**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: Поиск подстроки в строке.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 3343 |  | Ермолаева В. А. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2025

## Цель работы

Реализовать алгоритм Кнута-Морриса-Пратта для поиска подстроки в строке. Использовать алгоритм для того чтобы определить, является ли одна строка циклическим сдвигом другой.

## Задание

1) Реализуйте алгоритм КМП и с его помощью для заданных шаблона P (∣P∣≤25000) 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, разделённые запятой; если P не входит в T, то вывести -1.

2) Заданы две строки A (∣A∣≤5000000) и B (∣B∣≤5000000). Определить, является ли A циклическим сдвигом В (это значит, что A и В имеют одинаковую длину и A 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, то индекс начала строки B в A; иначе вывести -1. Если возможно несколько сдвигов, вывести первый индекс.

## 

## Выполнение работы

Алгоритм Кнута-Морриса-Пратта решает задачу поиска подстроки в строке. Для эффективного решения используется префикс-функция т.е. массив чисел p[0...n-1], где p[i] определяется следующим образом: это такая наибольшая длина наибольшего собственного суффикса подстроки s[0...i], совпадающего с её префиксом (собственный суффикс - значит не совпадающий со всей строкой).

Рассмотрим сравнение строк на позиции i, где образец S[0, m − 1] сопоставляется с частью текста T[i, i + m − 1]. Предположим, что первое несовпадение произошло между T[ i + j ] и S[ j ], где 1 < j < m. Тогда T[i , i + j − 1 ] = S[ 0 , j − 1 ] = P и a = T[ i + j ] ≠ S[ j ] = b.

При сдвиге вполне можно ожидать, что префикс образца S сойдется с каким-нибудь суффиксом текста P. Длина наиболее длинного префикса, являющегося одновременно суффиксом, есть значение префикс-функции от строки S для индекса j.

Это приводит нас к следующему алгоритму: пусть π[j] — значение префикс-функции от строки S[0, m − 1] для индекса j. Тогда после сдвига мы можем возобновить сравнения с места T[ i + j ] и S[ π[ j ] ] без потери возможного местонахождения образца.

Временная сложность алгоритма КМП составляет O(n + m), где n - длина образца, m - длина строки. Пространственная же сложность составляет O(m), так как хранится массив префикс-функции.

Описание реализованных функций:

* def constructLps(pat): строит префикс-функцию для заданного образца pat.
* def search(pat, txt): осуществляет поиск вхождений образца pat в текст txt.

Исходный код программы смотреть в приложении А.

## Тестирование

Результаты тестирования представлены в таблице 1.

Табл. 1. – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 1. | pat = "aba"  txt = "ababcaba" | 0, 5 | Результат соответствует ожиданиям. |
| 2. | str1 = "defabc"  str2 = "abcdef" | 3 | Результат соответствует ожиданиям. |
| 3. | pat = "ссс"  txt = "ababcaba" | -1 | Результат соответствует ожиданиям, образец не встречается в строке. |

## Выводы

В ходе выполнения лабораторной работы был реализован алгоритм Кнута-Морриса-Пратта для поиска подстроки в строке. Алгоритм был использован для того чтобы определить, является ли одна строка циклическим сдвигом другой.

# Приложение А

def construct\_lps(pat):

print(f"----------------\nШаг 1. Вычислим префикс-функцию для образца '{pat}'.")

lps = [0] \* len(pat)

j = 0

i = 1

while i < len(pat):

print(f"Сравним символ '{pat[i]}' строки на позиции [{i}] с символом наибольшего префикс-суффикса '{pat[j]}' на позиции [{j}]:")

if pat[i] == pat[j]:

print(f"\t -> Обнаружено совпадение: '{pat[i]}' на позиции [{i}] = '{pat[j]}' на позиции [{j}].")

print(f"\t -> Сдвигаемся дальше по обоим позициям [{j}] ---> [{j + 1}], [{i}] ---> [{i + 1}].")

j += 1

lps[i] = j

print(f"\t -> Значение префикс-функции в ячейке [{i}] = [{j}].\n\tПолучившаяся префикс-функция: {lps}.")

i += 1

else:

print(f"\t -> Символы не совпали: '{pat[i]}' на позиции [{i}] <> '{pat[j]}' на позиции [{j}].")

if j != 0:

print(f"\t -> Откатываемся по префикс-функции [{j}] ---> [{lps[j - 1]}].")

j = lps[j - 1]

else:

print(f"\t -> Совпадений не найдено. Сдвигаемся дальше по позиции в строке [{i}] ---> [{i + 1}].")

lps[i] = 0

i += 1

return lps

def search(pat, txt):

lps = construct\_lps(pat)

res = []

i = 0

j = 0

print(f"----------------\nШаг 2. Применим КМП для поиска вхождений '{pat}' в '{txt}'.")

while i < len(txt):

print(f"Сравним символ '{txt[i]}' текста на позиции [{i}] с символом образца '{pat[j]}' на позиции [{j}]:")

if txt[i] == pat[j]:

print(f"\t -> Обнаружено совпадение: '{txt[i]}' на позиции [{i}] = '{pat[j]}' на позиции [{j}].")

print(f"\t -> Сдвигаемся по обоим позициям [{j}] ---> [{j + 1}], [{i}] ---> [{i + 1}].")

i += 1

j += 1

if j == len(pat):

print(f"\tОбнаружено вхождение образца '{pat}' на позиции [{i - j}].")

print(f"\t -> Сдвигаемся по образцу в позицию [{lps[j - 1]}].")

res.append(i - j)

j = lps[j - 1]

else:

print(f"\t -> Символы не совпали: '{txt[i]}' на позиции [{i}] <> '{pat[j]}' на позиции [{j}].")

if j != 0:

print(f"\t -> Откатываемся по префикс-функции [{j}] ---> [{lps[j - 1]}].")

j = lps[j - 1]

else:

print(f"\t -> Совпадений не найдено. Сдвигаемся дальше по позиции в строке '{txt}' [{i}] ---> [{i + 1}].")

i += 1

if len(res) == 0:

print(f"Вхождений строки не было обнаружено.")

res.append(-1)

return res

var = int(input("Выберите задание:\n\t1. Поиск вхождений образца.\n\t2. Проверка на циклический сдвиг.\n"))

# Задание 1

if var == 1:

pat = "aba"

txt = "ababcaba"

answer = ", ".join(map(str, search(pat, txt)))

print(f"----------------\nШаг 3. Вывод результатов.")

print(f" -> Образец '{pat}' встретился в тексте '{txt}' на позициях: {answer}.")

else:

# Задание 2

str1 = "defabc"

str2 = "abcdef"

print(f" -> Удвоим первую строку: {str1} -> {str1+str1}.")

answer = ", ".join(map(str, search(str2, str1+str1)))

print(f"----------------\nШаг 3. Вывод результатов.")

print(f" -> Циклический сдвиг '{str1}' начинается в строке '{str2}' на позиции {answer}.")