**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.

1. Заданы две строки А (A∣≤5000000) и B(B∣≤5000000).  
    Определить, является ли А циклическим сдвигом В (это значит, что А и В имеют одинаковую длину и А 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, 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.

1. 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’а*.

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’а*, 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 c абсолютно разными строками.

- *test\_5.* Тест для проверки работы программы c 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()