МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по практической работе №4 по

дисциплине «Построение и анализ алгоритмов»

Тема: «Алгоритм Кнута-Морриса-Пратта»

Студент гр. 0304 **Мажуга Д.Р.**

Преподаватель **Фирсов М.А.**

Санкт-Петербург,

2022

Цель работы.

Изучение алгоритма Кнута-Морриса-Пратта и применение алгоритма в задачах со строками.

Задание.

Часть 1.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P($|P| \leq 15000$) и текста T ($|T| \leq 5000000$) найдите все вхождения P в T.

Вход:

Первая строка – P

Вторая строка – T

Выход:

Индексы начал вхождений P в T, разделенных запятой, если P не входит в T, то вывести −1

Sample Input:

ab

abab

Sample Output:

0,2

Часть 2.

Заданы две строки A ($|A| \leq 5000000$) и B ($|B| \leq 5000000$). Определить, является ли A циклическим сдвигом B (это значит, что A и B имеют одинаковую длину и A состоит из суффикса B, склеенного с префиксом B). Например, defabc является циклическим сдвигом abcdef.

Вход:

Первая строка - *A*

Вторая строка - *B*

Выход:

Если A вляется циклическим сдвигом B, индекс начала строки B в A, иначе вывести −1. Если возможно несколько сдвигов вывести первый индекс.

Sample Input:

defabc

abcdef

Sample Output:

3

Описание алгоритма.

Для реализации алгоритма КМП необходимо определить префиксфункцию: функцию, возвращающая для всех префиксов строки максимальную длину подстроки, являющейся собственными префиксом и суффиксом.

Алгоритм можно описать циклом. Рассматриваем каждый префикс строки, увеличивая их на 1 символ при каждой итерации. Если новый символ совпадает с тем, что следовал за предыдущим префиксом, значение префиксфункции для данной подстроки приравнивается к предыдущему значению, увеличенному на один. Иначе рассматривается префикс, номер которого соответствует длине текущего префикс-суффикса. Проверка на соответствие повторяется, пока длина проверяемого префикса не станет равна нулю.

Алгоритм КМП для поиска подстроки *sub* в строке *str* заключается в том, чтобы посчитать префикс-функцию от строки *sub* + *%* + *str*, где *%* — символ, не встречающийся в исследуемых строках. Когда префикс-функция будет посчитана, для поиска всех вхождений будет достаточно найти все подстроки, для которых значение префикс-функции будет равно длине *sub*.

Для проверки того, что одна строка *str1* является циклическим сдвигом другой *str2*, можно использовать алгоритм КМП следующим образом:

Найдем префикс-функцию строки *str1+%+str2+str2*. Если str1 — циклический сдвиг *str2*, то при конкатенации ее с самой собой *str1* будет подстрокой *str2+str2*. Далее, если найдется значение префикс-функции, равное длине *str1*, можно будет сделать вывод о том, что *str1* — циклический сдвиг *str2*.

В рамках данного задания была использована оптимизация: так как достаточно нахождения одного вхождения подстроки, не обязательно хранить все значения префикс-функции — достаточно только предыдущего, а также значений префикс-функции для *str1*. Как только подходящее значение будет найдено, вычисление останавливается.

Оценка сложности алгоритма.

Оценим сложность алгоритма для первого задания (поиск вхождений в строке). Если длины строки A и искомой подстроки B соответственно равны m и n, сложность и по памяти, и по времени составляет O(m+n). В памяти хранится массив значений префикс-функции длины m+n, время его построения также линейно зависит от размера входных данных.

Оценим сложность алгоритма для второго задания. Если длины строк входных данных не равны, то программа сразу выводит результат, то есть затраты по времени и памяти не зависят от входных данных — O(1). В противном случае префикс-функция строится не более чем для 3*n символов, следовательно, сложность по времени составляет O(n). При этом в памяти хранятся значения префикс-функции только для строки *str1*, значит, сложность по памяти так же составляет O(n).

Итого, алгоритм для нахождения подстроки в строке имеет сложность $O(n + m)$ по времени и памяти. Алгоритм проверки на циклический сдвиг имеет сложность $O(n)$ по памяти и времени.

Используемые функции.

Функция *prefix_func(s)* принимает на вход строку *s*, возвращает список длины *len(s)*, значения элементов — длины максимальных префикссуффиксов, заканчивающихся на данном символе.

Функция *find_occurrences(p, t)* принимает на вход две строки: *p (pattern)* и *t (text)*. Строится префикс-функция для строки *p+%+t*, затем по значениям префикс-функции, равным длине *p*, вычисляются все вхождения *p* в *t*. Эти вхождения выводятся в консоль. При их отсутствии выводится -1.

Функция *find_cyclic_shift(p, t)* так же строит префикс-функцию, но для строки (*p+%+t+t*). Ищется начало строки *p* в *t+t*. Если такое находится, то оно выводится в консоль, иначе выводится -1.

Тестирование программы.

Результаты тестирования первой программы:

Входные данные	Выходные данные	Комментарий
ab abab	0,2	Тест, совпадающий с первым тестом на Stepik
abba abba	0	Тест, в котором строки совпадают
a aaaaa	0,1,2,3,4	Искомая подстрока — одна буква
abab abababab	0,2,4	Вхождения подстроки пересекаются
ab adadad	-1	Первая строка не встречается во второй

Результат тестирования второй программы:

Входные данные	Выходные данные	Комментарий
defabc abcdef	3	Тест, совпадающий с первым тестом на Stepik
abba abba	0	Тест, в котором строки совпадают
abbab babab	2	
abcd abcde	-1	Строки разной длины
aaaab aabaa	2	
defabc abcdfe	-1	Строки одной длины, но не являются сдвигом друг друга

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
def  prefix_func(s):
    prefix = [0]*len(s)

    for i in range(1, len(s)):
        if debug:
            print(f'Префикс-функция для подстроки {s[0:i+1]}')
        k = prefix[i - 1] # возьмем последний префикс-суффикс
        if debug:
            print(f'Рассмотрим префикс: {s[0:k+1]}')
        while k > 0 and s[k] != s[i]: # пока символ не совпадает и можно взять префикс-
суффикс
            k = prefix[k - 1]
            if debug:
                print(f'Рассмотрим предыдущий префикс-суффикс:{s[0:k+1]}')
        if s[k] == s[i]:
            k += 1 # увеличим значение, если можно расширить префикс-суффикс
            if debug:
                print('Это префикс-суффикс')
        prefix[i] = k
        if debug:
            print(f'Для подстроки {s[0:i+1]} префикс-функция: {k}\n')
    return prefix

def find_cyclic_shift(p, t):
    len_1, len_2 = len(p), len(t)

    if len_1 != len_2:
        print(-1)
        return
    prefix = prefix_func(p)
    if debug:
        print(f'Префикс-функция: {prefix}')
    t *= 2
    len_2 = len(t)
    prev_prefix = int(t[0] == p[0])
    for i in range(1, len_2):
        cur_prefix = prev_prefix # нужно первое вхождение - храним только текущее
значение и предыдущее
        if debug:
            print(f'Префикс функция для {t[0:len_1+i - 2]}')
        while cur_prefix > 0 and p[cur_prefix] != t[i]:
            cur_prefix = prefix[cur_prefix - 1]
            if debug:
                print(f'Рассмотрим предыдущий префикс-суффикс:{t[0:cur_prefix+1]}')
            if p[cur_prefix] == t[i]:
                cur_prefix += 1
                if debug:
                    print(f'Найдено расширение префикс-суффикса{t[0:cur_prefix]}:
{t[0:len_1+cur_prefix]}')
            if debug:
                print(f'Для подстроки {t[0:len_1 + i - 2]}: префикс-функция: {cur_prefix}\n')
```

```
            if cur_prefix == len_1:
                print((i + 1) - len_1)
                return
        prev_prefix = cur_prefix # обновляем предыдущее значение
    print(-1)

def find_occurrences(p, t):
    prefix = prefix_func(p + '%' + t)

    if debug:
        print(f'Префикс-функция: {prefix}')

    nums = []
    len_substr = len(p)

    if debug:
        print(f'Ищем значение префикс-функции, равное {len_substr}')
    for ind in range(len(t)):
        if debug:
            print(f'\nРассмотрим индекс {ind}')
        if prefix[len_substr + ind + 1] == len_substr: # если значение префикс-функции
равно длине строки
            if debug:
                print(f'Значение префикс-функции: {len_substr} -найдено вхождение в
индексе {ind - len_substr + 1}')
            nums.append(ind - len_substr + 1)
        elif debug:
            print(f'Значение префикс-функции: {prefix[len_substr + ind+ 1]}')
    if nums:
        print(*nums, sep=',')
    else:
        print(-1)

if __name__ == '__main__':
    debug = True
    step = 1
    p = input()
    t = input()
    if step == 1:
        find_occurrences(p, t)
    else:
        find_cyclic_shift(p, t)
```