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](#_Toc179012155)

[Комментарий к общему коду 3](#_Toc179012156)

[Задача 1 4](#_Toc179012157)

[Текст задачи 4](#_Toc179012158)

[Код решения 4](#_Toc179012159)

[Результат выполнения 5](#_Toc179012160)

[Вывод 6](#_Toc179012161)

# 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 math test

Description automatically generated

## Код решения

from typing import List  
  
from common import solve  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Находит все вхождения строки p в строку t и возвращает количество вхождений и их позиции.  
 """* p: str = input\_lines[0]  
 t: str = input\_lines[1]  
  
 occurrences: List[int] = []  
 p\_length: int = len(p)  
 t\_length: int = len(t)  
  
 *# Проходим по строке t, чтобы найти все вхождения p* for i in range(t\_length - p\_length + 1):  
  
 *# Проверяем, совпадает ли подстрока с p* if t[i : i + p\_length] == p:  
  
 *# Добавляем позицию (1-индексация) в список вхождений* occurrences.append(i + 1)  
  
 *# Формируем результат: количество вхождений и их позиции* count: int = len(occurrences)  
 positions: str = " ".join(map(str, occurrences))  
  
 return [str(count), positions]  
  
  
solve(get\_solution)

## Результаты выполнения

input.txt

A close up of a logo

Description automatically generated

output.txt

A number on a white background

Description automatically generated

# Задача 3

## Текст задачи

A screenshot of a computer test

Description automatically generated

## Код решения

from typing import List  
  
from common import solve  
  
  
def get\_solution(input: List[str]) -> List[str]:  
 *"""  
 Реализует алгоритм Рабина-Карпа для поиска всех вхождений шаблона P в текст T.  
 Возвращает количество вхождений и их начальные позиции.  
 """  
  
 # Извлечение паттерна P и текста T из входных данных* P = input[0].rstrip("\n")  
 T = input[1].rstrip("\n")  
  
 len\_P = len(P)  
 len\_T = len(T)  
  
 *# База и модуль для хеширования* base = 256  
 mod = 10\*\*9 + 7  
  
 *# Предварительный расчет степени базы* power = 1  
 for \_ in range(len\_P - 1):  
 power = (power \* base) % mod  
  
 *# Вычисление хеша паттерна* hash\_P = 0  
 for char in P:  
 hash\_P = (hash\_P \* base + ord(char)) % mod  
  
 *# Вычисление начального хеша окна текста* hash\_T = 0  
  
 for char in T[:len\_P]:  
 hash\_T = (hash\_T \* base + ord(char)) % mod  
  
 result\_positions: List[int] = []  
  
 *# Сравнение хешей и проверка подстроки на совпадение* for i in range(len\_T - len\_P + 1):  
 *# Если хеши совпадают, проверить подстроки символ за символом* if hash\_T == hash\_P:  
 if T[i : i + len\_P] == P:  
 result\_positions.append(i + 1) *# Нумерация с 1  
  
 # Вычисление хеша следующего окна текста* if i < len\_T - len\_P:  
 hash\_T = (hash\_T - ord(T[i]) \* power) % mod  
 hash\_T = (hash\_T \* base + ord(T[i + len\_P])) % mod  
 hash\_T = (hash\_T + mod) % mod *# Обеспечить положительное значение  
  
 # Формирование выходных данных* count = len(result\_positions)  
 positions\_str = " ".join(map(str, result\_positions))  
  
 return [str(count), positions\_str]  
  
  
solve(get\_solution)

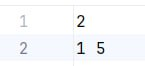
## Результаты выполнения

input.txt

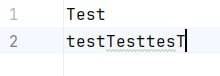
A close up of a word

Description automatically generated

output.txt



Пример 2



A number on a white background

Description automatically generated

Пример 3

A close up of a word

Description automatically generated

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.