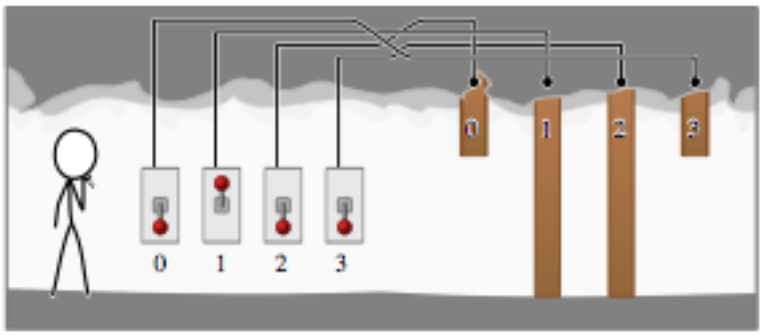
International Olympiad in Informatics 2013
6-13 July 2013
Brisbane, Australia
Day 2 tasks

cave

French — 1.0

En vous rendant au *UQ Centre*, vous vous égarez et tombez sur l'entrée d'une grotte secrète sous l'université. L'entrée est bloquée par un système de sécurité consistant en `N` portes consécutives, placées les unes derrières les autres, et `N` interrupteurs, chacun connecté à une porte distincte.



Les portes sont numérotées de `0` à `N - 1`, la porte 0 étant la plus proche de vous, la porte 1 la suivante, et ainsi de suite. Les interrupteurs sont également numérotés de `0` à `N - 1`, mais vous ne savez pas à quelle porte chacun est connecté.

Les interrupteurs sont tous situés à l'entrée de la grotte. Chacun peut être soit en position *haute* soit en position *basse*. Pour chaque interrupteur, une seule de ces deux positions est la bonne. Si un interrupteur est dans la bonne position alors la porte à laquelle il est connecté est ouverte. S'il est dans la mauvaise position, la porte est fermée. La bonne position peut être différente pour chaque interrupteur et vous ne savez pas a priori quelles positions sont les bonnes.

Vous aimeriez comprendre comment ce système de sécurité fonctionne. Pour cela, vous pouvez mettre chaque interrupteur dans la position qui vous convient, et ensuite venir constater à l'intérieur de la grotte quelle est la première porte fermée. Les portes ne sont pas transparentes, vous ne pouvez donc pas connaître l'état des portes derrière la première que vous voyez fermée.

Vous avez suffisamment de temps pour tester au plus `70 000` combinaisons de positions d'interrupteurs. Votre tâche est de déterminer la bonne position de chaque interrupteur et, pour chaque interrupteur, la porte à laquelle il est connecté.

Implémentation

Vous devez soumettre un fichier qui implémente la fonction `exploreCave()`. Celle-ci peut appeler la fonction `tryCombination()` de l'évaluateur jusqu'à 70 000 fois et doit se terminer par un appel à la fonction `answer()`. Ces fonctions sont décrites ci-dessous.

Fonction de l'évaluateur : `tryCombination()`

C/C++
```
int tryCombination(int S[]);
```

Pascal
```
function tryCombination(var S: array of LongInt) : LongInt;
```

Description

L'évaluateur contient cette fonction. Elle vous permet de tester une combinaison d'interrupteurs et de constater ensuite dans la grotte quelle est la première porte fermée. Si toutes les portes sont ouvertes, la fonction retourne `-1`. Cette fonction s'exécute en `O(N)`, c'est-à-dire que son temps d'exécution est proportionnel à `N` en pire cas.

Cette fonction ne peut être appelée qu'au plus `70 000` fois.

Paramètres

- `S` : un tableau de taille `N` indiquant la position de chacun des interrupteurs. L'élément `S[i]` correspond à l'interrupteur `i`. `0` indique que l'interrupteur est en position haute. `1` indique que l'interrupteur est en position basse.
- *Retourne* : le numéro de la première porte qui est fermée ou `-1` si toutes les portes sont ouvertes.

Fonction de l'évaluateur : `answer()`

C/C++
```
void answer(int S[], int D[]);
```

Pascal
```
procedure answer(var S, D: array of LongInt);
```

Description

Appelez cette fonction lorsque vous avez identifié la combinaison d'interrupteurs qui ouvre toutes les portes ainsi que la porte correspondant à chaque interrupteur.

Paramètres

- `S` : un tableau de taille `N` indiquant la bonne position de chaque interrupteur. Le format est le même que celui de la fonction `tryCombination()` décrite ci-dessus.
- `D` : un tableau de taille `N` indiquant la porte à laquelle chaque interrupteur est connecté. L'élément `D[i]` doit contenir le numéro de la porte à laquelle l'interrupteur `i` est connecté.
- *Retourne* : cette fonction ne retourne rien mais provoquera l'arrêt de votre programme.

Votre fonction : `exploreCave()`

C/C++
```
void exploreCave(int N);
```

Pascal
```
procedure exploreCave(N: longint);
```

Description

Votre soumission doit implémenter cette fonction.

Cette fonction doit utiliser la fonction `tryCombination()` de l'évaluateur afin de déterminer la bonne position de chaque interrupteur ainsi que la porte correspondant à chacun d'eux. Elle doit appeler `answer()` lorsqu'elle a déterminé cette information.

Paramètres

- `N` : le nombre d'interrupteurs et de portes dans la grotte.

Exemple de session

Supposons que les portes et les interrupteurs soient connectés comme représenté sur l'image précédente.

Appel de fonction	Retour	Explication
`tryCombination([1, 0, 1, 1])`	`1`	Cette configuration est celle montrée sur l'image. Les interrupteurs 0, 2 et 3 sont en position basse, tandis que le 1 est en position haute. La fonction retourne `1`, indiquant que la porte 1 est la première porte qui est fermée.
`tryCombination([0, 1, 1, 0])`	`3`	Les portes 0, 1 et 2 sont ouvertes tandis que la porte 3 est fermée.
`tryCombination([1, 1, 1, 0])`	`-1`	Baisser l'interrupteur 0 permet à toutes les portes d'être ouvertes, comme indiqué par la valeur de retour `-1`.
`answer([1, 1, 1, 0], [3, 1, 0, 2])`	*(Le programme se termine)*	Nous supposons que la bonne combinaison est `[1, 1, 1, 0]` et que les interrupteurs 0, 1, 2 et 3 sont respectivement connectés aux portes 3, 1, 0 et 2.

Contraintes

- Limite de temps : 2 secondes
- Limite de mémoire : 32 Mio
- $1 \leq N \leq 5\,000$

Sous-tâches

Sous-tâche	Points	Contraintes d'entrée supplémentaires
1	12	Pour tout `i`, l'interrupteur `i` est connecté à la porte `i`. Il vous suffit de déterminer la bonne combinaison.
2	13	La combinaison correcte sera toujours `[0, 0, 0, …, 0]`. Il vous suffit de déterminer quelle porte correspond à chaque interrupteur.
3	21	$N \leq 100$
4	30	$N \leq 2\,000$
5	24	*(Aucune)*

Expérimentation

L'évaluateur fourni sur votre ordinateur lira l'entrée dans le fichier `cave.in`, qui doit être au format suivant :

- ligne 1 : `N`
- ligne 2 : `S[0] S[1] … S[N - 1]`
- ligne 3 : `D[0] D[1] … D[N - 1]`

Ici `N` est le nombre de portes et d'interrupteurs, `S[i]` est la bonne position de l'interrupteur `i`, et `D[i]` est la porte à laquelle l'interrupteur `i` est connecté.

Par exemple, l'exemple ci-dessus doit être fourni au format suivant :

```
4
1 1 1 0
3 1 0 2
```

Remarques pour chaque langage

C/C++	Vous devez utiliser `#include "cave.h"`.
Pascal	Vous devez utiliser l'unité `unit Cave`, et devez importer les routines de l'évaluateur via `uses GraderHelpLib`. Tous les tableaux sont numérotés à partir de `0` et non `1`.

Prenez exemple sur les modèles de solution fournis sur votre machine.