**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: **Кнут-Моррис-Пратт**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Березовский М.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2025

**Цель работы:**

Изучить теоретические основы алгоритма Кнута-Морриса-Пратта.

**Задание:**

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P (∣P∣≤25000) и текста *T* (∣T∣≤5000000) найдите все вхождения *P* в *T*.

**Входные данные:**  
Вход:  
- Первая строка — P   
- Вторая строка — *T*  
Выход:  
индексы начал вхождений *P*  в  *T*, разделённые запятой; если *P* не входит в *T*, то вывести -1.

**Sample Input:**

ab

abab

**Sample Output:**

0,2

**Выполнение работы**

Описание алгоритма для решения задачи

Алгоритм Кнута–Морриса–Пратта предназначен для эффективного поиска всех вхождений образца (паттерна) в тексте. Он работает за время O(n + m), где n — длина текста, m — длина образца, так как каждый символ текста и образца просматривается не более фиксированного числа раз. В решении алгоритм разбит на две функции: compute\_prefix и kmp.

Алгоритм выполняется в два этапа:

**Построение префикс-функции (π-массива)**  
На первом этапе создаётся вспомогательный массив, который для каждой позиции i в образце хранит длину наибольшего собственного префикса подстроки pattern[0..i], который одновременно является её суффиксом.  
Этот массив нужен, чтобы в дальнейшем при возникновении несовпадений не возвращаться в самое начало образца, а "откатываться" на максимально возможную длину уже совпавшего префикса.  
Префикс-функция строится линейным проходом по шаблону с использованием двух индексов:

i - текущая позиция в образце, от 1 до m - 1

j - длина текущего совпавшего префикса  
На каждой итерации сравнивается pattern[i] и pattern[j]

При совпадении символов j увеличивается (j += 1), в pi[i] записывается j, и i сдвигается

При несовпадении, если j > 0, происходит откат j = pi[j - 1]

Если j == 0, в pi[i] записывается 0, и i увеличивается.  
Таким образом, для каждого символа образца вычисляется "граница", на которую можно безопасно откатиться при следующем несовпадении.

**Поиск образца в тексте**  
На втором этапе текст сканируется с начала до конца с двумя индексами:

i - текущая позиция в тексте

j - текущая позиция в образце (сколько символов уже совпало)  
Для каждого символа text[i] алгоритм:

Сравнивает его с pattern[j]

При совпадении увеличивает j

При несовпадении, если j > 0, откатывает j = pi[j - 1], иначе просто двигает i на один вперёд

Если после увеличения j достигает p\_len (т.е. совпало всё), значит найдено полное вхождение образца, и в список результатов записывается позиция начала совпадения: i - p\_len.  
После фиксации вхождения j снова сбрасывается на pi[j - 1], что позволяет находить перекрывающиеся вхождения без лишних сравнений.  
В конце, если список результатов пуст, возвращается строка -1, иначе индексы объединяются в строку через запятую.

**Оценка сложности алгоритма:**

**Оценка времени выполнения:**

Алгоритм строит префикс-функцию за время O(m), где m - длина шаблона. Далее выполняется проход по тексту длиной n, в котором каждый символ текста и шаблона обрабатывается не более одного раза благодаря откатам указателя по префикс-функции, поэтому время поиска составляет O(n).

Итоговая общая асимптотическая сложность алгоритма по времени равна O(n + m).

**Оценка использования памяти:**

Для работы алгоритма требуется хранить массив префикс-функции длиной m и список индексов найденных вхождений, который в худшем случае может содержать до k элементов. Дополнительно используется константное количество переменных для управления циклами и состоянием поиска. В сумме асимптотическая сложность по памяти составляет O(m + k).

**Тестирование**

Таблица 1. Тестирование.

|  |  |
| --- | --- |
| *Входные данные* | *Выходные данные* |
| ab  abab | 0,2 |
| лилила  лилилось лилилась | 9 |
| aa  aaaa | 0,1,2 |
| abracadabra  abradadabracatdabractadabrasdcjdnweoiheca | -1 |

**Вывод**

В ходе лабораторной работы был изучен и реализован алгоритм Кнута–Морриса–Пратта. В программе отдельно выделена функция вычисления префикс-массива π, а также функция поиска всех вхождений шаблона в текст. Код был подробно проверен на различных тестовых данных, что позволило убедиться в корректности вычислений и в том, что поиск действительно выполняется за время O(n + m). Полученная реализация демонстрирует, каким образом использование префикс-функции позволяет эффективно обрабатывать совпадения и откаты при сравнении строк, полностью избегая лишних проверок символов и обеспечивая стабильную и быструю работу даже при больших объёмах входных данных.