D. Jeu de devinette

Nom du problème	Jeu de devinette
Limite de temps	4 secondes
Limite de mémoire	1 gigaoctet

Dans le centre-ville historique de Lund, il existe une rue avec N maisons consécutives, numérotées de 0 à $N-1$. Emma vit dans une de ces maisons et ses amis Anna et Bertil veulent deviner de laquelle il s'agit. Mais au lieu de dire à ses amis là où elle habite, Emma décide de jouer à un jeu avec eux. Avant que le jeu ne commence, Anna et Bertil connaissent seulement le nombre de maisons dans la rue. Anna et Bertil choisissent alors un entier positif K et se mettent d'accord sur une stratégie. Après cela, toute communication est interdite.

Le jeu se joue en deux phases. Dans la première phase, Emma choisi un ordre de visite des maisons, tel que sa maison soit la dernière à être visitée. Elle mène alors Anna de maison en maison en suivant cet ordre, sans informer Anna de l'ordre à l'avance. Pour chaque maison (sauf la maison d'Emma), Anna peut écrire un unique entier entre 1 et K à la craie sur la porte de la maison. Pour la dernière maison visitée, la maison d'Emma, Emma écrit elle-même un entier entre 1 et K sur la porte.

Dans la seconde phase du jeu, Bertil marche le long de la rue de la maison 0 à la maison $N-1$ et lit les nombres écrits sur les portes par Anna et Emma. Il veut maintenant deviner dans quelle maison Emma habite. Il a deux essais pour trouver quelle maison est la bonne, lui et Anna gagnant le jeu s'il devine correctement. Dans le cas contraire, Emma gagne le jeu.

Pouvez-vous construire une stratégie telle que Anna et Bertil soient sûrs de gagner le jeu ? Votre stratégie sera évaluée en fonction de la valeur de K (plus elle est petite, mieux c'est).

Implémentation

Ceci est un problème à multi-exécution, ce qui veut dire que votre programme sera exécuté plusieurs fois. La première fois qu'il est exécuté, il implémentera la stratégie d'Anna. Après cela, il implémentera celle de Bertil.

La première ligne de l'entrée contient deux entiers, P et N, où P est soit 1 soit 2 (première ou deuxième phase), et N est le nombre de maisons. **Sauf pour l'exemple d'entrée (non-utilisé**

pour le score), N **sera toujours égal à** $100\,000$.

L'entrée qui suit dépend de la phase :

Phase 1

Votre programme doit commencer par afficher le nombre K sur une seule ligne $(1 \leq K \leq 1\,000\,000)$. Puis, $N-1$ fois, il doit lire une ligne contenant un indice i $(0 \leq i < N)$, puis afficher une ligne avec un entier A_i $(1 \leq A_i \leq K)$, avec A_i le nombre qu'Anna écrit sur la porte de la maison i. Chaque indice i à l'exception de l'indice de la maison d'Anna apparaîtra une unique fois, dans un ordre arbitraire décidé par l'évaluateur.

Phase 2

Votre programme doit lire une ligne avec N entiers, $A_0, A_1, \ldots, A_{N-1}$, où A_i est le nombre écrit sur la porte de la maison i.

Ensuite, il doit afficher une ligne avec deux entiers, s_1 et s_2 $(0 \leq s_i < N)$, les essais d'indices. s_1 et s_2 ont le droit d'être égaux.

Détails d'implémentation

Notez que lors de l'exécution de votre programme dans la Phase 2, votre programme est redémarré. Cela signifie que vous ne pouvez sauvegarder aucune information dans des variables entre les exécutions.

Après chaque ligne que vous affichez, assurez-vous de synchroniser la sortie standard, sinon votre programme risque d'être jugé comme Time Limit Exceeded. En Python, `print()` synchronise automatiquement. En C++, `cout << endl;` synchronise également en plus d'afficher une nouvelle ligne ; si vous utilisez printf, alors utilisez `fflush(stdout)`.

L'évaluateur de ce programme peut être **adaptif**, ce qui signifie qu'il peut changer son comportement en fonction de la sortie de votre programme pour empêcher les heuristiques de valider les tests. L'évaluateur peut faire une exécution test de la phase 1, analyser votre sortie, et ensuite exécuter à nouveau la phase 1 pour de vrai en utilisant les informations acquises lors de l'exécution précédente.

Votre programme *doit* être déterministe, c'est-à-dire se comporter de la même manière s'il est exécuté plusieurs fois sur la même entrée. Si vous souhaitez utiliser de l'aléatoire dans votre programme, assurez-vous d'utiliser une graine (seed) fixée. Cela peut être fait en donnant une constante codée en dur (hard-coded) à `srand` (en C++) ou `random.seed` (en Python), ou, si vous utilisez les générateurs de nombres aléatoires de C++11, en précisant la graine en construisant le générateur de nombres aléatoire. En particulier, vous ne pouvez pas utiliser `srand(time(NULL))`

en C++. Si l'évaluateur détecte que votre programme n'est pas déterministre, il vous sera donné un verdict Wrong Answer.

Si la *somme* des temps d'exécution des (jusqu'à 3) exécutions séparées de votre programme dépasse la limite de temps, votre soumission sera jugée comme Time Limit Exceeded.

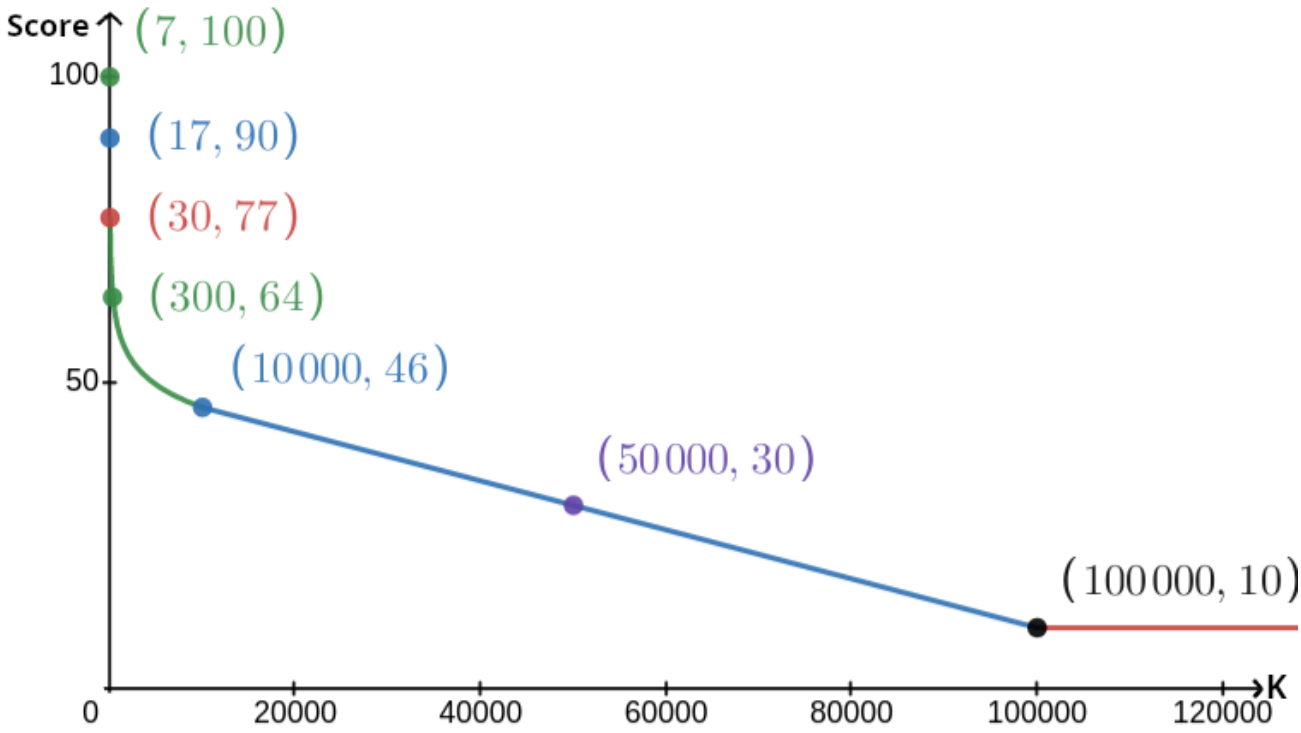
Score

Votre solution sera testée sur un certain nombre de tests. Si votre solution échoue sur *n'importe lequel* de ces tests (par exemple en donnant des réponses fausses (Wrong Answer), s'arrête brutalement (crash) (Run-Time Error), dépasse la limite de temps (Time Limit Exceeded), etc.), vous recevrez 0 points et le verdict approprié.

Si votre programme trouve avec succès l'indice de la maison d'Emma dans *tous* les tests, vous recevrez le verdict Accepted, et un score calculé comme suit. Soit K_{max} la valeur maximale de K que vous utilisez dans tous les tests. Selon K_{max} :

	Score
$K_{max} > 99\,998$	10 points
$10\,000 < K_{max} \leq 99\,998$	$10 + \lfloor 40(1 - K_{max}/10^5) \rfloor$ points
$30 < K_{max} \leq 10\,000$	$46 + \lfloor 31(4 - \log_{10}(K_{max}))/(4 - \log_{10}(30)) \rfloor$ points
$7 < K_{max} \leq 30$	$107 - K_{max}$ points
$K_{max} \leq 7$	100 points

La fonction de score est illustrée dans la figure ci-dessous.



L'exemple de test est ignoré pour le score, et votre solution n'a pas besoin de marcher pour cet exemple de test.

Outil de test

Pour vous aider à tester votre solution, il vous est fourni un outil simple que vous pouvez télécharger. Vous le trouverez dans “attachments” en bas de la page Kattis du problème. L'utilisation de cet outil est optionnelle, et vous avez le droit de le modifier. Notez que l'évaluateur officiel de Kattis est différent de l'outil de test.

Exemple d'utilisation (avec $N = 4$, $s = 2$, où s est le nombre écrit sur la dernière maison visitée):

Pour un programme Python, par exemple nommé `solution.py` (exécuté normalement avec `pypy3 solution.py`):

```
python3 testing_tool.py pypy3 solution.py <<<"4 2"
```

Pour un programme C++, compilez-le (par exemple avec `g++ -g -O2 -std=gnu++17 -static solution.cpp -o solution.out`) et ensuite exécutez :

```
python3 testing_tool.py ./solution.out <<<"4 2"
```

L'outil de test visitera les maisons dans un ordre aléatoire. Pour utiliser un ordre précis, modifiez l'outil de test à l'endroit indiqué “MODIFY HERE”.

Exemple d'interaction

L'exemple de test est ignoré pour le score, et votre solution n'a pas besoin de marcher pour cet exemple de test.

Supposons que $N = 4$ et que Emma vit dans la maison 1. Soit A la liste des nombres écrits sur les maisons. Initialement, $A = [0, 0, 0, 0]$, où 0 signifie qu'aucun nombre n'est écrit sur la maison correspondante.

Dans la première exécution de votre code :

$N = 4$ est donné. Votre solution répond avec $K = 3$.

A_2 est demandé. Votre solution répond avec 3. A est maintenant $[0, 0, 3, 0]$.

A_0 est demandé. Votre solution répond avec 1. A est maintenant $[1, 0, 3, 0]$.

A_3 est demandé. Votre solution répond avec 2. A est maintenant $[1, 0, 3, 2]$.

Enfin, l'évaluateur fixe $A_1 = 2$, donc $A = [1, 2, 3, 2]$ à la fin. Ceci conclut la phase 1.

Dans la phase 2 de votre code, votre solution reçoit la liste `1 2 3 2`.

Elle répond `1 3`.

Comme l'un des essais est l'indice correct de la maison (1), Anna et Bertil gagnent le jeu.

sortie de l'évaluateur	votre sortie
1 4	
	3
2	
	3
0	
	1
3	
	2

sortie de l'évaluateur	votre sortie
2 4	
1 2 3 2	
	1 3