

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

Ижболдин А.В.

Преподаватель

Жангиров Т. Р.

Санкт-Петербург

2025

Цель работы

Изучить и реализовать алгоритм Кнута–Морриса–Пратта (КМП) для поиска подстроки в строке. Получить практические навыки построения префикс-функции и анализа эффективности алгоритмов строкового поиска. Ознакомиться с преимуществами КМП по сравнению с наивными методами поиска.

Задание

Реализуйте алгоритм КМП и с его помощью для заданных шаблона PP ($|P| \leq 25000$ | $|P| \leq 25000$) и текста TT ($|T| \leq 5000000$ | $|T| \leq 5000000$) найдите все вхождения PP в TT .

Вход:

- Первая строка — PP
- Вторая строка — TT

Выход:

индексы начал вхождений PP в TT , разделённые запятой; если PP не входит в TT , то вывести -1.

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

Определить, является ли AA циклическим сдвигом BB (это значит, что AA и BB имеют одинаковую длину и AA состоит из суффикса BB , склеенного с префиксом BB). Например, $defabc$ является циклическим сдвигом $abcdef$.

Вход:

- Первая строка — AA
- Вторая строка — BB

Выход: Если AA является циклическим сдвигом BB , то индекс начала строки BB в AA ; иначе вывести -1. Если возможно несколько сдвигов, вывести первый индекс.

Выполнение работы

Функция **prefix_function** для вычисления массива префикс-функции проходит по всем позициям входной строки, начиная со второй, и для каждой вычисляет длину максимального собственного префикса, который одновременно является суффиксом префикса до этой позиции.

На каждом шаге она берёт значение, уже найденное для предыдущего индекса, и, пока текущий символ не совпадёт с символом в позиции, равной этому значению, откатывается к более короткому префиксу, чей размер хранится в массиве.

Если же символы совпадают, значение возрастает на единицу, и в итоговый массив заносится новая длина совпадающего префикса-суффикса.

Функция **kmp_optimized**. Сначала вычисляет префикс-функцию для образца, что позволяет при поиске по тексту не возвращаться к началу при каждом несовпадении, а «перескакивать» на ту позицию образца, на которой всё ещё может сохраняться совпавшая часть.

Затем он пробегает по всем символам текста, поддерживая указатель на текущую позицию в образце. При несовпадении возвращается к предыдущей возможной длине совпавшего фрагмента через массив префикс-функции, а при совпадении продвигает указатель вперёд.

Когда указатель достигает конца образца, это означает обнаружение полного вхождения, и его начальная позиция добавляется в список результатов.

Для задачи проверки циклического сдвига функция **cyclic** используется приём, допускающий представить бесконечное удвоение первой строки путём взятия её символов по модулю длины. В этой «развёрнутой» строке выполняется поиск второго слова с помощью того же механизма префикс-функции, которое рассчитывается один раз для второго слова.

При совпадении длина совпавшего фрагмента доходит до размера искомой строки, после чего вычисляется фактический сдвиг как остаток от деления пройденного расстояния на длину строки.

Это необходимо для того, чтобы не тратить память на создание новых строк и оптимизировать использование памяти.

Асимптотика алгоритма

Префикс функция.

Во первых: При инициализации $j = 0$ и в каждой итерации он может увеличиться макс. на 1. Т.е. макс. $j = n-1$

Во вторых: всегда до цикла $for\ j < i$ и в каждой итерации i, j увеличивается макс. на единицу, то для всех $i: p[i] < i$ ($p[i]$ массив префикс функции), то есть $while$ может только уменьшить значение j .

В третьих: $j \geq 0$, не может стать отрицательным.

Тогда получается, что общее количество уменьшений (выполнений цикла $while$) ограничено сверху общим увеличением j (макс $n-1$), т.е $O(n)$ увеличений + $O(n)$ уменьшений = $O(n)$. Память $O(n)$, для массива префикс функции.

КМП. Т.к. префикс. функцию можно вычислить за лин. сложность, то асимптотика КМП будет равно $O(n + p)$, где n - длина текста, а p - длина паттерна, по памяти $O(p)$ для массива префикс функции.

В функции нахождения цикл. смещения аналогично, т.к. там используется тот же самый КМП, но он находится прямо в этой функции, чтобы избежать лишнего копирования строк. Откуда временная сложность будет линейная $O(n)$ и память, будет также $O(n)$ так как новых строк не создается, максимум массив префикс. суммы.

Вывод

В ходе выполнения лабораторной работы были реализованы и протестированы алгоритмы, основанные на префикс-функции, включая алгоритм Кнута–Морриса–Пратта (КМП) для поиска подстроки в строке, а также алгоритм определения циклического сдвига строк. Реализация префикс-функции позволила добиться линейной временной сложности при сравнении строк, что делает данные методы эффективными при работе с большими объемами текста.

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
def prefix_function(text, debug=False):
    n = len(text)
    pref = [0] * n

    if debug:
        print("\n" + "="*50)
        print(f"АЛГОРИТМ ПРЕФИКС-ФУНКЦИИ для строки: '{text}'")
        print("="*50)
        print("Назначение: находит для каждой позиции длину
наибольшего собственного префикса, \nкоторый является суффиксом для
подстроки, заканчивающейся в этой позиции.")
        print("Позиция 0 всегда имеет значение 0")
        print(f"Символы строки: {' | '.join(text)}")
        print(f"Индексы: {' | '.join([str(i) for i in
range(n)])}")
        print("-"*50)

    j = 0
    for i in range(1, n):
        if debug:
            print(f"\nШаг {i}: Рассматриваем символ '{text[i]}' в
позиции {i}")
            print(f"Текущий префикс имеет длину j = {j}
('{text[:j]}')")

            while j > 0 and text[i] != text[j]:
                if debug:
                    print(f"    Несовпадение: '{text[i]}' !=
'{text[j]}'")
                    print(f"    Сокращаем префикс с {j} до {pref[j-1]}
(переходим по префикс-функции)")
                    j = pref[j - 1]

                if text[i] == text[j]:
                    j += 1
                    if debug:
                        print(f"    Совпадение: '{text[i]}' ==
'{text[j-1]}'")
                        print(f"    Увеличиваем длину префикса до j = {j}")

            pref[i] = j
            if debug:
                print(f"    Значение префикс-функции для позиции {i}:
pref[{i}] = {j}")
                current_pref = [str(x) for x in pref[:i+1]]
                while len(current_pref) < n:
                    current_pref.append(" ")
                print(f"    Текущий массив префикс-функции: [{' |
'.join(current_pref)}]")

        if debug:
```

```

        print("\nИтоговая префикс-функция:")
        print(f"Символы строки:      {' | '.join(text)}")
        print(f"Индексы:              {' | '.join([str(i) for i in
range(n)])}")
        print(f"Значения префикса: {' | '.join([str(x) for x in
pref])}")
        print("="*50)

    return pref

# abacaba
#

def kmp_optimized(pattern, text, debug=False):
    p = len(pattern)
    n = len(text)

    if debug:
        print("\n" + "="*50)
        print(f"АЛГОРИТМ КМП (КНУТА-МОРРИСА-ПРАТТА)")
        print("="*50)
        print(f"Шаблон для поиска: '{pattern}', длина = {p}")
        print(f"Текст для поиска:   '{text}', длина = {n}")
        print("-"*50)

    if p > n:
        if debug:
            print("Шаблон длиннее текста, поиск невозможен")
        return [-1]

    if debug:
        print("\nШаг 1: Вычисляем префикс-функцию для шаблона")

    pref = prefix_function(pattern, debug)

    matches = []
    j = 0

    if debug:
        print("\nШаг 2: Выполняем поиск шаблона в тексте")
        print("-"*50)

    for i in range(n):
        if debug:
            current_match = text[:i] + "[" + text[i] + "]" +
text[i+1:]
            ptr_pattern = " " * (i-j) + pattern if j > 0 else " "
            * i + pattern
            print(f"\nПозиция текста: {i}, символ '{text[i]}'")
            print(f"Текст:      {current_match}")
            if j > 0:
                print(f"Шаблон:   {ptr_pattern} (сдвиг = {i-j},
совпало {j} символов)")
            else:
                print(f"Шаблон:   {ptr_pattern}")

```



```

        while j > 0 and text[i] != pattern[j]:
            if debug:
                print(f"    Несовпадение: '{text[i]}' !=
'{{pattern[j]}}'")
                print(f"    Сокращаем префикс с {{j}} до {{pref[j-1]}}
(переходим по префикс-функции pref[{{j-1}}] = {{pref[j-1]}})")
                j = pref[j - 1]
            if debug and j > 0:
                ptr_pattern = " " * (i-j) + pattern
                print(f"    Новое положение шаблона: {{ptr_pattern}}
(сдвиг = {{i-j}})")

        if text[i] == pattern[j]:
            if debug:
                print(f"    Совпадение: '{text[i]}' ==
'{{pattern[j]}}'")
                j += 1
            if debug:
                print(f"    Увеличиваем счетчик совпадений: j =
{{j}}")

        if j == p:
            match_pos = i - p + 1
            if debug:
                print(f"\n    НАЙДЕНО ПОЛНОЕ СОВПАДЕНИЕ!")
                print(f"    Начальная позиция: {{match_pos}}")
                match_visual = text[:match_pos] + "[" +
text[match_pos:match_pos+p] + "]" + text[match_pos+p:]
                print(f"    {{match_visual}}")
                print(f"    Сокращаем префикс с {{j}} до {{pref[j-1]}}
(переходим по префикс-функции pref[{{j-1}}] = {{pref[j-1]}})")

            matches.append(match_pos)
            j = pref[j - 1]

        if debug:
            print("\n" + "="*50)
            if matches:
                print(f"Результат: найдено {{len(matches)}} совпадений
на позициях {{matches}}")
            else:
                print("Результат: совпадений не найдено")
            print("="*50)

    return matches if matches else [-1]

def cyclic(a, b, debug=False):
    n = len(a)
    m = len(b)

    if debug:
        print("\n" + "="*50)
        print(f"АЛГОРИТМ ПОИСКА ЦИКЛИЧЕСКОГО СДВИГА")
        print("="*50)
        print(f"Строка a: '{{a}}', длина = {{n}}")
        print(f"Строка b: '{{b}}', длина = {{m}}")

```

```

        print("Задача: найти такое k, что циклический сдвиг
строки a на k позиций даст строку b")
        print("-"*50)

        if n != m:
            if debug:
                print("Строки имеют разную длину, циклический сдвиг
не существует")
            return -1

        if debug:
            print("\nШаг 1: Вычисляем префикс-функцию для строки b")

        pref = prefix_function(b, debug)
        j = 0

        if debug:
            print("\nШаг 2: Ищем b в строке a+a (конкатенация a с
самой собой)")
            print(f"Строка a+a: '{a+a}'")
            print("-"*50)

        for i in range(n * 2):
            c = a[i % n]

            if debug:
                aa = a + a
                ptr = aa[:i] + "[" + aa[i] + "]" + aa[i+1:2*n]
                pattern_vis = " " * max(0, i-j) + b if j > 0 else " "

            * i + b

            print(f"\nИндекс i = {i} (в строке a это позиция {i %
n}))")

            print(f"a+a:      {ptr}")
            if j > 0:
                print(f"b:          {pattern_vis} (совпало {j}
СИМВОЛОВ)")
            else:
                print(f"b:          {pattern_vis}")

            while j > 0 and c != b[j]:
                if debug:
                    print(f" Несовпадение: '{c}' != '{b[j]}'")
                    print(f" Сокращаем префикс c {j} до {pref[j-1]}
(переходим по префикс-функции pref[{j-1}] = {pref[j-1]})")
                j = pref[j - 1]
                if debug and j > 0:
                    pattern_vis = " " * max(0, i-j) + b
                    # print(f" Новое положение b: {pattern_vis}")

            if c == b[j]:
                if debug:
                    print(f" Совпадение: '{c}' == '{b[j]}'")
                j += 1
                if debug:
                    print(f" Увеличиваем счетчик совпадений: l =
{j}")

```

```

        if j == m:
            result = (i - m + 1) % n
            if debug:
                print("\n" + "="*50)
                print(f"НАЙДЕН ЦИКЛИЧЕСКИЙ СДВИГ: k = {result}")
                print(f"a = '{a}'")
                print(f"Циклический сдвиг a на {result} позиций:
'{a[result:] + a[:result]]'")
                print(f"b = '{b}'")
                print("="*50)
            return result

    if debug:
        print("\n" + "="*50)
        print("Циклический сдвиг не найден")
        print("="*50)
    return -1

def find_rot(debug=False):
    if debug:
        print("\nПОИСК ЦИКЛИЧЕСКОГО СДВИГА")
        print("Введите две строки. Программа найдет, является ли
вторая строка циклическим сдвигом первой.")
        print("Строка 1:", end=" ")
        a = input()
        if debug:
            print("Строка 2:", end=" ")
        b = input()

    result = cyclic(a, b, debug)

def find_substr(debug=False):
    if debug:
        print("\nПОИСК ПОДСТРОКИ В ТЕКСТЕ (АЛГОРИТМ КМП)")
        print("Введите шаблон для поиска и текст, в котором нужно
искать.")
        print("Шаблон:", end=" ")

    pattern = input()

    if debug:
        print("Текст:", end=" ")

    text = input()

    matches = kmp_optimized(pattern, text, debug)

if __name__ == '__main__':
    debug_mode = True
    find_substr(debug=debug_mode)
    # find_rot(debug=debug_mode)

```