САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №2

по курсу «Алгоритмы и структуры данных»

Вариант 1

Выполнил:

Азаренков Георгий Денисович

К34421

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

[Содержание отчета 2](#_Toc179018521)

[Комментарий к общему коду 3](#_Toc179018522)

[Задача 1 4](#_Toc179018523)

[Текст задачи 4](#_Toc179018524)

[Код решения 4](#_Toc179018525)

[Результаты выполнения 4](#_Toc179018526)

[Задача 2 5](#_Toc179018527)

[Текст задачи 5](#_Toc179018528)

[Код решения 5](#_Toc179018529)

[Результаты выполнения 5](#_Toc179018530)

[Задача 12 6](#_Toc179018531)

[Текст задачи 6](#_Toc179018532)

[Код решения 6](#_Toc179018533)

[Результаты выполнения 6](#_Toc179018534)

[Задача 16 7](#_Toc179018535)

[Текст задачи 7](#_Toc179018536)

[Код решения 7](#_Toc179018537)

[Результаты выполнения 7](#_Toc179018538)

[Вывод 8](#_Toc179018539)

# Комментарий к общему коду

При использовании всех задач я использую код из файла “common.py”, который находится в корне репозитория.

from typing import List, Callable  
  
  
def solve(solver: Callable[[List[str]], List[str]]) -> None:  
 with open("../../input.txt", "r") as file:  
 lines = file.readlines()  
 input\_lines = list(map(lambda x: x.strip(), lines))  
  
 output\_lines = solver(input\_lines)  
  
 with open("../../output.txt", "w") as file:  
 for line in output\_lines:  
 file.write(f"{line}\n")

Использование функции “solve” позволило мне избавиться от необходимости раз за разом имплементировать логику чтения и сохранения данных.

# Задача 1

## Текст задачи

A paper with numbers and a number in it

Description automatically generated with medium confidence

## Код решения

from typing import List, Dict  
  
from common import solve  
  
  
class TreeNode:  
 *"""Класс для представления узла бинарного дерева."""* def \_\_init\_\_(self, key: int, left: int, right: int):  
 self.key = key  
 self.left = left  
 self.right = right  
  
  
def build\_tree(n: int, nodes\_info: List[List[int]]) -> Dict[int, TreeNode]:  
 *"""  
 Строит дерево из предоставленной информации о узлах.  
 """* tree = {}  
 for i in range(n):  
 key, left, right = nodes\_info[i]  
 tree[i] = TreeNode(key, left, right)  
 return tree  
  
  
def inorder\_traversal(node\_index: int, tree: Dict[int, TreeNode], result: List[int]) -> None:  
 *"""  
 Выполняет центрированный обход дерева.  
 """* if node\_index == -1:  
 return  
  
 *# Рекурсивно обходим левое поддерево* inorder\_traversal(tree[node\_index].left, tree, result)  
  
 *# Добавляем ключ текущего узла* result.append(tree[node\_index].key)  
  
 *# Рекурсивно обходим правое поддерево* inorder\_traversal(tree[node\_index].right, tree, result)  
  
  
def preorder\_traversal(node\_index: int, tree: Dict[int, TreeNode], result: List[int]) -> None:  
 *"""  
 Выполняет прямой обход дерева.  
 """* if node\_index == -1:  
 return  
  
 *# Добавляем ключ текущего узла* result.append(tree[node\_index].key)  
  
 *# Рекурсивно обходим левое поддерево* preorder\_traversal(tree[node\_index].left, tree, result)  
  
 *# Рекурсивно обходим правое поддерево* preorder\_traversal(tree[node\_index].right, tree, result)  
  
  
def postorder\_traversal(node\_index: int, tree: Dict[int, TreeNode], result: List[int]) -> None:  
 *"""  
 Выполняет обратный обход дерева.  
 """* if node\_index == -1:  
 return  
  
 *# Рекурсивно обходим левое поддерево* postorder\_traversal(tree[node\_index].left, tree, result)  
  
 *# Рекурсивно обходим правое поддерево* postorder\_traversal(tree[node\_index].right, tree, result)  
  
 *# Добавляем ключ текущего узла* result.append(tree[node\_index].key)  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Решает задачу обхода бинарного дерева тремя способами: in-order, pre-order, post-order.  
 """* n = int(input\_lines[0])  
  
 nodes\_info = []  
 for i in range(1, n + 1):  
 parts = list(map(int, input\_lines[i].split()))  
 nodes\_info.append(parts)  
  
 tree = build\_tree(n, nodes\_info)  
  
 in\_order = []  
 inorder\_traversal(0, tree, in\_order)  
  
 pre\_order = []  
 preorder\_traversal(0, tree, pre\_order)  
  
 post\_order = []  
 postorder\_traversal(0, tree, post\_order)  
  
 *# Формируем строки для вывода* in\_order\_str = " ".join(map(str, in\_order))  
 pre\_order\_str = " ".join(map(str, pre\_order))  
 post\_order\_str = " ".join(map(str, post\_order))  
  
 return [in\_order\_str, pre\_order\_str, post\_order\_str]  
  
  
solve(get\_solution)

## Результаты выполнения

Пример 1

input.txt

A screenshot of a computer

Description automatically generated

output.txt

A number on a white background

Description automatically generated

Пример 2

input.txt

A screenshot of a computer

Description automatically generated

output.txt

A number on a white background

Description automatically generated

# Задача 2

## Текст задачи

A paper with text and a graph

Description automatically generated

## Код решения

from typing import List, Optional  
  
from common import solve  
  
  
class TreeNode:  
 *"""  
 Узел двоичного дерева поиска.  
 """* def \_\_init\_\_(self, key: int):  
 self.key: int = key  
 self.left: Optional["TreeNode"] = None  
 self.right: Optional["TreeNode"] = None  
  
  
class BinarySearchTree:  
 *"""  
 Реализация простого двоичного дерева поиска.  
 """* def \_\_init\_\_(self):  
 self.root: Optional[TreeNode] = None  
  
 def insert(self, key: int) -> None:  
 *"""Добавляет ключ в дерево, если его еще нет."""* if self.root is None:  
 self.root = TreeNode(key)  
 return  
  
 current = self.root  
 while True:  
 if key < current.key:  
 if current.left is None:  
 current.left = TreeNode(key)  
 return  
 current = current.left  
 elif key > current.key:  
 if current.right is None:  
 current.right = TreeNode(key)  
 return  
 current = current.right  
 else:  
 *# Ключ уже существует, ничего не делаем* return  
  
 def exists(self, key: int) -> bool:  
 *"""Проверяет существование ключа в дереве."""* current = self.root  
 while current:  
 if key < current.key:  
 current = current.left  
 elif key > current.key:  
 current = current.right  
 else:  
 return True  
 return False  
  
 def find\_min(self, node: TreeNode) -> TreeNode:  
 *"""Находит минимальный узел в поддереве."""* while node.left:  
 node = node.left  
 return node  
  
 def delete(self, key: int) -> None:  
 *"""Удаляет ключ из дерева, если он существует."""* self.root = self.\_delete\_rec(self.root, key)  
  
 def \_delete\_rec(self, node: Optional[TreeNode], key: int) -> Optional[TreeNode]:  
 if node is None:  
 return None  
  
 if key < node.key:  
 node.left = self.\_delete\_rec(node.left, key)  
 elif key > node.key:  
 node.right = self.\_delete\_rec(node.right, key)  
 else:  
 *# Узел найден* if node.left is None:  
 return node.right  
 if node.right is None:  
 return node.left  
  
 *# Узел с двумя детьми* temp = self.find\_min(node.right)  
 node.key = temp.key  
 node.right = self.\_delete\_rec(node.right, temp.key)  
  
 return node  
  
 def next(self, key: int) -> Optional[int]:  
 *"""Находит минимальный элемент, строго больший заданного ключа."""* current = self.root  
 successor = None  
 while current:  
 if current.key > key:  
 successor = current.key  
 current = current.left  
 else:  
 current = current.right  
 return successor  
  
 def prev(self, key: int) -> Optional[int]:  
 *"""Находит максимальный элемент, строго меньший заданного ключа."""* current = self.root  
 predecessor = None  
 while current:  
 if current.key < key:  
 predecessor = current.key  
 current = current.right  
 else:  
 current = current.left  
 return predecessor  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Обрабатывает операции над двоичным деревом поиска и возвращает результаты операций exists, next, prev.  
 """* bst = BinarySearchTree()  
 output: List[str] = []  
  
 for line in input\_lines:  
 parts = line.strip().split()  
  
 if not parts:  
 continue  
  
 operation = parts[0]  
  
 x = int(parts[1])  
  
 if operation == "insert":  
 bst.insert(x)  
  
 elif operation == "delete":  
 bst.delete(x)  
  
 elif operation == "exists":  
 exists = bst.exists(x)  
 output.append("true" if exists else "false")  
  
 elif operation == "next":  
 nxt = bst.next(x)  
 output.append(str(nxt) if nxt is not None else "none")  
  
 elif operation == "prev":  
 prv = bst.prev(x)  
 output.append(str(prv) if prv is not None else "none")  
  
 return output  
  
  
solve(get\_solution)

## Результаты выполнения

input.txt

A screenshot of a computer

Description automatically generated

output.txt

A screenshot of a computer

Description automatically generated

# Задача 12

## Текст задачи

A paper with text and numbers

Description automatically generated

## Код решения

from typing import List  
  
from common import solve  
  
  
class Node:  
 *"""  
 Класс для представления узла бинарного дерева поиска.  
 """* def \_\_init\_\_(self, key: int, left: int, right: int):  
 self.key: int = key  
 self.left: int = left  
 self.right: int = right  
  
  
*# Функция для вычисления высоты поддерева*def compute\_height(index: int, nodes: List[Node], balance\_factors: List[int]) -> int:  
 if index == 0:  
 return 0  
  
 *# Получение соответствующего узла* node = nodes[index - 1]  
  
 *# Рекурсивный вызов для левого поддерева* left\_height = compute\_height(node.left, nodes, balance\_factors)  
  
 *# Рекурсивный вызов для правого поддерева* right\_height = compute\_height(node.right, nodes, balance\_factors)  
  
 *# Вычисление баланс-фактора текущего узла* balance\_factors[index] = right\_height - left\_height  
  
 *# Вычисление высоты текущего поддерева* current\_height = 1 + max(left\_height, right\_height)  
  
 return current\_height  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *# Парсинг количества вершин* n = int(input\_lines[0])  
  
 if n == 0:  
 return []  
  
 *# Создание списка узлов* nodes: List[Node] = []  
  
 for i in range(1, n + 1):  
 parts: List[int] = list(map(int, input\_lines[i].strip().split()))  
 key, left, right = parts  
 nodes.append(Node(key, left, right))  
  
 *# Массив для хранения баланс-факторов* balance\_factors = [0] \* (n + 1)  
  
 *# Вычисление высот и баланс-факторов начиная с корня (вершина 1)* compute\_height(1, nodes, balance\_factors)  
  
 *# Подготовка выходных данных* output: List[str] = []  
  
 for i in range(1, n + 1):  
 output.append(str(balance\_factors[i]))  
  
 return output  
  
  
solve(get\_solution)

## Результаты выполнения

input.txt

A screenshot of a cell phone

Description automatically generated

output.txt

A screenshot of a computer

Description automatically generated

# Задача 16

## Текст задачи

A paper with text and numbers

Description automatically generated

## Код решения

import bisect  
from typing import List  
  
from common import solve  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Решает задачу управления структурой данных для добавления, удаления элементов  
 и нахождения k-го максимума.  
 """* n = int(input\_lines[0]) *# Читаем количество команд* commands = input\_lines[1:] *# Остальные строки - команды* sorted\_list = [] *# Список для хранения элементов в отсортированном порядке* output = [] *# Список для хранения результатов команд типа 0* for command in commands:  
 parts = command.split()  
 if len(parts) == 2:  
 ci, ki = parts  
 ki = int(ki)  
 else:  
 ci = parts[0]  
 ki = None  
  
 if ci in ["+1", "1"]:  
 *# Добавляем элемент ki в отсортированный список* bisect.insort\_left(sorted\_list, ki)  
  
 elif ci == "-1":  
 *# Удаляем элемент ki из отсортированного списка* index = bisect.bisect\_left(sorted\_list, ki)  
 if index < len(sorted\_list) and sorted\_list[index] == ki:  
 sorted\_list.pop(index)  
  
 elif ci == "0":  
 *# Находим ki-й максимум  
 # Так как список отсортирован по возрастанию, k-й максимум - это элемент с индекс -k* kth\_max = sorted\_list[-ki]  
 output.append(str(kth\_max))  
  
 return output  
  
  
solve(get\_solution)

## Результаты выполнения

input.txt

A screenshot of a computer

Description automatically generated

output.txt

A screenshot of a computer

Description automatically generated

# Вывод

В ходе данной лабораторной работы я научился решать задачи по теме. Написанные программы были протестированы с измерением затрат времени и оперативной памяти. Все программы работаю корректно и укладываются в установленные ограничения.