Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
(Университет ИТМО)

Факультет **Инфокоммуникационных технологий**

Образовательная программа **Интеллектуальные системы в гуманитарной сфере**

Направление подготовки **45.03.04 Интеллектуальные системы в гуманитарной сфере**

О Т Ч Е Т

лабораторной работе 2

на тему: “Двоичные деревья поиска”

Обучающийся Королева Екатерина
К3143

Работа выполнена с оценкой ____

Преподаватель:

_____
(подпись)

Дата 22.06.2022

Санкт-Петербург, 2021

Задачи из варианта 7:

1 Задача. Обход двоичного дерева [5 s, 512 Mb, 1 балл]

В этой задаче вы реализуете три основных способа обхода двоичного дерева «в глубину»: центрированный (in-order), прямой (pre-order) и обратный (post-order). Очень полезно попрактиковаться в их реализации, чтобы лучше понять бинарные деревья поиска.

Вам дано корневое двоичное дерево. Выведите центрированный (in-order), прямой (pre-order) и обратный (post-order) обходы в глубину.

- **Формат ввода: стандартный ввод или input.txt.** В первой строке входного файла содержится количество узлов n. Узлы дерева пронумерованы от 0 до $n-1$. Узел 0 является корнем.

  Следующие n строк содержат информацию об узлах $0, 1, ..., n-1$ по порядку. Каждая из этих строк содержит три целых числа K_i, L_i и R_i. K_i – ключ i-го узла, L_i - индекс левого ребенка i-го узла, а R_i - индекс правого ребенка i-го узла. Если у i-го узла нет левого или правого ребенка (или обоих), соответствующие числа L_i или R_i (или оба) будут равны -1.

- **Ограничения на входные данные.** $1 \le n \le 10^5, 0 \le K_i \le 10^9, -1 \le L_i, R_i \le n-1$. Гарантируется, что данное дерево является двоичным деревом. В частности, если $L_i \ne -1$ и $R_i \ne -1$, то $L_i \ne R_i$. Кроме того, узел не может быть ребенком двух разных узлов. Кроме того, каждый узел является потомком корневого узла.

- **Формат вывода / выходного файла (output.txt).** Выведите три строки. Первая строка должна содержать ключи узлов при центрированном обходе дерева (in-order). Вторая строка должна содержать ключи узлов при прямом обходе дерева (pre-order). Третья строка должна содержать ключи узлов при обратном обходе дерева (post-order).

- Ограничение по времени. 5 сек.

- Ограничение по памяти. 512 мб.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


class Node:
    def __init__(self, val):
        self.left = None
        self.right = None
        self.val = val

    def insert_l(self, val):
        self.left = val

    def insert_r(self, val):
        self.right = val


def inOrder(root):
    global answ
    if root is None:
        return
    inOrder(root.left)
    answ += str(root.val) + ' '
```

```
    inOrder(root.right)


def preOrder(root):
    global answ
    if root is None:
        return
    answ += str(root.val) + ' '
    preOrder(root.left)
    preOrder(root.right)


def postOrder(root):
    global answ
    if root is None:
        return
    postOrder(root.left)
    postOrder(root.right)
    answ += str(root.val) + ' '


if __name__ == '__main__':
    with open('input.txt') as file:
        n = int(file.readline())
        sp = [list(map(int, file.readline().split())) for _ in
range(n)]

        tree = [Node(x[0]) for x in sp]

        for i in range(len(sp)):
            if sp[i][1] != -1:
                tree[i].insert_l(tree[sp[i][1]])

            if sp[i][2] != -1:
                tree[i].insert_r(tree[sp[i][2]])

        with open('output.txt', 'w') as file:
            answ = ''
            inOrder(tree[0])
            file.write(f'{answ}\n')

            answ = ''
            preOrder(tree[0])
            file.write(f'{answ}\n')
```

```
            answ = ''
            postOrder(tree[0])
            file.write(f'{answ}\n')

print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())

'''
Воспользуемся двоичным деревом. Расставим элементы
последовательно на места, и если рассматриваемый элемент не
является ребенком какого-либо узла, то добавим его в словарь
Q, чтобы сохранить элемент в памяти и добавить его позже.
'''
```

```
Время работы (в секундах): 0.0014318999999999998
Память 10027, и пик 27613
```

12 Задача. Проверка сбалансированности [2 s, 256 Mb, 2 балла]

АВЛ-дерево является сбалансированным в следующем смысле: для любой вершины высота ее левого поддерева отличается от высоты ее правого поддерева не больше, чем на единицу.

Введем понятие баланса вершины: для вершины дерева V ее баланс $B(V)$ равен разности высоты правого поддерева и высоты левого поддерева. Таким образом, свойство АВЛ-дерева, приведенное выше, можно сформулировать следующим образом: для любой ее вершины V выполняется следующее неравенство:

$$-1 \le B(V) \le 1$$

Обратите внимание, что, по историческим причинам, определение баланса в этой и последующих задачах этой недели «зеркально отражено» по сравнению с определением баланса в лекциях! Надеемся, что этот факт не доставит Вам неудобств. В литературе по алгоритмам – как российской, так и мировой – ситуация, как правило, примерно та же.

Дано двоичное дерево поиска. Для каждой его вершины требуется определить ее баланс.

- **Формат ввода / входного файла (input.txt).** Входной файл содержит описание двоичного дерева.

  В первой строке файла находится число N – число вершин в дереве. В последующих N строках файла находятся описания вершин дерева. В $(i+1)$-ой строке файла $(1 \le i \le N)$ находится описание i-ой вершины, состоящее из трех чисел K_i, L_i, R_i, разделенных пробелами – ключа K_i в i-ой вершине, номера левого L_i ребенка i-ой вершины $(i < L_i \le N$ или $L_i = 0$, если левого ребенка нет) и номера правого R_i ребенка i-ой вершины $(i < R_i \le N$ или $R_i = 0$, если правого ребенка нет). Все ключи различны. Гарантируется, что данное дерево является деревом поиска.

- **Ограничения на входные данные.** $0 \le N \le 2 \cdot 10^5, |K_i| \le 10^9$.

- **Формат вывода / выходного файла (output.txt).** Для i-ой вершины в i-ой строке выведите одно число – баланс данной вершины.

- **Ограничение по времени. 2 сек.**

- Ограничение по памяти. 256 мб.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


class node:

    def __init__(self):
        self.key = None
        self.left = None
        self.right = None
        self.parent = None
        self.height = None


class binTree:

    def __init__(self):
        self.root = None
        self.Q = dict()

    def search(self, key):
```

```
        if key in self.Q:
            return self.Q[key]
        return None

    def insert(self, key, left, right):
        t = self.search(key)
        if t is None:
            t = node()
            t.key = key
        if left == -1:
            t.left = None
        else:
            t.left = node()
            t.left.key = left
            t.left.parent = t
        if right == -1:
            t.right = None
        else:
            t.right = node()
            t.right.key = right
            t.right.parent = t
        if self.root is None:
            self.root = t
        self.Q[key] = t
        if t.left is not None:
            self.Q[left] = t.left
        if t.right is not None:
            self.Q[right] = t.right


with open('input.txt') as file1:
    n = int(file1.readline())
    Q = [(-1, 0, 0)]
    T = binTree()
    for i in range(n):
        Q.append(tuple(map(int, file1.readline().split())))
    for i in range(1, n+1):
        T.insert(Q[i][0], Q[Q[i][1]][0], Q[Q[i][2]][0])

    if n == 0:
        print(0)
    else:
        L = list()
        for i in range(n, 0, -1):
```

```
            t = T.search(Q[i][0])
            if t.left is None:
                if t.right is None:
                    L.append(0)
                    t.height = 1
                else:
                    L.append(t.right.height)
                    t.height = 1 + t.right.height
            elif t.right is None:
                L.append(t.left.height * -1)
                t.height = 1 + t.left.height
            else:
                L.append(t.right.height - t.left.height)
                t.height = 1 + max(t.right.height,
t.left.height)

        with open('output.txt', 'w') as file2:
            for j in range(n):
                file2.write(f'{L.pop()}\n')

print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())

'''
В задании класс узла содержит в себе ячейку height (высота
поддерева). Она подсчитывается в основной программе. В список
L, начиная с конца входного файла (т.е. с листьев дерева),
добавляем баланс поддерева. Затем выводим с конца список, т.е.
с начала входного файла.
'''
```

```
Время работы (в секундах): 0.0008208000000000035
Память 10518, и пик 27515
```

Замена 14 задачи на 16*

16 Задача. K-й максимум [2 s, 512 Mb, 3 балла]

Напишите программу, реализующую структуру данных, позволяющую добавлять и удалять элементы, а также находить k-й максимум.

- **Формат ввода / входного файла (input.txt).** Первая строка входного файла содержит натуральное число n – количество команд. Последующие n строк содержат по одной команде каждая. Команда записывается в виде двух чисел c_i и k_i – тип и аргумент команды соответственно. Поддерживаемые команды:
  - +1 (или просто 1): Добавить элемент с ключом k_i.
  - 0 : Найти и вывести k_i-й максимум.
  - -1 : Удалить элемент с ключом k_i.

  Гарантируется, что в процессе работы в структуре не требуется хранить элементы с равными ключами или удалять несуществующие элементы. Также гарантируется, что при запросе ki-го мак- симума, он существует.
- **Ограничения на входные данные.** $n \leq 100000$, $|k_i| \leq 10^9$.
- **Формат вывода / выходного файла (output.txt).** Для каждой команды нулевого типа в выходной файл должна быть выведена строка, содержащая единственное число – k_i-й максимум.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 512 мб.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


class Node:

    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
        self.parent = None
        self.size = None


class Tree:

    def __init__(self):
        self.root = None

    def find(self, data, root):
        if root is None or data == root.val:
            return root
        if data < root.val:
            if root.left is not None:
                return self.find(data, root.left)
            return root
```

```
        if root.right is not None:
            return self.find(data, root.right)
        return root

    def insert(self, data):
        if self.root is None:
            self.root = Node(data)
            self.root.size = 1
        else:
            t = self.find(data, self.root)
            if t.val == data:
                return
            elif data < t.val:
                t.left = Node(data)
                t.left.parent = t
                t.left.size = 1
            else:
                t.right = Node(data)
                t.right.parent = t
                t.right.size = 1
            while t is not None:
                t.size += 1
                t = t.parent

    def treeMin(self, root):
        while root.left is not None:
            root = root.left
        return root.val

    def rightAncestor(self, root):
        if root.parent is None:
            return 0
        if root.val < root.parent.val:
            return root.parent.val
        return self.rightAncestor(root.parent)

    def next(self, data):
        if self.root is None:
            return 0
        t = self.find(data, self.root)
        f = False
        if t.val != data:
            self.insert(data)
            f = True
```

```
            t = self.find(data, self.root)
        if t.right is not None:
            res = self.treeMin(t.right)
        else:
            res = self.rightAncestor(t)
        if f:
            self.delete(data)
        return res

    def delete(self, data):
        t = self.find(data, self.root)
        if t.right is None:
            prnt = t.parent
            if t.left is not None:
                t.left.parent = prnt
            if prnt is None:
                self.root = t.left
            elif prnt.left.val == t.val:
                prnt.left = t.left
            else:
                prnt.right = t.left
        else:
            x = self.find(self.next(t.val), self.root)
            y = x.right
            prnt = x.parent
            t.val = x.val
            if x.val != t.right.val:
                x.parent.left = y
                if y is not None:
                    y.parent = x.parent
            else:
                t.right = y
                if y is not None:
                    y.parent = t
        while prnt is not None:
            prnt.size -= 1
            prnt = prnt.parent

    def kMax(self, root, k):
        if root.right is None:
            s = 0
        else:
            s = root.right.size
        if k == s + 1:
```

```
            return root.val
        if k < s + 1:
            return self.kMax(root.right, k)
        return self.kMax(root.left, k - s - 1)


file1 = open('input.txt', 'r')
n = int(file1.readline())
T = Tree()
file2 = open('output.txt', 'w')
for i in range(n):
    com, key = list(map(int, file1.readline().split()))
    if com == 1:
        T.insert(key)
    elif com == -1:
        T.delete(key)
    else:
        file2.write(f'{T.kMax(T.root, key)}\n')


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())


'''
Для выполнения этой задачи воспользуемся функциями,
аналогичными функциям из заданий 3 (treeMin, rightAncestor,
next) и 4 (insert) (см дополнительные задачи). Дополнительно
изменим функцию delete и добавим функцию kMin для поиска k-го
максимума (аналогично поиску k-го минимума в задании 4).
'''
```

```
Время работы (в секундах): 0.0005840000000000012
Память 28175, и пик 36826
```

Дополнительные задачи:

3 Задача. Простейшее BST [2 s, 256 Mb, 1 балл]

В этой задаче вам нужно написать простейшее BST по явному ключу и отвечать им на запросы:

- «+ x» – добавить в дерево x (если x уже есть, ничего не делать).
- «> x» – вернуть минимальный элемент больше x или 0, если таких нет.

- **Формат ввода / входного файла (input.txt).** В каждой строке содержится один запрос. Все x - целые числа, количество запросов N не указано в начале, не более 300 000. Гарантируется, что все x выбраны равномерным распределением.
- Случайные данные! Не нужно ничего специально балансировать.
- **Ограничения на входные данные.** $1 \le x \le 10^9, 1 \le N \le 300000$
- **Формат вывода / выходного файла (output.txt).** Для каждого запроса вида «> x» выведите в отдельной строке ответ.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


class node:

    def __init__(self):
        self.key = None
        self.left = None
        self.right = None
        self.parent = None


class binTree:

    def __init__(self):
        self.root = None

    def search(self, data, root):
        if root is None or data == root.key:
            return root
        if data < root.key:
            if root.left is not None:
                return self.search(data, root.left)
            return root
        if root.right is not None:
            return self.search(data, root.right)
```

```
        return root

    def insert(self, data):
        if self.root is None:
            self.root = node()
            self.root.key = data
        else:
            t = self.search(data, self.root)
            if t.key == data:
                return
            elif data < t.key:
                t.left = node()
                t.left.key = data
                t.left.parent = t
            else:
                t.right = node()
                t.right.key = data
                t.right.parent = t

    def delete(self, data):
        if self.root is None:
            return
        if self.root.key == data:
            self.root = None
            return
        t = self.search(data, self.root)
        if data == t.key:
            if data < t.parent.key:
                t.parent.left = None
                return
            t.parent.right = None
            return

    def treeMin(self, root):
        while root.left is not None:
            root = root.left
        return root.key

    def rightAncestor(self, root):
        if root.parent is None:
            return 0
        if root.key < root.parent.key:
            return root.parent.key
        return self.rightAncestor(root.parent)
```

```
    def next(self, data):
        if self.root is None:
            return 0
        t = self.search(data, self.root)
        f = False
        if t.key != data:
            self.insert(data)
            f = True
            t = self.search(data, self.root)
        if t.right is not None:
            res = self.treeMin(t.right)
        else:
            res = self.rightAncestor(t)
        if f:
            self.delete(data)
        return res


with open('input.txt') as file:
    L = file.readlines()
    T = binTree()

for i in L:
    com, x = i.split()
    if com == '+':
        T.insert(int(x))
    else:
        print(T.next(int(x)))


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())

'''
Выполнение этой задачи аналогично задаче номер 2 с изменением
функций search и insert, а также добавлением некоторые новые
функций для нахождения ближайшего числа в дереве, большего
заданного.
'''
```

```
Время работы (в секундах): 0.00027340000000000003
Память 9464, и пик 22641
```

4 Задача. Простейший неявный ключ [2 s, 256 Mb, 1 балл]

В этой задаче вам нужно написать BST по **неявному** ключу и отвечать им на запросы:

- «+ x» – добавить в дерево x (если x уже есть, ничего не делать).
- «? k» – вернуть k-й по возрастанию элемент.

- **Формат ввода / входного файла (input.txt).** В каждой строке содержится один запрос. Все x - целые числа, количество запросов N не указано в начале, не более 300 000. Гарантируется, что все x выбраны равномерным распределением.
- Случайные данные! Не нужно ничего специально балансировать.
- **Ограничения на входные данные.** $1 \le x \le 10^9$, $1 \le N \le 300000$, в запросах «? k», число k от 1 до количества элементов в дереве.
- **Формат вывода / выходного файла (output.txt).** Для каждого запроса вида «? k» выведите в отдельной строке ответ.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


class Node:
    def __init__(self, val):
        self.left = None
        self.right = None
        self.val = val
        self.parent = None

    def insert_l(self, val):
        self.left = val

    def insert_r(self, val):
        self.right = val


class Tree:

    def __init__(self):
        self.root = None

    def find(self, data, root):
        if root is None or data == root.val:
            return root
        if data < root.val:
```

```
            if root.left is not None:
                return self.find(data, root.left)
            return root
        if root.right is not None:
            return self.find(data, root.right)
        return root

    def insert(self, data):
        if self.root is None:
            self.root = Node(data)

        else:
            x = self.find(data, self.root)
            if x.val == data:
                return
            elif data < x.val:
                x.left = Node(data)
                x.left.parent = x
            else:
                x.right = Node(data)
                x.right.parent = x

    def treeMin(self, root):
        while root.left is not None:
            root = root.left
        return root.val

    def rightAncestor(self, root):
        if root.parent is None:
            return 0
        if root.val < root.parent.val:
            return root.parent.val
        return self.rightAncestor(root.parent)

    def next(self, data):
        if self.root is None:
            return 0
        x = self.find(data, self.root)
        f = False
        if x.val != data:
            self.insert(data)
            f = True
            x = self.find(data, self.root)
        if x.right is not None:
```

```
            result = self.treeMin(x.right)
        else:
            result = self.rightAncestor(x)
        if f:
            self.delete(data)
        return result

    def delete(self, data):
        x = self.find(data, self.root)
        if x.right is None:
            parent = x.parent
            if x.left is not None:
                x.left.parent = parent
            if parent is None:
                self.root = x.left
            elif parent.left.val == x.val:
                parent.left = x.left
            else:
                parent.right = x.left
        else:
            temp = self.find(self.next(x.val), self.root)
            y = temp.right
            x.val = temp.val
            if temp.val != x.right.val:
                temp.parent.left = y
                if y is not None:
                    y.parent = temp.parent
            else:
                x.right = y
                if y is not None:
                    y.parent = x


def inOrder(root):
    global sp
    if root is None:
        return
    inOrder(root.left)
    sp.append(root.val)
    inOrder(root.right)


if __name__ == '__main__':
    file1 = open('input.txt')
```

```
    file2 = open('output.txt', 'w')
    tree = Tree()
    while line := file1.readline().split():
        operator, num = line[0], int(line[1])
        if operator == '+':
            tree.insert(num)
        else:
            sp = []
            inOrder(tree.root)
            file2.write(f'{str(sp[num - 1])}\n')


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())

'''
Аналогично заданию 3 с изменением функции вставки и
добавлением функции для поиска k-го минимального элемента.
'''
```

```
Время работы (в секундах): 0.0006174000000000041
Память 29038, и пик 37569
```

6 Задача. Опознание двоичного дерева поиска [10 s, 512 Mb, 1.5 балла]

В этой задаче вы собираетесь проверить, правильно ли реализована структура данных бинарного дерева поиска. Другими словами, вы хотите убедиться, что вы можете находить целые числа в этом двоичном дереве, используя бинарный поиск по дереву, и вы всегда получите правильный результат: если целое число есть в дереве, вы его найдете, иначе – нет.

Вам дано двоичное дерево с ключами - целыми числами. Вам нужно проверить, является ли это правильным двоичным деревом поиска. Для каждой вершины дерева V выполняется следующее условие:

- все ключи вершин из левого поддерева меньше ключа вершины V;
- все ключи вершин из правого поддерева больше ключа вершины V.

Другими словами, узлы с меньшими ключами находятся слева, а узлы с большими ключами – справа. Вам необходимо проверить, удовлетворяет ли данная структура двоичного дерева этому условию. Вам гарантируется, что входные данные содержат допустимое двоичное дерево. То есть это дерево, и каждый узел имеет не более двух ребенков.

- **Формат ввода / входного файла (input.txt).** В первой строке входного файла содержится количество узлов n. Узлы дерева пронумерованы от 0 до $n-1$. Узел 0 является корнем.

  Следующие n строк содержат информацию об узлах $0, 1, ..., n-1$ по порядку. Каждая из этих строк содержит три целых числа K_i, L_i и R_i. K_i – ключ i-го узла, L_i - индекс левого ребенка i-го узла, а R_i - индекс правого ребенка i-го узла. Если у i-го узла нет левого или правого ребенка (или обоих), соответствующие числа L_i или R_i (или оба) будут равны -1.

- **Ограничения на входные данные.** $0 \le n \le 10^5$, $-2^{31} \le K_i \le 2^{31}-1$, $-1 \le L_i, R_i \le n-1$. Гарантируется, что данное дерево является двоичным деревом. В частности, если $L_i \ne -1$ и $R_i \ne -1$, то $L_i \ne R_i$. Кроме того, узел не может быть ребенком двух разных узлов. Кроме того, каждый узел является потомком корневого узла.

  Все ключи во входных данных различны.

- **Формат вывода / выходного файла (output.txt).** Если заданное двоичное дерево является правильным двоичным деревом поиска, выведите одно слово «CORRECT» (без кавычек). В противном случае выведите одно слово «INCORRECT» (без кавычек).

- Ограничение по времени. 10 сек.

- Ограничение по памяти. 512 мб.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


class Node:
    def __init__(self, val):
        self.left = None
        self.right = None
        self.val = val

    def insert_l(self, val):
        self.left = val

    def insert_r(self, val):
        self.right = val


def isBST(root):
    stack = []
    prev = None
```

```
    while root or stack:
        while root:
            stack.append(root)
            root = root.left
        root = stack.pop()
        if prev and root.val < prev.val:
            return False
        prev = root
        root = root.right
    return True


if __name__ == '__main__':
    with open('input.txt') as file:
        n = int(file.readline())
        sp = []
        for i in range(n):
            x = list(map(int, file.readline().split()))
            sp.append(x)

        tree = []
        for i in range(len(sp)):
            tree.append(Node(sp[i][0]))

        for i in range(len(sp)):
            if sp[i][1] != -1:
                tree[i].insert_l(tree[sp[i][1]])

            if sp[i][2] != -1:
                tree[i].insert_r(tree[sp[i][2]])

    with open('output.txt', 'w') as file:
        if tree:
            if isBST(tree[0]):
                file.write('CORRECT')
            else:
                file.write('INCORRECT')
        else:
            file.write('CORRECT')


print("Время работы (в секундах):",
time.perf_counter()-t_start)
```

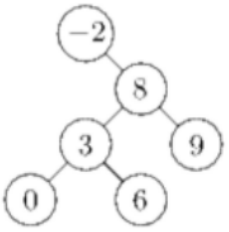
```
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())

'''
Для дерева запустим функцию isBST, проверяющую, является
ли корень меньше, чем предыдущее значение числа. Если да, то
выведем INCORRECT, иначе – CORRECT.
'''
```

```
Время работы (в секундах): 0.0008536000000000003
Память 5080, и пик 20285
```

8 Задача. Высота дерева возвращается [2 s, 256 Mb, 2 балла]

Высотой дерева называется максимальное число вершин дерева в цепочке, начинающейся в корне дерева, заканчивающейся в одном из его листьев, и не содержащей никакую вершину дважды.

Так, высота дерева, состоящего из единственной вершины, равна единице. Высота пустого дерева равна нулю. Высота дерева, изображенного на рисунке, равна четырем.



Дано **двоичное дерево поиска**. В вершинах этого дерева записаны ключи – целые числа, по модулю не превышающие 10^9. Для каждой вершины дерева V выполняется следующее условие:

- все ключи вершин из левого поддерева меньше ключа вершины V;
- все ключи вершин из правого поддерева больше ключа вершины V.

Найдите высоту данного дерева.

- **Формат ввода / входного файла (input.txt).** Входной файл содержит описание двоичного дерева. В первой строке файла находится число N – число вершин в дереве. В последующих N строках файла находятся описания вершин дерева. В $(i+1)$-ой строке файла $(1 \le i \le N)$ находится описание i-ой вершины, состоящее из трех чисел K_i, L_i, R_i, разделенных пробелами – ключа K_i в i-ой вершине, номера левого L_i ребенка i-ой вершины $(i < L_i \le N$ или $L_i = 0$, если левого ребенка нет) и номера правого R_i ребенка i-ой вершины $(i < R_i \le N$ или $R_i = 0$, если правого ребенка нет).
- **Ограничения на входные данные.** $0 \le N \le 2 \cdot 10^5, |K_i| \le 10^9$. Все ключи различны. Гарантируется, что данное дерево является деревом поиска.
- **Формат вывода / выходного файла (output.txt).** Выведите одно целое число – высоту дерева.
- **Ограничение по времени. 2 сек.**
- Ограничение по памяти. 256 мб.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


class Node:
    def __init__(self, val):
        self.left = None
        self.right = None
        self.val = val

    def insert_l(self, val):
        self.left = val

    def insert_r(self, val):
        self.right = val
```

```
def treeHeight(root):
    if root.val is None:
        return 0
    sp = []
    sp.append(root)
    height = 0
    while True:
        nodeCount = len(sp)
        if nodeCount == 0:
            return height
        height += 1
        while nodeCount > 0:
            node = sp[0]
            sp.pop(0)
            if node.left is not None:
                sp.append(node.left)
            if node.right is not None:
                sp.append(node.right)
            nodeCount -= 1


if __name__ == '__main__':
    with open('input.txt') as file:
        n = int(file.readline())
        sp = [list(map(int, file.readline().split())) for _ in
range(n)]

    tree = [Node(x[0]) for x in sp]

    for i in range(n):
        if sp[i][1] != 0:
            tree[i].insert_l(tree[sp[i][1] - 1])

        if sp[i][2] != 0:
            tree[i].insert_r(tree[sp[i][2] - 1])

    with open('output.txt', 'w') as file:
        if tree:
            file.write(str(treeHeight(tree[0])))
        else:
            file.write('0')
```

```
print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())

'''
В этом задании каждый узел будет содержать дополнительную
ячейку deep – глубину узла. Глубины всех листьев дерева мы
будем добавлять в массив, максимум из которого и будет высотой
дерева.
'''
```

```
Время работы (в секундах): 0.0006721999999999978
Память 6808, и пик 20829
```

Результаты тестов на OpenEdu:

в процессе

9 Задача. Удаление поддеревьев [2 s, 256 Mb, 2 балла]

Дано некоторое двоичное дерево поиска. Также даны запросы на удаление из него вершин, имеющих заданные ключи, причем вершины удаляются целиком вместе со своими поддеревьями.

После каждого запроса на удаление выведите число оставшихся вершин в дереве.

В вершинах данного дерева записаны ключи – целые числа, по модулю не превышающие 10^9. Гарантируется, что данное дерево является двоичным деревом поиска, в частности, для каждой вершины дерева V выполняется следующее условие:

- все ключи вершин из левого поддерева меньше ключа вершины V;
- все ключи вершин из правого поддерева больше ключа вершины V.

Высота дерева не превосходит 25, таким образом, можно считать, что оно сбалансировано.

- **Формат ввода / входного файла (input.txt).** Входной файл содержит описание двоичного дерева и описание запросов на удаление.

  В первой строке файла находится число N – число вершин в дереве. В последующих N строках файла находятся описания вершин дерева. В $(i+1)$-ой строке файла $(1 \le i \le N)$ находится описание i-ой вершины, состоящее из трех чисел K_i, L_i, R_i, разделенных пробелами – ключа K_i в i-ой вершине, номера левого L_i ребенка i-ой вершины $(i < L_i \le N$ или $L_i = 0$, если левого ребенка нет) и номера правого R_i ребенка i-ой вершины $(i < R_i \le N$ или $R_i = 0$, если правого ребенка нет).

  Все ключи различны. Гарантируется, что данное дерево является деревом поиска.

  В следующей строке находится число M – число запросов на удаление. В следующей строке находятся M чисел, разделенных пробелами – ключи, вершины с которыми (вместе с их поддеревьями) необходимо удалить. Все эти числа не превосходят 10^9 по абсолютному значению. Вершина с таким ключом не обязана существовать в дереве – в этом случае дерево изменять не требуется. Гарантируется, что корень дерева никогда не будет удален.

- **Ограничения на входные данные.** $1 \le N \le 2 \cdot 10^5$, $|K_i| \le 10^9$, $1 \le M \le 2 \cdot 10^5$
- **Формат вывода / выходного файла (output.txt).** Выведите M строк. На i-ой строке требуется вывести число вершин, оставшихся в дереве после выполнения i-го запроса на удаление.
- **Ограничение по времени. 2 сек.**
- Ограничение по памяти. 256 мб.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()

from collections import deque


class Node:

    def __init__(self):
        self.val = None
        self.left = None
        self.right = None
        self.parent = None
        self.iter = None


class Tree:

    def __init__(self):
```

```
        self.root = None
        self.nodes = {}

    def deleteSubtree(self, val):
        if val in self.nodes:
            root = self.nodes[val]
            if root.parent.left == root:
                root.parent.left = None
            else:
                root.parent.right = None
            delete_list = deque()
            delete_list.append(root)
            while len(delete_list) != 0:
                node = delete_list.popleft()
                if node.left:
                    delete_list.append(node.left)
                if node.right:
                    delete_list.append(node.right)
                self.nodes.pop(node.iter)
            global file
            file.write(str(len(self.nodes)) + '\n')


if __name__ == '__main__':

    with open('input.txt') as file:
        n = int(file.readline())
        tree = Tree()
        data = []
        nodes_iter = {}
        for i in range(1, n+1):
            data.append(list(map(int,
file.readline().split())))
            tree.nodes[i] = Node()
            tree.nodes[i].val = data[i - 1][0]
            tree.nodes[i].iter = i
            nodes_iter[data[i-1][0]] = i
        m = int(file.readline())
        for_delete = list(map(int, file.readline().split()))

    for i in range(1, n+1):
        if data[i-1][1] != 0:
            tree.nodes[i].left = tree.nodes[data[i-1][1]]
            tree.nodes[data[i-1][1]].parent = tree.nodes[i]
```

```
        if data[i-1][2] != 0:
            tree.nodes[i].right = tree.nodes[data[i-1][2]]
            tree.nodes[data[i-1][2]].parent = tree.nodes[i]
        if i == 1:
            tree.root = tree.nodes[i]

    with open('output.txt', 'w') as file:
        for j in for_delete:
            if j in nodes_iter:
                j = nodes_iter[j]
            else:
                j = 0
            if j in tree.nodes:
                tree.deleteSubtree(j)
            else:
                file.write(str(len(tree.nodes)) + '\n')


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())

'''
В класс Tree была добавили функция deleteSubtree, которая
удаляет требуемое поддерево, вершина которой является ключом,
введенным ранее.
'''
```

```
Время работы (в секундах): 0.0007203999999999996
Память 9944, и пик 25291
```

Результаты тестов на OpenEdu:

Верное решение!
Результаты работы Вашего решения

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.484	140443648	6029382	1077960
1	OK	0.015	9543680	58	12
2	OK	0.062	9478144	27	12
3	OK	0.031	9535488	34	15
4	OK	0.031	9588736	211	30
5	OK	0.031	9601024	246	30
6	OK	0.000	9650176	3437	457
7	OK	0.031	9650176	3363	483
8	OK	0.031	10006528	18842	4247
9	OK	0.031	10240000	25683	3739
10	OK	0.046	11137024	69351	14791
11	OK	0.046	11735040	88936	11629
12	OK	0.078	14868480	244892	40297
13	OK	0.062	15114240	255614	37596
14	OK	0.234	30474240	978616	141281
15	OK	0.250	30752768	992647	137802
16	OK	0.562	57835520	2488583	634135
17	OK	0.812	84443136	3489729	483105
18	OK	1.046	101900288	4639039	1077960
19	OK	1.468	139022336	6007604	931260
20	OK	1.484	140443648	6029382	916969

10 Задача. Проверка корректности [2 s, 256 Mb, 2 балла]

Свойство двоичного дерева поиска можно сформулировать следующим образом: для каждой вершины дерева выполняется следующее условие:

- все ключи вершин из левого поддерева меньше ключа вершины V;
- все ключи вершин из правого поддерева больше ключа вершины V.

Дано двоичное дерево. Проверьте, выполняется ли для него свойство двоичного дерева поиска.

- **Формат ввода / входного файла (input.txt).** Входной файл содержит описание двоичного дерева.

  В первой строке файла находится число N – число вершин в дереве. В последующих N строках файла находятся описания вершин дерева. В $(i+1)$-ой строке файла $(1 \le i \le N)$ находится описание i-ой вершины, состоящее из трех чисел K_i, L_i, R_i, разделенных пробелами – ключа K_i в i-ой вершине, номера левого L_i ребенка i-ой вершины $(i < L_i \le N$ или $L_i = 0$, если левого ребенка нет) и номера правого R_i ребенка i-ой вершины $(i < R_i \le N$ или $R_i = 0$, если правого ребенка нет).

- **Ограничения на входные данные.** $0 \le N \le 2 \cdot 10^5, |K_i| \le 10^9$.
- На 60% от при $0 \le N \le 2000$.
- **Формат вывода / выходного файла (output.txt).** Выведите «YES», если данное во входном файле дерево является двоичным деревом поиска, и «NO», если не является.
- **Ограничение по времени. 2 сек.**
- Ограничение по памяти. 256 мб.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()

class Node:
    def __init__(self, val):
        self.left = None
        self.right = None
        self.val = val

    def insert_l(self, val):
        self.left = val

    def insert_r(self, val):
        self.right = val


def isBST(root):
    stack = []
    prev = None

    while root or stack:
        while root:
            stack.append(root)
            root = root.left
        root = stack.pop()
```

```
        if prev and root.val <= prev.val:
            return False
        prev = root
        root = root.right
    return True


if __name__ == '__main__':
    with open('input.txt') as file:
        n = int(file.readline())
        sp = []
        for i in range(n):
            x = list(map(int, file.readline().split()))
            sp.append(x)

        tree = []
        for i in range(len(sp)):
            tree.append(Node(sp[i][0]))

        for i in range(len(sp)):
            if sp[i][1] != 0:
                tree[i].insert_l(tree[sp[i][1] - 1])

            if sp[i][2] != 0:
                tree[i].insert_r(tree[sp[i][2] - 1])

    with open('output.txt', 'w') as file:
        if tree:
            if isBST(tree[0]):
                file.write('YES')
            else:
                file.write('NO')
        else:
            file.write('YES')

print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())

'''
Создадим дерево и проверим его через функцию isBST. Если
корень меньше либо равен любому узлу, то выводим NO, иначе -
YES.
'''
```

```
Время работы (в секундах): 0.0006210999999999994
Память 5960, и пик 20693
```

Результаты тестов на OpenEdu:

Верное решение!
Результаты работы Вашего решения

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла	Группа тестов
Max		0.765	89464832	3988813	3	
1	OK	0.031	9113600	46	3	0
2	OK	0.031	9093120	3	3	
3	OK	0.031	9150464	24	2	
4	OK	0.015	9146368	11	3	
5	OK	0.031	9105408	18	3	
6	OK	0.031	9142272	4325	2	
7	OK	0.015	9134080	847	3	
8	OK	0.031	9080832	103	3	2
9	OK	0.031	9109504	103	2	
10	OK	0.031	9129984	155	3	2
11	OK	0.046	9142272	155	2	
12	OK	0.031	9113600	162	3	2
13	OK	0.031	9146368	161	2	
14	OK	0.015	9089024	117	3	2
15	OK	0.015	9093120	117	2	
16	OK	0.046	9117696	156	3	2
17	OK	0.031	9142272	160	2	
18	OK	0.031	9138176	2099	3	2
19	OK	0.015	9052160	2099	2	
20	OK	0.031	9142272	1197	3	2
21	OK	0.015	9142272	1197	2	
22	OK	0.015	9125888	2132	3	2
23	OK	0.031	9097216	2146	2	

24	OK	0.046	9113600	1430	3	2
25	OK	0.015	9121792	1430	2	
26	OK	0.000	9154560	2073	3	2
27	OK	0.015	9138176	2111	2	
28	OK	0.046	9142272	4325	3	2
29	OK	0.031	9125888	4325	2	
30	OK	0.031	9146368	5762	3	2
31	OK	0.000	9142272	5762	2	
32	OK	0.031	9089024	5928	3	2
33	OK	0.031	9158656	5934	2	
34	OK	0.062	9158656	3812	3	2
35	OK	0.031	9125888	3812	2	
36	OK	0.062	9089024	6102	3	2
37	OK	0.031	9142272	5805	2	
38	OK	0.031	9162752	8777	3	2
39	OK	0.031	9183232	8777	2	
40	OK	0.031	9424896	16336	3	2
41	OK	0.031	9379840	16336	2	
42	OK	0.031	9428992	16797	3	2
43	OK	0.015	9404416	16842	2	
44	OK	0.031	9281536	11983	3	2
45	OK	0.031	9224192	11983	2	
46	OK	0.031	9388032	17153	3	2
47	OK	0.031	9412608	17133	2	

47	OK	0.031	9412608	17133	2	2
48	OK	0.031	9412608	17704	3	2
49	OK	0.031	9420800	17704	2	
50	OK	0.031	9560064	26475	3	2
51	OK	0.031	9633792	26475	2	
52	OK	0.031	9650176	27231	3	2
53	OK	0.031	9625600	27244	2	
54	OK	0.031	9465856	18473	3	2
55	OK	0.015	9453568	18473	2	
56	OK	0.031	9633792	27600	3	2
57	OK	0.031	9613312	26824	2	
58	OK	0.031	9396224	17704	3	2
59	OK	0.031	9375744	17704	2	
60	OK	0.031	9785344	34899	3	2
61	OK	0.031	9764864	34899	2	
62	OK	0.031	9826304	35924	3	2
63	OK	0.031	9797632	35971	2	
64	OK	0.031	9691136	30116	3	2
65	OK	0.031	9670656	30116	2	
66	OK	0.031	9789440	34532	3	2
67	OK	0.031	9760768	36638	2	
68	OK	0.046	12263424	149518	3	2
69	OK	0.046	12328960	149518	2	
70	OK	0.046	12685312	164193	3	2
71	OK	0.031	12652544	164193	2	
72	OK	0.062	12685312	168785	3	2
73	OK	0.046	12664832	168853	2	
74	OK	0.046	12283904	147528	3	2
75	OK	0.046	12304384	147528	2	

76	OK	0.062	12742656	167534	3	2
77	OK	0.078	12681216	164402	2	
78	OK	0.125	22175744	624213	3	2
79	OK	0.125	22122496	624213	2	
80	OK	0.125	23011328	489475	3	2
81	OK	0.125	22962176	489475	2	
82	OK	0.140	22822912	654217	3	2
83	OK	0.109	22982656	654112	2	
84	OK	0.125	21704704	597441	3	2
85	OK	0.109	21630976	597441	2	
86	OK	0.125	22773760	638262	3	2
87	OK	0.125	22589440	638147	2	
88	OK	0.250	35606528	1259549	3	2
89	OK	0.234	35373056	1259549	2	
90	OK	0.453	58077184	2254723	3	2
91	OK	0.390	56991744	2254723	2	
92	OK	0.421	56832000	2313825	3	2
93	OK	0.437	57737216	2313957	2	
94	OK	0.406	52535296	2051218	3	2
95	OK	0.343	53186560	2051218	2	
96	OK	0.437	56954880	2289346	3	2
97	OK	0.421	58478592	2285336	2	
98	OK	0.484	62836736	2561292	3	2
99	OK	0.484	63635456	2561292	2	
100	OK	0.734	89018368	3888903	3	2
101	OK	0.718	89006080	3888903	2	
102	OK	0.765	89407488	3988548	3	2
103	OK	0.687	89391104	3988813	2	
104	OK	0.500	63561728	2613999	3	2
105	OK	0.468	63213568	2613999	2	
106	OK	0.765	89464832	3987751	3	2
107	OK	0.750	89370624	3894584	2	

Вывод:

В ходе работы были изучены принципы работы с двоичными деревьями поиска, в том числе сбалансированными. Их применение на практике помогает относительно просто и довольно эффективно искать элементы и их ближайших по значению «соседей» в структуре.