**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 **Мобильные и сетевые технологии**

Направление подготовки (специальность) **09.03.03 Прикладная информатика**

**О Т Ч Е Т П О Л А Б О Р А Т О Р Н О Й Р А Б О Т Е**

**по дисциплине «Алгоритмы и структуры данных»**

на тему: Двоичные деревья поиска

**Выполнила**:

Олейникова Полина Леонидовна,

Студент 1 курса гр. K3140

**Преподаватель**:

Харьковская Татьяна Александровна

**Дата сдачи**:

03.05.2022

Санкт-Петербург

2022 г.

**Оглавление**

[1 задача. Обход двоичного дерева (1 балла) 3](#_Toc102396393)

[2 задача. Гирлянда (1 балла) 5](#_Toc102396394)

[3 задача. Задача. Простейшее BST (1 балла) 7](#_Toc102396395)

[5 задача. Простое двоичное дерево поиска (1 балла) 10](#_Toc102396396)

[7 задача. Опознание двоичного дерева поиска (2.5 балла) 13](#_Toc102396397)

[8 задача. Высота дерева возвращается (2 балла) 16](#_Toc102396398)

[9 задача. Удаление поддеревьев (2 балла) 18](#_Toc102396399)

[10 задача. Проверка корректности (2 балла) 21](#_Toc102396400)

[12 задача. Проверка сбалансированности (2 балла) 23](#_Toc102396401)

# **1 задача. Обход двоичного дерева (1 балла)**

Задание. В этой задаче вы реализуете три основных способа обхода двоичного дерева «в глубину»: центрированный (inorder), 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 (postorder) 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 Ki, Li и Ri. Ki – ключ i-го узла, Li - индекс левого ребенка i-го узла, а Ri - индекс правого ребенка i-го узла. Если у i-го узла нет левого или правого ребенка (или обоих), соответствующие числа Li или Ri (или оба) будут равны −1.

• Ограничения на входные данные. 1 ≤ n ≤ 105, 0 ≤ Ki ≤ 109, −1 ≤ Li, Ri ≤ n−1. Гарантируется, что данное дерево является двоичным деревом. В частности, если Li ̸= −1 и Ri ̸= −1, то Li ̸= Ri. Кроме того, узел не может быть ребенком двух разных узлов. Кроме того, каждый узел является потомком корневого узла.

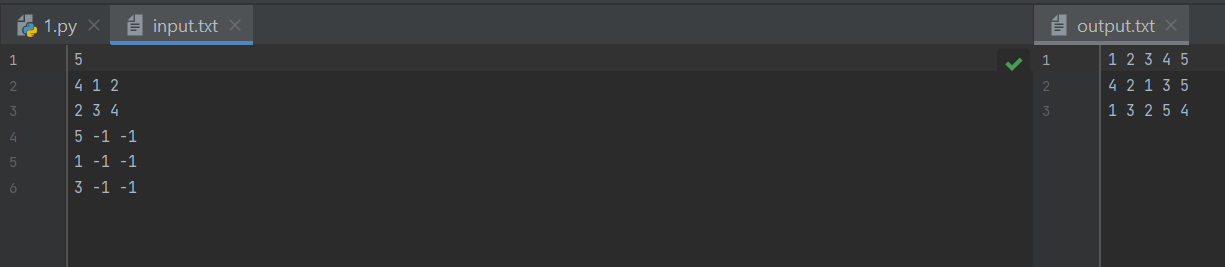
• Формат вывода / выходного файла (output.txt). Выведите три строки. Первая строка должна содержать ключи узлов при центрированном обходе дерева (in-order). Вторая строка должна содержать ключи узлов при прямом обходе дерева (pre-order). Третья строка должна содержать ключи узлов при обратном обходе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

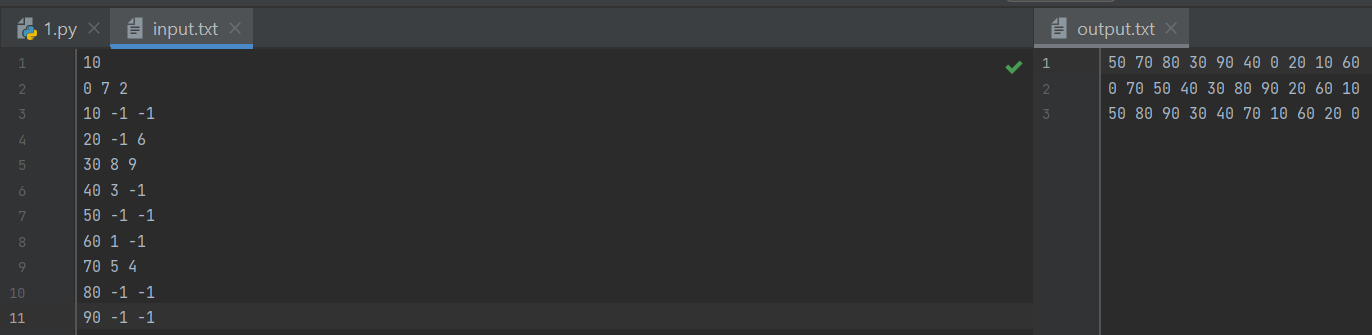
• Ограничение по времени. 5 сек.

• Ограничение по памяти. 512 мб.

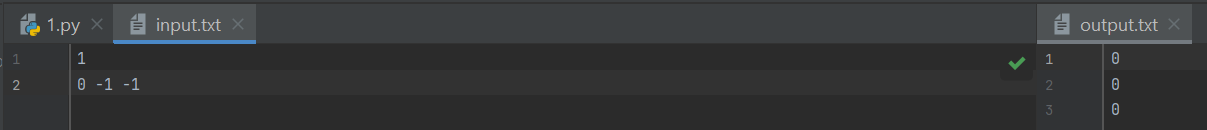
import time  
  
  
def inorder(self, res, tree):  
 if self[1] != -1:  
 inorder(tree[self[1]], res, tree)  
 res.append(self[0])  
 if self[2] != -1:  
 inorder(tree[self[2]], res, tree)  
 return res  
  
  
def preorder(self, res, tree):  
 res.append(self[0])  
 if self[1] != -1:  
 preorder(tree[self[1]], res, tree)  
 if self[2] != -1:  
 preorder(tree[self[2]], res, tree)  
 return res  
  
  
def postorder(self, res, tree):  
 if self[1] != -1:  
 postorder(tree[self[1]], res, tree)  
 if self[2] != -1:  
 postorder(tree[self[2]], res, tree)  
 res.append(self[0])  
 return res  
  
  
def main():  
 with open('input.txt') as f:  
 n = int(f.readline())  
 tree = []  
 for i in range(n):  
 k, l, r = map(int, f.readline().split())  
 tree.append([k, l, r])  
 with open('output.txt', 'w') as f:  
 f.write(" ".join(map(str, inorder(tree[0], [], tree))) + '\n')  
 f.write(" ".join(map(str, preorder(tree[0], [], tree))) + '\n')  
 f.write(" ".join(map(str, postorder(tree[0], [], tree))))  
  
  
time\_start = time.perf\_counter()  
main()  
print(time.perf\_counter() - time\_start)

Результаты тестирования для примеров из текста лабораторной:





Результаты тестирования для левой границы:



Пояснение. Полностью обходим одно поддерево, прежде чем исследовать соседнее поддерево.

# **2 задача. Гирлянда (1 балла)**

Задание. Гирлянда состоит из n лампочек на общем проводе. Один её конец закреплён на заданной высоте A мм (h1 = A). Благодаря силе тяжести гирлянда прогибается: высота каждой неконцевой лампы на 1 мм меньше, чем средняя высота ближайших соседей (hi = hi−1 + hi+1 2 − 1 для 1 < i < N).

Требуется найти минимальное значение высоты второго конца B (B = hn), такое что для любого ϵ > 0 при высоте второго конца B + ϵ для всех лампочек выполняется условие hi > 0. Обратите внимание на то, что при данном значении высоты либо ровно одна, либо две соседних лампочки будут иметь нулевую высоту.

Подсказка: для решения этой задачи можно использовать двоичный поиск.

• Формат ввода / входного файла (input.txt). В первой строке входного файла содержится два числа n и A.

• Ограничения на входные данные. 3 ≤ n ≤ 1000, n – целое, 10 ≤ A ≤ 1000, A – вещественное и дано не более чем с тремя знаками после десятичной точки.

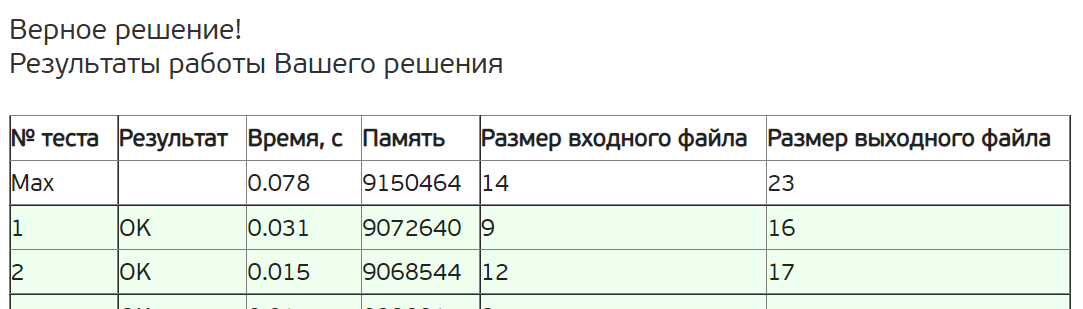
• Формат вывода / выходного файла (output.txt). Выведите одно вещественное число B – минимальную высоту второго конца. Ваш ответ будет засчитан, если он будет отличаться от правильного не более, чем на 10−6.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

def main():  
 with open('input.txt') as f:  
 n, height = f.readline().split()  
 n = int(n)  
 height = float(height)  
  
 h = [None for i in range(n)]  
 h[0] = height  
  
 l, r = 0, height  
  
 while r - l > 0.0000000001:  
 h[1] = (l + r) / 2  
 up = True  
 for i in range(2, n):  
 h[i] = 2 \* h[i - 1] - h[i - 2] + 2  
 if h[i] < 0:  
 up = False  
 break  
  
 if up:  
 r = h[1]  
 else:  
 l = h[1]  
  
 with open('output.txt', 'w') as f:  
 f.write(str(h[n - 1]))  
  
  
main()

Результаты тестирования:



# 

# **3 задача. Задача. Простейшее BST (1 балла)**

Задание. 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 ≤ x ≤ 109 , 1 ≤ N ≤ 300000

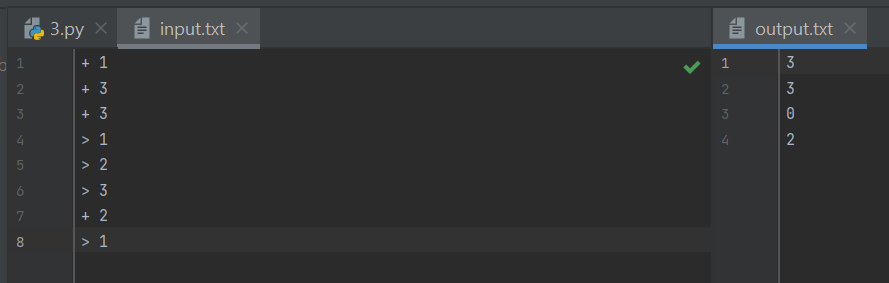
• Формат вывода / выходного файла (output.txt). Для каждого запроса вида «> x» выведите в отдельной строке ответ.

• Ограничение по времени. 2 сек.

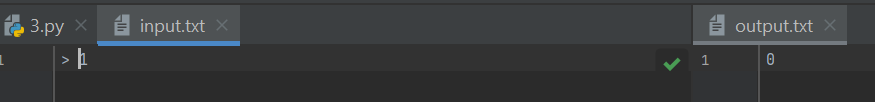
• Ограничение по памяти. 256 мб.

import random  
import time  
  
  
class BSTNode:  
 def \_\_init\_\_(self, val=None):  
 self.left = None  
 self.right = None  
 self.val = val  
  
 def insert(self, val):  
 if not self.val:  
 self.val = val  
 return  
  
 if self.val == val:  
 return  
  
 if val < self.val:  
 if self.left:  
 self.left.insert(val)  
 return  
 self.left = BSTNode(val)  
 return  
  
 if self.right:  
 self.right.insert(val)  
 return  
 self.right = BSTNode(val)  
  
 def next(self, res, val):  
 if self.val is not None and self.val > val:  
 if res == 0 or res > self.val:  
 res = self.val  
 if self.left is not None:  
 res = self.left.next(res, val)  
 if self.right is not None:  
 res = self.right.next(res, val)  
 return res  
  
  
def main():  
 bst = BSTNode()  
 res = []  
 with open('input.txt') as f:  
 for line in f.readlines():  
 if '+' in line:  
 bst.insert(int(line.replace('+', '')))  
 elif '>' in line:  
 res.append(bst.next(0, int(line.replace('>', ''))))  
  
 with open('output.txt', 'w') as f:  
 f.write('\n'.join(map(str, res)))  
  
  
def test():  
 with open('input.txt', 'w') as f:  
 symbol = ['+ ', '> ']  
 for i in range(3 \* 10 \*\* 3 - 1):  
 f.write(symbol[random.randint(0, 1)] + str(random.randint(1, 10 \*\* 9)) + '\n')  
 f.write(symbol[random.randint(0, 1)] + str(random.randint(1, 10 \*\* 9)))  
  
  
test()  
time\_start = time.perf\_counter()  
main()  
print(time.perf\_counter() - time\_start)

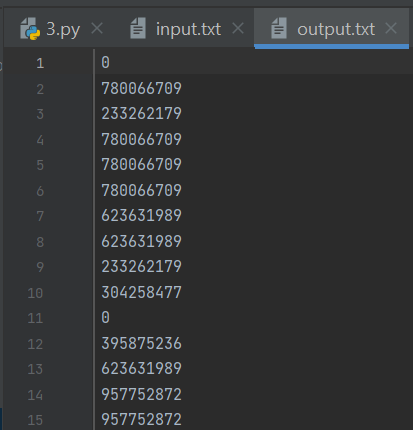
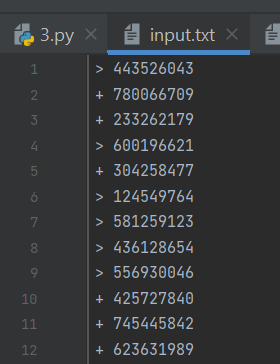
Результаты тестирования для примеров из текста лабораторной:



Результаты тестирования для левой границы:



Результаты тестирования для правой границы (Время 0.45629600000000003):



Пояснение. Создан класс двоичного дерева с функциями вставки и получения следующего элемента больше данного. Функции рекурсивная. Во второй происходит прямой обход дерева и ищется нужный элемент.

# **5 задача. Простое двоичное дерево поиска (1 балла)**

Задание. Реализуйте простое двоичное дерево поиска.

• Формат ввода / входного файла (input.txt). Входной файл содержит описание операций с деревом, их количество N не превышает 100. В каждой строке находится одна из следующих операций:

* insert x – добавить в дерево ключ x. Если ключ x есть в дереве, то ничего делать не надо;
* delete x – удалить из дерева ключ x. Если ключа x в дереве нет, то ничего делать не надо;
* exists x – если ключ x есть в дереве выведите «true», если нет – «false»;
* next x – выведите минимальный элемент в дереве, строго больший x, или «none», если такого нет;
* prev x – выведите максимальный элемент в дереве, строго меньший x, или «none», если такого нет. В дерево помещаются и извлекаются только целые числа, не превышающие по модулю 10:9 .

• Ограничения на входные данные. 0 ≤ N ≤ 100, |xi | ≤ 109 .

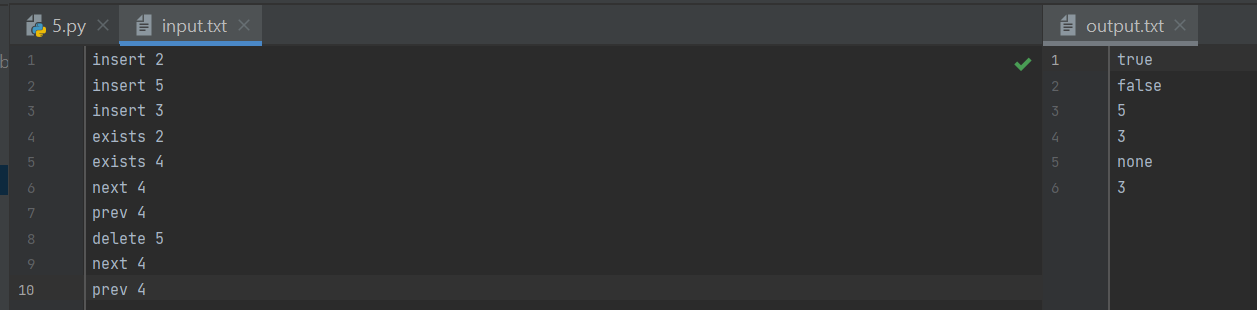
• Формат вывода / выходного файла (output.txt). Выведите последовательно результат выполнения всех операций exists, next, prev. Следуйте формату выходного файла из примера.

• Ограничение по времени. 2 сек.

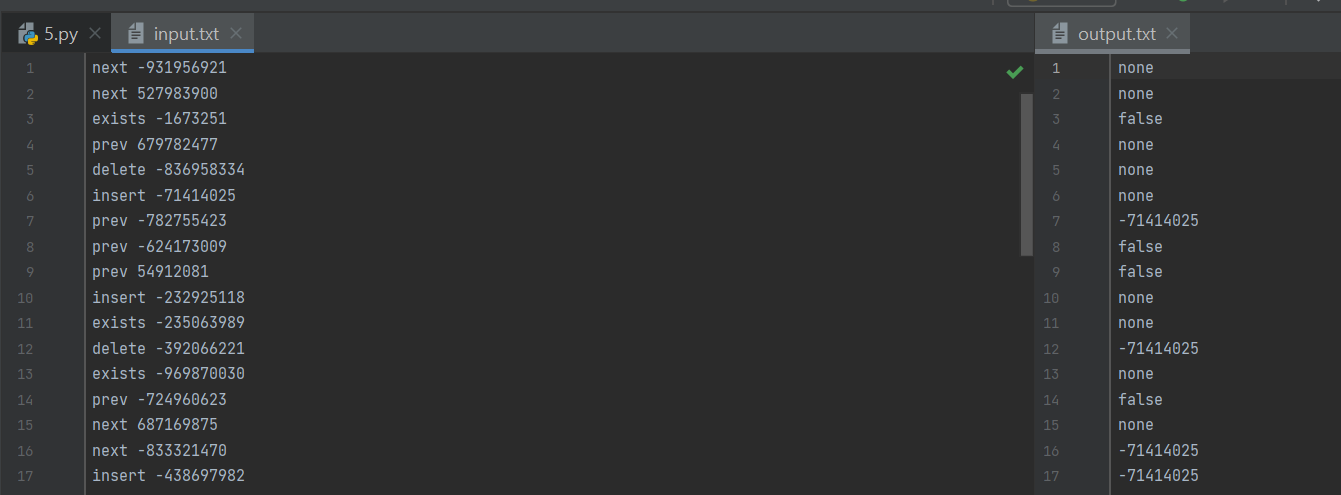
• Ограничение по памяти. 512 мб.

import random  
import time  
  
  
class BSTNode:  
 def \_\_init\_\_(self, val=None):  
 self.left = None  
 self.right = None  
 self.val = val  
  
 def insert(self, val):  
 if not self.val:  
 self.val = val  
 return  
  
 if self.val == val:  
 return  
  
 if val < self.val:  
 if self.left:  
 self.left.insert(val)  
 return  
 self.left = BSTNode(val)  
 return  
  
 if self.right:  
 self.right.insert(val)  
 return  
 self.right = BSTNode(val)  
  
 def delete(self, val):  
 if self.val is None:  
 return self  
  
 if val < self.val:  
 if self.left:  
 self.left = self.left.delete(val)  
 return self  
 if val > self.val:  
 if self.right:  
 self.right = self.right.delete(val)  
 return self  
 if self.right is None:  
 return self.left  
 if self.left is None:  
 return self.right  
 min\_larger\_node = self.right  
 while min\_larger\_node.left:  
 min\_larger\_node = min\_larger\_node.left  
 self.val = min\_larger\_node.val  
 self.right = self.right.delete(min\_larger\_node.val)  
 return self  
  
 def exists(self, val):  
 if not self.val:  
 return False  
  
 if val == self.val:  
 return True  
  
 if val < self.val:  
 if self.left is None:  
 return False  
 return self.left.exists(val)  
  
 if self.right is None:  
 return False  
 return self.right.exists(val)  
  
 def next(self, res, val):  
 if self.val is not None and self.val > val:  
 if res is None or res < self.val:  
 res = self.val  
 if self.left is not None:  
 res = self.left.next(res, val)  
 if self.right is not None:  
 res = self.right.next(res, val)  
 return res  
  
 def prev(self, res, val):  
 if self.val is not None and self.val < val:  
 if res is None or res < self.val:  
 res = self.val  
 if self.left is not None:  
 res = self.left.prev(res, val)  
 if self.right is not None:  
 res = self.right.prev(res, val)  
 return res  
  
  
def main():  
 bst = BSTNode()  
 res = []  
 with open('input.txt') as f:  
 for line in f.readlines():  
 if 'insert' in line:  
 bst.insert(int(line.replace('insert', '')))  
 if 'delete' in line:  
 bst.delete(int(line.replace('delete', '')))  
 if 'exists' in line:  
 res.append(bst.exists(int(line.replace('exists', ''))))  
 elif 'prev' in line:  
 res.append(bst.prev(None, int(line.replace('prev', ''))))  
 elif 'next' in line:  
 res.append(bst.next(None, int(line.replace('next', ''))))  
  
 with open('output.txt', 'w') as f:  
 f.write('\n'.join(map(lambda val: str(val).lower(), res)))  
  
  
def test():  
 with open('input.txt', 'w') as f:  
 symbol = ['insert ', 'exists ', 'next ', 'prev ', 'delete ']  
 for i in range(99):  
 f.write(symbol[random.randint(0, 4)] + str(random.randint(-10 \*\* 9, 10 \*\* 9)) + '\n')  
 f.write(symbol[random.randint(0, 4)] + str(random.randint(-10 \*\* 9, 10 \*\* 9)))  
  
  
test()  
time\_start = time.perf\_counter()  
main()  
print(time.perf\_counter() - time\_start)

Результаты тестирования для примеров из текста лабораторной:



Результаты тестирования для правой границы (Время 0.01168770000000001):

…

Пояснение. Создан класс двоичного дерева с функциями нужными в задании. Функции рекурсивные. В функциях next, prev, exists происходит прямой обход дерева и ищется нужный элемент. Функция insert такая же как в 3 задании.

# **7 задача. Опознание двоичного дерева поиска (2.5 балла)**

Задание. Эта задача отличается от предыдущей тем, что двоичное дерева поиска может содержать равные ключи. Вам дано двоичное дерево с ключами - целыми числами, которые могут повторяться. Вам нужно проверить, является ли это правильным двоичным деревом поиска. Теперь, 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 или равны ключу вершины V.

Другими словами, узлы с меньшими ключами находятся слева, а узлы с большими ключами – справа, дубликаты всегда справа. Вам необходимо проверить, удовлетворяет ли данная структура двоичного дерева этому условию.

• 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 Ki , Li и Ri . Ki – ключ i-го узла, Li - индекс левого ребенка i-го узла, а Ri - индекс правого ребенка i-го узла. Если у i-го узла нет левого или правого ребенка (или обоих), соответствующие числа Li или Ri (или оба) будут равны −1.

• Ограничения на входные данные. 0 ≤ n ≤ 105 , −2 31 ≤ Ki ≤ 2 31 − 1, −1 ≤ Li , Ri ≤ n − 1. Гарантируется, что данное дерево является двоичным деревом. В частности, если Li ̸= −1 и Ri ̸= −1, то Li ̸= Ri . Кроме того, узел не может быть ребенком двух разных узлов. Кроме того, каждый узел является потомком корневого узла. Обратите внимание, что минимальное и максимальное возможные значения 32-битного целочисленного типа могут быть ключами в дереве.

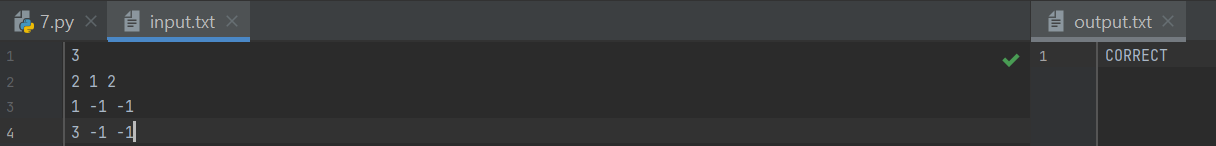
• Формат вывода / выходного файла (output.txt). Если заданное двоичное дерево является правильным двоичным деревом поиска, выведите одно слово «CORRECT» (без кавычек). В противном случае выведите одно слово «INCORRECT» (без кавычек).

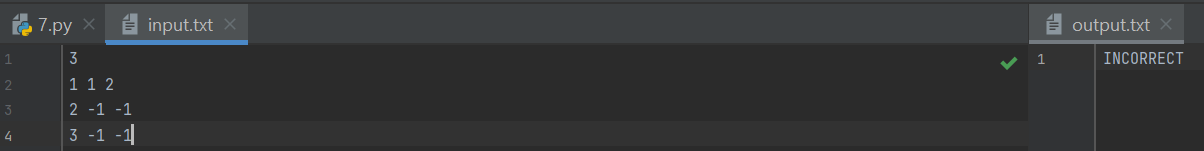
• Ограничение по времени. 10 сек.

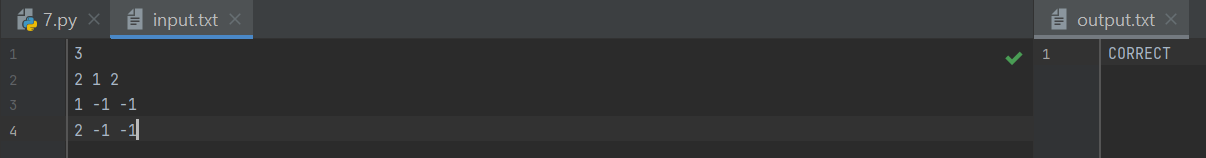
• Ограничение по памяти. 512 мб.

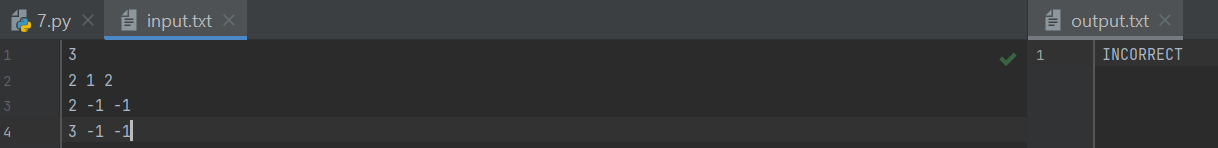
def main():  
 with open('input.txt') as f:  
 n = int(f.readline())  
 tree = []  
 for i in range(n):  
 k, l, r = map(int, f.readline().split())  
 tree.append([k, l, r])  
  
 f = open('output.txt', 'w')  
 if n != 0:  
 is\_correct\_tree = True  
  
 for i in range(n):  
 cur\_value = tree[i][0]  
 left\_index = tree[i][1]  
 right\_index = tree[i][2]  
  
 if left\_index != -1 and tree[left\_index][0] >= cur\_value:  
 is\_correct\_tree = False  
 break  
 if right\_index != -1 and tree[right\_index][0] < cur\_value:  
 is\_correct\_tree = False  
 break  
  
 if is\_correct\_tree:  
 f.write("CORRECT")  
 else:  
 f.write("INCORRECT")  
 else:  
 f.write("CORRECT")  
  
  
main()

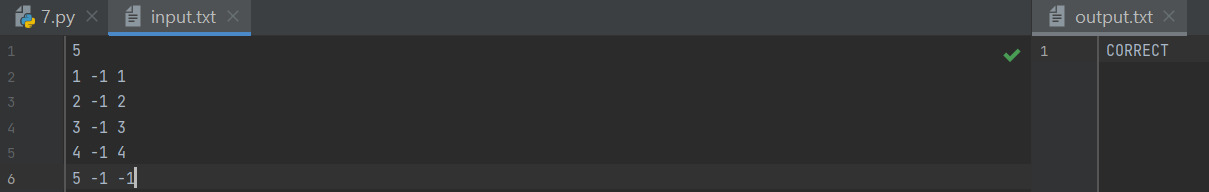
Результаты тестирования для примеров из текста лабораторной:

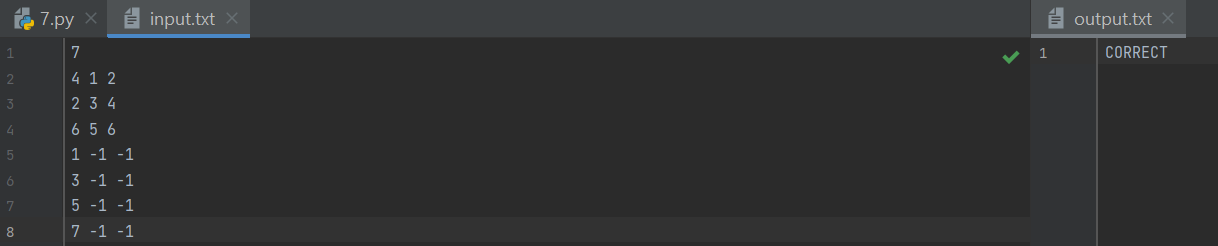


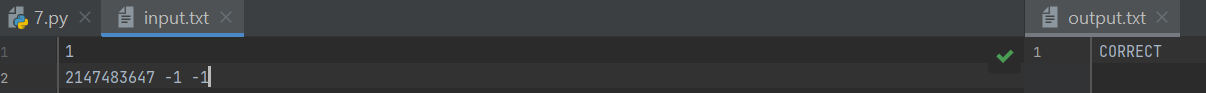




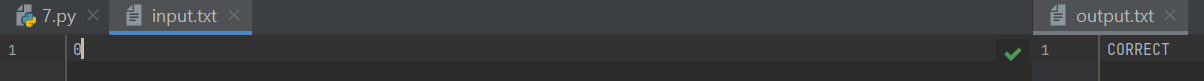








Результаты тестирования для левой границы:



Пояснение.

# **8 задача. Высота дерева возвращается (2 балла)**

Задание. 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 ≤ i ≤ N) находится описание i-ой вершины, состоящее из трех чисел Ki , Li , Ri , разделенных пробелами – ключа Ki в i-ой вершине, номера левого Li ребенка i-ой вершины (i < Li ≤ N или Li = 0, если левого ребенка нет) и номера правого Ri ребенка i-ой вершины (i < Ri ≤ N или Ri = 0, если правого ребенка нет).

• Ограничения на входные данные. 0 ≤ N ≤ 2·10^5, |Ki | ≤ 10^9. Все ключи различны. Гарантируется, что данное дерево является деревом поиска.

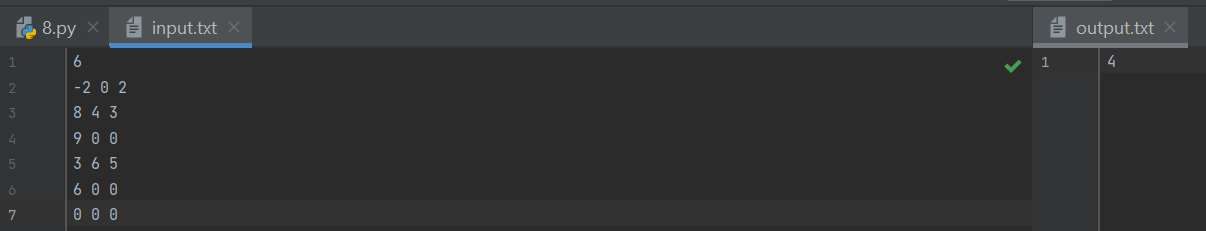
• Формат вывода / выходного файла (output.txt). Выведите одно целое число – высоту дерева.

• Ограничение по времени. 2 сек.

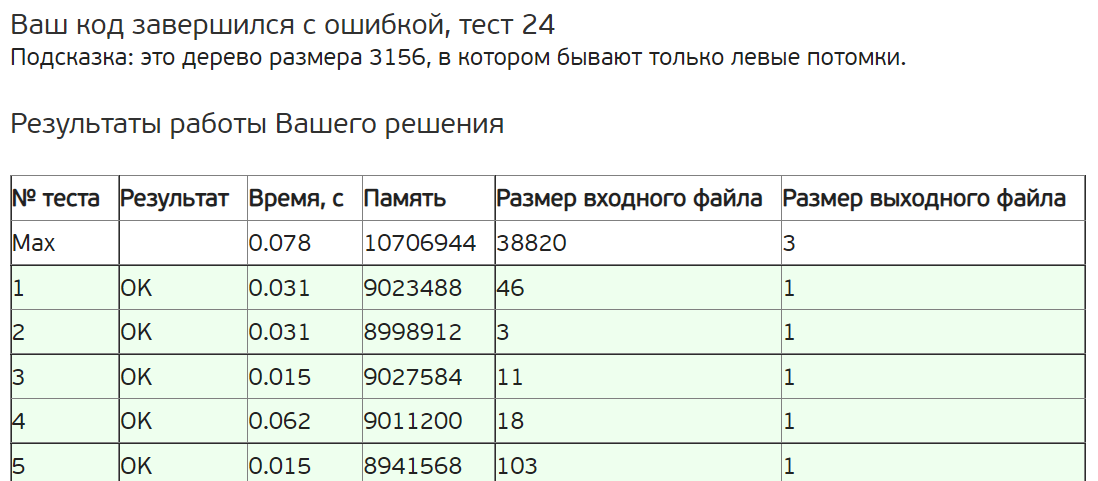
• Ограничение по памяти. 256 мб.

import time  
  
  
def height(tree, root):  
 lh = 0  
 if root[1] != 0:  
 lh = height(tree, tree[root[1] - 1])  
 rh = 0  
 if root[2] != 0:  
 rh = height(tree, tree[root[2] - 1])  
 return max(lh, rh) + 1  
  
  
def main():  
 with open('input.txt') as f:  
 n = int(f.readline())  
 tree = []  
 for i in range(n):  
 k, l, r = map(int, f.readline().split())  
 tree.append([k, l, r])  
  
 with open('output.txt', 'w') as f:  
 if len(tree) > 0:  
 f.write(str(height(tree, tree[0])))  
 else:  
 f.write('0')  
  
  
time\_start = time.perf\_counter()  
main()  
print(time.perf\_counter() - time\_start)

Результаты тестирования для примеров из текста лабораторной:



Результаты тестирования для правой границы:



Пояснение. Рекурсивная функция получения высоты. При этом проверяется есть ли левый или правый ребенок.

# **9 задача. Удаление поддеревьев (2 балла)**

Задание. 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 ≤ i ≤ N) находится описание i-ой вершины, состоящее из трех чисел Ki , Li , Ri , разделенных пробелами – ключа Ki в i-ой вершине, номера левого Li ребенка i-ой вершины (i < Li ≤ N или Li = 0, если левого ребенка нет) и номера правого Ri ребенка i-ой вершины (i < Ri ≤ N или Ri = 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 109 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 ≤ N ≤ 2 · 10^5, |Ki | ≤ 10^9, 1 ≤ M ≤ 2 · 10^5

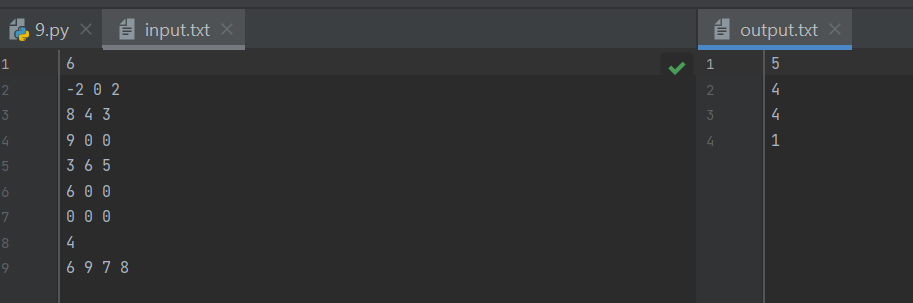
• Формат вывода / выходного файла (output.txt). Выведите M строк. На i-ой строке требуется вывести число вершин, оставшихся в дереве после выполнения i-го запроса на удаление.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

class BSTNode:  
 def \_\_init\_\_(self, key):  
 self.val = key  
 self.left = None  
 self.right = None  
  
  
def delete(self, val=None):  
 if self.val is None:  
 return  
  
 if val == self.val or val is None:  
 global count  
 count -= 1  
 self.val = None  
 if self.left is not None:  
 delete(self.left)  
 if self.right is not None:  
 delete(self.right)  
  
 elif val < self.val:  
 if self.left is not None:  
 delete(self.left, val)  
  
 elif val > self.val:  
 if self.right is not None:  
 delete(self.right, val)  
  
 return  
  
  
def main():  
 f = open('input.txt')  
 f2 = open('output.txt', 'w')  
  
 n = int(f.readline())  
 global count  
 count = n  
 tree\_list = [BSTNode(0) for i in range(n)]  
 for i in range(n):  
 val, left, right = map(int, f.readline().split())  
 left -= 1  
 right -= 1  
 tree\_list[i].val = val  
 if left != -1:  
 tree\_list[i].left = tree\_list[left]  
 if right != -1:  
 tree\_list[i].right = tree\_list[right]  
  
 m = int(f.readline())  
 delete\_values = list(map(int, f.readline().split()))  
  
 for i in range(m):  
 delete(tree\_list[0], delete\_values[i])  
 f2.write(str(count) + ('\n' if i < m - 1 else ''))  
  
  
count = 0  
main()

Результаты тестирования для примеров из текста лабораторной:



Результаты тестирования:



Пояснение. Рекурсивная функция удаления поддеревьев. Сначала ищем элемент, если нашли, то присваиваем значение None для него и всех поддеревьев, если они есть. Счетчик глобальный, изменяем его на шаге выше.

# **10 задача. Проверка корректности (2 балла)**

Задание. 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 ≤ i ≤ N) находится описание i-ой вершины, состоящее из трех чисел Ki , Li , Ri , разделенных пробелами – ключа Ki в i-ой вершине, номера левого Li ребенка i-ой вершины (i < Li ≤ N или Li = 0, если левого ребенка нет) и номера правого Ri ребенка i-ой вершины (i < Ri ≤ N или Ri = 0, если правого ребенка нет).

• Ограничения на входные данные. 0 ≤ N ≤ 2 · 10^5, |Ki | ≤ 10^9.

• На 60% от при 0 ≤ N ≤ 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 sys  
  
sys.setrecursionlimit(3500)

class BSTNode:  
 def \_\_init\_\_(self, key):  
 self.data = key  
 self.left = None  
 self.right = None  
  
  
def is\_bst(root, l=None, r=None):  
 if root is None:  
 return True  
  
 if l is not None and root.data <= l.data:  
 return False  
  
 if r is not None and root.data >= r.data:  
 return False  
  
 return is\_bst(root.left, l, root) and is\_bst(root.right, root, r)  
  
  
def main():  
 f = open('input.txt')  
 f2 = open('output.txt', 'w')  
 n = int(f.readline())  
 if n == 0:  
 f2.write("YES")  
 else:  
 tree\_list = [BSTNode(0) for i in range(n)]  
 for i in range(n):  
 val, left, right = map(int, f.readline().split())  
 left -= 1  
 right -= 1  
 tree\_list[i].data = val  
 if left != -1:  
 tree\_list[i].left = tree\_list[left]  
 if right != -1:  
 tree\_list[i].right = tree\_list[right]  
  
 if is\_bst(tree\_list[0]):  
 f2.write("YES")  
 else:  
 f2.write("NO")  
  
  
main()

Результаты тестирования:



Пояснение.

# **12 задача. Проверка сбалансированности (2 балла)**

Задание. 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 ≤ B(V ) ≤ 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 ≤ i ≤ N) находится описание i-ой вершины, состоящее из трех чисел Ki , Li , Ri , разделенных пробелами – ключа Ki в i-ой вершине, номера левого Li ребенка i-ой вершины (i < Li ≤ N или Li = 0, если левого ребенка нет) и номера правого Ri ребенка i-ой вершины (i < Ri ≤ N или Ri = 0, если правого ребенка нет). Все ключи различны. Гарантируется, что данное дерево является деревом поиска.

• Ограничения на входные данные. 0 ≤ N ≤ 2 · 105 , |Ki | ≤ 109 .

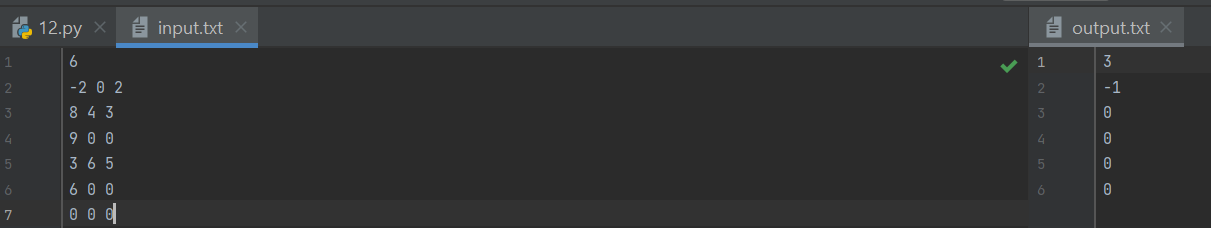
• Формат вывода / выходного файла (output.txt). Для i-ой вершины в i-ой строке выведите одно число – баланс данной вершины.

• Ограничение по времени. 2 сек.

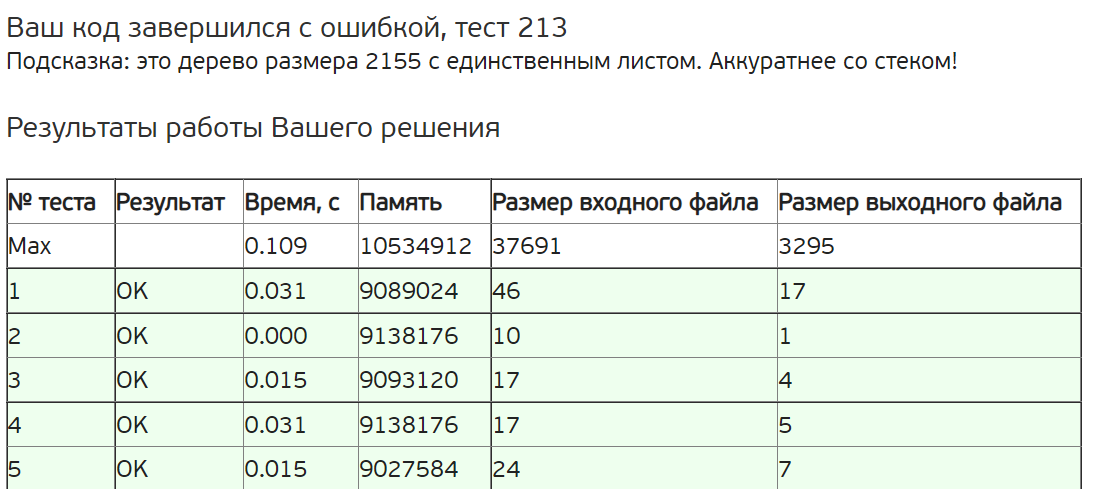
• Ограничение по памяти. 256 мб.

def height(root):  
 lh = 0  
 if root[1] != 0:  
 lh = height(tree[root[1] - 1])  
 rh = 0  
 if root[2] != 0:  
 rh = height(tree[root[2] - 1])  
 return max(lh, rh) + 1  
  
  
def balance(root):  
 lh = 0  
 if root[1] != 0:  
 lh = height(tree[root[1] - 1])  
 rh = 0  
 if root[2] != 0:  
 rh = height(tree[root[2] - 1])  
 return rh - lh  
  
  
with open('input.txt') as f:  
 n = int(f.readline())  
 tree = []  
 for i in range(n):  
 k, l, r = map(int, f.readline().split())  
 tree.append([k, l, r])  
  
with open('output.txt', 'w') as f:  
 for i in range(n):  
 f.write(str(balance(tree[i])) + ('' if i == n - 1 else '\n'))

Результаты тестирования для примеров из текста лабораторной:



Результаты тестирования для правой границы:



Пояснение. Рекурсивная функция получения высоты. При этом проверяется есть ли левый или правый ребенок. Функция получения баланса вызывается для каждого элемента. Баланс высчитывается как разность высот правого и левого поддеревьев.

**Выводы**

В данной работе изучили жадные алгоритмы и продолжили изучать динамическое программирование. Узнали, что такое безопасный ход. Познакомились с задачей о рюкзаке. Заполоняли точки минимальным количеством отрезков и наоборот. Решали задачи разбивая их на подзадачи.