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: Подстроки

Вариант 14

Выполнила:
Рудникова В.О.
К3143

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Задачи по варианту

Задача №5. Префикс-функция [1.5 балла]

Постройте префикс-функцию для всех непустых префиксов заданной строки s.

• Формат ввода / входного файла (input.txt). Одна строка входного файла содержит s. Строка состоит из букв
латинского алфавита.

• Ограничения на входные данные. $1 \leq |s| \leq 106$.

• Формат вывода / выходного файла (output.txt). Выведите значения префикс-функции для всех префиксов
строки s длиной 1, 2, ..., |s|, в указанном порядке.

• Ограничение по времени. 2 сек.

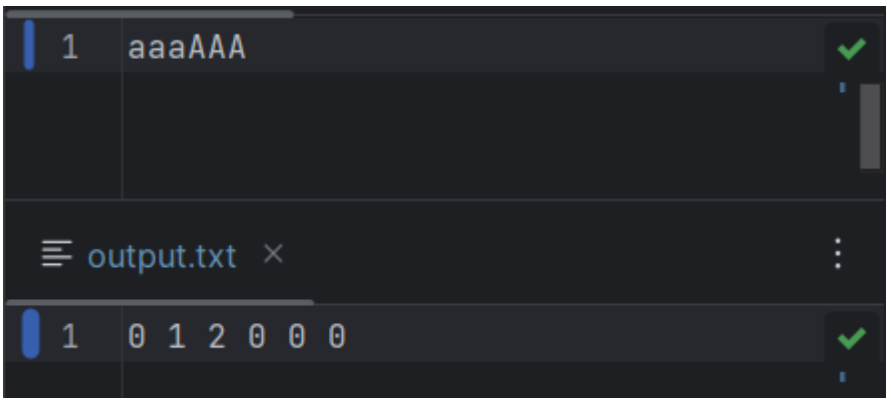
• Ограничение по памяти. 256 мб.

```
from time import process_time
from tracemalloc import start, get_traced_memory


def prefix_func(string):
pi = [0] * len(string)

for i in range(1, len(string)):
j = pi[i - 1]
while j > 0 and string[i] != string[j]:
j = pi[j - 1]
if string[i] == string[j]:
j += 1
pi[i] = j
return pi


if __name__ == '__main__':
start()
with open('../input.txt', 'r') as f:
string = f.readline()
```

```
res = prefix_func(string)
with open('../output.txt', 'w+') as g:
g.write(' '.join([str(i) for i in res]))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: мы перебираем символы строки и для каждого ищем такую длину префикса, чтобы он являлся ещё и суффиксом.

Результат работы кода на примерах из текста задачи:



Вывод по задаче: в задаче я научилась реализовывать префикс-функцию.

Задача №6. Z-функция [1.5 балла]

Постройте Z-функцию для заданной строки s.

• Формат ввода / входного файла (input.txt). Одна строка входного файла содержит s. Строка состоит из букв
латинского алфавита.

• Ограничения на входные данные. $2 \leq |s| \leq 106$.

• Формат вывода / выходного файла (output.txt). Выведите значения Z-функции для всех индексов 1, 2, ..., |s|
строки s, в указанном порядке.

• Ограничение по времени. 2 сек.

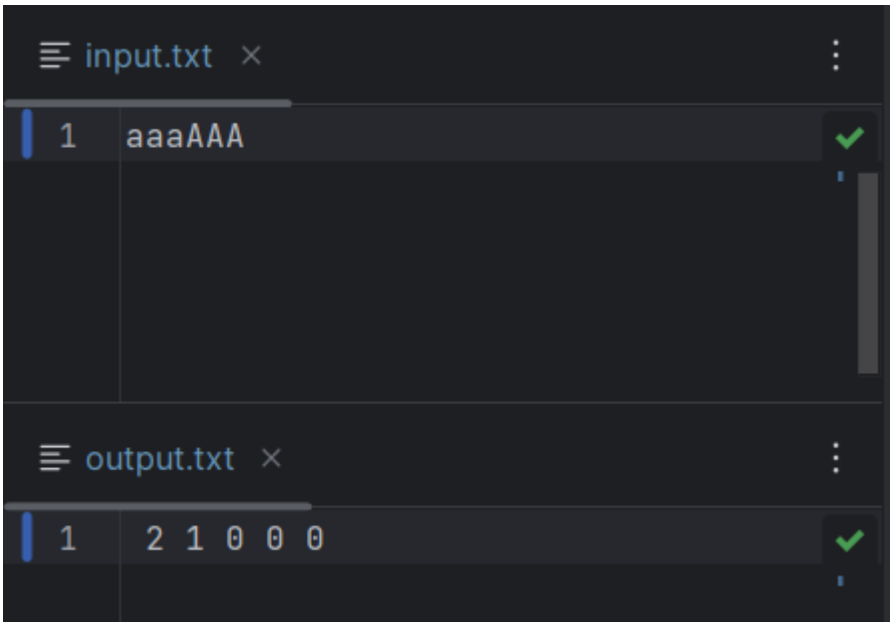
• Ограничение по памяти. 256 мб.

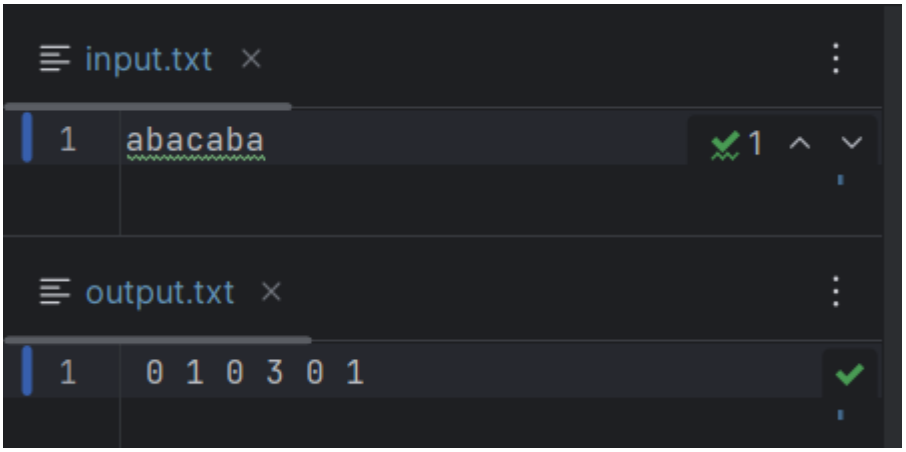
```
from time import process_time
from tracemalloc import start, get_traced_memory


def z_func(string):
N = len(string)
z = [0] * N
left, right = 0, 0

for i in range(1, N):
j = min(z[i - left], right - i + 1) if i <= right
else 0
while i + j < N and string[j] == string[i + j]:
j += 1
z[i] = j
if i + j - 1 > right:
left, right = i, i + j - 1
return z


if __name__ == '__main__':
start()
with open('../input.txt') as f:
string = f.readline()
result = ' '.join([str(i) for i in z_func(string)])
```

```
with open('../output.txt', 'w') as g:
g.write(f"{result[1:]}")

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: для каждого символа ищем длину наибольшего общего префикса суффикса подстроки, состоящей из всей строки, начиная с данного символа, и всей строки.

Результат работы кода на примерах из текста задачи:





Вывод по задаче: в задаче я реализовала Z-функцию.

Задача №8. Шаблоны с несовпадениями [2 балла]

Естественным обобщением задачи сопоставления паттернов, текстов является следующее: найти все места в тексте, расстояние (различие) от которых до образца достаточно мало. Эта проблема находит применение в текстовом поиске (где несовпадения соответствуют опечаткам) и биоинформатике (где несовпадения соответствуют мутациям).
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 и t[i : i + p) = titi+1...ti+n−1 различаются не более чем на k знаков.
• Формат ввода / входного файла (input.txt). Каждая строка входных данных содержит целое число k и две строки t и p, состоящие из строчных латинских букв.
• Ограничения на входные данные. 0 ≤ k ≤ 5, 1 ≤ |t| ≤ 200000, 1 ≤ |p| ≤ min |t|, 100000. Суммарная длина строчек t не превышает 200 000, общая длина всех p не превышает 100 000.
• Формат вывода / выходного файла (output.txt). Для каждой тройки (k, t, p) найдите все позиции 0 ≤ i1 < i2 < ... < il < |t| в которых строка p встречается в строке t с не более чем k несоответствиями. Выведите l и i1, i2, ..., il.

• Ограничение по времени. 40 сек. (Python), 2 сек (C++).
• Ограничение по памяти. 512 мб.

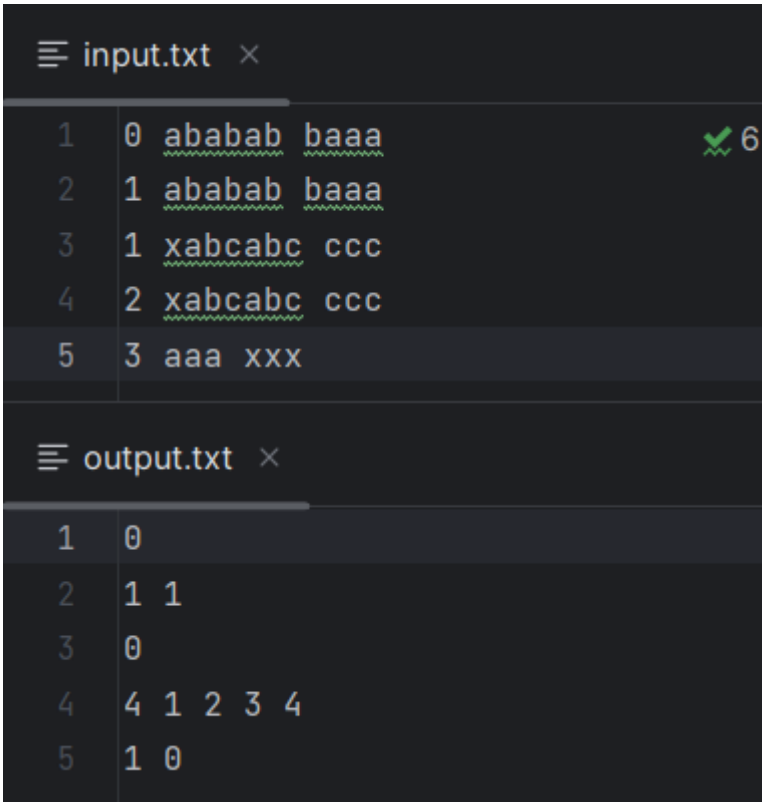
```
from time import process_time
from tracemalloc import start, get_traced_memory


def read_inf_from_file(filename):
input_file = open(filename, "r")
data = []

for line in input_file.readlines():
line = line.strip()
line_data = line.split()
line_data[0] = int(line_data[0])

data.append(line_data)
```

```
return data


def find_mistakes(data):
results = []

for query in data:
mistake_range = query[0]
string = query[1]
substring = query[2]

result = []

for i in range(len(string) - len(substring) + 1):
mistakes = 0

for j in range(len(substring)):
part_of_string = string[i + j]
part_of_substring = substring[j]

if part_of_string != part_of_substring:
mistakes += 1

if mistakes > mistake_range:
break

if mistakes == mistake_range:
result.append(i)

if len(result) == 0:
results.append([0])
else:
results.append([len(result)] + result)

return results
```

```
if __name__ == '__main__':
start()
data = read_inf_from_file("../input.txt")
with open('../output.txt', 'w+') as g:
st = ''
for string in find_mistakes(data):
st += (' '.join([str(i) for i in string]) + '\n')
g.write(st)

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: для каждого запроса программа проверяет все подстроки строки на наличие несовпадений. Если количество превысило разрешённое, то подстрока не удовлетворяет условию, если же нет - она добавляется в результаты.

Результат работы кода на примерах из текста задачи:



Вывод по задаче: в задаче я искала несовпадения в строках.

Дополнительные задачи

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

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

• Формат ввода / входного файла (input.txt). Первая строка входного файла содержит p, вторая – t. Строки
состоят из букв латинского алфавита.

• Ограничения на входные данные. $1 \leq |p|, |t| \leq 104$

.

• Формат вывода / выходного файла (output.txt). В первой строке выведите число вхождений строки p в строку t.
Во второй строке выведите в возрастающем порядке номера символов строки t, с которых начинаются вхождения
p. Символы нумеруются с единицы.

• Ограничение по времени. 2 сек.

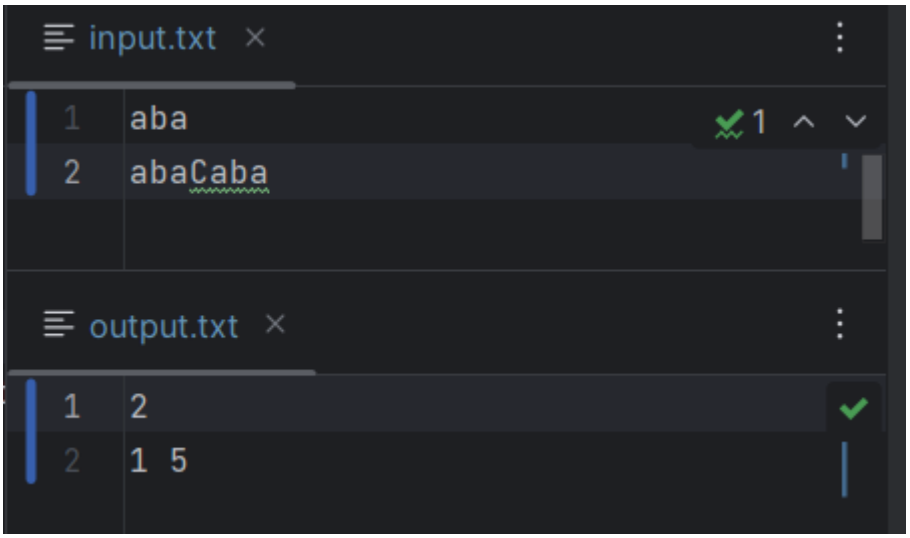
• Ограничение по памяти. 256 мб.

```
from time import process_time
from tracemalloc import start, get_traced_memory


def direct_search(string, substr):
count = 0
pos_arr = []
for i in range(len(string)):
if string[i : i + len(substr)] == substr:
count += 1
pos_arr.append(i + 1)

return count, pos_arr


if __name__ == "__main__":
start()
with open('../input.txt') as f:
```

```
substr, string = f.read().split("\n")

with open('../output.txt', 'w') as g:
nums = ' '.join([str(i) for i in
direct_search(string, substr)[1]])
g.write(f"{direct_search(string,
substr)[0]}\n{nums}")

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: в задаче мы пробегаемся по строке и ищем в ней нужную подстроку.

Результат работы кода на примерах из текста задачи:



Вывод по задаче: я научилась искать подстроку в строке перебором.

Вывод:
В работе я поработала с подстроками и узнала о них много нового, научилась реализовывать префикс-функцию и Z-функцию, искать подстроку в строке и несовпадения.