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

Студентка гр. 8303 _____ Потураева М.Ю.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить принцип работы алгоритма КМП для поиска подстроки в строке. Разработать программу, которая реализует алгоритм КМП.

Задание.

1)Реализуйте алгоритм КМП и с его помощью для заданных шаблона P(|P|<15000) и текста T (|T|<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

Sample Input:

ab

abab

Sample Output:

0,2

2) Заданы две строки A(|A|<5000000) и B(|B|<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.

Sample Input:

defabc

abcdef

Sample Output:

3

Индивидуализация.

Подготовка к распараллеливанию: работа по поиску разделяется на k равных частей, пригодных для обработки k потоками (при этом длина образца гораздо меньше длины строки 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 префикс-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 Кнута-Морриса-Пратта.

В начале алгоритма для образца вычисляется префикс-функция, т.е. массив чисел pi[0..n-1]:pi[i] хранит значение, равное максимальной длине

совпадающих префикса и суффикса подстроки в образе, которая заканчивается i-м символом. В частности, значение pi[0] полагается равным нулю.

Для единственного потока алгоритм работает следующим образом. При каждом совпадении i и j символов оба индекса увеличиваются на 1, i — индекс в тексте, j — индекс в шаблоне. Если j становится равной размеру строки-шаблона, то все символы строки-шаблона совпали, и вхождение найдено. Если очередной символ не совпал с i-ым символом, то, если j = 0, индекс i увеличивается на один, в обратном случае (j =/= 0) алгоритм обращается к префикс-функции и приравнивает j к значению префикс-функции для символа, предшествующего не совпавшему.

При разделении на потоки вместо всей строки-текста рассматриваются ее подстроки. Подстрока строится следующим образом. Сначала вычисляется flowLen - длина части текста. Она равняется частному от деления длины всего текста на количество потоков с округлением вверх, стартовая позиция строки потока определяется как f*flowLen, где f — номер потока. В худшем случае окажется, что последний символ части — первый символ вхождения и почти вся его длина находится в следующей части. Конечная позиция строки потока во всем тексте определяется сдвигом стартовой позиции на длину flowLen и прибавлением величины p-1, где p — длина шаблона, для учета вхождений на стыке частей. Для последнего потока конечная позиция — это конец строки-текста. Повтор одного и того же индекса исключается тем, что при добавлении p-1 невозможно захватить новое вхождение.

Сложность по времени будет O(n+m). Префикс-функция образца вычисляется за o(m), где m - длина шаблона. Поиск вхождения образца в текст происходит за один цикл, максимальная длина которого равняется длине строки-текста, тогда сложность поиска — o(n), где n — эта длина.

Сложность по памяти будет O(m+n). Т.к. для работы алгоритм вычисляет значение префикс функции для каждого символа образца и хранит эти значения в векторе, поэтому сложность алгоритма по памяти составляет O(m+n), где m - длина шаблона, n-длина текста.

Описание алгоритма проверки циклического сдвига.

В начале алгоритма считываются две строки одинаковой длины A и B, если длины не совпадают, то алгоритм сразу завершает работу и выводит -1, т.к. в таком случае A не является циклическим сдвигом. Затем с помощью той же функции вычисляется префикс-функция для строки B. После этого строка A удваивается, чтобы найти индекс строки B в строке A (если является циклическим сдвигом). После алгоритм заводит два индекса, которые указывают на символы в удвоенной строке A и строке B, после чего алгоритм проходит и сравнивает символы. Если произошло очередное совпадения, то индексы увеличиваются, если происходит несовпадение, то с помощью вектора префикс-функций для B осуществляется "сдвиг" образа. Когда индекс указывающий на символ в образе B равен длине этого образа, результат помещается в вектор результата и он выводится на экран. Т.к. A - это удвоенная строка, то мы определили что исходная строка A является циклическим сдвигом B и алгоритм завершает работу. В противном случае выводится -1.

Сложность по времени будет O (2|A|+| B|)= O(3| A|)= O(|A|+ |B|) ,т.к. для заполнения префикс-функции нужно B операций, а при поиске цикл проходит за линейное время по удвоенной строке A.

Сложность по памяти будет O(|A|+ |B|), т.к. в этом случае алгоритм строит префикс-функцию для строки B и на ее хранение требуется B памяти, а удвоение строки требует дополнительно A памяти.

Описание функций.

1) `void prefixFunction(string& p, std::vector<size_t>& pi, bool isForShift)`

Функция prefixFunction() вычисляет префикс-функцию для каждого символа переданной строки splice и сохраняет значения в векторе prefix.

2) `bool KMP(std::string& t, std::string& p, std::vector<int>& pi, int flow_count = 1)`

Функция KMP() получает на вход текст, образец, вектор префикс-функции, число переданных потоков и вычисляет индексы вхождений образца.

3) `void printInterInfo(string& splice, std::vector<size_t> vect, int n)`

Функция printInterInfo() выводит промежуточные данные - состояние префикс-функции для текущей части строки (часть строки передается через n).

4) `void cyclicShift(std::string& strA, std::string& strB, size_t flowCount)`

Функция cyclicShift() проверяет правильность введенных строк, чтобы проверить их на цикличность.

ТЕСТИРОВАНИЕ.

Промежуточные данные в выводе тестов для удобства были исключены.

№	Input	Output
Тестирования алгоритма Кнута-Морриса-Пратта		
1	sas fasawsasdsasasa 1	5,9,11
2	abab asababsdababsdababababab 4	2,8,14,16,18
3	as abjkhgtipahggsyop 3	-1
4	asdfas a 5	Length error
5	a a 1	0
Тестирования алгоритма поиска циклического сдвига		
1	aaa aaaa	Strings have different size
2	aaba abaa	1
3	an asd	Strings have different size
4	afdafd	2

	dafdaf	
5	aaa bbb	-1

Вывод.

В ходе выполнения лабораторной работы был реализован алгоритм КМП для поиска подстроки в строке, а также алгоритм поиска циклического сдвига на базе алгоритма КМП.

Приложения А. Исходный код.

cyclic.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <cmath>
using namespace std;


void printInterInfo(std::string& splice, std::vector<size_t> vect, int n) {//вывод
информации о префикс функции строки
    std::cout << "\nState of prefix function:  " << std::endl;
    for (int i = 0; i <= n; i++)
        std::cout << splice[i] << " ";//вывод символов строки
    std::cout << std::endl;
    for (int i = 0; i <= n; i++)
        std::cout << vect[i] << " ";//вывод значений префикс-функции
    std::cout << std::endl << std::endl;
}


void prefixFunction(std::string& p, std::vector<size_t>& pi) {
    std::cout << "Start prefix function:" << std::endl;
    int n = 1;
    size_t j = 0;
    pi[0] = 0;

    for (size_t i = 1; i < p.size();)//проходим по склееной строке
    {
        // поиск какой префикс-суффикс можно расширить
        if (p[j] == p[i]) {//если можно увеличить

            pi[i] = j + 1;
            j++;
            i++;
            std::cout << "p[i] == p[j],  j++, i++,  pi[i] = " << j << " ,i=" << i;
            printInterInfo(p, pi, n);
            n++;
        }
        else {//если нельзя
            if (j == 0) {//это первый символ
                pi[i] = 0;
                i++;
                std::cout << "p[i] != p[j],  prefix[i] = 0, i=1, k=0";
                printInterInfo(p, pi, n);
                n++;

            }
            else {//если не первый символ
                j = pi[j - 1];
                std::cout << "p[i] != p[j], i= " << i << ", j= " << j;
                printInterInfo(p, pi, n);
            }
        }


    }
```

```
}

void KMP(std::string& t, std::string& p, size_t flowCount, bool isForShift) {//функция для
поиска шаблонов в тексте

    vector<size_t> res;//результат
    vector<size_t> pi(p.size());//значения префикс функции
    prefixFunction(p, pi);//считаем префикс функцию
    cout << endl;

    size_t j, start, end;
    if (t.length() / flowCount < p.length())
        flowCount = t.length() / p.length();//максимальное количесвто потоков

    for (size_t f = 0; f < flowCount; f++) {//обрабатываем текст по потокам

        cout << "Flow " << f + 1 << ": ";
        j = 0;
        size_t r = t.length() % flowCount;
        size_t flowLen = t.length() / flowCount;//длина потока
        if (r > 0)
            flowLen++;
        start = f * flowLen;//выделяем начальную позицию для текущего потока
        end = start + flowLen;//выделяем конечную позицию для текущего потока
        end += p.length() - 1;//прибавляем шаблон
        bool isFound = false;
        if (end > t.size())//если больше текста
            end = t.size();
        for (size_t s = start; s < end; s++)//вывод потока
            cout << t[s];
        cout << endl << endl;

        for (size_t i = start; i < end;) {//проходим по потоку
            for (size_t k = start; k < i; k++) {
                cout << t[k];
            }
            cout << " (" << t[i] << ")";
            for (size_t k = i + 1; k < end; k++) {
                cout << t[k];
            }
            cout << endl;
            string shift = "";
            for (size_t k = 0; k < i - start - j + 1; k++) {
                shift += " ";
            }


            cout << shift;
            for (size_t k = 0; k < p.size(); k++) {
                if (k == j) {
                    cout << "(" << p[k] << ")";
                }
                else cout << p[k];
            }

            if (t[i] == p[j]) {//если нашли одинаковые символы
                cout << endl << endl;
                i++;
                j++;
                if (j == p.size()) {//нашли вхождение
                    res.push_back(i - p.size());
                    isFound = true;
```

```
                    if(isForShift){
                        cout << "Ciclic shift found: i  - p.size(), i="<<i<<" ,size=
"<<p.size() << endl;
                        break;
                    }

                }
            }
            else {//если символы не совпали
                cout << " - symbols don't match." << endl << endl;
                if (j == 0) {//если первый символ
                    i++;
                }
                else {//если не первый
                    j = pi[j - 1];//откат
                }
            }
        }
        cout << endl ;
        if(isFound == false && f == flowCount - 1 && res.empty()){
            cout << "It is not cyclic shift" << endl;
            cout<<-1<<endl;
            return;
        }
    }
    if(res.empty()){
        cout << "It is not cyclic shift" << endl;
        cout<<-1<<endl;
        return;
    }

    string sep = "";
    for(auto &el : res){
        cout << sep << el;
        sep = ",";
    }

    return;
}


void cyclicShift(std::string& strA, std::string& strB, size_t flowCount) {


if (strA == strB) {
        cout << "Strings are equal" << endl;
        cout<<0<<endl;
        return;
    }

    if (strA.size() != strB.size()) {
        cout << "Strings have different size" << endl;
        cout<<-1<<endl;
        return;
    }    strA += strA;
    KMP(strA, strB, flowCount, true);

}

// в начале вводится шаблон, затем
// текст и число "потоков"
int main()
{
    string p, t;
```

```
    size_t flowCount;

    cin >> p;//шаблон
    cin >> t;//текст
    cin >> flowCount;//потоки
    cyclicShift(t, p, flowCount);
    return 0;
}
```

kmp.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <cmath>
using namespace std;


void printInterInfo(std::string& splice, std::vector<size_t> vect, int n) {//вывод
информации о префикс функции строки
    std::cout << "\nState of prefix function:  " << std::endl;
    for (int i = 0; i <= n; i++)
        std::cout << splice[i] << " ";//вывод символов строки
    std::cout << std::endl;
    for (int i = 0; i <= n; i++)
        std::cout << vect[i] << " ";//вывод значений префикс-функции
    std::cout << std::endl<<std::endl;
}


void prefixFunction(std::string& p, std::vector<size_t>& pi) {
    std::cout << "Start prefix function:" << std::endl;
    int n = 1;
    size_t j = 0;
    pi[0] = 0;

    for (size_t i = 1; i < p.size();)//проходим по склееной строке
    {
        // поиск какой префикс-суффикс можно расширить
        if (p[j] == p[i]) {//если можно увеличить

            pi[i] = j + 1;
            j++;
            i++;
            std::cout << "p[i] == p[j],  j++, i++,  pi[i] = " << j << " ,i=" << i;
            printInterInfo(p, pi, n);
            n++;
        }
        else {//если нельзя
            if (j == 0) {//это первый символ
                pi[i] = 0;
                i++;
                std::cout << "p[i] != p[j],  prefix[i] = 0, i=1, k=0";
                printInterInfo(p, pi, n);
                n++;

            }
            else {//если не первый символ
                j = pi[j - 1];
                std::cout << "p[i] != p[j], i= "<<i<<", j= "<<j;
                printInterInfo(p, pi, n);
            }
        }
```

```
        }


}

void KMP(std::string& t, std::string& p, size_t flowCount) {///функция для поиска шаблонов в
тексте

    vector<size_t> res;//результат
    vector<size_t> pi(p.size());//значения префикс функции
    prefixFunction(p, pi);//считаем префикс функцию
    cout << endl;

    size_t j, start, end;
    if (t.length() / flowCount < p.length())
        flowCount = t.length() / p.length();//максимальное количесвто потоков

    for (size_t f = 0; f < flowCount; f++) {//обрабатываем текст по потокам

        cout << "Flow " << f + 1 << ": ";
        j = 0;
        size_t r = t.length() % flowCount;
        size_t flowLen = t.length() / flowCount;//длина потока
        if (r > 0)
            flowLen++;
        start = f * flowLen;//выделяем начальную позицию для текущего потока
        end = start + flowLen;//выделяем конечную позицию для текущего потока
        end += p.length() - 1;//прибавляем шаблон
        bool isFound = false;
        if (end > t.size())//если больше текста
            end = t.size();
        for (size_t s = start; s < end; s++)//вывод потока
            cout << t[s];
        cout << endl << endl;

        for (size_t i = start; i < end;) {//проходим по потоку
            for (size_t k = start; k < i; k++) {
                cout << t[k];
            }
            cout << " (" << t[i] << ")";
            for (size_t k = i + 1; k < end; k++) {
                cout << t[k];
            }
            cout << endl;
            string shift = "";
            for (size_t k = 0; k < i - start - j + 1; k++) {
                shift += " ";
            }


            cout << shift;
            for (size_t k = 0; k < p.size(); k++) {
                if (k == j) {
                    cout << "(" << p[k] << ")";
                }
                else cout << p[k];
            }

            if (t[i] == p[j]) {//если нашли одинаковые символы
                cout << endl << endl;
                i++;
                j++;
                if (j == p.size()) {//нашли вхождение
```

```
                    cout << "Occurrence of the sample" << endl;
                    cout << "Index: ";
                    cout << i - p.size() << endl << endl;
                    res.push_back(i - p.size());//помещаем в результат
                    isFound = true;
                    j = pi[j - 1];//откатываемся
                }
            }
            else {//если символы не совпали
                cout << " - symbols don't match." << endl << endl;
                if (j == 0) {//если первый символ
                    i++;
                }
                else {//если не первый
                    j = pi[j - 1];//откат
                }
            }
        }
        cout << endl;
    }
    if (res.empty()) {
        cout << "Not found" << endl;
        cout << -1 << endl;
        return;
    }

    cout << "\nResult:" << endl;
    string sep = "";
    for (auto& el : res) {
        cout << sep << el;
        sep = ",";
    }
    cout << endl;

}


// в начале вводится шаблон, затем
// текст и число "потоков"
int main()
{
    string p, t;
    size_t flowCount;

    cin >> p;//шаблон
    cin >> t;//текст
    cin >> flowCount;//потоки
    KMP(t, p, flowCount);
    return 0;
}
```