**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. 3344 |  | Тукалкин В.А. |
| Преподаватель |  | Фирсов М.А. |

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*∣≤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

**Пример входных данных**

ab

abab

**Соответствующие выходные данные**

0,2

**Задание 2**

Заданы две строки A (∣*A*∣≤5000000) и B (∣*B*∣≤5000000).

Определить, является ли А циклическим сдвигом В (это значит, что А и В имеют одинаковую длину и А состоит из суффикса В, склеенного с префиксом В). 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.

**Пример входных данных**

defabc

abcdef

**Соответствующие выходные данные**

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 kmp(pattern, text). 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 prefix(string) 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 prefix(string) 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(по памяти и сложности) |
| 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).

**Описание функций и структур данных**

Функции для первого задания:

* prefix(s: str) -> list[int]:

Вычисляет префикс-функцию для s.

Параметры:

s — строка-шаблон (str).

Возвращает:

Массив префикс-функции (list[int]).

* kmp(pattern: str, text: str) -> list[int]:

Выполняет поиск всех вхождений pattern в text с использованием алгоритма КМП.

Параметры:

pattern — строка-шаблон (str).

text — строка, в которой выполняется поиск (str).

Возвращает:

Список индексов вхождений pattern в text (list[int]).

Функции для второго задания:

* prefix(s: str) -> list[int]:

Вычисляет префикс-функцию для строки s.

Параметры:

s — строка-шаблон (str).

Возвращает:

Массив префикс-функции (list[int]).

* kmp(pattern: str, text: str) -> list[int]:

Реализует алгоритм КМП для поиска всех вхождений строки pattern в строку text.

Параметры:

pattern — строка-шаблон (str).

text — строка, в которой производится поиск (str).

Возвращает:

Список индексов первого вхождения шаблона в строку (list[int]), или -1, если вхождений не найдено.

* cyclicShift(A: str, B: str) -> int:

Определяет, является ли строка B циклическим сдвигом строки A.

Параметры:

A — первая строка (str).

B — вторая строка (str).

Возвращает:

Индекс первого вхождения строки A в строку B, если B является циклическим сдвигом A, или -1, если это не так.

**Тестирование**

Таблица 1 – Результаты тестирования для задания 1

|  |  |  |
| --- | --- | --- |
| № | Входные данные | Выходные данные |
| 1 | ab  ababcab | 0,3 |
| 2 | abc  xyz | -1 |
| 3 | aa  aaaaaa | 0,1,2,3,4 |
| 4 | abcabd  abcabeabcabcabd | 9 |

Таблица 2 – Результаты тестирования для задания 2

|  |  |  |
| --- | --- | --- |
| № | Входные данные | Выходные данные |
| 1 | abc  bca | 1 |
| 2 | abc  abcd | -1 |
| 3 | abcabc  abcabc | 0 |

Вывод с комментариями для случая P = “ab” T = “abab” задания 1:

Введите строку P: ab

Введите строку T: abab

>>> Построение префикс-функции <<<

Массив префикс-функции будет начинаться с 0

Итерация 1:

Текщий символ: s[1] = 'b'

Символы не совпали, k остается 0

Добавлено 0 на позицию 0

Префикс-функция: 0, 0

>>> Алгоритм Кнута-Морриса-Пратта <<<

Поиск abab в ab

Итерация 1:

Совпадение: text[i=0]='a' совпадает с pattern[j=0]='a', увеличиваем j до 1

Текущее состояние: j = 1

Итерация 2:

Совпадение: text[i=1]='b' совпадает с pattern[j=1]='b', увеличиваем j до 2

Найдено полное совпадение на позиции 0

Сдвигаем j на prefixTable[j-1] = 0

Текущее состояние: j = 0

Итерация 3:

Совпадение: text[i=2]='a' совпадает с pattern[j=0]='a', увеличиваем j до 1

Текущее состояние: j = 1

Итерация 4:

Совпадение: text[i=3]='b' совпадает с pattern[j=1]='b', увеличиваем j до 2

Найдено полное совпадение на позиции 2

Сдвигаем j на prefixTable[j-1] = 0

Текущее состояние: j = 0

>>> Результат: 0,2

Вывод с комментариями для случая P = “ab” T = “abab” задания 2:

Введите строку A: abcdef

Введите строку B: defabc

>>> Построение префикс-функции <<<

Массив префикс-функции будет начинаться с 0

Итерация 1:

Текщий символ: s[1] = 'e'

Символы не совпали, k остается 0

Добавлено 0 на позицию 0

Префикс-функция: 0, 0, 0, 0, 0, 0

Итерация 2:

Текщий символ: s[2] = 'f'

Символы не совпали, k остается 0

Добавлено 0 на позицию 1

Префикс-функция: 0, 0, 0, 0, 0, 0

Итерация 3:

Текщий символ: s[3] = 'a'

Символы не совпали, k остается 0

Добавлено 0 на позицию 2

Префикс-функция: 0, 0, 0, 0, 0, 0

Итерация 4:

Текщий символ: s[4] = 'b'

Символы не совпали, k остается 0

Добавлено 0 на позицию 3

Префикс-функция: 0, 0, 0, 0, 0, 0

Итерация 5:

Текщий символ: s[5] = 'c'

Символы не совпали, k остается 0

Добавлено 0 на позицию 4

Префикс-функция: 0, 0, 0, 0, 0, 0

>>> Алгоритм Кнута-Морриса-Пратта <<<

Поиск abcdefabcdef в defabc

Итерация 1:

Нет совпадений, j остаётся 0

Текущее состояние: j = 0

Итерация 2:

Нет совпадений, j остаётся 0

Текущее состояние: j = 0

Итерация 3:

Нет совпадений, j остаётся 0

Текущее состояние: j = 0

Итерация 4:

Совпадение: text[i=3]='d' совпадает с pattern[j=0]='d', увеличиваем j до 1

Текущее состояние: j = 1

Итерация 5:

Совпадение: text[i=4]='e' совпадает с pattern[j=1]='e', увеличиваем j до 2

Текущее состояние: j = 2

Итерация 6:

Совпадение: text[i=5]='f' совпадает с pattern[j=2]='f', увеличиваем j до 3

Текущее состояние: j = 3

Итерация 7:

Совпадение: text[i=6]='a' совпадает с pattern[j=3]='a', увеличиваем j до 4

Текущее состояние: j = 4

Итерация 8:

Совпадение: text[i=7]='b' совпадает с pattern[j=4]='b', увеличиваем j до 5

Текущее состояние: j = 5

Итерация 9:

Совпадение: text[i=8]='c' совпадает с pattern[j=5]='c', увеличиваем j до 6

Найдено полное совпадение на позиции 3

>>> Результат: 3

**Вывод**

Был реализован алгоритм Кнута-Морриса-Пратта (КМП) для поиска подстроки в строке и для проверки строк на циклический сдвиг.

**Приложение А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Имя файла: КМП с комментариями.py

def prefix(s):

"""

Функция высчитываем префикс-функцию

:type s: str

:rtype: list[int]

"""

print(">>> Построение префикс-функции <<<")

print("Массив префикс-функции будет начинаться с 0")

pi = [0] \* len(s)

for i in range(1, len(s)):

print(f"\nИтерация {i}:")

print(f"Текщий символ: s[{i}] = '{s[i]}'")

k = pi[i - 1] # Начинаем с предыдущего значения префикс-функции

# Поиск наибольшего префикса, который также является суффиксом

while k > 0 and s[k] != s[i]:

k = pi[k - 1]

# Если символы совпали, увеличиваем длину префикса

if s[k] == s[i]:

print(f'Символы совпали, увеличиваем k до {k + 1}')

k += 1

else:

print("Символы не совпали, k остается 0")

pi[i] = k

print(f"Добавлено {k} на позицию {i - 1}\nПрефикс-функция: ", end='')

print(\*pi, sep=', ')

return pi

def kmp(pattern, text):

"""

Флгоритм Кнута-Морриса-Пратта, ищет подстроку text в строке pattern

:type pattern: str

:type text: str

:rtype: list[int]

"""

arr = []

prefixTable = prefix(pattern)

j = 0 # индекс для pattern

print("\n\n>>> Алгоритм Кнута-Морриса-Пратта <<<")

print(f"Поиск {text} в {pattern}")

for i in range(len(text)):

print(f"\nИтерация {i + 1}: ")

# Корректировка j при несовпадении

while j > 0 and text[i] != pattern[j]:

print(f"Несовпадение: text[i={i}]='{text[i]}' vs pattern[j={j}]='{pattern[j]}'")

print(f"Сдвигаем j={j} -> j=prefixTable[j-1]={prefixTable[j - 1]}")

j = prefixTable[j - 1]

# Совпадение символов

if text[i] == pattern[j]:

print(

f"Совпадение: text[i={i}]='{text[i]}' совпадает с pattern[j={j}]='{pattern[j]}', увеличиваем j до {j + 1}")

j += 1

else:

print(f'Нет совпадений, j остаётся {j}')

# Полное совпадение паттерна

if j == len(pattern):

print(f"Найдено полное совпадение на позиции {i - len(pattern) + 1}")

arr.append(i - len(pattern) + 1)

print(f'Сдвигаем j на prefixTable[j-1] = {prefixTable[j - 1]}')

j = prefixTable[j - 1]

print(f"Текущее состояние: j = {j}")

return arr

def test():

P = 'ababababa'

T = 'ababababababababa'

print("\n\n>>> Результат:", kmp(P, T))

#test()

if \_\_name\_\_ == "\_\_main\_\_":

P = input("Введите строку P: ").strip()

T = input("Введите строку T: ").strip()

if len(P) > len(T) or not P:

print(">>> Результат: ", -1)

else:

arr = kmp(P, T)

print("\n\n>>> Результат:", ','.join([str(x) for x in arr]) if len(arr) != 0 else -1)

Имя файла: Поиск цикла с комментариями.py

def prefix(s):

"""

Функция высчитываем префикс-функцию

:type s: str

:rtype: list[int]

"""

print(">>> Построение префикс-функции <<<")

print("Массив префикс-функции будет начинаться с 0")

pi = [0] \* len(s)

for i in range(1, len(s)):

print(f"\nИтерация {i}:")

print(f"Текщий символ: s[{i}] = '{s[i]}'")

k = pi[i - 1] # Начинаем с предыдущего значения префикс-функции

# Поиск наибольшего префикса, который также является суффиксом

while k > 0 and s[k] != s[i]:

k = pi[k - 1]

# Если символы совпали, увеличиваем длину префикса

if s[k] == s[i]:

print(f'Символы совпали, увеличиваем k до {k + 1}')

k += 1

else:

print("Символы не совпали, k остается 0")

pi[i] = k

print(f"Добавлено {k} на позицию {i - 1}\nПрефикс-функция: ", end='')

print(\*pi, sep=', ')

return pi

def kmp(pattern, text):

"""

Алгоритм Кнута-Морриса-Пратта, ищет подстроку text в строке pattern

:type pattern: str

:type text: str

:rtype: list[int]

"""

lengthPattern = len(pattern)

prefixTable = prefix(pattern)

j = 0 # индекс для pattern

print("\n\n>>> Алгоритм Кнута-Морриса-Пратта <<<")

print(f"Поиск {text} в {pattern}")

for i in range(len(text)):

print(f"\nИтерация {i + 1}: ")

# Корректировка j при несовпадении

while j > 0 and text[i] != pattern[j]:

print(f"Несовпадение: text[i={i}]='{text[i]}' vs pattern[j={j}]='{pattern[j]}'")

print(f"Сдвигаем j={j} -> j=prefixTable[j-1]={prefixTable[j - 1]}")

j = prefixTable[j - 1]

# Совпадение символов

if text[i] == pattern[j]:

print(

f"Совпадение: text[i={i}]='{text[i]}' совпадает с pattern[j={j}]='{pattern[j]}', увеличиваем j до {j + 1}")

j += 1

else:

print(f'Нет совпадений, j остаётся {j}')

# Полное совпадение паттерна

if j == lengthPattern:

print(f"Найдено полное совпадение на позиции {i - lengthPattern + 1}")

return i - lengthPattern + 1

print(f"Текущее состояние: j = {j}")

return -1

def cyclicShift(A, B):

if len(A) != len(B):

print("\nСтроки разной длины")

return -1

index = kmp(B, A + A)

return index

def test():

A = 'defabc'

B = 'abcdef'

print("\n\n>>> Результат:", cyclicShift(A, B))

#test()

if \_\_name\_\_ == "\_\_main\_\_":

A = input("Введите строку A: ").strip()

B = input("Введите строку B: ").strip()

print("\n>>> Результат:", cyclicShift(A, B))