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: Алгоритм Кнута-Морриса-Пратта

Студент гр. 1304 Поршнев Р.А.

Преподаватель Шевелева А.М.

Санкт-Петербург

2023

Цель работы.

Изучить классический алгоритм Кнута-Морриса-Пратта для поиска образца в тексте.

Задание 1.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P ($|P| \leq 15000$) и текста T ($|T| \leq 5000000$) 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.

Задание 2.

Заданы две строки A ($|A| \leq 5000000$) и B ($|B| \leq 5000000$).

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.

1. Для решения данного задания реализованы следующие функции:

- *run()* – данная функция запускает логику решения задачи: запуск функции считывания строк, поиска решения задачи, вывода ответа на задачу;

- *get_read_text_and_pattern()* – данная функция предназначена для считывания паттерна и текста. Выходные данные: текст и паттерн в виде кортежа;
- *kmp_matcher(text, pattern)* – данная функция реализует алгоритм Кнута-Морриса-Пратта для поиска образца в тексте. Входные данные: текст и образец. Выходные данные: индексы, начиная с которых образец входит в текст;
- *compute_prefix_function(pattern)* – данная функция предназначена для расчёта префикс-функции образца, подаваемого на вход. Входные данные: образец. Выходные данные: значения префикс-функции для данного образца;
- *kmp_matcher(text, pattern)* – функция, в которой реализован алгоритм Кнутта-Морриса-Пратта и вызывается функция *compute_prefix_function(pattern).* Входные данные: текст и образец. Выходные данные: список индексов, начиная с которых образец входит в текст;
- *print_positions_of_occurrences(positions_of_occurrences)* – данная функция предназначена для вывода индексов, начиная с которых образец входит в текст. Входные данные: список индексов, начиная с которых образец входит в текст.

Код находится в Приложении А.

2. Реализация данной задачи от предыдущей отличается тем, что в качестве текста выступает удвоенная первая считанная строка, а в качестве образца – вторая считанная строка. Также отличие заключается в том, что поиск производится до тех пор, пока не найдётся первое совпадение образца в тексте. Реализованы следующие методы:

- *run()* – данная функция запускает логику решения задачи: запуск функции считывания строк, поиска решения задачи, вывода ответа на задачу;
- *get_read_text_and_pattern()* – данная функция предназначена для считывания двух строк. Выходные данные: удвоенная первая считанная строка и вторая строка в виде кортежа;
- *compute_prefix_function(pattern)* – данная функция предназначена для расчёта префикс-функции образца, подаваемого на вход. Входные данные: образец. Выходные данные: значения префикс-функции для данного образца;

- *find_shift_position(text, pattern)* – данная функция предназначена для поиска позиции сдвига строки *pattern* относительно *text*. Функция основана на алгоритме Кнута-Морриса-Пратта. До начала работы алгоритма вызывается *compute_prefix_function(pattern)*. Входные данные: текст и образец. Выходные данные: позиция сдвига образца относительно текста;
- *print_shift_position(shift_position)* – функция предназначена для вывода позиции сдвига образца относительно текста. Входные данные: позиция сдвига второй строки относительно первой.

Код находится в Приложении А.

Выводы.

В ходе выполнения данной лабораторной работы был изучен алгоритм Кнута-Моррис-Пратта для поиска образца в тексте. Также было изучено приложение данного алгоритма в задаче для поиска циклического сдвига одной строки относительно другой. Для решения данной задачи использован язык Python, повторён его синтаксис.

Разработаны программы на языке Python, которые реализуют алгоритм Кнута-Морриса-Пратта и задачу, являющейся его приложением, в которой нужно определить, является ли одна строка циклическим сдвигом другой. Обе программы имеют документацию к каждой реализованной функции, что повышает их понимание другими программистами.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab4_1.py

```
    # Данная функция вычисляет префикс-функцию для строки, переданной на
вход.
    # Входные данные: строка, для которой нужно вычислить префикс-функцию.
    # Выходные данные: префикс-функция для строки, переданной на вход.
    def compute_prefix_function(pattern):
        pattern_length = len(pattern)
        prefix_function = [0] * pattern_length
        j = 0
        for i in range(1, pattern_length):
            while j > 0 and pattern[i] != pattern[j]:
                j = prefix_function[j - 1]
            if pattern[j] == pattern[i]:
                j += 1
            prefix_function[i] = j
        return prefix_function


    # Данная функция предназначена для поиска индексов, начиная с которых
    # строка pattern входит в text.
    # Входные данные: текст и образец.
    # Выходные данные: индексы, начиная с которых pattern входит в text.
    def kmp_matcher(text, pattern):
        pattern_length = len(pattern)
        text_length = len(text)
        positions_of_occurrences = []
        j = 0
        prefix_function = compute_prefix_function(pattern)
        for i in range(text_length):
            while j > 0 and pattern[j] != text[i]:
                j = prefix_function[j - 1]
            if pattern[j] == text[i]:
                j += 1
            if j == pattern_length:
                positions_of_occurrences.append(str(i - pattern_length +
1))
                j = prefix_function[j - 1]
        return positions_of_occurrences


    # Данная функция предназначена для вывода индексов,
    # начиная с которых образец входит в текст.
    # Входные данные: индексы, начиная с которых образец входит в текст.
    def print_positions_of_occurrences(positions_of_occurrences):
        if len(positions_of_occurrences) != 0:
            print(','.join(positions_of_occurrences))
        else:
            print(-1)


    # Данная функция предназначена для ввода и получения образца и текста.
```

```
    # Выходные данные: считанные строки
    def get_read_text_and_pattern():
        pattern_input = input()
        text_input = input()
        return text_input, pattern_input


    # Данная функция запускает логику решения задачи: считывание образца
и текста,
    # поиск решения, вывод решения.
    def run():
        text, pattern = get_read_text_and_pattern()
        positions_of_occurrences = kmp_matcher(text, pattern)
        print_positions_of_occurrences(positions_of_occurrences)


    if __name__ == "__main__":
        run()
```

Название файла: lab4_2.py

```
    # Данная функция вычисляет префикс-функцию для строки, переданной на
вход.
    # Входные данные: строка, для которой нужно вычислить префикс-функцию.
    # Выходные данные: префикс-функция для строки, переданной на вход.
    def compute_prefix_function(pattern):
        pattern_length = len(pattern)
        prefix_function = [0] * pattern_length
        j = 0
        for i in range(1, pattern_length):
            while j > 0 and pattern[i] != pattern[j]:
                j = prefix_function[j - 1]
            if pattern[j] == pattern[i]:
                j += 1
            prefix_function[i] = j
        return prefix_function


    # Данный метод предназначен для поиска индекса начала строки pattern
в text
    # при том, что text -- это удвоенная первая введённая строка, а
pattern -- вторая
    # введённая строка.
    # Входные данные: первая введённая строка * 2, вторая строка.
    # Выходные данные: индекс начала второй введённой строки в первой,
умноженной на 2.
    def find_shift_position(text, pattern):
        pattern_length = len(pattern)
        text_length = len(text)
        shift_position = -1
        j = 0
        if pattern_length == text_length // 2:
            prefix_function = compute_prefix_function(pattern)
            for i in range(text_length):
                while j > 0 and pattern[j] != text[i]:
                    j = prefix_function[j - 1]
```

```
                if pattern[j] == text[i]:
                    j += 1
                if j == pattern_length:
                    shift_position = i - pattern_length + 1
                    break
        return shift_position


    # Данный метод предназначен для вывода индекса начала
    # второй введённой строки в первой, умноженной на 2.
    # Входные данные: индекс начала второй введённой строки в первой.
    def print_shift_position(shift_position):
        print(shift_position)


    # Данный метод предназначен для ввода и получения первой и второй
строки.
    # Выходные данные: считанные строки
    def get_read_text_and_pattern():
        text_input = input()
        pattern_input = input()
        return text_input * 2, pattern_input


    # Данный метод запускает логику решения задачи: считывание строк,
    # поиск решения, вывод решения.
    def run():
        text, pattern = get_read_text_and_pattern()
        shift_position = find_shift_position(text, pattern)
        print_shift_position(shift_position)


    if __name__ == "__main__":
        run()
```