**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе 5**

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

Тема: **Кнут-Моррис-Пратт**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3384 |  | Козьмин Н.В. |
| Преподаватель |  | Шевелева А.М. |

Санкт-Петербург

2025

## Цель работы.

Разработать программы, которые обрабатывают строки с помощью алгоритма Кнута-Морриса-Пратта.

## Задание.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P (∣P∣≤15000∣P∣≤15000) и текста T (∣T∣≤5000000∣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

Заданы две строки A (∣A∣≤5000000∣A∣≤5000000) и B (∣B∣≤5000000∣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 вляется циклическим сдвигом B, индекс начала строки B в A, иначе вывести −1. Если возможно несколько сдвигов вывести первый индекс.

## Описание работы.

Был выбран Python, так как не требуется высокая скорость выполнения и писать на этом ЯП быстрее и проще.

Первым делом была сделана префикс-функция, которая заполняла список, на каждой итерации используя предыдущее значение, если совпало несколько символов, то надо проверять только один символ с отступом от начала. Также используем предыдущие значения, чтобы при несовпадении символов, уменьшать с конца префикс и с начала суффикс для текущей позиции.

Затем был реализован алгоритм Кнута-Морриса-Пратта, используя модифицированную посимвольную проверку через применение префикс-функции для шаблона. Если символы не совпадают, то мы переходим на другую последовательность шаблона аналогично рассуждениям из префикс-функции.

Далее была сделана проверка циклического сдвига. Для этого мы просто применяем модифицированный алгоритм КМП, который проходит по двойному тексту и заканчивает работу при первом результате. Для уменьшения затрат по памяти вместо двойного текста используем остаток от деления.

## Описание функций.

Префикс-функция принимает строку и создает список для каждой позиции и проходит в цикле по позициям через предложенный алгоритм. В цикле проходим, если символы не совпали, пока дистанция между символами не ноль. Если совпали добавляем дистанцию и приравниваем для результата.

КМП поиск принимает две строки и также реализуется через уже предложенный алгоритм. Результаты храним в переменной occurrences, которую мы и возвращаем.

Проверка циклического сдвига принимает также две строки. Если длины не равны или результат не найден, то ответ -1. Возвращается индекс первого вхождения.

В main`е вызываем созданные функции.

## Выводы.

В ходе выполнения работы основной целью было разработать обработчики строк с помощью алгоритма Кнута-Морриса-Пратта. Реализованные программы успешно решают поставленные задачи.

Была сделана префикс-функция, которая используется в обеих программах. Две функции для соответствующих задач. Main с результатом.

Код получился лаконичным и работающим эффективно в естественных ограничениях языка.

Решения поставленных задач также были протестированы. Разработанный программный код см. в приложении А. Результаты тестирования см. в приложении Б.

# Приложение А Исходный код программы

Название файла: main.py

def compute\_prefix(text):

rslt = [0] \* len(text)

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

cur\_distance = rslt[i - 1]

while cur\_distance > 0 and text[i] != text[cur\_distance]:

cur\_distance = rslt[cur\_distance - 1]

if text[i] == text[cur\_distance]:

cur\_distance += 1

rslt[i] = cur\_distance

return rslt

def kmp\_search(pattern, text):

text\_len, pattern\_len = len(text), len(pattern)

pattern\_prefix = compute\_prefix(pattern)

cur\_distance = 0

occurrences = []

for i in range(text\_len):

while cur\_distance > 0 and text[i] != pattern[cur\_distance]:

cur\_distance = pattern\_prefix[cur\_distance - 1]

if text[i] == pattern[cur\_distance]:

cur\_distance += 1

if cur\_distance == pattern\_len:

occurrences.append(i - pattern\_len + 1)

cur\_distance = pattern\_prefix[cur\_distance - 1]

return occurrences if occurrences else [-1]

def cyclic\_shift\_check(text, pattern):

text\_len, pattern\_len = len(text), len(pattern)

if text\_len != pattern\_len:

return -1

pattern\_prefix = compute\_prefix(pattern)

cur\_distance = 0

for i in range(text\_len\*2):

char = text[i%text\_len]

while cur\_distance > 0 and char != pattern[cur\_distance]:

cur\_distance = pattern\_prefix[cur\_distance - 1]

if char == pattern[cur\_distance]:

cur\_distance += 1

if cur\_distance == pattern\_len:

return (i - pattern\_len + 1) % text\_len

return -1

def main():

fst\_string = input()

snd\_string = input()

print(",".join(map(str, kmp\_search(fst\_string, snd\_string))))

print(cyclic\_shift\_check(fst\_string, snd\_string))

if \_\_name\_\_ == "\_\_main\_\_":

main()

# Приложение Б Тестирование

В таблице Б.1 указаны результаты тестирования поиска КМП и проверки циклического сдвига соответственно на некоторых двух строчках.

Таблица Б.1 - Примеры тестовых случаев

|  |  |  |
| --- | --- | --- |
| № п/п | Выходные и входные данные | Комментарии |
|  | ab  abab  0,2  -1 |  |
|  | defabc  abcdef  -1  3 |  |
|  | acdf  abef  -1  -1 |  |
|  | abab  abab  0  0 |  |
|  | abab  ab  -1  -1 |  |