**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: Кнут-Моррис-Пратт

|  |  |  |
| --- | --- | --- |
| Студент гр. 1384 |  | Алиев Д.А. |
| Преподаватель |  | Шевелева А.М. |

Санкт-Петербург

2023

**Задание.**

**Первое задание.**

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, разделённых запятой, если P не входит в T, то вывести −1.

**Второе задание.**

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

**Выполнение работы.**

Задание 1.

Код содержит реализацию двух функций: **prefix\_function** и **knuth\_morris\_pratt**.

Функция **prefix\_function** вычисляет префикс-функцию заданной строки, которая используется в алгоритме KMP. Префикс-функция определяет наибольшую длину суффикса строки, который является ее префиксом.

Функция **knuth\_morris\_pratt** реализует алгоритм KMP для поиска всех вхождений заданного образца в заданный текст. Алгоритм работает следующим образом: сначала вычисляется префикс-функция для образца, затем образец и текст объединяются в одну строку с разделителем "#". Далее для каждой позиции в тексте вычисляется значение префикс-функции для соответствующей позиции в объединенной строке. Если значение префикс-функции для позиции, следующей за концом образца, равно длине образца, то обнаружено вхождение образца в тексте.

Функция **main** является основной функцией программы, которая получает ввод от пользователя, вызывает функцию **knuth\_morris\_pratt** для поиска вхождений образца в тексте, и выводит результаты на экран.

Код решает задачу поиска вхождений образца в тексте с помощью алгоритма KMP.

Задание 2.

Было замечено, что если строки являются циклическими сдвигами друг друга, то одна из них окажется подстрокой в строке, где вторая написана дважды, то есть:

S1 = ‘aabb’

S2 = ‘bbaa’

‘aabb’ is in S2\*2(‘bb**aabb**aa’)

Таким образом мы можем запустить алгоритм Кнута-Морриса-Пратта, в котором подстрокой выступает первая входная строка, а всем текстом удвоенная вторая входная строка. Для избежания столкновения с memory limit код был переписан с python на c++ и модифицирован. При передаче в функцию объект не копируется, так как передаётся его ссылка, что и позволяет экономить память.

Исходный код обеих программ представлен в приложении A.

**Выводы.**

В данной лабораторной работе был рассмотрен и изучен самый эффективный алгоритм поиска подстроки в строке (алгоритм Кнута-Морриса-Пратта). Были решены две задачи, одна и которых заключалась в реализации алгоритма, а вторая в решении задачи о циклическом сдвиге с использованием алгоритма из предыдущего задания. Для более эффективной работы программы второе задание было написано на языке C++. Оба кода успешно прошли проверку на сайте Stepik.org, что говорит о корректном решении поставленных задач.

Приложение A

Исходный код

task1.py:

def prefix\_function(string):

"""

Computes the prefix function of a given string.

Args:

string (str): The input string for which the prefix function needs to be computed.

Returns:

list: A list of integers representing the prefix function values of the string.

"""

string\_length = len(string)

max\_prefixes = [0 for \_ in range(string\_length)]

max\_prefixes[0] = 0

for i in range(1, string\_length):

tmp\_index = max\_prefixes[i - 1]

while tmp\_index > 0 and string[i] != string[tmp\_index]:

tmp\_index = max\_prefixes[tmp\_index - 1]

if string[i] == string[tmp\_index]:

tmp\_index += 1

max\_prefixes[i] = tmp\_index

return max\_prefixes

def knuth\_morris\_pratt(pattern, text):

"""

Implements the Knuth-Morris-Pratt (KMP) algorithm to find all occurrences of a pattern in a given text.

Args:

pattern (str): The pattern to be searched for in the text.

text (str): The text in which the pattern needs to be searched.

Returns:

list: A list of integers representing the starting indices of all occurrences of the pattern in the text.

"""

patternLength = len(pattern)

textLength = len(text)

answer = []

prefixes = prefix\_function(pattern + '#' + text)

for i in range(0, textLength):

if prefixes[patternLength + i + 1] == patternLength:

answer.append(i - patternLength + 1)

return answer

def main():

"""

Main function to take input from user, call the knuth\_morris\_pratt function, and print the results.

"""

pattern = input()

text = input()

answer = knuth\_morris\_pratt(pattern, text)

if len(answer):

print(\*answer, sep=',')

else:

print(-1)

main()

task2.cpp

def prefix\_function(string):

"""

Computes the prefix function of a given string.

Args:

string (str): The input string for which the prefix function needs to be computed.

Returns:

list: A list of integers representing the prefix function values of the string.

"""

string\_length = len(string)

max\_prefixes = [0 for \_ in range(string\_length)]

max\_prefixes[0] = 0

for i in range(1, string\_length):

tmp\_index = max\_prefixes[i - 1]

while tmp\_index > 0 and string[i] != string[tmp\_index]:

tmp\_index = max\_prefixes[tmp\_index - 1]

if string[i] == string[tmp\_index]:

tmp\_index += 1

max\_prefixes[i] = tmp\_index

return max\_prefixes

def knuth\_morris\_pratt(pattern, text):

"""

Implements the Knuth-Morris-Pratt (KMP) algorithm to find all occurrences of a pattern in a given text.

Args:

pattern (str): The pattern to be searched for in the text.

text (str): The text in which the pattern needs to be searched.

Returns:

list: A list of integers representing the starting indices of all occurrences of the pattern in the text.

"""

patternLength = len(pattern)

textLength = len(text)

answer = []

prefixes = prefix\_function(pattern + '#' + text)

for i in range(0, textLength):

if prefixes[patternLength + i + 1] == patternLength:

answer.append(i - patternLength + 1)

return answer

def main():

"""

Main function to take input from user, call the knuth\_morris\_pratt function, and print the results.

"""

pattern = input()

text = input()

answer = knuth\_morris\_pratt(pattern, text)

if len(answer):

print(\*answer, sep=',')

else:

print(-1)

main()