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.

Алгоритм Кнута-Морриса-Пратта.

Студент гр. 3343 _____ Поддубный В.А.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2025

Цель работы

Ознакомиться с принципами работы алгоритма Кнута–Морриса–Пратта (КМП) и реализовать функцию для вычисления префикс-функции строки. На основе этой функции разработать:

1. Программу для поиска всех вхождений подстроки в строку;

2. Алгоритм для нахождения индекса начала вхождения одной строки в другую при условии циклического сдвига.

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.

№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

Префикс-функция

Функция `computePrefixFunction` предназначена для построения префиксного массива для заданной строки `p`. Этот массив используется в алгоритме Кнута-Морриса-Пратта (КМП) для ускоренного поиска подстрок.

Принцип работы:

Инициализация:

- Вычисляется длина строки `p`, создаётся массив `pi` длины `m`, заполненный нулями.
- Переменная `k` хранит длину текущего совпадающего префикса и суффикса.

Основной цикл:

- Цикл начинается с индекса `i = 1`, так как `pi[0]` всегда равен 0.
- На каждой итерации символ `p[i]` сравнивается с `p[k]`:
- Если символы совпадают, `k` увеличивается, и значение сохраняется в `pi[i]`.
- Если символы не совпадают и `k > 0`, происходит «откат» по префикс-функции: `k = pi[k-1]`.
- Значения промежуточных состояний выводятся с помощью логгера `Logger.log`.

Алгоритм КМП

Функция `kmpSearch(text, pattern)`

Реализует стандартный алгоритм Кнута-Морриса-Пратта для поиска всех вхождений строки `pattern` в строку `text`.

Принцип работы:

Инициализация:

- Объединяются строки `pattern + "#" + text` в переменную `combined` (разделитель `#` исключает ложные совпадения).
- Вычисляется префикс-функция для `combined`.

Поиск:

- Цикл по `i` от `pattern.length + 1` до конца строки:
- Если `pi[i] == pattern.length`, фиксируется совпадение.
- Индекс начала совпадения: `i - 2 * pattern.length`.
- Все найденные позиции сохраняются в список `result`.

Циклический поиск (поиск сдвига)

Функция `findCyclicShiftIndex(a, b)`

Реализует модифицированный КМП-алгоритм для поиска строки `a` в циклически сдвинутой строке `b`.

Принцип работы:

Инициализация:

- Сначала проверяется, равны ли длины строк `a` и `b`. Если нет — сдвиг невозможен.
- Пустые строки считаются совпадающими со сдвигом 0.
- Вычисляется префикс-функция `pi` для строки `a`.
- Индекс `j` отслеживает позицию в строке `a`.

Поиск в удвоенной строке `b + b`:

- Проход по `i` от `0` до `2n - 1`, символы берутся по индексу `i % n`.
- При несовпадении: `j` уменьшается согласно префикс-функции.
- При совпадении: `j` увеличивается.
- Если `j == n`, найдено полное совпадение строки `a`:
- Вычисляется смещение: `shift = (n - (i - n + 1)) % n`.
- Если совпадение находится в пределах первой копии `b`, возвращается найденный сдвиг.
- Иначе происходит откат `j` и поиск продолжается.

Оценка сложности алгоритмов

1. Стандартный КМП-поиск (`kmpSearch`)

- **Время:** `O(m + n)`, где `m` — длина подстроки, `n` — длина строки.
- **Память:** `O(m + n)`. Используется объединённая строка и массив префикс-функции.

2. Поиск циклического сдвига (`findCyclicShiftIndex`)

- **Время:** `O(m + 2n)`, где `m = n = длина строки a`.
- **Память:** `O(m)` — для хранения префикс-функции.

Тестирование

Результаты тестирования представлены в таблице 1.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	abc abcabcabc	0,3,6	Тест для первого задания (подстрока содержится в поисковой строке). Результат вычислен верно.
2.	xyz abxyzcxyzxyz	2,6,9	Тест для первого задания (подстрока не содержится в поисковой строке). Результат вычислен верно.
3.	abcde deabc	3	Тест для второго задания (строки являются циклическими сдвигами). Результат вычислен верно.
4.	qwerty asdfgh	-1	Тест для второго задания (строки не являются циклическими сдвигами). Результат вычислен верно.

Табл. 1. – Результаты тестирования

Выводы

Был детально изучен принцип работы алгоритма Кнута-Морриса-Пратта, что позволило разработать программы, корректно решающие поставленные задачи с использованием функции, вычисляющей максимальную длину префикса для каждого символа.

ПРИЛОЖЕНИЕ А

KMP.kt

```
object Logger {
    var enabled = true

    fun log(msg: String) {
        if (enabled) println(msg)
    }
}



fun computePrefixFunction(p: String): IntArray {
    val m = p.length
    val pi = IntArray(m)
    var k = 0
    Logger.log("Строим префикс-функцию для строки: \"$p\"")

    for (i in 1 until m) {
        Logger.log("i = $i, p[i] = '${p[i]}', k = $k")
        while (k > 0 && p[i] != p[k]) {
            Logger.log("  Несовпадение: '${p[i]}' != '${p[k]}', откат k ->
pi[${k - 1}] = ${pi[k - 1]}")
            k = pi[k - 1]
        }
        if (p[i] == p[k]) {
            k++
            Logger.log("    Совпадение:  '${p[i]}'  ==  '${p[k  -  1]}',
увеличиваем k -> $k")
        }
        pi[i] = k
        Logger.log("  pi[$i] = $k")
    }

    Logger.log("Итоговая префикс-функция: ${pi.joinToString()}")
    return pi
}



fun kmpSearch(text: String, pattern: String): List<Int> {
```

```
        val combined = "$pattern#$text"
        Logger.log("Запуск КМР поиска подстроки \"$pattern\" в строке \"$text\"")
        Logger.log("Комбинированная строка для префикс-функции: \"$combined\"")

        val pi = computePrefixFunction(combined)
        val result = mutableListOf<Int>()
        val m = pattern.length

        for (i in m + 1 until combined.length) {
            Logger.log("Проверка позиции i = $i, pi[i] = ${pi[i]}")
            if (pi[i] == m) {
                val idx = i - 2 * m
                Logger.log("  Подстрока найдена! Начало в позиции $idx")
                result.add(idx)
            }
        }

        return result
    }



    fun main() {
        val pattern = readln()
        val text = readln()

        val positions = kmpSearch(text, pattern)
        if (positions.isEmpty()) {
            println(-1)
            return
        }
        println("Результат: ${positions.joinToString(",")}")
    }

    ShiftWithKMP.kt
    fun findCyclicShiftIndex(a: String, b: String): Int {
        val n = a.length
        Logger.log("Поиск циклического сдвига: A = \"$a\", B = \"$b\"")
```

```
        if (n != b.length) {
            Logger.log("  Длины строк не равны — невозможно")
            return -1
        }
        if (n == 0) {
            Logger.log("  Обе строки пустые — сдвиг = 0")
            return 0
        }

        val pi = computePrefixFunction(a)
        var j = 0

        for (i in 0 until 2 * n) {
            val ch = b[i % n]
            Logger.log("i = $i, символ из B = '$ch', сравниваем с A[$j] =
'${if (j < n) a[j] else "-"}'")

            while (j > 0 && ch != a[j]) {
                Logger.log("   Несовпадение: '$ch' != '${a[j]}', откат j ->
pi[${j - 1}] = ${pi[j - 1]}")
                j = pi[j - 1]
            }

            if (ch == a[j]) {
                j++
                Logger.log("  Совпадение: '$ch' == '${a[j - 1]}', j -> $j")
            }

            if (j == n) {
                val idxInBB = i - n + 1
                Logger.log("  Полное совпадение найдено в позиции $idxInBB в
B+B")

                if (idxInBB < n) {
                    val shift = (n - idxInBB) % n
                    Logger.log("  Корректный сдвиг: $shift")
                    return shift
                }
```

```
                Logger.log("   Совпадение за пределами первой половины B+B, продолжаем")
                j = pi[j - 1]
            }
        }

        Logger.log("Совпадений не найдено")
        return -1
    }

    fun main() {
        Logger.enabled = true

        val a = readln()
        val b = readln()

        val shift = findCyclicShiftIndex(a, b)
        println("Результат: $shift")
    }
```

NaiveSearch.kt

```
    fun findCyclicShiftIndex(a: String, b: String): Int {
    val n = a.length
    Logger.log("Поиск циклического сдвига: A = \"$a\", B = \"$b\"")

    if (n != b.length) {
        Logger.log("  Длины строк не равны — невозможно")
        return -1
    }
    if (n == 0) {
        Logger.log("  Обе строки пустые — сдвиг = 0")
        return 0
    }

    val pi = computePrefixFunction(a)
    var j = 0

    for (i in 0 until 2 * n) {
        val ch = b[i % n]
```

```
        Logger.log("i = $i, символ из B = '$ch', сравниваем с A[$j] = '${if (j < n) a[j] else "-"}'")

        while (j > 0 && ch != a[j]) {
            Logger.log("  Несовпадение: '$ch' != '${a[j]}', откат j -> pi[${j - 1}] = ${pi[j - 1]}")
            j = pi[j - 1]
        }

        if (ch == a[j]) {
            j++
            Logger.log("  Совпадение: '$ch' == '${a[j - 1]}', j -> $j")
        }

        if (j == n) {
            val idxInBB = i - n + 1
            Logger.log("  Полное совпадение найдено в позиции $idxInBB в B+B")

            if (idxInBB < n) {
                val shift = (n - idxInBB) % n
                Logger.log("  Корректный сдвиг: $shift")
                return shift
            }

            Logger.log("    Совпадение за пределами первой половины B+B, продолжаем")
            j = pi[j - 1]
        }
    }

    Logger.log("Совпадений не найдено")
    return -1
}

fun main() {
    Logger.enabled = true

    val a = readln()
    val b = readln()
```

```
    val shift = findCyclicShiftIndex(a, b)
    println("Результат: $shift")
}
```