

Ejudge - http://134.209.199.128/cgi-bin/new-client?contest_id=100

Deadline - 2019/11/19 1:00:00

При сдаче решений в систему читайте данные со стандартного потока ввода и пишите в стандартный поток вывода. В питоне это `input()` (построчно) и `print()`.

Во всех задачах обязательно использовать бинарное дерево поиска.

Строить его необходимо начиная с первого элемента.

1. Build_tree (2 балла)

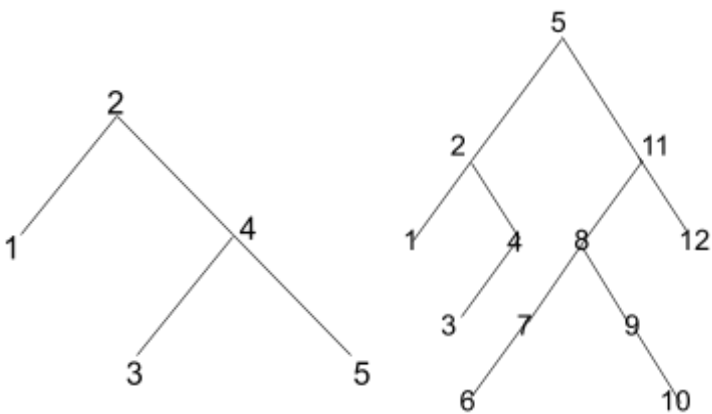
Получив на вход массив чисел, постройте по нему бинарное дерево поиска.

In:	Строка, где через пробел записаны числа.
Out:	Структура дерева, записанная в формате: (<i>node.left_child</i> , <i>node.data</i> , <i>node.right_child</i>)

Examples:

In:	2 1 4 5 3
Out:	(1, 2, (3, 4, 5))

In:	5 2 11 4 8 3 7 12 6 9 10 1
Out:	((1, 2, (3, 4, None)), 5, (((6, 7, None), 8, (None, 9, 10)), 11, 12))



Уточнение: при записи дерева лист записывать просто числом, а если у какого-то из узлов один ребенок, второго писать как None.

Указание: напишите свой класс *TreeNode* и пропишите в нем магический метод `__str__()`.

2. Find_element (1 балл)

Получив на вход массив элементов и элемент, который надо найти, выпишите путь по дереву до этого элемента.

In:	В первой строке через пробел записаны числа, по которым надо построить дерево. Во второй - искомый элемент.
Out:	Путь по дереву, записанный в формате <i>node1.data->node2.data-> ... ->nodeN.data</i>

Example:

In:	4 6 1 2 3 5 3
Out:	4->1->2->3

Примечание: можно заметить, что в такой постановке задачу легко решить и без дерева. Однако его все-таки нужно построить, без этого решение не засчитывается. Поверьте, вы не хотите принимать на вход готовое дерево в виде скобочной структуры...

3. Delete_element (3 балла)

Удалите элемент из дерева. Если у удаляемого элемента два потомка, ставьте на его место следующий по возрастанию элемент. Если такого нет, ставьте предыдущий.

In:	В первой строке через пробел записаны числа, по которым надо построить дерево. Во второй - удаляемый элемент.
Out:	Дерево после удаления в формате <i>(node.left_child, node.data, node.right_child)</i>

Examples:

In:	9 3 6 4 5 2 1 8 10 7 9
Out:	((((1, 2, None), 3, ((None, 4, 5), 6, (7, 8, None)))), 10, None)

In:	3 4 6 8 2 1 7 5 8
Out:	((1, 2, None), 3, (None, 4, (5, 6, 7)))

4. Count_height (1 балл)

Посчитайте максимальную высоту дерева - число ребер от корня до самого далекого листа.

In:	Строка, где через пробел записаны числа, по которым строится дерево.
Out:	Натуральное число - высота дерева.

Example:

In:	2 4 5 1 3
Out:	2

5. Rotate_right (1 балл)

Напишите [правое вращение](#) бинарного дерева вокруг выделенной связи.

In:	В первой строке через пробел записаны числа, по которым строится исходное дерево. Во второй тоже через пробел - две вершины, формирующие связь, вокруг которой надо вращать.
Out:	Дерево после вращения в формате <i>(node.left_child, node.data, node.right_child)</i>

Examples:

In:	6 4 2 1 3 7 5 4 2
Out:	((1, 2, (3, 4, 5)), 6, 7)

In:	7 8 4 5 2 3 1 6 4 2
Out:	((1, 2, (3, 4, (None, 5, 6))), 7, 8)

Дополнительные задания

6. RedMap (2 балла)

Реализуйте **ассоциативный контейнер** при помощи красно-черного дерева поиска.

В отличии от предыдущего задания, **реализовывать delete не требуется**.

Подробнее:

add key value - добавить в контейнер map пару (key, value). Если в таблице уже имеется пара с ключом key, то она удаляется. Таким образом, в любой момент времени произвольному ключу key может соответствовать максимум одно value. В обоих случаях функция по выполнению возвращает строку **Success**.

get key - по ключу key найти пару (key,value) в таблице и вернуть value. Если в таблице есть пара с таким ключом, возвращается строка **Success, <значение value>**. Иначе возвращается строка **Failure, None**

print - напечатать (в любом порядке) все пары (key, value), которые хранятся в таблице, через знак @. Если таблица пустая, вернуть пустую строку

В качестве ключей выступают строки, в качестве значений - числа

В наихудшем случае все операции должны выполняться за $O(\log N)$, где N - число пар в контейнере

In:	Набор строк, где на каждой строке записана команда, которую необходимо выполнить
Out:	Результаты выполнения команд

Examples:

In:	add apple 5 add dog 7 print add lemon 777
Out:	Success Success

	(dog, 7)@(apple, 5) Success
--	--------------------------------

In:	add apple 8 add dog 7 get dog print add apple 777 print
Out:	Success Success Success, 7 (dog, 7)@(apple, 8) Success (dog, 7)@(apple, 777)

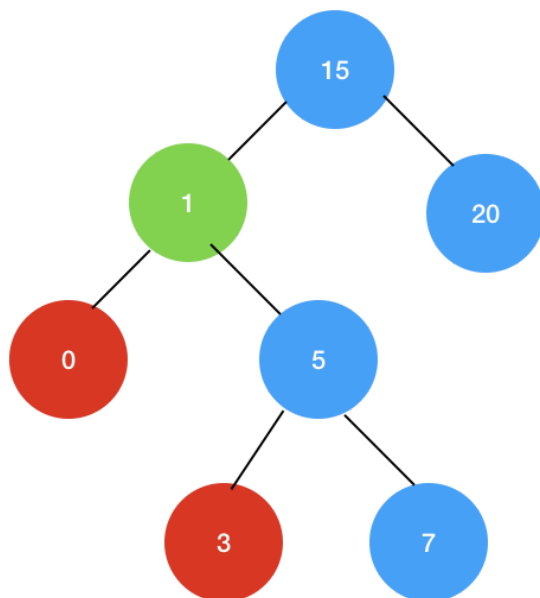
7. LCA (2 балла)

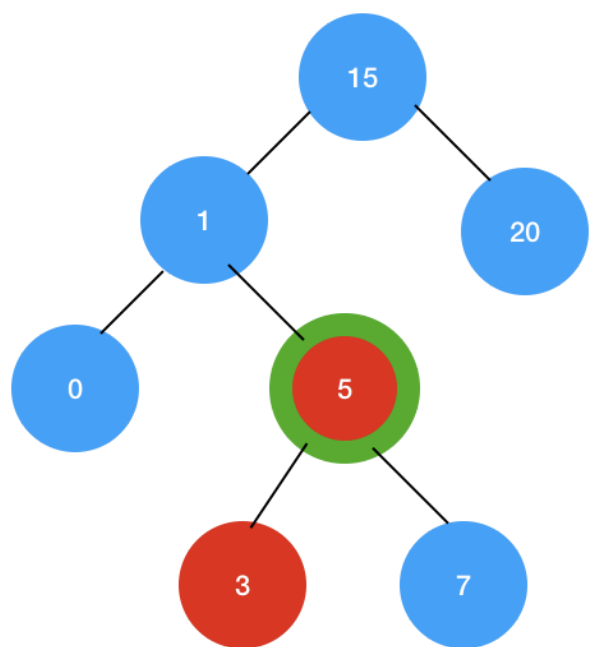
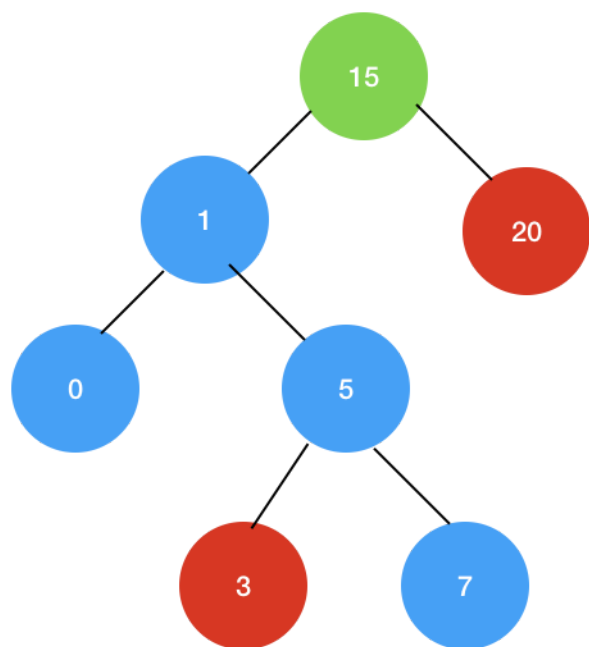
Реализуйте алгоритм поиска наименьшего общего предка. Он должен работать не более чем за $O(\log H)$ на запрос, где H - высота вашего дерева.

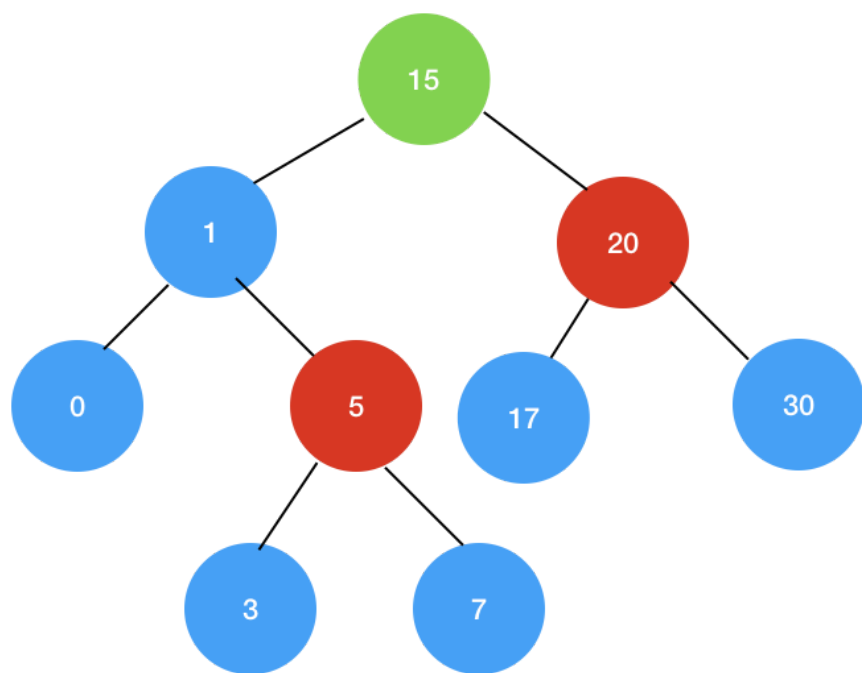
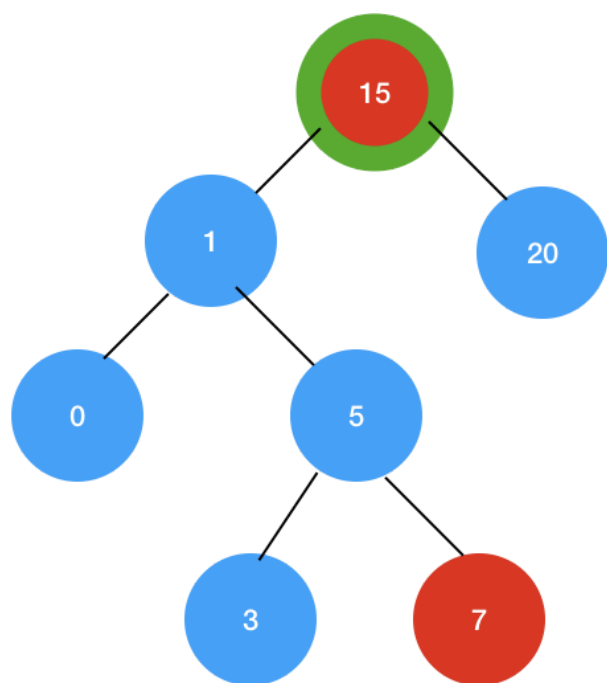
Используйте именно простое бинарное дерево поиска!

https://en.wikipedia.org/wiki/Lowest_common_ancestor

Примеры, зеленым выделен LCA, красным - узлы, для которых мы его ищем







In:	В первой строке через пробел записаны уникальные числа, по которым строится исходное дерево. В последующих строках записаны запросы в виде первое_число второе_число. Необходимо вывести их lca.
Out:	Ответы на lca-запросы

In:	15 20 30 17 1 0 5 3 7 0 3 3 20 20 3 3 5 7 15 5 20
Out:	1 15 15 5 15 15