САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №5

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](#_Toc179016137)

[Комментарий к общему коду 3](#_Toc179016138)

[Задача 1 4](#_Toc179016139)

[Текст задачи 4](#_Toc179016140)

[Код решения 4](#_Toc179016141)

[Результаты выполнения 5](#_Toc179016142)

[Задача 2 7](#_Toc179016143)

[Текст задачи 7](#_Toc179016144)

[Код решения 7](#_Toc179016145)

[Результаты выполнения 8](#_Toc179016146)

[Вывод 9](#_Toc179016147)

# 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 screenshot of a question

Description automatically generated

## Код решения

from typing import List  
  
from common import solve  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Проверяет, является ли заданный массив неубывающей пирамидой.  
 Возвращает "YES", если массив соответствует условиям, и "NO" в противном случае.  
 """  
  
 # Преобразование первой строки в целое число n* n = int(input\_lines[0])  
  
 *# Преобразование второй строки в список целых чисел* array = list(map(int, input\_lines[1].split()))  
  
 *# Итерация по каждому элементу массива для проверки условий кучи* for i in range(1, n + 1):  
 left\_index = 2 \* i  
 right\_index = 2 \* i + 1  
  
 *# Проверка наличия левого потомка и сравнение значений* if left\_index <= n:  
 if array[i - 1] > array[left\_index - 1]:  
 return ["NO"]  
  
 *# Проверка наличия правого потомка и сравнение значений* if right\_index <= n:  
 if array[i - 1] > array[right\_index - 1]:  
 return ["NO"]  
  
 *# Если все условия выполнены, возвращаем "YES"* return ["YES"]  
  
  
solve(get\_solution)

## Результаты выполнения

Пример 1

input.txt

A number on a white background

Description automatically generated

output.txt

A screenshot of a computer

Description automatically generated

Пример 2

input.txt

A number on a white background

Description automatically generated

output.txt

A close up of a text

Description automatically generated

# Задача 2

## Текст задачи

A paper with text and numbers

Description automatically generated

## Код решения

from typing import List  
  
from common import solve  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Вычисляет высоту произвольного дерева, заданного списком родительских узлов.  
 Функция принимает список строк, представляющих содержимое input.txt, и возвращает список строк для output.txt.  
 """  
  
 # Чтение количества узлов* n = int(input\_lines[0])  
  
 *# Чтение родительских узлов и конвертация в список целых чисел* parent = list(map(int, input\_lines[1].split()))  
  
 *# Инициализация списка смежности для представления дерева* children = [[] for \_ in range(n)]  
  
 root = -1  
  
 for i in range(n):  
 if parent[i] == -1:  
 *# Определение корневого узла* root = i  
 else:  
 *# Добавление дочернего узла к родителю* children[parent[i]].append(i)  
  
 *# Инициализация переменной для хранения высоты дерева* height = 0  
  
 *# Инициализация очереди для обхода дерева в ширину (BFS)* queue = [(root, 1)] *# Кортеж содержит узел и его текущую глубину* while queue:  
 current\_node, current\_depth = queue.pop(0)  
  
 *# Обновление максимальной высоты* if current\_depth > height:  
 height = current\_depth  
  
 *# Добавление дочерних узлов в очередь с увеличенной глубиной* for child in children[current\_node]:  
 queue.append((child, current\_depth + 1))  
  
 *# Возвращение высоты дерева в виде списка строк* return [str(height)]  
  
  
solve(get\_solution)

## Результаты выполнения

Пример 1

input.txt

A number on a white background

Description automatically generated

output.txt

A calendar with numbers and a number on it

Description automatically generated

Пример 2

input.txt

A number on a white background

Description automatically generated

output.txt

A screenshot of a calendar

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.