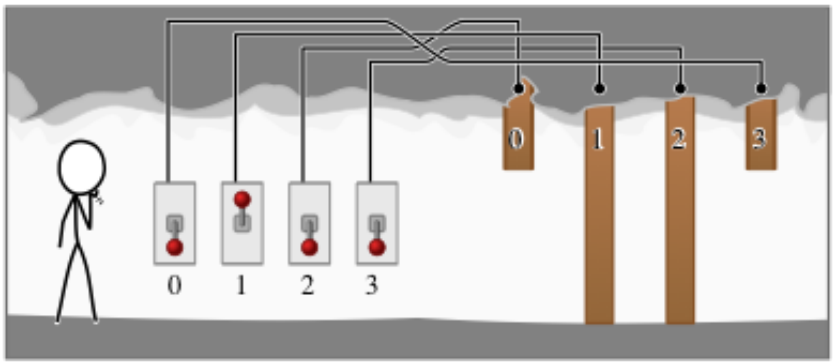
International Olympiad in Informatics 2013
6-13 July 2013
Brisbane, Australia
Day 2 tasks

cave

Spanicsh — 1.0

Mientras caminabas perdido entre tu habitación y el UQ Centre, has encontrado la entrada secreta a un sistema de cavernas que corren bajo la universidad. La entrada está bloqueada por un sistema de seguridad que consiste en `N` puertas consecutivas, una detrás de otra; y `N` interruptores, cada uno conectado a una puerta diferente.



Las puertas están numeradas `0, 1, ..., (N - 1)` de forma ordenada, con la puerta numero 0 siendo las más cercana a ti. Los interruptores están también numerados `0, 1, ..., (N - 1)`, pero tu no sabes que interruptor está conectado a que puerta.

Todos los interruptores se encuentran en la entrada de la caverna. Cada interruptor puede estar en una de dos posiciones: *arriba* o *abajo*. Sólo una de esas posiciones es la correcta para cada interruptor. Si un interruptor está en la posición correcta entonces la puerta a la que está conectado se abrirá, y si está en la posición incorrecta se cerrará. La posición correcta puede ser diferente para cada interruptor, y tú no sabes cuáles posiciones son las correctas.

A ti te gustaría entender este sistema de seguridad. Para ello, puedes poner los interruptores en cualquier combinación y luego caminar en la caverna para ver cuál es la primera puerta cerrada. Las puertas no son transparentes: una vez que encuentras una puerta cerrada no puedes ver ninguna de las puertas que se encuentran detrás de ella.

Tienes tiempo suficiente para probar `70.000` combinaciones de interruptores, pero no más. Tu tarea es determinar la posición correcta para cada interruptor, así como también a qué puerta está conectado.

Implementación

Debes enviar un archivo que implemente la siguiente función `exploreCave()`. Puedes llamar a la función `tryCombination()` a lo más 70.000 veces, y debes terminar llamando a la función `answer()`. Estas funciones se describen a continuación.

Función Provista: `tryCombination()`

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

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

Descripción

Esta función será provista. Ella te permite probar una combinación de interruptores, y luego entrar en la caverna para determinar cuál es la primera puerta cerrada. Si todas las puertas están abiertas, está función retorna `-1`. Esta función se ejecuta en tiempo `O(N)`; es decir, el tiempo de ejecución es en el peor caso proporcional a `N`.

Esta función puede ser llamada a lo mas `70.000` veces.

Parámetros

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

Función Provista: `answer()`

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

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

Descripción

Llama a esta función cuando hayas identificado la combinación de interruptores para abrir todas las puertas, y la puerta que está conectada a cada interruptor.

Parámetros

- `S`: Un arreglo de largo `N`, indicando la posición correcta de cada interruptor. El formato es el mismo que el usado en función `tryCombination()` que se describe anteriormente.
- `D`: Un arreglo de largo `N`, indicando la puerta a la que cada interruptor está conectado. Específicamente, el elemento `D[i]` debe contener el número de la puerta a la que el interruptor `i` está conectado.
- *Retorna*: Este procedimiento no retorna, si no que hace terminar el programa.

Tú Función: `exploreCave()`

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

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

Descripción

Tu envío debe implementar está función.

Esta función debe utilizar la función provista `tryCombination()` para determinar la posición correcta de cada interruptor y la puerta a la que está conectada, y debe llamar a `answer()` una vez que esta información ha sido determinada.

Parámetros

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

Sesión de Ejemplo

Supón que las puertas e interruptores están dispuestos como se muestra en la figura de más arriba:

Llamada a Función	Retorna	Explicación
`tryCombination([1, 0, 1, 1])`	`1`	Este corresponde a la figura. Interruptores 0, 2 y 3 están abajo, mientras interruptor 1 está arriba. La función retorna `1`, indicando que la puerta 1 es la primera puerta desde la izquierda que está cerrada.
`tryCombination([0, 1, 1, 0])`	`3`	Puertas 0, 1 y 2 están abiertas, mientras puerta 3 está cerrada.
`tryCombination([1, 1, 1, 0])`	`-1`	Mover el interruptor 0 hacia abajo hace que todas las puertas queden abiertas, indicado por el valor `-1` retornado.
`answer([1, 1, 1, 0], [3, 1, 0, 2])`	*(Programa termina)*	Adivinamos que la combinación correcta es `[1, 1, 1, 0]`, y que los interruptores 0, 1, 2 y 3 se conectan a las puertas 3, 1, 0 y 2 respectivamente.

Restricciones

- Time limit: 2 segundos
- Memory limit: 32 MiB
- $1 \le N \le 5{,}000$

Subtareas

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

Experimentación

El evaluador de ejemplo (sample grader) lee el input desde el archivo `cave.in`, que debe estar en el siguiente formato:

- línea 1: `N`
- línea 2: `S[0] S[1] … S[N - 1]`
- línea 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 está conectado el interruptor `i`.

Asi, el primer ejemplo de arriba debe ser entregado en el siguiente formato:

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

Notas del Lenguaje

C/C++	Debes incluir la línea `#include "cave.h"`.
Pascal	Debes definir la `unit Cave`, y debes también importar la rutinas de evaluación usando `uses GraderHelpLib`. Todos los arreglos están enumerados empezando por `0` (no `1`).

Mira las plantillas de solución que se encuentran en tu máquina.