq n \leq 100\,000$
- $1 \leq c \leq 100\,000$
- $1 \leq x_i \leq c$, pour tout i
- $0 \leq p \leq 100\,000$
- $1 \leq a_j, b_j \leq c$ pour tout j (Notez qu'il est possible d'avoir $a_j = b_j$)
- $(a_i, b_i) \neq (a_j, b_j)$ et $(a_i, b_i) \neq (b_j, a_j)$ pour tout $i \neq j$. C'est à dire, aucune paire de couleurs compatibles n'apparaitra deux fois.

De plus :

- Dans la sous-tâche 1 (4 points) : $c = 1$
- Dans la sous-tâche 2 (22 points): $c \leq 100$.
- Dans la sous-tâche 3 (15 points): $x_i = i$, pour tout i.
- Dans la sous-tâche 4 (26 points): $p = c - 1$, $a_j = j$ et $b_j > j$ pour tout j.
- Dans la sous-tâche 5 (33 points): Aucune contrainte supplémetaire.

Entrée

La première ligne de l'entrée contient deux entiers n, c. La seconde ligne contient n entiers x_1, x_2, …, x_n.

La troisième ligne contient un seul entier p. Suivent p lignes, décrivant les paires de couleurs compatibles. La j-ème d'entre elles contient les deux entiers a_j et b_j.

Sortie

Affichez un seul entier, le nombre de manières possibles pour passer du carreau le plus à gauche au carreau le plus à droite, modulo 1 000 000 007.

Entrée d'exemple 1

```
5 3
1 2 3 2 3
3
1 2
1 3
2 3
```

Sortie d'exemple 1

```
5
```

Entrée d'exemple 2

```
4 4
1 2 3 4
2
1 2
2 3
```

Sortie d'exemple 2

```
0
```

Entrée d'exemple 3

```
6 2
1 1 1 2 2 1
3
1 1
2 1
2 2
```

Sortie d'exemple 3

```
16
```

Dans le premier exemple il y a 5 possibilités, listées ci-dessous, avec les numéro des carreaux intermédiaires :

- $1 \to 2 \to 3 \to 4 \to 5$
- $1 \to 2 \to 5$
- $1 \to 4 \to 5$
- $1 \to 3 \to 4 \to 5$
- $1 \to 5$

Dans le deuxième exemple, il est impossible d'atteindre le carreau le plus à droite ! La réponse est donc 0.

Dans le troisième exemple, chaque couleur est compatible avec toute autre couleur (y compris avec elle même), tous les sauts sont donc autorisés. Il existe 16 possibilités.

Nav

Fichier d'entrée	Fichier de sortie	Limite de temps	Limite de mémoire
aucun	**aucun**	2 secondes	256 MiB

Dans le dernier jeu vidéo auquel vous avez joué, vous vous retrouvez dans un monde mystérieux. Le monde peut être représenté par un graphe connecté avec n sommets, numérotés de 0 à $n-1$. Il y a m arêtes bi-directionnelles dans ce graphe, numérotées de 0 à $m-1$, chacune reliant deux sommets différents. Toutes les arêtes ont la même longueur. Vous connaissez la structure exacte de ce graphe.

Pour tester vos vastes connaissances, vous allez devoir résoudre t tests.

Dans chaque test, un sommet a secrètement été choisi. Ce sommet est le sommet *principal*, et votre objectif est de l'identifier. Pour vous aider, on vous a fourni un appareil appelé le *Nav*. Le Nav est conçu pour vous aider à trouver le sommet principal. Il vous est possible d'interroger le Nav, de la manière suivante.

1. Vous entrez dans le Nav un sommet de votre choix, appelons le u. Si u est le sommet principal le Nav retournera -1.
2. Sinon, le Nav calcule un **plus court chemin** entre u et le sommet principal. Notez que si il existe plusieurs plus courts chemins, le Nav peut choisir n'importe lequel.
3. Le Nav retourne un entier entre 0 et $m-1$, ce qui correspond à une **arête** qui fait parti du plus court chemin.

Votre objectif est d'identifier le sommet principal, en utilisant le moins de requêtes possible.

Moins vous utilisez de requêtes, meilleur sera votre score pour ce problème. Voir la section *Sous-tâches et Contraintes* pour plus de détails.

Sous-tâches et Contraintes

Pour chaque sous-tâche, on vous garantit que :

- $n \geq 1$
- $n - 1 \leq m \leq 100\,000$
- Le graphe fourni est connexe.
- Chaque arête relie deux sommets *différents*.
- Deux arêtes de relient jamais les mêmes paires de sommets.

Pour chaque sous-tâche, il existe un nombre maximal de requêtes q que vous pouvez faire au Nav **pour chaque test**. Votre score pour une exécution de votre programme (t tests) est calculé de la manière suivante :

- Si vous faîtes plus de q requêtes pour un des tests, vous recevrez 0 points. Sur l'interface en ligne, le jugement obtenu sera `Not correct`.
- Si votre programme dépasse la limite de temps, vous recevrez 0 points.
- Si votre programme ne se termine pas avec succès, ou si il ne parvient pas à identifier le sommet principal, vous recevrez 0 points. Sur l'interface en ligne, le jugement obtenu sera `Not correct`.

- Sinon, si votre programme identifie toujours le sommet principal avec succès
  - Pour les sous-tâches 1 à 3 : vous obtiendrez tous les points disponibles. Sur l'interface en ligne le jugement obtenu sera `Correct`.
  - Pour la sous-tâche 4 : votre score sera calculé en fonction du nombre maximal de requêtes q_{max} que votre programme a fait sur les tests. Plus précisement, votre score sera :
    * 100%, si $q_{\max} \leq 9$;
    * 70%, si $10 \leq q_{\max} \leq 12$; ou
    * $\frac{2}{3} \times \frac{30-q_{\max}}{q_{\max}}$, si $13 \leq q \leq 30$.

Sous-tâche	Points	Maximum t	Maximum n	q	Contraintes additionnelles
1	7	750	$n \leq 300$	300	
2	15	250 000	$n \leq 100\,000$	17	$m = n - 1$, et pour chaque $0 \leq j \leq m - 1$, l'arête j relie le sommet j et le sommet $j + 1$. C'est à dire, le graphe est une ligne.
3	26	750	$n \leq 1000$	10	$m = n - 1$. C'est à dire, le graphe est un arbre.
4	52	750	$n \leq 300$	30	

On rappelle que votre score sur ce problème est la somme des scores sur chaque sous-tache. Votre score sur une sous-tâche est le score maximal de vos solutions sur cette sous-tâche. Par conséquent, vous pouvez résoudre différentes sous-tâches avec plusieurs soumissions différentes.

Entrée / Sortie

Ce problème n'a ni fichier d'entrée, ni fichier de sortie. À la place, votre solution doit interagir avec les fonctions définies dans le fichier header `"nav.h"`. Les fonctions fournies sont décrites en détail dans la section suivante. **Veuillez *ne rien* écrire sur la sortie standard, ou vous recevrez un score de 0 pour cette soumission.**

Implémentation

Veuillez **ne pas** implémenter de fonction `main`. À la place, veuiller ajouter `#include "nav.h"` et implémenter les deux fonction `init` et `findPrime`, décrites ci-dessous :

```
void init(int subtask, int n, int m, std::vector<int> A, std::vector<int>
B);
```

avec :

- `subtask` est le numéro de la sous-tâche.
- `n` est le nombre de sommets
- `m` est le nombre d'arêtes
- Pour tout $0 \leq j \leq m - 1$, l'arête j relie les sommets `A[j]` et `B[j]`.
- Cette fonction sera appellée une fois au début de l'exécution pour initialiser votre programme.

```
int findPrime();
```

Cette fonction doit retourner un entier entre 0 et $n-1$: l'identifiant du sommet principal.

L'évaluateur (*grader*) appellera cette fonction t fois, une pour chaque test. Votre programme ne connaît pas la valeur de t.

Dans votre implémentation, vous pouvez appeler la fonction suivante.

```
int nav(int u)
```

avec:

- `u` doit être un entier, entre 0 et $n-1$, correspondant à l'identifiant du sommet de votre choix.
- Si u est le sommet principal, la fonction renverra -1. Sinon, la fonction renverra un entier entre 0 et $m-1$, correspondant à une **arête** appartenant à un des plus courts chemins entre u et le sommet principal.

Si `nav` est appelée avec une valeur invalide de u, l'exécution se terminera obtiendra un score de 0. Sur l'interface en ligne, le jugement obtenu sera `Not correct`.

Vous pouvez considérer qu'au cours d'un test, l'évaluateur (*grader*) utilisé ne changera pas l'identité du sommet principal.

Un template de code `nav.cpp` vous est fourni.

Expérimentation

Afin de pouvoir tester votre code sur votre propre machine, veuillez dans un premier temps télécharger les fichiers `nav.h` et `grader.cpp`, et les placer dans le même dossier que votre code. Veuillez noter que l'évaluateur (*grader*) utilisé peut avoir un comportement différent de celui qui vous est fourni.

Compilez votre solution avec :

```
g++ -std=c++11 -O2 -Wall -static nav.cpp grader.cpp -o nav
```

Cette commande crée un exécutable `nav`, que vous pouvez lancer avec `./nav`. Si vous avez des difficultés pour compiler, veuiller nous envoyer un message via la section Communication de l'interface en ligne.

L'évaluateur d'exemple lit le numéro de la sous-tâche, n et m sur la première ligne.

Il lit ensuite m ligne. Sur la i-ème d'entre elles il lira deux entiers `A[i]` et `B[i]`.

Sur la ligne suivante, il lira le nombre de tests t, suivi de t lignes. Sur la j-ème d'entre elles, il lira l'identifiant du sommet principal pour le j-ème test.

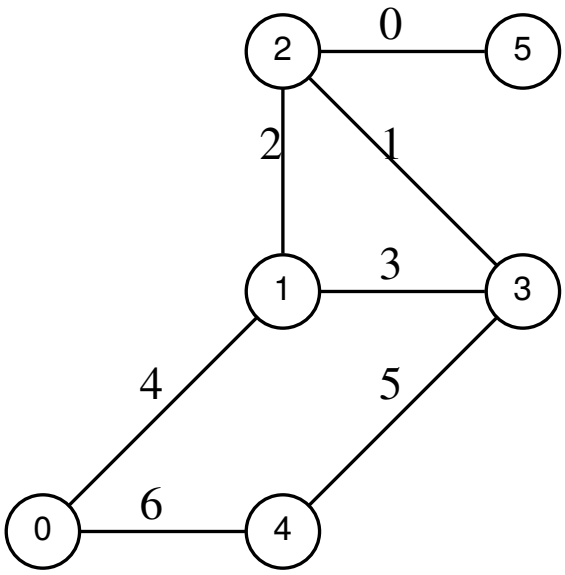
Pour chaque test, l'évaluateur d'exemple appelera `findPrime()` et affichera si votre programme a trouvé avec succès le sommet principal, ainsi que le nombre de requêtes au Nav effectuées.

Exemple d'exécution de l'évaluateur d'exemple

```
4 6 7
2 5
2 3
2 1
1 3
0 1
3 4
```

```
4 0
3
3
5
3
```

L'évaluateur d'exemple commence par lire le graphe, puit interagit avec votre programme.



Une interaction possible est détaillée ci-dessous :

Évaluateur	Étudiant	Description
`init(4, 6, 7, [2, 2, 2, 1, 0, 3, 4], [5, 3, 1, 3, 1, 4, 0])`		L'évaluateur initialise votre programme. Il s'agit d'un exemple de la sous-tâche 4.
`findPrime()`		L'évaluateur commence le premier test. Le sommet principal est le sommet 3.
	`nav(0)`	Votre programme appelle `nav` sur le sommet 0.
`nav` renvoie 6		L'évaluateur choisit le plus court chemin $0 \to 4 \to 3$, et décide de renvoyer l'arête 6 (l'arête entre les sommets 0 et 4).
	`nav(2)`	Votre programme appelle `nav` sur le sommet 2.
`nav` renvoie 1		L'évaluateur choisit le plus court chemin $2 \to 3$, et décide de renvoyer l'arête 1 (l'arête entre les sommets 2 et 3).
	`nav(3)`	Votre programme appelle `nav` sur le sommet 3.
`nav` renvoie -1		C'est le sommet principal.
	Vous renvoyez 3	Votre programme devine que le sommet principal est le sommet 3. Votre réponse est correcte.
`findPrime()`		L'évaluateur commence le deuxième test. Le sommet principal est le sommet 5.
	`nav(4)`	Votre programme appelle `nav` sur le sommet 4.

Évaluateur	Étudiant	Description
`nav` renvoie 1		L'évaluateur choisit le plus court chemin $4 \to 3 \to 2 \to 5$, et décide de renvoyer l'arête 1 (l'arête entre les sommets 2 et 3).
	Vous renvoyez 1	Votre programme devine que le sommet principal est le sommet 1. Votre réponse est incorrecte !
`findPrime()`		L'évaluateur commence le troisième test. Le sommet principal est à nouveau le sommet 3.
	Vous renvoyez 3	Votre programme devine que le sommet principal est le sommet 3 sans faire un seul appel à `nav` !. Votre réponse est correcte.

Dans cet exemple, vous n'avez résolu que deux des trois tests correctement, vous obtenez donc 0 points.