**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. 0382 _____ Кондратов Ю.А.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2022

**Цель работы.**

Написать две программы. Одну - реализовывающую поиск всех вхождений подстроки в строке, вторую – определяющую является ли одна строка циклическим сдвигом другой

**Задание.**

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

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

**Выполнение работы.**

Для решения обеих задач используется префикс-функция. Префикс-функцией от строки s называется массив p, где $p_i$ равно длине самого большого префикса строки $S_0, S_1, S_2 \ldots S_i$, который также является и суффиксом.

Префикс-функция была реализована в функции *prefix(s: str) -> list[int]* по алгоритму, представленному на рисунке 1.

```
COMPUTE_PREFIX_FUNCTION(P)
 1  m ← length[P]
 2  π[1] ← 0
 3  k ← 0
 4  for q ← 2 to m
 5        do while k > 0 и P[k + 1] ≠ P[q]
 6              do k ← π[k]
 7           if P[k + 1] = P[q]
 8              then k ← k + 1
 9           π[q] ← k
10  return π
```

Рисунок 1 – Алгоритм реализации префикс-функции

Для решения первой задачи используется версия алгоритма Кнута-Морисса-Пратта, в которой сначала из считанных строк *pattern* и *text* составляется строка *pattern#text*. Далее для этой строки считается префикс-функция *pi* и, так как символ # не входит ни в левую, ни в правую часть, если *pi[i] =len(pattern)*, то i – индекс конца вхождения *pattern* в *text.*

Для решения второй задачи используется реализация алгоритма, представленного на рисунке 2.

```
KMP_MATCHER(T, P)
 1  n ← length[T]
 2  m ← length[P]
 3  π ← COMPUTE_PREFIX_FUNCTION(P)
 4  q ← 0
 5  for i ← 1 to n
 6        do while q > 0 и P[q + 1] ≠ T[i]
 7              do q ← π[q]
 8           if P[q + 1] = T[i]
 9              then q ← q + 1
10           if q = m
11              then
12                   q ← π[q]
```

Рисунок 2 – Алгоритм для решения второй задачи

В качестве *P* выступает строка *B*, в качестве *T* выступает строка *A* + *A* (конкатенация *A* и *A*). Алгоритм реализован так, что завершает работы при нахождении первого вхождения *P* в *T*. Так как *T* содержит все циклические сдвиги *A*, то таким образом мы определим, является ли *B* циклическим сдвигом *A*.

**Оценка сложности алгоритмов.**

Алгоритмы для решения задач имеют одинаковую сложность, так как используют разные версии одного и того же алгоритма.

Вычисление префикс-функции происходит за O(n), где n – длина строки. Для определения всех вхождений необходимо один раз пройти по массиву, возвращённому префикс-функцией. Получаем итоговую сложность O(|T| + |P|).

**Тестирование.**

Тестирование производилось при помощи библиотеки pytest.

Рассмотренные при тестировании решения первой задачи случаи представлены в таблице 1.

Таблица 1 – Результаты тестирования решения первой задачи

| Входные данные | Выходные данные | Описание | Вердикт |
|---|---|---|---|
| ab<br>abab | 0,2 | Тест из условия | passed |
| abc<br>asdlkfj | -1 | Вхождения отсутствуют | passed |
| abab<br>abababababab | 0,2,4,6,8 | Вхождения пересекаются | passed |
| qwertyui<br>qwer | -1 | Паттерн длиннее текста | passed |
| qwert | 0 | Паттерн и текст совпадают | passed |

| | | | |
|---|---|---|---|
| qwert | | | |
| qwert<br>qwergqwertqwerhq<br>weryqwerqwerqwqw<br>qwerlkjqriqweroqern | 5 | В тексте много похожих паттернов но только одно точное вхождение | passed |

Рассмотренные при тестировании решения второй задачи случаи представлены в таблице 2.

Таблица 2 – Результаты тестирования решения второй задачи

| Входные данные | Выходные данные | Описание | Вердикт |
|---|---|---|---|
| defabc<br>abcdef | 3 | Тест из условия | passed |
| asdfg<br>adsfg | -1 | Первая строка не является циклическим сдвигом второй | passed |
| qwert<br>qwer | -1 | Строки разной длины | passed |
| abababa<br>bababab | -1 | Строки похожи, но первая не является циклическим сдвигом второй | passed |
| abababab<br>abababab | 0 | Строки совпадают | passed |

Протокол тестирования представлен на рисунке 3.

```
platform win32 -- Python 3.10.2, pytest-7.1.2, pluggy-1.0.0
rootdir: C:\LETI\DAA\lab4
collected 11 items

test_knut_morris_pratt.py ...........                                    [100%]

================================== 11 passed in 0.06s ==================================
```

Рисунок 3 – Протокол тестирования

**Выводы.**

В результате выполнения работы был изучен алгоритм поиска вхождений подстроки в строке – алгоритм Кнута-Морисса-Пратта. Было изучено понятие префикс-функции и способ её оптимального построения. Была написаны программы: выполняющая поиск вхождений подстроки в строку, определяющая является ли одна строка циклическим сдвигом другой.

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main1.py

```
def prefix(s: str) -> list[int]:
    pi = [0] * len(s)
    for i in range(1, len(s)):
        k = pi[i - 1]
        while k > 0 and s[k] != s[i]:
            k = pi[k - 1]
        if s[k] == s[i]:
            k += 1
        pi[i] = k
    return pi


def knut_morris_pratt(pattern, text):
    pi = prefix(pattern + '#' + text)
    res = []
    for i, l in enumerate(pi):
        if l == len(pattern):
            res.append(i - len(pattern) * 2)
    return res if res else [-1]


if __name__ == "__main__":
    pattern, text = input(), input()
    print(*knut_morris_pratt(pattern, text), sep=',')
```

Название файла: main2.py

```
def prefix(s: str) -> list[int]:
    pi = [0] * len(s)
    for i in range(1, len(s)):
        k = pi[i - 1]
        while k > 0 and s[k] != s[i]:
            k = pi[k - 1]
        if s[k] == s[i]:
            k += 1
        pi[i] = k
    return pi


def knut_morris_pratt(pattern: str, text: str) -> int:
    text *= 2
    p_len, t_len = len(pattern), len(text)

    if p_len != t_len / 2:
        return -1

    pi = prefix(pattern)

    q = 0
    for i in range(t_len):
        while q > 0 and pattern[q] != text[i]:
            q = pi[q - 1]
```

```
        if pattern[q] == text[i]:
            q += 1
        if q == p_len:
            return i - p_len + 1
    return -1


if __name__ == "__main__":
    pattern, text = input(), input()
    print(knut_morris_pratt(text, pattern))
```