САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №4

по курсу «Алгоритмы и структуры данных»

Тема: Substrings

Выполнил:

Лазарев Марк Олегович

К3241

Санкт-Петербург

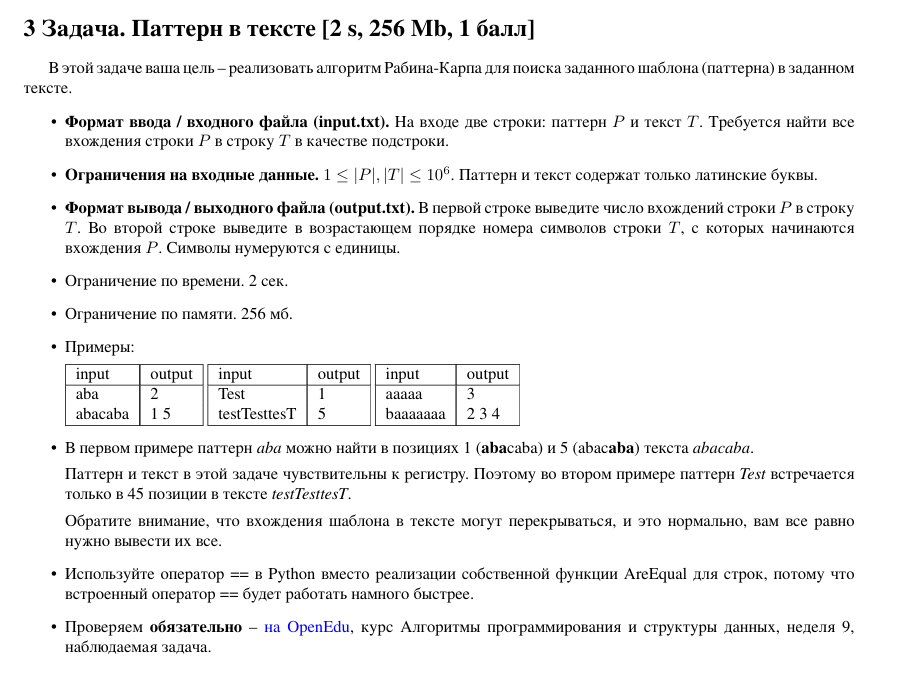
2025 г.

[**Задачи по варианту**](#_e0pij87brwfh)

[Задача №1.](#_y9xrr9qdf7ef) Паттерн в тексте  
[Задача №2.](#_g3d552ii7rgc) Префикс-Функция

[Задача №3.](#_y9xrr9qdf7ef) Шаблон с несовпадением

# Задачи по варианту



**Код программы:**

**def rabin\_karp(pattern, text):**

**p\_len = len(pattern)**

**t\_len = len(text)**

**if p\_len > t\_len:**

**return []**

**base = 256 # основание (кол-во возможных символов)**

**mod = 10\*\*9 + 7 # большое простое число для хэширования**

**# Вычисляем хэш шаблона и первую "окну" текста**

**pattern\_hash = 0**

**current\_hash = 0**

**power = 1 # base^(p\_len-1)**

**for i in range(p\_len):**

**pattern\_hash = (pattern\_hash \* base + ord(pattern[i])) % mod**

**current\_hash = (current\_hash \* base + ord(text[i])) % mod**

**if i != p\_len - 1:**

**power = (power \* base) % mod**

**positions = []**

**for i in range(t\_len - p\_len + 1):**

**if pattern\_hash == current\_hash:**

**if text[i:i + p\_len] == pattern:**

**positions.append(i + 1) # +1, т.к. индексация с единицы**

**if i < t\_len - p\_len:**

**# Удаляем первый символ из окна и добавляем следующий**

**current\_hash = (current\_hash - ord(text[i]) \* power) % mod**

**current\_hash = (current\_hash \* base + ord(text[i + p\_len])) % mod**

**current\_hash = (current\_hash + mod) % mod # избегаем отрицательного значения**

**return positions**

**with open("input3.txt", "r") as file:**

**pattern = file.readline().strip()**

**text = file.readline().strip()**

**# Поиск всех вхождений**

**matches = rabin\_karp(pattern, text)**

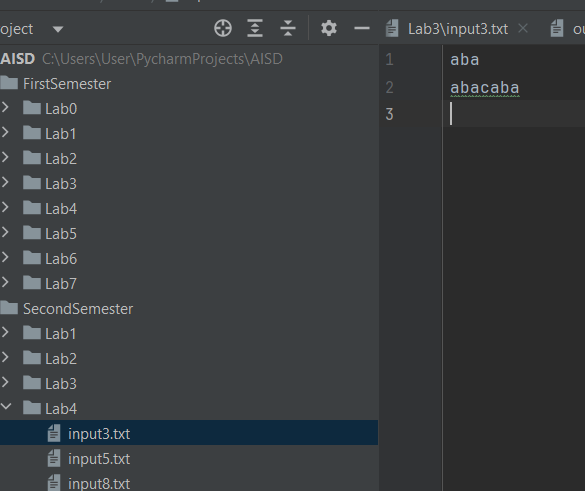
**with open("output3.txt", "w") as file:**

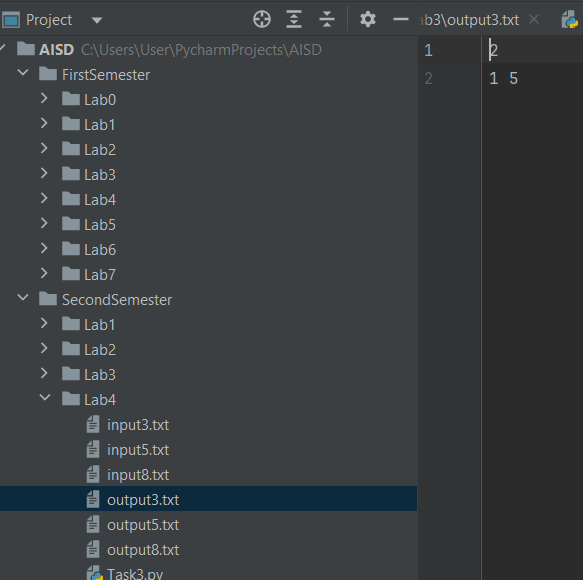
**file.write(f"{len(matches)}\n")**

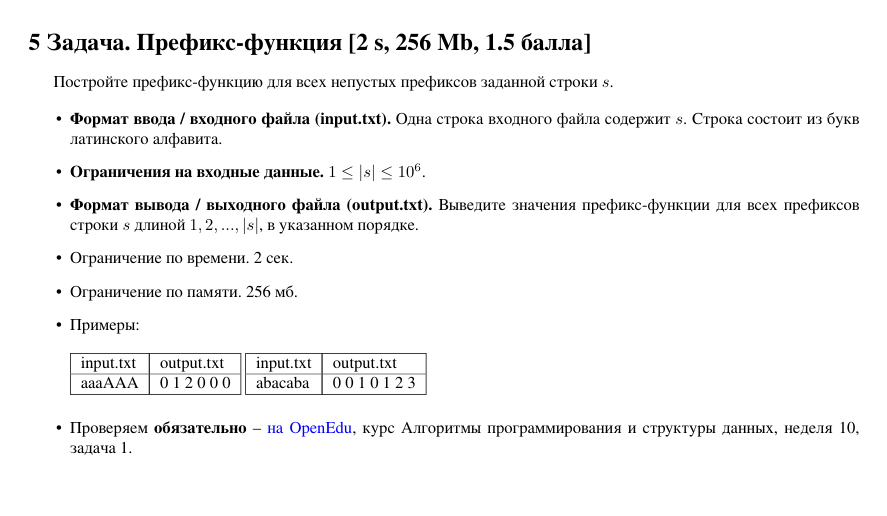
**if matches:**

**file.write(" ".join(map(str, matches)))**

Результат работы кода на примерах:







**Код программы:**

**def compute\_prefix\_function(s):**

**n = len(s)**

**prefix = [0] \* n**

**for i in range(1, n):**

**j = prefix[i - 1]**

**while j > 0 and s[i] != s[j]:**

**j = prefix[j - 1]**

**if s[i] == s[j]:**

**j += 1**

**prefix[i] = j**

**return prefix**

**with open("input5.txt", "r") as file:**

**s = file.readline().strip()**

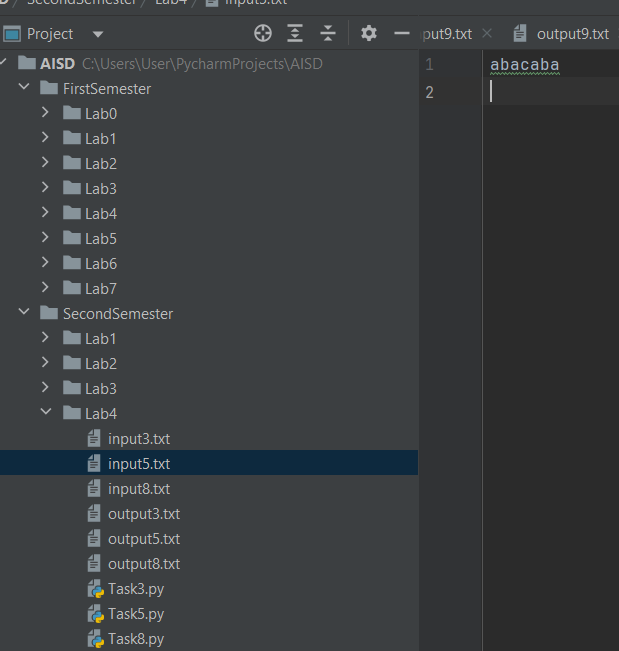
**# Вычисление префикс-функции**

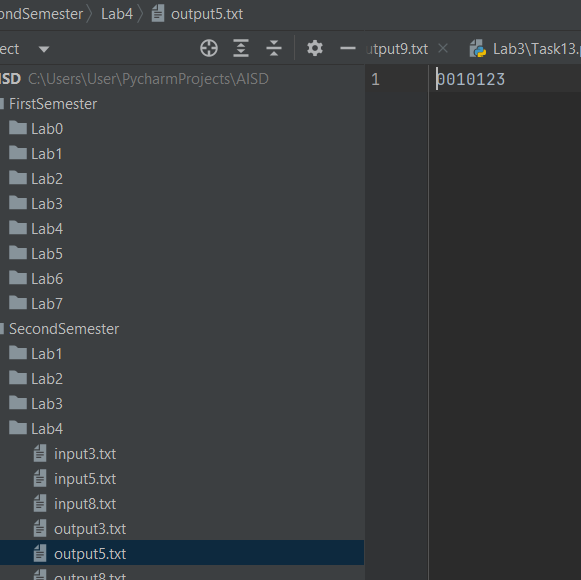
**prefix\_values = compute\_prefix\_function(s)**

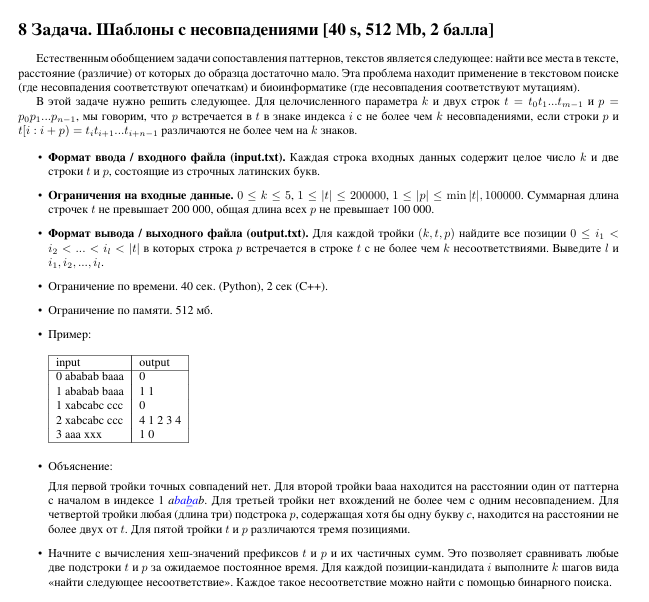
**with open("output5.txt", "w") as file:**

**file.write(''.join(map(str, prefix\_values)))**

Результат работы кода на примерах:

****

****



**Код программы:**

**def find\_approx\_matches(k, t, p):**

**len\_t = len(t)**

**len\_p = len(p)**

**result = []**

**for i in range(len\_t - len\_p + 1):**

**mismatches = 0**

**for j in range(len\_p):**

**if t[i + j] != p[j]:**

**mismatches += 1**

**if mismatches > k:**

**break**

**if mismatches <= k:**

**result.append(i)**

**return result**

**# Чтение входных данных**

**with open("input8.txt", "r") as file:**

**lines = file.readlines()**

**with open("output8.txt", "w") as output:**

**for line in lines:**

**if not line.strip():**

**continue**

**parts = line.strip().split()**

**k = int(parts[0])**

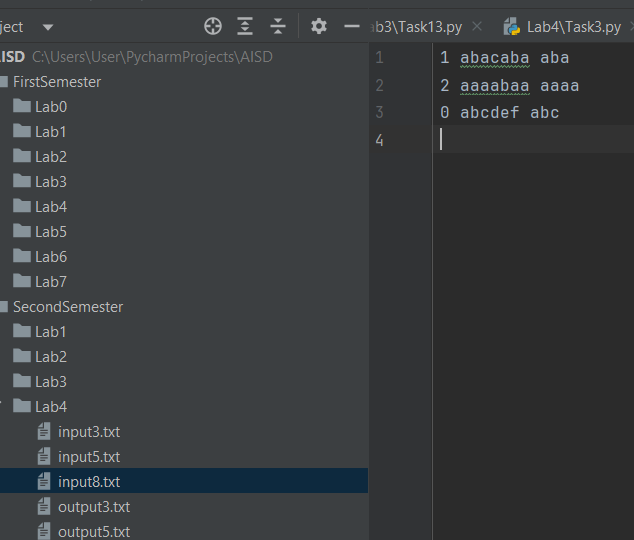
**t = parts[1]**

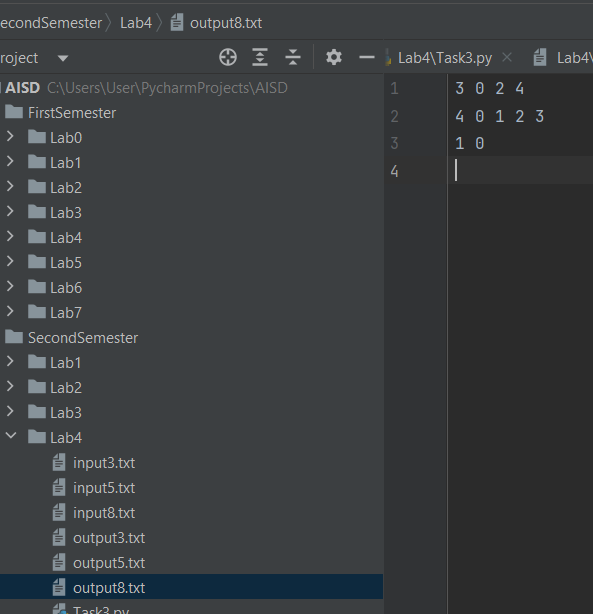
**p = parts[2]**

**matches = find\_approx\_matches(k, t, p)**

**output.write(f"{len(matches)} {' '.join(map(str, matches))}\n")**

Результат работы

****

****