

Реконструкція

Обмеження часу: 4 секунди

Обмеження пам'яті: 1024 MiB

Це інтерактивна задача.

Степан збирається взяти участь в організованій волонтерами гри з орієнтування *Пункт призначення*, у якій делегації EJOI змагатимуться, відвідуючи різні місця по всьому місту. Але Степана більше цікавить те, як була розроблена гра, ніж перемога в ній.

На карті є N місць і N команд. Кожна команда має відвідати всі місця, а початкові місця команд мають бути різними — команда i починає з місця i . Кожній команді надається власний маршрут.

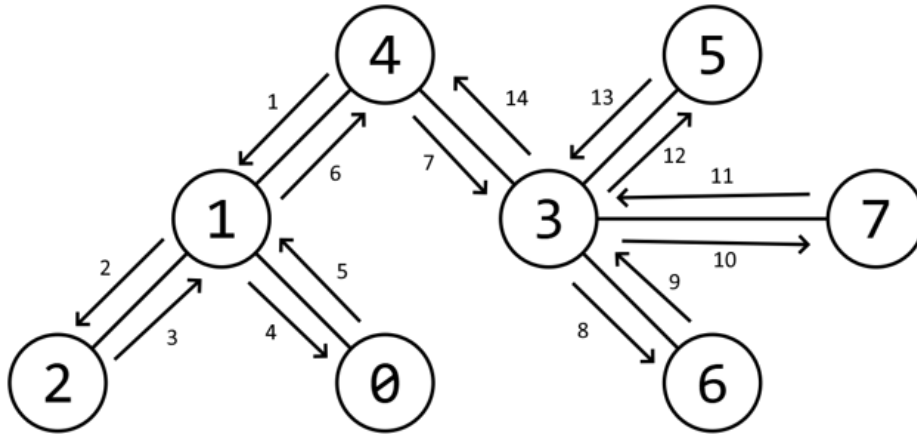
Степан також має деяку інсайдерську інформацію — маршрути побудовані на основі прихованої структури швидкісних ліній громадського транспорту. Точніше, журі змагання обрало $N - 1$ ліній, кожна з яких сполучає пару місць, так, що з будь-якого місця можна дістатися до будь-якого іншого, використовуючи ці лінії. Така структура також називається деревом. Потім для кожного початкового місця i журі згенерувало **довільний** DFS-обхід (визначений нижче) і призначило цей обхід маршрутом команди i .

Степан хоче визначити дерево громадського транспорту, яке обрало журі. Він ставить лише короткі запитання такого типу:

- Яким є j -те місце призначення i -ї команди?

Допоможіть Степану, написавши програму `find_tree`, яка знаходить приховане дерево.

DFS-обхід дерева — це порядок, у якому його вершини вперше відвідуються процедурою пошуку в глибину. Така процедура починається в заданій вершині v і рекурсивно переходить до одного з її невідвіданих сусідів, доки таких сусідів не залишиться — після цього вона повертається до попередньої відвіданої вершини та продовжує звідти. Приклад:



Тут $v = 4$, а стрілки вздовж ребер відповідають крокам пошуку в глибину. Отриманий обхід має вигляд $[4, 1, 2, 0, 3, 6, 7, 5]$. Зверніть увагу, що порядок відвідування сусідів має значення; інший обхід міг би мати вигляд $[4, 3, 5, 7, 6, 1, 0, 2]$.

Структура дерева та N DFS-обходів фіксуються до початку роботи вашої програми й не змінюються у відповідь на ваші запитання. Порядки сусідів вершин, використані під час таких DFS, можуть бути різними.

Деталі реалізації

Ви маєте реалізувати таку функцію:

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

- тут N — кількість місць;
- значення, що повертається, є списком ребер дерева — порядок ребер і порядок їхніх кінців не мають значення.

Для кожного тесту ця функція може бути викликана до T разів.

Для взаємодії з журі ви можете викликати таку функцію:

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

Вона повертає j -те місце в i -му обході дерева. Зверніть увагу, що `guess(i, 0)` поверне i . Ви можете вважати, що функція працює за сталий час $O(1)$ для всіх підзадач до 6 включно та за логарифмічний час $O(\log N)$ для підзадачі 7. Має виконуватися $0 \leq i, j \leq N - 1$, інакше ваше рішення отримає вердикт `Output isn't correct: Invalid call`.

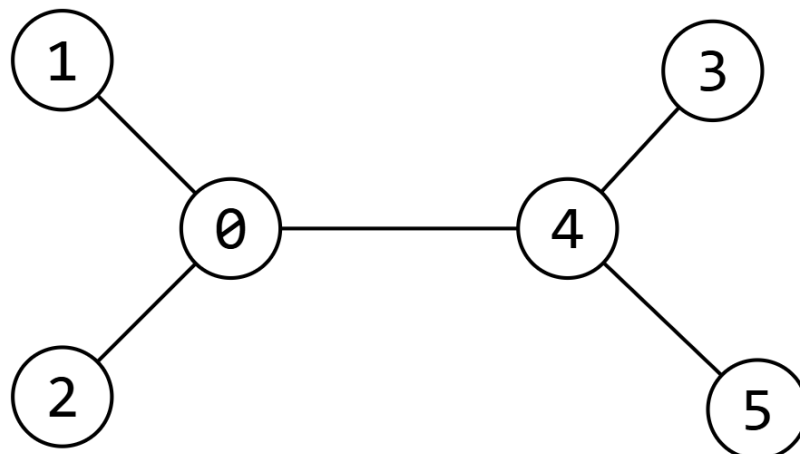
Обмеження

- $2 \leq N \leq 2^{16} + 1$
- T задовольняє обмеження, що залежать від максимального N (позначеного як N_{max}) серед викликів функції в одному тесті:
 - якщо $N_{max} \leq 9$, то $1 \leq T \leq 100$
 - якщо $N_{max} \leq 2^{10} + 1$, то $1 \leq T \leq 10$
 - якщо $N_{max} \leq 2^{16} + 1$, то $1 \leq T \leq 3$

Системний градер може використовувати до 280 МБ, які враховуються в обмеженні пам'яті вашого рішення.

Приклад

Припустімо, що прихована структура швидкісних ліній громадського транспорту має такий вигляд:



Нехай для початкового місця 0 обхід має вигляд $[0, 1, 2, 4, 3, 5]$. Тоді нижче наведено приклад взаємодії:

Програма учасника	Програма журі
	<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>	

Зверніть увагу, що інформації із запитів недостатньо, щоб однозначно визначити дерево, але насправді саме це є відповіддю для тесту з підзадачі 0.

Підзадачі

Підзадача	Бали	N	Додаткові обмеження
0	0	–	Приклад.
1	11	≤ 9	–
2	6	≤ 100	Кожна вершина сполучена не більш ніж із 2 іншими.
3	13	≤ 100	Кожен обхід генерується пошуком у глибину, який на кожному кроці надає перевагу руху від вершини 0. Якщо є декілька можливостей рухатися від вершини 0, одна з них вибирається довільно.
4	11	≤ 100	Будь-яка вершина, крім 0, сполучена не більш ніж із 2 іншими.
5	10	≤ 100	–
6	31	$\leq 2^{10} + 1$	–
7	18	$\leq 2^{16} + 1$	–

Оцінювання

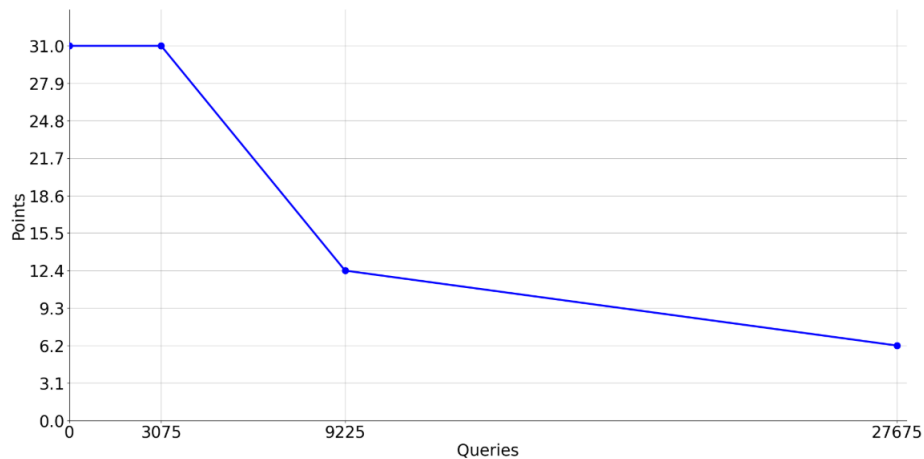
Для кожної підзадачі до підзадачі 5 включно ви отримуєте повну кількість балів, якщо успішно знаходите дерево в межах обмеження часу. Для підзадач 6 і 7 частка повної кількості балів S за певний тест залежить від **максимальної** кількості запитів, зроблених вами в одному з його підтестів, Q_{max} . Оцінювання для цих підзадач відбувається так:

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

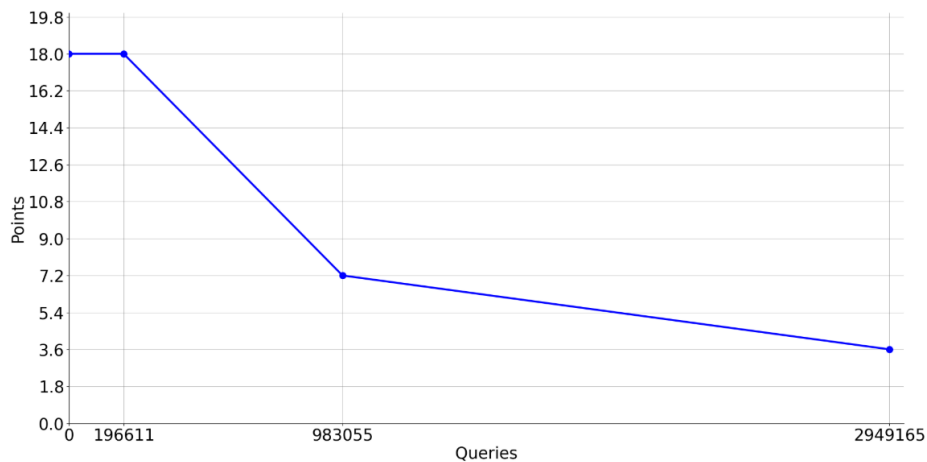
де $L_1 = 3 \times (2^{10} + 1) = 3075$ і $L_2 = 9 \times (2^{10} + 1) = 9225$ для підзадачі 6, а $L_1 = 3 \times (2^{16} + 1) = 196611$ і $L_2 = 15 \times (2^{16} + 1) = 983055$ для підзадачі 7. Частка балів, яку ви отримуєте за всю підзадачу, дорівнює мінімальному значенню S серед усіх тестів.

На графіках нижче зображено функцію оцінювання для останніх двох підзадач:

Підзадача 6



Підзадача 7



Приклад градера

Надаються два приклади градерів.

Для локального тестування: `Lgrader.cpp` можна скопіювати разом із вашою програмою, щоб отримати інструмент для тестування. Програма зчитує кількість T тестів, а для кожного з них очікує кількість місць N , після чого $N - 1$ рядків, кожен із парою цілих чисел, що

описують пряму лінію. Після цього йдуть N рядків по N цілих чисел – DFS-обходи. Зверніть увагу, що обхід i має починатися з вершини i . Ви можете дозволити градеру згенерувати обходи, встановивши константу `AUTO_GENERATE` у значення `true`. У відповідь градер або повідомляє про помилку, якщо результат неправильний, або виводить кількість запитів для кожного тесту та загальний максимум. Також зверніть увагу, що градер не працює для достатньо великих N (точніше, для обмежень підзадачі 7), у цьому випадку ви можете скористатися власною уявою.

Для системних користувацьких тестів: `stub.cpp` можна використовувати разом із вашою програмою для користувацьких тестів у системі. Формат вхідних даних такий самий, як і для іншого градера. Зверніть увагу, що для користувацьких тестів немає вбудованої можливості автоматично генерувати обходи.