

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 21

Выполнила:
Савченко А.С.
К3141

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Обход двоичного дерева [1 баллов]	3
Задача №12. Проверка сбалансированности [2 баллов]	7
Задача №16. Задача. К-й максимум [3 баллов]	8
Дополнительные задачи	15
Задача №2. Гирлянда [1 баллов]	15
Задача №3. Простейшее BST [1 баллов]	18
Задача №4. Простейший неявный ключ [1 баллов]	23
Задача №5. Простое двоичное дерево поиска [1 баллов]	27
Задача №6. Оpozнание двоичного дерева поиска [1,5 баллов]	34
Задача №7. Оpozнание двоичного дерева поиска (усложненная версия) [2.5 баллов]	38
Задача №8. Высота дерева возвращается [2 баллов]	41
Задача №9. Удаление поддеревьев [2 баллов]	46
Вывод	50

Задачи по варианту

Задача №1. Обход двоичного дерева [1 баллов]

Текст задачи:

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 $0, 1, \dots, n - 1$ 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.

- Примеры:

input	output.txt	input	output.txt
5	1 2 3 4 5	10	50 70 80 30 90 40 0 20 10 60
4 1 2	4 2 1 3 5	0 7 2	0 70 50 40 30 80 90 20 60 10
2 3 4	1 3 2 5 4	10 -1 -1	50 80 90 30 40 70 10 60 20 0
5 -1 -1		20 -1 6	
1 -1 -1		30 8 9	
3 -1 -1		40 3 -1	
		50 -1 -1	
		60 1 -1	
		70 5 4	
		80 -1 -1	
		90 -1 -1	

Листинг кода

```
from sys import setrecursionlimit

setrecursionlimit(10 ** 6)

from typing import List

class Node:
    def __init__(self, key: int) -> None:
```

```

        self.left = None
        self.right = None
        self.key = key

def in_order(root: Node, file_output) -> None:
    if root:
        in_order(root.left, file_output)
        file_output.write(str(root.key) + ' ')
        in_order(root.right, file_output)

def pre_order(root: Node, file_output) -> None:
    if root:
        file_output.write(str(root.key) + ' ')
        pre_order(root.left, file_output)
        pre_order(root.right, file_output)

def post_order(root: Node, file_output) -> None:
    if root:
        post_order(root.left, file_output)
        post_order(root.right, file_output)
        file_output.write(str(root.key) + ' ')

def build_tree(nodes: List[List[int]]) -> Node:
    tree = [Node(key) for key, _, _ in nodes]
    for i, (key, left, right) in enumerate(nodes):
        if left != -1:
            tree[i].left = tree[left]
        if right != -1:
            tree[i].right = tree[right]
    return tree[0]

```

```
def main() -> None:
    with open('input.txt', 'r', encoding='utf-8') as
input_file, open('output.txt', 'w',
encoding='utf-8') as output_file:
        n = int(input_file.readline().strip())
                nodes      =      [list(map(int,
input_file.readline().split())) for _ in range(n)]

        root = build_tree(nodes)

        in_order(root, output_file)
        output_file.write('\n')
        pre_order(root, output_file)
        output_file.write('\n')
        post_order(root, output_file)
        output_file.write('\n')
```

Текстовое объяснение решения.

Реализуем функции обхода двоичного дерева. Центрированный обход: левое поддерево, корень, правое поддерево. Прямой обход: корень, левое поддерево, правое поддерево. Обратный обход: левое поддерево, правое поддерево, корень.

Во избежание переполнения стека увеличим лимит рекурсии. И вместо создания списка в функциях обхода, будем сразу производить запись в выходной файл.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

input.txt		✓	output.txt	
1	5		1	1 2 3 4 5
2	4 1 2		2	4 2 1 3 5
3	2 3 4		3	1 3 2 5 4
4	5 -1 -1		4	
5	1 -1 -1			
6	3 -1 -1			

```
input.txt x      output.txt x
1 10 ✓ 1 50 70 80 30 90 40 0 20 10 60
2 0 7 2 2 0 70 50 40 30 80 90 20 60 10
3 10 -1 -1 3 50 80 90 30 40 70 10 60 20 0
4 20 -1 6 4
5 30 8 9
6 40 3 -1
7 50 -1 -1
8 60 1 -1
9 70 5 4
10 80 -1 -1
11 90 -1 -1
```

Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)

```
input.txt x      outp
1 1 ✓ 1 0
2 0 -1 -1 2 0
3 3 3 0
4 4
```

```
input.txt x      output.txt x
1 10 ✓ 1 50 70 80 30 90 40 0 20 10 60
2 0 7 2 2 0 70 50 40 30 80 90 20 60 10
3 10 -1 -1 3 50 80 90 30 40 70 10 60 20 0
4 20 -1 6 4
5 30 8 9
6 40 3 -1
7 50 -1 -1
8 60 1 -1
9 70 5 4
10 80 -1 -1
11 90 -1 -1
12
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00048489996697	15.53515625
Пример из задачи	0.000461199961	15.29296875
Пример из задачи	0.0006100999	15.28515625
Верхняя граница диапазона значений	0.2153499000	20.19140625

ВХОДНЫХ ДАННЫХ ИЗ текста задачи		
------------------------------------	--	--

Вывод по задаче:

В этой задаче я реализовала три основных способа рекурсивного обхода двоичного дерева(центрированный, прямой и обратный). При тестах на максимальных значениях удалось избежать проблем с переполнением стека.

Задача №12. Проверка сбалансированности [2 баллов]

Текст задачи:

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:

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

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 $(1 \leq i \leq N)$ 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.

- **Пример:**

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

- Проверить можно по [ссылке](#), OpenEdu, курс "Алгоритмы программирования и структуры данных 7 неделя, 1 задача.

Листинг кода. (именно листинг, а не скрины)

```
class Node:
def __init__(self, key):
```

```
        self.key = key
        self.left = None
        self.right = None

class BST:
    def __init__(self):
        self.root = None

    def add_node(self, key):
        if self.root is None:
            self.root = Node(key)
        else:
            self._add_node(self.root, key)

    def _add_node(self, node, key):
        if key < node.key:
            if node.left is None:
                node.left = Node(key)
            else:
                self._add_node(node.left, key)
        else:
            if node.right is None:
                node.right = Node(key)
            else:
                self._add_node(node.right, key)

    def height(self, node):
        if node is None:
            return 0
        return 1 + max(self.height(node.left),
self.height(node.right))

    def balance(self, node):
        if node is None:
```



```

        return 0
        left_height = self.height(node.left) if
node.left else 0
        right_height = self.height(node.right) if
node.right else 0
        return right_height - left_height

with open("input.txt", "r") as in_file,
open("output.txt", "w") as out_file:
    n = int(in_file.readline())

    if n == 0:
        out_file.write(str(0))
    else:

        nodes = [int(in_file.readline().split()[0]) for
_ in range(n)]

        tree = BST()

        for node in nodes:
            tree.add_node(node)

        queue = [tree.root]
        while queue:
            node = queue.pop(0)
            out_file.write(f"{tree.balance(node)}\n")
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

```

Текстовое объяснение решения.

Используя код из некоторых предыдущих задачи, создадим дерево добавлением узлов, а конкретно для этой задачи напомним метод `balance`, который определяет баланс узла как разности высот левого и правого поддеревьев. С помощью очереди совершим обход дерева по уровням, для каждого узла будем сразу печатать его баланс в файл.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

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

Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)

input.txt	output.txt
1 20000	1 0
2 183136364 0 -494203426	2 -3
3 -494203426 0 817857418	3 -9
4 817857418 0 179430000	4 11
5 179430000 0 -59277933	5 -17
6 -59277933 0 854349792	6 1
7 854349792 0 12431077	7 5
8 12431077 0 -262825416	8 -4
9 -262825416 0 562473343	9 5
10 562473343 0 184269356	10 2
11 184269356 0 584562684	11 -6
12 584562684 0 -141656655	12 21
13 -141656655 0 931073633	13 15
14 931073633 0 20502891	14 -3
15 20502891 0 891037111	15 -1
16 891037111 0 401967286	16 0
17 401967286 0 337841713	17 4
18 337841713 0 752359258	18 3
19 752359258 0 590742666	19 9
20 590742666 0 -204953153	20 -1
21 -204953153 0 687174744	21 3
22 687174744 0 857763096	22 3

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.0004600000102072	14.953125

входных данных из текста задачи		
Пример из задачи	0.000514699961058	14.85546875
Верхняя граница диапазона значений входных данных из текста задачи	0.168271000031381	17.28515625

Вывод по задаче:

Мне понравилась задача, сама идея вычисления баланса узлов показалась интересной, рада что все работает даже на максимальных тестах.

Задача №16. Задача. К-й максимум [3 баллов]

Текст задачи:

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 k_i -го макс-сима, 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.
- Пример:

input.txt	output.txt
11	7
+1 5	5
+1 3	3
+1 7	10
0 1	7
0 2	3
0 3	
-1 5	
+1 10	
0 1	
0 2	
0 3	

Листинг кода. (именно листинг, а не скрины)

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

        self.size = 1

class BST:
    def __init__(self):
```

```

        self.root = None

    def get_size(self, node):
        if node is None:
            return 0
        return node.size

    def _inserting(self, root, key):
        if key < root.val:
            if root.left is None:
                root.left = Node(key)
            else:
                root.left = self._inserting(root.left,
key)

        elif key > root.val:
            if root.right is None:
                root.right = Node(key)
            else:
                root.right =
self._inserting(root.right, key)
            root.size = 1 + self.get_size(root.left) +
self.get_size(root.right)
        return root

    def insert(self, key):
        if self.root is None:
            self.root = Node(key)
        else:
            self._inserting(self.root, key)

    def _min_val_node(self, node):
        current = node
        while current.left is not None:
            current = current.left
        return current

```

```

def _delete_node(self, root, key):
    if root is None:
        return root
    if key < root.val:
        root.left = self._delete_node(root.left,
key)
    elif key > root.val:
        root.right = self._delete_node(root.right,
key)
    else:
        if root.left is None:
            return root.right
        elif root.right is None:
            return root.left
        temp = self._min_val_node(root.right)
        root.val = temp.val
        root.right = self._delete_node(root.right,
temp.val)
        root.size = 1 + self.get_size(root.left) +
self.get_size(root.right)
        return root

def delete(self, key):
    self.root = self._delete_node(self.root, key)

def _find_k_max(self, root, k):
    if root is None:
        return None
    right_size = self.get_size(root.right)
    if right_size + 1 == k:
        return root.val
    elif k <= right_size:
        return self._find_k_max(root.right, k)
    else:

```

```

        return self._find_k_max(root.left, k -
right_size - 1)

    def find_k_max(self, k):
        return self._find_k_max(self.root, k)

with open('input.txt', 'r', encoding='utf-8') as
input_file, open('output.txt', 'w',
encoding='utf-8') as output_file:
    n = int(input_file.readline().strip())
    bst = BST()
    results = []

    for line in input_file:
        if line.startswith('+1'):
            _, x = line.split()
            bst.insert(int(x))
        elif line.startswith('-1'):
            _, x = line.split()
            bst.delete(int(x))
        elif line.startswith('0'):
            _, x = line.split()
            result = bst.find_k_max(int(x))
            if result is not None:
                results.append(result)

    if results is not None:
        for elem in results:
            output_file.write(f'{elem}\n')
    else:
        output_file.write(str(0))

```

Текстовое объяснение решения.

Задача довольно объемная, так что хорошо, что в предыдущих было что-то похожее, так что часть кода можно взять оттуда.

Из предыдущих задач берем код для вставки и удаления узлов. А в функции поиска к-го максимума: если дерево пустое возвращаем none, в остальных случаях вычисляем размер правого поддерева, и проверяем, если размер правого поддерева + 1 \leq k, то текущий узел это k-й максимум, возвращаем его значение.

Если размер правого поддерева + 1 $>$ k, рекурсивно вызываем поиск в левом поддереве.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

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

Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)

input.txt			output.txt		
1	0	✓	1		

Входные данные	Выходные данные
1 100000	1 995203354
2 -1 -834800723	2 998801864
3 -1 -212047746	3 997811529
4 -1 -891755724	4 997536682
5 +1 -499346656	5 998834277
6 -1 -219685264	6 999734773
7 +1 110879156	7 999336868
8 -1 -556558051	8 999791396
9 +1 645660382	9 999364256
10 -1 -664413414	10 999791396
11 -1 -775770936	11
12 -1 -824354196	
13 +1 -809309186	
14 +1 378160969	
15 +1 688530042	
16 +1 -667126224	
17 +1 740242917	
18 -1 -428110398	
19 -1 399748678	
20 +1 714394971	

67268 +1 -242484176	3 997811529
67269 -1 153953492	4 997536682
67270 +1 43580544	5 998834277
67271 +1 808854709	6 999734773
67272 +1 -43086418	7 999336868
67273 -1 670762267	8 999791396
67274 +1 705396767	9 999364256
67275 +1 781313983	10 999791396
67276 -1 -153201620	11
67277 0 6	
67278 -1 453631321	
67279 +1 978918134	
67280 -1 535522924	
67281 +1 -663862370	

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00039820000529289	14.90234375
Пример из задачи	0.000646500033326	15.04296875
Верхняя граница диапазона значений входных данных из текста задачи	0.40682460006792	16.19921875

Вывод по задаче:

Задача напомнила мне 4-ю(и 5), так что повезло и многое для решения этой задачи я взяла из четвертой. Алгоритм работает эффективно и поддерживает вставку, удаление элементов и поиск k -го максимума.

Дополнительные задачи

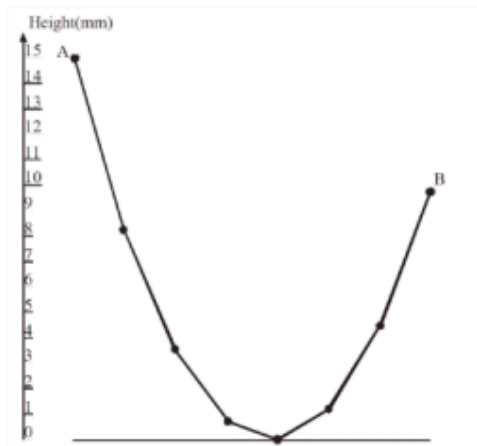
Задача №2. Гирлянда [1 баллов]

Текст задачи.

Гирлянда состоит из n лампочек на общем проводе. Один её конец закреплён на заданной высоте A мм ($h_1 = A$). Благодаря силе тяжести гирлянда прогибается: высота каждой неконцевой лампы на 1 мм меньше, чем средняя высота ближайших соседей ($h_i = \frac{h_{i-1} + h_{i+1}}{2} - 1$ для $1 < i < N$).

Требуется найти минимальное значение высоты второго конца B ($B = h_n$), такое что для любого $\epsilon > 0$ при высоте второго конца $B + \epsilon$ для всех лампочек выполняется условие $h_i > 0$. 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.** $3 \leq n \leq 1000$, n – целое, $10 \leq A \leq 1000$, 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.
- **Примеры:**

input.txt	output.txt
8 15	9.75
692 532.81	446113.34434782615

Листинг кода. (именно листинг, а не скрины)

```
def garland(n, a):
    lights_g = [0] * n
    lights_g[0] = a

    low, high = 0, a

    while high - low > 10 ** -12:
        lights_g[1] = (low + high) / 2
        flag_not_ground = True

        for i in range(1, n - 1):
            lights_g[i + 1] = 2 * lights_g[i] -
lights_g[i - 1] + 2
            if lights_g[i + 1] < 0:
                flag_not_ground = False
                break

        if flag_not_ground:
            high = lights_g[1]
        else:
            low = lights_g[1]

    return lights_g[-1]

with open('input.txt', 'r', encoding='utf-8') as
input_file, open('output.txt', 'w', encoding='utf-8')
as output_file:
    n, a = map(float, input_file.readline().split())
    n = int(n)
    result = garland(n, a)
    output_file.write(str(result))
```

Текстовое объяснение решения.

Инициализируем список из нулей длины n в нем будем хранить высоты каждой лампочки из гирлянды. Нулевой элемент списка известен, это введенная A - высота первого конца гирлянды. Инициализируем границы бинарного поиска. Запускаем цикл пока разница между ними будет меньше заданной погрешности 10^{-12} . Вычисляем высоту второй лампы как среднее. Инициализируем флаг который будет отвечать за то, чтобы лампочка не касалась земли. Во внутреннем цикле рассчитываем высоту следующей лампочки по формуле $lights_g[i + 1] = 2 * lights_g[i] - lights_g[i - 1] + 2$ (полученной из формулы из условия). Если эта высота оказывается меньше нуля меняем флаг на ложь и прерываем цикл. Проверяем если флаг = true: обновляем правую границу бинарного поиска $high$, в противном случае обновляем левую границу low . Функция возвращает высоту последней лампочки в гирлянде т.е. минимальное значение высоты второго конца B

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

```
input.txt  output.txt
1 8 15 ✓ 1 9.750000000001194
```

```
input.txt  output.txt
1 692 532.81 ✓ 1 446113.3443478297
```

Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)

```
inp  output.txt
1 3 10 ✓ 1 6.821210263296962e-13
```

Прим. $6.821210263296962 \times 10^{-13} = 0.0000000000006821210263296962$

—>это корректный вывод

```
input.txt  output.txt
1 1000 1000 ✓ 1 935814.25
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.0005036999937	14.87890625

входных данных из текста задачи		
Пример из задачи	0.00058079999	14.9375
Пример из задачи	0.00253439997	14.6796875
Верхняя граница диапазона значений входных данных из текста задачи	0.00388360000	14.80859375

Вывод по задаче:

Для этой задачи удобен алгоритм бинарного поиска. Задача интересна также тем что нужно соблюдать заранее заданную погрешность и использовать двоичный поиск. На минимальных значениях алгоритм вывел $6.821210263296962 \times 10^{-13}$, но по сути это $= 0$, так что код работает корректно. Если прям сильно хочется получать сразу корректный вывод можно использовать round при выводе.

Задача №3. Простейшее BST [1 баллов]

Текст 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.** $1 \leq x \leq 10^9$, $1 \leq N \leq 300000$
- **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.
- Пример:

input.txt	output.txt
+ 1	3
+ 3	3
+ 3	0
> 1	2
> 2	
> 3	
+ 2	
> 1	

Листинг кода. (именно листинг, а не скрины)

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

class BST:
    def __init__(self):
        self.root = None

    def insert(self, key):
        if self.root is None:
            self.root = Node(key)
        else:
            self.inserting(self.root, key)
```

```

def inserting(self, root, key):
    if key < root.val:
        if root.left is None:
            root.left = Node(key)
        else:
            self.inserting(root.left, key)
    elif key > root.val:
        if root.right is None:
            root.right = Node(key)
        else:
            self.inserting(root.right, key)

def finding(self, root, key, min_greater):
    if root is None:
        return 0 if min_greater == float('inf')
    else min_greater
        if root.val > key:
            min_greater = min(min_greater, root.val)
            return self.finding(root.left, key,
min_greater)
        else:
            return self.finding(root.right, key,
min_greater)

    def find_min(self, key):
        return self.finding(self.root, key,
float('inf'))

with open('input.txt', 'r', encoding='utf-8') as
input_file, open('output.txt', 'w',
encoding='utf-8') as output_file:
    bst = BST()
    result = []

```

```

for line in input_file:
    if line.startswith('+'):
        _, x = line.split()
        bst.insert(int(x))
    elif line.startswith('>'):
        _, x = line.split()
        result.append(bst.find_min(int(x)))

if result is not None:
    for elem in result:
        output_file.write(f'{elem}\n')
else:
    output_file.write(str(0))

```

Текстовое объяснение решения.

Если введенная строка начинается с '+' добавляем x в дерево с помощью метода insert. Рекурсивно находя правильное место для нового узла.

Если запрос начинается с '>' ищем минимальное значение больше x (def find_min)

def finding: Если введенное значение меньше текущего узла, рекурсивно вставляется в левое поддерево. Если введенное значение больше текущего узла, рекурсивно вставляем в правое поддерево.

В finding ищем минимальный элемент в дереве, который больше введенного x. -Если значение текущего узла больше x, обновляет минимальное значение и продолжаем поиск в левом поддереве, чтобы возможно найти ещё меньшее значение. -Если текущее значение меньше или равно x, поиск продолжается в правом поддереве.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

input.txt			output.txt	
1	+ 1	✓	1	3
2	+ 3		2	3
3	+ 3		3	0
4	> 1		4	2
5	> 2		5	
6	> 3			
7	+ 2			
8	> 1			

Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)

The image shows a code editor with two files open: `input.txt` and `output.txt`.

`input.txt` contains the following content:

```
1 > 1
2
3
4
5
6
7
8
```

`output.txt` contains the following content:

```
1 0
2 580281447
3 480227358
4 923675292
5 480227358
6 480227358
7 923675292
8 0
```

A warning message is visible in the background: "The file size (3,87 MB) exceeds the config".

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00058930006343	14.890625
Пример из задачи	0.00050369999371	15.023437
Верхняя граница диапазона значений входных данных из текста задачи	0.9306733999	17.65234375

Вывод по задаче:

Мне понравилась задача. Для этой задачи я реализовала класс простейшего бинарного дерева поиска BST с методами вставки элемента и нахождения минимального элемента больше чем заданного.

Задача №4. Простейший неявный ключ [1 баллов]

Текст задачи.

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.** $1 \leq x \leq 10^9$, $1 \leq N \leq 300000$, 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.
- Пример:

input.txt	output.txt
+ 1	1
+ 4	3
+ 3	4
+ 3	3
? 1	
? 2	
? 3	
+ 2	
? 3	

Листинг кода. (именно листинг, а не скрины)

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

```
        self.size = 1

class BST:
    def __init__(self):
        self.root = None

    def insert(self, key):
        if self.root is None:
            self.root = Node(key)
        else:
            self.inserting(self.root, key)

    def inserting(self, root, key):
        if key < root.val:
            if root.left is None:
                root.left = Node(key)
            else:
                root.left = self.inserting(root.left,
key)

        elif key > root.val:
            if root.right is None:
                root.right = Node(key)
            else:
                root.right = self.inserting(root.right,
key)

        root.size = 1 + self.get_size(root.left) +
self.get_size(root.right)
        return root

    def get_size(self, root):
        if root is None:
            return 0
        return root.size
```

```

def find_k_node(self, root, k):
    if root is None:
        return 0
    left_size = self.get_size(root.left)
    if k == left_size + 1:
        return root.val
    elif k < left_size + 1:
        return self.find_k_node(root.left, k)
    else:
        return self.find_k_node(root.right, k -
left_size - 1)

def find_k(self, k):
    return self.find_k_node(self.root, k)

with open('input.txt', 'r', encoding='utf-8') as
input_file, open('output.txt', 'w',
encoding='utf-8') as output_file:
    bst = BST()
    result = []
    for line in input_file:
        if line.startswith('+'):
            _, x = line.split()
            bst.insert(int(x))
        elif line.startswith('?'):
            _, x = line.split()
            result.append(bst.find_k(int(x)))

    if result is not None:
        for elem in result:
            output_file.write(f'{elem}\n')
    else:

```

```
output_file.write(str(0))
```

Текстовое объяснение решения.

Задача очень похожа на предыдущую так что используем код от нее, добавив некоторые доработки связанные с вычисление размера дерева и заменой знака '>' на '?'.

В класс узла Node добавим атрибут `self.size = 1`. Доработаем `def inserting` так чтобы метод обновлял размер дерева после вставки. Создадим новый метод `get_size`, возвращающий размер поддерева с корнем в узле. Создадим метод поиска к-го элемента по возрастанию, который использует размеры левого и правого поддеревьев и рекурсивно выполняет поиск. Создадим обертку `def find_k` которая будет просто возвращать результат к-й по возрастанию элемент с помощью предыдущего(описанного ранее) метода `find_k_node`.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

input.txt			output.txt	
1	+ 1	✓	1	1
2	+ 4		2	3
3	+ 3		3	4
4	+ 3		4	3
5	? 1		5	
6	? 2			
7	? 3			
8	+ 2			
9	? 3			

Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)

input.txt			output.txt	
1	? 3	✓	1	0

The file size (3,87 MB) e...

198496	+ 452844256	✓	87181	0
198497	+ 307714477		87182	0
198498	+ 377216669		87183	0
198499	? 933549620		87184	0
198500	+ 369040088		87185	0
198501	+ 847445036		87186	0
198502	? 740952151		87187	0
198503	? 522536632		87188	0
198504	? 668638403		87189	0
198505	+ 709966544		87190	0
198506	+ 480782779		87191	0
198507	? 188650960		87192	0
198508	+ 453036630		87193	0
198509	? 902559607		87194	0
198510	+ 311335299		87195	0
198511	+ 894226107		87196	0
198512	? 340776885		87197	0
198513	+ 206987995		87198	0

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000557800056412	15.0078125
Пример из задачи	0.000525200041	15.1015625
Верхняя граница диапазона значений входных данных из текста задачи	1.26666749990545	16.9609375

Вывод по задаче:

Немного доработав код предыдущей задачи мне удалось решить задачу на поиск k-го по возрастанию элемента в двоичном дереве.

Задача №5. Простое двоичное дерево поиска [1 баллов]

Текст задачи.

5 Задача. Простое двоичное дерево поиска [2 s, 512 Mb, 1 балл]

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.** $0 \leq N \leq 100$, $|x_i| \leq 10^9$.
- **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.

- Пример:

input.txt	output.txt
insert 2	true
insert 5	false
insert 3	5
exists 2	3
exists 4	none
next 4	3
prev 4	
delete 5	
next 4	
prev 4	

Листинг кода. (именно листинг, а не скрины)

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

        self.size = 1

class BST:
    def __init__(self):
        self.root = None

    def insert(self, key):
```

```

        if self.root is None:
            self.root = Node(key)
        else:
            self.inserting(self.root, key)

    def inserting(self, root, key):
        if key < root.val:
            if root.left is None:
                root.left = Node(key)
            else:
                root.left = self.inserting(root.left,
key)
        elif key > root.val:
            if root.right is None:
                root.right = Node(key)
            else:
                root.right = self.inserting(root.right,
key)
        root.size = 1 + self.get_size(root.left) +
self.get_size(root.right)
        return root

    def get_size(self, root):
        if root is None:
            return 0
        return root.size

    def exists_node(self, root, key):
        if root is None:
            return False
        if key == root.val:
            return True
        elif key < root.val:
            return self.exists_node(root.left, key)
        else:

```



```

        return self.exists_node(root.right, key)

def exists(self, key):
    return self.exists_node(self.root, key)

def next_node(self, root, key):
    if root is None:
        return "none"
    if key < root.val:
        left_next = self.next_node(root.left, key)
        if left_next != "none":
            return left_next
        else:
            return root.val
    else:
        return self.next_node(root.right, key)

def next(self, key):
    return self.next_node(self.root, key)

def prev_node(self, root, key):
    if root is None:
        return "none"
    if key > root.val:
        right_prev = self.prev_node(root.right,
key)

        if right_prev != "none":
            return right_prev
        else:
            return root.val
    else:
        return self.prev_node(root.left, key)

def prev(self, key):
    return self.prev_node(self.root, key)

```

```

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

def delete_node(self, root, key):
    if root is None:
        return root
    if key < root.val:
        root.left = self.delete_node(root.left,
key)
    elif key > root.val:
        root.right = self.delete_node(root.right,
key)
    else:
        if root.left is None:
            return root.right
        elif root.right is None:
            return root.left
        else:
            min_val = self.find_min(root.right)
            root.val = min_val
            root.right =
self.delete_node(root.right, min_val)
        if root is not None:
            root.size = 1 + self.get_size(root.left) +
self.get_size(root.right)
        return root

def delete(self, key):
    self.root = self.delete_node(self.root, key)

```

```

with open('input.txt', 'r', encoding='utf-8') as
input_file, open('output.txt', 'w',
encoding='utf-8') as output_file:
    bst = BST()

    for line in input_file:
        command, x = line.split()
        if command == "insert":
            bst.insert(int(x))
        elif command == "delete":
            bst.delete(int(x))
        elif command == "exists":
            result = bst.exists(int(x))
            output_file.write("true\n" if result else
"false\n")
        elif command == "next":
            result = bst.next(int(x))
            output_file.write(f"{result}\n")
        elif command == "prev":
            result = bst.prev(int(x))
            output_file.write(f"{result}\n")

```

Текстовое объяснение решения.

Аналогично предыдущей задаче просто добавим в коде пару новых методов и изменим обработку ввода и вывода.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

input.txt		output.txt
1 insert 2	✓	1 true
2 insert 5		2 false
3 insert 3		3 5
4 exists 2		4 3
5 exists 4		5 none
6 next 4		6 3
7 prev 4		7
8 delete 5		
9 next 4		
10 prev 4		

Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)

input.txt		:	output.txt	
1	exists 1	✓	1	false
2			2	

⚠ The file size (5,13 MB) e...		132457	365471517
		132458	false
133941	delete 952470 ✓	132459	530876316
133942	delete 868134996	132460	124137385
133943	exists 72811042	132461	487402413
133944	insert 743695852	132462	false
133945	prev 766866226	132463	502335400
133946	prev 88179204	132464	467749751
133947	exists 439236454	132465	692114218
133948	delete 766594910	132466	715343588
133949	delete 417860281	132467	580435476
133950	delete 763232979	132468	511621850
133951	next 194067903	132469	false
133952	exists 407900364	132470	false
133953	next 728470567	132471	156379145
133954	next 650218549	132472	false
133955	next 668110765	132473	false
133956	prev 999724325	132474	false
133957	next 132038434	132475	136292354

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00054020003	15.2421875
Пример из задачи	0.000523499911651	15.2421875
Верхняя граница диапазона значений входных данных из текста задачи	0.978570600040257	17.4296875

Вывод по задаче:

Задача несложная, но трудоемкая так объемная, решать ее однозначно стоит, если уже решали две предыдущие, чтобы использовать предыдущий код, тогда она уже не кажется такой большой.

Задача №6. Оpoznание двоичного дерева поиска [1,5 баллов]

Текст задачи.

6 Задача. Оpoznание 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 ребенок.

- **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 $0, 1, \dots, n - 1$ 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.** $0 \leq n \leq 10^5$, $-2^{31} \leq K_i \leq 2^{31} - 1$, $-1 \leq L_i, R_i \leq n - 1$. 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.

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.

Листинг кода. (именно листинг, а не скрины)

```
def is_BST(tree_nodes, i_node, low, high):
    # явл листом
    if i_node == -1:
        return True
    if tree_nodes[i_node][0] < low or tree_nodes[i_node][0] > high:
        return False
    return (is_BST(tree_nodes, tree_nodes[i_node][1], low, tree_nodes[i_node][0]) and
            is_BST(tree_nodes, tree_nodes[i_node][2], tree_nodes[i_node][0], high))
```

```

with open('input.txt', 'r', encoding='utf-8') as
input_file, open('output.txt', 'w', encoding='utf-8')
as output_file:
    n = int(input_file.readline())
    tree_nodes = [list(map(int,
input_file.readline().split())) for _ in range(n)]

    output_file.write('CORRECT' if len(tree_nodes) == 0
or is_BST(tree_nodes, 0, -2 ** 32, 2 ** 32) else
'INCORRECT')

```

Текстовое объяснение решения.

Выводит список узлов, каждый из которых содержит значение узла и индексы его левого и правого ребенка. Границы задаем как low и high float('-inf'), float('inf') соответственно (или по условию их можно задать как -2**32, 2**32).

Если индекс узла = -1, то это лист, дерево пустое, а значит корректное, возвращаем True. Если значения узла вне границ диапазона вернем false. В остальных случаях функция будет работать рекурсивно для левого и правого поддеревя, например в левом поддереве high обновится до значения узла и станет границей которую нельзя превышать, для правого аналогично.

Результат работы кода на примерах из текста задачи: (скрины input output файлов)

input.txt			output.txt	
1	3	✓	1	CORRECT
2	2 1 2			
3	1 -1 -1			
4	3 -1 -1			

input.txt			output.txt	
1	3	✓	1	INCORRECT
2	1 1 2			
3	2 -1 -1			
4	3 -1 -1			

input.txt	output.txt
1 0	1 CORRECT

input.txt	output.txt
1 5	1 CORRECT
2 1 -1 1	
3 2 -1 2	
4 3 -1 3	
5 4 -1 4	
6 5 -1 -1	

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

input.txt	output.txt
1 4	1 INCORRECT
2 4 1 -1	
3 2 2 3	
4 1 -1 -1	
5 5 -1 -1	

Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)

```

input.txt x
1 100000
2 -1858093534 22069 25230
3 519761280 74022 33039
4 -2118765577 57885 71529
5 1088577536 84940 49468
6 -1781485665 41586 20633
7 1014385489 82258 27095
8 1458227566 7416 94249
9 -1029100875 73141 81523
10 6860284 29320 56138
11 1563334895 78425 95116
12 -790217532 53071 1742
13 786625295 36652 97394
14 503891903 29521 82178
15 1615222997 21114 25301
16 61621589 57101 69177
17 1195307596 40243 80397
18 -974109833 44382 89878
19 587839693 1686 43989
20 -2007183405 32895 68847
21 249721918 32911 6493
22 1136137322 41845 37362

```

```

input.txt x
1 0
output.txt x
1 CORRECT

```

	Время выполнения	Затраты памяти
Пример из задачи	0.00049149990081	14.82421875
Пример из задачи	0.000578300096094	14.90625
Пример из задачи (Нижняя граница)	0.000535600003	14.65625
Пример из задачи	0.000562099972739	14.9140625
Пример из задачи	0.0005182999884	14.890625
Пример из задачи	0.00079760001972	14.6953125
Верхняя граница диапазона значений входных данных из текста задачи	0.0846138999331	18.3671875

Вывод по задаче:

В этой задаче я реализовала алгоритм для проверки является ли список узлов правильным бинарным деревом. Для всех тестов (а их тут необычно много) программа работает корректно. Меня даже удивило, насколько быстро алгоритм работает на максимальных значениях .

Задача №7. Оpozнание двоичного дерева поиска (усложненная версия) [2.5 баллов]

Текст задачи.

7 Задача. Оpozнание двоичного дерева поиска (усложненная версия) [10 s, 512 Мб, 2.5 балла]

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 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.** $0 \leq n \leq 10^5$, $-2^{31} \leq K_i \leq 2^{31} - 1$, $-1 \leq L_i, R_i \leq n - 1$. 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. 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).

Листинг кода. (именно листинг, а не скрины)

```
def is_BST(tree_nodes, i_node, low, high):
    if i_node == -1:
        return True
    if tree_nodes[i_node][0] < low:
        return False
    if tree_nodes[i_node][0] >= high:
        return False
```

```

        return (is_BST(tree_nodes, tree_nodes[i_node][1],
low, tree_nodes[i_node][0]) and
                is_BST(tree_nodes, tree_nodes[i_node][2],
tree_nodes[i_node][0], high))

with open('input.txt', 'r', encoding='utf-8') as
input_file, open('output.txt', 'w', encoding='utf-8')
as output_file:
    n = int(input_file.readline())
    tree_nodes = [list(map(int,
input_file.readline().split())) for _ in range(n)]

    output_file.write('CORRECT' if len(tree_nodes) == 0
or is_BST(tree_nodes, 0, -2 ** 32, 2 ** 32) else
'INCORRECT')

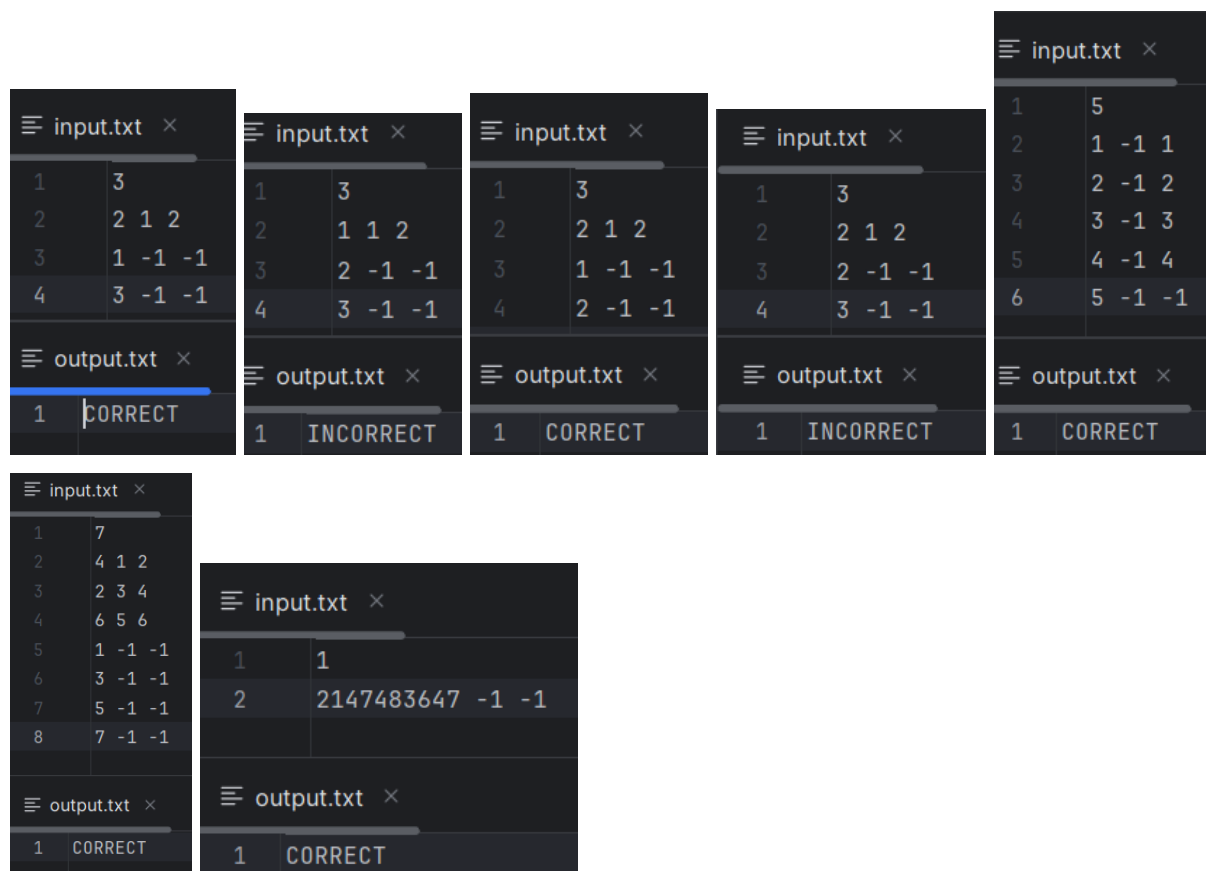
```

Текстовое объяснение решения.

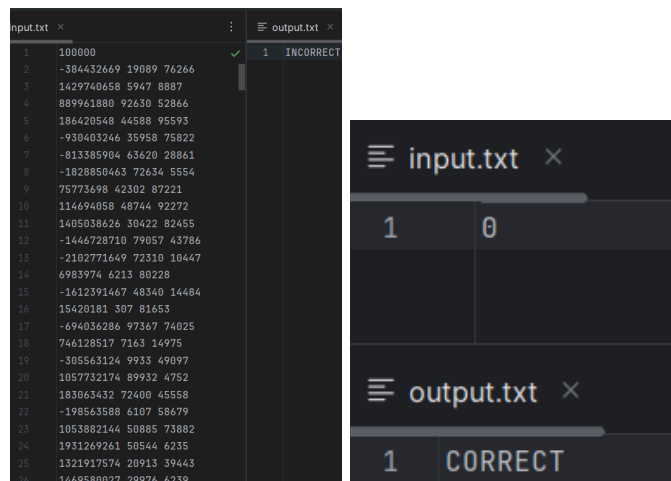
Абсолютно аналогично предыдущей задаче, отличие только в \geq .

В правом поддереве low будет обновляться до ключа узла, ключи могут как быть меньше этого значения так и равняться ему.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.000478800036944448	14.8828125

ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ		
Пример из задачи	0.0004674999509	14.921875
Пример из задачи	0.0004916000179	14.6953125
Пример из задачи	0.00042429997	14.91796875
Пример из задачи	0.00045749999117106	14.91796875
Пример из задачи	0.0004733999958	14.69921875
Пример из задачи	0.00051230005919	14.890625
Пример из задачи	0.000414000009	14.83984375
Верхняя граница диапазона значений входных данных из текста задачи	0.0827648999402299	19.015625

Вывод по задаче:

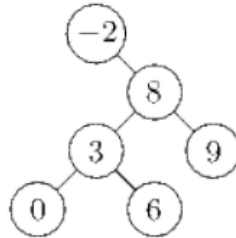
Задача аналогичная предыдущей.

Задача №8. Высота дерева возвращается [2 баллов]

Текст задачи.

Высотой дерева называется максимальное число вершин дерева в цепочке, начинающейся в корне дерева, заканчивающейся в одном из его листьев, и не содержащей никакой 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 $(1 \leq i \leq N)$ 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).
- **Ограничения на входные данные.** $0 \leq N \leq 2 \cdot 10^5, |K_i| \leq 10^9$. 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.

Листинг кода. (именно листинг, а не скрины)

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

class BST:
    def __init__(self):
        self.root = None

    def _add_node(self, node, key):
```

```

        if key < node.key:
            if node.left:
                self._add_node(node.left, key)
            else:
                node.left = Node(key)
        else:
            if node.right:
                self._add_node(node.right, key)
            else:
                node.right = Node(key)

    def add_node(self, key):
        if not self.root:
            self.root = Node(key)
        else:
            self._add_node(self.root, key)

    def height(self, node):
        if not node:
            return 0
        return 1 + max(self.height(node.left),
self.height(node.right))

with open('input.txt', 'r', encoding='utf-8') as
input_file, open('output.txt', 'w',
encoding='utf-8') as output_file:
    n = int(input_file.readline().strip())

    nodes = [list(map(int,
input_file.readline().strip().split())) for _ in
range(n)]
    tree = BST()
    for key, left, right in nodes:

```

```
tree.add_node(key)

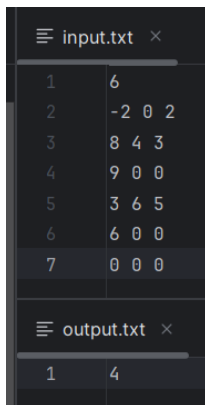
height = tree.height(tree.root)
output_file.write(str(height))
```

Текстовое объяснение решения.

Про класс BST: `__init__` инициализирует пустое дерево с корнем `root` `None`. Метод `add_node` обертка для `_add_node` который рекурсивно добавляет узел в дерево: если ключ меньше текущего узла идем в левое поддерево, иначе — в правое. Если дерево пустое, новый узел - корень. Метод `height` рекурсивно вычисляет высоту дерева. Если узел пустой возвр. 0. Иначе вычисляется высота левого и правого поддерева и возвращается максимальная плюс 1.

Считываем кол-во узлов. Считываем массив списков узлов, где каждый из списков содержит ключ узла и индексы левого и правого ребенка. В цикле добавляем все узлы в дерево по ключам. Вычисляем и выводим высоту дерева передавая корень.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



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

output.txt	
1	4

Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)

input.txt ×

The file size (4,86 MB) exce...

1 200000 ✓

2 -705950705 0 0

3 -314324130 171695 1

4 -426721012 99143 23

5 -431619234 70080 19

6 506394590 54362 192

7 910562666 91200 928

8 772108984 47181 832

9 -809221782 140976 1

10 688052662 183528 55

11 904773096 192130 36

12 615332622 153742 91

13 -102203823 177268 2

14 -908723047 37778 11

15 38342516 25893 6747

16 -676602379 6167 811

17 -270509769 67071 53

18 -468619470 133831 1

19 -543026106 192449 5

20 -445129070 89759 11

21 -680103660 169594 3

22 -463073948 28149 13

23 410846721 136509 16

24 -253746074 37790 17

25 -21669090 61336 256

output.txt ×

1 44

2

input.txt ×

1 0 ✓

output.txt ×

1 0

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004266001051291	14.890625
Пример из задачи	0.00047910003922879	14.84765625
Верхняя граница диапазона значений входных данных из текста задачи	0.81478470005095	18.921875

Вывод по задаче:

Для этой задачи мною был написан класс бинарного дерева, благодаря которому можно добавлять в дерево новые узлы по их ключам и вычислять высоту двоичного дерева.

Задача №9. Удаление поддеревьев [2 баллов]

Текст задачи.

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 ($1 \leq i \leq N$) 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.

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.** $1 \leq N \leq 2 \cdot 10^5$, $|K_i| \leq 10^9$, $1 \leq M \leq 2 \cdot 10^5$
- **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.
- **Пример:**

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

Листинг кода. (именно листинг, а не скрины)

```

def delete_subtree_node(bst, key):
    try:
        left_k, right_k = bst[key][0], bst[key][1]
        bst.pop(key)
        if left_k is not None:
            delete_subtree_node(bst, left_k)
        if right_k is not None:
            delete_subtree_node(bst, right_k)
    except:
        pass
    return len(bst)

with open('input.txt', 'r', encoding='utf-8') as input_file, open('output.txt', 'w',
encoding='utf-8') as output_file:
    n = int(input_file.readline())
    nodes = [[int(i) for i in
input_file.readline().split()] for _ in range(n)]
    bst = {}
    for key in nodes:
        left_k, right_k = key[1], key[2]
        if left_k == 0:
            left_k = None
        else:
            left_k = nodes[left_k - 1][0]
        if right_k == 0:
            right_k = None
        else:
            right_k = nodes[right_k - 1][0]
        bst[key[0]] = [left_k, right_k]

    m = int(input_file.readline())

```

```

        removes = [int(i) for i in
input_file.readline().split()]

    for key in removes:
        output_file.write(f'{delete_subtree_node(bst,
key) }\n')

```

Текстовое объяснение решения:

В отличии от предыдущей задачи будем использовать не классы, а словарь, чтобы уложиться во время 2 секунд.

На основе nodes ,читанного массив списков, создаем словарь (вида dict[int, list[int]]), где ключ - это ключ узла, значение - массив с ключом левого и ключом правого ребенка(None если их нет).(bst[key[0]] = [left_k, right_k] по ключу текущего узла добавляем ключи его детей)

В функции удаления поддерева. Используем обработчик try:code except: pass, чтобы избежать ошибок при удалении несуществующего узла. Распаковываем ключи детей. Удаляем текущий узел. Смотрим, есть ли у него дети, если да, то рекурсивно удаляем и их. Возвращаем длину словаря(оставшееся кол-во узлов)

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

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

Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)

input.txt	output.txt
1 20000	1 19999
2 1 0 0	2 19997
3 3 0 0	3 19995
4 2 1 2	4 19987
5 5 0 0	5 19986
6 8 0 0	6 19984
7 7 0 5	7 19983
8 6 4 6	8 19979
9 4 3 7	9 19975
10 10 0 0	10 19974
11 13 0 0	11 19973
12 12 0 10	12 19972
13 11 9 11	13 19968
14 15 0 0	14 19966
15 18 0 0	15 19964

input.txt	output.txt
1 1	1 0
2 0 0 0	2
3 1	
4 0	

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000460000010	14.703125
Пример из задачи	0.00046929996460	14.82421875
Верхняя граница диапазона значений входных данных из текста задачи	0.03239120007492602	19.0859375

Вывод по задаче:

Изначально я решила эту задачу аналогично предыдущей, классом BTS и методами `add_node` `remove_subtree`, однако на максимальных значения входных данных на 20000 узлов(+удаление 20000 поддеревьев) алгоритм работал слишком долго(Время работы: 13.170430399943143 секунд Затраты памяти: 17.453125 Мб, из которых 11.3 секунды уходило лишь только на добавление узлов в дерево). Поэтому было решено вместо классов

использовать словари т.к. в них операции добавления обхода и удаления гораздо быстрее.

Вывод

Во время выполнения лабораторной работы я разобралась с основными понятиями, такими как, корень, узел, предок и ребенок, высота, листья. А также научилась решать задачи про двоичные деревья поиска BST и сбалансированные AVL деревья. Отработала на практике обход бинарных деревьев в глубину, добавление и удаление узлов, поиск элемента в диапазоне.

Лабораторная показалась мне сложной, но при этом довольно интересной, я узнала много нового. Мне удалось написать эффективные алгоритмы, которые удовлетворяют ограничениям по памяти и времени.