

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 27

Выполнил:  
Филиппов А.Э.  
К3139

Проверила:  
Артамонова В.Е.

Санкт-Петербург  
2023 г.

## Содержание отчета

<b>Содержание отчета</b>	<b>2</b>
<b>Задачи по варианту</b>	<b>3</b>
Задача №2. Гирлянда [1 балл]	3\
Задача №8. Высота дерева возвращается [2 балла]	3
Задача №11. Сбалансированное двоичное дерево поиска [2 балла]	3
<b>Дополнительные задачи</b>	<b>4</b>
Задача №1. Обход двоичного дерева [1 балл]	4
Задача №3. Простейшее BST [1 балл]	4
Задача №4. Простейший неявный ключ [1 балл]	4
Задача №5. Простое двоичное дерево поиска [1 балл]	4
Задача №6. Опознание двоичного дерева поиска [1.5 балла]	4
Задача №7. Опознание двоичного дерева поиска (усложненная версия) [2.5 балла]	4
Задача №9. Удаление поддеревьев [2 балла]	4
Задача №10. Проверка корректности [2 балла]	4
<b>Вывод</b>	<b>5</b>

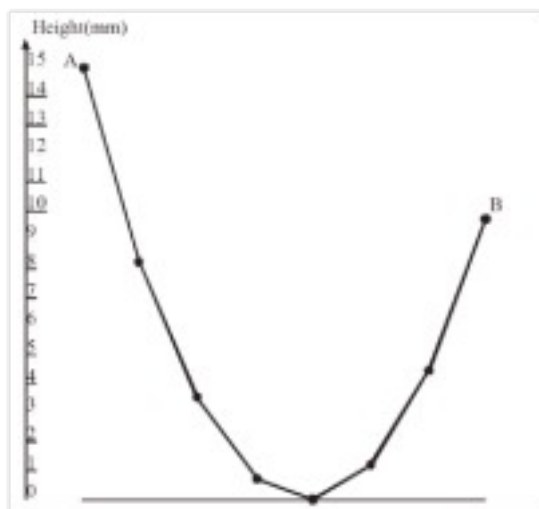
## Задачи по варианту

### Задача №2. Гирлянда [1 балл]

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  $1 < i < N$ ).

$$2 - 1 \text{ для } 1 < i < N).$$

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	446113.3443478 2615

- Проверить можно по [ссылке](#), OpenEdu, курс "Алгоритмы программирования и структуры данных 6 неделя, 2 задача.

Код:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

n, a = map(float, lines[0].split())
n = int(n)

girlande = [float(-1)] * n
girlande[0] = a

l = a
r = 0

while l - r > 0.0000000001:
    girlande[1] = (l+r) / 2
    check = True

    for i in range(2, n):
        girlande[i] = 2 * girlande[i-1] - girlande[i-2] + 2
        if girlande[i] < 0:
            check = False
```

```

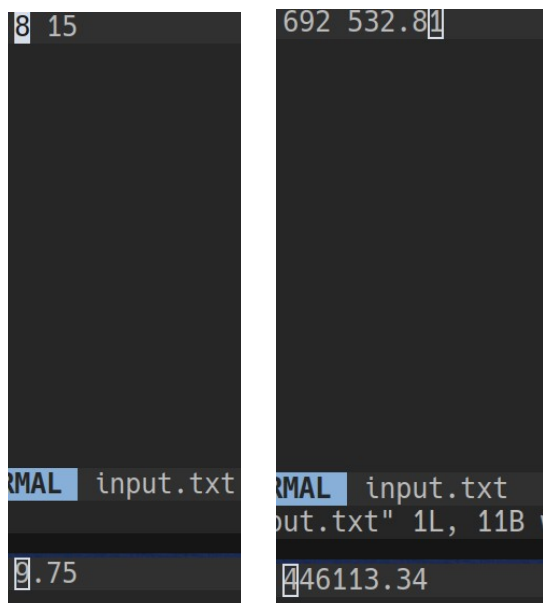
        break
    if check:
        l = girlande[1]
    else:
        r = girlande[1]

out.write("{:.2f}".format(girlande[n-1]))
print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

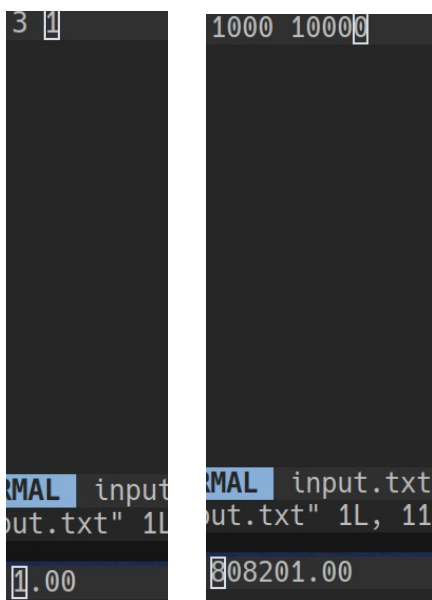
```

В цикле перебираются значения для l и соответственно для r. Изначальные значения для l — a, для r - 0

Результат работы кода на примерах из текста задачи:



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



	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.000279433999821776 5	13890
Пример из задачи	0.000467296999886457 34	13891
Пример из задачи	0.000279433999821776 5	13890
Верхняя граница диапазона значений входных данных из текста задачи	0.024636320999888994	44266

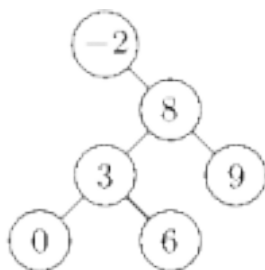
Вывод по задаче:

Не совсем понятно как эта задача относится к теме двоичных деревьев, ну и ладно. Вывод — динамика круто

### Задача №8. Высота дерева возвращается [2 балла]

Высотой дерева называется максимальное число вершин дерева в цепочке, начинающейся в корне дерева, заканчивающейся в одном из его листьев, и не содержащей никакую вершину дважды.

Так, высота дерева, состоящего из единственной вершины, равна единице. Высота пустого дерева равна нулю. Высота дерева, изображенного на рисунке, равна четырем.



Дано **двоичное дерево поиска**. В вершинах этого дерева записаны ключи — целые числа, по модулю не превышающие 109. 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.
- Пример:

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

- Во входном файле задано то же дерево, что и изображено на рисунке.

- Проверить можно по [ссылке](#), OpenEdu, курс "Алгоритмы программирования и структуры данных 6 неделя, 3 задача.

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

class Node:
    def __init__(self, val):
        self.l = 0
        self.r = 0
        self.val = val

def insert(i):
    k, l, r = inputs[i-1]
    el = Node(k)
    if l != 0:
        el.l = insert(l)
    else:
        el.l = 0

    if r != 0:
        el.r = insert(r)
    else:
        el.r = 0
    return el

def depth(node):
    erste = -1
    zweite = -1
    if node.l:
        erste = depth(node.l)
    if node.r:
        zweite = depth(node.r)
    if erste == zweite == -1:
        return 1
    return max(zweite, erste) + 1

n = int(lines[0])
inputs = []

for i in range(n):
    arr = list(map(int, lines[i+1].split()))
    inputs.append(arr)
```



```

root = Node(inputs[0][0])
if inputs[0][1] == 0:
    root.l = 0
else:
    root.l = insert(inputs[0][1])
if inputs[0][2] == 0:
    root.r = 0
else:
    root.r = insert(inputs[0][2])
out.write(str(depth(root)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Реализовал функционал двоичного дерева поиска через функции, узлы через класс. Немного кривая инициализация корня, но зато работает.

Результат работы кода на примерах из текста задачи:

```

6
-2 0 2
8 4 3
9 0 0
3 6 5
6 0 0
0 0 0

```

RMAL input  
projects/la

Результат работы кода на максимальных и минимальных значениях:

```

3
1 0 0
0 1 0
0 0 1

```

RMAL input.txt  
out.txt" 4L, 2

```

2000000
1 585314913 136370 169753
2 728245965 114027 53629
3 990107748 19563 140687
4 329371043 146769 151296
5 -367522634 155069 150794
6 111126422 28554 147520
7 473606217 37335 113257
8 -269761096 184583 189680
9 -763102908 169156 142446
10 -580643974 1393 50818
11 -389445604 157741 5934
12 418622365 70315 181461
13 -259981199 153996 6639

```

NORMAL input.txt

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00020195399883959908	15282
Пример из задачи	0.0005041999993409263	17822
Верхняя граница диапазона значений входных данных из текста задачи	0.9486706379993848	52024116

Вывод по задаче: Двоичное дерево довольно просто реализовать

### Задача №11. Сбалансированное двоичное дерево поиска [2 балла]

Реализуйте сбалансированное двоичное дерево поиска.

- **Формат ввода / входного файла (input.txt).** Входной файл содержит описание операций с деревом, их количество  $N$  не превышает 105. 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.**  $0 \leq N \leq 105$ ,  $|x_i| \leq 109$ .

- **Формат вывода / выходного файла (output.txt).** Выведите последовательно результат выполнения всех операций exists, next, prev. Следуйте формату выходного файла из примера.
- **Ограничение по времени. 2 сек.**
- **Ограничение по памяти. 512 мб.**
- **Пример:**

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

Код:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

class Node:
    def __init__(self, val):
        self.l = None
        self.r = None
        self.val = val
        self.height = 0

class AVLTree:
    def __init__(self):
        self.root = None
```

```

def insert(self, key, node=None):
    threshold = 1
    if not node:
        if not self.root:
            node = Node(key)
            self.root = node
            return self.root
        return Node(key)

    if (key < node.val):
        node.l = self.insert(key, node.l)
    else:
        node.r = self.insert(key, node.r)

    node.height = max(self.getHeight(node.l), self.getHeight(node.r)) +
1    balance = self.getBalance(node)

    if balance > threshold:
        if self.getBalance(node.l) >= 0:
            node = self.rotateRight(node)
        else:
            node = self.rotateLeftRight(node)
    elif balance < -threshold:
        if self.getBalance(node.r) <= 0:
            node = self.rotateLeft(node)
        else:
            node = self.rotateRightLeft(node)

    self.root = node

    return node

def exists(self, key, node=None):
    if not node:
        node = self.root
    if key == node.val:
        return "true"
    elif key < node.val:
        if node.l:
            return self.exists(key, node.l)
    elif key > node.val:
        if node.r:
            return self.exists(key, node.r)
    return "false"

def next(self, key, node=None):
    if node == None:
        node = self.root
    if node.val <= key:
        if node.r:
            return self.next(key, node.r)
    else:

```

```

        return "none"
    else:
        if node.l:
            var = self.next(key, node.l)
            if var == "none" and node.val > key:
                return node.val
            return var
        else:
            if node.val > key:
                return node.val
            return "none"

def prev(self, key, node=None):
    if node == None:
        node = self.root
    if node.val >= key:
        if node.l:
            return self.prev(key, node.l)
        else:
            return "none"
    else:
        if node.r:
            var = self.prev(key, node.r)
            if var == "none" and node.val < key:
                return node.val
            return var
        else:
            if node.val < key:
                return node.val
            return "none"

def getMinValueNode(self, node):
    if node is None or node.l is None:
        return node

    return self.getMinValueNode(node.l)

def delete(self, key, node=None):
    if not node:
        return node

    elif key < node.val:
        node.l = self.delete(key, node.l)

    elif key > node.val:
        node.r = self.delete(key, node.r)

    else:
        # cases in which we delete a node with one child
        if node.l is None:
            temp = node.r
            node = None
            self.root = temp

```

```

        return temp

    elif node.r is None:
        temp = node.l
        node = None
        self.root = temp
        return temp
    # cases in which a node with 2 children is encountered
    temp = self.getMinValueNode(node.r)
    node.val = temp.val
    node.r = self.delete(temp.val, node.r)

    if node is None:
        self.root = node
        return node

    node.height = max(self.getHeight(node.l), self.getHeight(node.r)) +

1
    balance = self.getBalance(node)

    if balance > 1 and self.getBalance(node.l) >= 0:
        self.root = self.rotateRight(node)
        return self.root

    if balance < -1 and self.getBalance(node.r) <= 0:
        self.root = self.rotateLeft(node)
        return self.root

    if balance > 1 and self.getBalance(node.l) < 0:
        self.root = self.rotateLeftRight(node)
        return self.root

    if balance < -1 and self.getBalance(node.r) > 0:
        self.root = self.rotateRightLeft(node)
        return self.root

    self.root = node

    return node

def rotateRight(self, x):
    y = x.l

    x.l = y.r
    y.r = x

    x.height = 1 + max(self.getHeight(x.l), self.getHeight(x.r))
    y.height = 1 + max(self.getHeight(y.l), self.getHeight(y.r))

    return y

def rotateLeft(self, x):

```

```

        y = x.r

        x.r = y.l
        y.l = x

        x.height = 1 + max(self.getHeight(x.l), self.getHeight(x.r))
        y.height = 1 + max(self.getHeight(y.l), self.getHeight(y.r))

        return y

    def rotateLeftRight(self, node):
        node.l = self.rotateLeft(node.l)

        return self.rotateRight(node)

    def rotateRightLeft(self, node):
        node.r = self.rotateRight(node.r)

        return self.rotateLeft(node)

    def getBalance(self, node):
        if not node:
            return 0

        return self.getHeight(node.l) - self.getHeight(node.r)

    def getHeight(self, node):
        if not node:
            return -1

        return node.height

tree = AVLTree()
for i in range(len(lines)):
    act, x = lines[i].split()
    print(act, x)
    x = int(x)
    if act == "insert":
        tree.insert(x, tree.root)
    elif act == "exists":
        out.write(tree.exists(x) + "\n")
    elif act == "next":
        out.write(str(tree.next(x)) + "\n")
    elif act == "prev":
        out.write(str(tree.prev(x)) + "\n")
    elif act == "delete":
        tree.delete(x, tree.root)

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")

```

```
tracemalloc.stop()
```

Стандартные функции AVL-дерева довольно похожи на функции обычного дерева. Единственная разница — дерево необходимо балансировать после операции.

Результат работы кода на примерах из текста задачи:

```
insert 2
insert 5
insert 3
exists 2
exists 4
next 4
prev 4
delete 5
next 4
prev 4

NORMAL input.txt
projects/labs/s

true
false
5
3
none
3
```

Результат работы кода на максимальных и минимальных значениях:

```
insert 2

NORMAL input.txt
out.txt" 1L, 9B
```

```
6 next 892601463
5 prev -467120561
4 prev 578182443
3 next -100412934
2 prev -2970830
1 next 49086057
6 delete 214066189
1 prev -161107045
2 delete -889833995
3 prev -64036000
4 exists 760082970
5 insert -920931539
6 prev -287181194
7 prev 118586367

NORMAL input.txt

none
1 none
2 none
3 none
4 false
5 none
6 none
7 372158830
8 372158830
9 none
10 372158830
11 none
12 none
13 false

NORMAL output.txt
```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из	0.000260891998550505 4	17594



текста задачи		
Пример из задачи	0.008742222000364563 3	23161
Верхняя граница диапазона значений входных данных из текста задачи	1.6891260850006802	11042212

Вывод по задаче: Основные функции АВЛ-дерева похожи на функции в обычном бинарном дереве. Единственным существенным отличием балансировки в каждой операции кроме функций просмотра.

## Дополнительные задачи

### Задача №1. Обход двоичного дерева [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.

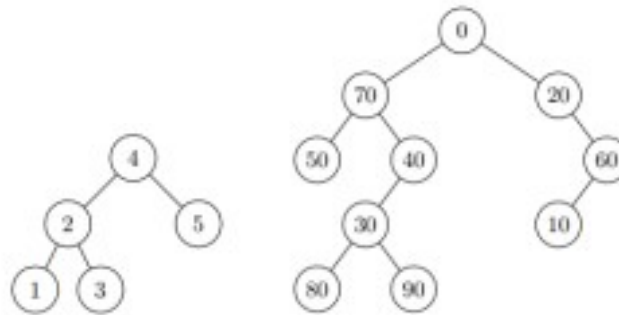
Следующие  $n$  строк содержат информацию об узлах 0, 1, ...,  $n - 1$  по порядку. Каждая из этих строк содержит три целых числа  $Ki$ ,  $Li$  и  $Ri$ .  $Ki$  – ключ  $i$ -го узла,  $Li$  - индекс левого ребенка  $i$ -го узла, а  $Ri$  - индекс правого ребенка  $i$ -го узла. Если у  $i$ -го узла нет левого или правого ребенка (или обоих), соответствующие числа  $Li$  или  $Ri$  (или оба) будут равны  $-1$ .

- **Ограничения на входные данные.**  $1 \leq n \leq 105$ ,  $0 \leq Ki \leq 109$ ,  $-1 \leq Li, Ri \leq n-1$ . Гарантируется, что данное дерево является двоичным деревом. В частности, если  $Li \neq -1$  и  $Ri \neq -1$ , то  $Li \neq Ri$ . Кроме того, узел не может быть ребенком двух разных узлов. Кроме того, каждый узел является потомком корневого узла.
- **Формат вывода / выходного файла (output.txt).** Выведите три строки. Первая строка должна содержать ключи узлов при центрированном обходе дерева (in-order). Вторая строка должна содержать ключи узлов при прямом обходе дерева (pre-order). Третья строка должна содержать ключи узлов при обратном обходе дерева (post-order).
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 512 мб.
- Примеры:

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

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

- Иллюстрации к обоим примерам:



- Что делать. Реализовать алгоритмы обхода из лекции. Обратите внимание, что в этой задаче дерево может быть очень глубоким, поэтому будьте осторожны, избегайте проблем с переполнением стека, если используете рекурсию, и обязательно протестируйте свое решение на дереве максимально возможной высоты.

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
```

```
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")
```

```
class Node:
    def __init__(self, val):
        self.l = None
        self.r = None
        self.val = val
```

```
def insert(i):
    k, l, r = inputs[i]
    el = Node(k)
    if l != -1:
        el.l = insert(l)
    else:
        el.l = -1

    if r != -1:
        el.r = insert(r)
    else:
        el.r = -1
    return el
```

```
def inOrder(root):
```

```

    if root == -1:
        return
    inOrder(root.l)
    out.write(str(root.val) + " ")
    inOrder(root.r)

def preOrder(root):
    if root == -1:
        return
    out.write(str(root.val) + " ")
    preOrder(root.l)
    preOrder(root.r)

def postOrder(root):
    if root == -1:
        return
    postOrder(root.l)
    postOrder(root.r)
    out.write(str(root.val) + " ")

n = int(lines[0])
inputs = []

for i in range(n):
    arr = list(map(int, lines[i+1].split()))
    inputs.append(arr)
root = Node(inputs[0][0])
root.l = insert(inputs[0][1])
root.r = insert(inputs[0][2])
inOrder(root)
out.write("\n")
preOrder(root)
out.write("\n")
postOrder(root)

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Реализован вставка в дерево, создан класс узла. Реализованы функции для вывода в разных порядках.

Результат работы кода на примерах из текста задачи:

```

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

RMAL input.txt

50 70 80 30 90 40 0 20 10 60
0 70 50 40 30 80 90 20 60 10
50 80 90 30 40 70 10 60 20 0

```

```

5
4 1 2
2 3 4
5 -1 -1
1 -1 -1
3 -1 -1
[]

RMAL input.txt
out.txt" 7L, 44B writt

[] 2 3 4 5
4 2 1 3 5
1 3 2 5 4

```

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

```

1
5 -1 -1

RMAL input.txt
put.txt" 2L, 10B wr

[] 5 5
5 5 5
5 5 5

```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0002220769997620664	17245
Пример из задачи	0.0003707129999384051	23001
Пример из задачи	0.00023763799981679767	19078
Верхняя граница	0.99	11042212

диапазона значений входных данных из текста задачи		
--	--	--

Вывод по задаче:

Для решения каждой задачи может пригодиться разные методы обхода дерева. Для какой-то inorder, а для какой-то postorder.

### Задача №3. Простейшее BST [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	

```

import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

class Node:
    def __init__(self, val):
        self.l = None
        self.r = None
        self.val = val

def insert(node, el):
    val = node.val
    if val == -1:
        node.val = el
        return
    if el >= val:
        if node.r:
            insert(node.r, el)
        else:
            node.r = Node(el)
    else:
        if node.l:
            insert(node.l, el)
        else:
            node.l = Node(el)

def get(node, x):
    if node.val <= x:
        if node.r:
            return get(node.r, x)
        else:
            return 0
    else:
        if node.l:
            var = get(node.l, x)
            if var == 0 and node.val > x:
                return node.val
            return var

```



```

        else:
            if node.val > x:
                return node.val
            return 0

line = lines[0]
root = Node(-1)
i = 0
while True:
    act, x = line.split()
    x = int(x)
    if act == "+":
        insert(root, x)
    else:
        out.write(str(get(root, x)) + "\n")

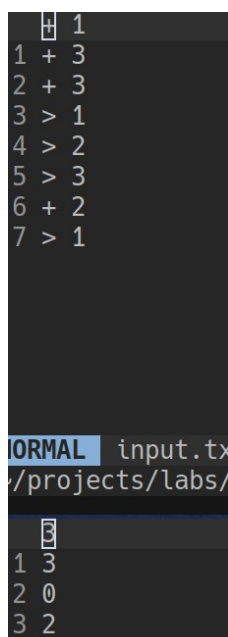
    if i == len(lines)-1:
        break
    i += 1
    line = lines[i]

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Было реализовано простейшее BST дерево. Узел представляет класс Node. Корень сохраняется как Node, вставка и получение реализованы через функции.

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



```

1
1 + 3
2 + 3
3 > 1
4 > 2
5 > 3
6 + 2
7 > 1

NORMAL input.txt
/projects/labs/

1 3
2 0
3 2

```

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

```
> 1

NORMAL input.txt
input.txt" 1L, 4B
```

```
12 > 552910436
11 > 585378167
10 > 320824256
9 + 873725978
8 + 5260527
7 + 817207589
6 + 152266869
5 > 934278081
4 > 228847099
3 + 422384831
2 > 444954512
1 + 563563910
00 + 589452212

MAL input.txt
ut.txt" 300000L, 356678

0
1 953105522
2 818085794
3 818085794
4 352705754
5 81301701
6 692170182
7 81301701
8 268229988
9 692170182
10 81301701
11 342811775
12 514343127
13 599632139
MAL output.txt
```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0001684890003161854	14694
Пример из задачи	0.00020295200010878034	16132
Верхняя граница диапазона значений входных данных из текста задачи	1.750515495000036	47888384

Вывод по задаче:

Чтобы реализовать простейшее BST не обязательно реализовывать данную структуру данных как класс. Достаточно использовать только узел.

#### Задача №4. Простейший неявный ключ [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	
+ 4	
+ 3	
+ 3	1
? 1	3
? 2	4
? 3	3
+ 2	
? 3	

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
```

```
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")
```

```
class Node:
    def __init__(self, val):
        self.l = None
```

```

        self.r = None
        self.val = val

def insert(node, el):
    val = node.val
    if el == val:
        return
    if val == -1:
        node.val = el
        return
    if el >= val:
        if node.r:
            insert(node.r, el)
        else:
            node.r = Node(el)
    else:
        if node.l:
            insert(node.l, el)
        else:
            node.l = Node(el)

def inOrder(node, x):
    global k
    if node is None:
        return
    else:
        if node.val == -1:
            return
        inOrder(node.l, x)
        k += 1
        if k == x:
            out.write(str(node.val) + "\n")
            return
        inOrder(node.r, x)

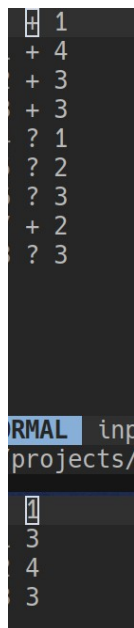
i = 0
line = lines[i]
root = Node(-1)
while True:
    act, x = line.split()
    x = int(x)
    if act == "+":
        insert(root, x)
    else:
        k = 0
        inOrder(root, x)
    if i == len(lines)-1:
        break
    i += 1
    line = lines[i]

```

```
print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()
```

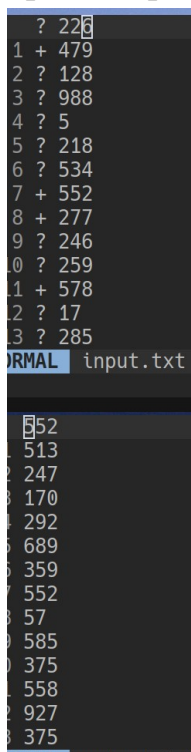
Вывести k-й по возрастанию элемент можно с помощью inorder вывода дерева. Используется модифицированная функция inorder из прошлого задания.

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

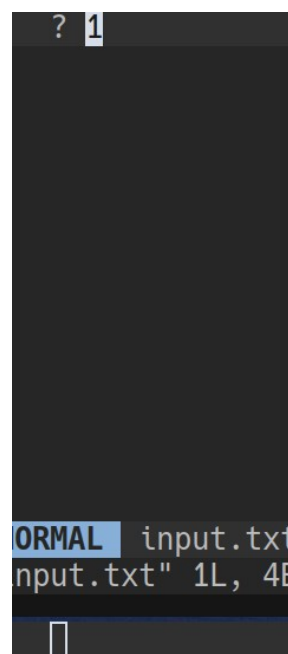


```
1
+ 4
+ 3
+ 3
? 1
? 2
? 3
+ 2
? 3
```

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



```
? 228
1 + 479
2 ? 128
3 ? 988
4 ? 5
5 ? 218
6 ? 534
7 + 552
8 + 277
9 ? 246
10 ? 259
11 + 578
12 ? 17
13 ? 285
```



```
? 1
```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.000150918000144884	14641
Пример из задачи	0.00023271200007002335	17097
Верхняя граница диапазона значений входных данных из текста задачи	0.7121889230002125	505481

Вывод по задаче:

Вывод в порядке `inorder` может быть полезным. Например, чтобы вывести  $k$ -й элемент по возрастанию.

#### **Задача №5. Простое двоичное дерево поиска [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 109. • **Ограничения на входные данные.**  $0 \leq N \leq 100$ ,  $|x_i| \leq 109$ .

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

- **Ограничение по времени. 2 сек.**

- **Ограничение по памяти. 512 мб.**

- **Пример:**

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

```
import time
import tracemalloc
```

```
class Node:
    def __init__(self, val):
        self.l = None
        self.r = None
        self.val = val
```

```
def insert(node, el):
    val = node.val
    if el == val:
        return
    if val == -1:
        node.val = el
        return
```

```

    if el >= val:
        if node.r:
            insert(node.r, el)
        else:
            node.r = Node(el)
    else:
        if node.l:
            insert(node.l, el)
        else:
            node.l = Node(el)

def next(node, x):
    if node.val <= x:
        if node.r:
            return next(node.r, x)
        else:
            return "none"
    else:
        if node.l:
            var = next(node.l, x)
            if var == "none" and node.val > x:
                return node.val
            return var
        else:
            if node.val > x:
                return node.val
            return "none"

def prev(node, x):
    if node.val >= x:
        if node.l:
            return prev(node.l, x)
        else:
            return "none"
    else:
        if node.r:
            var = prev(node.r, x)
            if var == "none" and node.val < x:
                return node.val
            return var
        else:
            if node.val < x:
                return node.val
            return "none"

def exists(node, x):
    val = node.val
    if val == x:
        return "true"
    if x >= val:

```



```

        if node.r:
            return exists(node.r, x)
        else:
            return "false"
    else:
        if node.l:
            return exists(node.l, x)
        else:
            return "false"

def getMinimumKey(curr):
    while curr.l:
        curr = curr.l
    return curr

def delete(root, x):
    parent = None

    curr = root

    while curr and curr.val != x:
        parent = curr

        if x < curr.val:
            curr = curr.l
        else:
            curr = curr.r

    if curr is None:
        return root

    if curr.l is None and curr.r is None:
        if curr != root:
            if parent.l == curr:
                parent.l = None
            else:
                parent.r = None
        else:
            root = None

    elif curr.l and curr.r:
        successor = getMinimumKey(curr.r)
        val = successor.val
        delete(root, successor.val)
        curr.val = val

    else:
        if curr.l:
            child = curr.l
        else:
            child = curr.r

```

```

        if curr != root:
            if curr == parent.l:
                parent.l = child
            else:
                parent.r = child
        else:
            root = child

    return root

tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

i = 0
line = lines[i]
root = Node(-1)
while True:
    act, x = line.split()
    x = int(x)
    if act == "insert":
        insert(root, x)
    elif act == "exists":
        out.write(str(exists(root, x)) + "\n")
    elif act == "delete":
        root = delete(root, x)
    elif act == "next":
        out.write(str(next(root, x)) + "\n")
    else:
        out.write(str(prev(root, x)) + "\n")
    if i == len(lines)-1:
        break
    i += 1
    line = lines[i]

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Был использован код из предыдущих задач. Была добавлена функция next, которая просто обходит дерево в поисках необходимого элемента. Удаление было реализовано через замену крайнего левого элемента правого ребенка выбранного элемента. Если необходимо заменить корень, то мы его меняем.

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

```

insert 2
insert 5
insert 3
exists 2
exists 4
next 4
prev 4
delete 5
next 4
prev 4

NORMAL input.txt
projects/labs/sem2

true
false
5
3
none
3

```

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

```

next 6553510060
exists -394484641
prev -538170383
insert -339711375
delete -633862395
insert 143131816
insert 477272696
prev -382235531
prev 703540396
delete -695069032
next 811912311
next -862676690
prev -96394006
exists -235889054
NORMAL input.txt

none
false
none
none
477272696
none
-339711375
-339711375
false
none
-339711375
false
false
false
NORMAL output.txt

```

```

exists 2

NORMAL input.txt
input.txt" 1L, 9B written

false

```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00016544000027352013	13727
Пример из задачи	0.0002162629998565535	14707
Верхняя граница диапазона значений входных данных из текста задачи	0.0006979030004004017	30430

Вывод по задаче:

Удаление можно реализовать через рекурсивную «замену» крайнего левого элемента правого ребенка выбранного элемента. А еще лучше все-таки создать отдельный класс для дерева.

#### **Задача №6. Опознавание двоичного дерева поиска [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 поддеревья меньше ключа вершины  $V$ ;
- все ключи вершин из правого поддеревья 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 ребенков.

- **Формат ввода / входного файла (input.txt).** В первой строке входного файла содержится количество узлов  $n$ . Узлы дерева пронумерованы от 0 до  $n - 1$ . Узел 0 является корнем.

Следующие  $n$  строк содержат информацию об узлах 0, 1, ...,  $n - 1$  по порядку. Каждая из этих строк содержит три целых числа  $Ki$ ,  $Li$  и  $Ri$ .  $Ki$  – ключ  $i$ -го узла,  $Li$  - индекс левого ребенка  $i$ -го узла, а  $Ri$  - индекс правого ребенка  $i$ -го узла. Если у  $i$ -го узла нет левого или правого ребенка (или обоих), соответствующие числа  $Li$  или  $Ri$  (или оба) будут равны  $-1$ .

- **Ограничения на входные данные.**  $0 \leq n \leq 105$ ,  $-231 \leq Ki \leq 231 - 1$ ,  $-1 \leq Li, Ri \leq n - 1$ . Гарантируется, что данное дерево является двоичным деревом. В частности, если  $Li \neq -1$  и  $Ri \neq -1$ , то  $Li \neq Ri$ . 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.

- Примеры:

input.txt	output.txt	input.txt	output.txt	input.txt	output.txt
3 2 1 2 1 -1 -1 3 -1 -1	CORRECT	3 1 1 2 2 -1 - 1 3 -1 -1	INCORRECT	0	CORRECT

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

- Примечание. Пустое дерево считается правильным двоичным деревом поиска. Дерево не обязательно должно быть сбалансировано.

```

import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

class Node:
    def __init__(self, val):
        self.l = -1
        self.r = -1
        self.val = val

def insert(i):
    k, l, r = inputs[i]
    el = Node(k)
    if l != -1:
        el.l = insert(l)
    else:
        el.l = -1

    if r != -1:
        el.r = insert(r)
    else:
        el.r = -1
    return el

```

```

def inOrder(root):
    if root == -1:
        return
    inOrder(root.l)
    baum.append(root.val)
    inOrder(root.r)

n = int(lines[0])
inputs = []

for i in range(n):
    arr = list(map(int, lines[i+1].split()))
    inputs.append(arr)
if n == 0:
    out.write("CORRECT")
else:
    root = Node(inputs[0][0])
    if inputs[0][1] == -1:
        root.l = -1
    else:
        root.l = insert(inputs[0][1])
    if inputs[0][2] == -1:
        root.r = -1
    else:
        root.r = insert(inputs[0][2])
    baum = []
    inOrder(root)
    fail = False
    for i in range(1, len(baum)):
        if baum[i-1] > baum[i]:
            fail = True
            break
    if not fail:
        out.write("CORRECT")
    else:
        out.write("INCORRECT")

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Для решения задачи используется простой прием. А именно: если вывести все элементы дерева в порядке inorder и они будут отсортированы в возрастающем порядке, то перед нами двоичное дерево поиска.

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

```
3
2 1 2
1 -1 -1
3 -1 -1

NORMAL input.txt
projects/labs/sem2/1
CORRECT
```

```
3
1 1 2
2 -1 -1
3 -1 -1

NORMAL input.txt
put.txt" 5L, 28B writ
INCORRECT
```

```
0

NORMAL input.txt
put.txt" 1L, 2B wr
CORRECT
```

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

```
0

NORMAL input.txt
put.txt" 1L, 2B wr
CORRECT
```

```
1000000
1 256450285 31002 26527
2 -948653376 45320 22008
3 511739781 16994 86363
4 -639945909 41732 54531
5 158060712 18255 34085
6 711680244 31076 90076
7 -349123541 16358 55931
8 -649927736 85172 56897
9 -17359264 19670 81880
10 143137727 28429 94759
11 -897703410 75958 75698
12 860704732 95300 59061
13 -747059394 27670 94572

NORMAL input.txt
put.txt" 100001L, 1927225B
INCORRECT
```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0001820839988795342	13888
Пример из задачи	0.008362788999875193	15912
Пример из задачи	0.00021827599994139746	15919



Пример из задачи	0.0001820839988795342	13888
Верхняя граница диапазона значений входных данных из текста задачи	0.46603517299990926	26042972

Вывод по задаче: Если вывести все элементы дерева в порядке inorder и они будут отсортированы в возрастающем порядке, то перед нами двоичное дерево поиска.

### Задача №7. Опознание двоичного дерева поиска (усложненная версия) [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.**  $0 \leq n \leq 105$ ,  $-231 \leq K_i \leq 231 - 1$ ,  $-1 \leq L_i$ ,  $R_i \leq n - 1$ . 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).
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 512 мб.
- Примеры:

input.txt	output.txt	input.txt	output.txt	input.txt	output.txt	input.txt	output.txt
3 2 1 2 1 -1 -1 3 -1 -1	CORRECT	3 1 1 2 2 -1 -1 3 -1 -1	INCORRECT	3 2 1 2 1 -1 -1 2 -1 -1	CORRECT	3 2 1 2 2 -1 -1 3 -1 -1	INCORRECT

input.txt	output.txt	input.txt	output.txt	input.txt	output.txt
5 1 -1 1 2 -1 2	CORRECT	7 4 1 2 2 3 4 6 5 6 1 -1 -	CORRECT	1 2147483647 - 1 -1	CORRECT

3 -1		1			
3		3 -1 -			
4 -1		1			
4		5 -1 -			
5 -1 -		1			
1		7 -1 -			
		1			

- Примечание. Пустое дерево считается правильным двоичным деревом поиска. Дерево не обязательно должно быть сбалансировано. Попробуйте адаптировать алгоритм из предыдущей задачи к случаю, когда допускаются повторяющиеся ключи, и остерегайтесь целочисленного переполнения!

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
```

```
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")
```

```
class Node:
    def __init__(self, val):
        self.l = -1
        self.r = -1
        self.val = val
```

```
def insert(i):
    k, l, r = inputs[i]
    el = Node(k)
    if l != -1:
        el.l = insert(l)
    else:
        el.l = -1

    if r != -1:
        el.r = insert(r)
    else:
        el.r = -1
    return el
```

```

def isBST(node, mini, maxi):
    if node.val < mini or node.val > maxi:
        return False
    if node.l != -1:
        erste = isBST(node.l, mini, node.val - 1)
    else:
        erste = True

    if node.r != -1:
        zweite = isBST(node.r, node.val, maxi)
    else:
        zweite = True

    return erste and zweite

n = int(lines[0])
inputs = []

for i in range(n):
    arr = list(map(int, lines[i+1].split()))
    inputs.append(arr)
if n == 0:
    out.write("CORRECT")
else:
    root = Node(inputs[0][0])
    if inputs[0][1] == -1:
        root.l = -1
    else:
        root.l = insert(inputs[0][1])
    if inputs[0][2] == -1:
        root.r = -1
    else:
        root.r = insert(inputs[0][2])
    if isBST(root, - (2**32+1), 2**32+1):
        out.write("CORRECT")
    else:
        out.write("INCORRECT")

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Рекурсивно проверяем на соответствие условиям двоичного дерева поиска. Эти условия прописаны в функции isBST

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

```

1
2147483647 -1 -1

```

ORMAL input.txt  
projects/labs/sem2/1\_alg/Д

CORRECT

```

4 3
8 2 1 2
2 1 -1 -1
1 3 -1 -1

```

ORMAL input.txt  
input.txt" 5L, 28B

CORRECT

```

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

```

ORMAL input.txt  
input.txt" 9L, 60B v

CORRECT

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

```

0

```

ORMAL input.tx  
put.txt" 1L, 2

CORRECT

```

1000000
1 256450285 31002 26527
2 -948653376 45320 22008
3 511739781 16994 86363
4 -639945909 41732 54531
5 158060712 18255 34085
6 711680244 31076 90076
7 -349123541 16358 55931
8 -649927736 85172 56897
9 -17359264 19670 81880
10 143137727 28429 94759
11 -897703410 75958 75698
12 860704732 95300 59061
13 -747059394 27670 94572

```

ORMAL input.txt  
projects/labs/sem2/1\_alg/Дополните

CORRECT

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0001702260015008505 4	13888
Пример из задачи	0.0003405480001674732	14923

Пример из задачи	0.0002085859996441286	16389
Пример из задачи	0.0002777339996100636	18257
Верхняя граница диапазона значений входных данных из текста задачи	0.4783357950000209	26038819

Вывод по задаче:

Можно проверить разными способами является ли двоичное дерево поиска корректным.

### Задача №9. Удаление поддеревьев [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 109. 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

$Li = 0$ , если левого ребенка нет) и номера правого  $Ri$  ребенка  $i$ -ой вершины ( $i < Ri \leq N$  или  $Ri = 0$ , если правого ребенка нет).

Все ключи различны. Гарантируется, что данное дерево является деревом поиска.

В следующей строке находится число  $M$  – число запросов на удаление. В следующей строке находятся  $M$  чисел, разделенных пробелами – ключи, вершины с которыми (вместе с их поддеревьями) необходимо удалить. Все эти числа не превосходят 109 по абсолютному значению. Вершина с таким ключом не обязана существовать в дереве – в этом случае дерево изменять не требуется. Гарантируется, что корень дерева никогда не будет удален.

- **Ограничения на входные данные.**  $1 \leq N \leq 2 \cdot 10^5$ ,  $|Ki| \leq 10^9$ ,  $1 \leq M \leq 2 \cdot 10^5$
- **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	
-2 0	
2	
8 4 3	5
9 0 0	4
3 6 5	4
6 0 0	1
0 0 0	
4	
6 9 7	
8	

- Проверить можно по [ссылке](#), OpenEdu, курс "Алгоритмы программирования и структуры данных 6 неделя, 4 задача.

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

class Node:
    def __init__(self, val):
        self.l = 0
        self.r = 0
        self.val = val

def insert(i):
    k, l, r = inputs[i-1]
    el = Node(k)
    if l != 0:
        el.l = insert(l)
    else:
        el.l = 0

    if r != 0:
        el.r = insert(r)
    else:
        el.r = 0
    return el

def countNodes(node):
    count = 1

    if node.l:
        count += countNodes(node.l)
    if node.r:
        count += countNodes(node.r)

    return count

def removeSubtree(node, key, prev=None, richtung=None):
    val = node.val
    if key == val:
        count = countNodes(node)
        if richtung == 'l':
```



```

        prev.l = None
        if richtung == 'r':
            prev.r = None
        return count
    if key < val:
        if node.l:
            return removeSubtree(node.l, key, node, 'l')
        else:
            return 0
    if key > val:
        if node.r:
            return removeSubtree(node.r, key, node, 'r')
        else:
            return 0

n = int(lines[0])
inputs = []
node_count = 0
for i in range(n):
    arr = list(map(int, lines[i+1].split()))
    inputs.append(arr)
    node_count += 1

root = Node(inputs[0][0])
if inputs[0][1] == 0:
    root.l = 0
else:
    root.l = insert(inputs[0][1])
if inputs[0][2] == 0:
    root.r = 0
else:
    root.r = insert(inputs[0][2])
m = int(lines[n+1])
queue = list(map(int, lines[n+2].split()))
for i in range(m):
    node_count -= removeSubtree(root, queue[i])
    out.write(str(node_count) + "\n")

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Был использован код из предыдущих решений. Была добавлена функция `removeSubtree`, которая рекурсивно удаляет связанные с поддеревом элементы.

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

```

6
1 -2 0 2
2 8 4 3
3 9 0 0
4 3 6 5
5 6 0 0
6 0 0 0
7 4
8 6 9 7 8

NORMAL input.txt
/projects/labs/

5
1 4
2 4
3 1

```

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

```

1
1 0 0
1
1
NORMAL input.txt
output.txt" 4L, 12
0

```

```

200000
1 463242589 10405 15307
2 -521748054 68060 123995
3 -173355167 172224 107900
4 305547885 93439 187186
5 -604380790 21435 130366
6 351951547 66401 158927
7 36203276 55075 194087
8 -362817725 66658 188905
9 -308409136 144974 70258
10 -199482744 164126 124957
11 -201067263 60191 148364
12 -647622391 54084 99009
13 560465887 91344 117578
NORMAL input.txt

200000
1 200000
2 200000
3 200000
4 200000
5 200000
6 200000
7 200000
8 200000
9 200000
10 200000
11 200000
12 200000
13 200000
NORMAL output.txt

```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0002163399985875003	15265
Пример из задачи	0.00853076099883765	18992
Верхняя граница	1.5708877470005973	71389848

диапазона значений входных данных из текста задачи		
--	--	--

Вывод по задаче:

Какой же можно сделать вывод? Конечно же нужно было писать класс дерева, но уже поздно.

### Задача №10. Проверка Корректности [2 балла]

Свойство двоичного дерева поиска можно сформулировать следующим образом: для каждой вершины дерева выполняется следующее условие:

- все ключи вершин из левого поддеревья меньше ключа вершины  $V$ ;
- все ключи вершин из правого поддеревья 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.**  $0 \leq N \leq 2 \cdot 10^5$ ,  $|K_i| \leq 10^9$ .
- На 60% от при  $0 \leq N \leq 2000$ .
- **Формат вывода / выходного файла (output.txt).** Выведите «YES», если данное во входном файле дерево является двоичным деревом поиска, и «NO», если не является.
- **Ограничение по времени.** 2 сек.
- **Ограничение по памяти.** 256 мб.

- Примеры:

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

output.txt YES

input.txt	output.txt
0	YES
input.txt	output.txt
3	NO
5 2 3	
6 0 0	
4 0 0	

- Проверить можно по [ссылке](#), OpenEdu, курс "Алгоритмы программирования и структуры данных 6 неделя, наблюдаемая задача.

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
```

```
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")
```

```
class Node:
    def __init__(self, val):
        self.l = 0
        self.r = 0
```

```

        self.val = val

def insert(i):
    k, l, r = inputs[i-1]
    el = Node(k)
    if l != 0:
        el.l = insert(l)
    else:
        el.l = 0

    if r != 0:
        el.r = insert(r)
    else:
        el.r = 0
    return el

def isBST(node, mini, maxi):
    if node.val < mini or node.val > maxi:
        return False
    if node.l != 0:
        erste = isBST(node.l, mini, node.val - 1)
    else:
        erste = True

    if node.r != 0:
        zweite = isBST(node.r, node.val+1, maxi)
    else:
        zweite = True

    return erste and zweite

n = int(lines[0])
if n == 0:
    out.write("YES")
else:
    inputs = []
    node_count = 0
    for i in range(n):
        arr = list(map(int, lines[i+1].split()))
        inputs.append(arr)
        node_count += 1

    root = Node(inputs[0][0])
    if inputs[0][1] == 0:
        root.l = 0
    else:
        root.l = insert(inputs[0][1])
    if inputs[0][2] == 0:
        root.r = 0
    else:

```

```

root.r = insert(inputs[0][2])

if isBST(root, -(10**9+1), 10**9+1):
    out.write("YES")
else:
    out.write("NO")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Рекурсивно проверяем на соответствие условиям двоичного дерева поиска. Эти условия прописаны в функции isBST

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

```

3
5 2 3
6 0 0
4 0 0

NORMAL input.txt
projects/labs/se
NO

```

```

6 6
5 -2 0 2
4 8 4 3
3 9 0 0
2 3 6 5
1 6 0 0
0 0 0

NORMAL input.txt
input.txt" 7L, 51B w
YES

```

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

```

0

NORMAL input.t
input.txt" 1L,
YES

```

```

2000000
1 5581523 52351 81214
2 -352499815 140283 147165
3 -333665846 145362 184824
4 455303164 74753 42353
5 236391191 102758 174596
6 -538608381 94576 12248
7 -207311337 78677 145258
8 983622275 133890 107968
9 -255560080 198818 87548
10 -305430626 164944 4318
11 -479318873 111798 102235
12 -922807891 129263 10597
13 -321571723 135869 56502

MAL input.txt
ut.txt" 200001L, 3321905B
YES

```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00017360099991492461	13888
Пример из задачи	0.008578999999372172	16380
Пример из задачи	0.0002506020009604981	18442
Верхняя граница диапазона значений входных данных из текста задачи	0.9254913579989079	48782682

Вывод по задаче: Можно проверить разными способами является ли двоичное дерево поиска корректным.

### Задача №12. Проверка сбалансированности [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	0

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

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
```



```

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

class Node:
    def __init__(self, val):
        self.l = 0
        self.r = 0
        self.val = val
        self.height = 0

def insert(i):
    k, l, r = inputs[i-1]
    el = Node(k)
    if l != 0:
        el.l = insert(l)
    else:
        el.l = 0

    if r != 0:
        el.r = insert(r)
    else:
        el.r = 0

    el.height = max(getHeight(el.l), getHeight(el.r)) + 1

    nodes[i-1] = el

    return el

def getHeight(node):
    if not node:
        return -1

    return node.height

def getBalance(node):
    if not node:
        return 0

    return getHeight(node.l) - getHeight(node.r)

n = int(lines[0])
nodes = [0] * n
inputs = []
for i in range(n):
    arr = list(map(int, lines[i+1].split()))

```

```

        inputs.append(arr)

root = Node(inputs[0][0])
if inputs[0][1] == 0:
    root.l = 0
else:
    root.l = insert(inputs[0][1])
if inputs[0][2] == 0:
    root.r = 0
else:
    root.r = insert(inputs[0][2])
nodes[0] = root

for i in range(n):
    out.write(str(-1*getBalance(nodes[i])) + "\n")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Добавим к каждому узлу переменную height. Она будет задаваться в зависимости от значений родителей. Проверить баланс определенного узла можно вычитая высоту левого и правого ребенка.

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



```

6
1 -2 0 2
2 8 4 3
3 9 0 0
4 3 6 5
5 6 0 0
6 0 0 0

NORMAL input.txt

6
1 -1
2 0
3 0
4 0
5 0

```

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

```

0
NORMAL input.t
input.txt" 1L,
0

```

```

1 200000
-97 70769 21213
1 92 61311 94017
2 38 91076 160680
3 58 6657 131181
4 -19 59024 41358
5 32 17098 66515
6 31 27667 86810
7 22 64660 84803
8 -99 72026 119017
9 -93 166150 119118
0 -100 51245 112791
1 8 14793 118666
2 21 161825 95718
NORMAL input.txt
1 0
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 0
11 0
12 0
13 0

```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00016436099986094632	13888
Пример из задачи	0.0002372300000038835	18333
Верхняя граница диапазона значений входных данных из текста задачи	1.1517925940001987	46285680

Вывод по задаче:

Чтобы найти дисбаланс между узлами необходимо сохранять высоту каждого узла. Баланс можно вычислять относительно левого или правого ребенка.

### Задача №1. Название задачи [N баллов]

Текст задачи.

Листинг кода. (именно листинг, а не скрины)

Текстовое объяснение решения.

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

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

Проверка задачи на (openedu, астр и тд при наличии в задаче). (скрин)

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи		
Пример из задачи		
Пример из задачи		
Верхняя граница диапазона значений входных данных из текста задачи		

Вывод по задаче:

### Задача №13. Делаю я левый поворот... [2 балла]

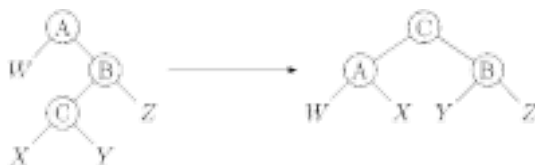
Для балансировки АВЛ-дерева при операциях вставки и удаления производятся *левые* и *правые* повороты. Левый поворот в вершине производится, когда баланс этой вершины больше 1, аналогично, правый поворот производится при балансе, меньшем -1.

Существует два разных левых (как, разумеется, и правых) поворота: *большой* и *малый* левый поворот. Малый левый поворот осуществляется следующим образом:



Заметим, что если до выполнения малого левого поворота был нарушен баланс только корня дерева, то после его выполнения все вершины становятся сбалансированными, за исключением случая, когда у правого ребенка корня баланс до поворота равен -1. В этом случае вместо малого

левого поворота выполняется большой левый поворот, который осуществляется так:



Дано дерево, в котором баланс корня равен 2. Сделайте левый поворот.

- **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. Баланс корня дерева (вершины с номером 1) равен 2, баланс всех остальных вершин находится в пределах от -1 до 1.

- **Ограничения на входные данные.**  $3 \leq N \leq 2 \cdot 10^5$ ,  $|K_i| \leq 10^9$ .
- **Формат вывода / выходного файла (output.txt).** Выведите в том же формате дерево после осуществления левого поворота. Нумерация вершин может быть произвольной при условии соблюдения формата. Так, номер вершины должен быть меньше номера ее детей.
- **Ограничение по времени.** 2 сек.
- **Ограничение по памяти.** 256 мб.

- **Пример:**

input.t xt	output.t xt
7	7
-2 7	3 2 3
2	-2 4 5

8 4 3	8 6 7
9 0 0	-7 0 0
3 6 5	0 0 0
6 0 0	6 0 0
0 0 0	9 0 0
-7 0 0	

- Проверить можно по [ссылке](#), OpenEdu, курс "Алгоритмы программирования и структуры данных 7 неделя, 2 задача.

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
```

```
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")
```

```
class Node:
    def __init__(self, val):
        self.l = 0
        self.r = 0
        self.val = val
        self.height = 0
```

```
def insert(i):
    k, l, r = inputs[i-1]
    el = Node(k)
    if l != 0:
        el.l = insert(l)
    else:
        el.l = 0

    if r != 0:
        el.r = insert(r)
    else:
        el.r = 0

    el.height = max(getHeight(el.l), getHeight(el.r)) + 1

    nodes[i-1] = el

    return el
```

```

def getHeight(node):
    if not node:
        return -1

    return node.height

def getBalance(node):
    if not node:
        return 0

    return getHeight(node.l) - getHeight(node.r)

def rotateLeft(x):
    y = x.r

    x.r = y.l
    y.l = x

    x.height = 1 + max(getHeight(x.l), getHeight(x.r))
    y.height = 1 + max(getHeight(y.l), getHeight(y.r))

    return y

```

```
antwort = []
```

```

def printOut(node, count):
    if node:
        antwort.append([node.val])
    else:
        return 0
    count2 = 0
    count3 = 0

    if node.l:
        antwort[count].append(count + 2)
        count2 = printOut(node.l, count + 1)
    else:
        antwort[count].append(0)
    if node.r:
        if count2:
            count3 = printOut(node.r, count2 + 1)
            antwort[count].append(count2+2)
        else:
            count3 = printOut(node.r, count + 1)
            antwort[count].append(count+2)
    else:
        antwort[count].append(0)

    return max(count, count2, count3)

```

```

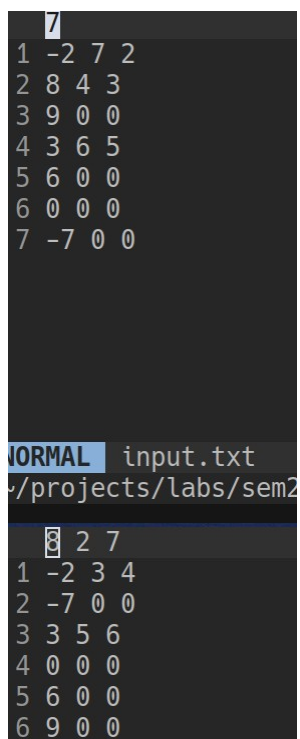
n = int(lines[0])
nodes = [0] * n
inputs = []
for i in range(n):
    arr = list(map(int, lines[i+1].split()))
    inputs.append(arr)

root = Node(inputs[0][0])
if inputs[0][1] == 0:
    root.l = 0
else:
    root.l = insert(inputs[0][1])
if inputs[0][2] == 0:
    root.r = 0
else:
    root.r = insert(inputs[0][2])
root = rotateLeft(root)
printOut(root, 0)
for i in range(n):
    out.write(" ".join(map(str, antwort[i])) + "\n")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Левый поворот можно реализовать заменой узла на правого ребенка, вместо него поставив левого ребенка. А на место левого поставить корень. Результат работы кода на примерах из текста задачи: (скрины input output файлов)



```

7
1 -2 7 2
2 8 4 3
3 9 0 0
4 3 6 5
5 6 0 0
6 0 0 0
7 -7 0 0

NORMAL input.txt
/projects/labs/sem2

8 2 7
1 -2 3 4
2 -7 0 0
3 3 5 6
4 0 0 0
5 6 0 0
6 9 0 0

```



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

```
1 4
  -2 2 3
1 -3 0 0
2 8 4 0
3 9 0 0

NORMAL input.txt
input.txt" 5L, 28B

  2 0
1 -2 3 4
2 -3 0 0
3 9 0 0
```

```
13 98 167417 67706
12 -32 17730 172078
11 1 104071 175000
10 39 2213 47955
9 -4 8733 100275
8 70 34241 168749
7 -56 54481 84798
6 10 22525 159418
5 3 13167 21926
4 82 184782 80721
3 -72 190457 42607
2 81 120641 62657
1 -64 139386 150097
  82 106590 70547
RMAL input.txt

-89 2 0
-40 3 0
-74 0 0
27 55321 124913
-72 190457 42607
  82 106590 70547
-96 191236 82302
66 29678 111874
29 150026 43811
34 52816 79956
85 46819 136977
98 167417 67706
1 104071 175000
-21 0 0
RMAL input.txt
```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00024726999981794506	18607
Пример из задачи	0.0003762870001082774	20507
Верхняя граница диапазона значений входных данных из текста задачи	0.8808973960003641	46040844

Вывод по задаче:

Осуществление левого поворота чем-то похоже на удаление из дерева. Также AVL-дерево лучше всего реализовывать через класс.

### Задача №14. Вставка в АВЛ-дерево [3 балла]

Вставка в АВЛ-дерево вершины  $V$  с ключом  $X$  при условии, что такой вершины в этом дереве нет, осуществляется следующим образом:

- находится вершина  $W$ , ребенком которой должна стать вершина  $V$ ;
- вершина  $V$  делается ребенком вершины  $W$ ;
- производится подъем от вершины  $W$  к корню, при этом, если какая-то из вершин несбалансирована, производится, в зависимости от значения баланса, левый или правый поворот.

Первый этап нуждается в пояснении. Спуск до будущего родителя вершины  $V$  осуществляется, начиная от корня, следующим образом:

- Пусть ключ текущей вершины равен  $Y$ .
- Если  $X < Y$  и у текущей вершины есть левый ребенок, переходим к левому ребенку.
- Если  $X < Y$  и у текущей вершины нет левого ребенка, то останавливаемся, текущая вершина будет родителем новой вершины.
- Если  $X > Y$  и у текущей вершины есть правый ребенок, переходим к правому ребенку.
- Если  $X > Y$  и у текущей вершины нет правого ребенка, то останавливаемся, текущая вершина будет родителем новой вершины.

Отдельно рассматривается следующий крайний случай – если до вставки дерево было пустым, то вставка новой вершины осуществляется проще: новая вершина становится корнем дерева.

- **Формат ввода / входного файла (input.txt).** Входной файл содержит описание двоичного дерева, а также ключа вершины, которую требуется вставить в дерево.

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

$Li = 0$ , если левого ребенка нет) и номера правого  $Ri$  ребенка  $i$ -ой вершины ( $i < Ri \leq N$  или  $Ri = 0$ , если правого ребенка нет).

Все ключи различны. Гарантируется, что данное дерево является корректным АВЛ-деревом.

В последней строке содержится число  $X$  – ключ вершины, которую требуется вставить в дерево. Гарантируется, что такой вершины в дереве нет.

- **Ограничения на входные данные.**  $0 \leq N \leq 2 \cdot 10^5$ ,  $|Ki| \leq 109$ ,  $|X| \leq 109$ .
- **Формат вывода / выходного файла (output.txt).** Выведите в том же формате дерево после осуществления операции вставки. Нумерация вершин может быть произвольной при условии соблюдения формата.
- **Ограничение по времени. 2 сек.**
- **Ограничение по памяти. 256 мб.**

• Пример:

input.t xt	output.t xt
2	3
3 0 2	4 2 3
4 0 0	3 0 0
5	5 0 0

- Проверить можно по [ссылке](#), OpenEdu, курс "Алгоритмы программирования и структуры данных 7 неделя, 3 задача.

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
```

```

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

```

```

class Node:

```

```

    def __init__(self, val):
        self.l = None
        self.r = None
        self.val = val
        self.height = 0

```

```

class AVLTree:

```

```

    def __init__(self):
        self.root = None

```

```

    def insert(self, key, node=None):

```

```

        threshold = 1
        if not node:
            if not self.root:
                node = Node(key)
                self.root = node
                return self.root
            return Node(key)

```

```

        if (key < node.val):
            node.l = self.insert(key, node.l)
        else:
            node.r = self.insert(key, node.r)

```

```

        node.height = max(self.getHeight(node.l), self.getHeight(node.r)) +

```

1

```

        balance = self.getBalance(node)

```

```

        if balance > threshold:
            if self.getBalance(node.l) >= 0:
                node = self.rotateRight(node)
            else:
                node = self.rotateLeftRight(node)
        elif balance < -threshold:
            if self.getBalance(node.r) <= 0:
                node = self.rotateLeft(node)
            else:
                node = self.rotateRightLeft(node)

```

```

        self.root = node

```

```

        return node

```

```

    def raw_insert(self, i, parent):

```

```

        k, l, r = inputs[i-1]
        el = Node(k)

```

```

    if parent:
        el.height = parent.height + 1
    else:
        el.height = 1
    if l != 0:
        el.l = self.raw_insert(l, el)
    else:
        el.l = None

    if r != 0:
        el.r = self.raw_insert(r, el)
    else:
        el.r = None
    if el.height == 1:
        self.root = el

    return el

def exists(self, key, node=None):
    if not node:
        node = self.root
    if key == node.val:
        return "true"
    elif key < node.val:
        if node.l:
            return self.exists(key, node.l)
    elif key > node.val:
        if node.r:
            return self.exists(key, node.r)
    return "false"

def next(self, key, node=None):
    if node == None:
        node = self.root
    if node.val <= key:
        if node.r:
            return self.next(key, node.r)
        else:
            return "none"
    else:
        if node.l:
            var = self.next(key, node.l)
            if var == "none" and node.val > key:
                return node.val
            return var
        else:
            if node.val > key:
                return node.val
            return "none"

def prev(self, key, node=None):
    if node == None:
        node = self.root

```

```

    if node.val >= key:
        if node.l:
            return self.prev(key, node.l)
        else:
            return "none"
    else:
        if node.r:
            var = self.prev(key, node.r)
            if var == "none" and node.val < key:
                return node.val
            return var
        else:
            if node.val < key:
                return node.val
            return "none"

def getMinValueNode(self, node):
    if node is None or node.l is None:
        return node

    return self.getMinValueNode(node.l)

def delete(self, key, node=None):
    if not node:
        return node

    elif key < node.val:
        node.l = self.delete(key, node.l)

    elif key > node.val:
        node.r = self.delete(key, node.r)

    else:
        # cases in which we delete a node with one child
        if node.l is None:
            temp = node.r
            node = None
            self.root = temp
            return temp

        elif node.r is None:
            temp = node.l
            node = None
            self.root = temp
            return temp

        # cases in which a node with 2 children is encountered
        temp = self.getMinValueNode(node.r)
        node.val = temp.val
        node.r = self.delete(temp.val, node.r)

    if node is None:
        self.root = node
    return node

```

1

```
node.height = max(self.getHeight(node.l), self.getHeight(node.r)) +

balance = self.getBalance(node)

if balance > 1 and self.getBalance(node.l) >= 0:
    self.root = self.rotateRight(node)
    return self.root

if balance < -1 and self.getBalance(node.r) <= 0:
    self.root = self.rotateLeft(node)
    return self.root

if balance > 1 and self.getBalance(node.l) < 0:
    self.root = self.rotateLeftRight(node)
    return self.root

if balance < -1 and self.getBalance(node.r) > 0:
    self.root = self.rotateRightLeft(node)
    return self.root

self.root = node

return node

def rotateRight(self, x):
    y = x.l

    x.l = y.r
    y.r = x

    x.height = 1 + max(self.getHeight(x.l), self.getHeight(x.r))
    y.height = 1 + max(self.getHeight(y.l), self.getHeight(y.r))

    return y

def rotateLeft(self, x):
    y = x.r

    x.r = y.l
    y.l = x

    x.height = 1 + max(self.getHeight(x.l), self.getHeight(x.r))
    y.height = 1 + max(self.getHeight(y.l), self.getHeight(y.r))

    return y

def rotateLeftRight(self, node):
    node.l = self.rotateLeft(node.l)

    return self.rotateRight(node)
```

```

def rotateRightLeft(self, node):
    node.r = self.rotateRight(node.r)

    return self.rotateLeft(node)

def getBalance(self, node):
    if not node:
        return 0

    return self.getHeight(node.l) - self.getHeight(node.r)

def getHeight(self, node):
    if not node:
        return -1

    return node.height

def printOut(node, count):
    if node:
        antwort.append([node.val])
    else:
        return 0
    count2 = 0
    count3 = 0

    if node.l:
        antwort[count].append(count + 2)
        count2 = printOut(node.l, count + 1)
    else:
        antwort[count].append(0)
    if node.r:
        if count2:
            count3 = printOut(node.r, count2 + 1)
            antwort[count].append(count2+2)
        else:
            count3 = printOut(node.r, count + 1)
            antwort[count].append(count+2)
    else:
        antwort[count].append(0)

    return max(count, count2, count3)

n = int(lines[0])
tree = AVLTree()
inputs = []
for i in range(n):
    arr = list(map(int, lines[i+1].split()))
    inputs.append(arr)
x = int(lines[n+1])

tree.raw_insert(1, None)

```



```

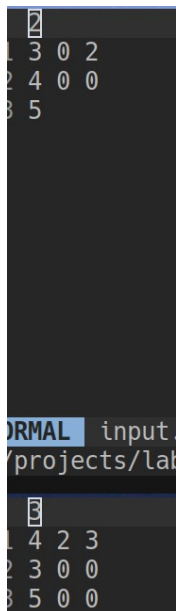
tree.insert(x, tree.root)
antwort = []
printOut(tree.root, 0)
out.write(str(len(antwort))+"\n")
for i in range(len(antwort)):
    out.write(" ".join(map(str, antwort[i])) + "\n")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

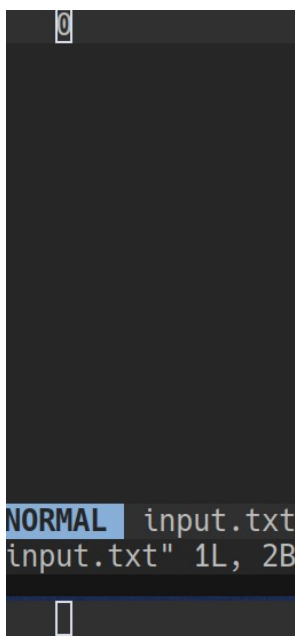
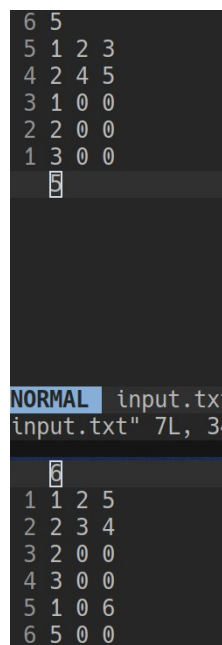
```

АВЛ-дерево было реализовано через класс AVLTree. У него имеются различные методы такие как insert и exists.

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 (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00016028800018830225	17731
Пример из задачи	0.00027420300011726795	22844
Верхняя граница диапазона значений входных данных из текста задачи	0.956889904999116	44444660

Вывод по задаче:

Самым сложным в реализации структуры АВЛ-дерева в данной задаче оказалось реализовать повороты.

### Задача №15. Удаление из АВЛ-дерева [3 балла]

Удаление из АВЛ-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, являю щегося 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 ввода / 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  $Ki$ ,  $Li$ ,  $Ri$ , разделенных пробелами – ключа  $Ki$  в  $i$ -ой вершине, номера левого  $Li$  ребенка  $i$ -ой вершины ( $i < Li \leq N$  или  $Li = 0$ , если левого ребенка нет) и номера правого  $Ri$  ребенка  $i$ -ой вершины ( $i < Ri \leq N$  или  $Ri = 0$ , 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.**  $1 \leq N \leq 2 \cdot 10^5$ ,  $|Ki| \leq 10^9$ ,  $|X| \leq 10^9$ .
- **Формат вывода / выходного файла (output.txt).** Выведите в том же формате дерево после осуществления операции удаления. Нумерация вершин может быть произвольной при условии соблюдения формата.
- **Ограничение по времени.** 2 сек.
- **Ограничение по памяти.** 256 мб.

- Пример:

input.t xt	output.t xt
3	
4 2 3	2
3 0 0	3 0 2
5 0 0	5 0 0
4	

- Проверить можно по [ссылке](#), OpenEdu, курс "Алгоритмы программирования и структуры данных 7 неделя, 4 задача.

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
```

```
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")
```

```
class Node:
    def __init__(self, val):
        self.l = None
        self.r = None
        self.val = val
        self.height = 0
```

```
class AVLTree:
    def __init__(self):
        self.root = None

    def insert(self, key, node=None):
        threshold = 1
        if not node:
            if not self.root:
                node = Node(key)
                self.root = node
            return self.root
        return Node(key)
```

1

```
    if (key < node.val):
        node.l = self.insert(key, node.l)
    else:
        node.r = self.insert(key, node.r)

    node.height = max(self.getHeight(node.l), self.getHeight(node.r)) +

    balance = self.getBalance(node)

    if balance > threshold:
        if self.getBalance(node.l) >= 0:
            node = self.rotateRight(node)
        else:
            node = self.rotateLeftRight(node)
    elif balance < -threshold:
        if self.getBalance(node.r) <= 0:
            node = self.rotateLeft(node)
        else:
            node = self.rotateRightLeft(node)

    self.root = node

    return node

def raw_insert(self, i, parent):
    k, l, r = inputs[i-1]
    el = Node(k)
    if parent:
        el.height = parent.height + 1
    else:
        el.height = 1
    if l != 0:
        el.l = self.raw_insert(l, el)
    else:
        el.l = None

    if r != 0:
        el.r = self.raw_insert(r, el)
    else:
        el.r = None
    if el.height == 1:
        self.root = el

    return el

def exists(self, key, node=None):
    if not node:
        node = self.root
    if key == node.val:
        return "true"
    elif key < node.val:
        if node.l:
            return self.exists(key, node.l)
```

```

    elif key > node.val:
        if node.r:
            return self.exists(key, node.r)
        return "false"

def next(self, key, node=None):
    if node == None:
        node = self.root
    if node.val <= key:
        if node.r:
            return self.next(key, node.r)
        else:
            return "none"
    else:
        if node.l:
            var = self.next(key, node.l)
            if var == "none" and node.val > key:
                return node.val
            return var
        else:
            if node.val > key:
                return node.val
            return "none"

def prev(self, key, node=None):
    if node == None:
        node = self.root
    if node.val >= key:
        if node.l:
            return self.prev(key, node.l)
        else:
            return "none"
    else:
        if node.r:
            var = self.prev(key, node.r)
            if var == "none" and node.val < key:
                return node.val
            return var
        else:
            if node.val < key:
                return node.val
            return "none"

def getMinValueNode(self, node):
    if node is None or node.l is None:
        return node

    return self.getMinValueNode(node.l)

def delete(self, key, node=None):
    if not node:
        return node

```

```

elif key < node.val:
    node.l = self.delete(key, node.l)

elif key > node.val:
    node.r = self.delete(key, node.r)

else:
    # cases in which we delete a node with one child
    if node.l is None:
        temp = node.r
        node = None
        self.root = temp
        return temp

    elif node.r is None:
        temp = node.l
        node = None
        self.root = temp
        return temp

    # cases in which a node with 2 children is encountered
    temp = self.getMinValueNode(node.r)
    node.val = temp.val
    node.r = self.delete(temp.val, node.r)

if node is None:
    self.root = node
    return node

node.height = max(self.getHeight(node.l), self.getHeight(node.r)) +

1

balance = self.getBalance(node)

if balance > 1 and self.getBalance(node.l) >= 0:
    self.root = self.rotateRight(node)
    return self.root

if balance < -1 and self.getBalance(node.r) <= 0:
    self.root = self.rotateLeft(node)
    return self.root

if balance > 1 and self.getBalance(node.l) < 0:
    self.root = self.rotateLeftRight(node)
    return self.root

if balance < -1 and self.getBalance(node.r) > 0:
    self.root = self.rotateRightLeft(node)
    return self.root

self.root = node

return node

```

```

def rotateRight(self, x):
    y = x.l

    x.l = y.r
    y.r = x

    x.height = 1 + max(self.getHeight(x.l), self.getHeight(x.r))
    y.height = 1 + max(self.getHeight(y.l), self.getHeight(y.r))

    return y

def rotateLeft(self, x):
    y = x.r

    x.r = y.l
    y.l = x

    x.height = 1 + max(self.getHeight(x.l), self.getHeight(x.r))
    y.height = 1 + max(self.getHeight(y.l), self.getHeight(y.r))

    return y

def rotateLeftRight(self, node):
    node.l = self.rotateLeft(node.l)

    return self.rotateRight(node)

def rotateRightLeft(self, node):
    node.r = self.rotateRight(node.r)

    return self.rotateLeft(node)

def getBalance(self, node):
    if not node:
        return 0

    return self.getHeight(node.l) - self.getHeight(node.r)

def getHeight(self, node):
    if not node:
        return -1

    return node.height

def printOut(node, count):
    if node:
        antwort.append([node.val])
    else:
        return 0
    count2 = 0
    count3 = 0

```



```

    if node.l:
        antwort[count].append(count + 2)
        count2 = printOut(node.l, count + 1)
    else:
        antwort[count].append(0)
    if node.r:
        if count2:
            count3 = printOut(node.r, count2 + 1)
            antwort[count].append(count2+2)
        else:
            count3 = printOut(node.r, count + 1)
            antwort[count].append(count+2)
    else:
        antwort[count].append(0)

    return max(count, count2, count3)

n = int(lines[0])
tree = AVLTree()
inputs = []
for i in range(n):
    arr = list(map(int, lines[i+1].split()))
    inputs.append(arr)
x = int(lines[n+1])

tree.raw_insert(1, None)
tree.delete(x, tree.root)

antwort = []
printOut(tree.root, 0)
out.write(str(len(antwort))+"\n")
for i in range(len(antwort)):
    out.write(" ".join(map(str, antwort[i])) + "\n")

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Удаление из АВЛ-дерева имеет схожий принцип работы как и в обычном двоичном дереве. Однако после удаления элемента дерево нужно сбалансировать

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

```

4 2 3
3 0 0
5 0 0
4

RMAL input.txt
projects/labs/

3 0 2
5 0 0

```

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

```

1
1 0 0
4

RMAL input.txt
put.txt" 3L, 10

1
1 0 0

```

```

2 200000
1 19 145505 186348
9 151600 81492
1 63 58486 193316
2 -77 11722 62971
3 -40 10998 174456
4 12 179028 79854
5 36 150179 8788
6 54 28550 98835
7 4 77639 27677
8 22 30078 119988
9 -29 155780 15779
10 33 81655 91871
11 61 54165 172208
RMAL input.txt
lines yanked

12 179028 79854
39 151600 81492
63 58486 193316
-77 11722 62971
-40 10998 174456
12 179028 79854
36 150179 8788
54 28550 98835
4 77639 27677
22 30078 119988
-29 155780 15779
33 81655 91871
61 54165 172208
RMAL output.txt

```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00023204399985843338	21175
Пример из задачи	0.00027193200003239326	23169
Верхняя граница диапазона значений входных данных из текста задачи	0.8435244960001	44448253

Вывод по задаче:

Удаление из AVL-дерева работает медленнее чем из стандартного. Поэтому AVL-дерево целесообразно использовать если задачей является процесс, который подразумевает частый lookup и редкую модификацию содержимого структуры

### Задача №16. K-й максимум [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  $ci$  и  $ki$  – тип и аргумент команды соответственно. Поддерживаемые команды:

- +1 (или просто 1): Добавить элемент с ключом  $ki$ .
- 0 : Найти и вывести  $ki$ -й максимум.
- -1 : Удалить элемент с ключом  $ki$ .

Гарантируется, что в процессе работы в структуре не требуется хранить элементы с равными ключами или удалять несуществующие элементы. Также гарантируется, что при запросе  $ki$ -го максимума, он существует.

- **Ограничения на входные данные.**  $n \leq 100000$ ,  $|ki| \leq 109$ .
- **Формат вывода / выходного файла (output.txt).** Для каждой команды нулевого типа в выходной файл должна быть выведена строка, содержащая единственное число –  $ki$ -й максимум.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 512 мб.
- Пример:

input.txt	output.txt
11	7

+1 5	
+1 3	
+1 7	
0 1	5
0 2	3
0 3	10
-1 5	7
+1 10	3
0 1	
0 2	
0 3	

```

import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

class Node:
    def __init__(self, val):
        self.l = None
        self.r = None
        self.val = val
        self.height = 0

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

    def insert(self, key, node=None):
        threshold = 1
        if not node:
            if not self.root:
                node = Node(key)
                self.root = node
                return self.root
            return Node(key)

        if (key < node.val):
            node.l = self.insert(key, node.l)
        else:

```

1

```

        node.r = self.insert(key, node.r)

    node.height = max(self.getHeight(node.l), self.getHeight(node.r)) + 1
    balance = self.getBalance(node)

    if balance > threshold:
        if self.getBalance(node.l) >= 0:
            node = self.rotateRight(node)
        else:
            node = self.rotateLeftRight(node)
    elif balance < -threshold:
        if self.getBalance(node.r) <= 0:
            node = self.rotateLeft(node)
        else:
            node = self.rotateRightLeft(node)

    self.root = node

    return node

def exists(self, key, node=None):
    if not node:
        node = self.root
    if key == node.val:
        return "true"
    elif key < node.val:
        if node.l:
            return self.exists(key, node.l)
    elif key > node.val:
        if node.r:
            return self.exists(key, node.r)
    return "false"

def next(self, key, node=None):
    if node == None:
        node = self.root
    if node.val <= key:
        if node.r:
            return self.next(key, node.r)
        else:
            return "none"
    else:
        if node.l:
            var = self.next(key, node.l)
            if var == "none" and node.val > key:
                return node.val
            return var
        else:
            if node.val > key:
                return node.val
            return "none"

```

```

def prev(self, key, node=None):
    if node == None:
        node = self.root
    if node.val >= key:
        if node.l:
            return self.prev(key, node.l)
        else:
            return "none"
    else:
        if node.r:
            var = self.prev(key, node.r)
            if var == "none" and node.val < key:
                return node.val
            return var
        else:
            if node.val < key:
                return node.val
            return "none"

def getMinValueNode(self, node):
    if node is None or node.l is None:
        return node

    return self.getMinValueNode(node.l)

def delete(self, key, node=None):
    if not node:
        return node

    elif key < node.val:
        node.l = self.delete(key, node.l)

    elif key > node.val:
        node.r = self.delete(key, node.r)

    else:
        # cases in which we delete a node with one child
        if node.l is None:
            temp = node.r
            node = None
            self.root = temp
            return temp

        elif node.r is None:
            temp = node.l
            node = None
            self.root = temp
            return temp

        # cases in which a node with 2 children is encountered
        temp = self.getMinValueNode(node.r)
        node.val = temp.val
        node.r = self.delete(temp.val, node.r)

```

1

```
    if node is None:
        self.root = node
        return node

    node.height = max(self.getHeight(node.l), self.getHeight(node.r)) +

    balance = self.getBalance(node)

    if balance > 1 and self.getBalance(node.l) >= 0:
        self.root = self.rotateRight(node)
        return self.root

    if balance < -1 and self.getBalance(node.r) <= 0:
        self.root = self.rotateLeft(node)
        return self.root

    if balance > 1 and self.getBalance(node.l) < 0:
        self.root = self.rotateLeftRight(node)
        return self.root

    if balance < -1 and self.getBalance(node.r) > 0:
        self.root = self.rotateRightLeft(node)
        return self.root

    self.root = node

    return node

def rotateRight(self, x):
    y = x.l

    x.l = y.r
    y.r = x

    x.height = 1 + max(self.getHeight(x.l), self.getHeight(x.r))
    y.height = 1 + max(self.getHeight(y.l), self.getHeight(y.r))

    return y

def rotateLeft(self, x):
    y = x.r

    x.r = y.l
    y.l = x

    x.height = 1 + max(self.getHeight(x.l), self.getHeight(x.r))
    y.height = 1 + max(self.getHeight(y.l), self.getHeight(y.r))

    return y

def rotateLeftRight(self, node):
    node.l = self.rotateLeft(node.l)
```

```

        return self.rotateRight(node)

    def rotateRightLeft(self, node):
        node.r = self.rotateRight(node.r)

        return self.rotateLeft(node)

    def getBalance(self, node):
        if not node:
            return 0

        return self.getHeight(node.l) - self.getHeight(node.r)

    def getHeight(self, node):
        if not node:
            return -1

        return node.height

def maxi(root, k):
    curr = root
    Klargest = None

    count = 0

    while (curr != None):
        if (curr.r == None):
            count += 1
            if (count == k):
                Klargest = curr

            curr = curr.l

        else:
            succ = curr.r

            while (succ.l != None and
                   succ.l != curr):
                succ = succ.l

            if (succ.l == None):
                succ.l = curr

            curr = curr.r

        else:
            succ.l = None
            count += 1
            if (count == k):

```



```

        Klargest = curr

    curr = curr.l

    return Klargest

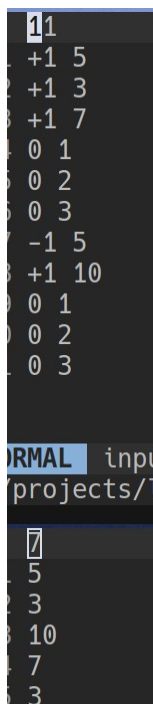
tree = AVLTree()
n = int(lines[0])
for i in range(n):
    act, x = lines[i+1].split()
    x = int(x)
    if act == "+1":
        tree.insert(x, tree.root)
    elif act == "0":
        out.write(str(maxi(tree.root, x).val) + "\n")
    elif act == "-1":
        tree.delete(x, tree.root)

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Был использован код из предыдущих задач для реализации AVL-дерева. Для поиска k-го элемента была реализована функция maxi, итеративно обходящая дерево.

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



```

1
+1 5
+1 3
+1 7
0 1
0 2
0 3
-1 5
+1 10
0 1
0 2
0 3

NORMAL input
projects/1

5
3
10
7
3

```

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

```

NORMAL input.tx
input.txt" 1L, 2

```

```

NORMAL input.txt
input.txt" 1L, 2

```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.000178371999936644	17587
Пример из задачи	0.0004745880014525028	22698
Верхняя граница диапазона значений входных данных из текста задачи	0.9306492159994377	949338

Вывод по задаче: Осуществление левого поворота чем-то похоже на удаление из дерева. Также АВЛ-дерево лучше всего реализовывать через класс.

### Задача №17. Множество с суммой [3 балла]

В этой задаче ваша цель – реализовать структуру данных для хранения набора целых чисел и быстрого вычисления суммы элементов в заданном диапазоне.

Реализуйте такую структуру данных, в которой хранится набор целых чисел  $S$  и доступны следующие операции:

- $\text{add}(i)$  – добавить число  $i$  в множество  $S$ . Если  $i$  уже есть в  $S$ , то ничего делать не надо;
- $\text{del}(i)$  – удалить число  $i$  из множества  $S$ . Если  $i$  нет в  $S$ , то ничего делать не надо;

- $\text{find}(i)$  – проверить, есть ли  $i$  во множестве  $S$  или нет;
- $\text{sum}(l, r)$  – вывести сумму всех элементов  $v$  из  $S$  таких, что  $l \leq v \leq r$ .

- **Формат ввода / входного файла (input.txt).** Изначально множество  $S$  пусто. Первая строка содержит  $n$  – количество операций. Следующие  $n$  строк содержат операции. Однако, чтобы убедиться, что ваше решение может работать в режиме онлайн, каждый запрос фактически будет зависеть от результата последнего запроса суммы. Обозначим  $M = 1\,000\,000\,001$ . В любой момент пусть  $x$  будет результатом последней операции суммирования или просто 0, если до этого операций суммирования не было. Тогда каждая операция будет являться одной из следующих:

– «+  $i$ » – добавить некоторое число в множество  $S$ . Но не само число  $i$ , а число  $((i + x) \bmod M)$ .

– «-  $i$ » – удалить из множества  $S$ , т.е.  $\text{del}((i + x) \bmod M)$ .

– «?  $i$ » –  $\text{find}((i + x) \bmod M)$ .

– «s  $l$   $r$ » – вывести сумму всех элементов множества  $S$  из определенного диапазона, т.е.  $\text{sum}((l+x) \bmod M, (r+x) \bmod M)$ .

- **Ограничения на входные данные.**  $1 \leq n \leq 30$ ,  $1 \leq i \leq 109$ .

- **Формат вывода / выходного файла (output.txt).** Для каждого запроса «find», выведите только «Found» или «Not found» (без кавычек, первая

буква заглавная) в зависимости от того, есть ли число  $((i + x) \bmod M)$  в  $S$  или нет.

Для каждого запроса суммы «sum» выведите сумму всех значений  $v$  из  $S$  из диапазона  $(l + x) \bmod M \leq v \leq (r + x) \bmod M$ , где  $x$  – результат подсчета прошлой суммы «sum», или 0, если еще не было таких операций.

- **Ограничение по времени. 120 сек. Python**

- Ограничение по памяти. 512 мб.

- Пример:

input	output
15	
? 1	
+ 1	
? 1	Not
+ 2	found
s 1 2	Found
+ 1000000000	3
? 1000000000	Found
- 1000000000	Not
? 1000000000	found
s 999999999	1
1000000000 - 2	Not
? 2	found
- 0	10
+ 9	
s 0 9	

- Пояснение. В первом примере для первых 5 операций  $x = 0$ . Для следующих 5 операций  $x = 3$ , и для оставшихся 5 операций  $x = 1$ . Добавление повторяющегося числа дважды не меняет множество. Попытки удалить элемент, которого во множестве нет, игнорируются.

- Еще примеры:

input	output	input	output
5		5	
? 0	Not	+ 491572259	Found
+	found	? 491572259	Not
0	Found	? 899375874	found
? 0	Not	s 310971296	4915722
- 0	found	877523306 +	59
? 0		352411209	

- Что делать. Используйте splay дерево для эффективного хранения множества, и добавления, удаления и поиска элементов. Для каждого узла дерева дополнительно сохраните сумму всех элементов поддеревя этого узла. Не забывайте обновлять эту сумму каждый раз при изменении дерева...

```

import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

class Node:
    def __init__(self, val):
        self.l = None
        self.r = None
        self.val = val
        self.sum = val

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

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

```

```

        return self.root

    self.root = self.splay(self.root, key)

    if self.root.val == key:
        return self.root

    node = Node(key)
    if self.root.val > key:
        node.r = self.root
        node.l = self.root.l
        self.root.l = None
    else:
        node.l = self.root
        node.r = self.root.r
        self.root.r = None

    self.root = node

    return node

def splay(self, root, key):
    if root is None or root.val == key:
        return root

    if root.val > key:
        if root.l is None:
            return root
        if root.l.val > key:
            root.l.l = self.splay(root.l.l, key)
            root = self.rotateRight(root)
        elif root.l.val < key:
            root.l.r = self.splay(root.l.r, key)
            if root.l.r is not None:
                root.l = self.rotateLeft(root.l)
            return (root.l is None) and root or self.rotateRight(root)
    else:
        if root.r is None:
            return root
        if root.r.val > key:
            root.r.l = self.splay(root.r.l, key)
            if root.r.l:
                root.r = self.rotateRight(root.r)
        elif root.r.val < key:
            root.r.r = self.splay(root.r.r, key)
            root = self.rotateLeft(root)
        return (root.r is None) and root or self.rotateLeft(root)

# def exists(self, key, node=None):
#     if not node:
#         if self.root:
#             node = self.root
#         else:

```

```

#         return "Not found"
#     if key == node.val:
#         return "Found"
#     elif key < node.val:
#         if node.l:
#             return self.exists(key, node.l)
#     elif key > node.val:
#         if node.r:
#             return self.exists(key, node.r)
#     return "Not found"
def exists(self, key):
    res = self.splay(self.root, key)
    if self.root:
        self.root = res
        return "Found" if res.val == key else "Not found"
    else:
        return "Not found"

def getMinValueNode(self, node):
    if node is None or node.l is None:
        return node

    return self.getMinValueNode(node.l)

def delete(self, key, node=None, parent=None):
    if not node:
        return node

    elif key < node.val:
        node.l = self.delete(key, node.l, node)

    elif key > node.val:
        node.r = self.delete(key, node.r, node)

    else:
        # cases in which we delete a node with one child
        if node.l is None:
            temp = node.r
            node = None
            self.root = temp
            return temp

        elif node.r is None:
            temp = node.l
            node = None
            self.root = temp
            return temp

        # cases in which a node with 2 children is encountered
        temp = self.getMinValueNode(node.r)
        node.val = temp.val
        node.r = self.delete(temp.val, node.r, node)

    if node is None:

```

```

        self.root = node
        return node

    if parent:
        mem = parent.val
        self.delete(mem, self.root)
        self.root = self.splay(self.root, mem)
    else:
        self.root = node

    return node

def next(self, key, node=None):
    if node == None:
        node = self.root
    if node.val <= key:
        if node.r:
            return self.next(key, node.r)
        else:
            return "none"
    else:
        if node.l:
            var = self.next(key, node.l)
            if var == "none" and node.val > key:
                return node.val
            return var
        else:
            if node.val > key:
                return node.val
            return "none"

def rotateRight(self, x):
    y = x.l

    x.l = y.r
    y.r = x

    return y

def rotateLeft(self, x):
    y = x.r

    x.r = y.l
    y.l = x

    return y

tree = SplayTree()
n = int(lines[0])
m = 1000000001
add = 0
for i in range(n):

```



```

inputs = lines[i+1].split()
if len(inputs) == 3:
    act, x, y = inputs
    y = int(y)
else:
    act, x = inputs
x = int(x)
if act == "+":
    tree.insert((x+add) % m)
elif act == "?":
    out.write(tree.exists((x+add) % m) + "\n")
elif act == "-":
    tree.delete((x+add) % m, tree.root)
elif act == "s":
    summ = 0
    el = (x+add) % m
    el2 = (y+add) % m
    while el != "none":
        if el <= el2:
            if tree.exists(el) == "Found":
                summ += el
            el = tree.next(el)
    add = summ
    out.write(str(summ) + "\n")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Был реализован класс SplayTree для данной структуры. Для нахождения суммы был использован метод next.

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

```

1 + 491572259
2 ? 491572259
3 ? 899375874
4 s 310971296 877523306
5 + 352411209

NORMAL input.txt
Found
1 Not found
2 491572259

```

```

3 ? 1
2 + 2
1 s 1 2
0 + 1000000000
9 ? 1000000000
8 - 1000000000
7 ? 1000000000
6 s 999999999 1000000000
5 - 2
4 ? 2
3 - 0
2 + 9
1 s 0 9
7 ?

NORMAL input.txt

Not found
1 Found
2 3
3 Found
4 Not found
5 1
6 Not found
7 10

```

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

```

1
? 1

NORMAL input.txt
input.txt" 39L, 261B

1 Not found
2 Not found
3 Not found
4 0
5 Not found
6 0
7 Not found
8 Not found
9 1595
10 Not found
11 1991
12 Not found
  
```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0001778030000423314	17705
Пример из задачи	0.000230933999773697	19913
Пример из задачи	0.00027135799973621033	22314
Верхняя граница диапазона значений входных данных из текста задачи	0.0003884369980369229	25938

Вывод по задаче:

Splay дерево эффективно для задач, где часто используются какой-то небольшой пул данных. Причем остальные данные тоже используются, но редко

## **Вывод**

С помощью различных типов двоичных деревьев поиска можно оптимизировать большой спектр прикладных задач. Например с помощью splay дерева можно реализовать быструю модификацию строки. В целом двоичные деревья поиска нужны тогда когда необходимо не только добавлять и искать, но и удалять данные из структуры.