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: Алгоритм Кнутта-Морриса-Пратта
Вариант 2

Студент гр. 8303 _____ Удод М.Н.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Изучение алгоритма Кнутта-Морриса-Пратта для поиска подстроки в строке.

Задание 1.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P ($|P|\leq1500$) и текста T ($|T|\leq5000000$) 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 ($|A|\leq5000000$) и B($|B|\leq5000000$).

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

Индивидуализация.

Оптимизация по памяти: программа должна требовать O(m) памяти, где m - длина образца.

Описание префикс-функции.

Префикс функция – это функция, возвращающая для заданного символа строки максимальную длину равных собственных префикса и суффикса в строке.

В программе префикс функция для очередного символа строки вычисляется на основе предыдущего. Так, если текущий символ и символ, который идет после префикса максимальной длины предыдущего символа, равны, то значение префикс функции для текущей строки равно значению префикс функции для предыдущего. Иначе, можем рассмотреть префикс, который был расширен для получения префикс-функции предыдущего символа. Это возможно, так как значение префикс-функции указывает на последний символ именно такого префикса.

В ходе вычисления префикс функции проходится вся строка. Всего n итераций, где n – длина строки. На каждом шаге значение префикс функции может быть увеличено не больше, чем на единицу или уменьшено. Так как общее количество уменьшений не превосходит N, то на каждом шаге выполняется не более O(1) операций и получаем результирующую сложность по времени O(n).

Описание алгоритма 1.

В программе используется алгоритм Кнутта-Морриса-Пратта. Он заключается в том, что создается строка следующего вида: <образ>+<уникальный символ>+<текст> и для каждого символа этой строки вычисляется префикс функция. Затем, в полученном массиве значений ищется значение, равное длине образа. Индекс такого значение в массиве и есть местоположение образа в тексте.

Так как в алгоритме используется вычисление префикс-функции для строки длиной m+n, где m – длина подстроки, n – длина строки, то его сложность по времени составляет O(m+n).

В ходе выполнения алгоритма символы строки считываются по одному, и для каждого из них вычисляется значение префикс функции в независимости от других символов строки. Это возможно, так как из-за уникального символа значение префикс функции для строки не превысит длины подстроки и расширятся смогут только префиксы подстроки. Отсюда следует вывод, что сложность по памяти – O(m)

Описание алгоритма 2.

Нахождение цикла может быть произведено с помощью удвоения исходной строки. Тогда сдвинутая строка обязательно будет содержаться в удвоенной. Так как программа использует алгоритм Кнутта-Морриса-Пратта, то сложность по времени O(n+2n)=O(3n)=O(n), где n – длина исходной строки. Так как в ходе алгоритма происходит удвоение строки, то сложность по памяти O(n+2n)=O(n).

Описание функций.

std::vector<int> prefix(std::string str)

str – строка, для которой требуется вычислить префикс-функцию

Функция возвращает набор значений префикс функции для каждого символа

int char_prefix(std::string &str, std::vector<int> &pref, char c, int prev_char_pref)

str – строка образец

pref – префикс строки образца

c – символ, для которого требуется вычислить префикс-функцию

prev_char_pref – значение префикс функции предыдущего символа

Функция возвращает значение префикс функции для символа c.

std::vector<int> find(std::string &needle, std::basic_istream<char> &haystack)

needle – строка образец

haystack – строка, в которой требуется найти образец

Функция возвращает массив индексов, в которых начинается вхождение образца в строку

Тестирование.

```
ab
aaaaaaaaaab
```

9

```
ab
abababababbbabaa
```

0,2,4,6,8,10,13

```
ab
bbbbbbbbbbbbb
```

-1

```
aaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaab
```

0

```
gc
fmcgralbgcahfc
```

8

```
gc
fmcgralbgcahfcgbrc jdsvesfcvfjsfgv shgdvshevfsgc
```

8,44

Вывод.

В ходе выполнения лабораторной работы был изучен алгоритм Кнутта-Морриса-Пратта для нахождения подстроки в строке.

Приложения А. Исходный код

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

//#define DEBUG

std::vector<int> prefix(std::string str){

#ifdef DEBUG

    std::cout << std::endl;
    std::cout << "\tCalculating prefix-function: " << std::endl;

#endif

    std::vector<int> pref(str.length(), 0);
    for (int i=1; i<str.length(); i++){

#ifdef DEBUG

    std::cout << "\tNow symbol: " << str[i] << " with index " << i <<
std::endl;

#endif

        int now_pref = pref[i-1];

#ifdef DEBUG
        std::cout << "\tPrevious symbol's prefix function: " << now_pref <<
```

```
std::endl;
#endif
        // До тех пора, пока не нашли префикс, который можно расширить
        while (now_pref > 0 && str[i] != str[now_pref]){

#ifdef DEBUG

            std::cout << "\tstr[i] != str[now_pref] <=> " << str[i] << " != "
<< str[now_pref] << now_pref << std::endl;
            std::cout << "\tGo to symbol with index " << now_pref-1 << now_pref
<< std::endl;

#endif

            // Переходи к предыдущему префиксу. Т.е. префиксу, который расширил
текущий префикс
            now_pref = pref[now_pref-1];

#ifdef DEBUG
            std::cout << "\tHis prefix-function: " << now_pref << std::endl;
#endif

        }
        // Если в результате цикла выше нашли префикс, то расширяем его
        if (str[i] == str[now_pref]){
#ifdef DEBUG
            std::cout << "\tEqual symbols founded" << std::endl;
#endif
            now_pref++;
        } else{

#ifdef DEBUG
            std::cout << "\tEqual symbols not founded" << std::endl;
#endif

        }
        // Созраняем результат префикс функции
        pref[i] = now_pref;
#ifdef DEBUG
        std::cout << "\tResult prefix-function: " << now_pref << std::endl;
#endif
    }

    return pref;

}

int char_prefix(std::string &str, std::vector<int> &pref, char c, int
prev_char_pref){

    int now_pref = prev_char_pref;

#ifdef DEBUG

    std::cout << std::endl;
    std::cout << "\tCalculating prefix-function for symbol " << c << std::endl;
```

```
    std::cout << "\tPrevious symbol's prefix function: " << prev_char_pref <<
std::endl;

#endif

    // До тех пора, пока не нашли префикс, который можно расширить
    while (now_pref > 0 && c != str[now_pref]){

#ifdef DEBUG

        std::cout << "\tc != str[now_pref] <=> " << c << " != " <<
str[now_pref] << now_pref << std::endl;
        std::cout << "\tGo to symbol with index " << now_pref-1 << now_pref <<
std::endl;

#endif

        // Переходи к предыдущему префиксу. Т.е. префиксу, который расширил
текущий префикс
        now_pref = pref[now_pref-1];

#ifdef DEBUG
        std::cout << "\tHis prefix-function: " << now_pref << std::endl;
#endif

    }
    // Если в результате цикла выше нашли префикс, то расширяем его
    if (c == str[now_pref]){
#ifdef DEBUG
        std::cout << "\tEqual symbols founded" << std::endl;
#endif
        now_pref++;
    } else{
#ifdef DEBUG
        std::cout << "\tEqual symbols not founded" << std::endl;
#endif
    }
#ifdef DEBUG
    std::cout << "\tResult prefix-function: " << now_pref << std::endl <<
std::endl;
#endif
    // Созраняем результат префикс функции
    return now_pref;

}

std::vector<int> find(std::string &needle, std::basic_istream<char> &haystack){

#ifdef DEBUG
    std::cout << "Searching " << needle << " in haystack" << std::endl;
#endif

    // Массив индексов, по которым расположены вхождения подстроки в строку
    std::vector<int> indexes = {};

    // Длина искомой подстроки
```

```
    int needle_length = needle.length();
    // Уникальный символ. Нужен, что бы префикс всегда соответсвовал искомой
подстроки
    needle += "@";

    // Префикс-функция искомой подстроки
    auto pref = prefix(needle);

#ifdef DEBUG
    std::cout << "Needle's prefix-function: ";
    for (auto p: pref) std::cout << p << " ";
    std::cout << std::endl;
#endif

    // Префикс-функция от предыдущего символа. Так как это всегда уникальный
символ, то инициализируем нулем
    int prev_pref = 0;

    char now_c;
    // Индекс текущего символа в строке
    int index = 0;
    // Чтение текущего символа
    while (haystack >> now_c){
#ifdef DEBUG
        std::cout << "Read symbol: " << now_c << std::endl;
#endif
        // Вычисления префикс-функци для него
        prev_pref = char_prefix(needle, pref, now_c, prev_pref);

#ifdef DEBUG
        std::cout << "Read symbol's prefix-function: " << prev_pref <<
std::endl;
#endif

        // Если ее значение равна длине, то это одно из вхождений
        if (prev_pref == needle_length){
#ifdef DEBUG
            std::cout << "Now prefix-function equal needle length. New enter
founded" << std::endl;
#endif
            indexes.push_back(index-needle_length+1);
        }

        // Переходим к следующему символу
        index++;
    }

    return indexes;

}

//int main(){
//
//    std::string a, b;
//    std::getline(std::cin, a);
//    std::getline(std::cin, b);
```

```
//
//    // Удвоение строки
//    a += a;
//
//    std::stringstream ss(a);
//
//    auto indexes = find(b, ss);
//
//    if (indexes.size()){
//        std::cout << indexes[0];
//    } else{
//        std::cout << -1;
//    }
//
//}

int main(){

    std::string p;
    std::getline(std::cin, p);

    auto indexes = find(p, std::cin);

    if (indexes.size()){
        std::cout << indexes[0];
        for (int i=1; i<indexes.size(); i++){

            std::cout << "," << indexes[i];

        }
    } else{
        std::cout << -1;
    }

}
```