САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных»
Тема: Подстроки
Вариант 16

Выполнила:
Бархатова Н.А.
К3139

Проверила:
Артамонова В.Е.

Санкт-Петербург
2023 г.

Содержание отчета

Задачи по варианту

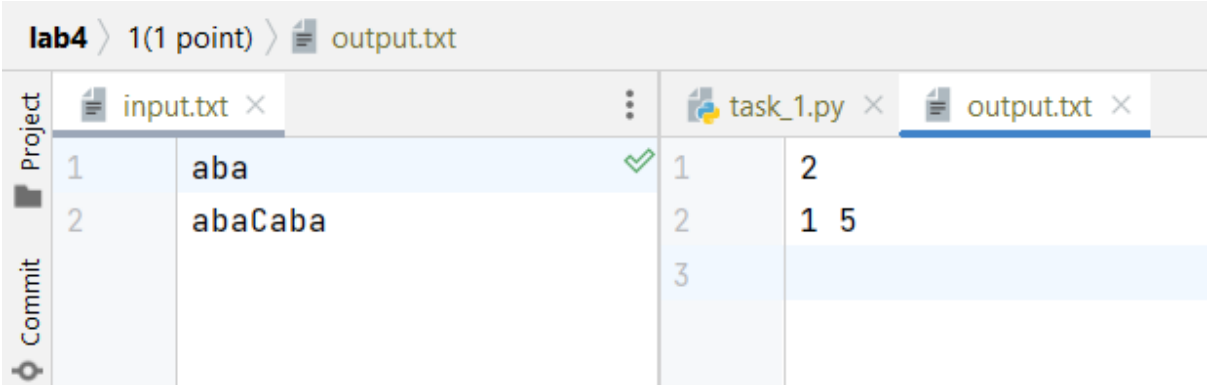
Задача №1. Наивный поиск подстроки в строке [1 балл]

Текст задачи.
Даны строки p и t. Требуется найти все вхождения строки p в строку t в качестве подстроки.

Листинг кода.

```
import time
import tracemalloc

start_time = time.perf_counter()
tracemalloc.start()
input_file = open('input.txt')
p = input_file.readline()[:-1]
t = input_file.readline()
len_p = len(p)
len_t = len(t)
i = 0
count = 0
start_points = []
while i < len_t - len_p + 1:
    if t[i:i + len_p] == p:
        start_points.append(i + 1)
        count += 1
    i += 1
with open('output.txt', 'w') as output_file:
    print(count, file=output_file)
    print(*start_points, file=output_file)
print("Время: ", time.perf_counter() - start_time)
print("Память: ", float(tracemalloc.get_tracemalloc_memory()) / (2 ** 20))
tracemalloc.stop()
```

Текстовое объяснение решения.
Перебор индексов.
Результат работы кода на примерах из текста задачи:(скрины input output файлов)



	Время выполнения	Затраты памяти
Пример из задачи	0.0008	0.0030

Вывод по задаче: легко и просто

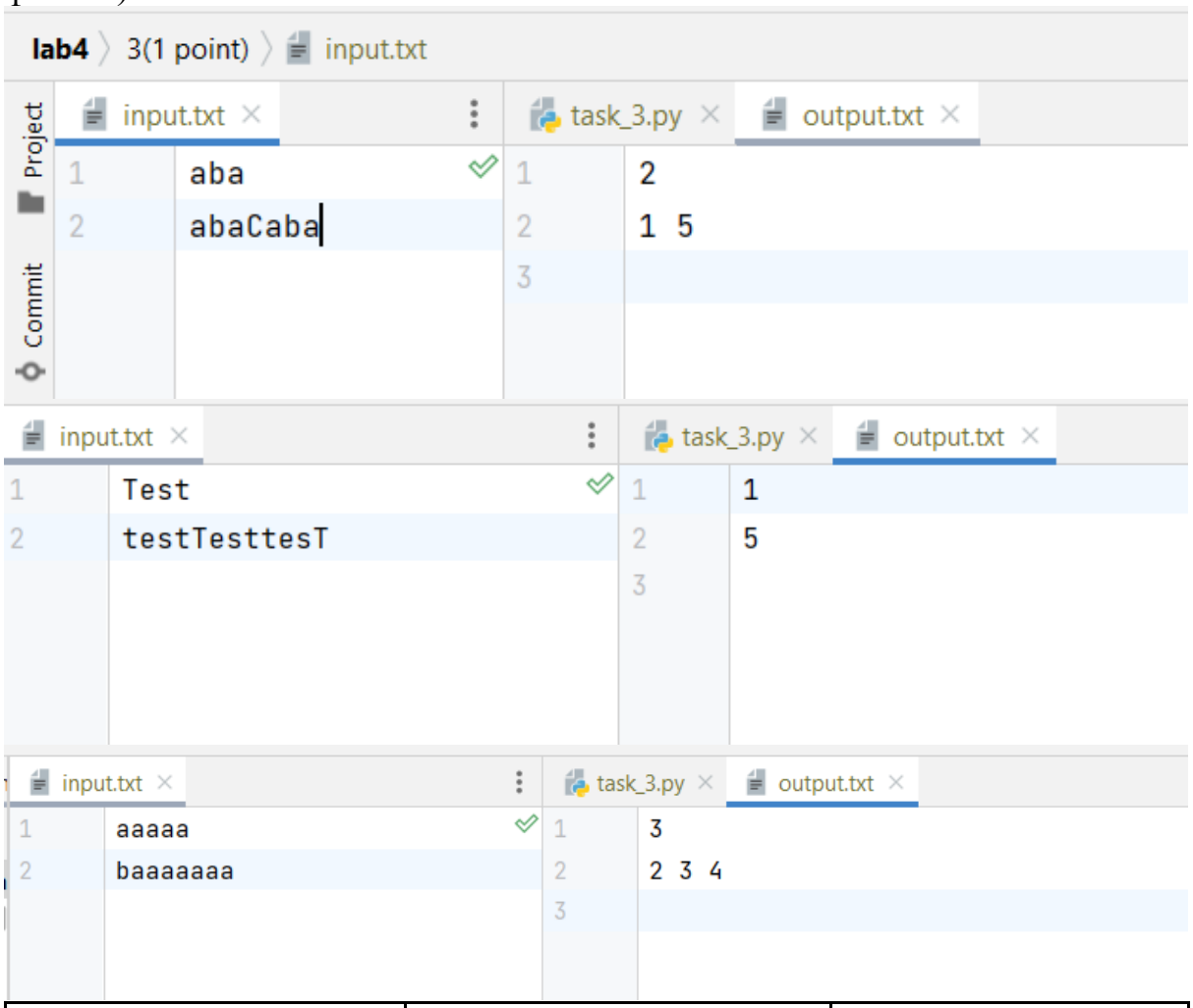
Задача №3. Паттерн в тексте [1 балл]

Текст задачи.
В этой задаче ваша цель – реализовать алгоритм Рабина-Карпа для поиска заданного шаблона (паттерна) в заданном тексте.
Листинг кода.

```
import time
import tracemalloc


def gorner_scheme(text):
    result = ord(text[0])
    base_x = 31
    for i in range(1, len(text)):
        result = result * base_x + ord(text[i])
    return result


def calculate_hash(text):
    q = 2 ** 31 - 1
    return gorner_scheme(text) % q


start_time = time.perf_counter()
tracemalloc.start()
input_file = open('input.txt')
p = input_file.readline()[:-1]
t = input_file.readline()
len_p, len_t = len(p), len(t)
p_hash = calculate_hash(p)
base_x = 31
q = 2 ** 31 - 1
i = 0
count = 0
start_points = []
current_hash = calculate_hash(t[0:len_p])
while True:
    if p_hash == current_hash:
        if p == t[i:i + len_p]:
            count += 1
            start_points.append(i + 1)
    if i >= len_t - len_p:
        break
    current_hash = ((current_hash - ord(t[i]) * base_x ** (len_p - 1)) *
base_x + ord(t[i + len_p])) % q
    i += 1
with open('output.txt', 'w') as output_file:
    print(count, file=output_file)
```

```
    print(*start_points, file=output_file)
print("Время: ", time.perf_counter() - start_time)
print("Память: ", float(tracemalloc.get_tracemalloc_memory()) / (2 ** 20))
tracemalloc.stop()
```

Текстовое объяснение решения.
Создаем хеш-функцию и через неё сравниваем строки.
Результат работы кода на примерах из текста задачи:(скрины input output файлов)



	Время выполнения	Затраты памяти
Пример из задачи	0.0008	0.0035
Пример из задачи	0.0008	0.0035

Вывод по задаче: вспомнила что такое хеш-функция. Весело.

Задача №8. Шаблоны с несовпадениями [2 балла]

Текст задачи.

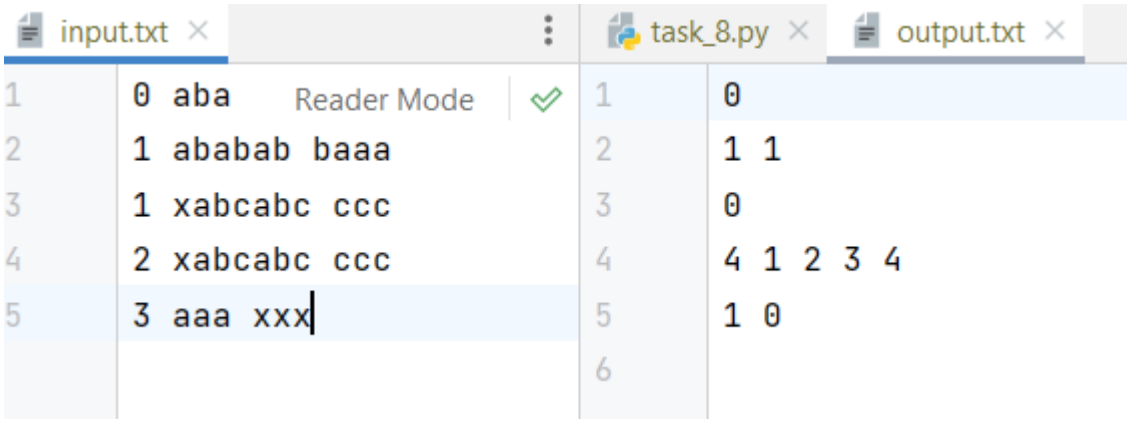
В этой задаче нужно решить следующее. Для целочисленного параметра k и двух строк t = t0t1...tm−1 и p =p0p1...pn−1, мы говорим, что p встречается в t в знаке индекса i с не более чем k несовпадениями, если строки p ит[i : i + p) = titi+1...ti+n−1 различаются не более чем на k знаков.

Листинг кода.

```
with open("input.txt", "r") as input_file:
    results = []
    for line in input_file.readlines():
        line = line.split()
        mismatch_tolerance = int(line[0])
        text = line[1]
        pattern = line[2]
        result = []
        text_index_max = len(text) - len(pattern) + 1
        print(text_index_max)
        for text_index in range(text_index_max):
            missed = 0
            for pattern_index in range(len(pattern)):
                text_char = text[text_index + pattern_index]
                pattern_char = pattern[pattern_index]
                if text_char != pattern_char:
                    missed += 1
                if missed > mismatch_tolerance:
                    break

            if missed == mismatch_tolerance:
                match = text[text_index: text_index + len(pattern)]
                result.append(text_index)
        if len(result) == 0:
            results.append([0])
        else:
            results.append([len(result)] + result)
        with open("output.txt", "w") as output_file:
            for result in results:
                output = ""
                for i in result:
                    output += str(i) + " "
                output_file.write(output + "\n")
```

Текстовое объяснение решения.

Проходимся по всем индексам и проверяем на совпадение каждый символ подстроки, начиная с 0 индекса главной строки. При несовпадении увеличиваем переменную missed. Если missed превышает необходимое количество несовпадений, передвигаем индекс главной строки на +1. Если missed равен необходимому количеству несовпадений, записываем подстроку в ответ.
Результат работы кода на примерах из текста задачи:(скрины input output файлов)



Вывод по задаче: интересная задача.

Дополнительные задачи

Задача №6. Z-функция [1.5 балла]

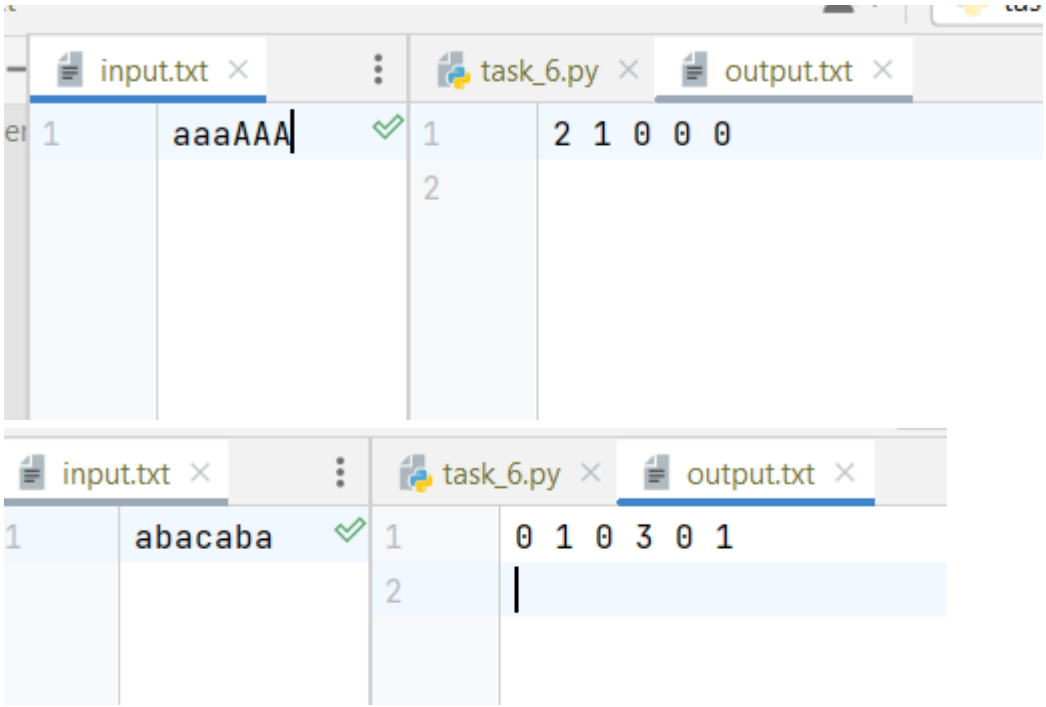
Текст задачи.
Постройте Z-функцию для заданной строки s.

Листинг кода.

```
import time
import tracemalloc

start_time = time.perf_counter()
tracemalloc.start()
input_file = open('input.txt')
s = input_file.readline()


def z_func(s):
    n = len(s)
    z = [0] * n
    l = r = 0
    '''Текущий самый правый отрезок совпадения полагается равным [0:0],
    т.е. заведомо маленький отрезок, в который не попадёт ни одно i'''
    for i in range(1, n):
        if i <= r:
            '''z[i] в [l:r] соответствует значение z[i-l] s [0:r-l].
Однако,
            значение z[i-l] может оказаться большим и при применении к z[i]
в [l:r]
            вылезти за границы r. Этого допустить нельзя, так как про
символее правее r
            мы ничего не знаем. Поэтому ищем минимум.'''
            z[i] = min(z[i - l], r - i + 1)
        while i + z[i] < n and s[i + z[i]] == s[z[i]]:
            z[i] += 1
            '''Тривиальный алгоритм. Также следим за тем, чтобы индексы не
вышли за пределы строки'''
        # if z[i] > r - i:
        if i + z[i] - 1 > r:
            l, r = i, i + z[i] - 1
            '''Мы продлеваем отрезок совпадения до i + z[i] - 1, если
требуется.'''
    return z


with open('output.txt', 'w') as output_file:
    print(*z_func(s)[1:], file=output_file)
print("Время: ", time.perf_counter() - start_time)
print("Память: ", float(tracemalloc.get_tracemalloc_memory()) / (2 ** 20))
tracemalloc.stop()
```

Текстовое объяснение решения.
В комментариях к коду

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



	Время выполнения	Затраты памяти
Пример из задачи	0.0015	0.0029
Пример из задачи	0.0015	0.0029

Вывод по задаче: сложно

Вывод

В ходе выполнения данной лабораторной работы были изучены алгоритмы и структуры данных, которые используются для работы с подстроками в строках. Были рассмотрены следующие алгоритмы поиска подстроки в строке: наивный алгоритм и алгоритм Рабина-Карпа. Каждый из этих алгоритмов имеет свои преимущества и недостатки, и каждый подходит для разных задач и типов входных данных. Таким образом, выполнение данной лабораторной работы позволило расширить знания о том, как работают алгоритмы и структуры данных для работы с подстроками в строках, а также научиться применять их в различных ситуациях.