# Fork Bomb

Matei a aflat recent de funcția **fork** la un curs de sisteme de operare. Pentru a termina laboratorul mai repede, el dorește acum să creeze un **fork bomb**, un program care creează procese prin funcția **fork** cu scopul de a crash-ui calculatorul.
În mod foarte simplificat, executarea instrucțiunii `fork()` de către un proces $P$ funcționează după cum urmează. Mai întâi, se creează un nou proces "copil" $Q$ căruia i se atribuie $P$ ca și proces "părinte". Mai apoi, $P$ și $Q$ își continuă execuția în paralel din dreptul instrucțiunii `fork()` aferente. Pentru a putea diferenția între procesul părinte și cel copil, valoarea returnată de funcția `fork()` este 1 dacă procesul aferent este procesul copil și 0 dacă este vorba de procesul părinte.
Din fericire, Matei lucrează cu un limbaj foarte simplu, care are doar două instrucțiuni posibile:

- Instrucțiunea `fork()`, care, așa cum îi indică numele, fork-uiește procesul în cadrul acelei instrucțiuni.
- Instrucțiunea `IF fork() THEN [...] ELSE [...] ENDIF`, care permite execuția codului în mod condiționat. Cele două blocuri marcate prin puncte de suspensie pot conține 0 sau mai multe instrucțiuni, care se vor executa în ordine.

Un exemplu de folosire al programului este:

```
IF fork() THEN
ELSE
    fork()
    IF fork() THEN
        fork()
    ELSE
    ENDIF
ENDIF
```

Execuția acestui program se realizează cum urmează:

Procesul principal $P_0$ creează un proces copil $P_1$ care va intra pe ramura `THEN` a instrucțiunii `IF` și va termina execuția. Procesul părinte $P_0$ însă va intra pe ramura `ELSE` și va crea un nou proces copil $P_2$. Ambele procese $P_0$ și $P_2$ vor executa cea de-a doua instrucțiune condițională `IF` și vor crea fiecare câte două procese copil $P_3$ și $P_4$, care vor crea procesele copil $P_5$ și $P_6$ respectiv, după care își vor termina execuția. Procesele $P_0$ și $P_2$ vor intra pe ramura `ELSE` și își vor termina execuția.

Matei a observat ca in laboratorul unde se află, calculatoarele crash-uiesc dacă și numai dacă există un *arbore al proceselor* specific. Altfel spus, arborele generat de procesele fork-uite trebuie să aibă o structură specifică.

*Notă: Pentru mai multe detalii, dacă sunteți pe un calculator care rulează Linux, puteți rula într-un terminal comanda* ***man fork****.*

## Date de Intrare

Pe prima linie se dă numărul $n$ de procese din arborele specific.
Pe următoarele $n-1$ linii se dau câte două numere $a_i$ și $b_i$, reprezentând o relație de părinte-copil între $a_i$ și $b_i$ ($0 \leq a_i < b_i < n$). Procesul identificat prin nodul 0 reprezintă procesul principal. Se garantează că datele de intrare descriu un arbore valid.

## Date de Ieșire

Se afișează un număr arbitrar de linii, care să conțină un program corect care să genereze arborele cerut. Mai exact, pentru programul afișat, trebuie să existe o etichetare a celor $n$ procese cu numere de la 0 la $n-1$ astfel încât procesul principal să aibă eticheta 0, iar procesele cu etichetele $a_i$ și $b_i$ să fie în relație tată-fiu, pentru toate perechile $(a_i, b_i)$ din datele de intrare.

Programul poate conține una sau mai multe instrucțiuni și poate fi formatat prin oricâte spații sau caractere "newline". Dacă există mai multe soluții posibile, se va aceepta oricare dintre ele.

## Constrângeri

- $2 \leq n \leq 100$.
- Programul generat nu poate efectua mai mult de $10^5$ instrucțiuni (în total, pentru toate procesele generate)

## Subtask-uri

1. (25 de puncte) Arborele care trebuie generat este un lanț.
2. (25 de puncte) Se garantează existența unei soluții care să nu folosească instrucțiunea `IF` (dar, in mod evident, se acceptă și soluții care folosesc instrucțiunea `IF`).
3. (50 de puncte) Nicio constrângere suplimentară.

## Exemplu

| Input Standard (*cin*) | Output Standard (*cout*) |
|---|---|
| 7<br>0 1<br>0 2<br>2 3<br>0 4<br>3 5<br>4 6 | IF fork() THEN<br>ELSE<br>  fork()<br>  IF fork() THEN<br>  fork()<br>  ELSE<br>  ENDIF<br>ENDIF |
| 4<br>0 1<br>0 3<br>1 2 | fork()<br>fork() |