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. 8303		Сенюшкин Е.В.
Преподаватель		Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучение алгоритма Кнута-Морриса-Пратта для нахождения подстроки в строке.

Задание 1.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P ($|P| \leq 15000$) и текста T ($|T| \leq 5000000$) найдите все вхождение P в T.

Вход:

Первая строка - P

Вторая строка - T

Выход:

Индексы начала вхождений P в T, разделенных запятой, если P не входит в T, то вывести -1.

Задание 2.

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.

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.

Индивидуализация.

Вар 2. Оптимизация по памяти: программа должна требовать O(m) памяти, где m - длинна образца. Это возможно, если не учитывать память, в которой хранится строка поиска.

Префикс-функция.

Префикс-функция — это функция, которая принимает строку и возвращает максимальную длину подстроки, которая является одновременно префиксом и суффиксом в этой строке.

В программе *префикс-функция* считается для всех префиксов строки, поэтому вычисление значение *префикс-функции* для следующего префикса можно на основе уже вычисленных значения.

Пусть значения *префикс-функции* в строке *s [0 … n]* были посчитаны для всех строк до *s [0 … i]*. Для вычисления значение *префикс-функции* для строки *s [0 … i]* берется значение п*рефикс-функции* k для строки *s [0 … i – 1]* на основе этого значение сравниваются *s[k]* и *s[i]*, если они равны, значит символ на который увеличилась строка *s [0 … i - 1]* дополняет строку являющиеся префиксом и суффиксом для строки *s [0 … i – 1]* в таком случае значение *k* увеличивается на 1, если символы различаются то берется значение *префикс-функции* для строки *s [0 … k] (потому что строка s [0 ... i - 1] начинается и заканчивается на строку s [0 … k] из этого следует, что значение префикс-функции для строки s [0 … k] так же является строкой, которая является префиксом и суффиксом для строки s[0 … i –1])* и алгоритм повторяется, если значение *k* становится равным 0, значит были проверены все строки, которые являются префиксом и суффиксом одновременно, в последний раз сравниваются s[0] и s[i].

Описание алгоритма 1.

В программе используется алгоритм Кнута-Морриса-Пратта. Даны цепочка T и образец P. Требуется найти все позиции, начиная с которых P выходит в T.

Вначале алгоритм считает значение прификс-функции для всех префиксов строки P и сохраняет их в массив.

После алгоритм считает значение префикс-функции для всех префиксов строки P + # + T, где # символ, который не присутствует ни в строке P, ни в строке T, но теперь значение префикс-функции нигде не сохраняются, потому что максимальный префикс, который может быть одновременно и суффиксом это строка P, а все значения для нее уже посчитаны. Если значение префикс-функции становится равной длине строки P, то было найдено вхождение P в строку S. Индекс начала вхождение строки сохраняется в массив. После того как алгоритм посчитал значение префикс-функции для всех префиксов, массив индексов возвращается из функции.

Сложность алгоритма.

По времени.

Сложность нахождения префикс функции для всех префиксов O(n) потому что на каждой итерации цикла k увеличивается на 1, то есть максимальный размер k может достигнуть размера n - 1, и на каждой итерации цикла вызывается while, который уменьшает k причем минимальный размер уменьшения 1, то есть в худшем случае будет совершенно 2 * n операций.

Все что делает алгоритм это считает префикс функцию для строки P + # + T, то есть в худшем случае будет совершенное $2 * n + 2 * m$ операция, где n длина шаблона, а m длинна текста.

Сложность по операциям $O(n + m)$ где n длина шаблона, а m длинна текста.

По памяти.

Вся дополнительная память, которая используется в программе это массив для сохранения значений префикс-функции для всех префиксов строки P

Сложность по памяти *O(n)*, где *n* длинна шаблона.

Описание алгоритма 2.

Алгоритм такой же, как и первый алгоритм, но теперь даны две строчки A и B и нужно определить является строка A циклическим сдвигом строки B.

В этот раз алгоритм считает и запоминает значение префикс-функции для всех префиксов строки B.

А после алгоритм считает значение префикс-функции в строке B + # + A + A, и если значение префикс-функции становится равным длине строке B, значит был найден циклический сдвиг, алгоритм завершается и возвращает индекс начала строки B в A, если алгоритм прошел всю строку B + # + A + A и не нашел индекс циклического сдвига, то возвращается - 1.

Сложность алгоритма.

По времени.

Сложность нахождения префикс функции для всех префиксов O(n) потому что на каждой итерации цикла k увеличивается на 1, то есть максимальный размер k может достигнуть размера n - 1, и на каждой итерации цикла вызывается while, который уменьшает k причем минимальный размер уменьшения 1, то есть в худшем случае будет совершенно 2 * n операций.

Все что делает алгоритм это считает префикс функцию для строки

B + # + A + A, то есть в худшем случае будет совершенное *6 * n* операция, где *n* длинна строк B и A.

Сложность по операциям *O (n),* где *n* длинна строк B и A.

По памяти.

Вся дополнительная память, которая используется в программе это массив для сохранения значений префикс-функции для всех префиксов строки B

Сложность по памяти $O(n)$, где n длинна строки B.

Описание функций.

void prefixFunction(string P) - функция, которая принимает строку и считает значение префикс-функции для всех префиксов строки P и сохроняет их в массив

vector<int> KMP(const string& P, const string& T) - функция принимает строку P и T и ищет все вхождения строки P в T. Возвращает массив индексов начала вхождение строки P в T.

int stringRotation(const string& A, const string& B) - функция принимает строку A и B и определяет является ли A циклическим сдвигом строки B. Возвращает индекс начала строки B в A или -1, если циклического сдвига нет.

Тестирование.

Алгоритм поиска подстроки

Ввод	Вывод
ab aaaaaaaaab	9
aba abaababbaababababa	0,3,9,11,13,15
abc dabc	1
saha shla_saha_po_shose_i_sosala_suhku	5
asdasdfasdf sadfasdf	-1

Алгоритм поиска циклического сдвига

Ввод	Вывод
qwertyuiopasdfghjklzxcvbnm tyuiopasdfghjklzxcvbnmqwer	4
sdfasdf asdfsdd	-1
abcdef defabc	3

Вывод.

В ходе выполнения лабораторной работы был изучен алгоритм Кнута-Мориса-Пратта и применен для поиска подстроки в строке, а также для проверки является ли строка циклическом сдвигом другой строки.

Приложение А.

Исходный код

KMP.cpp

```
#include <iostream>

#include <vector>

#include <string>

#define DBG


using namespace std;


vector<int> p; // массив для хранения значения префиксфункции искомой подстроки


void prefixFunction(string P) // функция считает префикс функцию от всех префиксов строки и записывет значение для всех префиксов в массив p

{

    p[0] = 0;

    for (size_t i = 1; i < P.size() + 1; i++)

    {

        int k = p[i - 1];               // получение значение максимальной прификс-функции, от строки s[0 ... i - 1]


        while (k > 0 && P[i] != P[k])   // перебираем все строки, которые являются прификсами и суфиксами

        {                               // строки s[0 ... i - 1] и пытаемся расширить их символом s[i]

            k = p[k - 1];

        }

        if (P[i] == P[k])               // если символ s[i] совпал с s[k] значит префикс удалось расширить

            k++;

        p[i] = k;                       // запоминаем длину прифекс-функции для строки s[0 ... i]

    }

}
```

```
vector<int> KMP(const string& P, const string& T){ // функция ищет все вхождение строки P в строку T и возвращает массив индексов этих вхождений
    vector<int> ans;
#ifdef DBG
    cout << "Prefix function of " << P << " : ";
    for (size_t i = 0;i < p.size(); i++)
        cout << p[i] << ' ';
    cout << endl;
#endif

    p.resize(P.size() + 1); // нахождение значение префикс-функции
    prefixFunction(P);              // для всех префиксов строки P

    int k = 0;
    for(size_t i = 0; i < T.size(); i++){  // считаем значение префикс-функции для всех префиксов строки P + T
        while (k > 0 && P[k] != T[i])
            k = p[k - 1];
        if (P[k] == T[i])
            k++;

#ifdef DBG
        const std::string green("\033[0;32m");
        const std::string reset("\033[0m");

        cout << "Prefix function of \""
        << green << P.substr(0, k) << reset
        << '|' + P.substr(k, P.size()-k) <<
        "\" and \"" + T.substr(0, i-k+1) + '|'
        << green << T.substr(i-k+1, k) << reset
        << "\" = " << k << endl;
#endif

        if (k == P.size())                // если длинна прификс-функции совпало с длинной строки P значит в строке T была найдена строка P
```

```
            ans.push_back(i - P.size() + 1);

    }
    return ans;
}


int main(){
    string P;
    string T;

    cin >> P >> T;

    vector<int> a = KMP(P, T);

    if (a.empty()){
        cout << -1;
    }else {
        for(size_t i = 0; i < a.size(); i++) {
            cout << a[i];
            if (i + 1 != a.size()) cout << ',';
        }
    }
    cout << endl;
    return 0;
}
```

string_rotation.cpp

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

```
using namespace std;


vector<int> p; // массив для хранения значения префикс функции искомой подстроки


void prefixFunction(string P) // функция считает префикс функцию от всех префиксов строки и записывет значение для всех префиксов в массив p
{
    p[0] = 0;
    for (size_t i = 1; i < P.size() + 1; i++)
    {
        int k = p[i - 1];
        while (k > 0 && P[i] != P[k]) // алгоритм перебирает все строки, которые являются префиксами и суффиксами, строки s[0 .. i - 1]
        {                             // в порядке уменьшения их длинны и проверят равенство элементов s[i] и s[k], если
            k = p[k - 1];             // они равны, значит алгоритм нашел строку, которая является префиксом и суффиксом строки s[0 .. i],
        }                             // если алгоритм дошел до k = 0, значит у строки s[0 .. i] нет строки, которая является префиксом и суффиксом
        if (P[i] == P[k])             // одновременно, алгоритм запоминает значение прификс функции для строки s[0 .. i]
            k++;
        p[i] = k;
    }
}



int stringRotation(const string& A, const string& B){


    p.resize(B.size() + 1);  // нахождением значение префикс функции
    prefixFunction(B);               // для всех префиксов строки B


#ifdef DBG
    cout << "Prefix function of " << B << " : ";
    for (size_t i = 1;i < p.size(); i++)
```

```
        cout << p[i] << ' ';
    cout << endl;
#endif


    int k = 0;


    for (size_t i = 0; i < A.size(); i++){ // алгоритм считает значение префикс функции для всех префиксов строки B + A + A,
        while (k > 0 && B[k] != A[i])      // чтобы не использовать дополнительной памяти и не сохранять нигде строку A + A алгоритм два раза проходит по
            k = p[k - 1];                  // строке A. Вычисление префикс функции начинается с начала строки A так как для строки B все уже вычислено
        if (B[k] == A[i])                  // если значение прификс функции для строки A + A стало равно длине строки B значит, алгоритм нашел значением циклического сдвига
            k++;



        if (k == B.size())
            return (static_cast<int>(i) - static_cast<int>(B.size()) + 1);
    }


    for (size_t i = A.size(); i < A.size() + A.size(); i++){
        while (k > 0 && B[k] != A[i - A.size()])
            k = p[k - 1];
        if (B[k] == A[i - A.size()])
            k++;
        if (k == B.size())
            return (static_cast<int>(i) - static_cast<int>(B.size()) + 1);
    }
    return -1;
}



int main(){
```

```
    string A;
    string B;

    cin >> A >> B;

    int a = stringRotation(A, B);

    cout << a;
    cout << endl;
    return 0;
}
```