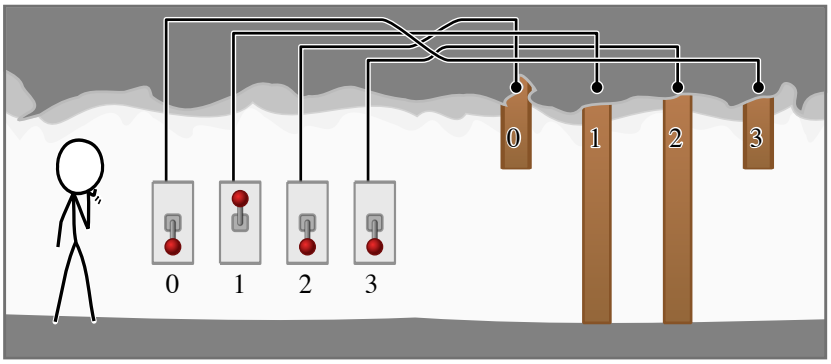
International Olympiad in Informatics 2013
6-13 July 2013
Brisbane, Australia
Day 2 tasks

cave

Deutsch — 1.0

Auf dem Weg vom College zum UQ Centre hast du dich verlaufen und bist auf den Eingang zu einer geheimen Höhle gestoßen. Der Eingang ist durch ein Sicherheitssystem aus `N` Toren versperrt, die hintereinander stehen. Außerdem gibt es `N` Schalter. Jeder Schalter ist mit einem anderen Tor verbunden.



Die Tore sind nacheinander mit `0, 1, ..., (N - 1)` nummeriert, das erste Tor mit `0`. Die Schalter sind ebenfalls mit `0, 1, ..., (N - 1)` nummeriert. Du weißt aber nicht, welcher Schalter mit welchem Tor verbunden ist.

Die Schalter befinden sich alle am Eingang zur Höhle. Für jeden Schalter gibt es zwei Stellungen, nämlich *oben* und *unten*. Für jeden Schalter ist nur eine Stellung richtig. Wenn ein Schalter in der *richtigen* Stellung ist, öffnet sich das mit dem Schalter verbundene Tor. Ist er in der *falschen* Stellung, schließt sich das mit dem Schalter verbundene Tor. Unterschiedliche Schalter können unterschiedliche richtige Stellungen haben. Leider weißt du nicht, welche Stellung jeweils richtig ist.

Du möchtest die Funktion des Sicherheitssystems erforschen. Dazu kannst du jeden Schalter in eine beliebige Stellung bringen und dann für diese Stellungskombination das erste noch geschlossene Tor bestimmen. Leider sind die Tore nicht durchsichtig, sodass du nicht hinter das erste geschlossene Tor sehen kannst.

Deine Zeit ist knapp und deshalb kannst du höchstens `70 000` Stellungskombinationen ausprobieren. Das muss genügen, um für jeden Schalter die richtige Stellung und das damit verbundene Tor herauszufinden.

Implementierung

Du sollst eine Datei einsenden, welche die Prozedur `exploreCave()` implementiert. Diese Prozedur darf die Graderfunktion `tryCombination()` höchstens `70.000` Mal aufrufen und muss am Ende die Graderprozedur `answer()` aufrufen. Diese Routinen werden im Folgenden beschrieben.

Graderfunktion: `tryCombination()`

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

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

Beschreibung

Diese Funktion wird vom Grader zur Verfügung gestellt. Du kannst sie aufrufen, um eine Stellungskombination auszuprobieren und für diese Kombination das erste geschlossene Tor zu bestimmen. Sind alle Tore offen, gibt die Funktion `-1` zurück. Die Laufzeit dieser Funktion liegt in `O(N)`, ist also höchstens proportional zu `N`.

Du darfst diese Funktion höchstens `70 000` Mal aufrufen.

Parameter

- `S`: Ein Array der Länge `N`, das die gewünschten Schalterstellungen angibt. Das Element `S[i]` gibt die Stellung von Schalter `i` an. Der Wert `0` steht für die Stellung *oben*, der Wert `1` für die Stellung *unten*.
- *Rückgabe*: Die Nummer des ersten geschlossenen Tores bzw. `-1`, falls alle Tore offen sind.

Grader-Prozedur: `answer()`

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

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

Beschreibung

Ruf diese Prozedur auf, sobald du für jeden Schalter die richtige Stellung und das damit verbundene Tor gefunden hast.

Parameter

- `S`: Ein Array der Länge `N`, welches die richtige Stellung für jeden Schalter angeben soll. Das Format ist identisch mit dem der oben beschriebenen Funktion `tryCombination()`.
- `D`: Ein Array der Länge `N`, das für jeden Schalter angibt, mit welchem Tor dieser verbunden ist. D.h. Element `D[i]` soll die Nummer jenes Tores enthalten, welches mit dem Schalter `i` verbunden ist.
- *Rückgabe*: Diese Prozedur kehrt nicht zurück, das Programm wird beendet.

Deine Prozedur: `exploreCave()`

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

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

Beschreibung

Deine Abgabe muss diese Prozedur implementieren.

Diese Prozedur soll die Grader-Routine `tryCombination()` verwenden, um die richtigen Schalterstellungen zu ermitteln und herauszufinden, welche Schalter mit welchen Toren verbunden sind. Danach muss sie `answer()` mit den ermittelten Ergebnissen aufrufen.

Parameter

- `N`: Die Anzahl der Schalter und Tore in der Höhle.

Beispiel-Session

Nimm an, dass die Schalter und Tore wie in der obigen Abbildung aufgebaut sind.

Funktionsaufruf	Rückgabewert	Erklärung
`tryCombination([1, 0, 1, 1])`	`1`	Dieser Fall entspricht der Abbildung. Schalter 0, 2 und 3 sind unten, Schalter 1 ist oben. Die Funktion gibt `1` zurück. Das bedeutet: Tor 1 ist das erste Tor, das geschlossen ist.
`tryCombination([0, 1, 1, 0])`	`3`	Tore 0, 1 und 2 sind offen, Tor 3 ist geschlossen.
`tryCombination([1, 1, 1, 0])`	`-1`	Ist auch Schalter 0 unten, dann sind alle Tore offen, was durch den Rückgabewert `-1` angezeigt wird.
`answer([1, 1, 1, 0], [3, 1, 0, 2])`	*(Programm wird beendet)*	Wir vermuten, dass die richtigen Schalterstellungen `[1, 1, 1, 0]` sind, und dass die Schalter 0, 1, 2 und 3 mit den entsprechenden Toren 3, 1, 0 und 2 verbunden sind.

Beschränkungen

- Zeitlimit: 2 Sekunden
- Speicherlimit: 32 MiB
- $1 \le N \le 5000$

Teilaufgaben

Teilaufgabe	Punkte	Zusätzliche Beschränkungen
1	12	Für jedes `i` ist Schalter `i` mit Tor `i` verbunden. Du sollst nur noch die richtigen Schalterstellungen bestimmen.
2	13	Die richtigen Schalterstellungen sind `[0, 0, 0, …, 0]`. Du sollst nur noch feststellen, welche Schalter mit welchen Toren verbunden sind.
3	21	$N \le 100$
4	30	$N \le 2000$
5	24	*(Keine)*

Testen

Der Beispielgrader auf deinem Computer liest die Eingabe aus der Datei `cave.in`, die wie folgt aufgebaut ist:

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

`N` ist die Anzahl der Tore und Schalter, `S[i]` ist die korrekte Schalterstellung für Schalter `i`, und `D[i]` ist das Tor, welches mit Schalter `i` verbunden ist.

Das gegebene Beispiel müsste folgendes Format besitzen:

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

Sprachspezifische Bemerkungen

C/C++	Du musst `#include "cave.h"` verwenden.
Pascal	Du musst `unit Cave` definieren und du musst auch die Grader-Routinen mit `uses GraderHelpLib` importieren. Alle Arrays sind mit `0` beginnend indiziert (nicht `1`).

Schau dir dazu die Beispieldateien auf deinem Computer an.