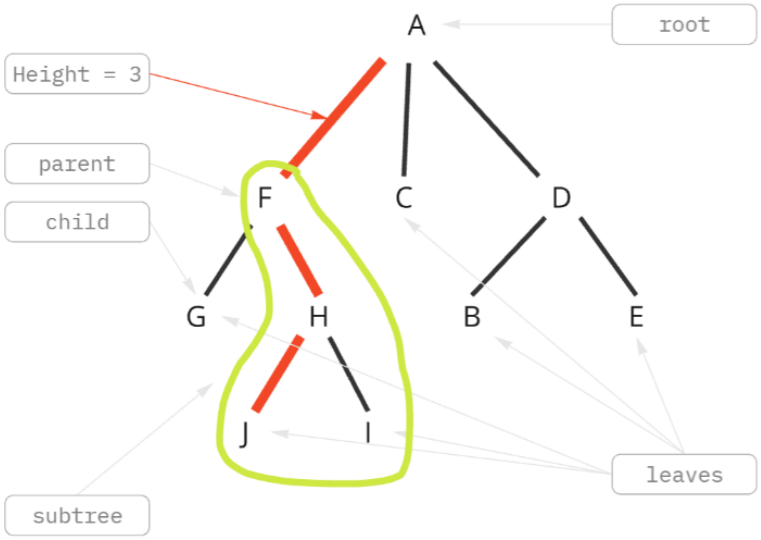
Двоичные деревья и двоичные деревья поиска

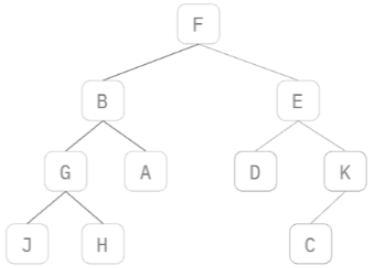
Терминология



Обходы двоичных деревьев

Pre-Order	In-Order	Post-Order

`func Traverse(node):`
`  process(node)`
`  Traverse(node->left)`
`  Traverse(node->right)`
	`func Traverse(node):`
	`  Traverse(node->left)`
	`  process(node)`
	`  Traverse(node->right)`
		`func Traverse(node):`
		`  Traverse(node->left)`
		`  Traverse(node->right)`
		`  process(node)`



Pre-Order	F B G J H A E D K C root left right
In-Order	J G H B A F D E C K left root right
Post-Order	J H G A B D K C E F left right root

A. *Создание двоичного дерева поиска*

Создать двоичное дерево поиска по ключам A_i и вывести высоты элементов, упорядоченных по значению.

В первой строке входных данных записано число N ($N < 10^4$) — количество вершин дерева. Во второй строке указаны N натуральных различных чисел — ключи, по которым надо создать двоичное дерево поиска.

Выведите N пар целых чисел — ключ вершины и её высоту (длину пути до корня). Пары должны быть упорядочены по значению ключа. В решении запрещается использовать сортировку и любые иные структуры данных, кроме тех, которые вы используете для хранения дерева.

Input	Output
7 4 2 3 7 5 9 8	2 1 3 2 4 0 5 2 7 1 8 3 9 2

B. *Количество вершин в поддереве*

Даны натуральные числа A_i — ключи ДДП в порядке их добавления в дерево и перечень некоторых из них V_k.

Требуется создать двоичное дерево поиска и для каждого из указанных ключей V_k найти соответствующую ему вершину и вывести количество элементов в поддереве, корнем которого она является.

В первой строке входных данных записано число N ($N < 10^4$) — количество вершин дерева. Во второй строке указаны N натуральных различных чисел — ключи, по которым надо создать двоичное дерево поиска.

В следующей строке записано число Q — количество запросов.

В последней строке записано Q чисел — ключи, соответствующие корням различных поддеревьев в созданном двоичном дереве поиска.

Выведите Q целых чисел — количество вершин в каждом поддереве (включая его корень).

Input	Output
7 4 2 3 7 5 9 8 4 2 4 7 8	2 7 4 1

C. *Все вершины поддерева*

Даны натуральные числа A_i — ключи ДДП в порядке их добавления в дерево и перечень некоторых из них V_k.

Требуется создать двоичное дерево поиска и для каждого из указанных ключей V_k найти соответствующую ему вершину и вывести в возрастающем порядке *элементы* поддерева, корнем которого она является.

В первой строке входных данных записано число N ($N < 10^4$) — количество вершин дерева. Во второй строке указаны N натуральных различных чисел — ключи, по которым надо создать двоичное дерево поиска.

В следующей строке записано число Q — количество запросов.

В последней строке записано Q чисел — ключи, соответствующие корням различных поддеревьев в созданном двоичном дереве поиска.

Выведите Q последовательностей целых чисел — ключи вершин каждого поддерева в возрастающем порядке.

Input	Output
7 4 2 3 7 5 9 8 4 2 4 7 8	2 3 2 3 4 5 7 8 9 5 7 8 9 8

D. *Все листья поддерева*

Даны натуральные числа A_i — ключи ДДП в порядке их добавления в дерево и перечень некоторых из них V_k.

Требуется создать двоичное дерево поиска и для каждого из указанных ключей V_k найти соответствующую ему вершину и вывести в возрастающем порядке *листы* поддерева, корнем которого она является.

В первой строке входных данных записано число N ($N < 10^4$) — количество вершин дерева. Во второй строке указаны N натуральных различных чисел — ключи, по которым надо создать двоичное дерево поиска.

В следующей строке записано число Q — количество запросов.

В последней строке записано Q чисел — ключи, соответствующие корням различных поддеревьев в созданном двоичном дереве поиска.

Выведите Q последовательностей целых чисел — листья каждого поддерева в указанном порядке.

Input	Output
7 4 2 3 7 5 9 8 4 2 4 7 8	3 3 5 8 5 8 8

E. *Все «развилки» поддерева*

Даны натуральные числа A_i — ключи ДДП в порядке их добавления в дерево и перечень некоторых из них V_k.

Требуется создать двоичное дерево поиска и для каждого из указанных ключей V_k найти соответствующую ему вершину и вывести в возрастающем порядке «*развилки*» поддерева, корнем которого она является.

Развилка — это вершина, у которой есть оба ребёнка.

В первой строке входных данных записано число N ($N < 10^4$) — количество вершин дерева. Во второй строке указаны N натуральных различных чисел — ключи, по которым надо создать двоичное дерево поиска.

В следующей строке записано число Q — количество запросов.

В последней строке записано Q чисел — ключи, соответствующие корням различных поддеревьев в созданном двоичном дереве поиска.

Выведите Q последовательностей целых чисел — «развилки» каждого поддерева в указанном порядке.

Input	Output
8 4 2 3 7 5 9 8 1 3 4 2 7	2 4 7 2 7

F. *Проверка двоичного дерева поиска*

Дано двоичное дерево. Определить, является ли оно двоичным деревом поиска.

В первой строке входных данных записано число N ($N < 10^4$) — количество вершин дерева, у которых есть хотя бы один ребёнок. Затем в N строках записана информация о детях таких вершин — три числа (отсутствующий ребёнок обозначается -1):

`<родитель> <левый ребёнок> <правый ребёнок>`

Выведите `YES` или `NO` в зависимости от того, является ли это двоичное дерево двоичным деревом поиска.

Input	Output
3 4 3 8 3 2 -1 8 6 9	YES
4 4 3 8 3 2 -1 8 6 9 6 5 10	NO

G. *Построение описания двоичного дерева по ДДП*

Двоичное дерево поиска однозначно определяется порядком добавления ключей.

Это не взаимнооднозначное соответствие. Хотя каждой последовательности ключей соответствует единственное дерево (доказательство конструктивное и даётся самой процедурой построения дерева), но, как правило, двоичному дереву поиска соответствует несколько последовательностей ключей, его определяющего.
Например, такие последовательности задают одно и то же дерево:

```
2 1 3
2 3 1
```

По последовательности ключей (попарно различные натуральные числа), задающей двоичное дерево поиска, постройте дерево в виде словаря (`dict` для Python, `map` для C++). Ключом является ключ родительской вершины, а соответствующим ей значением пара ссылок на левого и правого ребёнка (-1, если он отсутствует)
Листья дерева (ключи, которым соответствует пара ссылок на -1) также должны быть описаны.
Выведите такой словарь в порядке возрастания ключей вершин.

Input	Output
8 4 3 8 6 7 9 5 2	2 -1 -1 3 2 -1 4 3 8 5 -1 -1 6 5 7 7 -1 -1 8 6 9 9 -1 -1

H. *Максимальный элемент в поддереве*

Даны натуральные числа A_i — ключи ДДП в порядке их добавления в дерево и перечень некоторых из них V_k.
Требуется создать двоичное дерево поиска и для каждого из указанных ключей V_k найти соответствующую ему вершину и вывести максимальный элемент в поддереве, корнем которого она является.
В первой строке входных данных записано число N ($N < 10^4$) — количество вершин дерева. Во второй строке указаны N натуральных различных чисел — ключи, по которым надо создать двоичное дерево поиска.
В следующей строке записано число Q — количество запросов.
В последней строке записано Q чисел.
Выведите Q целых чисел — максимальные элементы в поддеревьях.

Input	Output
9 4 2 3 7 5 9 8 1 6 5 4 2 7 5 9	9 3 9 6 9

I. *Второй максимум в поддереве*

Даны натуральные числа A_i — ключи ДДП в порядке их добавления в дерево и перечень некоторых из них V_k.
Требуется создать двоичное дерево поиска и для каждого из указанных ключей V_k найти соответствующую ему вершину и вывести второй максимум в поддереве, корнем которого она является.
В первой строке входных данных записано число N ($N < 10^4$) — количество вершин дерева. Во второй строке указаны N натуральных различных чисел — ключи, по которым надо создать двоичное дерево поиска.
В следующей строке записано число Q — количество запросов.
В последней строке записано Q чисел.
Выведите Q целых чисел — вторые максимумы в соответствующих поддеревьях. Если в поддереве нет двух элементов, выведите число -1.

Input	Output
8 4 2 3 7 5 9 8 1 4 7 2 9 3	8 2 8 -1

J. *Проверка существования полного поддерева данной высоты*

Задано двоичное дерево и ключи некоторых его вершин. Нужно проверить — существует ли полное поддерево данной высоты с корнем в каждой из этих вершин.

В первой строке входных данных записано число N ($N < 10^4$) — количество вершин дерева. Затем в N строках записана информация о вершинах (отсутствующий ребёнок обозначается -1):

`<родитель> <левый ребёнок> <правый ребёнок>`

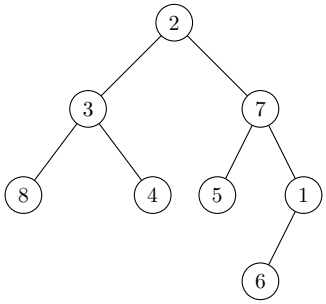
В следующей строке записано число Q — количество запросов.

В последней строке записано Q пар чисел V_k и H_k ($H_k > 0$): ключ и высота. Высота поддерева это максимальная длина пути от листа до корня.

Выведите Q строк `YES` или `NO` в зависимости от того, существует ли полное поддерево с корнем в V_k высоты H_k ($H_k > 0$, т.к. любая вершина представляет собой поддерево высоты 0 и такие запросы неинтересны).

Input	Output
8 2 3 7 3 8 4 8 -1 -1 4 -1 -1 7 5 1 5 -1 -1 1 6 -1 6 -1 -1 4 3 1 2 2 1 1 2 3	YES YES NO NO

Указание: на рисунке изображено дерево из теста. Найдите поддеревья указанной высоты с корнем в вершинах 3 и 2.



K. *Все максимальные полные двоичные поддеревья*

Задано двоичное дерево. Найдите в нём все полные двоичные поддеревья наибольшей глубины.

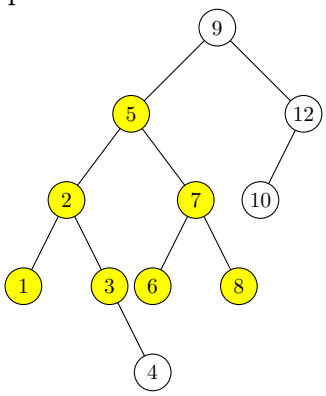
В первой строке входных данных записано число N ($N < 10^4$) — количество вершин дерева. Затем в N строках записана информация о вершинах и их детях в следующем формате(отсутствующий ребёнок обозначается -1):

`<родитель> <левый ребёнок> <правый ребёнок>`

Программа должна найти и вывести в возрастающем порядке все корни полных поддеревьев максимальной высоты. В каждой строке вывода указывается ключ вершины и высота максимального полного поддерева с корнем в ней.

Input	Output
11 9 5 12 5 2 7 12 10 -1 2 1 3 7 6 8 3 -1 4 1 -1 -1 4 -1 -1 6 -1 -1 8 -1 -1 10 -1 -1	5 2
4 8 -1 4 4 -1 7 7 -1 6 6 -1 -1	4 0 6 0 7 0 8 0

Иллюстрация к первому тесту: жёлтым цветом выделено полное поддерево с корнем в вершине 5 и высотой 2. Других полных поддеревьев высоты 2 в этом дереве нет.



Во втором тесте все вершины являются корнями полных поддеревьев высоты 0. Полных поддеревьев бóльшей высоты в этом дереве нет.

L. *Проверка pre-order обхода*

Необходимо проверить, может ли быть последовательность чисел результатом `pre-order` обхода какого-то двоичного дерева поиска.

В первой строке записано число Q — количество запросов. В следующих Q строках записаны последовательности вершин. Все вершины — натуральные числа, последнее число 0, означает конец ввода последовательности.

Для каждой последовательности программа должна вывести `YES` или `NO` в зависимости от того — может ли такая последовательность быть результатом работы `pre-order` обхода двоичного дерева поиска.

Input	Output
3 4 1 6 5 0 4 1 6 2 0 8848 0	YES NO YES

M. *Восстановить дерево по обходам*

Есть двоичное дерево (**не обязательно дерево поиска!**) и результаты двух его обходов: `in-order` и `pre-order`. Требуется по этим обходам восстановить структуру дерева.

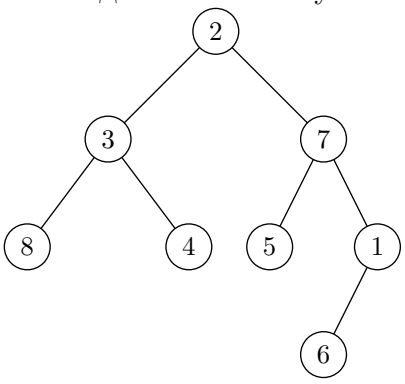
В первой строке записано число N — количество вершин в дереве. Во второй и третьей строках записаны результаты обходов `in-order` и `pre-order` соответственно.

Программа должна вывести граф в виде описания вершин в возрастающем порядке (отсутствующий ребёнок обозначается -1):

`<родитель> <левый ребёнок> <правый ребёнок>`

Input	Output
8 8 3 4 2 5 7 6 1 2 3 8 4 7 5 1 6	1 6 -1 2 3 7 3 8 4 4 -1 -1 5 -1 -1 6 -1 -1 7 5 1 8 -1 -1

Иллюстрация к тесту: указанным обходам соответствует такое дерево:

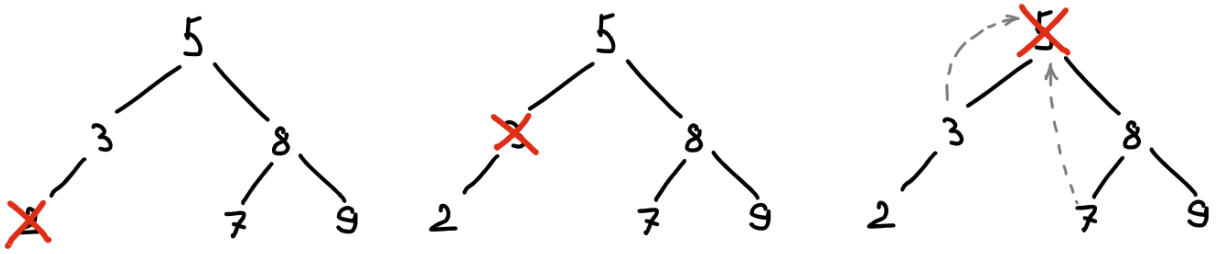


N. *Создание двоичного дерева поиска, поиск и удаление элементов*

Теперь к операции добавления элемента добавим операции поиска и удаления элемента.

Операция поиска мало чем отличается от операции добавления элемента — надо спускаться по ссылкам влево или вправо до тех пор, пока это можно делать и ключ текущего элемента не равен искомому. Если спускаться некуда, а ключ не найден, его в дереве нет.

Операция удаления несколько сложнее — надо рассмотреть три разных случая, изображённых на картинке ниже. Для удаления вершины, имеющей обоих детей, надо поставить на её место или максимальный элемент левого поддерева или минимальный элемент правого поддерева.



Указание: удаление можно реализовать рекурсивно (функция возвращает корень поддерева, в котором удаляется вершина с заданным ключом).

На вход программе даётся некоторое количество строк, в каждой из которых записана одна команда в следующем формате (все числа в командах положительные целые):

- `+<число>` — вставка ключа в ДДП
  - если ключ есть, вывести `<число> EXISTS`
  - если ключа нет, добавить ключ и вывести `<число> ADDED`)
- `?<число>` — проверка существования ключа в ДДП
  - если ключ есть, вывести `YES`
  - если ключа нет, вывести `NO`
- `-<число>` — удаление ключа из ДДП (для определённости — поменяйте с максимумом из левого поддерева)
  - если ключ есть, вывести `<число> DELETED`
  - если ключа нет, вывести `NO KEY TO DELETE`
- `?` — вывод ключей дерева
  - если дерево непустое, вывести ключи в порядке pre-order обхода
  - если дерево пустое, вывести `EMPTY`
- `stop` — означает окончание ввода, программа останавливается.

Input	Output
+7 +4 +9 ? ?6 ?7 +7 ? -7 +10 +8 +2 ? stop	4 ADDED 9 ADDED 4 9 7 NO YES 7 EXISTS 4 9 7 7 DELETED 10 ADDED 8 ADDED 2 ADDED 2 8 10 9 4
+5 +8 +1 +6 ? ?4 ?5 +1 +10 +12 +3 ? -5 ?5 ? stop	8 ADDED 1 ADDED 6 ADDED 1 6 8 5 NO YES 1 EXISTS 10 ADDED 12 ADDED 3 ADDED 3 1 6 12 10 8 5 5 DELETED NO 1 6 12 10 8 3

O. *Ближайший общий предок (LCA — lowest common ancestor)*

Для данного двоичного дерева поиска и пары ключей в нём определите ближайшего общего предка двух вершин с данными ключами.

Замечание: считайте, что любая вершина дерева является своим собственным предком. В первой строке входных данных записано число N ($N < 10^4$) — количество вершин дерева. Во второй строке указаны N натуральных различных чисел — ключи, по которым надо создать двоичное дерево поиска.

В следующей строке записано число Q — количество запросов.

Затем в Q строках записано по два числа — ключи вершин дерева.

Программа должна вывести Q чисел: ключи вершин, являющихся ближайшим общим предком каждой из указанных пар вершин.

Input	Output
12 5 2 4 1 9 13 15 6 3 11 10 12 2 12 6 4 13	9 5

Иллюстрация к ответу на первый тест.