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

Студентка гр. 1381 _____ Рымарь М.И.

Преподаватель _____ Токарев А.П.

Санкт-Петербург
2023

Цель работы.

Изучить алгоритм Кнутта-Морриса-Пратта. Написать программу, реализующую этот алгоритм для нахождения вхождений подстрок в строку и для определения, является ли строка циклическим сдвигом другой строки.

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

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

Основные теоретические положения.

Алгоритм Кнута-Морриса-Пратта (KMP) — это алгоритм поиска подстроки в строке, который позволяет найти все вхождения заданной подстроки за линейное время относительно длины строки и подстроки.

Алгоритм KMP основан на использовании префикс-функции, которая для каждой позиции в строке находит длину наибольшего префикса подстроки, который является ее суффиксом. Это позволяет эффективно перемещаться по строке, пропуская некоторые символы, если они не могут входить в искомую подстроку.

С помощью алгоритма KMP можно быстро найти все вхождения подстроки в строку, а также проверить, является ли строка периодической (т.е. состоит из повторяющихся блоков).

Выполнение работы.

1. Код реализует алгоритм Кнута-Морриса-Пратта для поиска подстроки *pattern* в строке *text*. В функции *prefix* находятся длины наибольших общих префиксов и суффиксов каждого префикса для строки *string*. Для этого используется цикл, который сравнивает символы строки и записывает в список *pref* максимальную длину совпадающего префикса-суффикса для каждой позиции в строке. Затем в функции *KMP* создается список *answer*, в котором будут храниться индексы всех вхождений подстроки *pattern* в строку *text*. Для этого используется префикс-функция для объединения строк *pattern* и *text*, разделенных символом #. Затем проходится по строке *text*, и если значение префикс-функции равно длине *pattern*, то это означает, что найдено вхождение подстроки *pattern* в строку *text*. Этот индекс добавляется в список ответов. Если вхождений не найдено, возвращается список ["-1"]. В конце кода функция *KMP* вызывается с аргументами, которые считываются из ввода, и результат выводится в виде строки, где индексы разделены запятой. Полный программный код см. в Приложении А.

2. Данный код реализует алгоритм Кнута-Морриса-Пратта (KMP) для поиска всех вхождений шаблона *string1* в текст *string2*. Сначала функция *prefix* создает префикс-функцию для шаблона, где для каждого индекса *i* определяется длина наибольшего префикса подстроки, заканчивающейся в этом индексе и являющейся суффиксом этой же подстроки. Затем функция KMP использует эту

префикс-функцию для поиска всех вхождений string1 в string2. Алгоритм работает следующим образом. Сначала проверяется, имеют ли строки *string1* и *string2* одинаковую длину. Если нет, то возвращается массив из одного элемента -1, что означает, что шаблон не может быть найден в тексте. Затем создается префикс-функция для *string2*. Далее, при помощи двух указателей *i* и *k*, происходит обход строки *string2*. Указатель *i* перебирает все символы строки *string1* циклически до конца, а указатель *k* указывает на текущий символ строки *string2*. Если *string1[i]* и *string2[k]* совпадают, то инкрементируется указатель *k*. Если *k* достигает длины строки *string2*, то это означает, что вхождение *string1* найдено, и его позиция добавляется в ответ. Если *k* не равен длине string2, то указатель *i* инкрементируется, и процесс продолжается сравнением символа *string1[i]* и *string2[k]*, при этом *k* обновляется с помощью префикс-функции. Если после окончания цикла ответ все еще пустой, то добавляется элемент -1, что означает, что вхождение не найдено. Код содержит проверку на ошибки, так как при попытке обращения к несуществующим индексам в массиве может возникнуть ошибка.

Тестирование.

1. Тесты покрывают основные случаи использования функции *KMP* и проверяют ее правильность в различных сценариях работы. Было написано 5 тестов. Первый тест *test1* проверяет, что функция правильно обрабатывает случай, когда подстрока не содержится в тексте. Второй тест *test2* проверяет, что функция правильно находит все вхождения подстроки в тексте. Третий тест *test3* проверяет, что функция правильно обрабатывает случай, когда подстрока совпадает с текстом. Четвертый тест *test4* проверяет, что функция правильно обрабатывает случай, когда подстрока длиннее текста. Пятый тест *test5* проверяет, что функция правильно находит первое вхождение подстроки в тексте, когда подстрока встречается только один раз в тексте. Результаты тестирования первой программы представлены на рисунке 1. Код тестирующего файла см. в Приложении А.

Рисунок 1 – Результаты тестирования первой программы

2. Было написано 5 тестов для проверки правильность написанной функции *KMP* в различных сценариях работы. Первый тест *test1* проверяет, что функция правильно обрабатывает случай, когда на вход подаются строки различной длины. Второй тест *test2* проверяет, что функция корректно работает, когда на вход подаются одинаковые строки. Третий тест *test3* и четвёртый тест *test4* проверяют, что функция правильно определяет индекс, с которого начинается смещение цикла. Пятый тест *test5* проверяет поданные на вход строки, состоящие из одинаковых наборов символов одинаковой длины, но не содержащие цикла. Результаты тестирования второй программы представлены на рисунке 2. Код тестирующего файла см. в Приложении В.

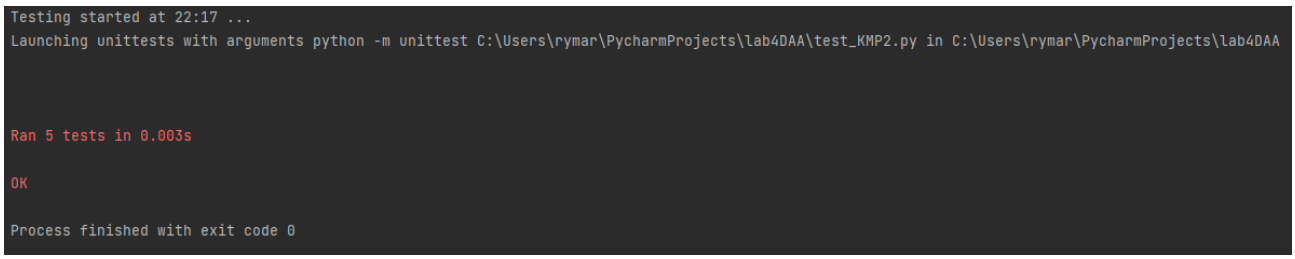

Рисунок 2 – Результаты тестирования второй программы

Выводы.

В ходе выполнения лабораторной работы изучен алгоритм Кнута-Морриса-Пратта, написана программа, реализующая этот алгоритм для нахождения вхождений подстрок в строку и определения, является ли строка циклическим сдвигом другой строки.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ ПРОГРАММНЫЙ КОД

Название файла: KMP1.py

```
def prefix(string):
    pref = [0] * len(string)
    for i in range(1, len(string)):
        k = pref[i - 1]
        while k > 0 and string[k] != string[i]:
            k = pref[k - 1]
        if string[k] == string[i]:
            k += 1
        pref[i] = k
    return pref


def KMP(pattern, text):
    answer = []
    pref = prefix(pattern + '#' + text)
    for i in range(len(text)):
        if pref[i + len(pattern) + 1] == len(pattern):
            answer.append(str(i - len(pattern) + 1))
    return answer if answer != [] else ["-1"]

if __name__ == '__main__':
    pattern = input()
    text = input()
    print(",".join(KMP(pattern, text)))
```

Название файла: KMP2.py

```
def prefix(string):
    pref = [0] * len(string)
    for i in range(1, len(string)):
        k = pref[i - 1]
        while k > 0 and string[k] != string[i]:
            k = pref[k - 1]
        if string[k] == string[i]:
            k += 1
        pref[i] = k
    return pref

def KMP(string1, string2):
    length1 = len(string1)
    length2 = len(string2)
    answer = []
    if length1 != length2:
        answer.append(-1)
        return answer
    pref = prefix(string2)
    k = 0
    for i in range(2*length2):
        while k > 0 and string1[i % length2] != string2[k]:
            k = pref[k-1]
            if k == 0:
                break
        if string1[i % len(string2)] == string2[k]:
```

```
            k += 1
        if k == length1:
            answer.append(i - length1 + 1)
            break
    if not answer:
        answer.append(-1)
    return answer

if __name__ == '__main__':
    string1 = input()
    string2 = input()
    print(*KMP(string1, string2), sep = ',')
```

Название файла: test-KMP1.py

```
import unittest
import KMP1

class MyTestCase(unittest.TestCase):
    def test1(self):
        self.assertEqual(KMP1.KMP("abc", "asdffghjkl"), ["-1"])
    def test2(self):
        self.assertEqual(KMP1.KMP("abc", "abcabcabcabc"), ['0', '3', '6',
'9'])
    def test3(self):
        self.assertEqual(KMP1.KMP("longword", "longword"), ['0'])
    def test4(self):
        self.assertEqual(KMP1.KMP("word", "wo"), ["-1"])
    def test5(self):
        self.assertEqual(KMP1.KMP("pattern", "rrrpatternllllll"), ['3'])

if __name__ == '__main__':
    unittest.main()
```

Название файла: test-KMP2.py

```
import unittest
import KMP2

class MyTestCase(unittest.TestCase):
    def test1(self):
        self.assertEqual(KMP2.KMP("word", "longword"), [-1])
    def test2(self):
        self.assertEqual(KMP2.KMP("word", "word"), [0])
    def test3(self):
        self.assertEqual(KMP2.KMP("word", "dwor"), [3])
    def test4(self):
        self.assertEqual(KMP2.KMP("word", "ordw"), [1])
    def test5(self):
        self.assertEqual(KMP2.KMP("word", "rodw"), [-1])

if __name__ == '__main__':
    unittest.main()
```