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. 3343 _____ Никишин С.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы.

Изучить и реализовать алгоритм Кнута-Морриса-Пратта.

Задание 1.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P ($|P| \leq 15000$) 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
```

Задание 2.

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
```

Sample Output:

```
3
```

Описание алгоритма.

Алгоритм Кнута — Морриса — Пратта (КМП) — это эффективный алгоритм поиска подстроки в строке. Основное преимущество алгоритма КМП заключается в том, что он позволяет избежать повторной обработки уже проверенных символов, что делает его значительно быстрее, чем наивный метод поиска подстроки, особенно для больших текстов.

Алгоритм реализован в функции *kmpAlgorithm(pattern: str, text: str)*. Основные этапы алгоритма при решении задач 1 и 2 одинаковы. Отличались лишь входные и выходные данные.

Основные этапы алгоритма:

1. Обработка образца с помощью префикс функции:

Функция *computePrefixFunction(string: str)* вычисляет префикс-функцию для образца. Т.е мы получаем массив, где для каждого символа образца хранится длина наибольшего собственного префикса, который одновременно является суффиксом для подстроки, заканчивающейся на этом символе.

Принцип работы префикс-функции *computePrefixFunction(string: str)* описан в разделе “Описание функций и структур данных”.

2. Поиск образца в тексте:

Инициализируется массив *positions* для хранения индексов начала вхождений образца в тексте (используется при решении задания 1). Переменная *k* хранит длину текущего совпавшего префикса.

1. Проходим по тексту, начиная с первого символа.

2.1. Если символы не совпадают (pattern[k] != text[i]), уменьшаем k до значения pi[k - 1].

2.2. Если символы совпадают (pattern[k] == text[i]), увеличиваем k на 1.

2.3. Если *k* становится равным длине образца (*k == patternLength*), значит, найдено вхождение.

2.3.1 Если рассматривается задание 1, записываем индекс начала вхождения в массив positions и сдвигаем образец, используя префикс-функцию (k = pi[k - 1]). После окончания цикла, возвращаем массив positions.

2.3.2 Если рассматривается задание 2, просто возвращаем первый индекс начала вхождения.

Отличия при решении задач 1 и 2 незначительные:

1. Меняется вывод (массив индексов вхождений для 1 задачи и первый индекс вхождения для 2 задачи),

2. Для задачи 2 устанавливается условие, что образец и текст одинаковы по длине, и текст увеличивается в 2 раза для определения наличия циклического сдвига.

Алгоритм имеет сложность по памяти O(m + n) (Худший случай, если текст состоит из повторяющегося образца)

Функции имеет сложность по времени:

Функция	Сложность
computePrefixFunction(string: str)	O(m)
kmpAlgorithm(pattern: str, text: str)	O(m + n)

,где m – длина образца, n – длина текста

Исходные коды обоих программ указаны в приложении (приложение 1 и приложение 2 соответственно)

Описание функций и структур данных.

Структуры данных:

1. pi – список значений префикс-функции

Функции:

1. *def computePrefixFunction(string: str)*: функция, вычисляющая префикс функцию.

Принцип работы:

1. Инициализируется нулями массив pi длиной, равной длине образца.

2. Проходим по образцу, начиная со второго символа.

3.1. Если символы не совпадают (string[k] != string[i]), уменьшаем k до значения pi[k - 1].

3.2. Если символы совпадают (string[k] == string[i]), увеличиваем k на 1.

4. Записываем значение k в pi[i].

5. В конце получаем массив значений, вычисленных префикс-функцией.

Параметры:

string – строка, по которой вычисляется префикс функция (тип *str*).

Возвращает:

pi — массив значений префикс-функции (тип *list*).

Сложность по времени: O(m)

2. *def kmpAlgorithm(pattern: str, text: str)*: функция, реализующая алгоритм Кнута-Морриса-Пратта для поиска всех вхождений образца в тексте.

Параметры:

pattern – строка образца (тип *str*).

text – строка текста (тип *str*).

Возвращает:

1. В задаче 1, возвращает массив индексов начал вхождений (тип list).

2. В задаче 2, возвращает первый индекс начала вхождения (тип int).

Сложность по времени: O(m + n)

,где m – длина образца, n – длина текста

Тестирование.

Тестирование программ представлено на таблицах.

Тестирование задание 1

№ Теста	Входные данные	Выходные данные	Комментарий
1	ab abab	0,2	Верно
2	faf abba	-1	Верно
3	hah ahahahahahah	1,3,5,7,9	Верно

Тестирование задание 2

№ Теста	Входные данные	Выходные данные	Комментарий
1	defabc abcdef	3	Верно
2	abc abc	0	Верно
3	abc bac	-1	Верно

Примеры работ программ с выводом промежуточных результатов при введённом значении указаны в приложении (приложение 3 и приложение 4 для программ 1 и 2 соответственно)

Выводы.

Был реализован алгоритм Кнута-Морриса-Пратта (КМП). Реализованный алгоритм был использован для задач на поиск подстроки в строке и на проверку строк на циклический сдвиг. Полученные при тестировании ответы подтвердили корректность работы алгоритма.

Приложение

Приложение 1

```
IMPORT SYS

DEF COMPUTE_PREFIX_FUNCTION(PATTERN):
    """Вычисляет префикс-функцию для шаблона"""
    M = LEN(PATTERN)
    PREFIX = [0] * M
    K = 0

    FOR Q IN RANGE(1, M):
        WHILE K > 0 AND PATTERN[K] != PATTERN[Q]:
            K = PREFIX[K - 1]

        IF PATTERN[K] == PATTERN[Q]:
            K += 1

        PREFIX[Q] = K

    RETURN PREFIX

DEF KMP_SEARCH(TEXT, PATTERN):
    """Находит все вхождения шаблона в текст с помощью алгоритма КМП"""
    IF NOT PATTERN OR NOT TEXT:
        RETURN []

    N = LEN(TEXT)
    M = LEN(PATTERN)

    IF M > N:
        RETURN []

    PREFIX = COMPUTE_PREFIX_FUNCTION(PATTERN)
    OCCURRENCES = []

    J = 0  # индекс в шаблоне
    FOR I IN RANGE(N):  # индекс в тексте
        WHILE J > 0 AND TEXT[I] != PATTERN[J]:
            J = PREFIX[J - 1]

        IF TEXT[I] == PATTERN[J]:
            J += 1

        IF J == M:
            # Найдено вхождение
            OCCURRENCES.APPEND(I - M + 1)
            J = PREFIX[J - 1]  # продолжаем поиск следующих вхождений

    RETURN OCCURRENCES

DEF MAIN():
    PRINT("=== Алгоритм Кнута-Морриса-Пратта (КМР) ===")
    PRINT("Поиск всех вхождений шаблона P в тексте T")
    PRINT()

    PRINT("Введите шаблон P (длина ≤ 25000):")
    PATTERN = SYS.STDIN.READLINE().RSTRIP('\N')

    PRINT("Введите текст T (длина ≤ 5000000):")
    TEXT = SYS.STDIN.READLINE().RSTRIP('\N')
```

```
    print(f"\nПоиск шаблона '{pattern}' в тексте длиной {len(text)} символов...")

    # Поиск вхождений
    occurrences = kmp_search(text, pattern)

    # Формирование вывода
    print("\nРезультат поиска:")
    if occurrences:
        print(f"Найдено вхождений: {len(occurrences)}")
        result = ','.join(map(str, occurrences))
        print(f"Индексы начал вхождений: {result}")
    else:
        print("Вхождений не найдено: -1")

if __name__ == "__main__":
    main()
```

Приложение 2

```
import sys

def compute_lps(B):
    """Вычисляет префикс-функцию (LPS - Longest Prefix Suffix) для строки B"""
    m = len(B)
    if m == 0:
        return []

    lps = [0] * m
    j = 0

    for i in range(1, m):
        while j > 0 and B[i] != B[j]:
            j = lps[j - 1]

        if B[i] == B[j]:
            j += 1
            lps[i] = j
        else:
            lps[i] = 0

    return lps

def kmp_search(A, B, lps):
    """Ищет вхождение B в удвоенной строке A с помощью алгоритма КМП"""
    j = 0
    n = len(A)

    for i in range(2 * n):
        while j > 0 and A[i % n] != B[j]:
            j = lps[j - 1]

        if A[i % n] == B[j]:
            j += 1

        if j == len(B):
            return i - j + 1

    return -1

def main():
    # Надписи при вводе
    print("Введите строку A:")
    A = sys.stdin.readline().rstrip('\n')
```

```
    PRINT("Введите строку B:")
    B = SYS.STDIN.READLINE().RSTRIP('\n')

    # Проверка условий
    IF LEN(A) != LEN(B):
        PRINT("\nРезультат: -1 (строки разной длины)")
        RETURN

    IF LEN(A) == 0:
        PRINT("\nРезультат: 0 (пустые строки)")
        RETURN

    # Вычисление префикс-функции и поиск
    LPS = COMPUTE_LPS(B)
    RESULT = KMP_SEARCH(A, B, LPS)

    # Вывод результата
    IF RESULT != -1:
        PRINT(F"\nРезультат: {RESULT} (A является циклическим сдвигом B)")
        PRINT(F"Сдвиг: первые {RESULT} символов B становятся суффиксом A")
    ELSE:
        PRINT(F"\nРезультат: -1 (A не является циклическим сдвигом B)")

IF __NAME__ == "__MAIN__":
    MAIN()
```