МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

"ЛЭТИ" ИМ. В.И.УЛЬЯНОВА(ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе

по дисциплине «Построение и анализ алгоритмов»

Тема: Кнут-Моррис-Пратт.

Студент гр. 1304 _____ Сулименко М.А.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург

2023

Цель работы

Изучить алгоритм Кнута-Морриса-Пратта для поиска подстроки в строке.

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, разделенных запятой, если P не входит в T, то вывести 1

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 длинуи 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

Для выполнения обоих заданий был использован алгоритм Кнута-Морриса-Пратта. Алгоритм Кнута-Морриса-Пратта - алгоритм поиска подстроки в строке. Основан на сравнении символов в строке и подстроке, и использует заранее подготовленную информацию о совпадениях между префиксами и суффиксами подстроки, чтобы избежать ситуации, при которой при переносе указателя будет пропущена нужная подстрока в строке. Время работы – линейное, и зависит от количества введённых данных.

Основная идея КМП-алгоритма заключается обнаружении несоответствия между символами подстроки и строки, и сдвиге указателя не на 1 символ, а сразу на несколько, в зависимости от соответствующего значения в массиве префиксов.

Для этого алгоритм использует префикс-функцию, которая предварительно вычисляет длины наибольших собственных суффиксов, совпадающих с префиксами подстроки. Эта информация позволяет алгоритму переходить через неподходящие символы в строке, сдвигая подстроку на наибольшее возможное расстояние, и тем самым ускоряет процесс поиска.

Когда происходит несовпадение между символами подстроки и строки, алгоритм КМП использует значения префикс-функции, чтобы определить новое положение подстроки в строке без повторного сравнения символов, которые уже совпали.

Таким образом, КМП-алгоритм позволяет эффективно искать подстроку в строке, избегая избыточных сравнений символов, благодаря использованию префикс-функции и умному сдвигу подстроки на максимально возможное расстояние после несоответствия символов.

При написании кода (Приложение А и Приложение В) было использовано три функции:

- get_prefixes - префикс функция для вычисления длин наибольших суффиксов
- execute_kmp_algorithm – функция для выполнения алгоритма КМП для поиска всех вхождений подстроки в строку.
- execute_kmp_circle_algorithm – функция для определения, является ли первое слово циклическим сдвигом второго при помощи алгоритма КМП

Вывод

В рамках данной лабораторной работы внимательно изучен алгоритм Кнута-Морриса-Пратта, представляющий собой эффективный метод поиска подстроки в тексте. Поняты принцип работы и основные идеи алгоритма, такие как использование массива префиксов для определения мест возможных совпадений. Рассмотрено приложение этого алгоритма в задаче определения циклического сдвига одной строки относительно другой, что позволяет решать практические задачи, связанные с поиском подстрок в тексте, в том числе с учетом циклических сдвигов.

Для реализации алгоритма и его приложения написаны программы на языке Python. Каждая функция в программе документирована, чтобы облегчить понимание кода другими программистами и сделать программы более поддерживаемыми.В процессе работы обновлены знания синтаксиса Python, применены его основные конструкции, и освоено использование массивов префиксов для оптимизации алгоритма. Эта лабораторная работа позволила более глубоко разобраться в алгоритме Кнута-Морриса-Пратта и его приложении, а также попрактиковаться в разработке программ на языке Python.

Приложение А

Исходный код к заданию 1

Файл task1.cpp

```
# Метод для генерации массива префиксов слова
# Необходим для работы алгоритма КМП
def get_prefixes(word):
    jCount = 0
    iCount = 1
    prefixArray = [0] * len(word)

    while iCount < len(word):
        if word[jCount] == word[iCount]:
            prefixArray[iCount] = jCount + 1
            iCount += 1
            jCount += 1
        else:
            if jCount == 0:
                prefixArray[iCount] = 0
                iCount += 1
            else:
                jCount = prefixArray[jCount - 1]

    return prefixArray


# Метод, в котором выполняется алгоритм КМП
# и находятся все вхождения слова в текст
def execute_kmp_algorithm(word, text):

    wordLength = len(word)
    textLength = len(text)
    prefixArray = get_prefixes(word)
    occurrenceArray = []
    iCount = 0
    jCount = 0

    while iCount < textLength:
        if text[iCount] == word[jCount]:
            iCount += 1
            jCount += 1
            if jCount == wordLength:
                occurrenceArray.append(iCount - wordLength)
                if jCount > 0:
                    jCount = prefixArray[jCount - 1]
                else:
                    iCount += 1
        else:
            if jCount > 0:
                jCount = prefixArray[jCount - 1]
            else:
                iCount += 1
```

```
    if len(occurrenceArray) > 0:
        return occurrenceArray

    return -1


word = input()
text = input()
print(*execute_kmp_algorithm(word, text), sep=',')
```

Приложение В

Исходный код к заданию 2

Файл task2.cpp

```
# Метод для генерации массива префиксов слова
# Необходим для работы алгоритма КМП
def get_prefixes(word):
    jCount = 0
    iCount = 1
    prefixArray = [0] * len(word)

    while iCount < len(word):
        if word[jCount] == word[iCount]:
            prefixArray[iCount] = jCount + 1
            iCount += 1
            jCount += 1
        else:
            if jCount == 0:
                prefixArray[iCount] = 0
                iCount += 1
            else:
                jCount = prefixArray[jCount - 1]

    return prefixArray


# Метод, в котором выполняется алгоритм КМП
# для определения, является ли word циклическим
# сдвигом text
def execute_kmp_circle_algorithm(word, text):

    wordLength = len(word)
    textLength = len(text)
    prefixArray = get_prefixes(word)
    shiftIndex = -1
    iCount = 0
    jCount = 0

    if wordLength != textLength:
        return shiftIndex

    while iCount < textLength * 2:
        if text[iCount % textLength] == word[jCount]:
            iCount += 1
            jCount += 1
            if jCount == wordLength:
                shiftIndex = iCount - wordLength
                return shiftIndex
        else:
            if jCount > 0:
                jCount = prefixArray[jCount - 1]
            else:
```

```
                iCount += 1

    return shiftIndex


text = input()
word = input()
print(execute_kmp_circle_algorithm(word, text))
```