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: BST

Выполнил:

Лазарев Марк Олегович

К3241

Санкт-Петербург

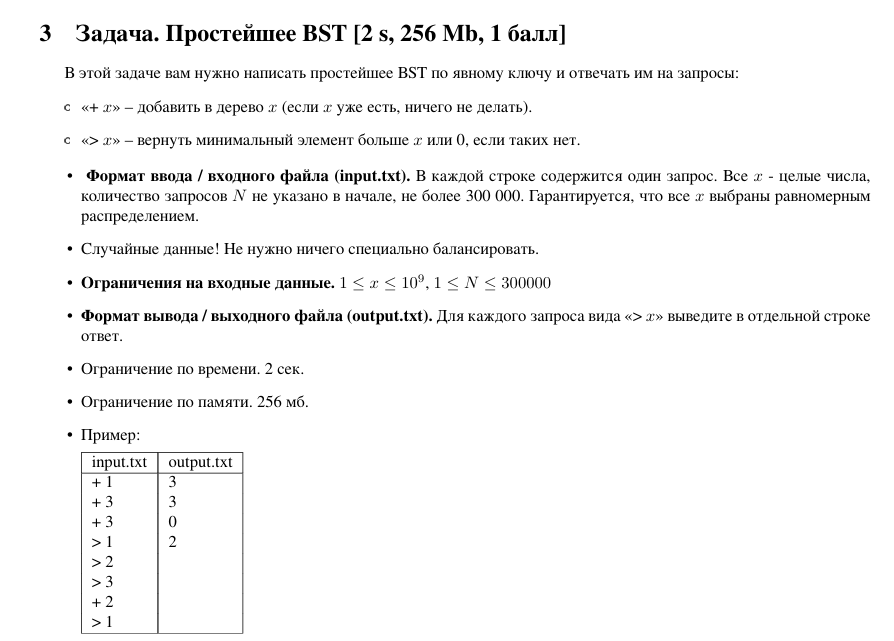
2025 г.

[**Задачи по варианту**](#_e0pij87brwfh)

[Задача №1.](#_y9xrr9qdf7ef) Простейшее BST  
[Задача №2.](#_g3d552ii7rgc) Удаление поддеревьев

[Задача №3.](#_y9xrr9qdf7ef) Делаю я левый поворот…

# Задачи по варианту



**Код программы:**

**class Node:**

**def \_\_init\_\_(self, key):**

**self.key = key**

**self.left = None**

**self.right = None**

**class BST:**

**def \_\_init\_\_(self):**

**self.root = None**

**def insert(self, key):**

**self.root = self.\_insert(self.root, key)**

**def \_insert(self, node, key):**

**if not node:**

**return Node(key)**

**if key < node.key:**

**node.left = self.\_insert(node.left, key)**

**elif key > node.key:**

**node.right = self.\_insert(node.right, key)**

**# если key == node.key, ничего не делаем**

**return node**

**def next\_greater(self, key):**

**return self.\_next\_greater(self.root, key)**

**def \_next\_greater(self, node, key):**

**result = 0**

**while node:**

**if node.key > key:**

**result = node.key**

**node = node.left**

**else:**

**node = node.right**

**return result**

**def main():**

**import sys**

**input\_file = 'input3.txt'**

**output\_file = 'output3.txt'**

**bst = BST()**

**results = []**

**with open(input\_file, 'r') as fin:**

**for line in fin:**

**if line.startswith('+'):**

**x = int(line[2:])**

**bst.insert(x)**

**elif line.startswith('>'):**

**x = int(line[2:])**

**results.append(str(bst.next\_greater(x)))**

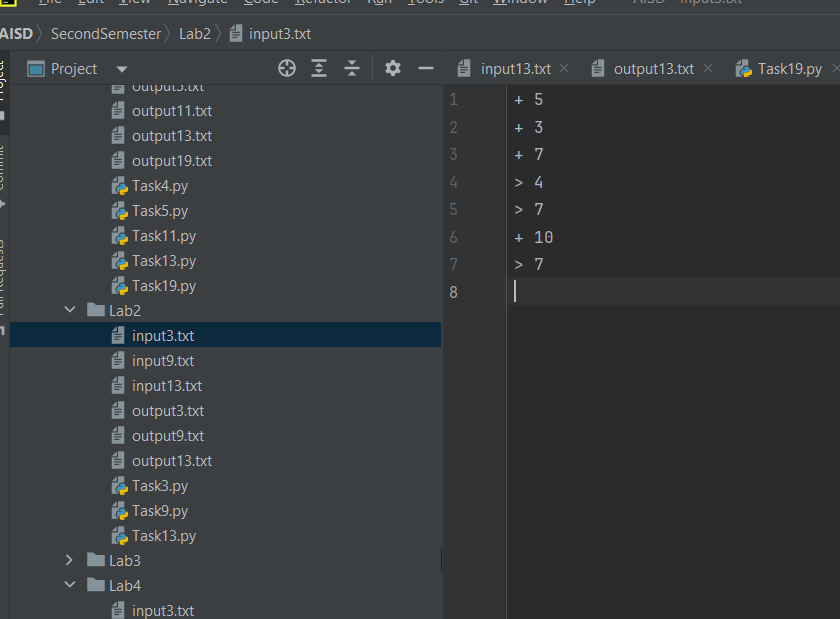
**with open(output\_file, 'w') as fout:**

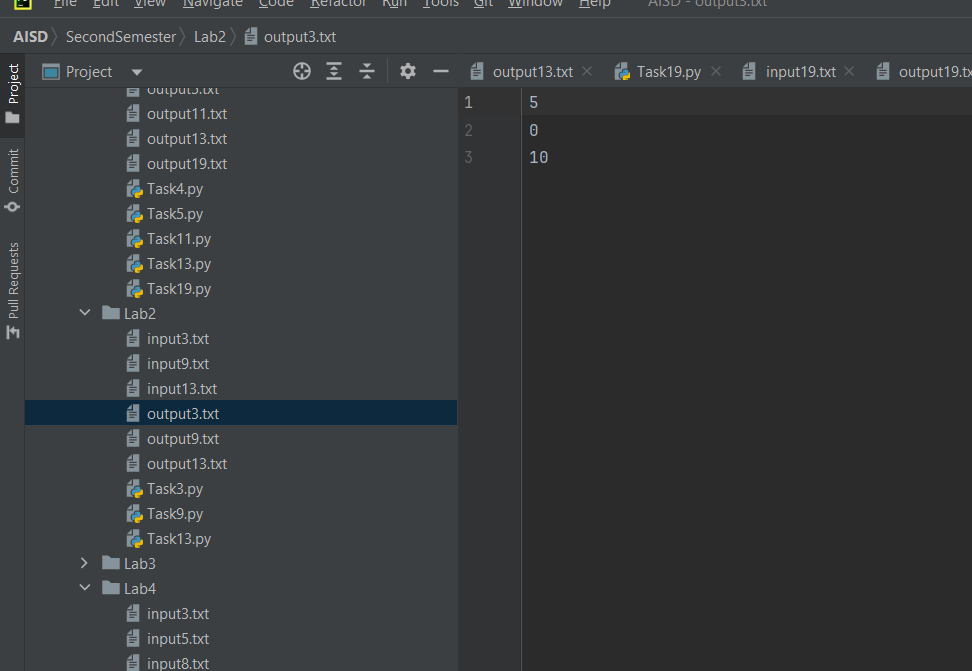
**fout.write('\n'.join(results))**

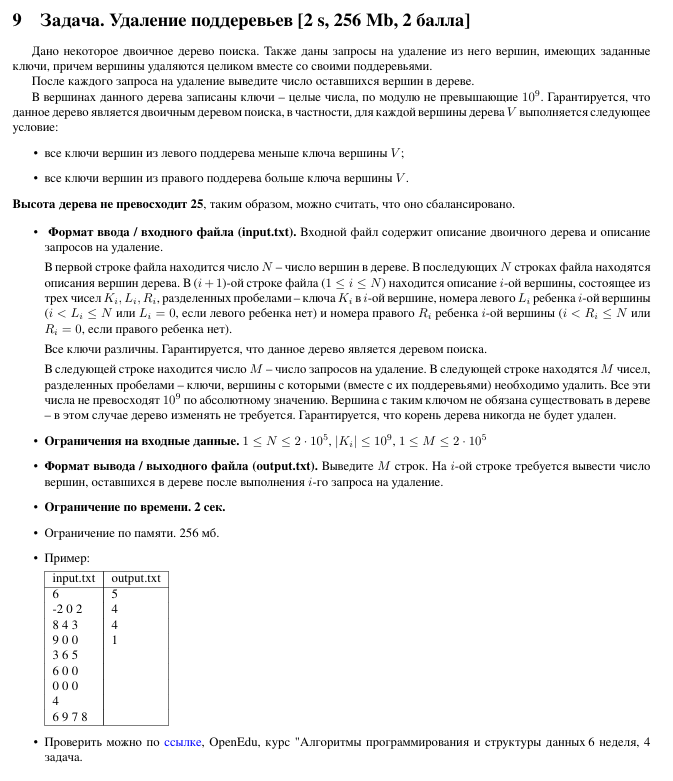
**if \_\_name\_\_ == '\_\_main\_\_':**

**main()**

Результат работы кода на примерах:







**Код программы:**

**import sys**

**sys.setrecursionlimit(1\_000\_000)**

**class Node:**

**def \_\_init\_\_(self, key):**

**self.key = key**

**self.left = None**

**self.right = None**

**self.size = 1**

**self.parent = None**

**def build\_tree(nodes\_info):**

**nodes = [None]**

**for key, \_, \_ in nodes\_info:**

**nodes.append(Node(key))**

**for i, (key, l, r) in enumerate(nodes\_info, start=1):**

**node = nodes[i]**

**if l != 0:**

**node.left = nodes[l]**

**nodes[l].parent = node**

**if r != 0:**

**node.right = nodes[r]**

**nodes[r].parent = node**

**def compute\_sizes(node):**

**if not node:**

**return 0**

**left = compute\_sizes(node.left)**

**right = compute\_sizes(node.right)**

**node.size = 1 + left + right**

**return node.size**

**compute\_sizes(nodes[1])**

**return nodes[1]**

**def find\_node(root, key):**

**node = root**

**while node:**

**if key < node.key:**

**node = node.left**

**elif key > node.key:**

**node = node.right**

**else:**

**return node**

**return None**

**def delete\_subtree(node):**

**if not node:**

**return 0**

**size = node.size**

**parent = node.parent**

**if parent:**

**if parent.left == node:**

**parent.left = None**

**elif parent.right == node:**

**parent.right = None**

**return size**

**def main():**

**with open('input9.txt', 'r') as fin:**

**n = int(fin.readline())**

**nodes\_info = [tuple(map(int, fin.readline().split())) for \_ in range(n)]**

**m = int(fin.readline())**

**deletion\_keys = list(map(int, fin.readline().split()))**

**root = build\_tree(nodes\_info)**

**total\_nodes = root.size**

**result = []**

**for key in deletion\_keys:**

**target = find\_node(root, key)**

**if target:**

**removed = delete\_subtree(target)**

**total\_nodes -= removed**

**result.append(str(total\_nodes))**

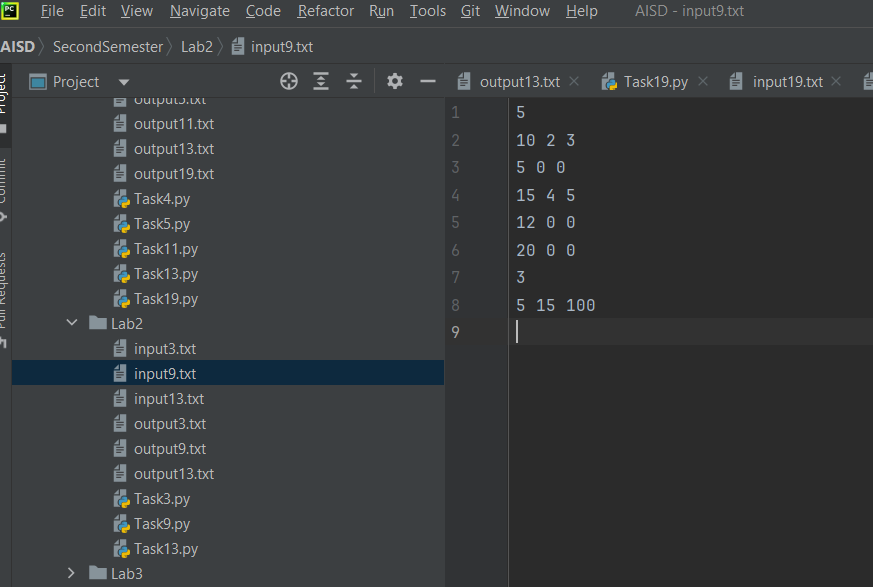
**with open('output9.txt', 'w') as fout:**

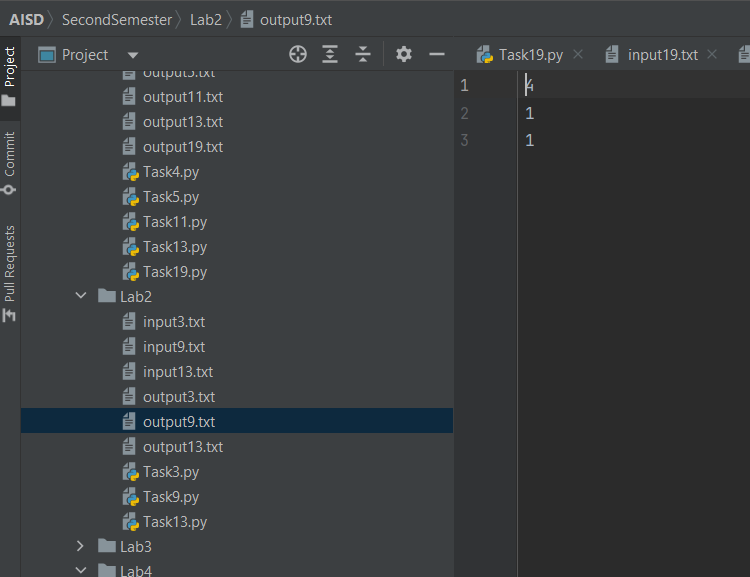
**fout.write('\n'.join(result))**

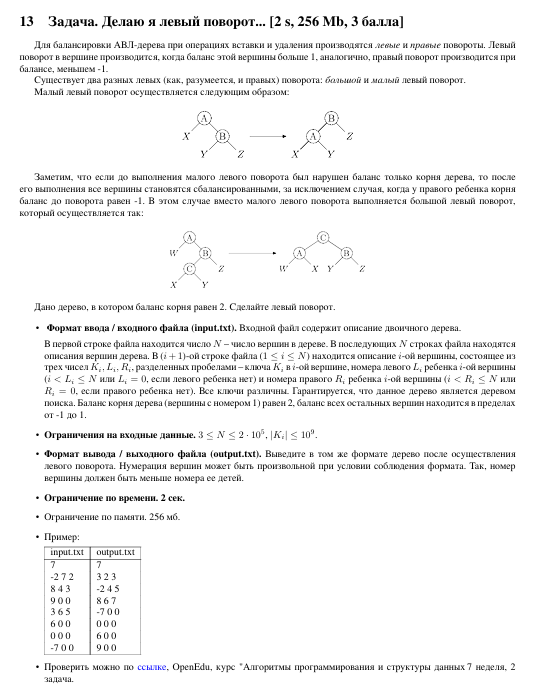
**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

Результат работы кода на примерах:

****

****



**Код программы:**

**import sys**

**sys.setrecursionlimit(1\_000\_000)**

**class Node:**

**def \_\_init\_\_(self, key, idx):**

**self.key = key**

**self.left = None**

**self.right = None**

**self.idx = idx**

**self.balance = 0**

**def build\_tree(data):**

**nodes = [None] + [Node(k, i) for i, (k, \_, \_) in enumerate(data, start=1)]**

**for i, (k, l, r) in enumerate(data, start=1):**

**nodes[i].left = nodes[l] if l != 0 else None**

**nodes[i].right = nodes[r] if r != 0 else None**

**return nodes[1]**

**def height(node):**

**if not node:**

**return 0**

**return 1 + max(height(node.left), height(node.right))**

**def compute\_balance(node):**

**if not node:**

**return 0**

**node.balance = height(node.right) - height(node.left)**

**compute\_balance(node.left)**

**compute\_balance(node.right)**

**def small\_left\_rotate(a):**

**b = a.right**

**a.right = b.left**

**b.left = a**

**return b**

**def big\_left\_rotate(a):**

**b = a.right**

**c = b.left**

**b.left = c.right**

**a.right = c.left**

**c.left = a**

**c.right = b**

**return c**

**def left\_rotate(root):**

**compute\_balance(root)**

**if root.right.balance >= 0:**

**return small\_left\_rotate(root)**

**else:**

**return big\_left\_rotate(root)**

**def serialize(root):**

**result = []**

**idx\_map = {}**

**counter = [1]**

**def dfs(node):**

**if not node:**

**return 0**

**idx = counter[0]**

**counter[0] += 1**

**idx\_map[node] = idx**

**l = dfs(node.left)**

**r = dfs(node.right)**

**result.append((node.key, l, r))**

**return idx**

**dfs(root)**

**return result[::-1]**

**def main():**

**with open("input13.txt", "r") as fin:**

**n = int(fin.readline())**

**data = [tuple(map(int, fin.readline().split())) for \_ in range(n)]**

**root = build\_tree(data)**

**root = left\_rotate(root)**

**output = serialize(root)**

**with open("output13.txt", "w") as fout:**

**fout.write(f"{len(output)}\n")**

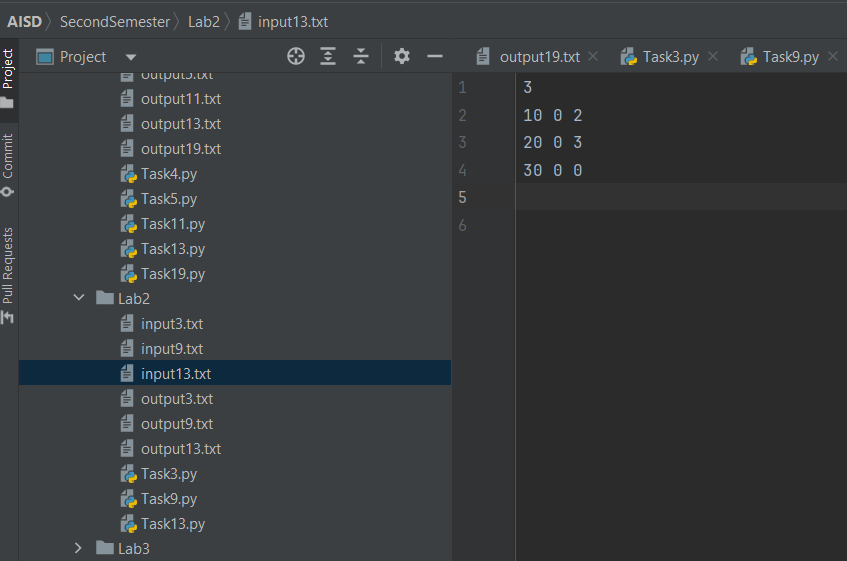
**for k, l, r in output:**

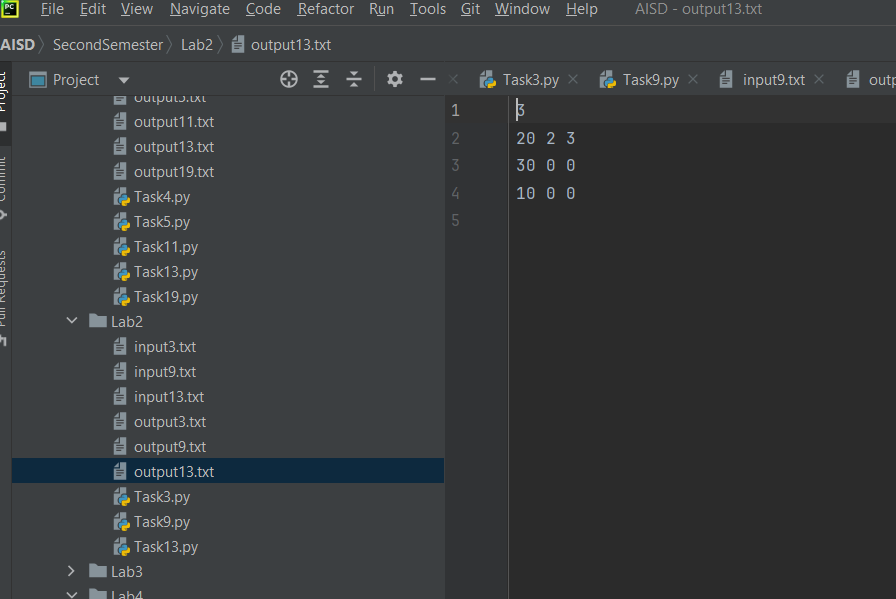
**fout.write(f"{k} {l} {r}\n")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

Результат работы

****

****