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-Пратта

Студент гр. 7383 _____ Тян Е.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ

Цель работы: исследовать и реализовать задачу поиска вхождения подстроки в строке, используя алгоритм Кнута-Морриса-Пратта.

Формулировка задачи: необходимо разработать программу, которая реализует алгоритм Кнута-Морриса-Пратта и с его помощью для заданных шаблона P ($|P| \leq 15000$) и текста T ($|T| \leq 5000000$) найти все вхождения P в T. Если P не входит в T, то вывести -1.

Также следует разработать программу для решения следующей задачи: заданы две строки A ($|A| \leq 5000000$) и B ($|B| \leq 5000000$).

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. Если A является циклическим сдвигом B, индекс начала строки B в A, иначе вывести -1. Если возможно несколько сдвигов вывести первый индекс.

Входные данные:

Первая строка - P.

Вторая строка - T.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

В данной работе используются главная функция (int main()) и дополнительные функции (void kmp(string pat, string txt, int* pi), void preFunc(string pat, int* pi)).

Параметры, хранящиеся в функции void kmp(string pat, string txt, int* pi):

- pat – строка, вхождения которой нужно найти;
- txt – строка, где нужно найти вхождения;
- pi – массив, содержащий значения префикс-функции.

Параметры, содержащиеся в функции void preFunc(string pat, int* pi):

- pat – строка, вхождения которой нужно найти;
- pi – массив, содержащий значения префикс-функции.

В функции main() считываются строки *P* и *T*. Проходит проверка на возможность вхождения *P* в *T*, путем сравнения длин строк. Если это возможно, тогда вызывается функция void kmp(string pat, string txt, int* pi). Вызывается функция void preFunc(string pat, int* pi), которая ищет значения префикс-функции для строки *P*. После того, как был заполнен массив int* pi, содержащий значения префикс-функции для строки *P*, начинается поиск первого индекса вхождения *P* в *T*.

Для более понятной работы алгоритма рассмотрим пример работы программы для строк «аасаа» и «аасаааааасаасаа». Так как длина первой строки не превышает длины второй строки, то возможны вхождения первой строки во вторую. Префикс-функция вычисляет значения для первой строки: функция находит наибольшую длину наибольшего собственного суффикса подстроки, совпадающего с ее префиксом. Значение для первого символа всегда полагается равным 0. Далее для подстроки «аа» максимальный суффикс равный префиксу 1. Для «аас» – 0, т.к. нет суффикса равного префиксу. Для «ааса» – 1. Для «аасаа» – 2. Когда вычислены все значения префикс-функции для первой строки, то начинается поиск первой строки во второй: начинаем сравнение символов с начала в обеих строках, если

символы равны – продолжаем сравнение пока не дойдем до первого символа, не совпадающего с символом в первой строке, или пока не дойдем до конца первой строки. Если дошли до конца первой строки, значит первый индекс вхождения первой строки во вторую найден, вычисляем первый индекс, запоминаем и продолжаем операцию сравнения для следующего символа во второй строке и для последнего символа, лежащего по адресу, хранящемуся для данного символа в массиве int* pi, в первой строке. Аналогично, если символы не совпадают.

Для задания с циклическим сдвигом производилось склеивание двух строк с разделителем: *P* + <*разделитель*> + *T*. Далее использовалась префикс-функция для всей получившейся строки. После использовалась функция void kmp(string pat, string txt, int* pi), которая осуществляла поиск циклического сдвига, используя данные префикс-функции.

3. ТЕСТИРОВАНИЕ

Программа была собрана в компиляторе G++ в OS Linux Ubuntu 12.04. Программа может быть скомпилирована с помощью команды:

g++ <имя файла>.cpp -std=c++11

Тестовые случаи представлены в Приложении А.

Исходя из тестовых случаев можно увидеть, что тестовые случаи не выявили некорректной работы программы, что говорит о том, что по результатам тестирования было показано, что поставленная задача была выполнена.

4. ИССЛЕДОВАНИЕ

Компиляция была произведена в компиляторе в g++ Version 4.2.1. В других ОС и компиляторах тестирование не проводилось.

В начале работы программа вычисляет значения префикс функции для каждого символа первой строки. Допустим длина первой строки P. После начинается сравнение каждого символа второй строки с символами первой строки. Пусть длина второй строки T. Тогда сложность алгоритма по времени будет составлять O(P + T).

По памяти сложность алгоритма составляет O(P), т.к. вычисляются значения префикс-функции только для первой строки.

5. ВЫВОД

В ходе выполнения лабораторной работы была решена задача поиска подстроки в строке с помощью алгоритма Кнута-Морриса-Пратта на языке C++, и исследован алгоритм Кнута-Морриса-Пратта. Полученный алгоритм имеет сложность линейную как по времени, так и по памяти.

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

№	Ввод	Вывод
1	aabaa aabaabaabaabaaaaaa	0,3,6,9
2	abcdef htyuifabcef	-1
3	bsjkdhfvnd df	-1
4	2017 20182020172019	6
5	defabc abcdef	3

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp:

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

using namespace std;

void preFunc(string pat, int* pi)
{
    pi[0] = 0;
    int len = 0;
    for(int i = 1; i < pat.length();)
    {
        if(pat[i] == pat[len])
        {
            pi[i] = ++len;
            i++;
        }
        else
        {
            if(len != 0)
                len = pi[len - 1];
            else
            {
                pi[i] = len;
                i++;
            }
        }
    }
}

void kmp(string pat, string txt, int* pi)
{
    preFunc(pat, pi);
    vector<int> answer;
    int i = 0, j = 0;
    while(i < txt.length())
    {
        if(txt[i] == pat[j])
        {
            i++;
            j++;
        }
        if(j == pat.length())
        {
            answer.push_back(i - j);
            j = pi[j - 1];
        }else
            if(txt[i] != pat[j])
            {
                if(j != 0)
                    j = pi[j - 1];
```

```
            else
            {
                i++;
            }
        }
    }
    if(answer.empty())
        cout << "-1";
    else
        for(int m = 0; m < answer.size(); m++){
            cout << answer[m];
            if(m != answer.size() - 1)
                cout << ',';
        }
    cout << endl;
}

int main()
{
    string pat, txt;
    cin >> pat >> txt;
    int pi[pat.length()];
    if(pat.length() > txt.length())
        cout << "-1" << endl;
    else
        kmp(pat, txt, pi);
    return 0;
}
```

cycle.cpp:

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

using namespace std;

void preFunc(string pat, int* pi)
{
    pi[0] = 0;
    int len = 0;
    for(int i = 1; i < pat.length();)
    {
        if(pat[i] == pat[len])
        {
            pi[i] = ++len;
            i++;
        }
        else
        {
            if(len != 0)
                len = pi[len - 1];
            else
            {
                pi[i] = len;
```

```
            i++;
        }
    }
}

void kmp(string pat, string txt, int* pi, int pat_len)
{
    preFunc(pat, pi);
    int i = 0, j = pi[pat.length() - 1];
    while(j <= pat_len)
    {
        if(txt[i] == pat[j])
        {
            i++;
            j++;
        }
        if(j == pat_len)
        {
            int dif;
            dif = pat_len - i;
            cout << txt.length() - dif << endl;
            return;
        }else
            if(txt[i] != pat[j])
            {
                break;
            }
    }
    cout << "-1" << endl;
}

int main()
{
    string pat, txt;
    cin >> txt >> pat;
    int pat_len = pat.length();
    pat = pat + '@' + txt;
    int* pi = new int[pat.length()];
    if(pat_len > txt.length())
        cout << "-1" << endl;
    else
        kmp(pat, txt, pi, pat_len);
    delete [] pi;
    return 0;
}
```