Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
(Университет ИТМО)

Факультет **Инфокоммуникационных технологий**

Образовательная программа **Интеллектуальные системы в гуманитарной сфере**

Направление подготовки **45.03.04 Интеллектуальные системы в гуманитарной сфере**

О Т Ч Е Т

лабораторной работе 4

на тему: “Подстроки”

Обучающийся Королева Екатерина
К3143

Работа выполнена с оценкой ____

Преподаватель:

_____
(подпись)

Дата 22.06.2022

Санкт-Петербург, 2021

Задачи из варианта 7:

1 Задача. Наивный поиск подстроки в строке [2 s, 256 Mb, 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 ≤ |p|, |t| ≤ 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.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


def areEqual(s1, s2):
    if len(s1) != len(s2):
        return False
    for i in range(len(s1)):
        if s1[i] != s2[i]:
            return False
    return True


def FindPattern(t, p):
    result = []
    for i in range(len(t) - len(p) + 1):
        if areEqual(t[i:i + len(p)], p):
            result.append(str(i + 1))
    return result


if __name__ == '__main__':
    file = open('input.txt')
    p = file.readline()[:-1]
    t = file.readline()
```

```
    file = open('output.txt', 'w')
    result = FindPattern(t, p)
    file.write(f'{len(result)}\n')
    file.write(' '.join(result))


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())

'''
Считываем подстроку p и ищем все ее вхождения в строку t
наивным алгоритмом. Функция FindPattern проходится по всем
подстрокам длины p основной строки, для каждой из них
запускает фунцию areEqual, которая сравнивает  каждый символ
двух строк. Если все символы совпадают, возвращает True и
записывает индекс начала подстроки в основной строке.
'''
Время работы (в секундах): 0.0006875000000000006
Память 10577, и пик 19520

Тесты на Openedu пока не работают (задачи 1-3, 5, 6)
```

4 Задача. Равенство подстрок [10 s, 512 Mb, 1.5 балла]

В этой задаче вы будете использовать хеширование для разработки алгоритма, способного предварительно обработать заданную строку s, чтобы ответить эффективно на любой запрос типа «равны ли эти две подстроки s?» Это, в свою очередь, является основной частью во многих алгоритмах обработки строк.

• Формат ввода / входного файла (input.txt). Первая строка содержит строку s, состоящую из строчных латинских букв. Вторая строка содержит количество запросов q. Каждая из следующих q строк задает запрос тремя целыми числами a, b и l.

• Ограничения на входные данные. $1 \le |s| \le 500000$, $1 \le q \le 100000$, $0 \le a, b \le |s|-l$ (следовательно, индексы a и b начинаются с 0).

• Формат вывода / выходного файла (output.txt).Для каждого запроса выведите «Yes», если подстроки $s_a s_{a+1} \ldots s_{a+l-1} = s_b s_{b+1} \ldots s_{b+l-1}$ равны, и «No» – если не равны.

• Ограничение по времени. 10 сек.

• Ограничение по памяти. 512 мб.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


def poly_hash(P, p, x=128):
    h = 0
    for i in reversed(range(len(P))):
        h = (h * x + ord(P[i])) % p
    return h % p


def precompute_hashes(T, P, p, x):
    H = [0] * (len(T) - len(P) + 1)
    S = T[len(T) - len(P) : len(T)]
    ind = len(T) - len(P)
    H[ind] = poly_hash(S, p, x)
    y = 1
    for i in range(1, len(P) + 1):
        y = (y * x) % p
    for i in range(len(T) - len(P) - 1, -1, -1):
        H[i] = (x * H[i + 1] + ord(T[i]) - y * ord(T[i +
len(P)]) + p) % p
```

```
    return H


if __name__ == '__main__':

    with open("input.txt") as file:
        word = file.readline()[:-1]
        n = int(file.readline())
        sp = []
        for _ in range(n):
            sp.append(list(map(int, file.readline().split())))

    hashes = dict()
    coll = dict()
    k = 1
    l = len(word)
    p, o = 10**9 + 7, 10**9 + 9

    while k <= l:
        hashes[k] = []
        coll[k] = []
        for i in range(l-k+1):
            hashes[k].append(poly_hash(word[i:i+k], p))
            coll[k].append(poly_hash(word[i:i+k], o))
        k += 1

    with open('output.txt', 'w') as file:
        for i in range(n):
            count = sp[i][-1]
            first = sp[i][0]
            second = sp[i][1]
            if hashes[count][first] == hashes[count][second]
and coll[count][first] == coll[count][second]:
                file.write(f'YES\n')
            else:
                file.write(f'NO\n')


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())
```

```
'''
Считываем строку и записываем хэши каждой ее подстроки
в словарь с количеством символов этой подстроки. Создаем
второй словарь с таким же условием, чтобы избежать коллизии.
Затем проходим по командам, считываем индекс первого элемента
первой подстроки и индекс первого элемента второй подстроки, и
количество элементов этих подстрок. Затем в словарях
сравниваем значения индексов первых элементов в ключе
количества символов. Если значения совпадают в обоих словарях,
то выводим YES, в противном случае NO.
'''
```

Время работы (в секундах): 0.000877799999999998
Память 5680, и пик 18567

6 Задача. Z-функция [2 s, 256 Mb, 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 ≤ |s| ≤ 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.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


def z_arr(s):

    Z = [0] * len(s)
    rt, lt = 0, 0
    for k in range(1, len(s)):
        if k > rt:
            n = 0
            while n + k < len(s) and s[n] == s[n+k]:
                n += 1
            Z[k] = n
            if n > 0:
                lt = k
                rt = k+n-1
        else:
            p = k - lt
            right_part_len = rt - k + 1
            if Z[p] < right_part_len:
                Z[k] = Z[p]
            else:
                i = rt + 1
                while i < len(s) and s[i] == s[i - k]:
                    i += 1
                Z[k] = i - k
                lt = k
                rt = i - 1
    return Z
```

```
if __name__ == '__main__':
    with open('input.txt') as file:
        word = file.readline()

    result = list(map(str, z_arr(word)))

    with open('output.txt', 'w') as file:
        file.write(f'{" ".join(result[1:-1])}')


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())

'''
Суть Z-функции состоит в нахождении количества одинаковых
символов всех суффиксов строки и самой строки. То есть мы
берем каждый суффикс, начиная с самого большого, и подставляем
его в начало строки, сравнивая каждый символ
'''

Время работы (в секундах): 0.0005539999999999989
Память 1691, и пик 17920
```

Дополнительные задачи:

2 Задача. 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.

• Формат ввода / входного файла (input.txt). В единственной строке входного файла дано послание, написанное на карте.

• Ограничения на входные данные. Длина послания не превышает 3 · 105. Гарантируется, что послание может содержать только строчные буквы английского алфавита и пробелы. Также гарантируется, что послание не пусто. Послание не может начинаться с пробела или заканчиваться им.

• Формат вывода / выходного файла (output.txt). Выведите одно число x – число способов вычеркнуть из послания все буквы кроме трех так, чтобы оставшееся слово одинаково читалось слева направо и справа налево.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


if __name__ == '__main__':
   with open('input.txt') as file:
       word = file.readline().split(' ')
```

```
        word = ''.join(word)
    alph, q = dict(), dict()

    for i in range(len(word)):
        letter = word[i]
        try:
            alph[letter] += 1
        except:
            alph[letter] = 0
            alph[letter] += 1

    c = 0
    for key in alph:
        copy = word
        if alph[key] >= 2:
            c += 1
            q[key] = []
            for i in range(len(word)):
                if word[i] == key:
                    q[key].append(i)
    result = 0

    for letter in q:
        coef = len(q[letter]) - 1
        for i in reversed(range(len(q[letter]))):
            result += q[letter][i] * coef
            coef -= 2
        result -= (len(q[letter]) - 1) * len(q[letter]) // 2

    with open('output.txt', 'w') as file:
        file.write(str(result))


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())

'''
Считываем строку и проходимся по ней, записывая в  словарь
alph количество символов в строке, а в словарь q – индексы
этих символов, если их больше 1. Затем считываем количество
возможных вариантов, считая количество символов между двумя
повторяющимися для каждого символа.
'''
```

Время работы (в секундах): 0.0006173999999999971
Память 2905, и пик 17788

3 Задача. Паттерн в тексте [2 s, 256 Mb, 1 балл]

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

• Формат ввода / входного файла (input.txt). На входе две строки: паттерн P и текст T. Требуется найти все вхождения строки P в строку T в качестве подстроки.

• Ограничения на входные данные. $1 \leq |P|, |T| \leq 106$. Паттерн и текст содержат только латинские буквы.

• 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.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


def pref(S):
    l = len(S)
    P = [0] * l
    i, j = 0, 1
    while j < l:
        if S[i] == S[j]:
            P[j] = i + 1
            i += 1
            j += 1
        elif i:
            i = P[i - 1]
        else:
            P[j] = 0
            j += 1
    return P


def kmp(text, sub):
    sub_len, text_len = len(sub), len(text)
    if not text_len or sub_len > text_len:
        return []
```

```
    P = pref(sub)
    entries = []
    i = j = 0
    while i < text_len and j < sub_len:
        if text[i] == sub[j]:
            if j == sub_len - 1:
                entries.append(str(i - sub_len + 2))
                j = P[j]
            else:
                j += 1
            i += 1
        elif j:
            j = P[j - 1]
        else:
            i += 1
    return entries


if __name__ == '__main__':
    file = open('input.txt')
    sub = file.readline()[:-1]
    s = file.readline()
    P = kmp(s, sub)
    file = open('output.txt', 'w')
    file.write(f'{len(P)}\n')
    file.write(f'{" ".join(P)}')


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())

'''
Считываем строку и подстроку и запускаем функцию
Кнутта-Морриса-Пратта. С помощью префикс функции мы считаем
для каждого символа образа количество совпадений, то есть
количество символов, на которые мы можем сдвинуть образ вдоль
строки. Количество этих символов считается как количество
префиксов и суффиксов подстроки одинаковой длины и равных по
значению. То есть префикс-функция для i-ого символа образа
возвращает значение, равное максимальной длине совпадающих
префикса и суффикса подстроки в образе, которая заканчивается
i-м символом. На выходе мы получаем массив образа.
```

Затем проходим АКМП по строке. Индексы i и j указывают на начальные символы строки и образа. В случае совпадения символов мы сдвигаем оба индекса вправо на 1. В случае несовпадения мы обращаем внимание на символ в образе, предшествующий не совпавшему. Его значение в массиве образа является индексом того символа, который нужно сдвинуть на j-ый индекс. В случае совпадения записываем индекс начала.
'''

Время работы (в секундах): 0.0006876999999999994
Память 10577, и пик 19652

5 Задача. Префикс-функция [2 s, 256 Mb, 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.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


def pref(S):
    l = len(S)
    P = [0]*l
    i, j = 0, 1
    while j < l:
        if S[i] == S[j]:
            P[j] = i + 1
            i += 1
            j += 1
        elif i:
            i = P[i - 1]
        else:
            P[j] = 0
            j += 1
    return P


if __name__ == '__main__':
    file = open('input.txt')
    word = file.readline()
    result = list(map(str, pref(word)))
    file = open('output.txt', 'w')
    file.write(f'{" ".join(result[:-1])}')
```

```
print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())

'''
Считываем строку и запускаем префикс-функцию (см. задание 3)
Задание 3: Считываем строку и подстроку и запускаем функцию
Кнутта-Морриса-Пратта. С помощью префикс функции мы считаем
для каждого символа образа количество совпадений, то есть
количество символов, на которые мы можем сдвинуть образ вдоль
строки. Количество этих символов считается как количество
префиксов и суффиксов подстроки одинаковой длины и равных по
значению. То есть префикс-функция для i-ого символа образа
возвращает значение, равное максимальной длине совпадающих
префикса и суффикса подстроки в образе, которая заканчивается
i-м символом. На выходе мы получаем массив образа. Затем
проходим АКМП по строке. Индексы i и j указывают на начальные
символы строки и образа. В случае совпадения символов мы
сдвигаем оба индекса вправо на 1. В случае несовпадения мы
обращаем внимание на символ в образе, предшествующий не
совпавшему. Его значение в массиве образа является индексом
того символа, который нужно сдвинуть на j-ый индекс. В случае
совпадения записываем индекс начала.
'''

Время работы (в секундах): 0.0007441000000000045
Память 10630, и пик 19751
```

7 Задача. Наибольшая общая подстрока [15 s, 512 Mb, 2 балла]

В задаче на наибольшую общую подстроку даются две строки s и t, и цель состоит в том, чтобы найти строку w максимальной длины, которая является подстрокой как s, так и t. Это естественная мера сходства между двумя строками. Задача имеет применения для сравнения и сжатия текстов, а также в биоинформатике. Эту проблему можно рассматривать как частный случай проблемы расстояния редактирования (Левенштейна), где разрешены только вставки и удаления. Следовательно, ее можно решить за время O(|s||t|) с помощью динамического программирования. Есть также весьма нетривиальные структуры данных для решения этой задачи за линейное время O(|s| + |t|). В этой задаче ваша цель – использовать хеширование для решения почти за линейное время.

• Формат ввода / входного файла (input.txt).Каждая строка входных данных содержит две строки s и t, состоящие из строчных латинских букв.

• Ограничения на входные данные. Суммарная длина всех s, а также суммарная длина всех s не превышает 100000.

• Формат вывода / выходного файла (output.txt). Для каждой пары строк si и ti найдите ее самую длинную общую подстроку и уточните ее параметры, выведя три целых числа: ее начальную позицию в s, ее начальную позицию в t (обе считаются с 0) и ее длину. Формально выведите целые числа $0 \le i < |s|$, $0 \le j < |t|$ и $l \ge 0$ такие, что и l максимально. (Как обычно, если таких троек с максимальным l много, выведите любую из них.)

• Ограничение по времени. 15 сек.

• Ограничение по памяти. 512 мб.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


def poly_hash(P, p, x=128):
    h = 0
    for i in reversed(range(len(P))):
        h = (h * x + ord(P[i])) % p
    return h % p


if __name__ == '__main__':
    p, o = 10 ** 9 + 7, 10 ** 9 + 9
```

```
    with open('input.txt') as file:
        text = list(map(str, file.read().split('\n')))
        file = open('output.txt', 'w')
        for i in range(len(text)):
            word1, word2 = text[i].split(' ')
            hashes, coll = dict(), dict()
            k = 1
            w1, w2 = len(word1), len(word2)

            while k <= w1:
                hashes[k] = []
                coll[k] = []
                for i in range(w1 - k + 1):
                    hashes[k].append(poly_hash(word1[i:i + k],
p))
                    coll[k].append(poly_hash(word1[i:i + k],
o))
                k += 1

            k = 1
            maxim = 0
            result, indexes = [], [0, 0]
            while k <= w1:
                for i in range(w2 - k + 1):
                    x = poly_hash(word2[i:i+k], p)
                    y = poly_hash(word2[i:i+k], o)
                    if x in hashes[k] and y in coll[k]:
                        if k > maxim:
                            maxim = k
                            indexes[1], indexes[0] = i,
hashes[k].index(x)
                            k += 1
                            break
                k += 1

            result = list(map(str, (*indexes, maxim)))
            file.write(f'{" ".join(result)}\n')


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())
```

```
'''
Считываем строки, первое слово разделяем на подстроки и
хэшируем каждую. Второе слово начинаем разделять на подстроки
начиная с 1, хэшируем и проверяем, есть ли такой хэш в словаре
с количеством символов. Если такой хэш есть, то переменную
максим обновляем и увеличиваем количество символов в подстроке
второго слова. На выходе пишем максимальную длину совпавшей
подстроки.
'''
```

```
Время работы (в секундах): 0.0007947999999999983
Память 13566, и пик 22716
```

Поиск подстроки

(Время: 0,2 сек. Память: 16 Мб Сложность: 38%)

Найти все вхождения строки T в строке S.

Входные данные

В первой строке входного файла INPUT.TXT записана строка S, во второй строке записана строка T. Обе строки состоят только из английских букв. Длины строк могут быть в диапазоне от 1 до 50 000 включительно.

Выходные данные

В выходной файл OUTPUT.TXT нужно вывести все вхождения строки T в строку S в порядке возрастания. Нумерация позиций строк начинается с нуля.

```
import time
t_start = time.perf_counter()
tracemalloc.start()


with open('input.txt') as f:
   t = f.readline()[:-1]
   p = f.readline()[:-1]

res = []
for i in range(len(t) - len(p) + 1):
   if t[i] == p[0]:
       part = t[i:i+len(p)]
       if part == p:
           res.append(str(i))

print(' '.join(res))


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())
```

'''

Для решения данной задачи достаточно обычного наивного поиска подстроки.

'''

```
Время работы (в секундах): 0.00018399999999999667
Память 1256, и пик 17849
```

Результат теста на acmp:

Автор	Задача	Язык	Результат
Екатерина Максимовна Королева	0202	Python	Accepted

Подстроки из одинаковых букв

(Время: 1 сек. Память: 16 Мб Сложность: 32%)

В заданной строке, состоящей из малых английских букв, необходимо найти пару самых длинных подстрок, состоящих из одних и тех же букв (возможно, в разном порядке). Например, в строке twotwow это будут подстроки wotwo и otwow.

Входные данные

Входной файл INPUT.TXT содержит исходную строку, длиной от 1 до 100 символов.

Выходные данные

Выходной файл OUTPUT.TXT должен содержать единственное число – длину подстрок в максимальной паре, или 0, если таких подстрок в строке нет.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


with open('input.txt') as f:
    s = f.readline()

l = len(s)
flag = False
for k in range(l-1, 0, -1):
    for i in range(l-k):
        if s[i] == s[i+k]:
            print(k)
            flag = True
            break
    if flag:
      break
if not flag:
    print(0)


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" % tracemalloc.get_traced_memory())
```

```
'''
Чтобы найти две подстроки максимальной длины, состоящей из
одних и тех же символов, достаточно найти самую длинную
подстроку, которая начинается и оканчивается одним и тем же
символом. Тогда максимальной длинной таких подстрок будет
длина найденной подстроки без одного символа. В программе мы
идем счетчиком k от l-1 до 1 – максимизируем длину возможных
подстрок. Далее счетчиком i сравниваем i-й и (i+k)-й символы.
Если они одинаковые, то выводим k. Если в течение циклов
строки с двумя одинаковыми символами на концах не нашлось, то
выводим 0.
'''
```

Время работы (в секундах): 0.00016449999999999798
Память 917, и пик 17782

Результат теста на acmp:

Екатерина Максимовна Королева	0361	Python	Accepted

Вывод: В ходе данной лабораторной работы были решены различные задачи по работе с подстроками. Были изучены и применены на практике наивный поиск подстрок, Z-функция и алгоритм Кнута-Морриса-Пратта.