

Taastamine

Ajapiirang: 4 sekundit

Mälupiirang: 1024 MiB

See on interaktiivne ülesanne

Bissy on valmis osalema vabatahtlike korraldatud orienteerumismängus *Final destination*, kus EJOI delegatsioonid võistlevad, et külastada erinevaid asukohti üle linna. Kuid Bissy huvitab mängu võitmisest hoopis rohkem see, *kuidas* mäng on loodud.

Kaardil on N asukohta ja N võistkonda. Iga võistkond peab külastama kõiki asukohti ning igal võistkonnal on erinev alguskoht — võistkond i alustab asukohast i . Igale võistkonnale antakse oma marsruut.

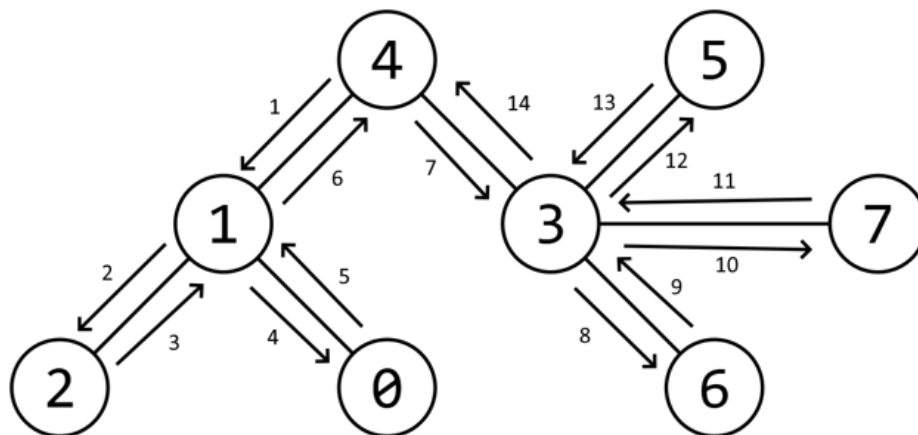
Bissyl on ka natuke siseinfot — marsruudid põhinevad kiirete ühistranspordiliinide varjatud struktuuril. Täpsemalt, võistluse žürii valis $N - 1$ liini, millest igaüks ühendab kahte asukohta nii, et iga asukoht on kõigist teistest nende liinide abil ligipääsetav. Sellist struktuuri nimetatakse ka puuks. Seejärel, iga alguspunkti i jaoks, koostas žürii **suvalise** *DFS-jalutuskäigu* (defineeritud allpool) ja määras selle jalutuskäigu marsruudiks meeskonnale i .

Bissy tahab välja mõelda, millise ühistranspordipuu žürii valis. Ta esitab ainult kiireid küsimusi kujul:

- Mis on i -nda meeskonna j -s sihtkoht?

Aita Bissy ja kirjuta programm `find_tree`, mis leiab peidetud puu.

Puu *DFS-läbimine* on järjekord, milles selle tipud külastatakse esimest korda sügavuti läbimise (depth-first-search) protseduuri poolt. Selline protseduur algab antud tipust v ja liigub rekursiivselt ühte selle külastamata naabritest, kuni sellist naabrit enam ei leidu — siis läheb see tagasi eelnevalt külastatud tippu ja jätkab sealt. Näide:



Siin $v = 4$ ja servade peal olevad nooled vastavad sügavuti läbimise sammudele. Loodud jalutuskäik on $[4, 1, 2, 0, 3, 6, 7, 5]$. Pane tähele, et järjekord, milles me naabreid külastame, on oluline; teine võimalik jalutuskäik võiks olla $[4, 3, 5, 7, 6, 1, 0, 2]$.

Puu struktuur ja N DFS-jalutuskäiku on kindlaks määratud enne sinu programmi käivitumist ega muutu sinu küsimustele vastates. Selliste DFS-ide ajal kasutatud tippude naabrite järjekorrad võivad olla erinevad.

Teostuse üksikasjad

Sul tuleb realiseerida järgmine funktsioon:

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

- siin on N asukohtade arv;
- tagastatav väärtus on puu servade loend — servade ja nende otspunktide järjekord ei ole oluline.

Iga testi puhul saab seda funktsiooni kutsuda kuni T korda.

Žüriiga suhtlemiseks saad kutsuda järgmist funktsiooni:

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

See tagastab puu i -nda jalutuskäigu j -nda asukoha. Pane tähele, et `guess(i, 0)` tagastab i . Võid eeldada, et funktsioon vastab konstantse ajaga $O(1)$ kõigi alamülesannete puhul kuni 6 ja logaritmilise ajaga $O(\log N)$ alamülesande 7 puhul. Peab kehtima, et $0 \leq i, j \leq N - 1$, vastasel juhul saab su lahendus otsuse `Output isn't correct: Invalid call.`

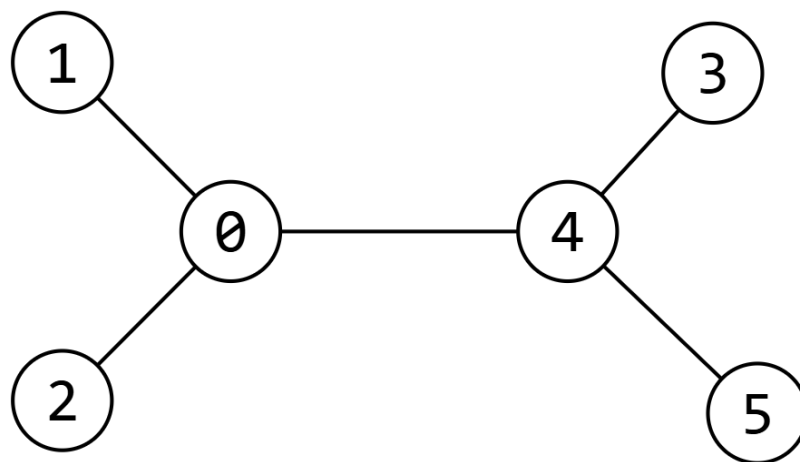
Piirangud

- $2 \leq N \leq 2^{16} + 1$
- T järgib piiranguid, mis põhinevad ühe testi funktsiooni väljakuutsete seas esineval suurimal N väärtusel (tähistatud kui N_{max}):
 - kui $N_{max} \leq 9$, siis $1 \leq T \leq 100$
 - kui $N_{max} \leq 2^{10} + 1$, siis $1 \leq T \leq 10$
 - kui $N_{max} \leq 2^{16} + 1$, siis $1 \leq T \leq 3$

Hindaja võib kasutada kuni 280 MiB, mis läheb arvesse sinu lahenduse mälu hulka.

Näide

Oletame, et kiirete ühistranspordiliinide varjatud struktuur on järgmine:



Oletame, et alguskoha 0 korral on jalutuskäik $[0, 1, 2, 4, 3, 5]$. Siis on järgnev näide suhtlusest:

Võistleja programm	Žürii programm
	<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>	

Pane tähele, et päringutest saadud teave ei ole piisav, et puud üheselt kindlaks määrata, kuid see on tegelikult vastus testile alamülesandes 0.

Alamülesanded

Alamülesanne	Punktid	N	Täiendavad piirangud
0	0	–	Näide.
1	11	≤ 9	–
2	6	≤ 100	Iga tipp on ühendatud kuni 2 teise tipuga.
3	13	≤ 100	Iga jalutuskäik on loodud sügavuti läbimise abil, mis eelistab igal sammul liikuda tipust 0 eemale. Kui on mitu võimalust tipust 0 eemale liikuda, on valitud neist üks suvaliselt.
4	11	≤ 100	Iga tipp peale tipu 0 on ühendatud kuni 2 teise tipuga.
5	10	≤ 100	–
6	31	$\leq 2^{10} + 1$	–
7	18	$\leq 2^{16} + 1$	–

Hindamine

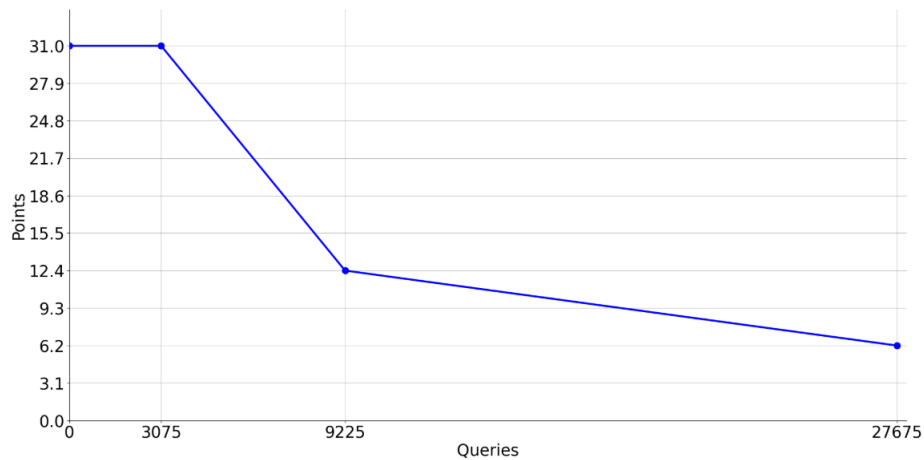
Iga alamülesande puhul kuni alamülesandeni 5 saad täispunktid, kui leiad puu edukalt ajapiirangute raames. Alamülesannete 6 ja 7 puhul sõltub antud testi osakaal S täispunktidest päringute **maksimaalsest** arvust, mille esitad selle ühel alamtestil, Q_{max} . Nende alamülesannete punktiarvestus on järgmine:

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

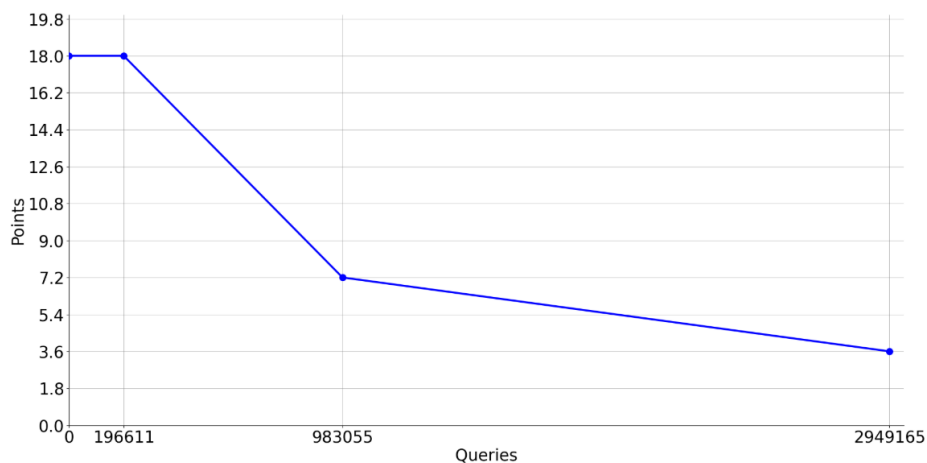
kus $L_1 = 3 \times (2^{10} + 1) = 3075$ ja $L_2 = 9 \times (2^{10} + 1) = 9225$ alamülesande 6 jaoks, ja $L_1 = 3 \times (2^{16} + 1) = 196611$ ja $L_2 = 15 \times (2^{16} + 1) = 983055$ alamülesande 7 jaoks. Osakaal täispunktidest, mille sa kogu alamülesande eest saad, on väikseim S kõikide testide seas.

Alltoodud graafikud näitavad kahe viimase alamülesande hindamisfunktsiooni:

Alamülesanne 6



Alamülesanne 7



Näidishindaja

On antud kaks näidishindajat.

Kohalikuks testimiseks: `Lgrader.cpp` saab kompileerida koos sinu programmiga, et pakkuda testimisvahendit. Programm loeb testide arvu T ja iga testi puhul ootab asukohtade arvu N , millele järgneb $N - 1$ rida, igal real täisarvude paar, mis kirjeldab otseühendust. Seejärel järgneb N rida, igal real N täisarvu – DFS-jalutuskäigud. Pane tähele, et jalutuskäik i peab algama tipust i . Sa võid lasta hindajal jalutuskäigud sinu eest genereerida, seades konstandi `AUTO_GENERATE` väärtuseks `true`. Väljundina hindaja kas teatab veateate, kui tulemus on vale, või iga testi kohta esitatud päringute arvu ja üldise maksimumi. Pane samuti tähele, et hindaja ei tööta piisavalt suurte N väärtuste puhul (täpsemalt alamülesande 7 piirangutega), selle juhu jaoks võid vabalt oma kujutlusvõimet kasutada.

Süsteemi testide jaoks: `stub.cpp` saab kasutada koos teie programmiga süsteemi testide jaoks. Sisendi formaat on sama, mis teisel hindajal. Pange tähele, et nende testide jaoks ei ole

sisseehitatud võimalust jalutuskäigu automaatseks genereerimiseks.