МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №43

по дисциплине «Построение и анализ алгоритмов»

Тема: Кнут-Моррис-Пратт

Студен гр. 1381 _____ Возмитель В.Е.

Преподаватель _____ Токарев А. П

Санкт-Петербург

2023

Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта и применить его на практике.

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.

Ход работы.

1. Для реализации алгоритма были написаны функции *prefix_search* и *kmp_search.*

Функция *prefix_search* принимает аргументом строку и вычисляет для определенных подстрок длину максимального бордера, путем сравнивания символов префикса и суффикса, при равенстве символов - увеличивается длина бордера. Функция возвращает список длин бордеров для подстрок.

Также была написана функция *kmp*, которая принимает аргументами строки шаблона и текста, считает префикс-функцию для шаблона и в основном цикле *while* находить все вхождения шаблона, путем сравнивания символов каждой строки. Если символы не равны, то обращаемся к массиву бордеров. При равенстве счетчика с длиной шаблона записываем нужное значение в массив-результат.

В итоге программа выводит индексы начала вхождений подстроки, разделенные запятыми, а если таких не оказалось, то выводится -1.

2. Для определения циклического сдвига была изменена функция kmp_*search*. Основная идея решения состоит в том, чтобы удвоить строку *txt*, и с помощью алгоритма КМП искать первое вхождение подстроки в строку. Если такое имеется, и длины совпадают, значит *txt* является циклическим сдвигом *pattern'a*.

В самом начале алгоритма происходит сравнение длин паттерна и строки, если они не совпадают – определенно одно не является циклическим сдвигом другого, - выходим из функции и возвращаем -1.

Аналогично первой задачи сначала высчитываем массив бордеров. Далее удваиваем строку *txt* и ищем первое вхождение *patterna'a*, и с помощью *break* выходим из основного цикла. Если не было получено результата, то программа выведет -1. Иначе одна строка является циклическим сдвигом другой, следовательно, программа вернет индекс первого вхождения.

Исходный код программы см. в приложении А.

Тестирование

Для проверки работы программ были разработаны коды тестовых программ.

- *test_1*. Тест для проверки работы программы взят с сайта размещения лабораторной.

- *test_2*. Данный тест был взят для проверки программы при нормальных условиях.

- *test_3*. Тест для проверки работы программ с одинаковыми символами в строках.

- test_4. Тест для проверки работы программы с абсолютно разными строками.

- *test_5*. Тест для проверки работы программы с пустыми строками.

Коды файлов с тестами находятся в приложении А.

Выводы.

В ходе лабораторной работы был изучен алгоритм Кнута-Морриса-Пратта и были написаны программы, реализующие алгоритм поиска подстроки в строке и определения циклического сдвига.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла PIAA_4_1.py:

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


def kmp_search(ptrn, txt):
    i = 0  # text
    j = 0  # pattern
    lsp = prefix_search(ptrn)
    res = []
    while i < len(txt):
        if txt[i] == ptrn[j]:
            i += 1
            j += 1
        else:
            if j != 0:
                j = lsp[j - 1]
            else:
                i += 1
        if j == len(ptrn):
            res.append(i - j)
            j = lsp[j - 1]

    return res


def main(txt="", ptrn=""):
    if txt == "" and ptrn == "":
        ptrn = input()
        txt = input()
    res = kmp_search(ptrn, txt)
    return ",".join(map(str, res)) if res else "-1"


if __name__ == '__main__':
    print(main())
```

Название файла PIAA_4_2.py:

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


def kmp_search(txt, ptrn):
    ptrn_len = len(ptrn)
    txt_len = len(txt)
    res = []
    j = 0

    prefix = prefix_search(ptrn)
    if ptrn_len == txt_len:
        for i in range(0, 2 * txt_len):
            while j > 0 and txt[i % txt_len] != ptrn[j]:
                j = prefix[j - 1]
                if j == 0:
                    break

            if txt[i % txt_len] == ptrn[j]:
                j += 1

            if j == ptrn_len:
                res.append(i - ptrn_len + 1)
                break

    if not res:
        res.append(-1)

    return res


def main():
    res = kmp_search(input(), input())
    print(*res, sep=',')


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

Название файла Test4.py:

```
import unittest
import PIAA_4_1
import PIAA_4_2


class TestMethods(unittest.TestCase):
    def test_1(self):
        self.assertEqual(PIAA_4_1.main("abab", "ab"), ",".join(map(str,
[0, 2])))
        self.assertEqual(PIAA_4_2.kmp_search("abcdef", "defabc"), [3])

    def test_2(self):
        self.assertEqual(PIAA_4_1.main('qwertyqw', 'qw'),
",".join(map(str, [0, 6])))
        self.assertEqual(PIAA_4_2.kmp_search("qwerty", "rtyqwe"), [3])

    def test_3(self):
        self.assertEqual(PIAA_4_1.main('aaaaaaa', 'aa'),
",".join(map(str, [0, 1, 2, 3, 4, 5])))
        self.assertEqual(PIAA_4_2.kmp_search("abc", "abc"), [0])

    def test_4(self):
        self.assertEqual(PIAA_4_1.main('abc', 'f'), "-1")
        self.assertEqual(PIAA_4_2.kmp_search("ae", "c"), [-1])

    def test_5(self):
        self.assertEqual(PIAA_4_2.kmp_search("", ""), [-1])


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