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

Студент гр. 3388 _____ Лексин М.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы.

Исследование и решение двух классических задач в области обработки строк с использованием эффективных алгоритмов. Первая задача состоит в разработке метода для быстрого обнаружения всех мест, где указанный образец встречается внутри большого текстового массива данных. Вторая задача направлена на определение специфического структурного отношения между двумя строками — является ли одна из них циклической перестановкой другой.

Задание.

Первое задание:

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P ($|P| \leq 25000$) и текста T ($|T| \leq 5000000$) 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

Второе задание:

Заданы две строки A ($|A| \leq 5000000$) и B ($|B| \leq 5000000$).

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 вляется 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

Описание алгоритма.

Задание 1.

Алгоритм КМП состоит из двух основных фаз: вычисление префикс-функции и поиск в тексте.

Фаза 1: вычисление префикс функции. На этом этапе анализируется только строка-шаблон P. Цель — построить вспомогательную структуру данных, называемую префикс-функцией.

Префикс функция Pi для шаблона P длиной m представляет собой массив целых чисел размером m. Значение Pi[i] определяется как длина наибольшего собственного префикса подстроки P[0…i] (то есть префикса, неравного самой подстроке P[0…i]), который одновременно является суффиксом этой же подстроки P[0…i].

Значение Pi[i] используется в фазе поиска для определения величины «умного» сдвига шаблона вдоль текста при возникновении несовпадения символов. Если несовпадение произошло при сравнении T[…] с P[j], то следующее сравнение будет происходить с P[Pi[j-1]], без необходимости возвращаться назад по тексту T.

Префикс-функция вычисляется за линейное время O(m) с использованием динамического программирования. Значение Pi[i] вычисляется на основе уже известного значения Pi[i-1]. Инициализируется Pi[0] = 0. Для i от 1 до m-1: сравнивается символ P[i] с символом P[k], где k = Pi[i-1]. Если они не совпадают, значение k рекурсивно заменяется на Pi[k-1] до тех пор, пока не найдется совпадение или k не станет 0. Если Pi[i] и Pi[k] совпадают, то Pi[i] устанавливается в k+1.

Фаза 2: поиск в тексте. На этом этапе происходит сканирование текста T слева направо с использованием вычисленной префикс-функции Pi. Используются два указателя: i на текущей позиции в тексте T и j для текущей позиции в шаблоне P. Указатель j также неявно представляет длину текущего префикса шаблона, который совпадает с подстрокой текста, заканчивающейся на позиции i-1. Процесс следующий:

1. Инициализация и итерация по тексту
2. Пока j > 0 и текущий символ текста T[i] не совпадает с символом шаблона P[j], происходит «сдвиг» шаблона путем прививания j = Pi[j-1]. Это позволяет продолжить сравнение с более коротким префиксом шаблона, который гарантировано совпадает с суффиксом уже просмотренного участка текста, без смещения указателя i назад.
3. Если T[i] == P[j], это означает, что совпадение текущих символов найдено, и мы продолжаем по шаблону: j = j + 1.
4. Если j достигает значения m, это означает, что найдено полное вхождения шаблона P. Оно заканчивается на текущей позиции i в тексте T. Начальный индекс этого вхождения равен i-m+1.

5. После обнаружения вхождения, чтобы найти возможные последующие вхождения, которые могут перекрываться с текущим, мы не сбрасываем j в 0, а снова используем префикс функцию: j = Pi[j-1]. Это эффективно сдвигает шаблон на максимально возможную величину, пропуская уже проверенные символы, и подготавливает к поиску следующего возможного совпадения.

Задание 2.

Решение основано на сведении задачи к поиску подстроки с использованием алгоритма КМП. Ключевой идеей является использование конкатенации строки B. С самой собой.

Если предварительные проверки пройдены (строки непустые, имеют одинаковую длину N, но не идентичны), создается новая строка T, равная конкатенации строки B с самой собой: T = B+B. Длина строки T будет 2N.

Вызывается алгоритм КМП для поиска первого вхождения строки A (шаблон) в строке T (текст). Алгоритм КМП выполняет:

1. Вычисление префикс функции Pi для A.

2. Сканирование строки T с использованием префикс-функции для эффективного поиска A.

Эта функция возвращает массив всех найденных начальных индексов. Для данной задачи нас интересует только самый первый индекс в этом массиве (если он не пуст). Этот индекс, обозначим его k, соответствует наименьшей неотрицательной величине сдвига, при которой A является подстрокой T.

Если КМП не нашел вхождений, то список найденных индексов пуст. Это означает, что A не является подстрокой T, и следовательно, A не является циклическим сдвигом B. Алгоритм возвращает -1.

Условие задачи требует «индекс начала строки B в A». Если сдвиг равен k, то начало строки B окажется в строке A на позиции с индексом N-k (за исключением случая k=0). Поскольку КМП работает с байтовыми представлениями строк (об этом будет написано в анализе сложности

алгоритма) и возвращает байтовый индекс k, то и длина N в формуле должна быть байтовой длиной. Алгоритм вычисляет и возвращает N_bytes - k.

Оценка сложности.

Для эффективности доступа к символам строк в Swift (где прямой доступ по целочисленному индексу к String не O(1)), строки P и T преобразуются в массивы байтов (Array<UInt8>) с помощью Array(string.utf8). Все сравнения и индексация в алгоритме производятся над этими байтовыми массивами, что обеспечивает O(1) доступ к элементам.

Задание 1.

Преобразование строки P в Array<UInt8> занимает O(m) времени.

Основной цикл вычисления префикс-функции Pi выполняется m-1 раз. Внутри цикла находится while, который уменьшает значение k. Переменная k увеличивается не более чем на 1 на каждой итерации внешнего цикла for. Суммарное увеличение k за все время работы не превышает m. Поскольку k неотрицательно, общее число выполнения операций уменьшения k не может превышать общее число увеличений k. Следовательно, суммарное время работы внутреннего цикла while по всем итерациям внешнего цикла for является O(m). Общее время работы фазы 1 составляет O(m) + O(m) = O(m).

Фаза 2. Преобразование строки T в Array<UInt8> занимает O(n) времени.

Основной цикл поиска выполняется n раз. Переменная j увеличивается не более чем на 1 на каждой итерации внешнего цикла for. Суммарное увеличение j не превышает n. Каждая итерация while уменьшает j. Общее число уменьшений j не может превысить общее число увеличений j. Следовательно, суммарное время работы внутреннего цикла while по всем итерациям внешнего цикла for является O(n). Общее время работы фазы 2 составляет O(n) + O(n) = O(n).

Таким образом, общая временная сложность алгоритма КМП для решения данной задачи составляет O(n+m).

Оценим объем памяти, используемой алгоритмом помимо памяти для хранения исходных строк P и T, которые требуют O(m) и O(n) памяти соответственно.

Массив префикс-функций хранит m целых чисел и требует O(m) памяти.

Строки копируются в массивы байтов, это требует дополнительно O(m) и O(n) памяти для хранения копий.

Объем памяти, необходимый для работы алгоритма КМП: pi (O(m)) + байковые копии (O(n+m)), что снова дает O(n+m).

Задание 2.

Предварительные проверки (сравнение, проверка на пустоту, сравнение строк) - O(N).

Конкатенация строк занимает время, пропорциональное суммарной длине результирующей строки - 2N => O(N).

Внутри КМП:

1. Преобразование двух срок в UInt8 - O(N).
2. Вычисление префикс функции - O(N).
3. Поиск в тексте: O(2N) = O(N).

Итого, общая сложность алгоритма для проверки циклического сдвига составляет O(N).

Оценим объем памяти, используемой алгоритмом, помимо O(N) для хранения исходных строк A и B.

Строка T требует O(N) памяти.

Внутри КМП:

1. textBytes = O(2N) = O(N) памяти
2. PatternBytes = O(N) памяти
3. Массив префикс-функции = O(N) памяти

Итого получаем O(N) памяти.

Тестирование.

Задание 1.

Табл.1

Входные данные	Выходные данные
aba ababa	0,2
aabaaab aabaaacaabaac	-1
test test string testing another test	0,12,28

Задание 2.

Табл.2

Входные данные	Выходные данные
defabc abcdef	3
cdeab abcde	3
hello hello	0

Выводы.

Исследованы и решены две задачи обработки строк. Для быстрого поиска образца в тексте реализован эффективный алгоритм КМП с линейной сложностью. Определение циклического сдвига строк выполнено путем сведения к поиску подстроки (также с КМП) в удвоенной строке, с корректным вычислением требуемого индекса отношения.

ПРИЛОЖЕНИЕ

Код.

Файл kmp.swift

```
import Foundation

private func computePi(patternBytes: [UInt8]) -> [Int] {
    let m = patternBytes.count
    guard m > 0 else { return [] }

    var pi = [Int](repeating: 0, count: m)
    var k = 0

    for i in 1..<m {
        while k > 0 && patternBytes[i] != patternBytes[k] {
            k = pi[k-1]
        }

        if patternBytes[i] == patternBytes[k] {
            k += 1
        }

        pi[i] = k
    }
    return pi
}

func findAllOccurencesKMP(pattern: String, text: String) -> [Int]
{
    let n = text.count
    let m = pattern.count
    guard m > 0 else {
        print("Warning: Pattern is empty.")
        return []
    }
    guard n >= m else { return [] }

    let textBytes = Array(text.utf8)
    let patternBytes = Array(pattern.utf8)

    let pi = computePi(patternBytes: patternBytes)

    var occurences: [Int] = []
    var j = 0

    for i in 0..<textBytes.count {
        while j > 0 && textBytes[i] != patternBytes[j] {
            j = pi[j-1]
```

```
        }
        if textBytes[i] == patternBytes[j] {
            j += 1
        }

        if j == m {
            let startIndex = i - m + 1
            occurences.append(startIndex)
            j = pi[j-1]
        }
    }
    return occurences
}

guard let pattern = readLine() else {
    fatalError("Не удалось прочитать строку шаблона P")
}

guard let text = readLine() else {
    fatalError("Не удалось прочитать строку текста T")
}

guard pattern.count <= 25000 else {
    fatalError("Длинна шаблона P превышает ограничение в 25000
символов")
}

let resultIndices = findAllOccurencesKMP(pattern: pattern, text:
text)

if resultIndices.isEmpty {
    print("-1")
} else {
    let resultString = resultIndices.map
{ String($0) }.joined(separator: ",")
    print(resultString)
}
```

Файл cycle.swift

```
import Foundation

private func computePi(patternBytes: [UInt8]) -> [Int] {
    let m = patternBytes.count
    guard m > 0 else { return [] }
    var pi = [Int](repeating: 0, count: m)
    var k = 0
    for i in 1..<m {
        while k > 0 && patternBytes[i] != patternBytes[k] {
            k = pi[k - 1]
        }
```

```
        if patternBytes[i] == patternBytes[k] {
            k += 1
        }
        pi[i] = k
    }
    return pi
}

private func findAllOccurrencesKMP(pattern: String, text: String)
-> [Int] {
    let n_bytes_text = text.utf8.count
    let m_bytes_pattern = pattern.utf8.count

    guard m_bytes_pattern > 0 else { return [] }
    guard n_bytes_text >= m_bytes_pattern else { return [] }

    let textBytes = Array(text.utf8)
    let patternBytes = Array(pattern.utf8)

    let pi = computePi(patternBytes: patternBytes)

    var occurrences: [Int] = []
    var j = 0

    for i in 0..<textBytes.count {
        while j > 0 && textBytes[i] != patternBytes[j] {
            j = pi[j - 1]
        }
        if textBytes[i] == patternBytes[j] {
            j += 1
        }
        if j == m_bytes_pattern {
            let startIndex = i - m_bytes_pattern + 1
            occurrences.append(startIndex)
            j = pi[j - 1]
        }
    }
    return occurrences
}

func checkCyclicShift(a: String, b: String) -> Int {
    guard a.count == b.count else {
        return -1
    }

    let n_bytes = a.utf8.count

    guard n_bytes > 0 else {
        return 0
    }
```

```
    if a == b {
        return 0
    }

    let concatenatedB = b + b
    let occurrences = findAllOccurrencesKMP(pattern: a, text:
concatenatedB)

    if let k = occurrences.first {
        return n_bytes - k
    } else {
        return -1
    }
}

guard let stringA = readLine() else {
    fatalError("Не удалось прочитать строку A")
}

guard let stringB = readLine() else {
    fatalError("Не удалось прочитать строку B")
}

let resultIndex = checkCyclicShift(a: stringA, b: stringB)
print(resultIndex)
```