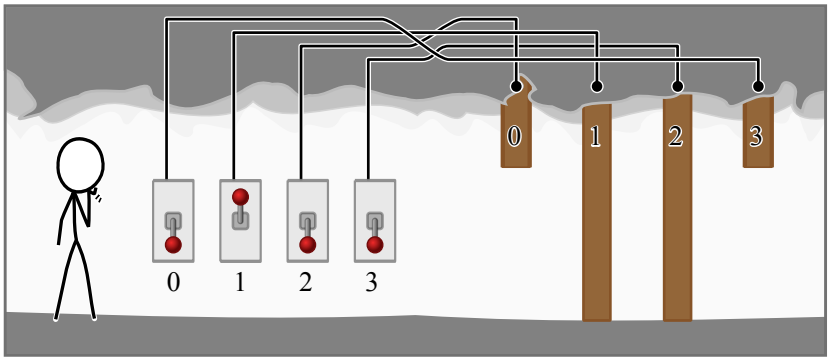
International Olympiad in Informatics 2013
6-13 July 2013
Brisbane, Australia
Day 2 tasks

cave

Spanish — 1.0

Mientras te encuentras perdido en tu larga camita hacia el Centro UQ, te has encontrado con la entrada a un sistema secreto de cuevas debajo de la universidad. La entrada está bloqueada por un sistema de seguridad que consta de `N` puertas consecutivas, cada puerta detrás de la anterior, y con `N` interruptores, cada uno conectado a una puerta distinta.



Las puertas están numeradas de la forma `0, 1, …, (N - 1)` en orden, siendo la puerta 0 la más cercana a ti. Los interruptores también están numerados de la forma `0, 1, …, (N - 1)`, sin embargo no conoces a que puerta está conectado cada interruptor.

Los interruptores se encuentran en la entrada de la cueva, cada interruptor puede estar hacia arriba o hacia abajo. Solo una de estas posiciones es la correcta para cada interruptor. Si un interruptor se encuentra en la posición correcta, la puerta a la que está conectado se abrirá, de no ser así la puerta a la que está conectado se cerrará. La posición correcta puede variar para diferentes interruptores, y tú no sabes cuales posiciones son las correctas.

Te gustaría entender este sistema de seguridad. Para hacer esto, puedes acomodar los interruptores en cualquier combinación, y después caminar a través de la cueva para ver cuál es la primera puerta que se encuentra cerrada. Las puertas no son transparentes, es decir, cuando te encuentres con la primera puerta cerrada no podrás ver ninguna de las puertas que se encuentren detrás de ella.

Tienes tiempo suficiente para probar `70,000` combinaciones de interruptores, y no más. Tu tarea es determinar la posición correcta de los interruptores y a que puerta está conectado cada interruptor.

Implementación

Debes enviar un archivo que implemente el procedimiento `exploreCave()`. Este podrá llamar a la función `tryCombination()` hasta 70,000 veces, y deberá terminar llamando al procedimiento `answer()`. Estos procedimientos y funciones son descritos a continuación.

Función del evaluador `tryCombination()`

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

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

Descripción

El evaluador proveerá esta función, la cual te permite probar una combinación de interruptores, y después entrar en la cueva para determinar la primera puerta cerrada. Si todas las puertas se encuentran abiertas, entonces la función regresará `-1`. Esta función corre en tiempo `O(N)`, es decir, el tiempo de ejecución en el peor caso es proporcional a `N`.

Esta función podrá ser llamada a lo más `70,000` veces.

Parámetros

- `S`: Un arreglo de longitud `N`, indicando la posición de cada interruptor. El elemento `S[i]` corresponde al interruptor `i`. Un valor de `0` indica que el interruptor se encuentra arriba, un valor de `1` indica que el interruptor se encuentra abajo.
- *Returns*: El número de la primera puerta que se encuentra cerrada, o `-1` si todas las puertas están abiertas.

Procedimiento del evaluador: `answer()`

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

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

Descripción

Llama a este procedimiento cuando hayas identificado la combinación de interruptores para abrir todas las puertas, y la puerta a la cual está conectado cada interruptor.

Parámetros

- `S`: Un arreglo de longitud `N`, indicando la posición correcta de cada interruptor. El formato cuadra con el usado para la función `tryCombination()` descrita arriba.
- `D`: Un arreglo de longitud `N`, indicando la puerta a la que esta conectado cada interruptor. Específicamente, el elemento `D[i]` deberá contener el número de la puerta conectada al interruptor `i`.
- *Returns*: Este procedimiento no regresa datos, pero hará que el programa termine.

Tu procedimiento: `exploreCave()`

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

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

Descripción

Tu envío debe implementar este procedimiento.

Este procedimiento debe usar la rutina del evaluador `tryCombination()` para determinar la posición correcta para cada interruptor y la puerta a la que cada interruptor esta conectado, una vez que determines esto, deberás llamar al procedimiento `answer()`

Parámetros

- `N`: El número de interruptores y puertas en la cueva.

Sesión de ejemplo

Supón que las puertas e interruptores están acomodados como en la figura de arriba:

Llamado a función	Returns	Explicación
`tryCombination([1, 0, 1, 1])`	`1`	Esto corresponde a la figura. Los interruptores 0, 2 y 3 están hacia abajo, mientras el interruptor 1 está hacia arriba. La función regresa `1`, indicando que la puerta 1 es la primera puerta desde la izquierda que se encuentra cerrada.
`tryCombination([0, 1, 1, 0])`	`3`	Las puertas 0, 1 y 2 están abiertas, mientras la puerta 3 está cerrada.
`tryCombination([1, 1, 1, 0])`	`-1`	Mover el interruptor 0 hacia abajo causa que todas las puertas se abran, indicado por el valor de retorno `1`.
`answer([1, 1, 1, 0], [3, 1, 0, 2])`	*(Program exits)*	Sabemos que la combinación correcta es `[1, 1, 1, 0]`, y que los interruptores 0, 1, 2 y 3 conectan las puertas 3, 1, 0 y 2 respectivamente.

Restricciones

- Tiempo límite: 2 segundos
- Límite de Memoria: 32 MiB
- $1 \le N \le 5{,}000$

Subtareas

Subtarea	Puntos	Restricciones Adicionales a la Entrada
1	12	Para cada `i`, el interruptor `i` está conectado a la puerta `i`. Tu tarea consiste simplemente en encontrar la combinación correcta.
2	13	La combinación correcta siempre será `[0, 0, 0, …, 0]`. Tu tarea consiste simplemente en determinar con que puerta se conecta cada interruptor.
3	21	$N \le 100$
4	30	$N \le 2{,}000$
5	24	*(Ninguna)*

Experimentación

El evaluador de ejemplo de tu computadora leerá la entrada del archivo `cave.in`, el cual deberá seguir el siguiente formato:

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

Aquí `N` es el número de puertas e interruptores, `S[i]` es la posición correcta para el interruptor `i`, y `D[i]` es la puerta a la que el interruptor `i` está conectado.

Por ejemplo, el caso de arriba debiera ser provisto en el siguiente formato:

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

Notas de lenguaje

C/C++	Debes incluir `#include "cave.h"`.
Pascal	Debes definir la unidad `unit Cave`, y debes importar las rutinas del evaluador a traves de `uses GraderHelpLib`. Todos los arreglos estan numerados iniciando con `0` (no `1`).

Observa los templates de las soluciones en tu computadora para ver ejemplos.