**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: Алгоритм Кнута-Морриса-Пратта**

| Студентка гр. 1304 |  | Чернякова А.Д. |
| --- | --- | --- |
| Преподаватель |  | Шевелева А.М. |

Санкт-Петербург

2023

## **Цель работы.**

Изучить префикс-функцию и алгоритм Кнута-Морриса-Пратта и реализовать поиск подстроки в строке с помощью данного алгоритма.

## **Задание.**

Задание 1.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона 𝑃 (|𝑃| ≤ 15000) и текста T (|𝑇| ≤ 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.

Задание 2.

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

Вход: Первая строка – A; вторая строка - B.

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

## **Выполнение работы.**

* + 1. Задание 1.

Для алгоритма Кнута-Морриса-Пратта реализовано 4 функции: *prefix\_arr(), in\_str(), solve(), main().*

Функция ***prefix\_arr(s)*** - принимает на вход строку *s*, вычисляет для нее значения префикс-функции и возвращает список с результатами значений.

Префикс функция для *i*-того символа образа возвращает значение, равное максимальной длине совпадающих префикса и суффикса подстроки в образе, которая заканчивается *i*-м символом.

* + - * 1. Для реализации создается список значений префикс функции *p* длиной равной длине слова и заполненный нулями. Инициализируется нулем переменная *j* и реализуется цикл *for* от 1 до длины строки *s*, уменьшенной на 1(индексация с нуля), в цикле for реализуется цикл *while*: пока *j > 0* и *s[i]* не равно *s[j] j = p[j - 1]*, если *s[i]* равно *s[j]*, то *j* увеличивается на 1 и *p[j*] становится равным *j*. Таким образом точно будут найдены верные значения, равные максимальной длине совпадающих префикса и суффикса.
      1. Функция ***in\_str(t, p)*** - принимает на вход строки *t* и *p,* в *t* реализуется поиск подстроки *p* и возвращается список с индексами начал вхождений *p* в *t.* Для реализации *j* инициализируется нулем, вызывается префикс-функция для *p*, которая возвращает список *arr*. Реализуется цикл *for* по длине строки *t:* если *t[i]* равно *p[i]*, то *j* увеличивается на 1, и если *j* равно длине *p*, то найдено вхождение и в список result записывается индекс начала вхождения *p* в *t,* а *j* становится равным значению префикс-функции индекса *j - 1*; если же *t[i]* не равно *p[i]*, то реализуется цикл *while*: пока *j > 0* и *t[i]* не равно *p[j] j* становится равным значению префикс-функции индекса *j - 1*, если *t[i]* равно *p[j]*, то *j* увеличивается на 1. Далее идет очередная проверка: если *j* равно длине *p*, то найдено вхождение и в список *result* записывается индекс начала вхождения *p* в *t,* а *j* становится равным значению префикс-функции индекса *j - 1*. Затем идет проверка списка *result*: если он пуст, то возвращается -1, в противном случае строка, в которой элементы списка *arr* представлены через пробел.
      2. Функция ***solve()*** - ничего не принимает на вход, считывает строки *p* и *t*, вызывает функцию *in\_str(t, p)* и выводит результат функции в консоль.
      3. Функция ***main()*** - единственный вызов функции solve().
         1. Задание 2.
         2. Для проверки является ли A циклическим сдвигом B также используется алгоритма Кнута-Морриса-Пратта и 4 функции: *prefix\_arr(), in\_str(), solve(), main().*
         3. Функции ***prefix\_arr(s)*** *и* ***main()***аналогичные с заданием 1: *prefix\_arr(s)*- принимает на вход строку *s*, вычисляет для нее значения префикс-функции и возвращает список с результатами значений, *main()*осуществляет единственный вызов функции solve().

Функция ***solve()*** немного изменена: она по-прежнему ничего не принимает на вход, но считывает сначала строку *t,* а потом строку *p* (так как необходимо проверитьявляется ли *t* циклическим сдвигом *p*, и в случае если является, вывести в консоль индекс начала строки *p* в *t* , а не *t* в *p*), вызывает функцию *in\_str(t, p)* и выводит результат функции в консоль.

Функция ***in\_str(t, p)*** - принимает на вход строки *t* и *p* и первым делом проверяет одинаковой ли они длины, если разной, то функция возвращает -1, так как t точно не может быть циклическим сдвигом p. Далее идет проверка на совпадение строк t и p: в случае совпадения возвращается 0, так как сдвига не будет. После частных случаев идет реализация для общего. *j* инициализируется нулем, вызывается префикс-функция для *p*, которая возвращает список *arr*. Реализуется цикл *for* по удвоенной длине строки *t (*изначально рассматривался случай, когда t является склейкой двух t и цикл просто идет по строке, но такая реализация оказалась очень затратной по памяти, поэтому код был оптимизирован: строка не удваивалась, а просто for шел циклически по строке)*: i* присваивается остаток от деления i на длину t, реализуется цикл *while*: пока *j > 0* и *t[i]* не равно *p[j] j* становится равным значению префикс-функции индекса *j - 1*, если *t[i]* равно *p[j]*, то *j* увеличивается на 1, если *j* равно длине *p*, то найдено вхождение и функция возвращает индекс начала вхождения *p* в t. Если цикл for отработал полностью, значит вхождение не было найдено и функция возвращает -1.

## **Выводы.**

Изучены префикс-функция и алгоритм Кнута-Морриса-Пратта. С помощью алгоритма К-М-П реализованы задачи по поиску подстроки в строке и проверке, является ли одна строка циклическим сдвигом другой. Рассмотренный алгоритм позволяет осуществлять поиск подстроки в строке за линейное время, а также имеет простую реализацию, как и наивный алгоритм поиска. На платформе Stepik все тесты успешно пройдены, что гарантирует правильность выполнения лабораторной работы.

# **ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: task1.py

def prefix\_arr(s):

p = [0] \* len(s)

j = 0

for i in range(1, len(s)):

while j > 0 and s[i] != s[j]:

j = p[j - 1]

if s[i] == s[j]:

j += 1

p[i] = j

return p

def in\_str(t, p):

j = 0

arr = prefix\_arr(p)

result = []

for i in range(len(t)):

if t[i] == p[j]:

j += 1

if j == len(p):

result.append(str(i - len(p) + 1))

j = arr[j - 1]

else:

while j > 0 and t[i] != p[j]:

j = arr[j - 1]

if t[i] == p[j]:

j += 1

if j == len(p):

result.append(str(i - len(p) + 1))

j = arr[j - 1]

if not result:

return -1

else:

return ",".join(result)

def solve():

p = input()

t = input()

print(in\_str(t, p))

def main():

solve()

if \_\_name\_\_ == '\_\_main\_\_':

main()

Название файла: task2.py

def prefix\_arr(s):

p = [0] \* len(s)

j = 0

for i in range(1, len(s)):

while j > 0 and s[i] != s[j]:

j = p[j - 1]

if s[i] == s[j]:

j += 1

p[i] = j

return p

def in\_str(t, p):

if len(t) != len(p):

return -1

if t == p:

return 0

j = 0

arr = prefix\_arr(p)

for i in range(2\*len(t)):

i = i % len(t)

while j > 0 and t[i] != p[j]:

j = arr[j - 1]

if t[i] == p[j]:

j += 1

if j == len(p):

return str(i + 1)

return -1

def solve():

t = input()

p = input()

print(in\_str(t, p))

def main():

solve()

if \_\_name\_\_ == '\_\_main\_\_':

main()