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 Кнута-Морриса-Прата

Студент гр. 7304 _____ Пэтайчук Н.Г.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2019

Цель работы

Исследование алгоритмов поиска подстроки в строке, в частности алгоритм Кнута-Морриса-Прата.

Постановка задачи

- Найти все вхождения подстроки P в строку T (|P| <= 15000, |T| <= 5000000);
  - Входные данные: строка P, затем строка T;
  - Выходные данные: все индексы вхождения подстроки P в T через запятую, либо -1, если подстрока P не входит в строку T;
- Даны две строки A и B. Определить, является ли строка A циклическим сдвигом строки B (то есть длина A равна длине B и строка A состоит из суффикса B, к которому присоединили оставшийся префикс B);
  - Входные данные: строка A, затем строка B;
  - Выходные данные: -1, если A не является циклическим сдвигом B, либо первый индекс начала строки B в A (если A можно получить различными циклическими сдвигами из B);

Ход работы

1. Создание префикс-функции, которая является главным компонентом в алгоритме Кнута-Морриса-Прата. Данная функция по символу в строке определяет максимальную длину префикса, который в то же время является суффиксом подстроки, которая заканчивается данным символом;
2. Определение функции,принимающей две строки и реализующей алгоритм КМП поиска второй строки в первой. Первым делом вычисляются для каждого символа в строке-шаблоне значения префикс-функции, далее идёт подсчёт значений префикс-функции для символов строки-текста, где мы

ищем все вхождения строки-шаблона, при этом мы считаем, что строка-шаблон является префиксом строки-текста. В тот момент, когда префикс-функция для очередного символа выдаёт в качестве значения длину строки-шаблона, мы вычисляем индекс, с которого начинается суффикс, который является и префиксом, и записываем его в результирующий массив, который является возвращаемым значением данной функции. Пустой массив говорит о том, что строка-шаблон не входит в строку-текст;

3. Реализации функции проверки на то, что первая передаваемая строка является циклическим сдвигом второй передаваемой строки. Данная функция первым делом вычисляет значения префикс-функции от символом сконкатенированной строки, где между двумя переданными строками есть символ-разделитель, не встречающийся ни в одной из строк. Затем берётся значение префикс-функции от последнего символа сконкатенированной строки, которое является индексом начала оригинальной строки в циклически сдвинутой строке и гарантом того, что префикс строки перед символом-разделителем идентичен суффиксу строки после символа-разделителя. Остаётся лишь проверить на идентичность остальные символы, что делается за линейное время;
4. Реализация головной функции, где происходит ввод/вывод данных;

Весь исходный код программы представлен в Приложении 1.

Тестирование программы

1) Демонстрация работы функции, реализующей алгоритма Кнута-Морриса-Прата:

```
aaa
abaaabbaaababababaaabaaabbb
2,7,15,19
```

2) Демонстрация работы функции, проверяющей, сдвинута ли циклически первая строка по отношению ко второй:

```
abcdefghijk
efghijkabcd
4
```

Вывод

В ходе данной лабораторной работы был исследован алгоритм поиска подстроки в строке Кнута-Морриса-Прата. Были получены знания о том, что такое префикс-функция и как она работает, а также то, каким образом благодаря ней можно быстро (за линейное время) найти все вхождения подстроки в строку. Также с использованием префикс-функции была решена другая задача, а именно проверка того, является ли одна строка циклическим сдвигом другой строки, что говорит о исключительной полезности префикс-функции в решении строковых задач.

Приложение 1: Исходный код программы

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

using namespace std;

vector<int> prefix_func(string &str)
{
    int str_length = str.size();
    vector<int> pi(str_length);

    pi[0] = 0;
    for (int index = 1; index < str_length; index++)
    {
        int PF_value = pi[index - 1];
        while ((PF_value > 0) && (str[index] != str[PF_value]))
            PF_value = pi[PF_value - 1];
        if (str[index] == str[PF_value])
            PF_value++;
        pi[index] = PF_value;
    }
    return pi;
}

vector<int> findSubstr_KMP(string &text, string &pattern)
{
    size_t pattern_length = pattern.size();
    vector<int> answer;
    vector<int> PF_values = prefix_func(pattern);
    int last_PF_value = 0;

    for (int index = 0; index < text.size(); index++)
    {
        while ((last_PF_value > 0) && (text[index] !=
pattern[last_PF_value]))
            last_PF_value = PF_values[last_PF_value - 1];
        if (text[index] == pattern[last_PF_value])
            last_PF_value++;
        if (last_PF_value == pattern_length)
            answer.push_back(index - pattern_length + 1);
    }
    return answer;
}

int check_isShiftedString(string &shifted, string &original)
{
    string concatinated = shifted + "@" + original;
    vector<int> PF_values = prefix_func(concatinated);
    int index = PF_values[PF_values.size() - 1];
```

```
    for (int i = index; i < shifted.size(); i++)
        if (shifted[i] != original[i - index])
            return -1;
    return index;
}

int main()
{
    /*string text, pattern;

    cin >> pattern >> text;

    vector<int> substr_indexes = findSubstr_KMP(text, pattern);
    if (substr_indexes.empty())
        cout << "-1";
    else
    {
        for (int i = 0; i < substr_indexes.size() - 1; i++)
            cout << substr_indexes[i] << ",";
        cout << substr_indexes[substr_indexes.size() - 1];
    }*/
    string shifted, original;

    cin >> shifted >> original;

    if (original.size() != shifted.size())
        cout << "-1";
    else if (original == shifted)
        cout << "0";
    else
    {
        int index = check_isShiftedString(shifted, original);
        cout << index;
    }
    return 0;
}
```