Задача A. Общий предок

Имя входного файла:	`lca.in`
Имя выходного файла:	`lca.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Дано подвешенное дерево с корнем в 1-й вершине и M запросов вида «найти у двух вершин наименьшего общего предка».

Формат входного файла

В первой строке файла записано одно число N — количество вершин. В следующих $N-1$ строках записаны числа. Число x на строке $2 \le i \le N$ означает, что x — отец вершин i. ($x < i$). На следующей строке число M. Следующие M строк содержат запросы вида (x, y) — найти наименьшего предка вершин x и y Ограничения: $1 \le N \le 5 \cdot 10^4, 0 \le M \le 5 \cdot 10^4$.

Формат выходного файла

M ответов на запросы.

Пример

lca.in	lca.out
5 1 1 2 3 2 2 3 4 5	1 1

Задача B. LCA offline

Имя входного файла:	`lca.in`
Имя выходного файла:	`lca.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайт

Изначально имеется дерево состоящее только из корня (вершина с номером 1). Требуется научиться отвечать на следующие запросы:

- ADD a b — подвесить вершину b за вершину a (гарантируется, что вершина a уже существует).
- GET a b — вернуть LCA вершин a и b.

Все номера вершин от 1 до N.
В каждый момент времени у нас есть одно дерево.

Формат входного файла

В первой строке входного файла содержится число k — количество запросов. Следующие k строк содержат сами запросы. Гарантируется, что число запросов каждого из типов не превосходит 500 000.

Формат выходного файла

Для каждого запроса типа GET выведите в отдельную строку одно целое число — ответ на соответствующий запрос.

Примеры

lca.in	lca.out
9 ADD 1 2 ADD 1 3 ADD 2 4 GET 1 3 GET 2 3 GET 3 4 ADD 2 5 GET 4 5 GET 5 5	1 1 1 2 5

Задача C. LCA Problem Revisited

Имя входного файла:	`lca_rmq.in`
Имя выходного файла:	`lca_rmq.out`
Ограничение по времени:	4 секунды
Ограничение по памяти:	64 мегабайта

Задано подвешенное дерево, содержащее n ($1 \le n \le 100\,000$) вершин, пронумерованных от 0 до $n-1$. Требуется ответить на m ($1 \le m \le 10\,000\,000$) запросов о наименьшем общем предке для пары вершин.

Запросы генерируются следующим образом. Заданы числа a_1, a_2 и числа x, y и z. Числа $a_3, \ldots, a_{2m}$ генерируются следующим образом: $a_i = (x \cdot a_{i-2} + y \cdot a_{i-1} + z) \bmod n$. Первый запрос имеет вид $\langle a_1, a_2 \rangle$. Если ответ на $i-1$-й запрос равен v, то i-й запрос имеет вид $\langle (a_{2i-1} + v) \bmod n, a_{2i} \rangle$.

Формат входного файла

Первая строка содержит два числа: n и m. Корень дерева имеет номер 0. Вторая строка содержит $n-1$ целых чисел, i-е из этих чисел равно номеру родителя вершины i. Третья строка содержит два целых числа в диапазоне от 0 до $n-1$: a_1 и a_2. Четвертая строка содержит три целых числа: x, y и z, эти числа неотрицательны и не превосходят 10^9.

Формат выходного файла

Выведите в выходной файл сумму номеров вершин — ответов на все запросы.

Примеры

lca_rmq.in	lca_rmq.out
3 2 0 1 2 1 1 1 0	2

Задача D. LCA-3

Имя входного файла:	`lca3.in`
Имя выходного файла:	`lca3.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

> *Подвешенное дерево* — это ориентированный граф без циклов, в котором в каждую вершину, кроме одной, называемой *корнем* ориентированного дерева, входит одно ребро. В корень ориентированного дерева не входит ни одного ребра. *Отцом* вершины называется вершина, ребро из которой входит в данную.

(по материалам Wikipedia)

Дан набор подвешенных деревьев. Требуется выполнять следующие операции:

1. 0 u v Для двух заданных вершин u и v выяснить, лежат ли они в одном дереве. Если это так, вывести вершину, являющуюся их наименьшим общим предком, иначе вывести 0.
2. 1 u v Для корня u одного из деревьев и произвольной вершины v другого дерева добавить ребро (v, u). В результате эти два дерева соединятся в одно.

Вам необходимо выполнять все операции online, т.е. вы сможете узнать следующий запрос только выполнив предыдущий.

Формат входного файла

На первой строке входного файла находится число n — суммарное количество вершин в рассматриваемых деревьях, $1 \le n \le 50\,000$. На следующей строке расположено n чисел — предок каждой вершины в начальной конфигурации, или 0, если соответстующая вершина является корнем. Затем следует число k — количество запросов к вашей программе, $1 \le k \le 100\,000$. Каждая из следующих строк содержит по три целых числа: вид запроса (0 — для поиска LCA или 1 — для добавления ребра) и два числа x, y. Вершины, участвующие в запросе можно вычислить по формуле: $u = (x - 1 + ans) \bmod n + 1$, $v = (y - 1 + ans) \bmod n + 1$, где ans - ответ на последний запрос типа 0 ($ans = 0$ для первого запроса).

Формат выходного файла

Для каждого запроса типа 0, выведите в выходной файл одно число на отдельной строке — ответ за этот запрос.

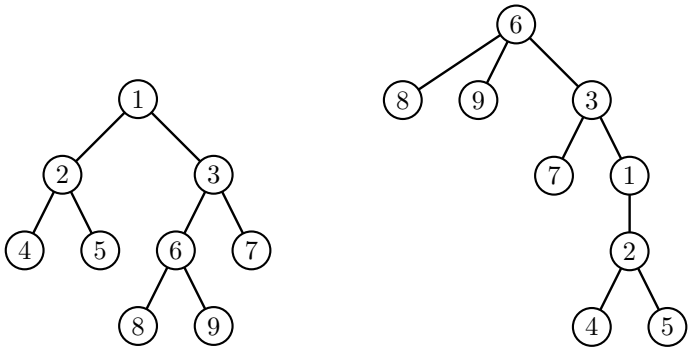
Примеры

lca3.in	lca3.out
5 0 0 0 0 0 12 1 5 3 0 2 5 1 4 2 1 1 5 0 1 5 1 3 4 0 1 5 0 3 1 0 4 2 0 1 4 0 5 2 0 4 1	0 5 5 3 2 3 3 2

Задача E. Dynamic LCA

Имя входного файла:	`dynamic.in`
Имя выходного файла:	`dynamic.out`
Ограничение по времени:	3 seconds
Ограничение по памяти:	256 megabytes

Наименьший общий предок в дереве для вершин u и v – вершина $lca(u, v)$, которая является предком и u, и v и при этом имеем максимальную глубину. Например, $lca(8, 7) = 3$ в дереве на картинке слева. В дереве можно поменять корень. $chroot(u)$ делает корнем вершину u. Например, после операции $chroot(6)$, как видно на картинке справа, $lca(8, 7) = 6$.



Вам дано дерево с корнем в вершине 1. Напишите программу, отвечающую на запросы $lca(u, v)$ и $chroot(u)$.

Формат входного файла

Во входных данных дано насколько тестов.

Каждый тест начинается с количества вершин в дереве n ($1 \le n \le 100\,000$). Следующие $n - 1$ строк содержат пары целых чисел от 1 до n – рёбра дерева. Далее число запросов m ($1 \le m \le 200\,000$). Каждая из следующих m строк имеет вид "`?` u v" для запроса $lca(u, v)$ или "`!` u" для запроса $chroot(u)$.

Последний тест имеет $n = 0$, его обрабатывать не нужно.

Сумма n по всем тестам не более 100 000. Сумма m по всем тестам не более 200 000.

Формат выходного файла

Для каждой операции вида "`?` u v" выведите $lca(u, v)$.

Примеры

dynamic.in	dynamic.out
9 1 2 1 3 2 4 2 5 3 6 3 7 6 8 6 9 10 ? 4 5 ? 5 6 ? 8 7 ! 6 ? 8 7 ? 4 5 ? 4 7 ? 5 9 ! 2 ? 4 3 0	2 1 3 6 2 3 6 2

Задача F. Самое дешевое ребро

Имя входного файла:	`minonpath.in`
Имя выходного файла:	`minonpath.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Дано подвешенное дерево с корнем в первой вершине. Все ребра имеют веса (стоимости). Вам нужно ответить на M запросов вида "найти у двух вершин минимум среди стоимостей ребер пути между ними".

Формат входного файла

В первой строке файла записано одно числ — n (количество вершин).

В следующих $n-1$ строках записаны два числа — x и y. Число x на строке i означает, что x — предок вершины i, y означает стоимость ребра.

$x < i, |y| \leq 10^6$.

Далее m запросов вида (x, y) — найти минимум на пути из x в y ($x \neq y$).

Ограничения: $2 \leq n \leq 5 \cdot 10^4, 0 \leq m \leq 5 \cdot 10^4$.

Формат выходного файла

m ответов на запросы.

Пример

minonpath.in	minonpath.out
5 1 2 1 3 2 5 3 2 2 2 3 4 5	2 2

Задача G. Опекуны карнотавров

Имя входного файла:	`carno.in`
Имя выходного файла:	`carno.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Карнотавры очень внимательно относятся к заботе о своем потомстве. У каждого динозавра обязательно есть старший динозавр, который его опекает. В случае, если опекуна съедают (к сожалению, в юрский период такое не было редкостью), забота о его подопечных ложится на плечи того, кто опекал съеденного динамозавра. Карнотавры — смертоносные хищники, поэтому их обычаи строго запрещают им драться между собой. Если у них возникает какой-то конфликт, то, чтобы решить его, они обращаются к кому-то из старших, которому доверяют, а доверяют они только тем, кто является их опекуном или опекуном их опекуна и так далее (назовем таких динозавров суперопекунами). Поэтому для того, чтобы решить спор двух карнотавров, нужно найти такого динозавра, который является суперопекуном для них обоих. Разумеется, беспокоить старших по пустякам не стоит, поэтому спорщики стараются найти самого младшего из динозавров, который удовлетворяет этому условию. Если у динозавра возник конфликт с его суперопекуном, то этот суперопекун сам решит проблему. Если у динозавра нелады с самим собой, он должен разобраться с этим самостоятельно, не беспокоя старших. Помогите динозаврам разрешить их споры.

Формат входного файла

Во входном файле записано число M, обозначающее количество запросов ($1 \le M \le 200\,000$). Далее на отдельных строках следуют M запросов, обозначающих следующие события:

- `+ v` — родился новый динозавр и опекунство над ним взял динозавр с номером v. Родившемуся динозавру нужно присвоить наименьший натуральный номер, который до этого еще никогда не встречался.
- `- v` — динозавра номер v съели.
- `? u v` — у динозавров с номерами u и v возник конфликт и вам надо найти им третейского судью.

Изначально есть один прадинозавр номер 1; гарантируется, что он никогда не будет съеден.

Формат выходного файла

Для каждого запроса типа «?» в выходной файл нужно вывести на отдельной строке одно число — номер самого молодого динозавра, который может выступить в роли третейского судьи.

Примеры

carno.in	carno.out
11 + 1 + 1 + 2 ? 2 3 ? 1 3 ? 2 4 + 4 + 4 - 4 ? 5 6 ? 5 5	1 1 2 2 5

Задача H. Ancestor. Предок

Имя входного файла:	`ancestor.in`
Имя выходного файла:	`ancestor.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Напишите программу, которая для двух вершин дерева определяет, является ли одна из них предком другой.

Формат входного файла

Первая строка входного файла содержит натуральное число n ($1 \le n \le 100\,000$) — количество вершин в дереве. Во второй строке находится n чисел. При этом i-ое число второй строки определяет непосредственного родителя вершины с номером i. Если номер родителя равен нулю, то вершина является корнем дерева.

В третьей строке находится число m ($1 \le m \le 100\,000$) — количество запросов. Каждая из следующих m строк содержит два различных числа a и b ($1 \le a, b \le n$).

Формат выходного файла

Для каждого из m запросов выведите на отдельной строке число `1`, если вершина a является одним из предков вершины b, и `0` в противном случае.

Пример

ancestor.in	ancestor.out
6 0 1 1 2 3 3 5 4 1 1 4 3 6 2 6 6 5	0 1 1 0 0

Задача I. Дерево

Имя входного файла:	`tree.in`
Имя выходного файла:	`tree.out`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Дано взвешенное дерево. Найти кратчайшее расстояние между заданными вершинами.

Формат входного файла

Первая строка входного файла содержит натуральное число $N \leq 150\,000$ — количество вершин в графе. Вершины нумеруются целыми числами от 0 до $N-1$. В следующих $N-1$ строках содержится по три числа u, v, w, которые соответствуют ребру весом w, соединяющему вершины u и v. Веса — целые числа от 0 до 10^9. В следующей строке содержится натуральное число $M \leq 75\,000$ — количество запросов. В следующих M строках содержится по два числа u, v — номера вершин, расстояние между которыми необходимо вычислить.

Формат выходного файла

Для каждого запроса выведите на отдельной строке одно число — искомое расстояние. Гарантируется, что ответ помещается в знаковом 32-битном целом типе.

Пример

tree.in	tree.out
3 1 0 1 2 0 1 9 0 0 0 1 0 2 1 0 1 1 1 2 2 0 2 1 2 2	0 1 1 1 0 2 1 2 0

Задача J. Черепахи и повороты

Имя входного файла:	`turtles.in`
Имя выходного файла:	`turtles.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Для тренировки боевых черепах военные построили прямоугольный полигон размером $W \times H$ клеток. Некоторые клетки проходимы для черепах, а некоторые — нет. Черепахи могут перемещаться только параллельно сторонам полигона. Полигон сконструирован таким образом, что существует единственный способ добраться от любой проходимой клетки до любой другой проходимой клетки, не проходя при этом по одной и той же клетке дважды. Известно, что черепахи очень быстро бегают по прямой, но испытывают трудности при повороте на 90 градусов. Поэтому сложность маршрута определяется как количество поворотов, которое придётся совершить черепахе при переходе от начальной до конечной клетки маршрута. Вы должны написать программу, вычисляющую сложность маршрута по его начальной и конечной клетке.

Формат входного файла

В первой строке через пробел записаны два целых числа H и W — размеры полигона ($1 \leqslant W \cdot H \leqslant 100\,000$). Далее задаётся карта полигона — H строк по W символов в каждой. Символ '#' обозначает проходимую клетку, а '.' — непроходимую. В $H+2$-й строке записано целое число Q — количество маршрутов, для которых нужно посчитать сложность ($1 \leqslant Q \leqslant 50\,000$). В каждой из следующих Q строк через пробел записаны четыре целых числа: номер строки и номер столбца начальной клетки маршрута, номер строки и номер столбца конечной клетки маршрута. Гарантируется, что начальная и конечная клетки маршрута являются проходимыми. Строки занумерованы числами от 1 до H сверху вниз, а столбцы — числами от 1 до W слева направо.

Формат выходного файла

Для каждого маршрута выведите в отдельной строке одно число — его сложность.

Примеры

turtles.in	turtles.out
5 4 .#.. ###. ..## .##. 4 1 2 2 1 2 3 4 3 4 2 3 4 1 2 4 2	1 0 2 3