Възстановяване

Време: 4 секунди
Памет: 1024 MiB

Това е интерактивна задача.

Биси ще се включи в доброволческата игра за ориентиране *Последен изход*, където EJOI делегациите ще се състезават да посещават различни локации в града. Но Биси е по-заинтересована да разбере *как* е организирана играта, а не да я спечели.

На картата има N локации и играта ще се играе от N отбора. Всеки отбор трябва да мине през всички локации и има уникална начална позиция — отбор i започва маршрута си от локация i. Всеки отбор има свой маршрут, по който ще посещава локациите.

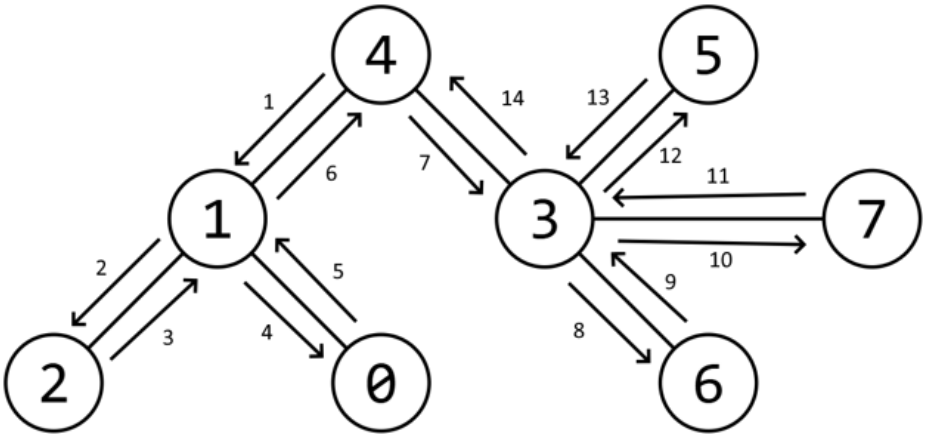
Допълнително, Биси има вътрешна информация — маршрутите са базирани на скритата структура на мрежата на градския транспорт. По-точно, журито на състезанието е определило $N-1$ връзки между двойки локации, такива че всяка локация е достижима от всяка друга, ползвайки тези връзки. Такава структура може да познавате като дърво. За всяка начална позиция i, журито е генерирало **произволно** *DFS обхождане* (дефинирано по-долу) и е възложило на отбор i да го следва за маршрут.

Биси иска да установи дървото, ползвано от журито. Тя може да задава въпроси от следния тип:

- Каква е j-тата дестинация, посетена от i-тия отбор?

Помогнете на Биси, като напишете програма `find_tree`, която намира скритото дърво.

> *DFS обхождане* на дърво е редът, в който върховете на дървото са посетени за пръв път от обхождане в дълбочина. Обхождане в дълбочина на дърво започва във връх v и рекурсивно посещава някой от непосетените му съседи, докато съществува такъв съсед. Когато всички съседи са посетени, то се връща към предходно посетения връх и продължава от там. Пример:



Тук $v = 4$ и стрелките по връзките съответстват на стъпките на обхождане в дълбочина. Генерираният маршрут е `[4,1,2,0,3,6,7,5]`. Забележете, че реда на посещаване има значение: друг възможен маршрут е `[4,3,5,7,6,1,0,2]`.

Структурата на дървото и маршрутите са фиксирани преди началото на програмата и не се променят в зависимост от зададените въпроси. Списъците от съседи на върховете, използвани по време на обхожданията в дълбочина могат да се различават.

Детайли по имплементацията

Трябва да имплементирате следната функция:

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

- параметърът N представлява броя на локациите;
- функцията трябва да връща списък от връзките в дървото — подредбата на ребрата и краищата им е без значение.

За всеки тест тази функция ще бъде извикана T пъти.

За да задавате въпроси на журито, може да извършвате извиквания към следната функция:

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

Функцията връща j-тата посетена локация в маршрута на i-тия отбор. Забележете, че `guess(i, 0)` ще върне i. Може да приемете, че функцията работи с константна сложност $O(1)$ за първите 6 подзадачи и логаритмична сложност $O(\log N)$ за подзадача 7. Трябва да е вярно, че $0 \leq i, j \leq N - 1$. Ако това не е вярно, ще получите резултат `Output isn't correct: Invalid call`.

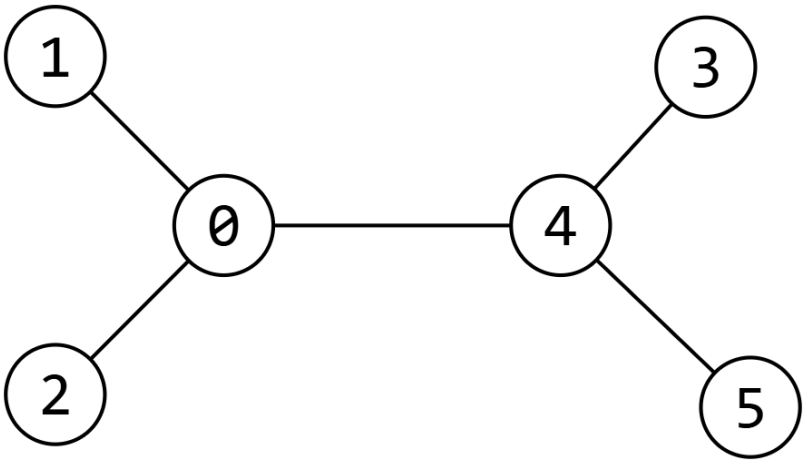
Ограничения

- $2 \leq N \leq 2^{16} + 1$
- Стойността на T следва следните ограничения в зависимост от максималната стойност на N (означена с N_{max}):
  - ако $N_{max} \leq 9$, тогава $1 \leq T \leq 100$
  - ако $N_{max} \leq 2^{10} + 1$, тогава $1 \leq T \leq 10$
  - ако $N_{max} \leq 2^{16} + 1$, тогава $1 \leq T \leq 3$

Системният грейдър може да използва до 280 MiB, които ще са включени в паметта, използвана от програмата Ви.

Пример

Нека скритата структура на градския транспорт е както следва:



Нека маршрутът на отбор 0 е $[0, 1, 2, 4, 3, 5]$. Примерна интеракция може да е следната:

Участник	**Жури**
	`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}};`	

Забележете, че информацията в заявките не е достатъчна, за да определи дървото еднозначно, все пак това е отговорът на теста в подзадача 0.

Подзадачи

Подзадача	Точки	N	Допълнителни ограничения
0	0	–	Пример.
1	11	≤ 9	–
2	6	≤ 100	Всеки връх е свързан с най-много 2 други.
3	13	≤ 100	Всеки маршрут е генериран чрез обхождане в дълбочина, което на всяка стъпка приоритизира да се движи по-далеч от връх 0. В случай, че има няколко опции, които го отдалечават от връх 0, една от тях се избира произволно.
4	11	≤ 100	Всеки връх освен връх 0 има най-много 2 съседа.
5	10	≤ 100	–
6	31	$\leq 2^{10}+1$	–
7	18	$\leq 2^{16}+1$	–

Оценяване

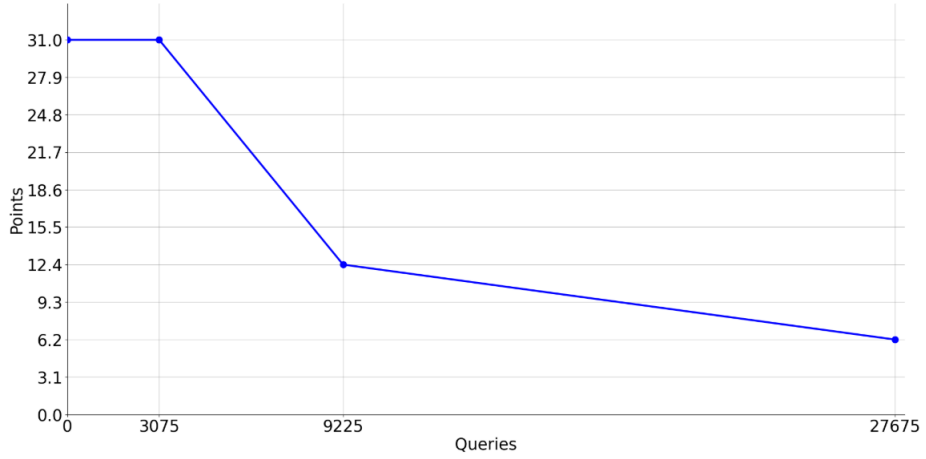
За първите пет подзадачи ще получите пълен брой точки за съответната подзадача, ако успешно възстановите дървото, спазвайки ограничението по време. За подзадачи 6 и 7, коефициентът, по който се умножава резултата Ви за един тест, S зависи от **максималния** брой въпроси, които задавате в негов подтест. Оценяването на подзадачите е както следва:

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

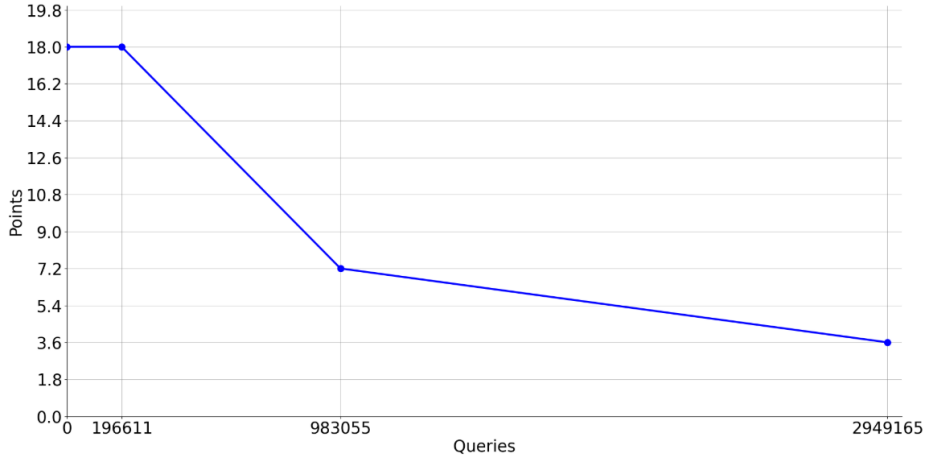
където $L_1 = 3 \times (2^{10}+1) = 3075$ и $L_2 = 9 \times (2^{10}+1) = 9225$ за подзадача 6 и $L_1 = 3 \times (2^{16}+1) = 196611$ и $L_2 = 15 \times (2^{16}+1) = 983055$ за подзадача 7. Резултатът, който ще получите за дадена подзадача, е произведението на предвидените точки за нея и минималната стойност на коефициента S, достигната сред всички тестове в подзадачата.

Графиките по-долу описват оценяването на последните две подзадачи:

Подзадача 6



Подзадача 7



Примерен грейдър

Предоставени са Ви два примерни грейдъра.

За локално тестване: Може да компилирате `Lgrader.cpp` с решението Ви, за да получите програма, с който да тествате решението си. Програмата първо чете броя на тестовете T и за всеки един от тях чете броя локации N, последвано от $N-1$ двойки, описващи връзка между две локации. След това се въвеждат N реда с по N числа всеки, които описват маршрутите на отборите. Забележете, че маршрут i започва с връх i. Може да оставите грейдъра да генерира маршрутите вместо вас, като зададете стойността на константата `AUTO_GENERATE` на `true`. Изходът на програмата ще бъде или съобщение за грешка, ако решението Ви е грешно, или броя на зададените въпроси за всеки подтест, както и максимумът сред тях. Забележете, че грейдърът не работи за достатъчно високи стойности на N (например тези на подзадача 7).

За системно тестване: `stub.cpp` може да бъде използван заедно с програмата Ви за потребителските тестове на системата. Форматът на входа е същият като на другия грейдър.

Забележете, че тук липсва вградената опция за автоматично генериране на маршрут.