

Rekonstrukcija

Laika ierobežojums: 4 sekundes

Atmiņas ierobežojums: 1024 MiB

Šis ir interaktīvs uzdevums.

Bisija drīzumā piedalīsies brīvprātīgo organizētajā orientēšanās spēlē *Pēdējā pietura*, kurā EJOI delegācijas sacentīsies, apmeklējot dažādas vietas pilsētā. Taču Bisiju vairāk interesē tas, kā spēle tika izveidota, nevis uzvara tajā.

Kartē ir N vietas un N komandas. Katrai komandai jāapmeklē visas vietas, un visām komandām ir atšķirīgas sākuma vietas — komanda i sāk no vietas i . Katrai komandai tiek dots savs maršruts.

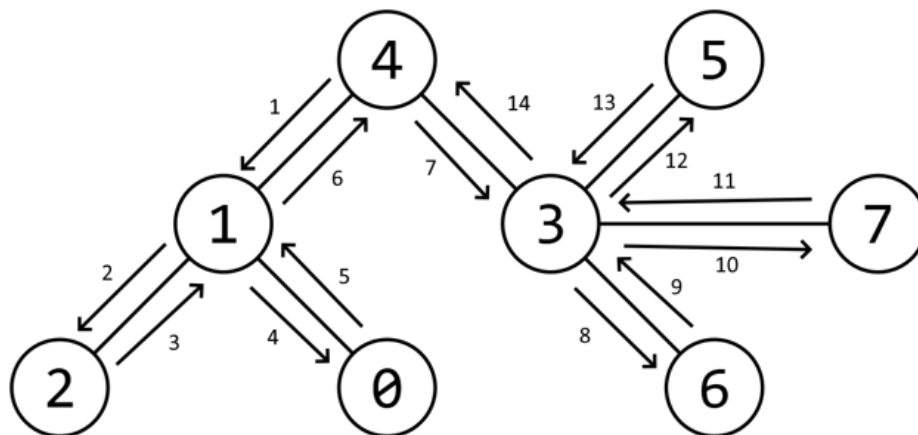
Bisijas rīcībā ir arī iekšēja informācija — maršruti ir balstīti uz slēptu ātrā sabiedriskā transporta līniju struktūru. Precīzāk, sacensību žūrija izvēlējās $N - 1$ līnijas, katra no tām savieno vienu vietu pāri, tā, ka no jebkuras vietas, izmantojot šīs līnijas, ir sasniedzama jebkura cita vieta. Šādu struktūru sauc arī par koku. Pēc tam katrai sākuma vietai i žūrija izvēlējās **patvaļīgu** DFS apstaigu (definēta zemāk) un piešķīra šo apstaigu kā maršrutu komandai i .

Bisija vēlas noskaidrot, kuru sabiedriskā transporta koku žūrija ir izvēlējusies. Viņa var uzdot tikai vienkāršus vaicājumus:

- Kāda ir i -tās DFS apstaigas j -tā vieta?

Palīdziet Bisijai, uzrakstot programmu `find_tree`, kas atrod slēpto koku.

Koka DFS apstaiga ir secība, kādā tā virsotnes pirmo reizi tiek apmeklētas procedūras "meklēšana dziļumā" (*depth-first search*) laikā. Šāda procedūra sākas dotajā virsotnē v un rekursīvi pāriet uz kādu no tās neapmeklētajiem kaimiņiem, līdz tādu vairs nav — tad tā atgriežas iepriekš apmeklētajā virsotnē un turpina no turienes. Piemērs:



Šeit $v = 4$, un bultas gar šķautnēm atbilst "meklēšanas dziļumā" soļiem. Iegūtā apstaiga ir $[4, 1, 2, 0, 3, 6, 7, 5]$. Nemiet vērā, ka secība, kādā tiek apmeklēti kaimiņi, ir svarīga; cita iespējamā apstaiga ir $[4, 3, 5, 7, 6, 1, 0, 2]$.

Koka struktūra un N DFS apstaigas tiek fiksēti pirms jūsu programmas palaišanas un nemainās atkarībā no jūsu vaicājumiem. Virsotņu kaimiņu apmeklēšanas secība dažādās DFS apstaigās var atšķirties.

Implementācijas detaļas

Jums ir jāimplementē šāda funkcija:

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

- N : vietu skaits.

Šai funkcijai ir jāatgriež koka šķautņu saraksts. Šķautņu secībai un to galapunktu secībai nav nozīmes.

Katrā testā šī funkcija var tikt izsaukta līdz T reizēm.

Jūsu risinājums var veikt vaicājumus, izsaucot funkciju:

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

Šī funkcija atgriež j -to vietu i -tajā DFS apstaigā.

Nemiet vērā, ka `guess(i, 0)` atgriež i . Var pieņemt, ka funkcija atbild konstantā laikā $O(1)$ visos apakšuzdevumos līdz 6. apakšuzdevumam (ieskaitot), un logaritmiskā laikā $O(\log N)$ 7. apakšuzdevumā. Jāievēro nosacījums $0 \leq i, j \leq N - 1$, pretējā gadījumā jūsu risinājums saņems verdiktu `Output isn't correct: Invalid call`.

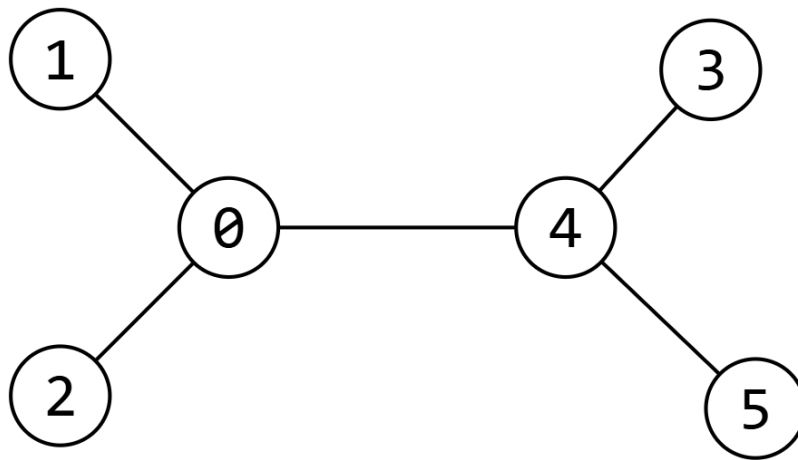
Ierobežojumi

- $2 \leq N \leq 2^{16} + 1$
- T ierobežojumi ir atkarīgi no lielākā N (apzīmēts ar N_{max}) starp funkcijas izsaukumiem vienā testā:
 - ja $N_{max} \leq 9$, tad $1 \leq T \leq 100$
 - ja $N_{max} \leq 2^{10} + 1$, tad $1 \leq T \leq 10$
 - ja $N_{max} \leq 2^{16} + 1$, tad $1 \leq T \leq 3$

Vērtēšanas sistēma var izmantot līdz 280 MiB, un tas tiek ieskaitīts jūsu risinājuma atmiņas patēriņā.

Piemērs

Pieņemsim, ka slēptā ātrā sabiedriskā transporta līniju struktūra ir šāda:



Pieņemsim, ka sākuma vietai 0 apstaiga ir $[0, 1, 2, 4, 3, 5]$. Tad iespējama šāda mijiedarbība:

Jūsu programma	Vērtēšanas sistēma
	<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>	

Šajā piemērā vaicājumos iegūtā informācija nav pietiekama, lai viennozīmīgi noteiktu koku, taču tieši šī ir atbilde šim piemēram.

Apakšuzdevumi

Apakšuzdevums	Punkti	N	Papildu ierobežojumi
0	0	-	Piemērs.
1	11	≤ 9	-
2	6	≤ 100	Katra virsotne ir savienota ar ne vairāk kā 2 citām.
3	13	≤ 100	Katra apstaiga ir izveidota ar meklēšanu dziļumā, kas katrā solī dod priekšroku pārvietoties prom no virsotnes 0. Ja ir vairākas iespējas pārvietoties prom no virsotnes 0, viena no tām tiek izvēlēta patvaļīgi.
4	11	≤ 100	Katra virsotne, izņemot virsotni 0, ir savienota ar ne vairāk kā 2 citām.
5	10	≤ 100	-
6	31	$\leq 2^{10} + 1$	-
7	18	$\leq 2^{16} + 1$	-

Vērtēšana

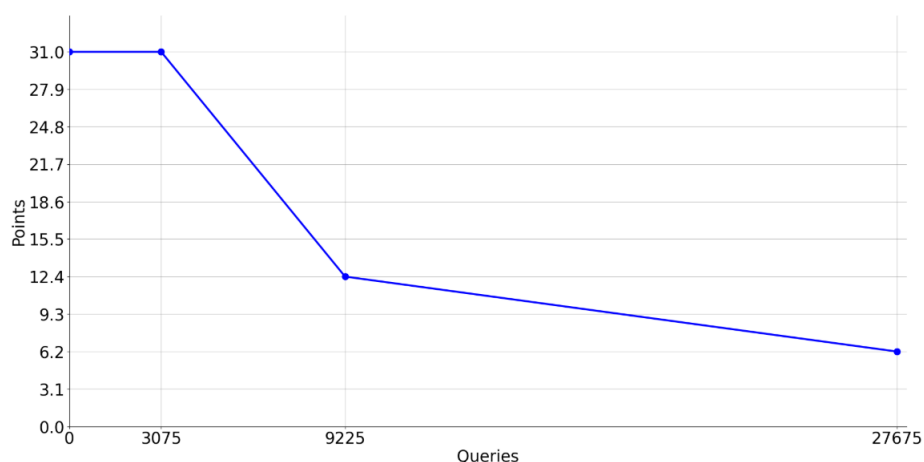
Par katru apakšuzdevumu līdz 5. apakšuzdevumam (ieskaitot) jūs iegūstat 100% punktu, ja veiksmīgi atrodat koku, ievērojot laika ierobežojumu. 6. un 7. apakšuzdevumā iegūtā punktu daļa S konkrētajam testam ir atkarīga no **lielākā** vaicājumu skaita, ko uzdod kādā no tā apakštestiem, Q_{max} . Vērtēšana šajos apakšuzdevumos ir šāda:

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

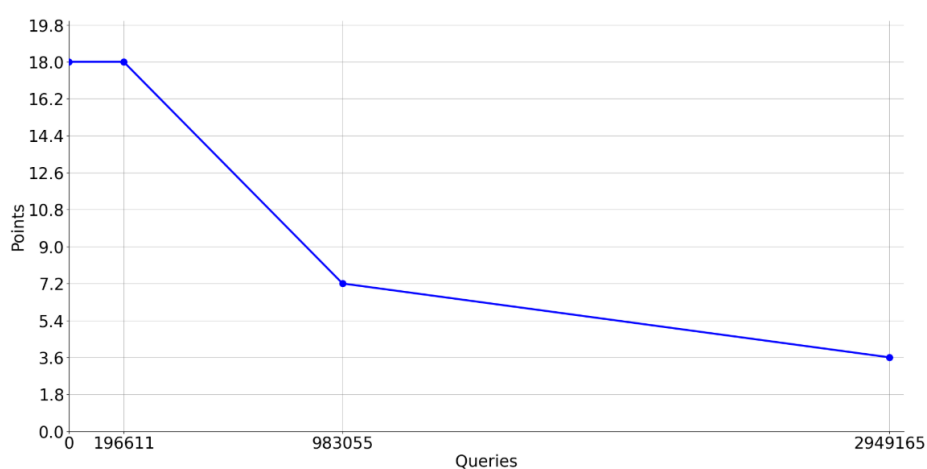
kur 6. apakšuzdevumam $L_1 = 3 \times (2^{10} + 1) = 3075$ un $L_2 = 9 \times (2^{10} + 1) = 9225$, bet 7. apakšuzdevumam $L_1 = 3 \times (2^{16} + 1) = 196611$ un $L_2 = 15 \times (2^{16} + 1) = 983055$. Punktu daļa, ko iegūstat par visu apakšuzdevumu, ir mazākais S starp visiem tā testiem.

Zemāk redzamie grafiki attēlo vērtēšanas funkciju pēdējiem diviem apakšuzdevumiem:

6. apakšuzdevums



7. apakšuzdevums



Piemēra vērtētājprogramma

Ir pieejamas divas piemēra vērtētājprogrammas.

Lokālai testēšanai: `Lgrader.cpp` var tikt kompilēts kopā ar jūsu programmu, nodrošinot testēšanas rīku. Programma nolasa testu skaitu T un katram no tiem gaida vietu skaitu N , kam seko $N - 1$ rindas, katrā divi veseli skaitļi, kas apraksta vienu tiešo līniju. Pēc tam seko N rindas ar N veseliem skaitļiem – *DFS apstaigas*. Ņemiet vērā, ka i -tajai apstaigai jāsākas virsotnē i . Vērtētājprogrammai iespējams ģenerēt apstaigas jūsu vietā, nomainot konstanti `AUTO_GENERATE` uz `true`. Kā izvadi vērtētājs vai nu ziņo kļūdas paziņojumu, ja rezultāts ir nepareizs, vai arī katram testam uzdoto vaicājumu skaitu un kopējo maksimumu. Vērtētājs nedarbojas pietiekami lieliem N (precīzāk, 7. apakšuzdevuma ierobežojumiem), tādā gadījumā varat izmantot savu iztēli.

Sistēmas lietotāja testiem: `stub.cpp` var izmantot kopā ar jūsu programmu lietotāja testos sistēmā. Ievaddatu formāts ir tāds pats kā pirmajai vērtētājprogrammai. Sistēmas lietotāja testiem nav iebūvētas iespējas automātiski ģenerēt apstaigas.