

Újjáépítés

Időlimit: 4 másodperc

Memórialimit: 1024 MiB

Ez egy interaktív feladat.

Bissy hamarosan részt vesz a *Final Destination* nevű, önkéntesek által szervezett tájfutó játékban, amelyben az EJOI-küldöttségek versengenek egymással, hogy meglátogassák a város különböző helyszíneit. Bissy azonban nem annyira a győzelem érdekli, hanem inkább az, hogy *hogyan* tervezték meg a játékot.

A térképen N helyszín található és N csapat van. Minden csapatnak meg kell látogatnia az összes helyszínt, és a csapatok indulási helyszínei különbözőek – az i -edik csapat az i -edik helyszínről indul. Minden csapat megkapja a saját útvonalát.

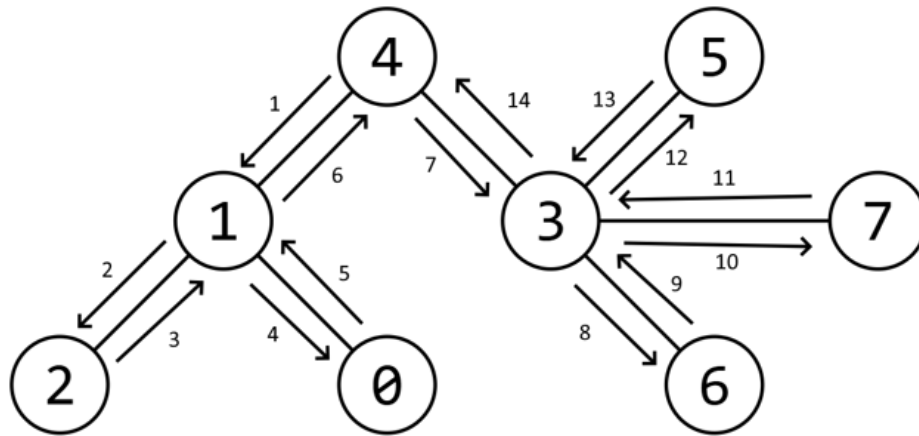
Bissynek van némi belső információja is – az útvonalak a gyors tömegközlekedési vonalak rejtett szerkezetén alapulnak. Pontosabban: a verseny zsűrije kiválasztott $N - 1$ járatot, amelyek mindegyike két helyszínt köt össze úgy, hogy ezekkel a járatokkal minden helyszín elérhető bármely másiktól. Az ilyen szerkezetet fának is nevezik. Ezután minden i -edik kiindulási helyszínhez a zsűri generált egy **tetszőleges mélységi bejárást** (ezt alább definiáljuk) és ezt a bejárást jelölte ki az i -edik csapat útvonalaként.

Bissy ki akarja deríteni, hogy a zsűri mely tömegközlekedési fát választotta ki. Csak olyan gyors kérdéseket tehet fel, mint például:

- Mi az i -edik csapat j -edik helyszíne?

Segíts Bissynek, és írd egy `find_tree` nevű programot, amely megtalálja a rejtett fát.

Egy fa *mélységi bejárása* az a sorrend, amelyben a csúcsait egy mélységi keresés során először meglátogatják. Egy ilyen eljárás egy adott v csúcsból indul, és rekurzív módon annak az egyik, még meg nem látogatott szomszédjához halad mindaddig, amíg ilyen szomszéd már nem létezik — ekkor visszatér az előzőleg meglátogatott csúcsra, és onnan folytatja. Példa:



Itt $v = 4$, és az élek mentén lévő nyilak a mélységi keresés lépéseinek felelnek meg. A generált bejárás: $[4, 1, 2, 0, 3, 6, 7, 5]$. Megjegyzendő, hogy a szomszédok meglátogatásának sorrendje fontos; egy másik bejárás például $[4, 3, 5, 7, 6, 1, 0, 2]$ lehetne.

A fa szerkezete és az N mélységi bejárás a program indítása előtt rögzítésre kerül, és a futtatás során nem változik. A szomszédos csúcsok sorrendje az ilyen mélységi bejárások során eltérő lehet.

Megvalósítás

A következő függvényt kell megvalósítanod:

```
std::vector<std::pair<int, int>> find_tree(int N)
```

- itt N a helyszínek száma;
- a visszatérési érték a fa éleinek listája — az élek sorrendje és végpontjaik nem számítanak.

Ez a függvény minden tesztesetnél legfeljebb T -szer hívható meg.

A zsűrivel való kommunikációhoz a következő függvényt hívhatod meg:

```
int guess(int i, int j)
```

Ez a függvény a fa i -edik útvonalának j -edik helyszínét adja vissza. Ne felejtse el, hogy a `guess(i, 0)` hívás i -t ad vissza. Feltételezheted, hogy a függvény a 6. részfeladatig az összes részfeladat esetén konstans időben, $O(1)$ -ben, a 7. részfeladat esetén pedig logaritmikus időben, $O(\log N)$ -ben válaszol. Erre érvényesnek kell lennie, hogy $0 \leq i, j \leq N - 1$, ellenkező esetben a megoldásod a következő értékelést kapja: `Output isn't correct: Invalid call.`

Korlátok

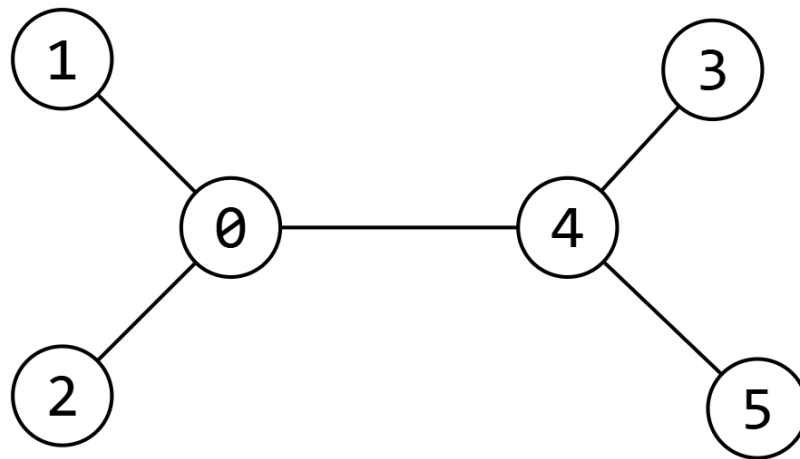
- $2 \leq N \leq 2^{16} + 1$

- T -nek meg kell felelnie a korlátozásoknak, amelyek az egy tesztben előforduló függvényhívások közül a legnagyobb N -re (amelyet N_{max} -ként jelölünk) vonatkoznak:
 - ha $N_{max} \leq 9$, akkor $1 \leq T \leq 100$
 - ha $N_{max} \leq 2^{10} + 1$, akkor $1 \leq T \leq 10$
 - ha $N_{max} \leq 2^{16} + 1$, akkor $1 \leq T \leq 3$

A rendszer értékelője legfeljebb 280 MiB-ot használhat fel, ami beleszámít a memórialimitbe.

Példa

Legyen a gyors tömegközlekedési járatok rejtett szerkezete a következő:



Tegyük fel, hogy a 0 kiindulási helyszín esetében a bejárás útvonala $[0, 1, 2, 4, 3, 5]$. Ekkor egy példa a lehetséges interakcióra:

Saját programod	Értékelőprogram
	<code>find_tree(6)</code>
<code>guess(0, 0)</code>	0
<code>guess(0, 1)</code>	1
<code>guess(0, 2)</code>	2
<code>guess(0, 3)</code>	4
<code>guess(0, 4)</code>	3
<code>guess(0, 5)</code>	5
<code>return {{0,1},{0,2},{4,0},{5,4},{3,4}};</code>	

Figyeljük meg, hogy a lekérdezésekből származó információk nem elegendőek a fa egyértelmű meghatározásához, de ez valójában a 0. részfeladat tesztjének megoldása.

Részfeladatok

Részfeladat	Pontszám	N	További korlátok
0	0	-	A példa.
1	11	≤ 9	-
2	6	≤ 100	Minden csúcs legfeljebb 2 másik csúccsal van összekötve.
3	13	≤ 100	Minden út egy olyan mélységi bejárással generálódik, amely minden lépésnél elsőbbséget ad a 0 csúcstól való eltávolodásnak. Ha több lehetőség is van a 0 csúcstól való eltávolodásra, akkor ezek közül tetszőlegesen választjuk ki az egyiket.
4	11	≤ 100	Bármely 0-tól eltérő csúcs legfeljebb 2 másik csúccsal van összekötve.
5	10	≤ 100	-
6	31	$\leq 2^{10} + 1$	-
7	18	$\leq 2^{16} + 1$	-

Pontozás

Az 5. részfeladatig minden részfeladat esetében a maximális pontszámot kapod, ha a megadott időkorláton belül sikeresen megtalálod a fát. A 6. és 7. részfeladatok esetében az adott teszt teljes pontszámának S aránya attól függ, hogy **maximum** hány lekérdezést hajtasz végre valamelyik teszteseten, azaz a Q_{max} értéktől.

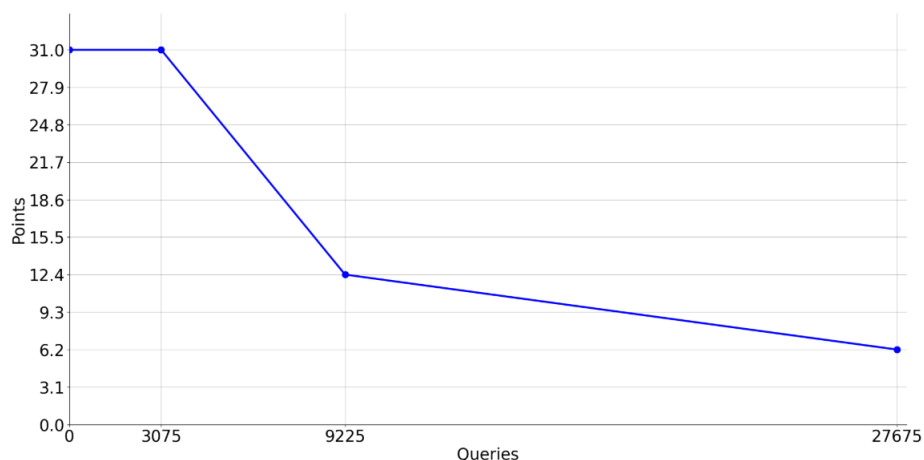
Ezeknek a részfeladatoknak a pontozása a következő:

- ha $Q_{max} \leq L_1$, akkor $S = 1, 0$;
- ha $L_1 < Q_{max} \leq L_2$, akkor $S = 0, 4 + 0, 6 \times \frac{L_2 - Q_{max}}{L_2 - L_1}$;
- ha $L_2 < Q_{max} \leq 3L_2$, akkor $S = 0, 2 + 0, 2 \times \frac{3L_2 - Q_{max}}{2L_2}$;
- ha $3L_2 < Q_{max}$, akkor $S = 0, 2$;

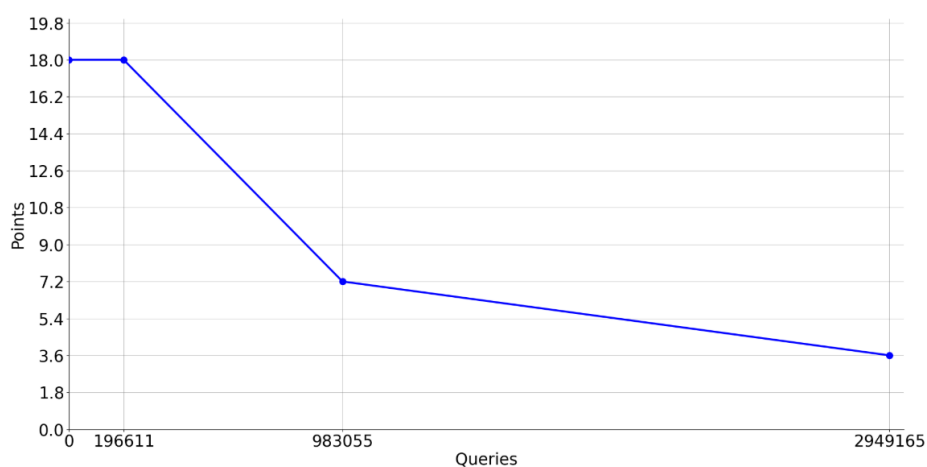
ahol a 6. részfeladat esetén $L_1 = 3 \times (2^{10} + 1) = 3075$ és $L_2 = 9 \times (2^{10} + 1) = 9225$, a 7. részfeladat esetén pedig $L_1 = 3 \times (2^{16} + 1) = 196611$ és $L_2 = 15 \times (2^{16} + 1) = 983055$. Az egész részfeladatból kapott pontok aránya az összes teszt eset közül a legkisebb S érték.

Az alábbi grafikonok az utolsó két részfeladat pontszámítási függvényét mutatják:

6. részfeladat



7. részfeladat



Mintaértékelő

Két mintaértékelő áll rendelkezésre.

Helyi teszteléshez: az `Lgrader.cpp` fájlt a programoddal együtt lefordíthatod, így egy tesztelő eszközt kapsz. A program beolvassa a tesztesetek T számát, mindegyik tesztesetre a helyszínek N számát, amelyet $N - 1$ sor követ, amelyek az egy egyes tömegközlekedési járatokat leíró egész számokból álló számpárokat tartalmazzák. Ezt követően N sor következik, amelyek mindegyike N egész számot tartalmaz – ezek a mélységi bejárások. Ne felejtse el, hogy az i -edik bejárásnak az i csúcsból kell indulnia. Az `AUTO_GENERATE` konstans `true` értékre állításával az értékelőprogramot beállíthatod úgy, hogy automatikusan generálja a bejárásokat. Kimenetként az értékelőprogram hibaüzenetet ad, ha az eredmény helytelen, egyébként pedig az egyes tesztesetekre vonatkozó lekérdezések számát és az összesített maximumot adja meg. Figyelj arra is, hogy az értékelő nem működik nagy N esetén (pontosabban a 7. részfeladatra), itt szabadon használhatod a fantáziádat.

Versenyszerbrendszertel tesztetekhez: a `stub.cpp` fájl a programoddal együtt használható a rendszeren végzett felhasználói tesztetekhez. A bemenet formátuma megegyezik a másik értékelőével. Figyelj

arra, hogy a felhasználói tesztekhez nincs beépített opció a bejárások automatikus generálására.