САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №2

по курсу «Алгоритмы и структуры данных»

Тема: Двоичные деревья поиска

Вариант 7

Выполнил:

Крылов Михаил Максимович

К3240

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

**Содержание отчета**

Оглавление

[**Содержание отчета** 2](#_Toc179017151)

[**Задачи по варианту** 3](#_Toc179017152)

[**Задача №1. Обход двоичного дерева [5 s, 512 Mb, 1 балл]** 3](#_Toc179017153)

[**Задача №12. Проверка сбалансированности [2 s, 256 Mb, 2 балла]** 7](#_Toc179017154)

[**Задача №16. *K*-й максимум [2 s, 512 Mb, 3 балла][\*Замена]** 10](#_Toc179017155)

[**Вывод** 13](#_Toc179017156)

**Задачи по варианту**

**Задача №1. Обход двоичного дерева [5 s, 512 Mb, 1 балл]**

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.

Листинг кода

def postorder(root, data, result=[]):

    if data[root][0] != -1:

        postorder(data[root][0], data, result)

    if data[root][1] != -1:

        postorder(data[root][1], data, result)

    result.append(str(root))

    return result

def preorder(root, data, result=[]):

    result.append(str(root))

    if data[root][0] != -1:

        preorder(data[root][0], data, result)

    if data[root][1] != -1:

        preorder(data[root][1], data, result)

    return result

def inorder(root, data, result=[]):

    if data[root][0] != -1:

        inorder(data[root][0], data, result)

    result.append(str(root))

    if data[root][1] != -1:

        inorder(data[root][1], data, result)

    return result

def main():

    with open('input.txt') as file:

        N = int(file.readline())

        root, l, r = map(int, file.readline().split())

        data = {}

        data[root] = (l, r)

        for \_ in range(1, N):

            node, l, r = map(int, file.readline().split())

            data[node] = (l, r)

    result\_inorder = inorder(root, data)

    result\_preorder = preorder(root, data)

    result\_postorder = postorder(root, data)

    with open('output.txt', 'w') as file:

        file.write(' '.join(result\_inorder)+'\n')

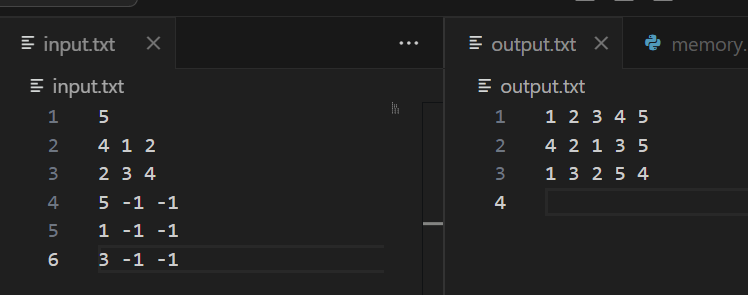
        file.write(' '.join(result\_preorder)+'\n')

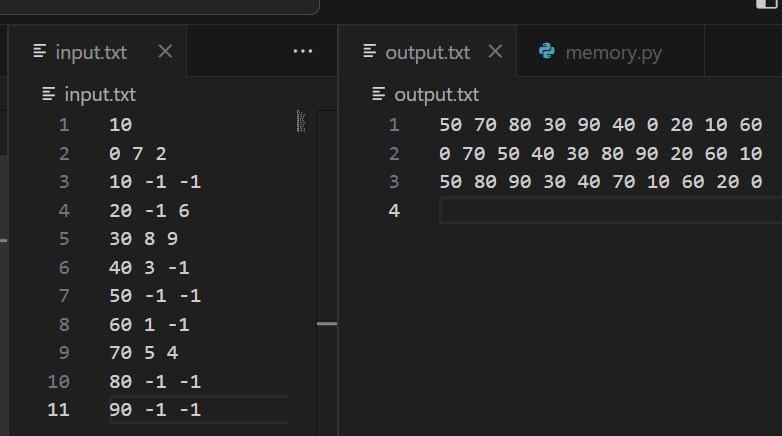
        file.write(' '.join(result\_inorder)+'\n')

if \_\_name\_\_ == '\_\_main\_\_':

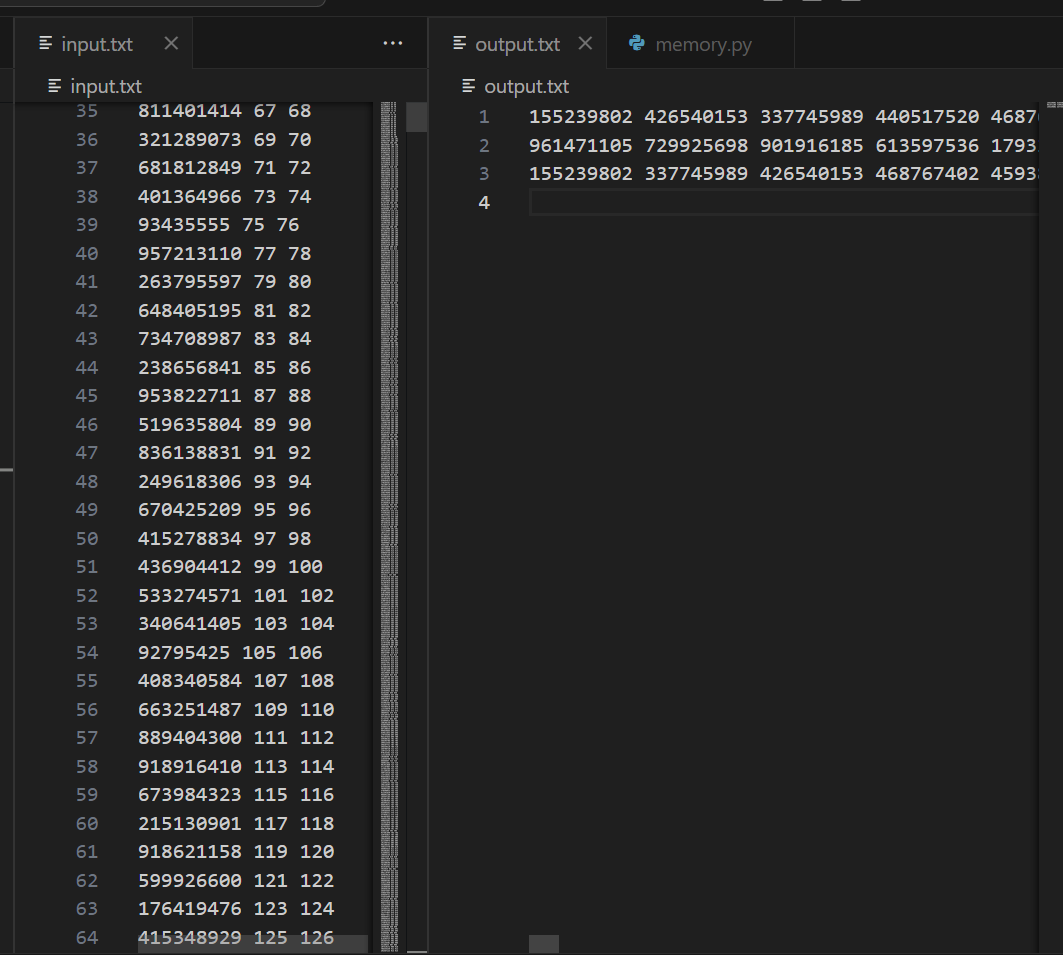
    main()

Запускаем функцию main. В ней считываем данные, каждый узел и его детей записываем в словарь. Далее, запускаем 3 функции. Каждая из функции представляет бинарное дерево в одном из видов – in order; pre order; in order. Работают функции с помощью рекурсии, таким образом получаем доступ к каждому узлу.





Результат работы кода на максимальных и минимальных значениях:



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.001943 секунды | 0.017133 МБ |
| Пример из задачи | 0.002013 секунды | 0.018333 МБ |
| Пример из задачи | 0.007544 секунды | 0.018488 МБ |
| Верхняя граница диапазона значений входных данных из текста задачи | 1.623032 секунды | 28.696539 МБ |

Вывод по задаче: попрактиковались в реализации создании двоичного дерева и разные его обходы.

**Задача №12. Проверка сбалансированности [2 s, 256 Mb, 2 балла]**

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.

Листинг кода

def calculate\_balance(root, data, result=[]):

    if root == 0:

        return 0, 0

    left\_height, \_ = calculate\_balance(data[root][1], data, result)

    right\_height, \_ = calculate\_balance(data[root][2], data, result)

    balance = right\_height - left\_height

    result.append(str(balance))

    return max(left\_height, right\_height) + 1, result

def main():

    with open('input.txt') as file:

        N = int(file.readline())

        data = {}

        for i in range(N):

            node, l, r = map(int, file.readline().split())

            data[i+1] = (node, l, r)

    result = calculate\_balance(1, data)

    result[1].reverse()

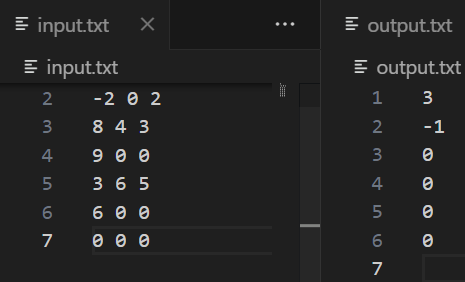
    with open('output.txt', 'w') as file:

        file.write('\n'.join(result[1]) + '\n')

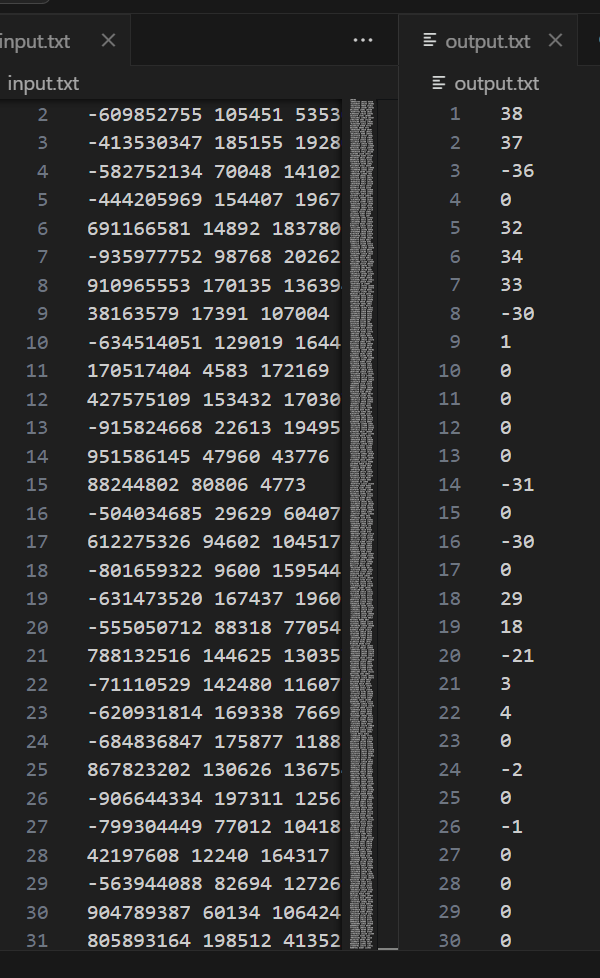
if \_\_name\_\_ == '\_\_main\_\_':

    main()

При запуске программы записываем все данные в словарь. Далее, запускаем рекурсию с 1-го элемента. В рекурсии проверяем, существует ли узел, если нет – возвращаем 0. Далее запускаем рекурсию для дочерних элементов. Высчитываем баланс, записываем в список. В конце возвращаем максимальную высоту дочерних элементов + 1 данного узла.



Результат работы кода на максимальных и минимальных значениях:



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.001104 секунды | 0.016319 МБ |
| Пример из задачи | 0.001013 секунды | 0.018349 МБ |
| Верхняя граница диапазона значений входных данных из текста задачи | 2.566 секунды | 43.003 МБ |

**Задача №16. *K*-й максимум [2 s, 512 Mb, 3 балла][\*Замена]**

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 ci и ki – тип и аргумент команды соответственно. Поддерживаемые команды: – +1 (или просто 1): Добавить элемент с ключом ki . – 0 : Найти и вывести ki-й максимум. – -1 : Удалить элемент с ключом ki . 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 ≤ 100000, |ki | ≤ 109 .

• Формат вывода / выходного файла (output.txt). Для каждой команды нулевого типа в выходной файл должна быть выведена строка, содержащая единственное число – ki-й максимум.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 512 мб.

Листинг кода

class Struct:

    def \_\_init\_\_(self) -> None:

        self.data = []

    def add(self, key: int) -> None:

        index = self.binary\_search\_insert\_index(key)

        self.data.insert(index, key)

    def binary\_search\_insert\_index(self, key: int) -> int:

        low, high = 0, len(self.data)

        while low < high:

            mid = (low + high) // 2

            if self.data[mid] < key:

                low = mid + 1

            else:

                high = mid

        return low

    def remove(self, key: int):

        index = self.binary\_search\_insert\_index(key)

        self.data.pop(index)

    def k(self, key: int):

        return str(self.data[-key])

def main():

    with open('input.txt') as file:

        N = int(file.readline())

        struct = Struct()

        result = []

        for \_ in range(N):

            c, k = map(str, file.readline().split())

            k = int(k)

            if c in ['1', '+1']:

                struct.add(k)

            elif c == '-1':

                struct.remove(k)

            else:

                result.append(

                    struct.k(k)

                )

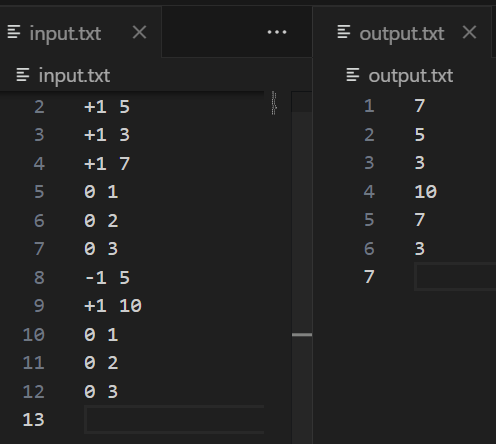
    with open('output.txt', 'w') as file:

        file.write('\n'.join(result) + '\n')

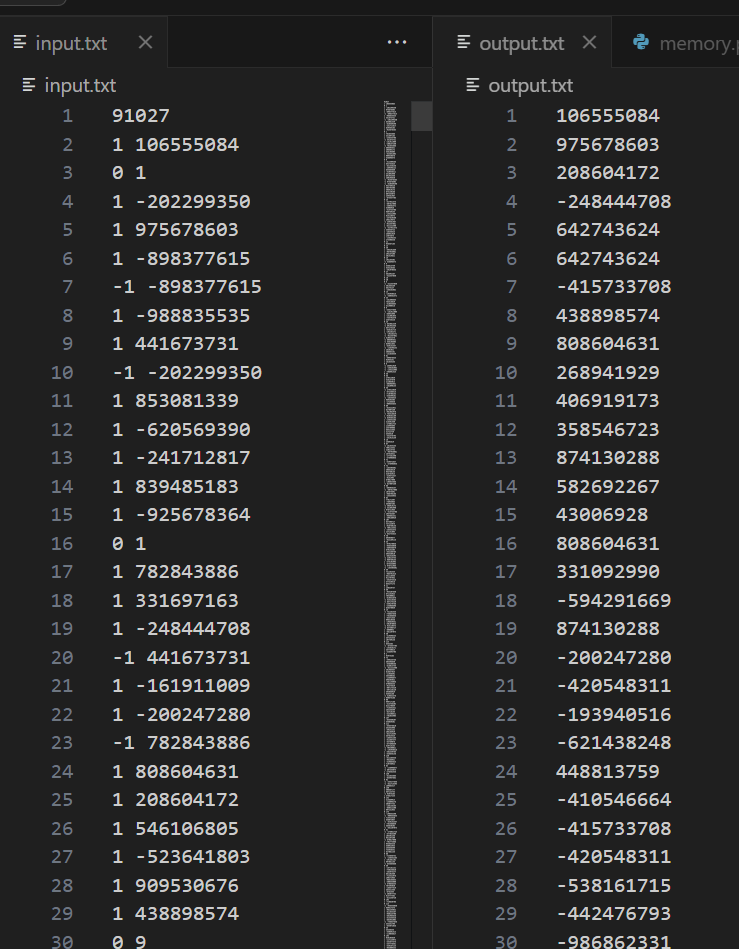
if \_\_name\_\_ == '\_\_main\_\_':

    main()

Реализовываем структуру, которая хранит в себе в списке все данные. Добавление реализовано через бинарный поиск, для того, чтоб хранить все элементы в сортированном порядке. Удаление элемента также реализовано с помощью бинарного поиска.



Результат работы кода на максимальных и минимальных значениях:



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.001 секунды | 0.016 МБ |
| Пример из задачи | 0.001 секунды | 0.018 МБ |
| Верхняя граница диапазона значений входных данных из текста задачи | 1.971 секунды | 3.188 МБ |

Вывод по задаче: реализовали структуру, которая хранит данные и использует бинарный поиск для большей эффективности.

**Вывод**

Данная лабораторная работа закрепила знания работы с бинарными деревьями.