

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 №4

Выполнил:  
Волжева М.И.  
К3141

Проверила:  
Артамонова В.Е.

Санкт-Петербург  
2023 г.

## Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №4. Простейший неявный ключ [2 s, 256 Mb, 1 балл]	3
Задача №8. Высота дерева возвращается [2 s, 256 Mb, 2 балла]	5
Задача №15. Удаление из АВЛ-дерева [2 s, 256 Mb, 3 балла]	9
Задачи по выбору	15
Задача №1. Обход двоичного дерева [5 s, 512 Mb, 1 балл]	15
Задача № 2. Гирлянда [2 s, 256 Mb, 1 балл]	18
Задача №5. Простое двоичное дерево поиска [2 s, 512 Mb, 1 балл]	20
Задача №16. К-й максимум [2 s, 512 Mb, 3 балла]	24
Задача №10. Проверка корректности [2 s, 256 Mb, 2 балла]	31
Вывод	33

## Задачи по варианту

### Задача №4. Простейший неявный ключ [2 s, 256 Mb, 1 балл]

- Текст задачи:

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 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.

- 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 выходного файла (output.txt).

Для каждого запроса вида «? k» выведите в отдельной строке ответ.

- Ограничение по времени. 2сек.

- Ограничение по памяти. 256 мб.

- Пример:

Input.txt	+ 1 + 4 + 3 + 3 ? 1 ? 2 ? 3 + 2 ? 3
Output.txt	1 3 4 3

#### Листинг кода:

```
import time
import os, psutil

class Node:
```

```

def __init__(self, data):
    self.data = data
    self.left = self.right = None

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

    def __find(self, node, parent, value):
        if node is None:
            return None, parent, False

        if value == node.data:
            return node, parent, True

        if value < node.data:
            if node.left:
                return self.__find(node.left, node, value)

        if value > node.data:
            if node.right:
                return self.__find(node.right, node, value)

        return node, parent, False

    def append(self, obj):
        if self.root is None:
            self.root = obj
            return obj

        s, p, fl_find = self.__find(self.root, None, obj.data)

        if not fl_find and s:
            if obj.data < s.data:
                s.left = obj
            else:
                s.right = obj

        return obj

    def find_k_element(self, node, result, k, flag, file):
        if len(result) == k:
            flag = True
            file.write(str(result[k - 1]) + "\n")
        if node is None:
            return
        if not flag:
            self.find_k_element(node.left, result, k, flag, file)
            result.append(node.data)
            self.find_k_element(node.right, result, k, flag, file)

t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("input.txt")
m = open("output.txt", "w")
t = Tree()
for each in f.readlines():
    operation, number = each.split()
    if operation == "+":

```

```

        t.append(Node(int(number)))
    if operation == "?":
        t.find_k_element(t.root, [], int(number), False, m)

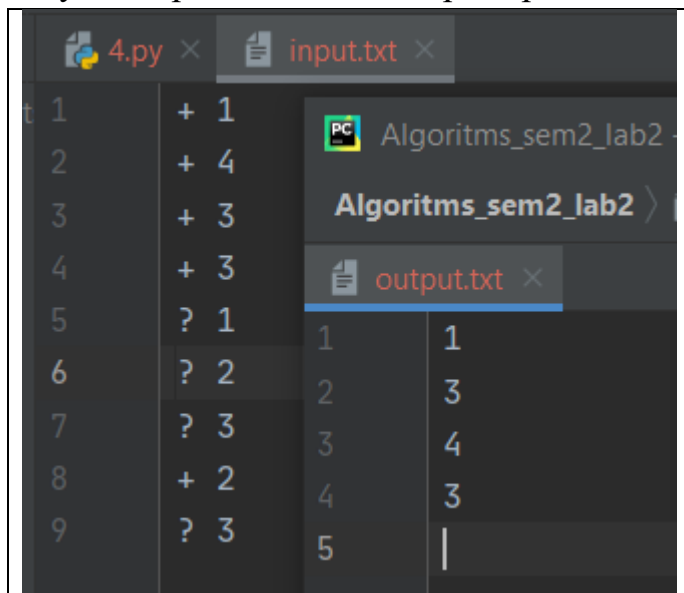
f.close()
m.close()
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

### Текстовое объяснение решения:

Мы реализовали бинарное дерево, как класс. Поиск k-ого по возрастанию элемента осуществляется поиском в глубину, который в определенный момент останавливается.

### Результат работы кода на примерах из текста задачи:



### Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Пример	0.0008377999999999997	14.1953125

Вывод по задаче: Мы научились реализовывать бинарное дерево, как класс и можем искать k-по возрастанию элемент.

### **Задача №8. Высота дерева возвращается [2 s, 256 Mb, 2 балла]**

- 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. 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 выходного файла (output.txt). Выведите одно целое число – высоту дерева.
- Пример:

Input.txt	6 -2 0 2 8 4 3 9 0 0 3 6 5 6 0 0 0 0 0
Output.txt	4

#### Листинг кода:

```
import time
import os, psutil

class Node:

    def __init__(self):
        self.key_t = None
```

```

        self.key = None
        self.left = None
        self.right = None
        self.parent = None
        self.height = 0

class BinTree:

    def __init__(self):
        self.root = None
        self.nodes = {}

    def set_height(self, node):
        node.height = 1
        while node.parent:
            parent = node.parent
            if parent.height <= node.height:
                parent.height = node.height + 1
            node = parent

t_start = time.perf_counter()
process = psutil.Process(os.getpid())

with open('input.txt') as f:
    n = int(f.readline())
    if n == 0:
        with open('output.txt', 'w') as f:
            f.write('0')
            exit()
    tree = BinTree()
    data = []
    leaves = []
    nodes = {}
    for i in range(1, n+1):
        data.append(list(map(int, f.readline().split())))
        tree.nodes[i] = Node()
        tree.nodes[i].key = data[i-1][0]
        if data[i-1][1] == 0 and data[i-1][2] == 0:
            leaves.append(i)

    for i in range(1, n+1):
        if data[i-1][1] != 0:
            tree.nodes[i].left = tree.nodes[data[i-1][1]]
            tree.nodes[data[i-1][1]].parent = tree.nodes[i]
        if data[i-1][2] != 0:
            tree.nodes[i].right = tree.nodes[data[i-1][2]]
            tree.nodes[data[i-1][2]].parent = tree.nodes[i]
        if i == 1:
            tree.root = tree.nodes[i]

    for i in leaves:
        tree.set_height(tree.nodes[i])
    with open('output.txt', 'w') as f:
        f.write(str(tree.root.height))

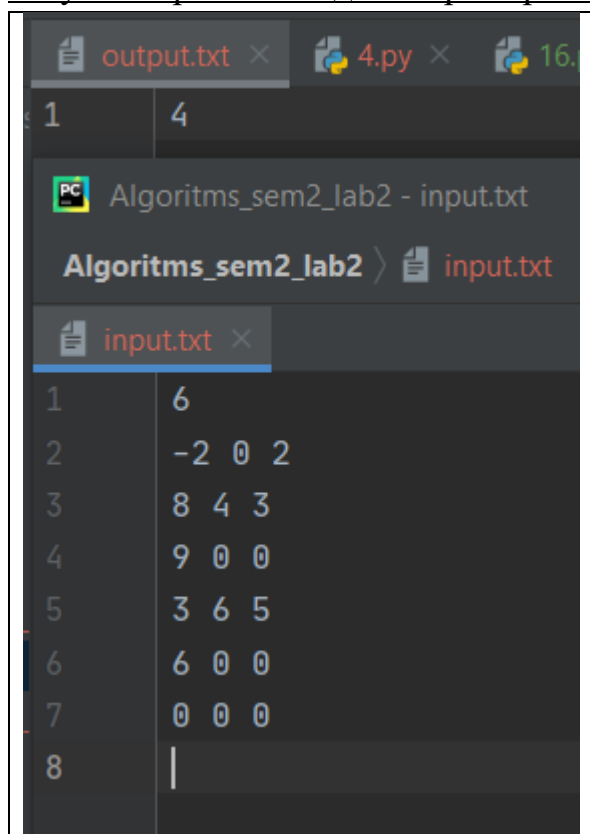
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

### Текстовое объяснение решения:

Мы создали два класса – лист и дерево. Сначала мы читаем данные из файла, а далее для каждого листа применяем функцию `set_height`. Ответом будет является значение этой функции от корня. Немного про функцию (. Высота начального корня = 1 Пока дальше есть родитель, он становится корнем. Если высота родителя меньше или равна высоты корня, то мы увеличиваем высоту корня. Иначе корень становится родителем.)

### Результат работы кода на примерах из текста задачи:



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

The screenshot shows a code editor with three tabs: `output.txt`, `4.py`, and `16.`. The active tab is `input.txt`, which contains the following data:

Line	Value
1	4
2	
3	
4	
5	
6	
7	
8	

### Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Пример	0.0012832999999999872	14. 23828125

Вывод по задаче: Мы создали 2 класса и нашли высоту дерева.



### Задача №15. Удаление из AVL-дерева [2 s, 256 Mb, 3 балла]

- Текст задачи:

Удаление из AVL-дерева вершины с ключом  $X$ , при условии ее наличия, осуществляется следующим образом:

- путем спуска от корня и проверки ключей находится  $V$  – удаляемая вершина;
- если вершина  $V$  – лист (то есть, у нее нет детей):
  - удаляем вершину;
  - поднимаемся к корню, начиная с бывшего родителя вершины  $V$ , при этом если встречается несбалансированная вершина, то производим поворот.
- если у вершины  $V$  не существует левого ребенка:
  - следовательно, баланс вершины равен единице и ее правый ребенок – лист;
  - заменяем вершину  $V$  ее правым ребенком;
  - поднимаемся к корню, производя, где необходимо, балансировку.
- иначе:
  - находим  $R$  – самую правую вершину в левом поддереве;
  - переносим ключ вершины  $R$  в вершину  $V$ ;
  - удаляем вершину  $R$  (у нее нет правого ребенка, поэтому она либо лист, либо имеет левого ребенка, являющегося листом);
  - поднимаемся к корню, начиная с бывшего родителя вершины  $R$ , производя балансировку.

Исключением является случай, когда производится удаление из дерева, состоящего из одной вершины - корня. Результатом удаления в этом случае будет пустое дерево. Указанный алгоритм не является единственно возможным, но мы просим Вас реализовать именно его, так как тестирующая система проверяет точное равенство получающихся деревьев.

- Формат входного файла (input.txt).

Входной файл содержит описание двоичного дерева, а также ключа вершины, которую требуется удалить из 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. В последней строке содержится число  $X$  – ключ вершины, которую требуется удалить из дерева. Гарантируется, что такая вершина в дереве существует.

- Ограничения на входные данные.

$1 \leq N \leq 2 \cdot 10^5$ ,  $|K_i| \leq 10^9$ ,  $|X| \leq 10^9$

- Формат выходного файла (output.txt).

Выведите в том же формате дерево после осуществления операции удаления. Нумерация вершин может быть произвольной при условии соблюдения формата.

- Пример:

Input.txt	3 4 2 3 3 0 0 5 0 0 4
Output.txt	2 3 0 2 5 0 0

#### Листинг кода:

```
import time
import os, psutil
import sys
from collections import deque

res = []
class Node:
    def __init__(self, data):
        self.data = data
        self.par = None
        self.left = None
        self.right = None
        self.height = -1
        self.id = 0
        self.next = None

def dfs(root):
    if root.left is not None:
        dfs(root.left)
    if root.right is not None:
        dfs(root.right)
```

```

fix_height(root)

def height_right(root):
    if root.right is None:
        return 0
    return root.right.height

def height_left(root):
    if root.left is None:
        return 0
    return root.left.height

def fix_height(root):
    root.height = max(height_left(root), height_right(root)) + 1

def blc(root):
    r = 0
    l = 0
    if root.right is not None:
        r = root.right.height
    if root.left is not None:
        l = root.left.height
    return r - l

def Rotate(node, side):
    if side == 'left':
        if node is None or node.right is None:
            return node
        parent = node.par
        right = node.right
        right_left = right.left
        if parent:
            if parent.right == node:
                parent.right = right
            else:
                parent.left = right
        right.par = parent
        right.left = node
        node.par = right
        node.right = right_left
        if right_left:
            right_left.par = node
        fix_height(node)
        fix_height(right)
        return right
    else:
        if node is None or node.left is None:
            return node
        parent = node.par
        left = node.left
        left_right = left.right
        if parent:
            if parent.left == node:
                parent.left = left
            else:
                parent.right = left

```

```

        left.par = parent
        left.right = node
        node.par = left
        node.left = left_right
        if left_right:
            left_right.par = node
        fix_height(node)
        fix_height(left)
        return left

def getMax(root):
    if root is None:
        return root
    while root.right is not None:
        root = root.right
    return root

def Balance(root):
    fix_height(root)
    balance = blc(root)
    if balance > 1:
        if blc(root.right) < 0:
            root.right = Rotate(root.right, 'right')
        return Rotate(root, 'left')
    elif balance < -1:
        if blc(root.left) > 0:
            root.left = Rotate(root.left, 'left')
        return Rotate(root, 'right')
    return root

def delete(root, key):
    if root is None:
        return root
    elif key < root.data:
        root.left = delete(root.left, key)
    elif key > root.data:
        root.right = delete(root.right, key)
    else:
        if root.left is None and root.right is None:
            return None
        if root.left is None:
            root = root.right
            return Balance(root)
        temp = getMax(root.left)
        root.data = temp.data
        root.left = delete(root.left, temp.data)
    return Balance(root)

def printBST(root, n):
    global res
    queue = deque()
    queue.append((root, (-1, -1)))
    while queue:
        u, v = queue.popleft()
        if v[0] >= 0 and v[1] >= 0:
            res[v[0]][v[1]] = len(res) + 1
        if u is None:

```

```

        continue
    tmp = [0, 0, 0]
    tmp[0] = u.data
    res.append(tmp)
    cur = len(res)
    if u.left is not None:
        queue.append((u.left, (cur - 1, 1)))
    if u.right is not None:
        queue.append((u.right, (cur - 1, 2)))

t_start = time.perf_counter()
process = psutil.Process(os.getpid())
sys.stdin = open("input.txt", "r")
sys.stdout = open("output.txt", "w")
n = int(sys.stdin.readline())
nodes = []
for i in range(n + 10):
    nodes.append(Node(0))
for i in range(n):
    k, l, r = map(int, sys.stdin.readline().split())
    nodes[i + 1].data = k
    if l:
        nodes[i + 1].left = nodes[l]
        nodes[l].par = nodes[i + 1]
    if r:
        nodes[i + 1].right = nodes[r]
        nodes[r].par = nodes[i + 1]

val = int(sys.stdin.readline())
dfs(nodes[1])
nodes[1] = delete(nodes[1], val)
printBST(nodes[1], n)
sys.stdout.write(str(len(res)) + "\n")
n = len(res)
for i, j, k in res:
    sys.stdout.write(str(i) + ' ' + str(j) + ' ' + str(k) + '\n')
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss / (1024 * 1024), "mb")

```

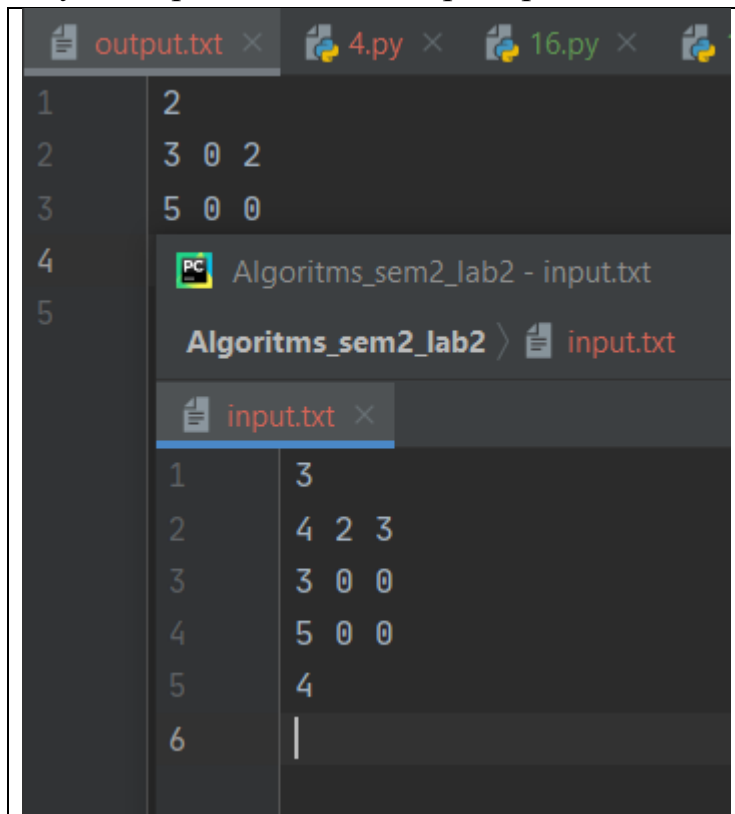
### Текстовое объяснение решения:

Основной принцип решения включает:

1. Определение класса Node, представляющего узел дерева, с полями для данных, ссылок на родителя и потомков, а также для высоты узла.
2. Реализацию функций для обхода и балансировки дерева, а также для удаления узла.
3. Считывание данных из файла ввода, создание и связывание узлов дерева на основе считанных значений.
4. Удаление узла с заданным значением из дерева.
5. Обход дерева в ширину для сохранения результата в переменную.
6. Запись результата в выходной файл.

7. Вывод информации о времени выполнения программы и доступной памяти.

Результат работы кода на примерах из текста задачи:



```
output.txt x 4.py x 16.py x
1 2
2 3 0 2
3 5 0 0
4
5
Algoritms_sem2_lab2 - input.txt
Algoritms_sem2_lab2 > input.txt
input.txt x
1 3
2 4 2 3
3 3 0 0
4 5 0 0
5 4
6 |
```

Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Пример	0.0008377999999999997	14.1953125

Вывод по задаче: В этой задаче я научилась удалять из AVL-дерева вершины с ключом X

## Задачи по выбору

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

- Текст задачи:

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

- Формат ввода.

Стандартный ввод или input.txt. В первой строке входного файла содержится количество узлов  $n$ . Узлы дерева пронумерованы от 0 до  $n - 1$ . Узел 0 является корнем. Следующие  $n$  строк содержат информацию об узлах 0, 1, ...,  $n - 1$  по порядку. Каждая из этих строк содержит три целых числа  $K_i$ ,  $L_i$  и  $R_i$ .  $K_i$  – ключ  $i$ -го узла,  $L_i$  – индекс левого ребенка  $i$ -го узла, а  $R_i$  – индекс правого ребенка  $i$ -го узла. Если у  $i$ -го узла нет левого или правого ребенка (или обоих), соответствующие числа  $L_i$  или  $R_i$  (или оба) будут равны  $-1$ .

- Ограничения на входные данные.

$1 \leq n \leq 105$ ,  $0 \leq K_i \leq 109$ ,  $-1 \leq L_i, R_i \leq n-1$ . 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 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).

- Пример:

Input.txt	5 4 1 2 2 3 4 5 -1 -1 1 -1 -1 3 -1 -1	
-----------	--	--

Output.txt	1 2 3 4 5 4 2 1 3 5 1 3 2 5 4	
------------	-------------------------------------	--

### Листинг кода:

```
import time
import os, psutil

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

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

    def __find(self, node, parent, value):
        if node is None:
            return None, parent, False

        if value == node.data:
            return node, parent, True

        if value < node.data:
            if node.left:
                return self.__find(node.left, node, value)

        if value > node.data:
            if node.right:
                return self.__find(node.right, node, value)

        return node, parent, False

    def append(self, obj):
        if self.root is None:
            self.root = obj
            return obj

        s, p, fl_find = self.__find(self.root, None, obj.data)

        if not fl_find and s:
            if obj.data < s.data:
                s.left = obj
            else:
                s.right = obj

        return obj

    def find_k_element(self, node, result, k, flag, file):
        if len(result) == k:
            flag = True
            file.write(str(result[k - 1]) + "\n")
        if node is None:
            return
        if not flag:
```



```

        self.find_k_element(node.left, result, k, flag, file)
        result.append(node.data)
        self.find_k_element(node.right, result, k, flag, file)

t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("input.txt")
m = open("output.txt", "w")
t = Tree()
for each in f.readlines():
    operation, number = each.split()
    if operation == "+":
        t.append(Node(int(number)))
    if operation == "?":
        t.find_k_element(t.root, [], int(number), False, m)

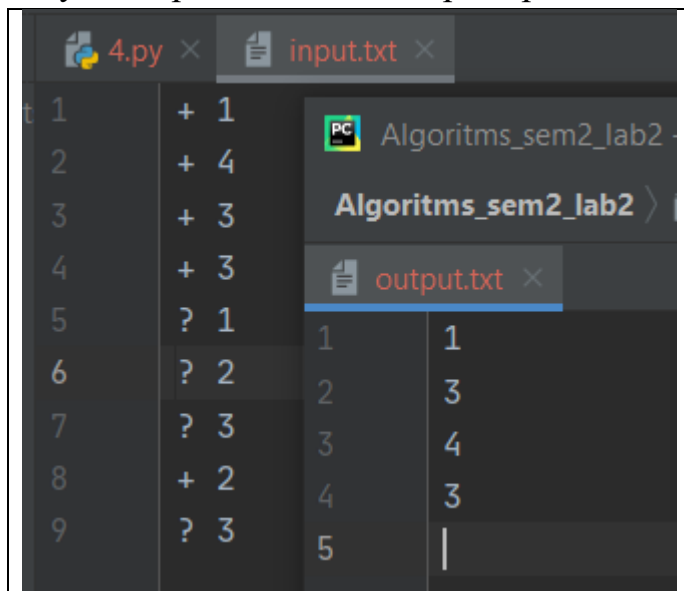
f.close()
m.close()
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

### Текстовое объяснение решения:

Мы реализовали бинарное дерево, как класс. Поиск k-ого по возрастанию элемента осуществляется поиском в глубину, который в определенный момент останавливается.

### Результат работы кода на примерах из текста задачи:



### Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Пример	0.0008377999999999997	14.1953125

Вывод по задаче: Мы научились реализовывать бинарное дерево, как класс и можем искать k-по возрастанию элемент.

## Задача № 2. Гирлянда [2 s, 256 Mb, 1 балл]

- 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 ( $h_i = (h_{i-1} + h_{i+1})/2 - 1$  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.

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 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  $10^{-6}$

- Пример:

Input.txt	8 15	692 532.81
Output.txt	9.75	446113.34434782615

### Листинг кода:

```
import time
import os, psutil

def high(h, n):
    left = 0
    right = h[0]
    while (right - left > 0.0000000001):
        h[1] = (left + right) / 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:
            right = h[1]
        else:
            left = h[1]
        return h[n - 1]

t_start = time.perf_counter()
process = psutil.Process(os.getpid())

with open('input.txt') as m:
    a = list(map(float, m.readline().split()))
    n = int(a[0])
    A = a[1]

h = []
for i in range(n):
    h.append(0)
h[0]=float(A)
res = high(h, n)

f = open('output.txt', 'w')
f.write(str(res))

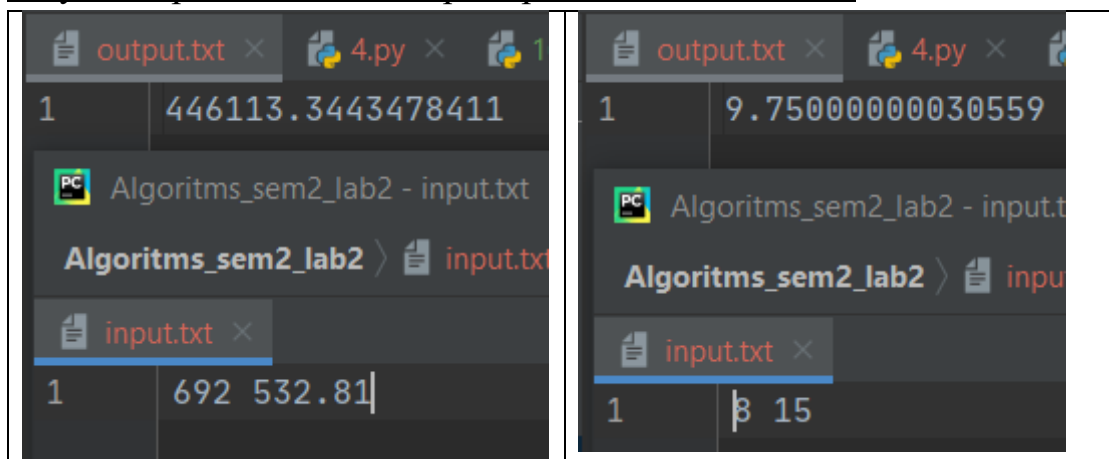
f.close()
m.close()
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

### Текстовое объяснение решения:

Одушевлять перебор по всем значениям будет не очень эффективно. Поэтому мы ищем среднее между левым и правым, и дальше отталкиваясь от этого решаем в которую сторону нужно двигаться и решаем дальше.

### Результат работы кода на примерах из текста задачи:



### Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Пример	0.017354399999999992	14.1953125
Пример	0.004493500000000011	14.08203125

Вывод по задаче: Мы научились оптимизировать перебор. Мы нашли минимальное значение второго конца гирлянды.

#### **Задача №5. Простое двоичное дерево поиска [2 s, 512 Mb, 1 балл]**

- Текст задачи:  
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 109.
- Ограничения на входные данные.  $0 \leq N \leq 100$ ,  $|x| \leq 109$ .
- Формат выходного файла (output.txt).  
Выведите последовательно результат выполнения всех операций exists, next, prev. Следуйте формату выходного файла из примера.

Пример:

Input.txt	insert 2 insert 5 insert 3 exists 2 exists 4 next 4
-----------	--

	prev 4 delete 5 next 4 prev 4
Output.txt	true false 5 3 none 3

### Листинг кода:

```
import time
import os, psutil

t_start = time.perf_counter()
process = psutil.Process(os.getpid())

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

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

    def insert(self, value):
        if self.root is None:
            self.root = Node(value)
        else:
            self._insert(self.root, value)

    def _insert(self, node, value):
        if value < node.value:
            if node.left is None:
                node.left = Node(value)
            else:
                self._insert(node.left, value)
        elif value > node.value:
            if node.right is None:
                node.right = Node(value)
            else:
                self._insert(node.right, value)

    def delete(self, value):
        self.root = self._delete(self.root, value)

    def _delete(self, node, value):
        if node is None:
            return None
        if value < node.value:
            node.left = self._delete(node.left, value)
```

```

        elif value > node.value:
            node.right = self._delete(node.right, value)
        else:
            if node.left is None:
                return node.right
            elif node.right is None:
                return node.left
            else:
                min_node = self._find_min(node.right)
                node.value = min_node.value
                node.right = self._delete(node.right, min_node.value)
    return node

def exists(self, value):
    return self._exists(self.root, value)

def _exists(self, node, value):
    if node is None:
        return False
    if node.value == value:
        return True
    elif value < node.value:
        return self._exists(node.left, value)
    else:
        return self._exists(node.right, value)

def next(self, value):
    node = self.root
    result = None
    while node is not None:
        if node.value > value:
            if result is None or node.value < result.value:
                result = node
            node = node.left
        else:
            node = node.right
    if result is None:
        return "none"
    else:
        return str(result.value)

def prev(self, value):
    node = self.root
    result = None
    while node is not None:
        if node.value < value:
            if result is None or node.value > result.value:
                result = node
            node = node.right
        else:
            node = node.left
    if result is None:
        return "none"
    else:
        return str(result.value)

with open("input.txt") as f_in, open("output.txt", "w") as f_out:
    tree = BinarySearchTree()
    for line in f_in:
        operation, value = line.strip().split()

```

```

value = int(value)
if operation == "insert":
    tree.insert(value)
elif operation == "delete":
    tree.delete(value)
elif operation == "exists":
    f_out.write(str(tree.exists(value)).lower() + "\n")
elif operation == "next":
    f_out.write(tree.next(value) + "\n")
elif operation == "prev":
    f_out.write(tree.prev(value) + "\n")

print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

### Текстовое объяснение решения:

Для реализации двоичного дерева поиска были определены два класса: Node и BinarySearchTree. Класс Node представляет узел дерева и содержит ссылки на левое и правое поддеревья, а также на значение, хранящееся в узле. Класс BinarySearchTree содержит ссылку на корень дерева и определенные методы для выполнения операций над деревом. Входные данные читаются из файла, результаты выводятся в выходной файл. Общая идея заключается в использовании базовых операций над двоичным деревом для выполнения заданных операций.

### Результат работы кода на примерах из текста задачи:

output.txt		input.txt	
1	true	1	insert 2
2	false	2	insert 5
3	5	3	insert 3
4	3	4	exists 2
5	none	5	exists 4
6	3	6	next 4
7		7	prev 4
		8	delete 5
		9	next 4
		10	prev 4
		11	

### Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
--	----------------------------	----------------------

Пример	0. 0008553999999999923	14. 23828125
--------	------------------------	--------------

Вывод по задаче: В этой задаче я реализовала простое двоичное дерево поиска

### Задача №16. К-й максимум [2 s, 512 Mb, 3 балла]

- Текст задачи:  
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.  $N \leq 10^5$ ,  $|K_i| \leq 10^9$ .
- Формат выходного файла (output.txt).  
Для каждой команды нулевого типа в выходной файл должна быть выведена строка, содержащая единственное число –  $k_i$ -й максимум.
- Пример:

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



	0 1 0 2 0 3
Output.txt	7 5 3 10 7 3

### Листинг кода:

```
import time
import psutil
import sys

t_start = time.perf_counter()

class Node:
    def __init__(self, v=int()): #значение (value
        self.value = v
        self.left = None #ссылки на левого и правого потомков (left и
right)
        self.right = None
        self.count_left = 0 #количество узлов в поддеревьях слева и справа
        self.count_right = 0
        self.height_left = 0 #высоту левого и правого поддеревьев
        self.height_right = 0
        self.parent = None #ссылка на родительский узел

class AVLTree:
    root = None

    def __init__(self): #дерево в данный момент не содержит ни одного узла
и является пустым
        self.root = None

    def insert(self, value): #вставляет новый узел со значением value в
дерево, сохраняя его самобалансировку.
        if not self.find(value):
            time = Node(value)
            if self.root is None:
                self.root = time
            else:
                index = self.root
                flag = True
                while flag:
                    if time.value > index.value and index.right is not
None:
                        index = index.right
                    elif time.value < index.value and index.left is not
None:
                        index = index.left
                    else:
                        flag = False
```

```

        if time.value > index.value:
            time.parent = index
            index.right = time
        else:
            time.parent = index
            index.left = time
        self.balance_number(index)
        while index is not None:
            self.balance_number(index)
            index = self.balance(index)
            self.root = index
            index = index.parent
        return time

    def delete(self, value): #удаляет узел с заданным значением value из
        #дерева, сохраняя его самобалансировку.
        self.delete_vertex(self.find(value))

    def delete_vertex(self, time): #удаление узла time из AVL-дерева.
        if time:
            if max(time.height_right, time.height_left) == 0:
                if time.parent is None:
                    self.root = None
                else:
                    index = time.parent
                    if time.parent.left == time:
                        time.parent.left = None
                    else:
                        time.parent.right = None
                    while index is not None:
                        self.balance_number(index)
                        index = self.balance(index)
                        self.root = index
                        index = index.parent
            else:
                if time.height_right > time.height_left:
                    time.value = time.right.value
                    self.delete_vertex(time.right)
                else:
                    time.value = time.left.value
                    self.delete_vertex(time.left)

    def find(self, value): #выполняет поиск узла с заданным значением value
        #в дереве и возвращает его. Если узел не найден, возвращается None.
        index = self.root
        flag = True
        while flag and index is not None:
            if value > index.value:
                index = index.right
            elif value < index.value:
                index = index.left
            else:
                flag = False
        if index is None:
            return None
        else:
            return index

    # log(n)
    def exists(self, value): #проверяет, существует ли узел с заданным
        #значением value в дереве. Возвращает строку 'true', если узел существует, и

```

```

'false' в противном случае.
    if self.find(value):
        return 'true'
    return 'false'

    def next(self, value): #находит минимальное значение, большее заданного
value в дереве, и возвращает его. Если такого значения нет, возвращается
строка 'none'.
        time = self.next_time(value, self.root)
        if time == 10 ** 10:
            return 'none'
        else:
            return time

    def next_time(self, value, link): #рекурсивно находит минимальное
значение, которое больше заданного значения value в дереве, начиная с узла
link.
        if link is not None:
            min_l = self.next_time(value, link.left)
            if min_l <= value:
                min_l = 10 ** 10
            min_r = self.next_time(value, link.right)
            if min_r <= value:
                min_r = 10 ** 10
            return min(min_l, min_r, link.value if link.value > value else
10 ** 10)
        else:
            return 10 ** 10

    def prev(self, value): #использует prev_time(value, link) для поиска
максимального значения, меньшего заданного значения value в дереве.
        time = self.prev_time(value, self.root) #Функция prev возвращает
найденное значение или строку 'none', если такого значения нет.
        if time == -10 ** 11:
            return 'none'
        else:
            return time

    def prev_time(self, value, link): #рекурсивно находит максимальное
значение, которое меньше заданного значения value в дереве, начиная с узла
link.
        if link is not None:
            min_l = self.prev_time(value, link.left)
            if min_l >= value:
                min_l = -10 ** 11
            min_r = self.prev_time(value, link.right)
            if min_r >= value:
                min_r = -10 ** 11
            return max(min_l, min_r, link.value if link.value < value else
-10 ** 11)
        else:
            return -10 ** 11

    def balance_number(self, vertex): #пересчитывает значения высоты и
количества узлов в поддеревьях для указанного узла vertex.
        if vertex:
            vertex.height_left = 0
            vertex.height_right = 0
            if vertex.left:
                vertex.height_left = 1 + max(vertex.left.height_right,
vertex.left.height_left)

```

```

        if vertex.right:
            vertex.height_right = 1 + max(vertex.right.height_right,
vertex.right.height_left)
            self.balance_sum(vertex)

    def balance_sum(self, vertex): # пересчитывает суммарное количество
узлов в поддеревьях для указанного узла vertex.
        if vertex:
            vertex.count_left = 0
            vertex.count_right = 0
            if vertex.left:
                vertex.count_left = 1 + vertex.left.height_right +
vertex.left.height_left
            if vertex.right:
                vertex.count_right = 1 + vertex.right.height_right +
vertex.right.height_left

    def balance(self, vertex): #выполняет балансировку дерева для
указанного узла vertex в случае несоответствия баланса.
        if vertex:
            dif = vertex.height_right - vertex.height_left
            if dif > 1:
                if vertex.right.height_right < vertex.right.height_left:
                    vertex = self.blr(vertex)
                else:
                    vertex = self.slr(vertex)
            elif dif < -1:
                if vertex.left.height_right > vertex.left.height_left:
                    vertex = self.brr(vertex)
                else:
                    vertex = self.srr(vertex)
        return vertex

    def slr(self, vertex): #операцию малого левого поворота (single left
rotation) для балансировки AVL-дерева.
        if vertex is not None and vertex.height_right - vertex.height_left
> 0 and vertex.right.height_left <= vertex.right.height_right:
            time_par = vertex.parent
            time_left = vertex.right.left
            vertex.right.left = vertex
            vertex.parent = vertex.right
            vertex.right = time_left
            vertex.parent.parent = time_par
            if time_par is not None:
                if time_par.left == vertex:
                    time_par.left = vertex.parent
                else:
                    time_par.right = vertex.parent
            self.balance_number(vertex)
            vertex = vertex.parent
            self.balance_number(vertex)
        return vertex

    def srr(self, vertex):
        if vertex is not None and vertex.height_left - vertex.height_right
> 0 and vertex.left.height_right <= vertex.left.height_left:
            time_par = vertex.parent
            time_right = vertex.left.right
            vertex.left.right = vertex
            vertex.parent = vertex.left
            vertex.left = time_right

```

```

        vertex.parent.parent = time_par
        if time_par is not None:
            if time_par.left == vertex:
                time_par.left = vertex.parent
            else:
                time_par.right = vertex.parent
        self.balance_number(vertex)
        vertex = vertex.parent
        self.balance_number(vertex)
    return vertex

    def blt(self, vertex):
        if vertex is not None and vertex.height_right - vertex.height_left
>= 2 and vertex.right.height_left > vertex.right.height_right:
            vertex.right = self.srt(vertex.right)
            vertex = self.slt(vertex)
        return vertex

    def brt(self, vertex):
        if vertex is not None and vertex.height_left - vertex.height_right
>= 2 and vertex.left.height_right <= vertex.left.height_left:
            vertex.left = self.slt(vertex.left)
            vertex = self.srt(vertex)
        return vertex

    def find_max(self, n, vertex=None): #находит n-ое по величине значение
в дереве и возвращает его.
        if not vertex:
            vertex = self.root
        if vertex.count_right + 1 == n:
            return vertex.value
        elif vertex.count_right >= n:
            return self.find_max(n, vertex.right)
        else:
            return self.find_max(n - vertex.count_right - 1, vertex.left)

sys.stdin = open("input.txt", "r")
sys.stdout = open("output.txt", "w")

tree = AVLTree()
n = int(sys.stdin.readline())
for i in range(n):
    s = sys.stdin.readline().split()
    if s[0] == '+1' or s[0] == '1':
        tree.insert(int(s[1]))
    elif s[0] == '-1':
        tree.delete(int(s[1]))
    elif s[0] == '0':
        print(tree.find_max(int(s[1])))

print("Время работы: %s секунд " % (time.perf_counter() - t_start))
print(str(psutil.virtual_memory().available * 100 /
psutil.virtual_memory().total) + " MB")

```

Текстовое объяснение решения:

В данной задаче воспользуемся обходом узлов в отсортированном порядке. Функции insert для вставки ктото узла, delete для удаления ктото узла. В функции удаления есть

важные моменты:

- 1) если удаляемый узел – лист, то просто удаляем его,
- 2) У удаляемого узла есть только один дочерний элемент – скопируем дочерний элементв узел и удалим его,
- 3) У узла, который нужно удалить, есть два потомка - найдем неупорядоченный преемник узла, скопируем содержимое неупорядоченного преемника в узел и удалим неупорядоченного преемника.

Для каждого узла мы знаем, сколько элементов слева и справа. На основании этого смотрим условия.

Результат работы кода на примерах из текста задачи:

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

Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Пример	0.0008553999999999923	14.23828125

Вывод по задаче: В этой задаче я написала программу, реализующую структуру данных, позволяющую добавлять и удалять элементы, а также находить k-й максимум.

### Задача №10. Проверка корректности [2 s, 256 Mb, 2 балла]

- Текст задачи:

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  $(1 \leq i \leq N)$  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.  $0 \leq N \leq 2 \cdot 10^5$ ,  $|K_i| \leq 109$ .
- Формат выходного файла (output.txt).

Выведите «YES», если данное во входном файле дерево является двоичным деревом поиска, и «NO», если не является.

#### Листинг кода:

```
import time
import os, psutil

t_start = time.perf_counter()
process = psutil.Process(os.getpid())

class BNode:
    def __init__(self, value=0, left=0, right=0):
        self.value = value
        self.left = left
        self.right = right

def tree_input():
```

```

with open("input.txt", "r") as f:
    n = int(f.readline())
    if n == 0:
        return True
    arr = []
    for _ in range(n):
        inp = list(map(int, f.readline().split()))
        arr.append(BNode(inp[0], inp[1] - 1, inp[2] - 1))
    result = binary_tree_check(arr, 0, -float('inf'), float('inf'))
    if result:
        return True
    return False

def binary_tree_check(inp, i, left, right):
    if i == -1:
        return True
    if inp[i].value <= left or inp[i].value >= right:
        return False
    check = binary_tree_check(inp, inp[i].left, left, inp[i].value) and
    binary_tree_check(inp, inp[i].right, inp[i].value, right)
    return check

if __name__ == "__main__":
    res = tree_input()
    with open("output.txt", "w") as f:
        if res:
            f.write('YES')
        else:
            f.write('NO')

print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

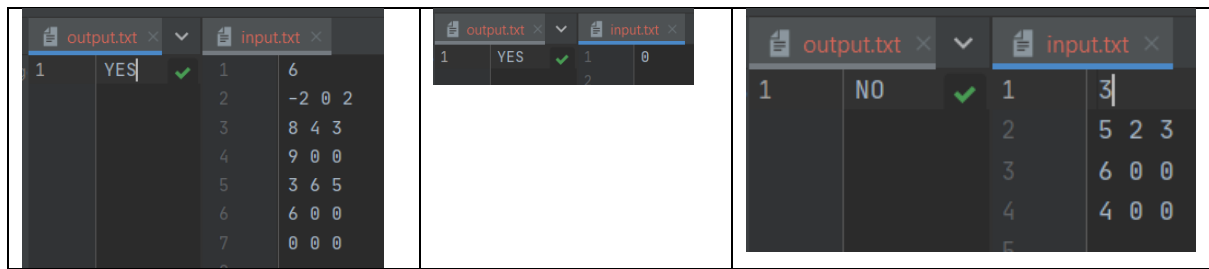
```

### Текстовое объяснение решения:

- 1) Определение класса BNode для узлов дерева, имеющего значение, левого и правого потомков.
- 2) Определение функции tree\_input для чтения данных из файла, создания дерева и проверки на соответствие условиям двоичного дерева.
- 3) Определение функции binary\_tree\_check для проверки дерева на соответствие условиям двоичного дерева.
- 4) Запись результата проверки в файл output.txt.
- 5) Вывод времени выполнения и использованной памяти в консоль.

### Результат работы кода на примерах из текста задачи:





### Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Пример	0. 0008553999999999923	14. 23828125
Пример	0. 0008553999999999923	14. 23828125
Пример	0. 0008553999999999923	14. 23828125

Вывод по задаче: В этой задаче я научилось проверять является ли дерево двоичным деревом поиска.

### **Вывод**

Я вспомнила как работать с классами, узнала, что такое бинарное дерева и узнала как выполнять проверки и производить какие-то действия с данной структурой данных.