



Feladat: Speedrun

C++ header `speedrun.h`

Marcel a „Tree Souls III” nevű játék „speedrunning”¹ világcúcsát próbálja megdönteni. Sajnos a legújabb stratégiákat és húzásokat sehogy sem tudja megtanulni, így nem tisztességes módon szeretne előnyt szerezni.

A játék „Teljes fa%” kategóriájában játszik, ami – ahogy a játék nevéből következik – egy fán játszódik. A fa egy N csúcsú körmentes gráf, $N - 1$ éllel, a csúcsokat 1-től N -ig sorszámozzuk.

A játék a következőképpen játszódik: A játékos karaktere a fa egy tetszőleges csúcsából indul, és minden lépésben választhat egy x egész számot. Amennyiben a jelenlegi csúcs és az x csúcs között van él, akkor a játékos az x csúcsba lép. Ha nincs ilyen él, akkor semmi sem történik, a játékos helyben marad. A speedrun akkor érvényes, ha Marcel minden csúcsot legalább egyszer meglátogatott.

És akkor a csalás: mivel Marcel nem ismeri az él-teleportálás-hibát, ezért a játék kódjában átírta a véletlenszám generátort, így előre ismerni fogja a játéktáblát. Viszont ha egyszerűen csak megtanulja azt, akkor túl egyértelmű lesz a csalás, és még az is lehet hogy örökre kitiltják a speedrunning közösségből. Ezért inkább átírta a kódot úgy, hogy minden csúcsban elhelyezett egy l hosszú bináris (0-kat és 1-eseket tartalmazó) tömböt segítségképpen. (Az l hossz azonos az összes csúcsra.) Amikor egy csúcsban áll, Marcel látja az adott csúcsához tartozó segítséget.

Kommunikációs protokoll

Ez egy kétlépcsős interaktív feladat!

A programodat kétszer fogjuk futtatni: először *segítség-generáló*, majd *speedrunning* módban.

A következő függvényeket kell megírnod (és *main* függvényt nem):

```
void assignHints(int subtask, int N, int A[], int B[]);  
void speedrun(int subtask, int N, int start);
```

Ezekben a következő segédfüggvényeket használhatod:

```
void setHintLen(int l);  
void setHint(int i, int j, bool b);  
int getLength();  
bool getHint(int j);  
bool goTo(int x);
```

Segítség-generáló fázis: Ez a program első futása. Itt a bizottság által írt kód az `assignHints` függvényedet hívja meg, pontosan egyszer. A `subtask` paraméter tartalmazza a részfeladat számát, N a csúcsok száma és az A és B tömbök a gráf éleit kódolják: minden i -re ($i = 1, \dots, N - 1$) az $A[i]$ és $B[i]$ csúcsokat egy él köti össze. Ebben a függvényben először a `setHintLen` függvényt hívd meg (pontosan egyszer), a paraméter a segéd tömbök általad választott l hosszát adja meg. Ezután pedig akárhányszor meghívhatod a `setHint(i, j, b)` függvényt. Ez az i -edik csúcsban elhelyezett segítség j -edik bitjét b értékre állítja. A biteket 1-től l -ig számozzuk. A segéd tömbök elemei eredetileg 0-kat tartalmaznak. Ebben a futásban nem hívhatod meg a `getLength`, `getHint` és `goTo` függvényeket. Az `assignHints` függvényből a programod térjen vissza, amint végzett a segéd tömbök beállításával.

Speedrunning fázis: Ez a program második futása. A bizottság által írt kód a `speedrun(subtask, N, start)` függvényedet hívja meg, pontosan egyszer. A `start` paraméter a speedrun induló csúcsát tartalmazza, N a csúcsok száma, míg a `subtask` paraméter tartalmazza a részfeladat számát. Ezután meghívhatod a `getLength()` függvényt, ami a segéd tömbök l hosszát adja vissza (amit a programod az

¹Speedrunning: egy játék minél rövidebb idő alatti végigjátszása.



első futás során választott), a `getHint(j)` függvényt, ami a jelenlegi csúcsban található segítség j -edik bitjét adja vissza, illetve a `goTo(x)` függvényt. Amennyiben a `goTo` függvénynek megadott x szám egy szomszédos csúcs azonosítója, akkor a függvény `true` értékkel tér vissza, és az x csúcsba mozgatja a játékost. Különben a visszatérési érték `false`, és a játékos helyben marad. Ebben a futásban nem hívhatod meg a `setHint` és `setHintLen` függvényt. A programod térjen vissza a `speedrun` függvényből, amint az összes csúcsot meglátogatta.

Korlátok

- $1 \leq N \leq 1\,000$
- Tegyük fel, hogy a `goTo` függvény pontosan Q alkalommal tért vissza `false` értékkel. A részfeladatok az l és Q értékére vonatkozó korlátokat szabják meg, hogy az adott részért pontot kapj.

1. részfeladat (21 pont)

- $l \leq N$
- $Q \leq 2000$
- Ha minden tesztre helyesen fut a programod, 21 pontot kapsz, különben 0 pontot.

2. részfeladat (8 pont)

- $l \leq 20$
- $Q \leq 2000$
- A fa csillag alakú, vagyis létezik egy x csúcs ($1 \leq x \leq N$), amelyik minden más csúccsal szomszédos.
- Ha minden tesztre helyesen fut a programod, 8 pontot kapsz, különben 0 pontot.

3. részfeladat (19 pont)

- $l \leq 20$
- $Q \leq 2000$
- Minden csúcs fokszáma legfeljebb 2.
- Ha minden tesztre helyesen fut a programod, 19 pontot kapsz, különben 0 pontot.

4. részfeladat (12 pont)

- $l \leq 316$
- $Q \leq 32000$
- Ha minden tesztre helyesen fut a programod, 12 pontot kapsz, különben 0 pontot.

5. részfeladat (40 pont)

- $Q \leq 2000$
- Minden tesztre a következő módon számoljuk ki a pontszámot (jelöljük ezt s -sel). Ha $l > 40$, akkor $s = 0$. Ha $l \leq 20$, akkor $s = 40$. Különben, $s = 60 - l$. Azaz ha $l = 40$, akkor a tesztre a pontok felét kapod, ha $l \leq 20$, akkor pedig a teljes pontszámot. A részfeladatra kapott végső pontszám a tesztesetekre járó s pontszámok minimuma.

Példa interakció

Az első futás (segítség-generálás) során a versenyző programját a következő paraméterekkel hívjuk:

```
assignHints(  
  /* subtask = */ 1,  
  /* N       = */ 5,  
  /* A       = */ {-, 1, 2, 3, 3},  
  /* B       = */ {-, 2, 3, 4, 5});
```



Ez a függvény például választhatja az $l = 2$ értéket:

```
setHintLen(/* l = */ 2);
```

És ezután például hagyhatja az összes bitet 0-n, kivéve a 2. csúcs 1. bitjét, amit 1-re állít:

```
setHint(  
    /* i = */ 2,  
    /* j = */ 1,  
    /* b = */ 1);
```

Ezután visszatér. Ezen a ponton a versenyző programját leállítjuk, így a memóriában tárolt értékek törlődnek. Most a segítség minden csúcsban "00", kivéve a 2-dikban, ahol "10".

A második futásban (speedrunning fázis), a versenyző függvényét az alábbi paraméterekkel hívjuk:

```
speedrun(  
    /* subtask = */ 1,  
    /* N       = */ 5,  
    /* start   = */ 1);
```

Válaszul a versenyző kódja elkezd a speedrun-t:

```
getLength(); // = 2  
getHint(1);  // = 0  
getHint(2);  // = 0  
goTo(2);     // = true  
getHint(1);  // = 1  
getHint(2);  // = 0
```

Ezen a ponton, a játékos eredetileg az 1-es csúcsból indult, megtudta, hogy $l = 2$, illetve az 1. csúcsban tárolt segítség "00", ezután sikeresen a 2. csúcsba lépett, és megtudta, hogy az ottani segítség "10". Ezután így folytatódhat a futás:

```
goTo(2);      // = false (a 2. csúcsból önmagába nem léphetsz)  
goTo(5);      // = false (a 2. csúcsból az 5. csúcsba nincs él)  
goTo(3);      // = true  
goTo(4);      // = true  
goTo(3);      // = true  
goTo(5);      // = true
```

Ezen a ponton a játékos minden csúcsot elért, így a `speedrun` függvény kiléphet. $Q = 2$, mivel a program 2 olyan hívást intézett a `goTo` függvényhez, ami `false` értéket adott vissza.