**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. (КМП)

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Кулач Д.В. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2025

**Цель работы:**

Изучить принцип работы алгоритма Кнута-Морриса-Пратта для нахождения подстрок в строке. Решить с его помощью задачи.

**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

**Sample Input:**

ab

abab

**Sample Output:**

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 является 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

**Реализация**

Описание алгоритма Кнута-Морриса-Пратта:

Алгоритм Кнута-Морриса-Пратта (KMP) разработан для быстрого поиска всех мест в тексте T, где встречается заданная подстрока P. Он повышает эффективность поиска, минимизируя лишние сравнения символов благодаря использованию префикс-функции, которая позволяет пропускать уже обработанные участки текста при несовпадении.

**Шаги алгоритма**

1. **Проверка размеров**: сначала проверяются длины P и T. Если P пустая или её длина превышает длину T, возвращается пустой список.
2. **Вычисление префикс-функции π для P**:
   * Для каждого символа P[i] (где i от 1 до m−1, а m — длина P) вычисляется значение π[i].
   * Если символы P[k] и P[i] не совпадают, значение k уменьшается с использованием π[k−1], пока не будет найдено совпадение или k не станет равным 0.
   * Если P[k] совпадает с P[i], значение k увеличивается.
   * Итоговое значение π[i] равно k.
3. **Поиск вхождений P в T**:
   * Проходим по тексту T с индексом i, отслеживая количество совпавших символов q (из P).
   * Если P[q] не равно T[i], q уменьшается с использованием π[q−1].
   * Если P[q] равно T[i], q увеличивается.
   * Когда q достигает m (длина P), это означает полное совпадение, и позиция i−m+1 добавляется в список результатов. Затем q уменьшается с использованием π[q−1] для продолжения поиска.

Описание функций и структур:

• list[int] compute\_prefix\_function\_verbose(const str& P) – функция, которая вычисляет префикс-функцию для шаблона P. Возвращает список, в котором каждый элемент pi[i] показывает длину наибольшего префикса, совпадающего с суффиксом подстроки P[0..i]. В процессе работы выводит подробные комментарии (отладочную информацию) по каждому шагу.

• list[int] kmp\_search\_verbose(const str& P, const str& T) – функция, которая ищет все вхождения строки P в строке T с использованием алгоритма Кнута–Морриса–Пратта (KMP). Возвращает список индексов, с которых начинаются вхождения шаблона в тексте. Пошагово выводит ход поиска.

• int kmp\_search\_shift\_verbose(const str& P, const str& T) – функция, которая проверяет, является ли строка P циклическим сдвигом строки T. Возвращает индекс, начиная с которого строка T при сдвиге совпадает с P, или -1, если сдвига не существует. Использует KMP и выводит подробные шаги.

• void main() – основная функция, которая запрашивает у пользователя выбор задачи:

* если введено 1, выполняется поиск подстроки P в тексте T с помощью kmp\_search\_verbose;
* если введено 2, проверяется, является ли строка B циклическим сдвигом строки A с помощью kmp\_search\_shift\_verbose;
* если введён некорректный номер задачи, выводится соответствующее сообщение.

Оценка сложности алгоритма:

**Временная сложность**

Вычисление префикс-функции:

* + Проход по P длиной m: O(m).
  + Итог: O(m).

Поиск:

* + Проход по T длиной n: O(n).
  + Внутренний цикл while уменьшает q по pi, но общее число шагов равно O(n), так как каждое уменьшение компенсируется предыдущим увеличением.
  + Добавление позиций: O(z), где z — число вхождений, но z≤n.
  + Итог: O(n).

Общая*:* O(m+n)

**Пространственная сложность**

Префикс-функция:

* + pi: O(m) для массива длиной m.

Поиск:

* + occurrences: O(z) для хранения позиций, где z≤n.

Итого: O(m + z)

**Тестирование**

Таблица 1. Тестирование.

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| alksdmlaksmdlkamdlksamdlkamdlkamdaldmlakdmlakdmalkmd  amdaldmlakdmlakdmalkmdalksdmlaksmdlkamdlksamdlkamdlk | 30 |
| aba  ababababa | 0,2,4,6 |
| bab  bbaababbabaabab | 4,7,12 |

**Вывод**

В ходе лабораторной работы были написаны программы с использованием алгоритма Ахо-Корасика. Также дополнительно было сделано: подсчёт вершин и определение пересечений.

**Исходный код программы см. в ПРИЛОЖЕНИИ А.**

**ПРИЛОЖЕНИЕ А.**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

main.py

def compute\_prefix\_function\_verbose(P):  
 print("Построение pi-функции:")  
 n = len(P)  
 pi = [0] \* n  
 j = 0  
 for i in range(1, n):  
 while j > 0 and P[i] != P[j]:  
 print(f"q={i}: {P[i]}!={P[j]} и k={j}; переход к pi[{j - 1}]={pi[j - 1]}")  
 j = pi[j - 1]  
 if P[i] == P[j]:  
 j += 1  
 print(f"q={i}: {P[i]}=={P[j-1]}; pi[{i}]={j}")  
 else:  
 print(f"q={i}: {P[i]}!={P[j]} и k=0; pi[{i}]=0")  
 pi[i] = j  
 print(f"Результат pi: {pi}\n")  
 return pi  
  
def kmp\_search\_verbose(P, T):  
 print("Поиск вхождений:")  
 pi = compute\_prefix\_function\_verbose(P)  
 result = []  
 j = 0  
 for i in range(len(T)):  
 while j > 0 and T[i] != P[j]:  
 print(f"i={i}: {T[i]}!={P[j]}, q={j} -> q={pi[j - 1]}")  
 j = pi[j - 1]  
 if T[i] == P[j]:  
 j += 1  
 print(f"i={i}: {T[i]} совпало, q-> {j}")  
 else:  
 print(f"i={i}: {T[i]}!={P[j]}, q={j}")  
 if j == len(P):  
 print(f"Найдено вхождение с {i - len(P) + 1}")  
 result.append(i - len(P) + 1)  
 j = pi[j - 1]  
 return result  
  
def kmp\_search\_shift\_verbose(P, T):  
 print("Проверка циклического сдвига:")  
 pi = compute\_prefix\_function\_verbose(P)  
 j = 0  
 for i in range(len(T)):  
 while j > 0 and T[i] != P[j]:  
 print(f"i={i}: {T[i]}!={P[j]}, q={j} -> q={pi[j - 1]}")  
 j = pi[j - 1]  
 if T[i] == P[j]:  
 j += 1  
 print(f"i={i}: {T[i]} совпало, q-> {j}")  
 else:  
 print(f"i={i}: {T[i]}!={P[j]}, q={j}")  
 if j == len(P):  
 print(f"Найден сдвиг {i - len(P) + 1}")  
 return i - len(P) + 1  
 return -1  
  
def main():  
 task = input("").strip()  
  
 if task == "1":  
 P = input().strip()  
 T = input().strip()  
 matches = kmp\_search\_verbose(P, T)  
 if matches:  
 print(",".join(map(str, matches)))  
 else:  
 print(-1)  
  
 elif task == "2":  
 A = input().strip()  
 B = input().strip()  
 if len(A) != len(B):  
 print(-1)  
 else:  
 A2 = A + A  
 result = kmp\_search\_shift\_verbose(B, A2)  
 print(result)  
  
 else:  
 print("Некорректный выбор. Введите 1 или 2.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()