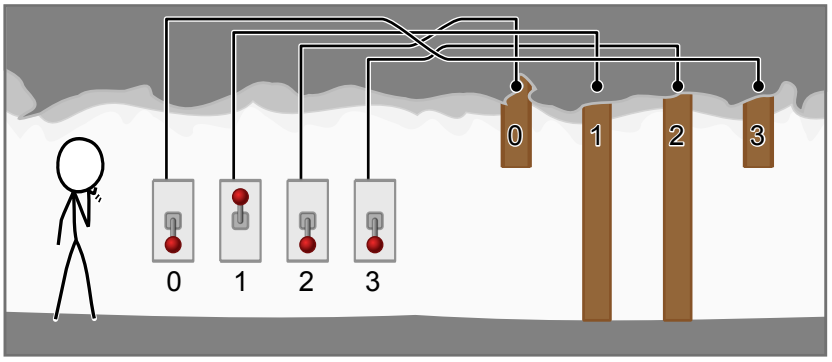
International Olympiad in Informatics 2013
6-13 July 2013
Brisbane, Australia
Day 2 tasks

cave

French — 1.0

Alors que vous vous êtes égaré lors de votre trajet de vos dortoirs aux UQ Centre, vous découvrez 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)` selon leur position, la porte 0 étant la plus proche de vous. Les interrupteurs sont également numérotés de `0` à `(N-1)`, cependant vous ne savez pas quel interrupteur est connecté à quelle porte.

Les interrupteurs sont tous situés à l'entrée de la grotte. Chacun peut être soit vers le haut ou vers le bas (leur deux uniques positions possibles). Un interrupteur est dans une bonne position s'il ouvre la porte à laquelle il est connecté. Dans sa mauvaise position, la porte reste 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'entrée 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 maximum `70 000` combinaisons de positions d'interrupteurs. Votre tâche est de déterminer la bonne position de chaque interrupteur et la correspondance entre chaque interrupteur et la porte à laquelle il correspond.

Implémentation

Vous devez soumettre un fichier qui implémente la procédure `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 procédure `answer()`. Ces modules sont décrits 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 fournira (donnera) cette fonction. Elle vous permet de tester une combinaison d'interrupteurs, et ensuite de constater dans la grotte quelle est la première porte fermée. Si toutes les portes sont ouvertes, la fonction renvoie `-1`. Cette fonction s'exécute en `O(N)` fois, c'est-à-dire le temps d'exécution est au pire des cas proportionnel à `N`.

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

Paramètres

- `S`: Un tableau de taille `N` indiquant la position de chacun des interrupteurs. La valeur `S[i]` correspond à l'interrupteur `i`. Une valeur de `0` indique que l'interrupteur est vers le haut tandis qu'une valeur de `1` indique que l'interrupteur est vers le bas.
- *Retourne*: Le numéro de la première porte qui est fermée, ou `-1` si toutes les portes sont ouvertes.

Procédure de l'évaluateur: `answer()`

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

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

Description

Appelez cette procédure 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()` déja décrite.
- `D`: Un tableau de taille `N`, indiquant la porte à laquelle chaque interrupteur est connecté. La valeur `D[i]` doit contenir le numéro de la porte à laquelle l'interrupteur `i` est connecté.
- *Retourne*: Cette procédure ne retourne rien, mais provoquera l'arrêt du programme.

Votre procédure : `exploreCave()`

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

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

Description

Votre soumission doit implémenter cette procédure.

Cette procédure doit utiliser le module `tryCombination()` de l'évaluateur afin de déterminer la position correcte de chaque interrupteur ainsi que la porte correspondant à chacun. Elle doit appeler `answer()` lorsque elle a déterminé cette information.

Paramètres

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

Exemple d'interactions

Supposons que les portes et les interrupteurs sont connectés comme suit :

Appel de fonction	Retour	Description
`tryCombination([1, 0, 1, 1])`	`1`	Cette configuration est celle montrée sur l'image. Les interrupteurs 0, 2 et 3 sont en bas, tandis que le 1 est en haut. La fonction retourne `1`, indiquant que la porte 1 est la première porte (de gauche à droite) 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`	Le changement de position de l'interrupteur 0 vers le bas permet à ce que toutes les portes soient ouvertes, comme indiqué par la valeur de retour de `-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]`, est 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 (Mille Octets)
- `1 <= N <= 5,000`

Sous-tâches

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

Implémentation

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.

L'exemple donné 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 définir l'unité `unit Cave`, et vous devez importer les routines (modules) de l'évaluateur via `uses GraderHelpLib`. Tous les tableaux sont numérotés à partir de `0` et non de `1`.

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