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. 3388 _____ Шубин П.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

Цель работы.

Разобраться с принципом работы алгоритма Кнута-Морриса-Пратта для поиска подстрок в строке. Использовать его для решения задач: поиска шаблона в тексте и проверки, является ли одна строка циклическим сдвигом другой.

Задание.

Задача 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.

Задача 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.

Выполнение работы

Алгоритм Кнута – Морриса – Пратта предназначен для быстрого поиска всех вхождений шаблона в текст. Основная идея заключается в предварительном вычислении префикс-функции для образца, благодаря чему при несовпадении символов не возвращаются к самому началу строки, а перескакивают к позиции, основанной на уже найденных совпадениях. Это позволяет сократить общее число сравнений и значительно ускоряет обработку, что делает KMP особенно полезным при поиске подстрок в больших текстах.

Реализация

Для решения поставленной задачи были реализованы следующие функции:

1. computePrefixFunction(p string) []int – вычисляет массив π, где π[i] равен длине наибольшего собственного префикса строки p[0…i], совпадающего с её суффиксом, что используется для ускорения поиска.
2. kmpSearch(text, pattern string) []int – находит все стартовые позиции вхождений pattern в text, применяя префикс-функцию, чтобы при несовпадении перескакивать по уже найденным совпадениям.
3. *solveTask1(reader bufio.Reader) – читает из входного потока шаблон и текст, выполняет kmpSearch и выводит через запятую все найденные позиции либо -1, если совпадений нет.
4. *solveTask2(reader bufio.Reader) – принимает две строки A и B одинаковой длины, проверяет, можно ли получить B циклическим сдвигом A, и выводит минимальный сдвиг или -1.
5. main() – определяет номер задачи (1 или 2) из входа, создаёт bufio.Reader и перенаправляет выполнение на solveTask1 или solveTask2; в остальных случаях выдаёт ошибку.

Анализ сложности алгоритма

Временная сложность

- computePrefixFunction(p): один проход по длине шаблона m с внутрициклами, которые суммарно также делают не больше m шагов ⇒ O(m).
- kmpSearch(text, pattern):
  - построение префикс-функции: O(m)
  - сканирование текста длины n с переходами по π-массиву, которые в сумме не превышают n шагов ⇒ O(n)
  - итого для поиска: O(m + n)
- solveTask1: помимо O(m + n) для kmpSearch, все остальные действия (чтение, вывод) — O(n) в худшем случае (количество вхождений ≤ n), но асимптотически не превосходит O(m + n).
- solveTask2: объединение строк A+A даёт длину 2·m, затем поиск B в doubleA — O(m + 2m) = O(m). Выбор первого подходящего сдвига — ещё до m проверок ⇒ O(m). Итого: O(m).

Итоговая временная сложность

- Для задачи 1: O(m + n)
- Для задачи 2: O(m)

Пространственная сложность

- Хранение массива π длины m ⇒ O(m).
- В kmpSearch — динамический массив result длиной k (число вхождений), в худшем случае k ≤ n ⇒ O(n), но обычно считается дополнительным выходным буфером.
- В solveTask2 создаётся строка doubleA длины 2m ⇒ дополнительная память O(m).

Итоговая пространственная сложность

- Для задачи 1: O(m + k) ≃ O(m + n) (где k — число вхождений)

- Для задачи 2: O(m) (для π-массива и объединённой строки)

Тестирование:

Input	Output
aa aaaaa	0,1,2,3
xyz abcdefgh	-1

Таблица 1 — тестирование решения задания 1

Input	Output
AABCD CDAAB	3
ABAB BABA	1

Таблица 2 — тестирование решения задания 2

Выводы:

В рамках разработки был создан и проверен алгоритм поиска шаблона в тексте, использующий массив префиксных значений для ускорения сравнения. Включён подробный вывод диагностических сообщений, позволяющий легко отслеживать ход выполнения и выявлять ошибки. Решение надёжно обрабатывает входные данные, находит все позиции вхождений шаблона и выводит их в возрастающем порядке. Тестирование показало, что алгоритм сохраняет высокую скорость работы даже при больших объёмах текста.