

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
Задачи по варианту.....	3
Задача №1. Обход двоичного дерева [5 s, 512 Mb, 1 балл]	3
Задача №12. Проверка сбалансированности [2 s, 256 Mb, 2 балла].....	7
Задача №16. К-й максимум [2 s, 512 Mb, 3 балла][*Замена]	10
Вывод.....	13

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 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. $1 \leq n \leq 10^5$, $0 \leq K_i \leq 10^9$, $-1 \leq L_i, R_i \leq n-1$. Гарантируется, что данное дерево является двоичным деревом. В частности, если $L_i \neq -1$ и $R_i \neq -1$, то $L_i \neq R_i$. 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.

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)

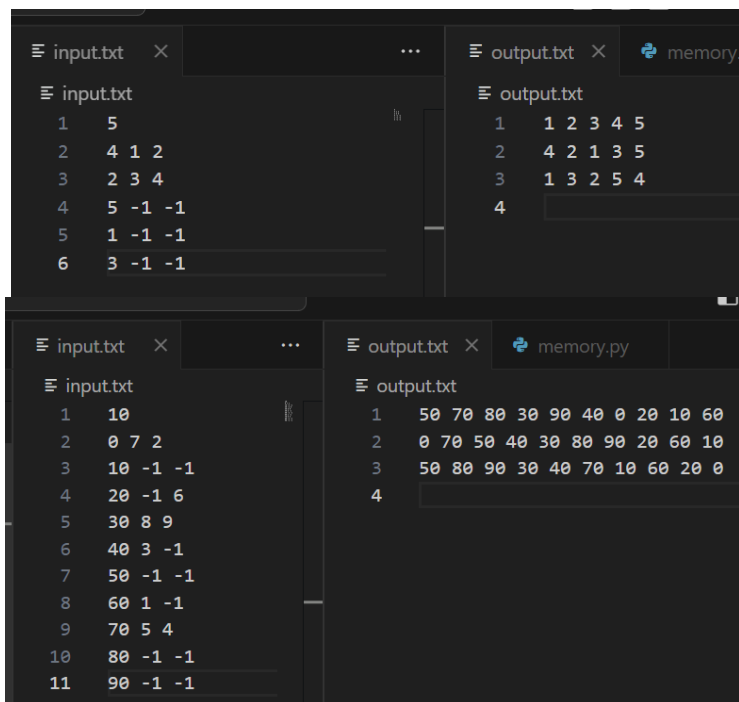
        result_inorder = inorder(root, data)
        result_preorder = preorder(root, data)
        result_postorder = postorder(root, data)

        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')

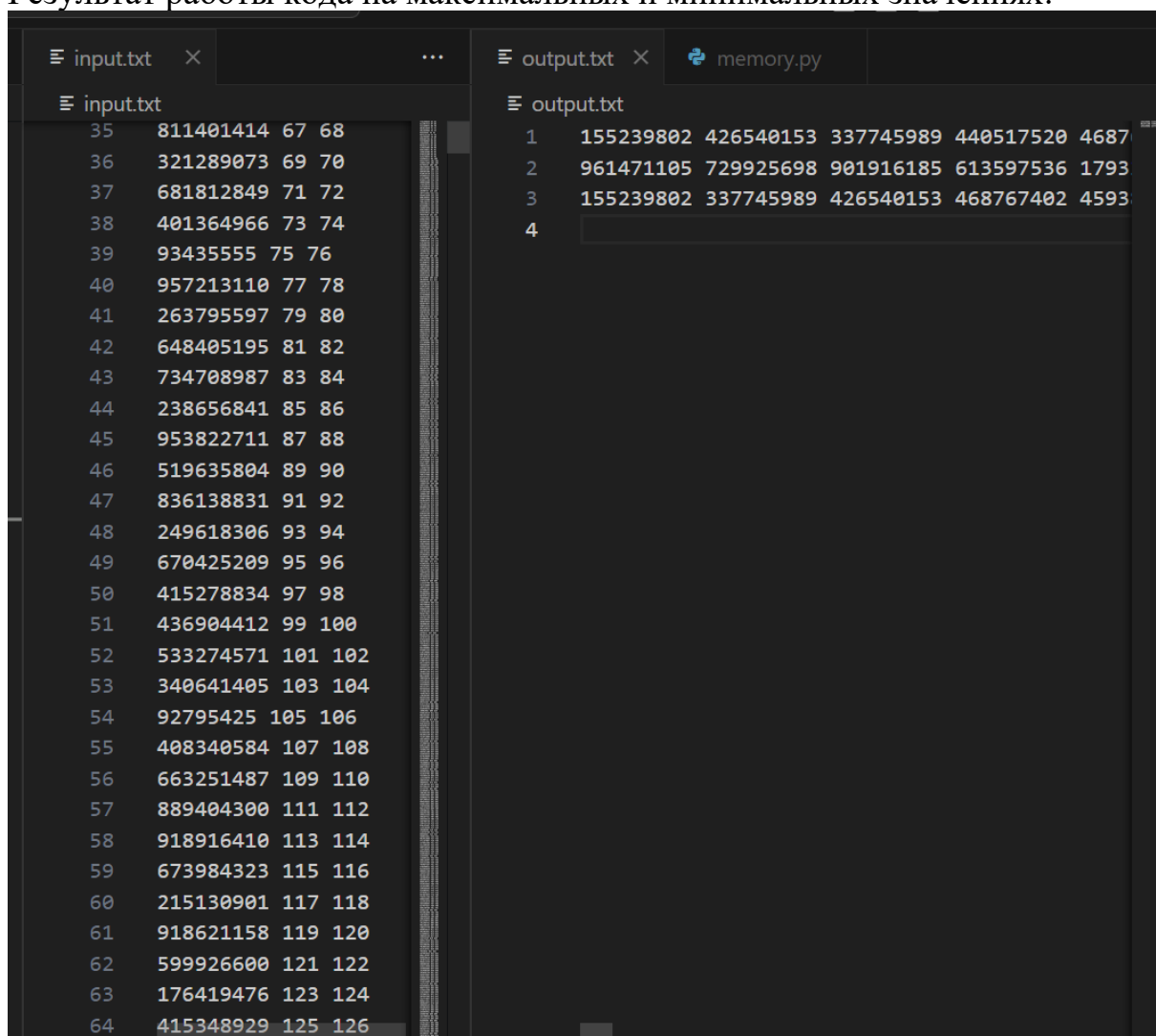
if __name__ == '__main__':
    main()

```

Запускаем функцию main. В ней считываем данные, каждый узел и его детей записываем в словарь. Далее, запускаем 3 функции. Каждая из функций 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 ($1 \leq i \leq N$) 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 ($i < L_i \leq N$ или $L_i = 0$, если левого ребенка нет) и номера правого R_i ребенка i -ой вершины ($i < R_i \leq N$ или $R_i = 0$, если правого ребенка нет). Все ключи различны. Гарантируется, что данное дерево является деревом поиска.
- Ограничения на входные данные. $0 \leq N \leq 2 \cdot 10^5$, $|K_i| \leq 10^9$.
- 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.

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. Вычисляем баланс, записываем в список. В конце возвращаем максимальную высоту дочерних элементов + 1 данного узла.

input.txt	output.txt
2 -2 0 2	1 3
3 8 4 3	2 -1
4 9 0 0	3 0
5 3 6 5	4 0
6 6 0 0	5 0
7 0 0 0	6 0
	7 0

Результат работы кода на максимальных и минимальных значениях:

input.txt	output.txt
2 -609852755 105451 5353	1 38
3 -413530347 185155 1928	2 37
4 -582752134 70048 14102	3 -36
5 -444205969 154407 1967	4 0
6 691166581 14892 183780	5 32
7 -935977752 98768 20262	6 34
8 910965553 170135 13639	7 33
9 38163579 17391 107004	8 -30
10 -634514051 129019 1644	9 1
11 170517404 4583 172169	10 0
12 427575109 153432 17030	11 0
13 -915824668 22613 19495	12 0
14 951586145 47960 43776	13 0
15 88244802 80806 4773	14 -31
16 -504034685 29629 60407	15 0
17 612275326 94602 104517	16 -30
18 -801659322 9600 159544	17 0
19 -631473520 167437 1960	18 29
20 -555050712 88318 77054	19 18
21 788132516 144625 13035	20 -21
22 -71110529 142480 11607	21 3
23 -620931814 169338 7669	22 4
24 -684836847 175877 1188	23 0
25 867823202 130626 13675	24 -2
26 -906644334 197311 1256	25 0
27 -799304449 77012 10418	26 -1
28 42197608 12240 164317	27 0
29 -563944088 82694 12726	28 0
30 904789387 60134 106424	29 0
31 805893164 198512 41352	30 0

	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. К-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 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 k_i -го максимума, он существует.
- Ограничения на входные данные. $n \leq 100000$, $|k_i| \leq 109$.
- 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.

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 отсортированном порядке. Удаление элемента также реализовано с помощью бинарного поиска.

input.txt	output.txt
2 +1 5	1 7
3 +1 3	2 5
4 +1 7	3 3
5 0 1	4 10
6 0 2	5 7
7 0 3	6 3
8 -1 5	7
9 +1 10	
10 0 1	
11 0 2	
12 0 3	
13	

Результат работы кода на максимальных и минимальных значениях:

```

input.txt
1 91027
2 1 106555084
3 0 1
4 1 -202299350
5 1 975678603
6 1 -898377615
7 -1 -898377615
8 1 -988835535
9 1 441673731
10 -1 -202299350
11 1 853081339
12 1 -620569390
13 1 -241712817
14 1 839485183
15 1 -925678364
16 0 1
17 1 782843886
18 1 331697163
19 1 -248444708
20 -1 441673731
21 1 -161911009
22 1 -200247280
23 -1 782843886
24 1 808604631
25 1 208604172
26 1 546106805
27 1 -523641803
28 1 909530676
29 1 438898574
30 0 9

output.txt
1 106555084
2 975678603
3 208604172
4 -248444708
5 642743624
6 642743624
7 -415733708
8 438898574
9 808604631
10 268941929
11 406919173
12 358546723
13 874130288
14 582692267
15 43006928
16 808604631
17 331092990
18 -594291669
19 874130288
20 -200247280
21 -420548311
22 -193940516
23 -621438248
24 448813759
25 -410546664
26 -415733708
27 -420548311
28 -538161715
29 -442476793
30 -986862331
  
```

	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.