



Problema Speedrun

C++ header `speedrun.h`

Marcel s-a apucat de speedrun-uri, sperând că va obține un nou WR (Record Mondial). Din păcate, el nu poate să stăpânească niciuna din strategiile actuale, așa că trebuie să găsească o modalitate de a obține un avantaj nedrept.

Jocul pe care încearcă să îl învețe se numește *Tree Souls III*, și încearcă să obțină un WR la categoria *full-tree%*.

Jocul se desfășoară, așa cum sugerează și numele, pe un arbore (i.e. un graf aciclic cu N noduri, numerotate de la 1 la N , și $N - 1$ muchii).

Jocul funcționează în felul următor: jucătorul este poziționat arbitrar într-un nod din arbore. El poate alege un număr întreg x , și să încerce să meargă din nodul în care se află în nodul x . Dacă există o muchie între nodul x și nodul în care se află, o să meargă în nodul x , altfel nu se întâmplă nimic. Speedrun-ul este considerat valid dacă vizitează toate nodurile cel puțin o dată.

Acum, intervine trișatul: pentru că el nu cunoaște eroarea de teleportare-pe-muchii, el o să își pregenereze harta, știind astfel cum arată arbore de dinainte să înceapă jocul. Dacă ține minte arborele, va fi prea evident că el trișează, așa că va fi exclus imediat din comunitatea de speedrunning, drept urmare, el va pune în fiecare nod niște indicii (modificând codul). Un indice este un string binar de mărime l (l este același pentru fiecare indice din fiecare nod). Când se află într-un nod, poate să citească indicele din nodul acesta.

Protocolul de interacțiune

Problema aceasta este o problemă cu două interacțiuni!

Codul tău va fi rulat de două ori, o dată pentru prima interacțiune (*etapa de setare a indicilor*), și o dată pentru a doua interacțiune (*etapa de speedrunning*).

Trebuie să implementezi funcțiile următoare (și nu funcția `main`).

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

Folosind următoare funcții care pot fi apelate:

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

Etapa de setare a indicilor. Aceasta este prima rulare a codului tău. În această etapă, codul comisiei o să apeleze funcția `assignHints` o singură dată. O să îi dea ca parametrii *subtask*, care este indexul *subtask*-ului curent, N și muchiile arborelui în A și B în felul următor: pentru fiecare $i = 1, \dots, N - 1$, o să existe o muchie între $A[i]$ și $B[i]$. Apoi trebuie să apelezi funcția `setHintLen` o singură dată, oferindu-i ca parametru lungimea indicilor pe care TU ai ales-o. După ce ai apelat funcția `setHintLen`, poți să apelezi funcția `setHint(i, j, b)` de mai multe ori. Aceasta îi schimbă valoarea bitului j al indicelui nodului i în valoarea b . Biții sunt indexați de la 1 la l . Indiciile au inițial toți biții egali cu zero. În această etapă nu ai voie să apelezi funcțiile `getLength`, `getHint` sau `goTo`. Executarea funcției `assignHints` trebuie să se oprească atunci când termini de setat indiciile.

Etapa de speedrunning. Aceasta este a doua rulare a codului tău. În această etapă, codul comisiei va apela funcția `speedrun(subtask, N, start)` o singură dată, oferindu-i ca parametrii *subtask*, care



este indexul subtask-ului curent, valoarea N și nodul **start** din care pornește speedrunner-ul. Poți apela apoi funcția `getLength()`, care returnează l , lungimea indicilor aleasă de tine în prima etapă, funcția `getHint(j)` care returnează bitul j al indicelui din nodul în care te afli, și funcția `goTo`. Dacă parametrul funcției `goTo` este indexul unui nod pentru care există o muchie între el și nodul în care te afli, atunci funcția returnează **true** și mergi în acel nod. Altfel, returnează **false** și rămâi în nodul curent. În această etapă nu ai voie să apelezi funcțiile `setHint` și `setHintLen`. Executarea funcției `speedrun` trebuie să se oprească după ce ai vizitat toate nodurile din arbore.

Restricții

- $1 \leq N \leq 1000$
- Fie Q numărul de apeluri al funcției `goTo` care au returnat *false*. În ceea ce urmează, conturăm restricții pentru l și Q care trebuie să fie respectate pentru a obține scorul pe subtask-urile respective.

Subtask 1 (21 de puncte)

- $l \leq N$
- $Q \leq 2000$
- Scorul pe subtask va fi 21 dacă toate testele au fost rezolvate corect, 0 altfel.

Subtask 2 (8 puncte)

- $l \leq 20$
- $Q \leq 2000$
- Arborele este o stea; i.e. există un nod x ($1 \leq x \leq N$) care are muchie către toate celelalte noduri.
- Scorul pe subtask va fi 8 dacă toate testele au fost rezolvate corect, 0 altfel.

Subtask 3 (19 puncte)

- $l \leq 20$
- $Q \leq 2000$
- Gradul fiecărui nod este cel mult 2.
- Scorul pe subtask va fi 19 dacă toate testele au fost rezolvate corect, 0 altfel.

Subtask 4 (12 puncte)

- $l \leq 316$
- $Q \leq 32000$
- Scorul pe subtask va fi 12 dacă toate testele au fost rezolvate corect, 0 altfel.

Subtask 5 (40 de puncte)

- $Q \leq 2000$
- Scorul s pentru fiecare test este calculat în felul următor. Dacă $l > 40$, atunci $s = 0$. Dacă $l \leq 20$, atunci $s = 40$. Altfel, $s = 60 - l$. Astfel, dacă $l = 40$, o să obții jumătate din punctele pe acel test, iar dacă $l \leq 20$, o să obții maximul de puncte pe acel test. Scorul pe subtask va fi minimul valorilor s pe toate testele din acel subtask.

Exemplu de interacțiune

În prima interacțiune (etapa de setare a indicilor), funcția concurentului va fi apelată în felul următor:

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



Această funcție, poate alege $l = 2$:

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

Și să lase valoarea biților tuturor indicilor să fie 0, cu excepția bitului 1 al indicelui din nodul 2, care va fi egal cu 1:

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

Apoi, se oprește. Prima instanță a programului concurentului va fi oprită, și orice informație stocată în memoria acestui program se va pierde. Până acum, indiciile sunt "00" pentru fiecare nod, exceptând nodul 2, care are indicele "10".

În a doua interacțiune (etapa de speedrunning), funcția concurentului va fi apelată:

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

În schimb, funcția va începe să facă speedrunning pe arbore:

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

În acest punct, te aflii inițial în nodul 1, aflii că $l = 2$ și că indicele nodului 1 este "00", iar apoi te muți în nodul 2 și aflii că indicele nodului 2 este "10". Execuția poate continua în felul următor:

```
goTo(2);      // = false (nu poti merge din nodul 2 in el insusi)  
goTo(5);      // = false (nu exista muchie intre nodul 2 si nodul 5)  
goTo(3);      // = true  
goTo(4);      // = true  
goTo(3);      // = true  
goTo(5);      // = true
```

În acest punct au fost vizitate toate nodurile, așa că funcția `speedrun` se poate termina. Valoarea $Q = 2$, se datorează celor 2 apeluri ale funcției `goTo` care au returnat `false`.