D. Garden Decorations

Nom du problème	Décorations de jardin
Limite de temps	7 secondes
Limite mémoire	1 gigaoctet

Chaque jour, lors de ses trajets entre l'école et la maison, Detje marche le long d'une rue de N maisons, numérotées de 0 à $N-1$. Actuellement, la maison i est habitée par la personne i. Pour se changer les idées, les habitants ont décidé d'échanger leurs maisons entre eux. La personne qui déménagera dans la maison i est la personne a_i (qui habite actuellement la maison a_i).

Chaque maison possède une statue d'oiseau dans son jardin. Les statues peuvent être dans deux états différents: leurs ailes sont soit *ouvertes* (comme si l'oiseau volait), soit *fermées* (comme si l'oiseau marchait au sol). Les habitants ont de très fortes préférences concernant l'état de la statue dans leurs jardins, et refusent de déménager dans leur nouvelle maison jusqu'à ce que la statue dans leur nouveau jardin ressemble exactement à celle de leur ancien jardin. Detje souhaite aider les habitants à changer l'état des statues d'oiseau, afin qu'ils puissent enfin déménager.

Pour cela, chaque fois qu'elle marche dans la rue (soit pour aller à l'école, soit pour rentrer à la maison), elle observe les oiseaux un à un en passant à coté et peut décider de modifier l'état de certaines statues (en ouvrant ou en fermant leurs ailes). Comme elle est très occupée à l'école et à la maison, **elle ne se rappelle pas l'état des oiseaux qu'elle a vus lors de ses derniers voyages**. Heureusement, elle a écrit sur un papier la liste $a_0, a_1, ..., a_{N-1}$, donc elle sait quel résident déménage où.

Aide Detjie à imaginer une stratégie qui déciderait quels oiseaux manipuler afin d'adapter les états des statues aux préférences des habitants. Elle peut marcher le long de la rue au plus 60 fois, mais pour obtenir un meilleur score, elle devrait faire moins de trajets.

Implémentation

Ceci est un problème interactif à exécutions multiples, cela signifie que votre problème sera exécuté plusieurs fois.

A chaque exécution, vous devez lire une première ligne avec deux entiers w et N, le numéro du trajet et le nombre de maisons. Lors de la première exécution de votre program on a $w = 0$, lors

de la seconde $w = 1$, etc. (plus de détails sont donnés ci-dessous)

La seconde ligne de l'entrée contient N entiers $a_0, a_1, ..., a_{N-1}$, ce qui signifie que la personne qui déménagera dans la maison i habite actuellement dans la maison a_i. Les a_is forment une *permutation*: c'est à dire que chaque nombre entre 0 et $N-1$ apparaît exactement une fois dans la liste des a_is. Notez qu'un habitant peut choisir de ne pas déménager; on peut donc avoir $a_i = i$.

Les habitants déménagent au plus une fois. Cela signifie que pour un test donné, la valeur de N et de la liste des a_is sera la même pour toutes les exécutions de votre programme.

Première exécution.

Pour la première exécution de votre programme, $w = 0$. Dans cette exécution, vous devez seulement afficher un unique entier W ($0 \leq W \leq 60$), le nombre de trajets que vous souhaitez que Detje fasse. Votre programme devra alors terminer. Après cela, votre programme sera à nouveau exécuté W fois de plus.

Exécutions suivantes.

Lors de la prochaine exécution de votre programme, $w = 1$, lors de la suivante $w = 2$, et ainsi de suite jusqu'à la dernière exécution où $w = W$.

Après avoir lu w, N et les $a_0, a_1, ..., a_{N-1}$, Detje commence à marcher le long de la rue.

- Si w est impair, Detje marche de sa maison à l'école, et elle passera devant les maisons dans l'ordre : $0, 1, ..., N-1$.

Votre program doit maintenant lire une ligne contenant b_0, qui peut être soit 0 (fermé), soit 1 (ouvert), l'état actuel de la statue en face de la maison 0. Après avoir lu b_0, vous devez affiner une ligne avec soit 0 soit 1, la nouvelle valeur que vous voulez affecter à b_0.

Votre programme doit ensuite lire une ligne contenant b_1, l'état actuel de la statue en face de la maison 1, et afficher la nouvelle valeur de b_1. Et ainsi de suite pour chacune des N maisons. Après que Dekje soit passée à côté de la dernière maison (c'est-à-dire, après que avoir lu et écrit b_{N-1}) **votre programme doit se terminer**.

Notez que votre programme peut seulement lire la prochaine valeur b_{i+1} après que vous ayez écrit la valeur de b_i. Si w est pair, Detjie marche de l'école à sa maison, elle passe le long des maisons dans l'ordre $N-1, N-2, ..., 0$.

Le processus est le même que lorsque w est impair, à la différence que vous commencez par lire et écrire b_{N-1}, puis b_{N-2}, et ainsi de suite jusqu'à b_0.

Quand $w = 1$, les valeurs d'entrée $b_0, b_1, ..., b_{N-1}$ correspondent à l'état initial des statues d'oiseau. Quand $w > 1$, les valeurs d'entrée $b_0, b_1, ..., b_{N-1}$ de votre programme correspondront à ce que les

exécutions précédentes de votre programme les ont définies.

A la fin, après la dernière exécution de votre programme, la valeur de b_i doit être égale à la valeur initiale de b_{a_i} pour tous les i, sinon vous obtiendrez le verdict Wrong Answer (mauvaise réponse).

Détails.

Si la *somme* des temps d'exécution des $W+1$ exécutions distinctes de votre programme excède la limite de temps, votre soumission sera jugée comme Time Limit Exceeded (limite de temps dépassée).

Faites attention à bien mettre à jour (flush) la sortie standard après avoir affiché chaque ligne, faute de quoi votre programme pourrait être jugé comme Time Limit Exceeded. En python, cela se fait automatiquement tant que vous utilisez `input()` pour lire des lignes. En C++, `cout << endl;` met à jour la sortie standard en plus d'écrire un retour à la ligne. Si vous utilisez printf, utilisez `fflush(stdout)` juste après.

Contraintes et Répartition des points

- $2 \leq N \leq 500$.
- Vous pouvez utiliser au plus $W \leq 60$ trajets.

Votre solution sera testée sur un ensemble de sous-tâches, chaque sous-tâche rapportant un certain nombre de points. Chaque sous-tâche contient un ensemble de tests. Pour récupérer les points d'une sous-tâche, vous devez valider tous les tests de cette sous-tâche.

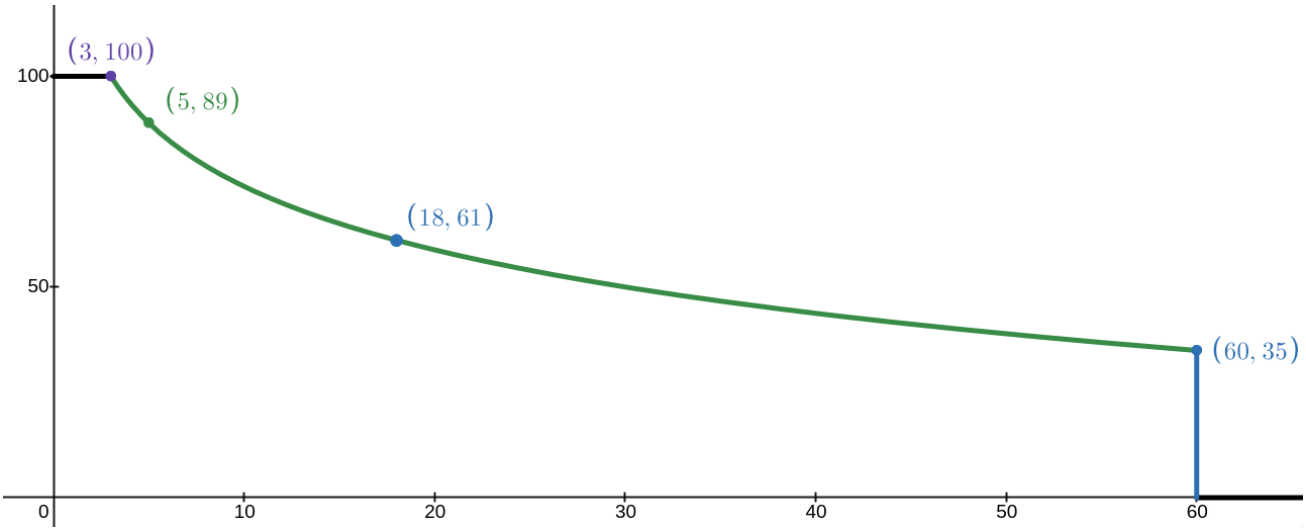
Sous-tâche	Points	Contraintes
1	10	$N = 2$
2	24	$N \leq 15$
3	9	$a_i = N - 1 - i$
4	13	$a_i = (i + 1) \bmod N$
5	13	$a_i = (i - 1) \bmod N$
6	31	Pas de contraintes supplémentaires

Pour chaque sous-tâche que votre progranne résout correctement, vous recevrez un score calculé avec la formule suivante:

$$\text{score} = S_g \cdot \left(1 - \tfrac{1}{2} \log_{10}(\max(W_g, 3)/3)\right),$$

où S_g est le score maximum pour la sous-tâche, et W_g est la valeur maximum de W utilisée pour n'importe lequel des tests appartenant à cette sous-tâche. Votre score pour chaque sous-tâche sera arrondie à l'entier le plus proche.

Le graphe ci-dessous montre le nombre de points que votre programme obtiendrait s'il résout toutes les sous-tâches avec la même valeur de W. En particulier, afin d'obtenir 100 points sur ce problème, vous devez résoudre toutes les sous-tâches avec $W \leq 3$.



Outils de test

Afin de faciliter les tests de votre solution, nous vous fournissons un outil simple que vous pouvez télécharger. Voir "attachments" (pièces jointes) au bas de la page du problème sur Kattis. L'utilisation de cet outil est optionnelle. Notez que le programme officiel d'évaluation de ce problème sur Kattis est différent de l'outil de test.

Pour utiliser cet outil, créez un fichier d'entrée, par exemple “test1.in”, qui devrait commencer avec un nombre N suivi d'une ligne avec N nombres spécifiant la permutation, et une autre ligne avec N bits (0 ou 1) spécifiant les états initiaux des oiseaux. Par exemple:

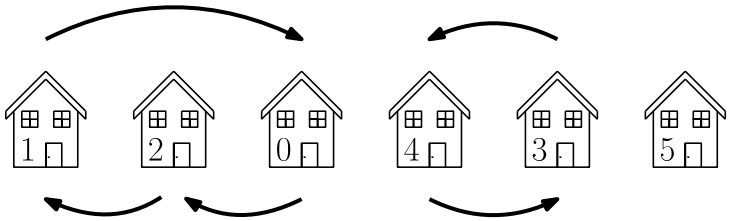
```
6
1 2 0 4 3 5
1 1 0 0 1 0
```

Pour les programmes Python, par exemple `solution.py` (normalement exécuté avec `pypy3 solution.py`): python3 testing_tool.py pypy3 solution.py < sample1.in

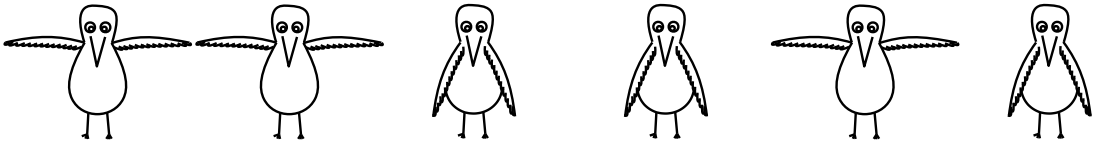
Pour les programmes C++, compilez d'abord (par exemple avec `g++ -g -O2 -std=gnu++20 -static solution.cpp -o solution.out`) puis exécutez: python3 testing_tool.py ./solution.out < sample1.in

Exemple

Dans ce test, on nous donne la permutation suivante de personnes qui souhaitent déménager:

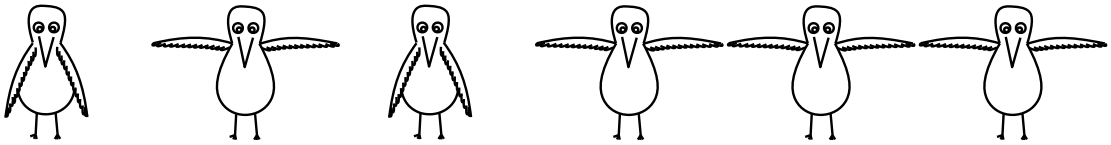


La première fois que le programme d'exemple est exécuté (avec $w = 0$), il affiche $W = 2$, ce qui signifie que Detje marchera deux fois le long de la rue (et que le programme s'exécutera deux fois de plus). Avant le premier trajet, les oiseaux dans les jardins ont les états suivants:



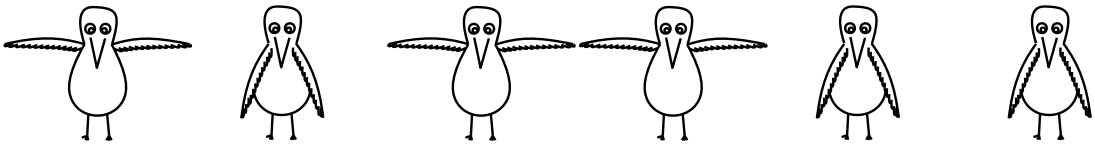
Le programme est ensuite exécuté avec $w = 1$: il s'agit du premier trajet de Detje. Elle marche d'un oiseau à l'autre, en partant de la gauche, et change leurs états si elle le souhaite. Le programme d'exemple doit afficher l'état de l'oiseau i avant de pouvoir connaître l'état de l'oiseau $(i + 1)$.

Après que Detje soit arrivée à l'école, les oiseaux sont dans les états suivants:



Lors de la dernière exécution du programme (avec $w = 2$), Detje retourne à sa maison après l'école. Notez que dans ce cas, elle parcourt les oiseaux de la droite vers la gauche et change leurs états dans l'ordre opposé ! Cela signifie qu'elle doit déterminer l'état de l'oiseau i avant de voir l'oiseau $i - 1$.

Après qu'elle ait fini son trajet, les oiseaux sont maintenant dans les états suivants:



En effet, il s'agit bien de la configuration valide. Par exemple, la statue d'oiseau 3 (c'est à dire, la quatrième en partant de la gauche) est ouverte (maintenant $b_3 = 1$), ce qui est correct comme la personne 4 va y déménager ($a_3 = 4$) et qu'elle avait initialement une statue d'oiseau ouverte (initialement $b_4 = 1$).

Sortie de l'évaluateur	Votre sortie
0 6	
1 2 0 4 3 5	
	2

Sortie de l'évaluateur	Votre sortie
1 6	
1 2 0 4 3 5	
1	
	0
1	
	1
0	
	0
0	
	1
1	
	1
0	
	1

Sortie de l'évaluateur	Votre sortie
2 6	
1 2 0 4 3 5	
1	
	0
1	
	0
1	
	1
0	
	1
1	
	0
0	
	1