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: Поиск подстроки в строке. (КМП)

Студент гр. 3388 _____ Трунов Б.Г.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

Цель работы:

Изучить принцип работы алгоритма Кнута-Морриса-Пратта для нахождения подстрок в строке. Решить с его помощью задачи.

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

Sample Input:

ab

abab

Sample Output:

0,2

Задание 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.

Sample Input:

defabc

abcdef

Sample Output:

3

Реализация

Описание алгоритма Кнута-Морриса-Пратта:

Алгоритм Кнута-Морриса-Пратта устраняет недостатки наивного поиска, где при несовпадении символов происходит возврат к началу подстроки и сдвиг на 1 символ. Вместо этого KMP использует префикс-суффикс функцию (LPS), чтобы определить, на сколько символов можно безопасно сдвинуть шаблон без потери потенциальных совпадений.

Шаги алгоритма:

- **Проверка соответствия размеров:**
  - Если длина текста или длина шаблона равна 0, то возвращаем -1 в функции поиска и пустой массив в префикс-суффикс функции.
  - Если длина шаблона больше длины текста, то аналогично возвращаем -1 в функции поиска и пустой массив в префикс-суффикс функции.
    - Для данного шага используется *_validate_data(text : str, pattern : str) -> bool*
- **Вычисление префикс-суффикс функции:**
  - Создаётся массив длинной, равной длине шаблона, начальная инициализация нулями.
  - Для каждого символа шаблона (начиная с индекса 1) определяется длина наибольшего префикса, который также является суффиксом.
  - Далее используя два указателя *current_length* и *i* сравниваем символы шаблона и заполняем массив *LPS.*
    - Если *pattern[i]* равен *pattern[current_length]*, то *lps[i]=current_length*, и оба указателя увеличиваются на единицу.
    - Иначе два случая:
      - *current_length* не равен *0*, тогда *current_length=lps[current_length – 1].*

- *current_length* равен *0* тогда *lps[i] = 0*, указатель *i* увеличивается на единицу.

- Для данного шага используется *_makeLongestPrefixSuffix(self) -> list[uint]*.

- **Поиск шаблона в тексте:**
  - Используются два указателя: *i* (для текста) и *j* (для шаблона).
  - Последовательно сравниваются символы текста и шаблона:
    - Совпадение:
      - Оба указателя сдвигаются вправо.
    - Несовпадение:
      - Если *j > 0,* указатель *j* сдвигается на *lps[j – 1]*.
      - *Если j* равен *0,* указатель *i* сдвигается вправо на *1*.
    - При полном совпадении *(j* равен длине шаблона*)* фиксируется позиция вхождения шаблона, а значение *j* корректируется через *lps*.
    - Для данного шага используется *_search(self) -> list[uint]*.

Оценка сложности алгоритма:

Временная сложность

Вычисление префикс-суффикс функции:

- Проход по шаблону длиной m: $O(m)$.
- Итог: $O(m)$.

Поиск:

- Проход по тексту длиной n: $O(n)$.
- Итог: $O(n)$.

Общая: $O(m+n)$

Пространственная сложность

Префикс-суффикс функция:

- *lps*: $O(m)$ для массива длиной m (длина шаблона).

Поиск:

- *result_search*: $O(k)$ для хранения индексов вхождений шаблона в текст, где $k \leq n$.

Итого: $O(m + k)$

Тестирование

Таблица 1. Тестирование.

Входные данные	Выходные данные
Введите шаблон для поиска: 123 Введите текст для поиска: 123321123321123 Включить режим отладки? (y/n): n	Результат: Индексы вхождений шаблона в тексте: [0, 6, 12]
Введите шаблон для поиска: cppiscool Введите текст для поиска: cppiscoolcppiscoolcppisreallycoolcppiscoolcppcoolcoolcoocowcl Включить режим отладки? (y/n): n	Результат: Индексы вхождений шаблона в тексте: [0, 9, 33]
Введите шаблон для поиска: hello Введите текст для поиска: hello123321HELLOisYOOOOYAYAYA Включить режим отладки? (y/n): n	Результат: Индексы вхождений шаблона в тексте: [0]
Введите шаблон для поиска: Введите текст для поиска: 123 Включить режим отладки? (y/n): n	Результат: Совпадений не найдено
Введите шаблон для поиска: 123 Введите текст для поиска: Включить режим отладки? (y/n): n	Результат: Совпадений не найдено
Введите шаблон для поиска: 123 Введите текст для поиска: 12 Включить режим отладки? (y/n): n	Результат: Совпадений не найдено

Вывод

В ходе выполнения лабораторной работы был изучен и успешно реализован алгоритм Кнута-Морриса-Пратта (KMP), предназначенный для эффективного поиска подстроки в тексте. Основным преимуществом этого алгоритма перед наивным подходом является исключение избыточных сравнений символов за счёт использования префикс-суффикс-функции (*LPS*), что позволяет снизить вычислительную сложность до линейной — $O(n + m)$, где n и m — длины текста и шаблона соответственно. Это достигается благодаря тому, что KMP избегает возвратов по тексту, делая его особенно полезным для обработки потоковых данных или работы с большими объёмами информации.