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 ____ Хохлов Г.О.

Преподаватель ____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта и алгоритм циклического сдвига. Написать программу, реализующую эти алгоритмы работы со строками.

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

Алгоритм Кнута-Морриса-Пратта

Задание.

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

Описание алгоритма.

На вход алгоритма передается строка-образец, вхождения которой нужно найти, и строка-текст, в которой нужно найти вхождения.

Оптимизация алгоритма состоит в том, что строка-текст считывается посимвольно, в памяти хранится только текущий считанный символ.

Алгоритм сначала вычисляет префикс-функцию строки-образца, для каждого символа в строке вычисляется его значение префикс-функции и заносится в вектор значений.

Далее посимвольно считывается строка-текст. Переменная-счетчик изначально k = 0. При каждом совпадении k-го символа образца и l-го символа текста счетчик увеличивается на 1. Если k = размер образца, значит вхождение найдено. Результат заносится в вектор ответов, который хранит индексы вхождений. Если очередной символ текста не совпал с k-ым символом образца и если k был больше нуля, то происходит “откат” к концу предыдущего совпавшего префикса. Этот цикл повторяется до тех пор, пока счетчик l не пройдет по каждому символу строки.

Программа печатает в консоль найденные индексы строки-образца в строке поиска, также выводятся промежуточные результаты, такие как пошаговое вычисление префикс-функции строки и пошаговая работа алгоритма Кнута-Морриса-Пратта.

Сложность алгоритма по операциям: O (m + n), m – длина образца, n – длина текста.

Сложность алгоритма по памяти: O (m + n), m – длина образца, n – длина текста.

Описание функций и структур данных.

void prefixFunction (std::vector<int>& pi, const std::string& string)

Функция, заполняющая массив префикс-функций для работы алгоритма Кнута-Морриса-Пратта

pi – массив префикс-функций, который будет заполнен в результате работы функции.

string – строка, для который будет рассчитываться префикс-функция

std::vector<int> KMP(std::ifstream& input, const std::string& pattern, const std::string& text)

Функция реализующая алгоритм Кнута-Морриса-Пратта, находит индексы вхождения одной строки в другую

input – входной поток, из которого считываются символы строки для поиска в ней строки образца.

pattern – строка-образец, поиск которой будет осуществляться.

Text – текст, в котором необходимо определить вхождения образца.

Возвращаемым значением является вектор индексов вхождения строки-образца в строку, в которой осуществляется поиск.

Тестирование.

Входные данные:

ab

abab

Результат работы программы:

```
----------------------------------------
Calculating prefix function

a b
0 0
j i
s[i] != s[j], i++, pi[i] = 0


Prefix function for string:
a b
0 0

----------------------------------------
Finding substring in the string

Read symbol: a
a b
0 0
```

k
read symbol == pattern[k], k++

Read symbol: b
a b
0 0
  k
read symbol == pattern[k], k++

Found string entry in index: 0

Read symbol: a
a b
0 0
    k

read symbol != pattern[k], k = pi[k-1] = 0
a b
0 0
k
read symbol == pattern[k], k++

Read symbol: b
a b
0 0
  k
read symbol == pattern[k], k++

Found string entry in index: 2

Result of algorithm work:
0,2

Входные данные:

aba
abcabaabac

Результат работы программы:

```
----------------------------------------
Calculating prefix function

a b a
0 0 0
j i
s[i] != s[j], i++, pi[i] = 0

a b a
0 0 0
j   i
s[i] == s[j], i++, j++, pi[i] = 1


Prefix function for string:
a b a
0 0 1

----------------------------------------
Finding substring in the string

Read symbol: a
a b a
0 0 1
k
read symbol == pattern[k], k++

--------------------------------------
Read symbol: b
a b a
0 0 1
  k
read symbol == pattern[k], k++

--------------------------------------
Read symbol: c
a b a
0 0 1
    k

read symbol != pattern[k], k = pi[k-1] = 0
a b a
```

```
0 0 1
k
read symbol != pattern[k]

---------------------------------------
Read symbol: a
a b a
0 0 1
k
read symbol == pattern[k], k++

---------------------------------------
Read symbol: b
a b a
0 0 1
  k
read symbol == pattern[k], k++

---------------------------------------
Read symbol: a
a b a
0 0 1
    k
read symbol == pattern[k], k++

Found string entry in index: 3

---------------------------------------
Read symbol: a
a b a
0 0 1
      k

read symbol != pattern[k], k = pi[k-1] = 1
a b a
0 0 1
  k

read symbol != pattern[k], k = pi[k-1] = 0
a b a
0 0 1
k
read symbol == pattern[k], k++

---------------------------------------
```

Read symbol: b
a b a
0 0 1
  k
read symbol == pattern[k], k++

Read symbol: a
a b a
0 0 1
    k
read symbol == pattern[k], k++

Found string entry in index: 6

Read symbol: c
a b a
0 0 1
      k

read symbol != pattern[k], k = pi[k-1] = 1
a b a
0 0 1
  k

read symbol != pattern[k], k = pi[k-1] = 0
a b a
0 0 1
k
read symbol != pattern[k]

Result of algorithm work:
3,6

Циклический сдвиг.

Задание.

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 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

Описание алгоритма.

Для того, чтобы вычислить, является ли одна строка циклическим сдвигом другой, можно воспользоваться префикс функцией.

На вход алгоритму поступают две строки. Сначала алгоритмсравнивает размеры строк, если они не совпадают – строки не могут являться циклическим сдвигом.

Далее складывается первая строка с двумя вторыми (лексикографически), потом вычисляется префикс-функция от строки результата сложения. Вычисляется префикс-функция для каждого символа в строке и заносится в массив значений префикс-функций.

Строка результат имеет вид ABB (A – вторая строка, B – первая строка). Если в B_2 у какого-нибудь символа префикс-функция равна длине строки (size(A) = size(B) = prefix(i)), то строки являются циклическим сдвигом.

Программа печатает в консоль индекс циклического сдвига, также выводятся промежуточные результаты, такие как пошаговое вычисление префикс-функции получившейся строки и пошаговая работа алгоритма циклического сдвига.

Сложность алгоритма по операциям: O (4n), n – длина строки

Сложность алгоритма по памяти: O (3n), n – длина строки

Описание функций и структур данных.

int* prefixFunction(char* string, int size)

Функция, заполняющая массив префикс-функций для работы алгоритма

string – строка, для который будет рассчитываться префикс-функция

size – длина строки

Возвращаемым значение является указатель на массив префикс-функций

void cyclicShift(const std::string& firstString, const std::string& secondString)

Функция, реализующая алгоритм проверки циклического сдвига. Если одна строка, является циклическим сдвигом другой, то функция выводит индекс сдвига

firstString – первая строка

secondString – вторая строка

Тестирование.

Входные данные:

defabc

abcdef

Результат работы программы:

Calculating prefix function

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j i

s[i] != s[j], i++, pi[i] = 0

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j i

s[i] != s[j], i++, pi[i] = 0

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j i

s[i] != s[j], i++, pi[i] = 0

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j i

s[i] != s[j], i++, pi[i] = 0

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j   i

s[i] != s[j], i++, pi[i] = 0

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j   i

s[i] != s[j], i++, pi[i] = 0

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j   i

s[i] != s[j], i++, pi[i] = 0

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j   i

s[i] != s[j], i++, pi[i] = 0

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j   i

s[i] == s[j], i++, j++, pi[i] = 1

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

  j   i

s[i] == s[j], i++, j++, pi[i] = 2

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0

j i

s[i] == s[j], i++, j++, pi[i] = 3

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 2 3 0 0 0 0 0 0

j i

s[i] == s[j], i++, j++, pi[i] = 4

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 2 3 4 0 0 0 0 0

j i

s[i] == s[j], i++, j++, pi[i] = 5

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 2 3 4 5 0 0 0 0

j i

s[i] == s[j], i++, j++, pi[i] = 6

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 0 0 0

j i

s[i] != s[j], j = 0

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 0 0 0

j i

s[i] == s[j], i++, j++, pi[i] = 1

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 0 0

j i

s[i] == s[j], i++, j++, pi[i] = 2

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 2 0

j i

s[i] == s[j], i++, j++, pi[i] = 3

Prefix function for string:

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 2 3

Find cyclic shift

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 2 3

i

Cyclic shift still not found, size = 6, pi[i] = 3, size != pi[i]

Go next, i++

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 2 3

i

Cyclic shift still not found, size = 6, pi[i] = 4, size != pi[i]

Go next, i++

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 2 3

i

Cyclic shift still not found, size = 6, pi[i] = 5, size != pi[i]

Go next, i++

a b c d e f d e f a b c d e f a b c

0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 2 3

i

Found cyclic shift!!! Size = 6, pi[i] = 6

Cyclic shift index: 3

Входные данные:

abcdef

efabcd

Результат работы программы:

Calculating prefix function

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j i

s[i] != s[j], i++, pi[i] = 0

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j i

s[i] != s[j], i++, pi[i] = 0

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j   i

s[i] != s[j], i++, pi[i] = 0

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j   i

s[i] != s[j], i++, pi[i] = 0

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j   i

s[i] != s[j], i++, pi[i] = 0

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j   i

s[i] != s[j], i++, pi[i] = 0

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j   i

s[i] != s[j], i++, pi[i] = 0

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j   i

s[i] != s[j], i++, pi[i] = 0

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j i

s[i] != s[j], i++, pi[i] = 0

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

j i

s[i] == s[j], i++, j++, pi[i] = 1

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

j i

s[i] == s[j], i++, j++, pi[i] = 2

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 1 2 0 0 0 0 0 0

j i

s[i] == s[j], i++, j++, pi[i] = 3

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 1 2 3 0 0 0 0 0

j i

s[i] == s[j], i++, j++, pi[i] = 4

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 1 2 3 4 0 0 0 0

j i

s[i] == s[j], i++, j++, pi[i] = 5

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 0 0 0

j i

s[i] == s[j], i++, j++, pi[i] = 6

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 0 0

j i

s[i] != s[j], j = 0

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 0 0

j i

s[i] == s[j], i++, j++, pi[i] = 1

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 0

j i

s[i] == s[j], i++, j++, pi[i] = 2

Prefix function for string:

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 2

Find cyclic shift

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 2

i

Cyclic shift still not found, size = 6, pi[i] = 2, size != pi[i]

Go next, i++

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 2

i

Cyclic shift still not found, size = 6, pi[i] = 3, size != pi[i]

Go next, i++

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 2

i

Cyclic shift still not found, size = 6, pi[i] = 4, size != pi[i]

Go next, i++

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 2

i

Cyclic shift still not found, size = 6, pi[i] = 5, size != pi[i]

Go next, i++

e f a b c d a b c d e f a b c d e f

0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 1 2

i

Found cyclic shift!!! Size = 6, pi[i] = 6

Cyclic shift index: 4

Выводы.

В ходе выполнения лабораторной работы были получены навыки работы с алгоритмом Кнута-Морриса-Пратта и алгоритмом циклического сдвига. Были написаны программу, реализующую эти алгоритмы работы со строками.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД
АЛГОРИТМ КНУТА-МОРРИСА-ПРАТТА

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


void prefixFunction (std::vector<int>& pi, const std::string&
string) {        //Заполнение массива значениями префикс-функций
    std::cout << "----------------------------------------
"<<std::endl;
    std::cout << "Calculating prefix
function"<<std::endl<<std::endl;
    pi[0] = 0;
    int j = 0;            //Вспомогательный указатель на индекс
    int i = 1;            //Текущий указатель на индекс в строке


    while(i < string.size()){        //Пока не пройдены все символы
строки-поиска


        for (char k : string)            //Вывод промежуточных
результатов
            std::cout << k << " ";
        std::cout << std::endl;
        for (int k : pi)
            std::cout << k <<" ";
        std::cout << std::endl;
        for (int k=0; k<j; k++)
            std::cout << "  ";
        std::cout << "j ";
        for (int k=0; k<i-j-1; k++)
            std::cout << "  ";
        std::cout << "i " << std::endl;


        if (string[i] == string[j]){     //Если символы i и j индекса
совпадают, то оба смещаются вправо, изменяется массив префикс-функций
            pi[i] = j+1;
            i++;
            j++;
            std::cout << "s[i] == s[j],  i++,  j++,  pi[i] = " << j
<< std::endl << std::endl;
        }
        else{
            if (j==0){                    //Иначе, если j указывает на
начальный символ строки, то на i символ значение префикс-функции 0
```

```
                pi[i] = 0;
                i++;                        //i-тый указатель смещается
вправо
                std::cout << "s[i] != s[j],  i++,  pi[i] = 0" <<
std::endl << std::endl;
            }
            else{
                j = pi[j-1];            //Если указатель j не указывает на
начальный символ, то сравниваются префиксы и суффиксы меньшего размера
                std::cout << "s[i] != s[j],  j = " << pi[j] <<
std::endl << std::endl;
            }
        }

    }



    std::cout << "\nPrefix function for string:\n";
//Вывод строки со значением префикс-функций каждого символа
    for (char k : string)
        std::cout << k << " ";
    std::cout << std::endl;
    for (int i=0; i<string.size(); i++)
        std::cout << pi[i] << " ";
    std::cout <<"\n\n";
    std::cout << "----------------------------------------
"<<std::endl;
}



std::vector<int> KMP(std::ifstream& input, const std::string&
pattern, const std::string& text) {   //Функция, реализующая алгоритм
Кнута-Морриса-Пратта
    std::vector<int> result;                                    //Вектор
индексов вхождения подстроки
    std::vector<int> pi(pattern.size());                        //Вектор
значений префикс-функций


    prefixFunction(pi, pattern);                               //Заполнение
вектора префикс-функций
    std::cout << "Finding substring in the
string"<<std::endl<<std::endl;


    int k = 0;                              //Указатель на текущий элемент в
строке-шаблоне
    int l = 1;                              //Счетчик символов в строке
поиска
    int i = 0;


    while (i < text.size()) {
```

```
        char ch = text[i++];                    //Считывается очередной
символ
        std::cout << "Read symbol: " << ch << std::endl;    //Вывод
промежуточных значений
        for (char i : pattern)
            std::cout << i << " ";
        std::cout << std::endl;
        for (int i : pi)
            std::cout << i <<" ";
        std::cout << std::endl;
        for (int i=0; i<k; i++)
            std::cout << "  ";
        std::cout << "k" << std::endl;



        while ((k > 0) && (pattern[k] != ch)) {             //Если
считанный символ не удовлетворяет условию, то изменяется указатель k
            k = pi[k-1];

            std::cout << "\nread symbol != pattern[k], k = pi[k-1] =
" << k << std::endl;
            for (char i : pattern)
                std::cout << i << " ";
            std::cout << std::endl;
            for (int i : pi)
                std::cout << i <<" ";
            std::cout << std::endl;
            for (int i=0; i<k; i++)
                std::cout << "  ";
            for (int i=0; i<k-1; i++)
                std::cout << "  ";
            std::cout << "k" << std::endl;
        }

        if (pattern[k] == ch) {          //Если считанный символ совпадает
с текущим в строке-шаблоне, то указатель k сдвигается вправо
            k++;
            std::cout << "read symbol == pattern[k], k++" <<
std::endl << std::endl;
        }
        else{
            std::cout << "read symbol != pattern[k]"<< std::endl
<<std::endl;
        }

        if (k == pattern.size()) {           //Если найдена подстрока, то
индекс заносится в массив результата
            result.push_back(l - pattern.size());
            std::cout << "Found string entry in index: " << l -
pattern.size() << std::endl<< std::endl;
```

```
        }
        l++;
        std::cout << "---------------------------------------" <<
std::endl;
    }

    if (result.empty()) {                    //Если не было найдено
подстроки
        result.push_back(-1);
    }

    return result;
}



int main() {
    std::ifstream input;
    input.open("test.txt");
    std::string pattern;
    std::string text;
    input >> pattern >> text;
    auto result = KMP(input, pattern, text);          //Запуск
алгоритма Кнута-Морриса-Пратта

    std::cout << "\n\nResult of algorithm work:" << std::endl;
//Вывод результата
    if (result[0] == -1)
        std::cout << "Substring not found" << std::endl;
    else {
        for (int i = 0; i < result.size(); i++) {
            std::cout << result[i];
            if (i != result.size() - 1) {
                std::cout << ',';
            }
        }
    }

    input.close();

    return 0;
}
```

АЛГОРИТМ ЦИКЛИЧЕСКОГО ПОИСКА

```
#include <iostream>
#include <fstream>


int* prefixFunction(char* string, int size) {
//Заполнение массива значениями префикс-функций
    std::cout << "----------------------------------------
"<<std::endl;
    std::cout << "Calculating prefix function"<<std::endl<<std::endl; int*
    pi = new int[size];                          //Массив со значениями
префикс функций
    for (int i=0; i<size; i++) pi[i]
        = 0;
    int j = 0;                                  //Вспомогательный
указатель на индекс
    int i = 1;                                  //Текущий указатель на
индекс в строке

    while(i < size) {                           //Пока не пройдены все
символы строки-поиска

        for (int k = 0; k<size; k++)               //Вывод
промежуточных результатов
            std::cout << string[k] << " ";
        std::cout << std::endl;
        for (int k = 0; k<size; k++)
            std::cout << pi[k] <<" ";
        std::cout << std::endl; for
        (int k=0; k<j; k++)
            std::cout << "  ";
        std::cout << "j ";
        for (int k=0; k<i-j-1; k++)
            std::cout << "  ";
        std::cout << "i " << std::endl;

        if (string[i] == string[j]){                    //Если символы i и j
индекса совпадают, то оба смещаются вправо, изменяется массив префикс-
функций
            pi[i] = j+1; i++;
            j++;
            std::cout << "s[i] == s[j],  i++,  j++,  pi[i] = " << j <<
std::endl << std::endl;
        }
        else{
            if (j==0){                                  //Иначе, если j
указывает на начальный символ строки, то на i символ значение префикс-
функции 0
                pi[i] = 0;
                i++;                                     //i-тый
```

```
указатель смещается вправо
                std::cout << "s[i] != s[j],  i++,  pi[i] = 0" <<
std::endl << std::endl;
            }
            else{
                j = pi[j-1];                                //Если
указатель j не указывает на начальный символ, то сравниваются префиксы
и суффиксы меньшего размера
                std::cout << "s[i] != s[j],  j = " << pi[j] <<
std::endl << std::endl;
            }
        }
    }

    std::cout << "\nPrefix function for string:\n";          //Вывод
строки со значением префикс-функций каждого символа
    for (int i=0; i<size; i++)
        std::cout << string[i] << " ";
    std::cout << std::endl;
    for (int i=0; i<size; i++)
        std::cout << pi[i] << " ";
    std::cout <<"\n\n";
    std::cout << "----------------------------------------
"<<std::endl;

    return pi;
}


void cyclicShift(const std::string& firstString, const std::string&
secondString) {    //Функция поиска циклического сдвига
    if (firstString.size() != secondString.size()) {
//Проверка на равенство длины двух строк
        std::cout << -1;
//Если не равны, то одна не может быть циклическим сдвигом другой
        return;
    }

    int size = firstString.size();

    char* buff = new char[3 * size];                          //Создание
вспомогательной строки для поиска результата

    int i = 0;
    for (; i < size; ++i) {
        buff[i] = secondString[i];
//Заносится первая введенная строка
    }

    for (int k = 0; k < 2; ++k) {                             //Далее
дважды заносится вторая введенная строка
        for (int j = 0; j < size; ++j) {
            buff[i++] = firstString[j];
        }
```

```
    }

    int* pi = prefixFunction(buff, size * 3);           //Получается
массив их префикс функций

    std::cout << "Find cyclic shift" << std::endl<<std::endl;
    for (int i = 2 * size - 1; i < 3 * size; ++i) {
//Обходится вспомогательная строка

        for (int k = 0; k<size*3; k++)                //Вывод
промежуточных результатов
            std::cout << buff[k] << " ";
        std::cout << std::endl;
        for (int k = 0; k<size*3; k++)
            std::cout << pi[k] <<" ";
        std::cout << std::endl;
        for (int k=0; k<i; k++)
            std::cout << "  ";
        std::cout << "i ";

        if (pi[i] == size) {                                  //Если
значение префикс функции на символе равно размеру введенной строки, то
найден циклический сдвиг
            std::cout << "\nFound cyclic shift!!! Size = " << size <<
", pi[i] = " <<pi[i] <<std::endl;
            std::cout <<"Cyclic shift index: "<< i + 1 - 2 * size;
//Вывод результата
            delete [] buff;
            delete [] pi;
            return;
        }

        else{
            std::cout << "\nCyclic shift still not found, size = " <<
size << ", pi[i] = " <<pi[i] << ", size != pi[i]"<<std::endl;
            std::cout << "Go next, i++" << std::endl << std::endl;
        }
    }

    std::cout << -1;                                //Если циклический
сдвиг не найден
    delete [] buff;
    delete [] pi;
}


int main() {
    std::ifstream input;
    input.open("test.txt");
    std::string firstString;
    std::string secondString;

    input >> firstString >> secondString;       //Считывание двух
строк, для поиска циклического сдвига
```

```
    cyclicShift(firstString, secondString);  //Запуск алгоритма поиска
циклического сдвига

    return 0;
}
```