

Reconstruct

Time limit: 4 seconds

Memory limit: 1024 MiB

Это интерактивная задача.

Бисси собирается принять участие в организованной волонтерами игре по ориентированию *Финальное назначение*, в которой делегации EJOI будут соревноваться, посещая различные места по всему городу. Но Бисси больше интересует то, *как* эта игра была придумана, чем победа в ней.

На карте есть N мест и N команд. Каждая команда должна посетить все места и имеет уникальное начальное место — команда i начинает маршрут в месте i . Каждой команде выдаётся свой маршрут.

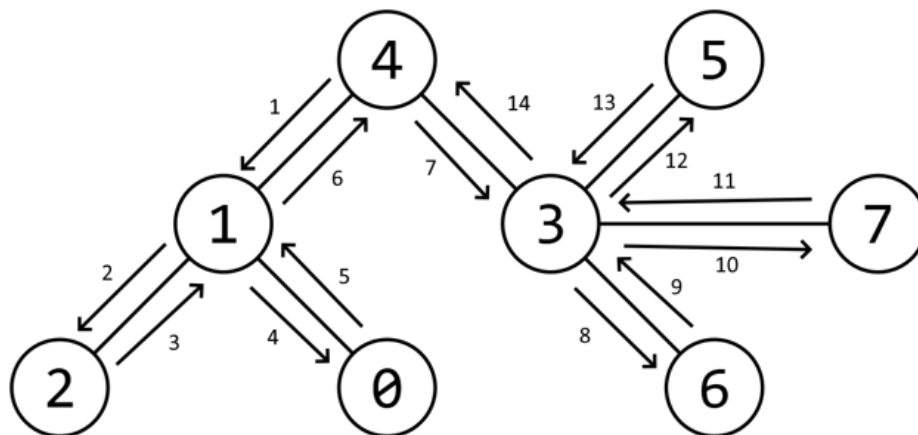
Бисси также владеет инсайдерской информацией — маршруты основаны на скрытой структуре линий быстрого общественного транспорта. Точнее, жюри соревнования выбрало $N - 1$ линий, каждая из которых соединяет пару мест, так что из любого места можно добраться до любого другого места, используя эти линии. Такая структура также называется деревом. Затем для каждого начального места i жюри сгенерировало **произвольный** обход *в глубину (DFS)* (определённый ниже) и назначило этот обход в качестве маршрута для команды i .

Бисси хочет выяснить, какое дерево общественного транспорта выбрало жюри. Она задаёт только быстрые вопросы такого вида:

- Какое место находится на позиции с индексом j в маршруте команды i ?

Помогите Бисси, написав программу `find_tree`, которая находит скрытое дерево.

Обход в глубину (DFS) дерева — это порядок, в котором его вершины впервые посещаются процедурой поиска в глубину. Такая процедура начинается в заданной вершине v и рекурсивно переходит к одному из её непосещённых соседей, пока таких соседей не останется — тогда она возвращается к ранее посещённой вершине и продолжает оттуда. Пример:



Здесь $v = 4$, и стрелки вдоль рёбер соответствуют шагам поиска в глубину. Полученный обход имеет вид $[4, 1, 2, 0, 3, 6, 7, 5]$. Обратите внимание, что порядок, в котором мы посещаем соседей, имеет значение; другим обходом мог бы быть $[4, 3, 5, 7, 6, 1, 0, 2]$.

Структура дерева и N обходов DFS фиксируются до запуска вашей программы и не изменяются в ответ на ваши вопросы. Порядки просмотра соседей вершин в таких обходах DFS могут различаться.

Implementation details

Вам нужно реализовать следующую функцию:

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

- здесь N — количество мест;
- возвращаемое значение — это список рёбер дерева; порядок рёбер и их концов не имеет значения.

Для каждого теста эта функция может быть вызвана до T раз.

Для взаимодействия с жюри вы можете вызывать следующую функцию:

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

Она возвращает j -ю вершину в i -м обходе дерева. Обратите внимание, что `guess(i, 0)` вернёт i . Вы можете считать, что функция выполняется за константное время $O(1)$ для всех подзадач до 6 включительно, и за логарифмическое время $O(\log N)$ для подзадачи 7. Должно выполняться условие $0 \leq i, j \leq N - 1$, иначе ваше решение получит вердикт `Output isn't correct: Invalid call`.

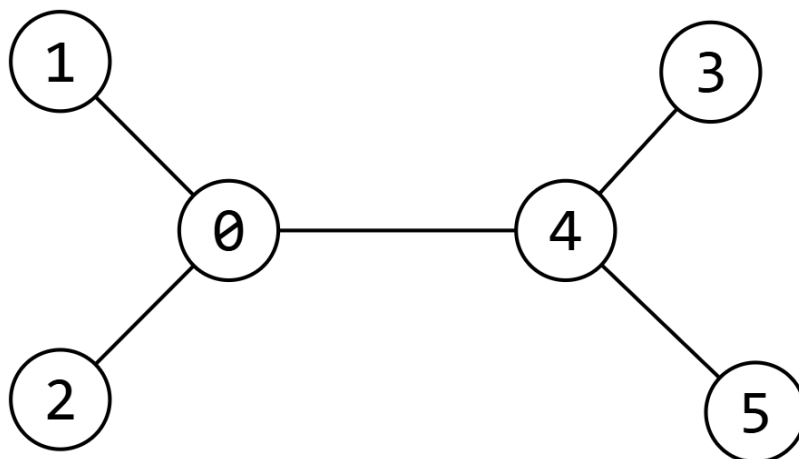
Constraints

- $2 \leq N \leq 2^{16} + 1$
- T удовлетворяет ограничениям, основанным на максимальном значении N (обозначаемом как N_{max}) среди вызовов функций в одном тесте:
 - если $N_{max} \leq 9$, то $1 \leq T \leq 100$
 - иначе если $N_{max} \leq 2^{10} + 1$, то $1 \leq T \leq 10$
 - иначе если $N_{max} \leq 2^{16} + 1$, то $1 \leq T \leq 3$

Системный грейдер может использовать до 280 МиБ, которые учитываются в ограничении памяти вашего решения.

Example

Предположим, что скрытая структура линий быстрого общественного транспорта следующая:



Предположим, что для начального места 0 обход равен $[0, 1, 2, 4, 3, 5]$. Тогда ниже приведён пример взаимодействия:

Программа участника	Программа жюри
	<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>	

Обратите внимание, что информации, полученной из запросов, недостаточно, чтобы однозначно определить дерево, однако это действительно правильный ответ для теста в подзадаче 0.

Subtasks

Подзадача	Баллы	N	Дополнительные ограничения
0	0	–	Пример
1	11	≤ 9	–
2	6	≤ 100	Каждая вершина соединена не более чем с 2 другими.
3	13	≤ 100	Каждый обход генерируется поиском в глубину, который на каждом шаге отдаёт приоритет отдалению от вершины 0. Если есть несколько вариантов, как отдаляться от вершины 0, один из них выбирается произвольно.
4	11	≤ 100	Любая вершина, кроме 0, соединена не более чем с 2 другими.
5	10	≤ 100	–
6	31	$\leq 2^{10} + 1$	–
7	18	$\leq 2^{16} + 1$	–

Scoring

Для каждой подзадачи до подзадачи 5 включительно вы получаете полное количество баллов, если успешно находите дерево с соблюдением ограничения по времени.

Для подзадач 6 и 7 доля S от полного количества баллов за данный тест зависит от **максимального** числа запросов Q_{max} , сделанных в одном из его подтестов.

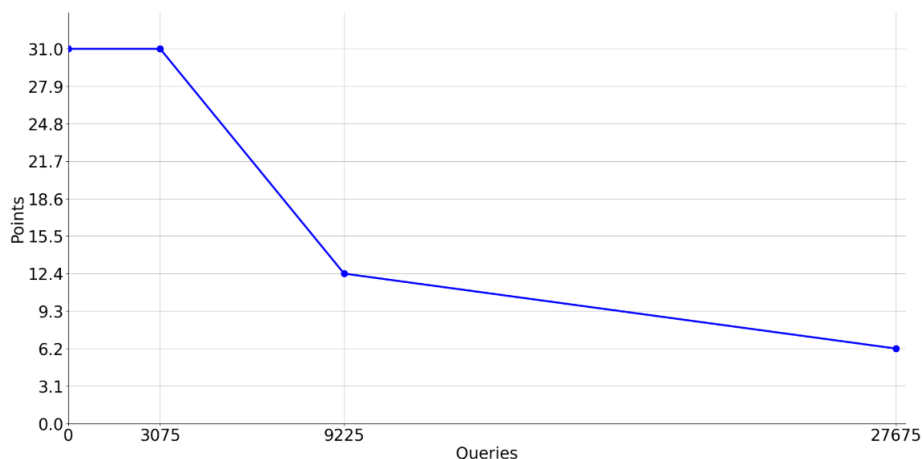
Начисление баллов для этих подзадач следующее:

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

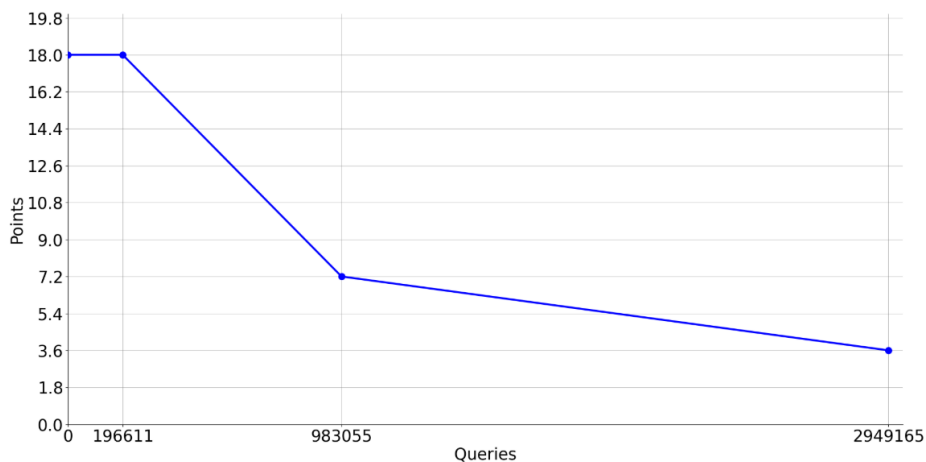
где $L_1 = 3 \times (2^{10} + 1) = 3075$ и $L_2 = 9 \times (2^{10} + 1) = 9225$ для подзадачи 6, и $L_1 = 3 \times (2^{16} + 1) = 196611$ и $L_2 = 15 \times (2^{16} + 1) = 983055$ для подзадачи 7. Доля баллов, которую вы получаете за всю подзадачу, равна минимальному значению S среди всех тестов этой подзадачи.

Графики ниже показывают функцию начисления баллов для последних двух подзадач:

Subtask 6



Subtask 7



Sample grader

Предоставляются два примера проверяющих программ (грейдеров).

Для локального тестирования: `Lgrader.cpp` можно скомпилировать вместе с вашей программой, чтобы получить инструмент для тестирования. Программа считывает число T тестовых случаев, и для каждого из них ожидает число мест N , за которым следуют $N - 1$ строк, каждая из которых содержит пару целых чисел, задающих линию между двумя местами. После этого следуют N строк по N целых чисел – обходы в глубину (DFS). Обратите внимание, что обход i должен начинаться с вершины i . Вы можете позволить грейдеру генерировать обходы за вас, установив константу `AUTO_GENERATE` в значение `true`. В качестве вывода грейдер либо сообщает об ошибке, если результат неверен, либо выводит число запросов, сделанных для каждого тестового случая, и общий максимум. Также обратите внимание, что грейдер не работает при достаточно больших N (точнее, при ограничениях подзадачи 7), для этого случая вы можете использовать своё воображение.

Для user test в CMS: `stub.cpp` можно использовать вместе с вашей программой для пользовательских тестов в системе CMS. Формат ввода такой же, как и у другого грейдера. Обратите внимание, что для user tests нет встроенной опции для автоматической генерации обходов.