

Reconstruct

Limita de timp: 4 secunde

Limita de memorie: 1024 MiB

Aceasta este o problemă interactivă.

Bissy este pe cale să participe la jocul de orientare organizat de voluntari numit *Destinație finală*, unde delegațiile EJOI vor concura să viziteze diferite locații din oraș. Dar Bissy este mai interesat de *felul în care* jocul este construit decât de câștigarea lui.

Pe hartă se află N locații și există N echipe. Fiecare echipă trebuie să viziteze toate locațiile și să aibă locații de start unice — echipa i pornește din locația i . Fiecare echipă primește propria sa rută.

Bissy are și unele informații din interior — rutele se bazează pe o structură ascunsă de linii de transport public rapid. Mai precis, juriul competiției a ales $N - 1$ linii, fiecare conectând o pereche de locații, astfel încât fiecare locație este accesibilă din oricare alta folosind aceste linii. O astfel de structură se numește și arbore. Apoi, pentru fiecare locație de start i , juriul a generat o *parcursere DFS arbitrară* (definită mai jos) și a atribuit această parcursere ca rută pentru echipa i .

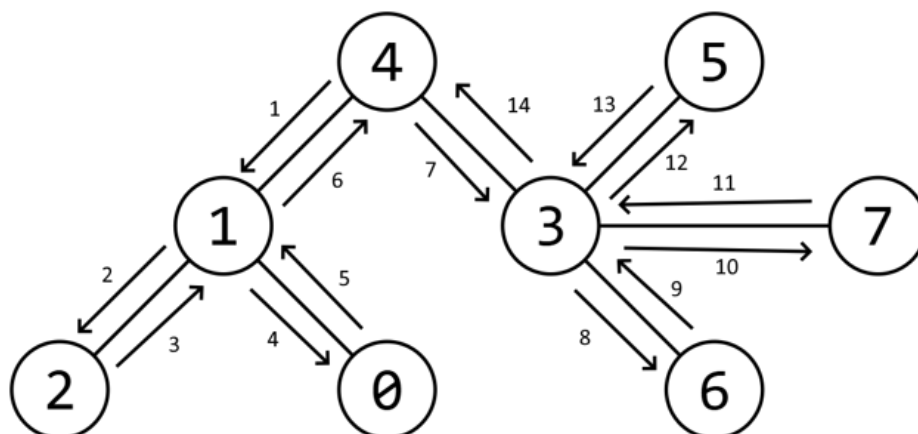
Bissy dorește să afle structura de transport public (arborele) pe care l-a selectat juriul. Ea pune doar întrebări rapide de felul:

- Care este destinația a j -a a echipei a i -a?

Ajută-o pe Bissy scriind un program `find_tree` care găsește arborele ascuns.

O *parcursere DFS* a unui arbore este ordinea în care nodurile sale sunt vizitate pentru prima dată de o procedură depth-first-search. O astfel de procedură începe de la un nod dat v și merge recursiv la unul dintre vecinii săi nevizitați până când nu mai există niciun astfel de vecin — apoi se întoarce la nodul vizitat anterior și continuă de acolo.

Exemplu:



Aici $v = 4$ și săgețile de-a lungul muchiilor corespund pașilor unei parcurgeri depth-first-search. Parcurgerea generată este $[4, 1, 2, 0, 3, 6, 7, 5]$. Observați că ordinea în care vizităm vecinii contează; o altă parcurgere ar putea fi $[4, 3, 5, 7, 6, 1, 0, 2]$.

Structura arborelui și N parcurgeri DFS sunt fixate înainte ca programul vostru să înceapă și nu se modifică în funcție de întrebările voastre. Ordinea vecinilor vârfurilor utilizată în aceste DFS-uri poate fi diferită.

Detalii de implementare

Trebuie să implementați următoarea funcție:

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

- unde N este numărul de locații;
- valoarea returnată este o listă cu muchiile arborelui — nici ordinea muchiilor și nici a capetelor lor nu contează.

Pentru fiecare test, această funcție poate fi apelată de cel mult T ori.

Pentru a interacționa cu juriul, puteți apela următoarea funcție:

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

Returnează locația a j -a în a i -a parcurgere a arborelui. Rețineți că `guess(i, 0)` va returna i . Puteți presupune că funcția răspunde în timp constant $O(1)$ pentru toate subtaskurile până la 6, și în timp logaritm $O(\log N)$ pentru subtask-ul 7. Condiția $0 \leq i, j \leq N - 1$ trebuie să fie îndeplinită, în caz contrar soluția voastră va primi verdictul `Output isn't correct: Invalid call` (Răspuns greșit: Apel invalid).

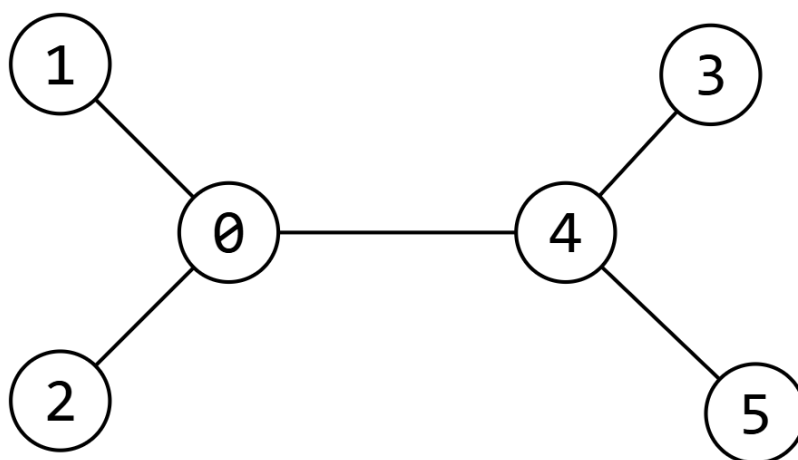
Restricții

- $2 \leq N \leq 2^{16} + 1$
- T îndeplinește restricțiile în funcție de valoarea maximă a lui N (notată cu N_{max}) dintre apelurile funcției dintr-un singur test:
 - dacă $N_{max} \leq 9$, atunci $1 \leq T \leq 100$
 - dacă $N_{max} \leq 2^{10} + 1$, atunci $1 \leq T \leq 10$
 - dacă $N_{max} \leq 2^{16} + 1$, atunci $1 \leq T \leq 3$

Grader-ul sistemului poate utiliza cel mult 280 MiB, care se adaugă la memoria utilizată de soluția voastră.

Exemplu

Să presupunem că structura ascunsă a sistemului de transport rapid este următoarea:



Să presupunem că locația de start este 0, și parcurgerea este $[0, 1, 2, 4, 3, 5]$. Atunci, ceea ce urmează este un exemplu de interacțiune:

Programul concurentului	Programul juriului
	<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>	

Rețineți că informațiile din interogări nu sunt suficiente pentru a determina în mod unic arborele, dar acesta este de fapt răspunsul la testul din subtask-ul 0.

Subtask-uri

Subtask	Puncte	N	Restricții suplimentare
0	0	-	Exemplul.
1	11	≤ 9	-
2	6	≤ 100	Fiecare vârf este conectat cu cel mult 2 alte vârfuri.
3	13	≤ 100	Fiecare parcurgere este generată de un depth-first-search care la fiecare pas prioritizează îndepărtarea de vârful 0. Dacă există mai multe opțiuni pentru a se îndepărta de vârful 0, una dintre ele este aleasă arbitrar.
4	11	≤ 100	Orice vârf diferit de 0 este conectat cu cel mult 2 alte vârfuri.
5	10	≤ 100	-
6	31	$\leq 2^{10} + 1$	-
7	18	$\leq 2^{16} + 1$	-

Punctare

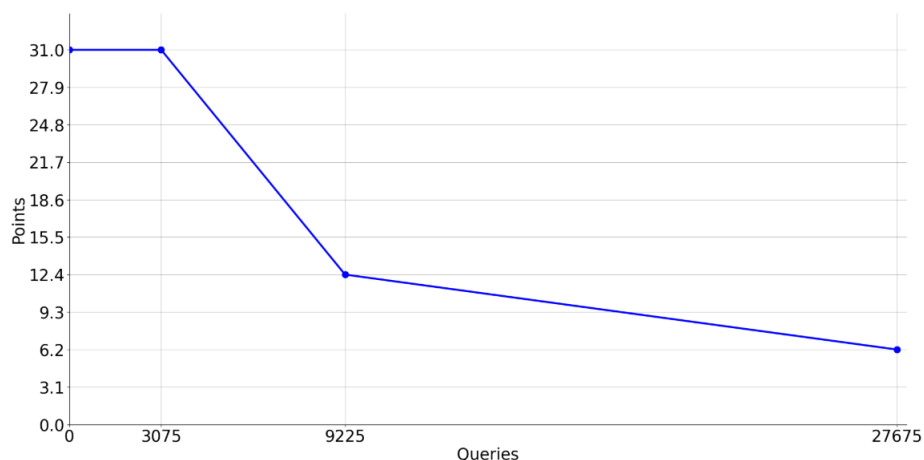
Pentru fiecare subtask până la subtask-ul 5, obțineți punctajul complet dacă găsiți cu succes arborele în limitele de timp impuse. Pentru subtaskurile 6 și 7, fracțiunea din punctajul complet S pentru un test dat depinde de **maximul** numărului de întrebări pe care le puneți într-unul dintre subtestele sale, Q_{max} . Punctajul pentru aceste subtask-uri este următorul:

- dacă $Q_{max} \leq L_1$, atunci $S = 1.0$;
- dacă $L_1 < Q_{max} \leq L_2$, atunci $S = 0.4 + 0.6 \times \frac{L_2 - Q_{max}}{L_2 - L_1}$;
- dacă $L_2 < Q_{max} \leq 3L_2$, atunci $S = 0.2 + 0.2 \times \frac{3L_2 - Q_{max}}{2L_2}$;
- dacă $3L_2 < Q_{max}$, atunci $S = 0.2$;

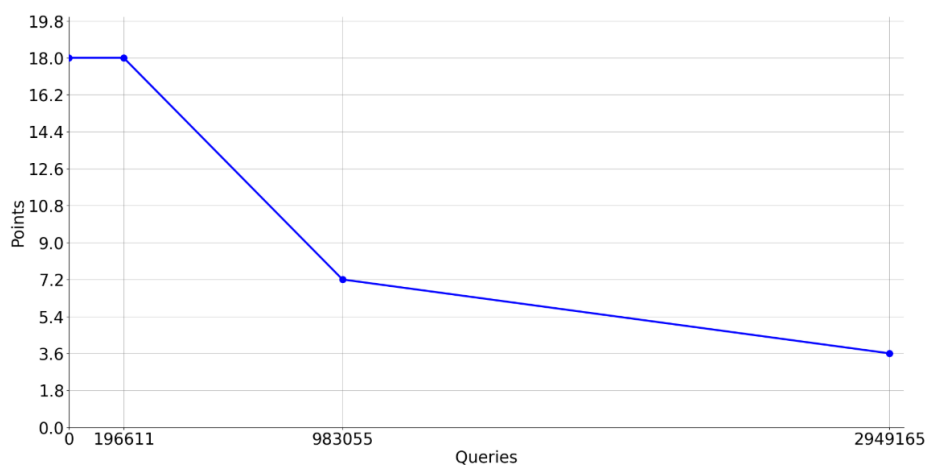
unde $L_1 = 3 \times (2^{10} + 1) = 3075$ și $L_2 = 9 \times (2^{10} + 1) = 9225$ pentru subtask-ul 6, și $L_1 = 3 \times (2^{16} + 1) = 196611$ și $L_2 = 15 \times (2^{16} + 1) = 983055$ pentru subtask-ul 7. Frațiunea din punctaj pe care o primiți pentru întregul subtask este minimul lui S pentru toate testele.

Graficele următoare ilustrează funcția de punctare pentru ultimele două subtask-uri:

Subtask 6



Subtask 7



Sample grader

Sunt oferite două sample graders.

Pentru testare locală: `Lgrader.cpp` poate fi compilat împreună cu programul vostru pentru a oferi o unealtă de testare.

Programul citește numărul T de cazuri de testare și pentru fiecare așteaptă numărul de locații N urmat de $N - 1$ linii, fiecare cu o pereche de numere întregi care descrie o linie directă. Urmează N linii cu câte N numere întregi – parcurgerile DFS. Rețineți că parcurgerea i trebuie să înceapă din vârful i . Puteți lăsa graderul să genereze parcurgeri pentru voi setând constanta `AUTO_GENERATE` la `true`. Ca rezultat, graderul raportează fie un mesaj de eroare dacă rezultatul este incorect, fie numărul de interogări efectuate pentru fiecare test și un maxim general. Rețineți, de asemenea, că grader-ul nu funcționează pentru N suficient de mare (mai precis, pentru restricțiile subtask-ului 7), sunteți liberi să vă folosiți imaginația pentru acel caz.

Pentru teste create de utilizator pe sistemul de evaluare: `stub.cpp` poate fi utilizat împreună cu programul vostru pentru teste create de utilizator pe sistemul de evaluare. Formatul de intrare este același ca la celalalt grader. Rețineți că pentru testele utilizatorului nu există o opțiune încorporată pentru generarea automată a parcurgerilor.