САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

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](#_Toc179018003)

[Комментарий к общему коду 3](#_Toc179018004)

[Задача 1 5](#_Toc179018005)

[Текст задачи 5](#_Toc179018006)

[Код решения 5](#_Toc179018007)

[Результаты выполнения 6](#_Toc179018008)

[Задача 3 8](#_Toc179018009)

[Текст задачи 8](#_Toc179018010)

[Код решения 8](#_Toc179018011)

[Результаты выполнения 9](#_Toc179018012)

[Задача 8 10](#_Toc179018013)

[Текст задачи 10](#_Toc179018014)

[Код решения 10](#_Toc179018015)

[Результаты выполнения 11](#_Toc179018016)

[Задача 17 13](#_Toc179018017)

[Текст задачи 13](#_Toc179018018)

[Код решения 13](#_Toc179018019)

[Результаты выполнения 14](#_Toc179018020)

[Вывод 16](#_Toc179018021)

# 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(input\_data: List[str]) -> List[str]:  
 *"""  
 Решает задачу о дробном рюкзаке. Вычисляет максимальную стоимость предметов, которые можно поместить в рюкзак.  
 """  
  
 # Разбор первой строки для получения количества предметов и вместимости рюкзака* first\_line = input\_data[0].strip().split()  
 n = int(first\_line[0])  
 W = int(first\_line[1])  
  
 items = []  
  
 *# Считывание данных о каждом предмете* for i in range(1, n + 1):  
 p, w = map(int, input\_data[i].strip().split())  
 if w == 0:  
 value\_per\_weight = 0  
 else:  
 value\_per\_weight = p / w  
 items.append((value\_per\_weight, p, w))  
  
 *# Сортировка предметов по убыванию стоимости на единицу веса* items.sort(reverse=True, key=lambda x: x[0])  
  
 total\_value = 0.0  
 remaining\_capacity = W  
  
 *# Выбор предметов для рюкзака* for value\_per\_weight, p, w in items:  
 if remaining\_capacity == 0:  
 break  
  
 if w <= remaining\_capacity:  
 total\_value += p  
 remaining\_capacity -= w  
 else:  
 fraction = remaining\_capacity / w  
 total\_value += p \* fraction  
 remaining\_capacity = 0  
  
 *# Форматирование результата с четырьмя знаками после запятой* result = f"{total\_value:.4f}"  
  
 return [result]  
  
  
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 close up of numbers

Description automatically generated

output.txt

A close up of numbers

Description automatically generated

# Задача 3

## 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]:  
 *"""  
 Распределяет объявления по слотам для максимизации общего дохода.  
 """  
  
 # Чтение числа объявлений* n = int(input\_lines[0])  
  
 *# Чтение прибыли за клик для каждого объявления* a = list(map(int, input\_lines[1].split()))  
  
 *# Чтение ожидаемого количества кликов для каждого слота* b = list(map(int, input\_lines[2].split()))  
  
 *# Сортировка прибыли по убыванию* a\_sorted = sorted(a, reverse=True)  
  
 *# Сортировка кликов по убыванию* b\_sorted = sorted(b, reverse=True)  
  
 *# Вычисление максимальной суммы произведений* total = 0  
 for i in range(n):  
 total += a\_sorted[i] \* b\_sorted[i]  
  
 return [str(total)]  
  
  
solve(get\_solution)

## Результаты выполнения

Пример 1

input.txt

A white background with black numbers

Description automatically generated

output.txt

A number on a white background

Description automatically generated

Пример 2

input.txt

A white background with black numbers

Description automatically generated

output.txt

A screenshot of a calendar

Description automatically generated

# Задача 8

## 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]:  
 *"""  
 Решает задачу выбора максимального количества непересекающихся лекций.  
 """* n = int(input\_lines[0])  
  
 *# Создаем список пар (начало, конец) для каждой лекции* lectures: List[tuple] = []  
  
 for i in range(1, n + 1):  
 start, end = map(int, input\_lines[i].split())  
 lectures.append((start, end))  
  
 *# Сортируем лекции по времени окончания* lectures.sort(key=lambda x: x[1])  
  
 *# Инициализируем счетчик выбранных лекций и время окончания последней выбранной лекции* count = 0  
 last\_end\_time = -1  
  
 *# Проходим по отсортированным лекциям и выбираем непересекающиеся* for lecture in lectures:  
 start, end = lecture  
  
 if start >= last\_end\_time:  
 count += 1  
 last\_end\_time = end  
  
 return [str(count)]  
  
  
solve(get\_solution)

## Результаты выполнения

Пример 1

input.txt

A number on a white background

Description automatically generated

output.txt

A number on a calendar

Description automatically generated

Пример 2

input.txt

A screenshot of a computer

Description automatically generated

output.txt

A screenshot of a calendar

Description automatically generated

# Задача 17

## Текст задачи

A paper with text and numbers

Description automatically generated

## Код решения

from typing import List  
  
from common import solve  
  
MOD = 10\*\*9  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Определяет количество телефонных номеров заданной длины, которые можно набрать ходом коня на клавиатуре.  
 """* n = int(input\_lines[0])  
  
 *# Определяем переходы для каждого ключа* moves: dict[str, List[str]] = {  
 "1": ["6", "8"],  
 "2": ["7", "9"],  
 "3": ["4", "8"],  
 "4": ["3", "9", "0"],  
 "5": [],  
 "6": ["1", "7", "0"],  
 "7": ["2", "6"],  
 "8": ["1", "3"],  
 "9": ["2", "4"],  
 "0": ["4", "6"],  
 }  
  
 *# Начальные цифры (не 0 и не 8)* initial\_digits: List[str] = ["1", "2", "3", "4", "5", "6", "7", "9"]  
  
 *# Инициализируем динамическое программирование* dp\_prev: dict[str, int] = {digit: 1 for digit in initial\_digits}  
  
 *# Обрабатываем номера длины от 2 до N* for \_ in range(2, n + 1):  
 dp\_current: dict[str, int] = {digit: 0 for digit in moves.keys()}  
  
 for digit, count in dp\_prev.items():  
 for next\_digit in moves[digit]:  
 dp\_current[next\_digit] = (dp\_current[next\_digit] + count) % MOD  
  
 dp\_prev = dp\_current  
  
 *# Вычисляем итоговую сумму по всем возможным цифрам* total = sum(dp\_prev.values()) % MOD  
  
 return [str(total)]  
  
  
solve(get\_solution)

## 

## Результаты выполнения

Пример 1

input.txt

A number on a calendar

Description automatically generated

output.txt

A screenshot of a calendar

Description automatically generated

Пример 2

input.txt



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.