САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №4

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](#_Toc179015821)

[Комментарий к общему коду 3](#_Toc179015822)

[Задача 1 5](#_Toc179015823)

[Текст задачи 5](#_Toc179015824)

[Код решения 5](#_Toc179015825)

[Результаты выполнения 6](#_Toc179015826)

[Задача 3 7](#_Toc179015827)

[Текст задачи 7](#_Toc179015828)

[Код решения 7](#_Toc179015829)

[Результаты выполнения 8](#_Toc179015830)

[Задача 6 10](#_Toc179015831)

[Текст задачи 10](#_Toc179015832)

[Код решения 10](#_Toc179015833)

[Результаты выполнения 11](#_Toc179015834)

[Задача 7 13](#_Toc179015835)

[Текст задачи 13](#_Toc179015836)

[Код решения 13](#_Toc179015837)

[Результаты выполнения 14](#_Toc179015838)

[Вывод 15](#_Toc179015839)

# 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 text and numbers

Description automatically generated

## Код решения

from typing import List  
  
from common import solve  
  
  
def get\_solution(commands: List[str]) -> List[str]:  
 *"""  
 Функция реализует работу стека.  
 Принимает массив строк с командами и возвращает массив строк с результатами операций извлечения.  
 """* stack = []  
 results = []  
  
 for command in commands[1:]:  
 if command.startswith("+"):  
 *# Извлекаем число из команды и добавляем его в стек* \_, num\_str = command.split()  
 num = int(num\_str)  
 stack.append(num)  
  
 elif command == "-":  
 *# Извлекаем верхний элемент из стека и добавляем его в результаты* popped = stack.pop()  
 results.append(str(popped))  
  
 return results  
  
  
solve(get\_solution)

## Результаты выполнения

input.txt

A screenshot of a math test

Description automatically generated

output.txt

A close up of numbers

Description automatically generated

# Задача 3

## Текст задачи

A text on a piece of paper

Description automatically generated

## Код решения

from typing import List  
  
from common import solve  
  
  
def is\_valid(sequence: str) -> bool:  
 *"""  
 Проверяет, является ли данная последовательность скобок правильной.  
 """* stack = []  
 bracket\_map = {")": "(", "]": "["}  
  
 for char in sequence:  
 if char in bracket\_map.values():  
 *# Если символ открывающей скобки, добавляем его в стек* stack.append(char)  
  
 elif char in bracket\_map:  
 *# Если символ закрывающей скобки, проверяем соответствие* if not stack:  
 return False  
  
 top = stack.pop()  
  
 if bracket\_map[char] != top:  
 return False  
  
 else:  
 *# Если символ не скобка, игнорируем (по условию такие не встречаются)* pass  
  
 *# Если стек пуст, все скобки сбалансированы* return not stack  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Функция проверяет каждую скобочную последовательность из входного списка и возвращает список "YES" или "NO" в зависимости от того, является ли последовательность правильной.  
 """* results = []  
  
 n = int(input\_lines[0])  
  
 for i in range(1, n + 1):  
 sequence = input\_lines[i].strip()  
  
 if is\_valid(sequence):  
 results.append("YES")  
 else:  
 results.append("NO")  
  
 return results  
  
  
solve(get\_solution)

## Результаты выполнения

input.txt

A screenshot of a computer

Description automatically generated

output.txt

A screenshot of a cell phone

Description automatically generated

# Задача 6

## Текст задачи

A page of a math test

Description automatically generated with medium confidence

## Код решения

from collections import deque  
from typing import List  
  
from common import solve  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Реализует очередь с операциями добавления, удаления и запроса минимального элемента.  
 """  
  
 # Чтение количества команд* m = int(input\_lines[0])  
  
 *# Инициализация основной очереди и очереди для минимальных элементов* queue = deque()  
 min\_queue = deque()  
  
 *# Список для хранения результатов запросов* results = []  
  
 *# Обработка каждой команды* for line in input\_lines[1 : m + 1]:  
 parts = line.strip().split()  
  
 if parts[0] == "+":  
 number = int(parts[1])  
  
 *# Добавление элемента в основную очередь* queue.append(number)  
  
 *# Удаление элементов из min\_queue, которые больше добавляемого числа* while min\_queue and min\_queue[-1] > number:  
 min\_queue.pop()  
  
 *# Добавление нового элемента в очередь минимумов* min\_queue.append(number)  
  
 elif parts[0] == "-":  
 *# Извлечение элемента из основной очереди* removed = queue.popleft()  
  
 *# Если извлеченный элемент равен текущему минимуму, удаляем его из min\_queue* if removed == min\_queue[0]:  
 min\_queue.popleft()  
  
 elif parts[0] == "?":  
 *# Добавление текущего минимума в результаты* results.append(str(min\_queue[0]))  
  
 return results  
  
  
solve(get\_solution)

## Результаты выполнения

input.txt

A screenshot of a math test

Description automatically generated

output.txt

A screenshot of a computer

Description automatically generated

# Задача 7

## Текст задачи

A paper with text and numbers

Description automatically generated

## Код решения

from collections import deque  
from typing import List  
  
from common import solve  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Находит максимумы в каждом окне заданной ширины в массиве чисел.  
 """  
  
 # Парсинг входных данных* n: int = int(input\_lines[0].strip())  
  
 array: List[int] = list(map(int, input\_lines[1].strip().split()))  
  
 m: int = int(input\_lines[2].strip())  
  
 *# Инициализация деки для хранения индексов элементов* dq: deque = deque()  
  
 result: List[int] = []  
  
 for i in range(n):  
 *# Удаление индексов элементов, которые вышли за пределы окна* while dq and dq[0] <= i - m:  
 dq.popleft()  
  
 *# Удаление индексов элементов, меньших текущего элемента* while dq and array[i] >= array[dq[-1]]:  
 dq.pop()  
  
 *# Добавление текущего индекса в деку* dq.append(i)  
  
 *# Добавление максимума текущего окна в результат* if i >= m - 1:  
 result.append(array[dq[0]])  
  
 *# Преобразование результатов в строку* output: List[str] = [" ".join(map(str, result))]  
  
 return output  
  
  
solve(get\_solution)

## Результаты выполнения

input.txt

A number on a white background

Description automatically generated

output.txt

A number on a white background

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.