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 _____ Потапов Р.Ю.

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:

Алгоритм Кнута–Морриса–Пратта работает в два этапа и позволяет находить все вхождения одного образца в текст за время, пропорциональное сумме длин текста и образца. Сначала по самому образцу строится так называемая префикс-функция (π-массив): для каждой позиции i в образце вычисляется длина наибольшего собственного суффикса префикса образца, совпадающего с его префиксом. Это делается одним линейным проходом по символам, где переменная j хранит текущий размер совпавшего бордера и при каждом шаге либо растёт на единицу при совпадении символов, либо уменьшается до π[j–1] при несовпадении. Во втором этапе алгоритм сканирует текст слева направо, сравнивая текстовый символ с тем, на который указывает j, и аналогично при совпадении увеличивает j, а при несовпадении откатывает j в π[j–1], не возвращаясь в начало образца. Когда j достигает длины образца, фиксируется вхождение, его индекс сохраняется, и j сразу сбрасывается в π[j–1], чтобы продолжить поиск перекрывающихся вхождений без лишних сравнений. В результате каждый символ текста обрабатывается лишь константное число раз, и общая сложность поиска получается O(n+m).

Основные этапы работы:

1. Сначала строится префикс-функция (pi-массив) для шаблона: заводим pi[0]=0 и переменную j=0, затем для каждого i от 1 до m–1, пока j>0 и pattern[i] != pattern[j], возвращаем j=pi[j–1]; если после этого pattern[i]==pattern[j], делаем j=j+1 и записываем pi[i]=j.
2. Затем переходим к сканированию текста: заводим j=0 и для каждого i от 0 до n–1 повторяем ту же логику откатов — пока j>0 и text[i] !=pattern[j], пишем j=pi[j–1]; если text[i]==pattern[j], делаем j=j+1.

3. Каждый раз, когда j достигает m (длины шаблона), мы фиксируем вхождение с начала i–m+1, добавляем этот индекс в результат и сразу сбрасываем j=pi[j–1], чтобы продолжить поиск без лишних сравнений.

4. В конце, если список найденных позиций пуст, выводим –1, иначе возвращаем (или печатаем) все индексы вхождений.

Описание функций KMP:

1 pi_func(pattern) – строит массив префикс-функции для шаблона. Вычисляет для каждой позиции i длину наибольшего собственного суффикса префикса pattern[0…i], совпадающего с его префиксом. Заводит pi[0]=0 и счётчик j=0, затем для i от 1 до len(pattern)–1 при несовпадении несколько раз откатывает j=pi[j–1], а при совпадении увеличивает j и записывает pi[i]=j. В результате возвращает готовый массив pi.

2 kmp_search(text, pattern) – ищет все вхождения pattern в text с помощью KMP. Сначала получает pi через pi_func, затем проходит по тексту индексы i от 0 до len(text)–1, поддерживая j = число уже совпавших символов: при несовпадении откатывает j=pi[j–1], при совпадении увеличивает j. Когда j достигает len(pattern), фиксирует найденное вхождение на позиции i–len(pattern)+1, добавляет её в список результатов и сбрасывает j=pi[j–1] для поиска следующих.

3 main() – считывает шаблон и текст двумя input(), убирая окончательные ‘\n’. Если шаблон пуст или длиннее текста, сразу печатает –1. Иначе вызывает kmp_search, получает список индексов. Если он пуст, выводит –1; иначе печатает найденные индексы через запятую.

Оценка сложности алгоритма:

Временная сложность

Оценка временной сложности алгоритма Кнута–Морриса–Пратта сводится к двум этапам: сначала строится префикс-функция (pi-массив) для шаблона длины m, что делается за время O(m), поскольку каждый символ обрабатывается ровно один раз с небольшим числом откатов по ранее вычисленным значениям pi. Затем проводится проход по тексту длины n, и на каждом шаге мы либо совпадаем и двигаем указатели, либо при несовпадении одним присваиванием j = pi[j–1] быстро откатываемся, без возврата в начало шаблона. В итоге поиск занимает O(n). Суммарная временная сложность получается O(n + m). Пространственная сложность алгоритма — O(m) на хранение pi-массива, плюс константный дополнительный объём памяти на несколько индексов и счётчиков.

Тестирование

Таблица 1. Тестирование.

Входные данные	*Выходные данные*
abab ababab	0,2
nigger gg	2
dthrd jhefvjewrfverfvaverjfvjeravfbrejhvfjrecgvhbrjvxfj gfrebjgnetskbglrgkhctkjgbklhrtvsjfvrekhgkjrtnkhb jevrtbvteabkbxjhlfrsthgkbaeljvjkhrbfilr	-1
ghf ufreufuerjvhirebfjverfghjrstngkhlvejhfvrekbgklaer tjhverxjhgbkhetvjhferhgvfjkestbghkvtrjhfblrtghke 5jfwekhghlvftgbgiyevgvehjrfbkrtsbghtruyfgehghg iegrohgu;aelhiu5rgl	-1

Вывод

В ходе лабораторной работы мы изучили и реализовали алгоритм Кнута–Морриса–Пратта. Мы написали функцию вычисления префикс-массива π и функцию поиска вхождений, подробно отладили их работу на разных примерах и убедились, что алгоритм находит все совпадения за линейное время O(n+m). Полученная реализация демонстрирует, как с помощью π-массива удаётся жёстко избежать повторных сравнений и обеспечивает стабильную производительность даже на длинных строках.