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 _____ Дирксен А.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучение алгоритма Кнута-Морриса-Пратта.

Задание 1

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

Пример входных данных

```
ab
abab
```

Пример выходных данных

```
0,2
```

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

Пример входных данных

```
defabc
abcdef
```

Пример выходных данных

```
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 префикса и суффикса в строке.

В программе префикс-функция для очередного символа строки вычисляется на основе предыдущего. Так, если текущий символ и символ с индексом равным префикс-функции предыдущего символа + 1, равны, то значение префикс-функции для текущей строки равно значению префикс-функции для предыдущего + 1. 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 число уменьшений не может превосходить N, то получаем результирующую сложность по времени O(n).

Описание алгоритма поиска подстроки.

В программе используется алгоритм Кнутта-Морриса-Пратта. Он заключается в том, что сначала вычисляется префикс-функция для образа, а затем применяется алгоритм префикс-функции для каждого символа текста, только на этот раз сравниваются символы не одной строки, а двух. Если значение префикс-функции совпадает с размером образа, значит обнаружено вхождение подстроки.

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

В ходе выполнения алгоритма префикс-функция вычисляется только для символов подстрок, а значит сложность по памяти составляет – O(m).

Описание алгоритма проверки циклического сдвига.

Нахождение цикла может быть произведено с помощью удвоения исходной строки. Тогда сдвинутая строка обязательно будет содержаться в удвоенной. Но, в целях экономии памяти, в алгоритме вместо удвоения строки исходная строка обходится два раза. В этом алгоритме, аналогично алгоритму поиска подстроки, подсчитывается сначала префикс-функция для одной строки(O(n)). А затем считается префикс-функция для второй строки за два прохода(O(2n)), но эти значения не записываются. Тогда сложность по времени O(3n)=O(n), где n – длина исходной строки.

Так как в ходе алгоритма в памяти хранится значения префикс-функции только для одной строки, то сложность по памяти O(n).

Описание структур данных.

vector<int> p — вектор для хранения префиксной функции

Описание функций.

vector<int> prefix(string str)

str — обрабатываемая строка

Функция подсчета префиксной функции строки. Возвращает вектор значений префиксной функции.

vector<int> findSubstring(string temp, string text)

temp — строка, которую необходимо найти.

text — строка, в которой происходит поиск.

Функция поиска всех вхождений подстроки в строке. Возвращает вектор вхождений.

int findLoop(string A, string B)

Функция проверяет, является ли строка A циклическим сдвигом строки B. Если является возвращает индекс начала строки B в строке A. Иначе возвращает -1.

Тестирование.

Алгоритм поиска подстроки	
Input	Output
asd etawdasdtqwe	5
asd asdasdqweqweqweasdasdqweqweqwe	0,3,15,18
hsega asbvesasgehseg	-1
abab abababababababababababab	0,2,4,6,8,10,12,14,16,18,20
Алгоритм поиска циклического сдвига	
defabc abcdef	3
qazaqstan stanqazaq	5
hesagun hesagun	0
abababba abababab	-1

Вывод.

В ходе выполнения лабораторной работы были изучен алгоритм Кнута-Морриса-Пратта и применен для поиска подстроки в строке, а также для проверки является ли одна строка циклическим сдвигом другой.

Приложения А. Исходный код

lab4.cpp

```
#include <iostream>
#include <string>
#include <vector>
#define DBG
using namespace std;

//функция подсчета префиксной функции для строки
vector<int> prefix(string str){
    //создаем вектор необходимого размера
    vector<int> p(str.size() + 1, 0);
    p[0] = 0;
    for (int i = 1; i <= str.size(); i++) {
        //начинаем проверять равенство символов начиная с символа под
номером k,
        //который равен префиксной функции на предыдущем шаге
        int k = p[i - 1];
        //проверяем до тех пор пока символы не совпадут либо k не станет
равно 0
        while (k > 0 && str[k] != str[i])
            k = p[k - 1];
        //если мы вышли по условию совпадения символов увеличим k, после
этого запишем
        //значение префиксной функции в вектор
        if (str[i] == str[k])
            k++;
        p[i] = k;
    }
    return p;
}

vector<int> findSubstring(string temp, string text){
    //считаем префиксную функцию для шаблона
    vector<int> p = prefix(temp);
#ifdef DBG
    cout << "Prefix function of " << temp << " : ";
    for (int i : p)
        cout << i << ' ';
    cout << endl;
#endif
    vector<int> res;
    int m = temp.size();
    int k = 0;
    //начинаем проверять равенство символов начиная с нулевого символа
    for (int i = 0; i < text.size(); i++) {

        while (k > 0 && temp[k] != text[i])
            k = p[k - 1];
        //если мы вышли по условию совпадения символов увеличим k, но в
вектор уже ничего не добавляем
        if (temp[k] == text[i])
            k++;
#ifdef DBG
        const std::string green("\033[0;32m");
```

```
        const std::string reset("\033[0m");

        cout << "Prefix function of \""
        << green << temp.substr(0, k) << reset
        << '|' + temp.substr(k, temp.size()-k) <<
        "\" and \"" + text.substr(0, i-k+1) + '|'
        << green << text.substr(i-k+1, k) << reset
        << "\" = " << k << endl;
#endif
        //если префиксная функция равна длине шаблона, добавляем индекс
начала вхождения в результат
        if (k == m)
            res.push_back(i - m + 1);
    }
    return res;
}

int main() {
    string temp;
    string text;
    cin >> temp >> text;

    vector<int> res = findSubstring(temp, text);

    for (int i = 0; i < res.size(); i++) {
        cout << res[i] << ((i == res.size() - 1) ? "" : ",");
    }
    if (res.empty())
        cout << -1;
    return 0;
}
```

lab4_2.cpp

```
#include <iostream>
#include <string>
#include <vector>
//#define DBG
using namespace std;

//функция подсчета префиксной функции для строки
vector<int> prefix(string str){
    //создаем вектор необходимого размера
    vector<int> p(str.size() + 1, 0);
    p[0] = 0;
    for (int i = 1; i <= str.size(); i++) {
        //начинаем проверять равенство символов начиная с символа под
номером k,
        //который равен префиксной функции на предыдущем шаге
        int k = p[i - 1];
        //проверяем до тех пор пока символы не совпадут либо k не станет
равно 0
        while (k > 0 && str[k] != str[i])
            k = p[k - 1];
        //если мы вышли по условию совпадения символов увеличим k, после
этого запишем
        //значение префиксной функции в вектор
```

```
        if (str[i] == str[k])
            k++;
        p[i] = k;
    }
    return p;
}

int findLoop(string A, string B){
    //если длины строк не равны сразу возвращаем -1
    if (A.size() != B.size())
        return -1;
    //считаем префиксную функцию для шаблона
    vector<int> p = prefix(A);
#ifdef DBG
    cout << "Prefix function of " << A << " : ";
    for (int i : p)
        cout << i << ' ';
    cout << endl;
#endif
    vector<int> res;
    int m = A.size();
    int k = 0;
    //начинаем проверять равенство символов начиная с нулевого символа и
проходим строку два раза
    for (int i = 0; i < 2*B.size(); i++) {
        while (k > 0 && A[k] != B[i%B.size()])
            k = p[k - 1];
        //если мы вышли по условию совпадения символов увеличим k, но в
вектор уже ничего не добавляем
        if (A[k] == B[i%B.size()])
            k++;
        if (i >= B.size() && k == 0){
            return -1;
        }
#ifdef DBG
        const std::string green("\033[0;32m");
        const std::string reset("\033[0m");
        int st = max(0, (int)(i-B.size())+1);
        cout << "Prefix function of \""
             << green <<  A.substr(0, k) << reset
             <<'|' + A.substr(k, A.size()-k)
             <<"\" and \""
             << B.substr(st,i-k+1-st) + '|'
             << green <<  B.substr(i-k+1, k-st)
             << B.substr(0, st)<< reset + "\" = " << k << endl;
#endif
        //если префиксная функция равна длине шаблона, добавляем индекс
начала вхождения в результат
        if (k == m)
            return B.size() - i%B.size() - 1;
    }
    return -1;
}

int main() {
    string A;
```

```
    string B;
    cin >> A >> B;

    cout << findLoop(A, B);

    return 0;
}
```