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 23

Выполнил:
Тарасов А.Н.
К3244

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Лабораторная работа №4.

Задача №2. Карта [2 s, 256 Mb, 1 балл]

В далеком 1744 году во время долгого плавания в руки капитана Александра Смоллетта попала древняя карта с указанием местонахождения сокровищ. Однако расшифровать ее содержание было не так уж и просто. Команда Александра Смоллетта догадалась, что сокровища находятся на x шагов восточнее красного креста, однако определить значение числа она не смогла. По возвращению на материк Александр Смоллетт решил обратиться за помощью в расшифровке послания к знакомому мудрецу. Мудрец поведал, что данное послание таит за собой некоторое число. Для вычисления этого числа необходимо было удалить все пробелы между словами, а потом посчитать количество способов вычеркнуть все буквы кроме трех так, чтобы полученное слово из трех букв одинаково читалось слева направо и справа налево. Александр Смоллетт догадывался, что число, зашифрованное в послании, и есть число x. Однако, вычислить это число у него не получилось. После смерти капитана карта была безнадежно утеряна до тех пор, пока не оказалась в ваших руках. Вы уже знаете все секреты, осталось только вычислить число x.

```
import time

nach = time.time()

def tresure_map(s):
    s = ''.join(s.split())
    prev_char = {}
    my_l = [[0, 0] for _ in range(len(s))]

    for ind, char in enumerate(s):
        prev_i = prev_char.get(char, None)
        if prev_i is not None:
            my_l[ind][0] = my_l[prev_i][0] + 1
            my_l[ind][1] = my_l[prev_i][1] + my_l[prev_i][0] * (ind -
prev_i) + (ind - prev_i - 1)
        else:
            my_l[ind] = [0, 0]
        prev_char[char] = ind

    total = sum(i[1] for i in my_l)
    return total


def main():
    with open("input.txt", "r", encoding='utf-8') as input_file,
open("output.txt", "w",
```

```
encoding='utf-8') as output_file:
        s = input_file.read()
        result = tresure_map(s)
        output_file.write(str(result))


if __name__ == '__main__':
    main()

kon = time.time()
c = kon - nach
print('Время :', c)

import os, psutil; print(psutil.Process(os.getpid()).memory_info().rss /
1024 ** 2)
```

При чтении данных из файла сразу удалим пробелы. Создадим хэш-таблицу(словарь) prev_char, его будем исп-ть для хранения последних индексов символов. Ключ- символы строки, значение — их последний встреченный индекс. Перебираем каждый символ char и его индекс ind в строке s . В цикле для каждого символа, если он уже в словаре - получаем его индекс.После обработки текущего символа обновляем хэш-таблицу, записывая текущий индекс для символа char. Обновляем таблицу my_l[ind][0] чтобы учесть новые палиндромные тройки заканчивающиеся на новом индексе.Если prev_i =None, символ char встречается впервые. В этом случае инициализируем my_l[ind] как [0, 0].

Задача №5. Префикс-функция [2 s, 256 Mb, 1.5 балла]

Постройте префикс-функцию для всех непустых префиксов заданной строки s

```
import time

nach = time.time()

def prefix_function(s):
    j = 0
    prefix = [0] * len(s)
    for i in range(1, len(s)):
        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
```

```
def main():
    with open("input.txt", "r", encoding='utf-8') as input_file,
open("output.txt", "w",

encoding='utf-8') as output_file:
        s = input_file.readline().strip()
        output_file.write(' '.join(map(str, prefix_function(s))))


if __name__ == '__main__':
    main()

kon = time.time()
c = kon - nach
print('Время :', c)

import os, psutil; print(psutil.Process(os.getpid()).memory_info().rss /
1024 ** 2)
```

Создаем список значений префикс-функции изначально все элементы равны 0. Проходим по строке по индексам i, начиная с первого. prefix[i] хранит значение префикс-функции для подстроки s[0:i+1]. индексы: i для текущего символа и j для отслеживания текущей длины совпадения. Если символы совпадают, обновляем j и pi[i] иначе уменьшаем j, пока не найдём совпадение или пока j != 0

Задача №9. Декомпозиция строки [2 s, 256 Mb, 2 балла]

Строка ABCABCDEDEDEF содержит подстроку ABC , повторяющуюся два раза подряд, и подстроку DE , повторяющуюся три раза подряд. Таким образом, ее можно записать как ABC*2+DE*3+F , что занимает меньше места, чем исходная запись той же строки. Ваша задача – построить наиболее экономное представление данной строки s в виде, продемонстрированном выше, а именно, подобрать такие s1, a1, ..., sk, ak, где si - строки, a ai - числа, чтобы s = s1 · a1 + ... + sk · ak. Под операцией умножения строки на целое положительное число подразумевается конкатенация одной или нескольких копий строки, число которых равно числовому множителю, то есть, ABC*2=ABCABC . При этом требуется минимизировать общую длину итогового описания, в котором компоненты разделяются знаком + , а умножение строки на число записывается как умножаемая строка и множитель, разделенные знаком * . Если же множитель равен единице, его, вместе со знаком * , допускается не указывать.

```
import time

nach = time.time()

def encode_string(s):
    n = len(s)
    dp = [["" for _ in range(n)] for _ in range(n)]

    for l in range(n):
        for i in range(n - l):
            j = i + l
            substr = s[i:j + 1]
            dp[i][j] = substr

            if j - i < 4:
                continue

            for k in range(i, j):
                if len(dp[i][j]) > len(dp[i][k] + '+' + dp[k + 1][j]):
                    dp[i][j] = dp[i][k] + '+' + dp[k + 1][j]

            for k in range(1, len(substr)):
                repeat_str = substr[:k]
                if repeat_str and len(substr) % len(repeat_str) == 0 and
substr == repeat_str * (len(substr) // len(repeat_str)):
                    encoded = f"{dp[i][i + k - 1]}*{len(substr) //
len(repeat_str)}"
                    if len(encoded) < len(dp[i][j]):
                        dp[i][j] = encoded

    return dp[0][n - 1]

def main():
    with open('input.txt', 'r') as file:
        s = file.readline().strip()

    result = encode_string(s)

    with open('output.txt', 'w') as file:
        file.write(result + '\n')

if __name__ == "__main__":
    main()

kon = time.time()
c = kon - nach
print('Время :', c)

import os, psutil; print(psutil.Process(os.getpid()).memory_info().rss /
1024 ** 2)
```

Дополнительные задачи

Задача №8. Шаблоны с несовпадениями [40 s, 512 Mb, 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.

```
import time

nach = time.time()
def k_mismatch_search(p, t, k):
    matches = []

    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:
            matches.append(i)
    ans = f'{len(matches)} {" ".join(map(str, matches))}\n'
    return ans


def main():
    with open("input.txt", "r", encoding='utf-8') as input_file,
open("output.txt", "w",

encoding='utf-8') as output_file:
        for line in input_file:
            k, t, p = line.strip().split()
            result = k_mismatch_search(p, t, int(k))
            output_file.write(result)


if __name__ == '__main__':
    main()

kon = time.time()
c = kon - nach
print('Время :', c)

import os, psutil; print(psutil.Process(os.getpid()).memory_info().rss /
1024 ** 2)
```

Проходимся по всем позициям главной строки с которых может начинаться подстрока. Инициализируем счетчик несовпадений = 0. Перебираем все символы в паттерне. Сравниваем символы строки и паттерна, если они не

совпадают увеличиваем счетчик несовпадений. Если этот счетчик превышает заданное k прерываем цикл, если не превышает добавляем позицию в результирующий список позиций.

Вывод

В ходе работы над лабораторной я узнал о суффиксах и префиксах строки.