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. 9383 _____ Хотяков Е.П.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург

2021

Цель работы.

Познакомиться с алгоритмом Кнут-Моррис-Пратта, реализовать алгоритм на одном из языков программирования.

Задание.

1. Поиск индексов вхождения подстроки в строке

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

2. Проверка строки на циклический сдвиг

Заданы две строки A (|A|≤5000000) и B (|B|≤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 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

Основные теоретические положения.

Подстрока — это непустая связная часть строки.

Пусть $L = c_0 \ldots c_{n-1}$ – строка длины n.

Любая строка $S = c_i \ldots c_j$, где $0<=i<=j<=n-1$, является *подстрокой* L длины $j-i+1$.

Если i=0, то S называется *префиксом* L длины $j+1$.

Если $j=n-1$, то S – *суффикс* L длины $j-i+1$.

Префикс-функция от строки S и позиции i в ней — длина k наибольшего собственного (не равного всей подстроке) префикса подстроки S[1...i], который одновременно является суффиксом этой подстроки.

Алгоритм Кнута—Морриса—Пратта (КМП-алгоритм) — эффективный алгоритм, осуществляющий поиск подстроки в строке.

Описание алгоритма.

1. Префикс-функция.

Элементу с индексом 0 присваиваем значение 0, так как строка длины 1 элемента не рассматривается. Переменная i отвечает за индекс суффикса, j за

длину рассматриваемого образца, то есть текущее значение префикс-функции. Далее действует по следующим правилам:

- Если символы не равны и j отличен от 0, то присваиваем ему значение префикс-функции от j-1. Если же j равен 0, то присваиваем нулевое значение префикс-функции текущего элемента i.

- Если символы равны, то увеличиваем j на единицу.

- Алгоритм заканчивается, когда i дойдет до конца строки.

2. Алгоритм КМП.

Дан текст t и строка s, требуется найти и вывести позиции всех вхождений строки s в текст t.

Обозначим для удобства через n длину строки s, а через m — длину текста t.

Образуем строку s + | + t, где символ | — это разделитель, который не должен нигде более встречаться. Посчитаем для этой строки префикс-функцию. Теперь рассмотрим её значения, кроме первых n+1 (которые, как видно, относятся к строке s и разделителю). По определению, значение pi[i] показывает наидлиннейшую длину подстроки, оканчивающейся в позиции i и совпадающего с префиксом. Но в нашем случае это pi[i] — фактически длина наибольшего блока совпадения со строкой s и оканчивающегося в позиции i. Больше, чем n, эта длина быть не может — за счёт разделителя. А вот равенство pi[i] = n (там, где оно достигается), означает, что в позиции i оканчивается искомое вхождение строки s (только не надо забывать, что все позиции отсчитываются в склеенной строке s + | + t).

Таким образом, если в какой-то позиции i оказалось pi[i] = n, то в позиции i - (n + 1) - n + 1 = i - 2n строки t начинается очередное вхождение строки s в строку t.

Сложность этого алгоритма по операциям будет равна O(m+n), где n – длина подстроки, а m – длина строки, сложность построения префикс-функции

O(n), а сложность сравнений в алгоритме O(m), так как зависит только от длины строки, по которой мы ходим.

3. Проверка на циклический сдвиг.

Проверка осуществляется за счет все того же алгоритма КМП. В качестве подстроки мы отправляем строку – оригинал, а в качестве текста – удвоенную строку, которая нуждается в проверке на цикличность. Потом мы просто ищем вхождение подстроки в новую строку, если она была найдена, то строка является циклическим сдвигом.

Описание функций и структур данных.

std::vector<int> PrefixFunction(std::string &KMP_string)

Функция для вычисления префикс-функции. Принимает строку и возвращает вектор значение префикс-функций.

void KMP(std::string &P, std::string &T)

Функция реализация алгоритма КМП. Принимает на вход подстроку и строку. Также в функции сразу выводится результат работы алгоритма

void KMP_Circle(std::string &P, std::string &T)

Функция для проверки на циклический сдвиг. Принимает основную строку и строку для проверки.

Тестирование.

1. Поиск индексов вхождения подстроки в строку

Входные данные:

ab

abab

Выходные данные:

0,2

Входные данные:

efefe

qqefefeqqefefe

Выходные данные:

2,9

Входные данные:

abca

qqqq

Выходные данные:

-1

2. Проверка на циклический сдвиг

Входные данные:

defabc

abcdef

Выходные данные:

3

Входные данные:

abfdec

abcdef

Выходные данные:

-1

Входные данные:

ababab

bababa

Выходные данные:

1

Выводы.

В ходе выполнения лабораторной работы был изучен и реализован алгоритм Кнута—Морриса—Пратта, который находит индексы вхождений подстроки в строку, для чего была реализована функция вычисления префикс-функции. Также был реализован алгоритм проверки строки на циклический сдвиг другой строки на основе алгоритма КМП.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл kmp.cpp:

```
#include <iostream>
#include <vector>
#include <string>
std::vector<int> PrefixFunction(std::string &KMP_string)
{
    int length = KMP_string.length();
    std::vector<int> PrefixArray(length);
    for (int i = 1; i < length; i++)
    {
        int k = PrefixArray[i - 1];
        while (k > 0 && KMP_string[i] != KMP_string[k])
            k = PrefixArray[k - 1];
        if (KMP_string[i] == KMP_string[k])
            ++k;
        PrefixArray[i] = k;
    }
    return PrefixArray;
}


void KMP(std::string &P, std::string &T)
{
    std::string KMP_string = P + '|' + T;
    std::string result = "";
    std::vector<int> PrefixArray = PrefixFunction(KMP_string);
```

```
    int length_P = P.length();
    for (int i = 0; i < KMP_string.length(); i++)
    {
        if (PrefixArray[i] == length_P)
            result += std::to_string(i - 2 * length_P) + ",";
    }
    if (result.empty())
    {
        std::cout << "-1";
        return;
    }
    result.pop_back(); //убираем последюю запятую
    std::cout << result;
}

int main()
{
    std::string P, T;
    std::cin >> P;
    std::cin >> T;
    KMP(P, T);
    return 0;
}
```

Файл KMP_Circle.cpp:

```
#include <iostream>
#include <vector>
#include <string>
```

```
std::vector<int> PrefixFunction(std::string &KMP_string)
{
    int length = KMP_string.length();
    std::vector<int> PrefixArray(length);
    for (int i = 1; i < length; i++)
    {
        int k = PrefixArray[i - 1];
        while (k > 0 && KMP_string[i] != KMP_string[k])
            k = PrefixArray[k - 1];
        if (KMP_string[i] == KMP_string[k])
            ++k;
        PrefixArray[i] = k;
    }
    return PrefixArray;
}

void KMP_Circle(std::string &P, std::string &T)
{
    if (P.empty() || T.empty())
    {
        std::cout << "-1";
        return;
    }
    int length_P = P.length();
    P += '|' + T;
    std::vector<int> PrefixArray = PrefixFunction(P);
    for (int i = 0; i < P.length(); i++)
```

```
    {
        if (PrefixArray[i] == length_P)
        {
            std::cout << std::to_string(i - 2 * length_P);
            return;
        }
    }
    std::cout << "-1";
}

int main()
{
    std::string P, T;
    std::cin >> P;
    std::cin >> T;
    if (P.length() != T.length())
    {
        std::cout << "-1";
        return 0;
    }
    P += P;
    KMP_Circle(T, P);
    return 0;
}
```