Reconstruct

Time Limit: 4 seconds
Memory Limit: 1024 MiB

Այս խնդիրը ինտերակտիվ է

Մամիկոնը պատրաստվում է մասնակցել կամավորների կողմից կազմակերպված ուղղորդման խաղին՝ *Վերջնակետին*, որում EJOI մասնակից թիմերը մրցելու են քաղաքում տարբեր տեսարժան վայրեր այցելելու գործում: Մամիկոնին այդքան էլ չի հետաքրքրում հաղթանակը: Նրա նպատակն է հասկանալ՝ ինչպիսին է խաղի կառուցվածքը:

Քաղաքում կան N տեսարժան վայրեր և N թիմեր. Յուրաքանչյուր թիմ պետք է այցելի բոլոր տեսարժան վայրերը և ունենա իր սկզբնակետը. i թիմի սկզբնակետը i տեսարժան վայրն է: Յուրաքանչյուր թիմ ունի իր համար որոշված երթուղին:

Մամիկոնը ունի հավելյալ տեղեկություն. ուղիները հիմնված են արագընթաց հասարակական տրանսպորտի ճանապարհների գաղտնի սխեմայի վրա: Ավելի կոնկրետ՝ մրցույթի ժյուրին ընտրել է $N-1$ ճանապարհներ, որոնցից յուրաքանչյուրը միացնում է տեսարժան վայրերի որոշակի զույգ, այնպես, որ յուրաքանչյուր վայրից կարելի է գնալ ցանկացած ուրիշ վայր՝ օգտագործելով տրված ճանապարհները: Այսպիսի սխեման նաև անվանում են ծառ: Ապա, յուրաքանչյուր i սկզբնակետի համար ժյուրին գեներացրել է **կամայական** *DFS զբոսանք* (սահմանումը ստորև) և այդ զբոսանքը նշանակել i թիմի երթուղի:

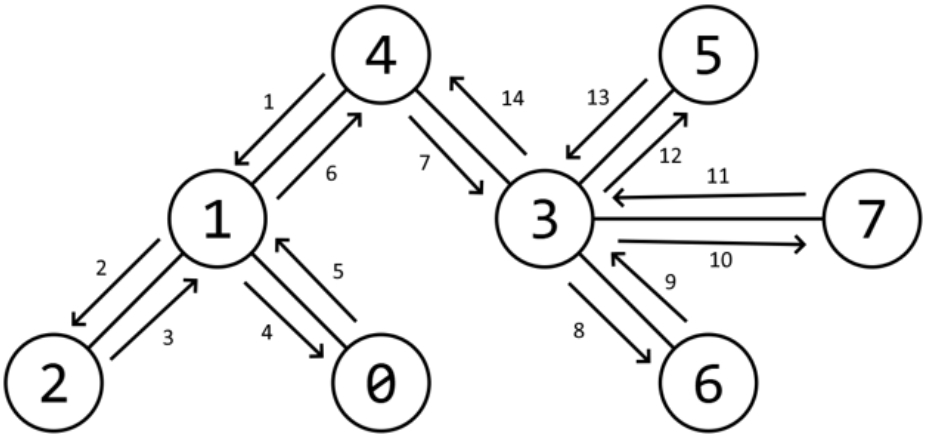
Մամիկոնը ցանկանում է գտնել ժյուրիի ընտրած ծառը: Նա կարող է տալ հետևյալ տեսքի հարցեր.

- ո՞րն է i թիմի այցելած j-րդ վայրը:

Օգնե՛ք Մամիկոնին գրել `find_tree` ծրագիրը, որը գտնում է ժյուրիի մտապահած ծառը:

> Ծառի *DFS զբոսանքը* այն հերթականությունն է, որով ծառի գագաթները այցելվում են առաջին անգամ *dfs (depth_first_search)* ընթացակարգով: Հետևյալ ընթացակարգը սկսում է տրված v գագաթից և ռեկուրսիվ ճանապարհորդում դեպի այդ գագաթի դեռ չայցելված հարևաններից որևէ մեկը, քանի դեռ այդպիսին գոյություն ունի: Ապա այն վերադառնում է

նախորդիվ այցելված գագաթը և շարունակում ընթացքը այդ գագաթից: Օրինակ՝



Այստեղ $v = 4$ և կողերի երկայնքով սլաքները համապատասխանում են dfs-ի քայլերին: Գեներացված զբոսանքն է `[4,1,2,0,3,6,7,5]`: Նկատենք, որ այն հերթականությունը որով այցելում ենք հարևաններին էական է. մեկ այլ զբոսանք կարող է լինել `[4,3,5,7,6,1,0,2]`-ը:

Ծառի կառուցվածքը և *DFS զբոսանքները* ֆիքսված են մինչև ծրագրի աշխատանքի սկիզբը և ձեր հարցերին ի պատասխան չեն փոխվում: Հարևանների հերթականությունը տարբեր *DFS զբոսանքների* համար կարող է տարբերվել:

Իրականացման մանրամասներ

Դուք պետք է իրականացնեք հետևյալ ֆունկցիան.

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

- այստեղ N-ը տեսարժան վայրերի քանակն է
- ֆունկցիան պետք է վերադարձնի ծառի կողերի ցուցակը. կողերի և նրանց ծայրակետերի հերթականությունը էական չէ:

Ամեն թեստի համար այս ֆունկցիան կարող է կանչվել առավելագույնը T անգամ:

Ժյուրիին հարց ուղղելու համար կարող եք կանչել հետևյալ ֆունկցիան:

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

Այն վերադարձնում է ծառի i զբոսանքի այցելած j-րդ գագաթը: Նշենք, որ `guess(i, 0)` միշտ վերադարձնում է i: Կարող եք ենթադրել, որ այս ֆունկցիան պատասխանում է հաստատուն $O(1)$ ժամանակում բոլոր ենթախնդիրներում բացի 7-րդից, որում այն աշխատում է լոգարիթմիկ $O(\log N)$ ժամանակում: Անհրաժեշտ է, որ տեղի ունենա

$0 \le i, j \le N-1$ անհավասարումը, այլապես ձեր լուծումը կստանա `Output isn't correct: Invalid call` արդյունքը:

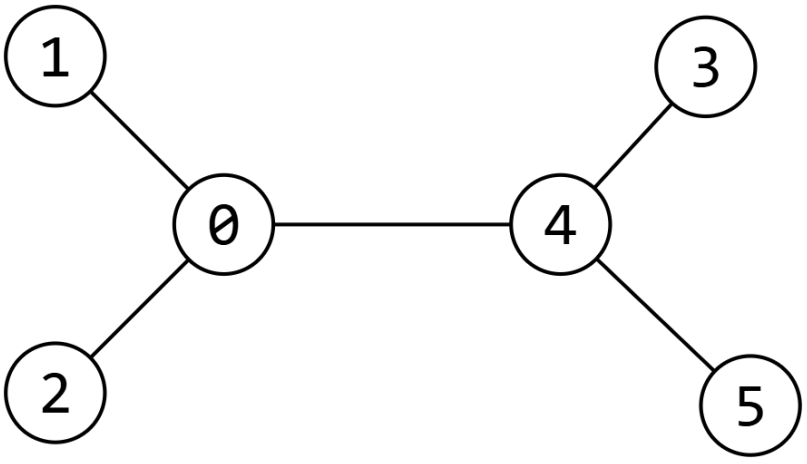
Սահմանափակումներ

- $2 \le N \le 2^{16}+1$
- T-ի (թեստերի քանակի) արժեքը կախված է մեծագույն N-ից (այսուհետ նշանակենք N_{max}) մեկ ենթախնդրի բոլոր կանչերում:
  - եթե $N_{max} \le 9$, ապա $1 \le T \le 100$
  - եթե $N_{max} \le 2^{10}+1$, ապա $1 \le T \le 10$
  - եթե $N_{max} \le 2^{16}+1$, ապա $1 \le T \le 3$

Համակարգի գրեյդերը կարող է օգտագործել մինչև 280 MiB հիշողություն, ինչը հաշվի է առնվելու ձեր լուծման ծախսած հիշողության մեջ:

Օրինակ

Ենթադրենք ծառի թաքնված կառուցվածքը ներկայացված է հետևյալ նկարում.



Դիցուք 0 սկզբնակետի զբոսանքն է $[0, 1, 2, 4, 3, 5]$: Ստորև ներկայացված է ժյուրիի հետ փոխազդեցության օրինակ.

Մասնակցի ծրագիր	Ժյուրիի ծրագիր
	`find_tree(6)`
`guess(0, 0)`	`0`
`guess(0, 1)`	`1`
`guess(0, 2)`	`2`
`guess(0, 3)`	`4`
`guess(0, 4)`	`3`
`guess(0, 5)`	`5`
`return {{0,1},{0,2},{4,0},{5,4},{3,4}};`	

Նկատենք, որ հարցումներից ստացված տեղեկությունը բավարար չէ ծառի կառուցվածքը միարժեքորեն վերականգնելու համար, սակայն մասնակիցը փաստացի վերադարձրեց 0-րդ ենթախնդրի թեստի պատասխանը:

Ենթախնդիրներ

Ենթախնդիր	Միավորներ	N	Լրացուցիչ սահմանափակումներ
0	0	-	Օրինակներ:
1	11	≤ 9	-
2	6	≤ 100	Յուրաքանչյուր գագաթ ունի առավելագույնը 2 հարևան:
3	13	≤ 100	Ամեն զբոսանք գեներացված է այնպիսի dfs-ով, որը ամեն քայլի նախընտրում է հեռվանալ 0 համարով գագաթից: Եթե կան այդպիսի մի քանի գագաթներ, նրանցից մեկը ընտրվում է կամայականորեն:
4	11	≤ 100	Յուրաքանչյուր գագաթ բացի 0 համարով գագաթից ունի առավելագույնը 2 հարևան:
5	10	≤ 100	-
6	31	$\leq 2^{10}+1$	-
7	18	$\leq 2^{16}+1$	-

Գնահատում

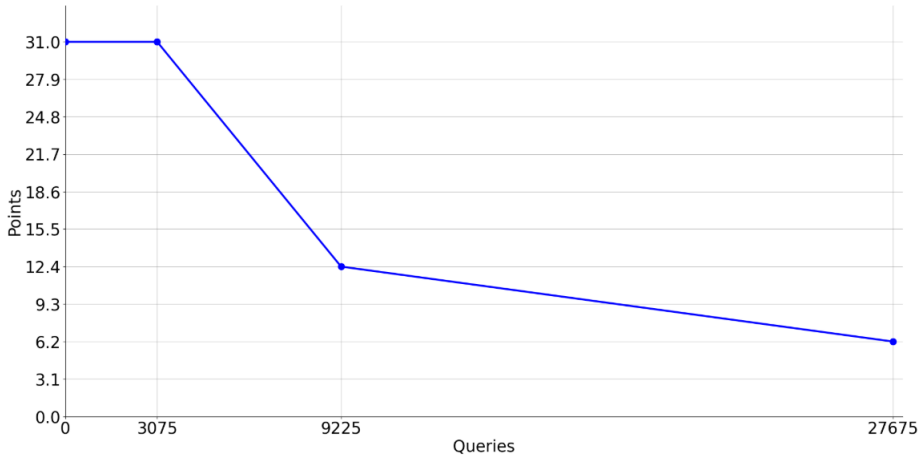
1-5 ենթախնդիրների համար դուք կստանաք հնարավոր առավելագույն միավորը, եթե հաջողությամբ գտնեք ծառը՝ տեղավորվելով ժամանակային սահմանափակման մեջ: 6-րդ և 7-րդ ենթախնդիրների համար առավելագույն միավորի ձեր վաստակած S մասը կախված է ենթախնդրում բոլոր թեստերում կատարված հարցումների քանակների **մաքսիմումից** (նշանակենք Q_{max}): Այդ ենթանխնդիրների գնահատումը այսպիսին է.

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

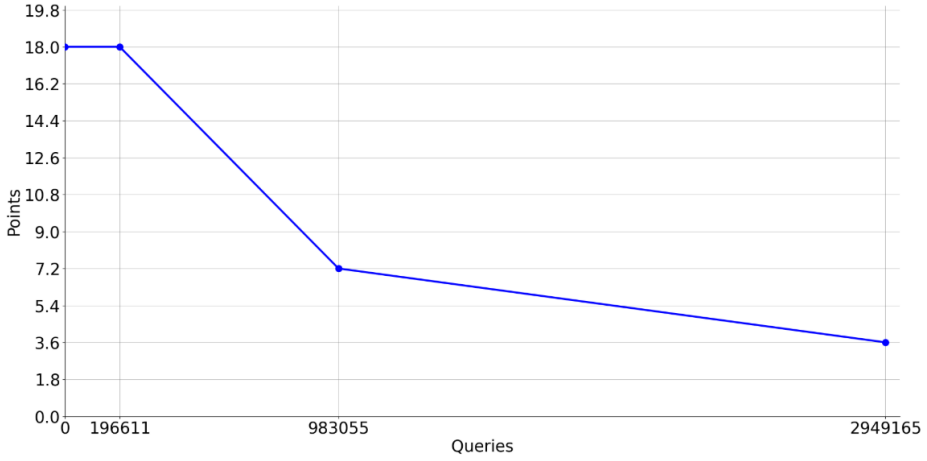
որտեղ $L_1 = 3 \times (2^{10} + 1) = 3075$ և $L_2 = 9 \times (2^{10} + 1) = 9225$ 6-րդ ենթախնդրում, և $L_1 = 3 \times (2^{16} + 1) = 196611$ և $L_2 = 15 \times (2^{16} + 1) = 983055$ 7-րդ ենթախնդրում: Ամբողջական միավորի ձեր վաստակած մասը բոլոր թեստերի S-երից նվազագույնն է:

Ստորև ներկայացված գրաֆիկը ցույց է տալիս վերջին 2 ենթախնդիրների գնահատումը.

Ենթախնդիր 6



Ենթախնդիր 7



Գրեյդերի նմուշ

Ձեզ տրված են երկու գրեյդերի նմուշներ:

Լոկալ ստուգման համար. `Lgrader.cpp` կարող է կոմպիլացվել ձեր ծրագրի հետ, որպեսզի տրամադրի ստուգման համար գործիք: Ծրագիրը կարդում է T մուտքային տվյալների քանակը, ապա տեսարժան վայրերի քանակ N-ը, ապա $N-1$ ամբողջ թվերի զույգեր, որոնք ինչ-որ կողի ծայրակետեր են: Դրանից հետո գրեյդերի նմուշը կարդում է N հատ N երկարության հերթականություններ՝ *DFS զբոսանքները*: Նկատենք, որ i-րդ զբոսանքը պետք է սկսվի i գագաթից: Դուք ունեք հնարավորություն թողնելու գրեյդերի նմուշին գեներացնել զբոսանքները ձեր փոխարեն՝ `AUTO_GENERATE` հաստատունը `true` սահմանելով: Գրեյդերի նմուշը, որպես ելքագրում, տեղեկացնում է սխալի մասին, եթե ծրագրի արդյունքը սխալ է, կամ տպում է յուրաքանչյուր թեստում օգտագործված հարցումների քանակները և դրանց մաքսիմումը: Նաև նկատենք, որ գրեյդերի նմուշը չի աշխատում բավականաչափ մեծ N-երի համար (ավելի կոնկրետ 7 -րդ ենթախնդրի սահմանափակումների համար): Կարող եք օգտագործել ձեր երևակայությունը այդ ենթախնդրի համար:

Համակարգի օգտատերերի ստուգման համար: `stub.cpp` կարող է օգտագործվել ձեր ծրագրի հետ՝ օգտատերերի ստուգումները իրականացնելու համար: Մուտքային տվյալների ֆորմատը նույնն է ինչ նախորդ նմուշում: Նկատենք, որ այս նմուշը չունի զբոսանքները ինքնուրույն գեներացնելու հնարավորություն: