

**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

\_\_\_\_\_

Стукалев А.И.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2020

### **Цель работы.**

Изучить и реализовать на языке программирования C++ алгоритм Кнута-Морриса-Пратта поиска подстроки в строке, также с помощью этого алгоритма определить является ли одна строка циклическим сдвигом другой строки.

### **Индивидуализация.**

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

### **Формулировка задания для КМП.**

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 для КМП.**

ab

abab

### **Соответствующие выходные данные для КМП.**

0,2

### **Формулировка задания для алгоритма по определению циклического сдвига.**

Заданы две строки  $A(|A| \leq 5000000)$  и  $B(|B| \leq 5000000)$ .

Определить, является ли  $A$  циклическим сдвигом  $B$  (это значит, что  $AA$  и  $BB$  имеют одинаковую длину и  $AA$  состоит из суффикса  $BB$ , склеенного с

префиксом В). Например, defabc является циклическим сдвигом abcdef.

Вход:

Первая строка - A

Вторая строка - ВВ

Выход:

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

**Пример входных данных для алгоритма по определению циклического сдвига.**

defabc

abcdef

**Соответствующие выходные данные для алгоритма по определению циклического сдвига.**

3

**Описание алгоритмов.**

**КМП:**

Рассмотрим сравнение строк на позиции , где образец сопоставляется с частью текста . Предположим, что первое несовпадение произошло между и где . Тогда и .

При сдвиге вполне можно ожидать, что префикс (начальные символы) образца сойдется с каким-нибудь суффиксом (конечные символы) текста . Длина наиболее длинного префикса, являющегося одновременно суффиксом, есть значение префикс-функции от строки для индекса .

Это приводит нас к следующему алгоритму: пусть — значение префикс-функции от строки для индекса . Тогда после сдвига мы можем возобновить сравнения с места и без потери возможного местонахождения образца. Можно показать, что таблица может быть вычислена (амортизационно) за сравнений перед началом поиска. А

поскольку строка будет пройдена ровно один раз, суммарное время работы алгоритма будет равно  $O(n)$ , где  $n$  — длина текста.

### **Алгоритм для нахождения циклического сдвига.**

В данном алгоритме можно обойтись без удваивания строки. В самом начале происходит проверка на соответствие длин строк. Если соответствия не было обнаружено, то выводится -1. Инициализируются два счётчика для первой и второй строки. Далее сравниваются символы первой и второй строки, если символы совпадают переход к следующим, счётчики увеличиваются, если совпадения не обнаружено, счётчик для второй строки уменьшается. В том случае, если счётчик второй строки равен её длине, то сдвиг найден, а если счётчик первой строки равен её длине, то происходит его обнуление, таким образом строка зацикливается и можно обойтись без удвоения строки.

### **Префикс функция.**

Префикс-функция от строки и позиции в ней — длина наибольшего собственного префикса подстроки, который одновременно является суффиксом этой подстроки.

То есть, в начале подстроки длины нужно найти такой префикс максимальной длины, который был бы суффиксом данной подстроки.

Например, для строки префикс-функция будет такой: .

Сложность алгоритма КМП по операциям:  $O(n + m)$ ,  $n$  — длина подстроки,  $m$  — длина строки.

Сложность алгоритма поиска циклического сдвига:  $O(n + n) = O(n)$ .

Сложность алгоритма КМП по памяти:  $O(m)$ , если не учитывать память, в которой хранится строка поиска.

Сложность алгоритма поиска циклического сдвига по памяти  $O(2n) = O(n)$ .

## **Описание структур данных алгоритма КМП и алгоритма нахождения циклического сдвига.**

1.

```
class SubStr
{
    std::vector <int> ind_lengths;
    std::string input_string;
}
```

Класс необходимый для работы алгоритма КМП, input\_string – строка-образец, ind\_lengths – массив для префиксов.

2.

```
class Cycle
{
    std::vector <int> ind_lengths;
    std::string first_string;
    std::string second_string;
}
```

Класс, необходимый для работы алгоритма поиска циклического сдвига. first\_string – первая строка, second\_string – вторая строка, ind\_lengths – массив префиксов для first\_string.

## **Описание функций КМП.**

1. void prefix\_func()

Функция класса SubStr вычисления префикс функции для input\_string и записи результата в ind\_lengths.

2. void KMP()

Функция класса SubStr нахождения подстроки в строке. Посимвольно считывает строку, в которой необходимо проводить поиск, имеет два счётчика – счётчик нахождения в input\_string (j) и счётчик количества введённых символов(i),

необходимый для вычисления позиция вхождения подстроки в строку. С каждым добавленным элементом второй счётчик увеличивается, первый же счётчик увеличивается лишь в том случае, если произошло равенство считанного элемента и `input_string[j]`. Если `j` равняется длине `input_string`, то подстрока найдена и результат выводится в консоль, `j` становится равным `ind_lenghts[j-1]`. Если считанный элемент и `input_string[j]` не равны, `j` становится равным `ind_lenghts[j-1]`. Алгоритм завершает работу, когда следующий символ считать не возможно.

### 3. `void input_data()`

Функция класса `SubStr`. Происходит считывание строки в `input_string` и инициализация нулями `ind_lenghts`

#### **Описание функций алгоритма нахождения циклического сдвига.**

### 1. `void prefix_func()`

Функция класса `SubStr` вычисления префикс функции для `first_string` и записи результата в `ind_lenghts`.

### 2. `void cycle()`

Функция класса `Cycle` для нахождения циклического сдвига. Затем происходит проверка на соответствие длин строк. Если соответствия не было обнаружено, то выводится -1. Инициализируются два счётчика для первой и второй строки. Далее сравниваются символы первой и второй строки, если символы совпадают переход к следующим, счётчики увеличиваются, если совпадения не обнаружено, счётчик для второй строки уменьшается, в том случае, если он не равен нулю. В том случае, если счётчик второй строки равен её длине, то сдвиг найден, а если счётчик первой строки равен её длине, то происходит его обнуление, таким образом строка зацикливается и можно обойтись без удвоения строки, во избежания бесконечного цикла проход по строке осуществляется не более двух раз.

### 3. `void input_data()`

Функция класса `Cycle`. Происходит считывание первой строки в `first_string`, считывание второй строки в `second_string`.

## Способ хранения частичных решений.

Частичные решения, т.е. значения префикс функции, в массиве `ind_lengths`.

## Тестирование.

### КМП

#### 1. Тестирование входных данных из примера

```
D:\Qt\Tools\QtCreator\
ab
abab
0, 2
```

#### 2. Тестирование на несовпадение.

```
D:\Qt\Tools\QtCreator\bin\qt
aaaaaaaaaabb
aaaaaaaaaaaa
-1
```

#### 3. Тестирование на неоднократные вхождения

```
D:\Qt\Tools\QtCreator\bin\qtcreator_
ba
abababaaabababbabaabbbaaa
1, 3, 5, 9, 11, 14, 16, 21
```

#### 4. Тестирование на несовпадение.

```
D:\Qt\Tools\QtCreator\bin\
baaa
baabbbabababaabbab
-1
```

5.

```
D:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
gts
Value function prefix: 0 0 0
abcdefghgtsabcdefghgts
Non equal:      g[0]!=a[0]
Non equal:      g[0]!=b[1]
Non equal:      g[0]!=c[2]
Non equal:      g[0]!=d[3]
Non equal:      g[0]!=e[4]
Non equal:      g[0]!=f[5]
Equal elements: g[0]==g[6]
Non equal:      t[1]!=h[7]
Non equal:      g[0]!=h[7]
Equal elements: g[0]==g[8]
Equal elements: t[1]==t[9]
Equal elements: s[2]==s[10]
Sub string founded, position: 8
Non equal:      g[0]!=a[11]
Non equal:      g[0]!=b[12]
Non equal:      g[0]!=c[13]
Non equal:      g[0]!=d[14]
Non equal:      g[0]!=e[15]
Non equal:      g[0]!=f[16]
Non equal:      g[0]!=h[17]
Equal elements: g[0]==g[18]
Equal elements: t[1]==t[19]
Equal elements: s[2]==s[20]
Sub string founded, position: 18
```

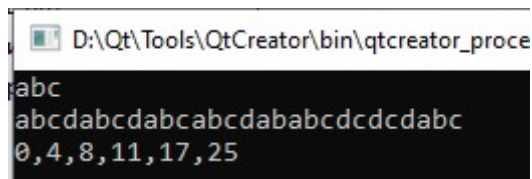
Для введённой подстроки значение префикс функции 000, далее происходит посимвольное считывание строки, и на каждом шаге проверяется соответствие символа строки и подстроки. В данном случае первое соответствие обнаруживается на 6 символе строки, если считать с нуля, счётчик подстроки увеличивается

(0→1), но уже на следующей итерации уменьшается (0), так как следующий символ строки (7) и подстроки (1) не совпадают. Затем сверяются 0-ой символ подстроки и 7-ой символ строки, совпадения также обнаружено не было. Поиск продолжается и совпадение обнаруживается на 8-ом символе строки, счётчик подстроки увеличивается (0→1), счётчик строки также увеличивается. На следующем шаге совпадение снова обнаруживается: 1-ый символ подстроки совпал с 9-ым символом строки, счётчики увеличиваются, аналогично и для следующего символа. Так как счётчик подстроки дошёл до её конца, значит подстрока найдена и происходит вывод позиции начала вхождения подстроки



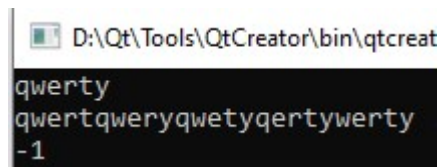
в строку. Аналогично подстрока находится начиная с 18-го символа строки.

#### 6. Тестирование на неоднократное вхождение.



```
D:\Qt\Tools\QtCreator\bin\qtcreator_proce
abc
abcdabcdabcdabcdabcdcdcdabc
0,4,8,11,17,25
```

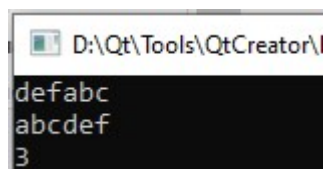
#### 7. Тестирование на проверку всех символов подстроки



```
D:\Qt\Tools\QtCreator\bin\qtcreat
qwerty
qwertyqwertyqwertyqwerty
-1
```

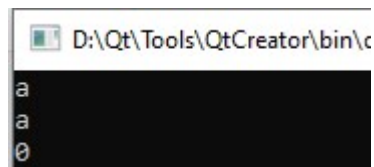
### Алгоритм нахождения циклического сдвига

#### 1. Тестирование входных данных из примера



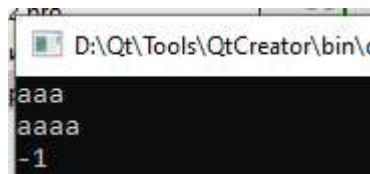
```
D:\Qt\Tools\QtCreator\I
defabc
abcdef
3
```

#### 2. Тестирование случая совпадения из одного символа



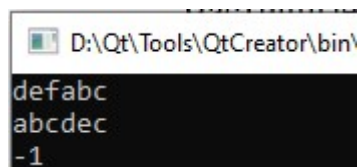
```
D:\Qt\Tools\QtCreator\bin\c
a
a
0
```

#### 3. Тестирование на строках разной длины



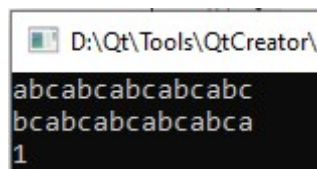
```
D:\Qt\Tools\QtCreator\bin\c
aaa
aaaa
-1
```

#### 4. Тестирование на отсутствие циклического сдвига



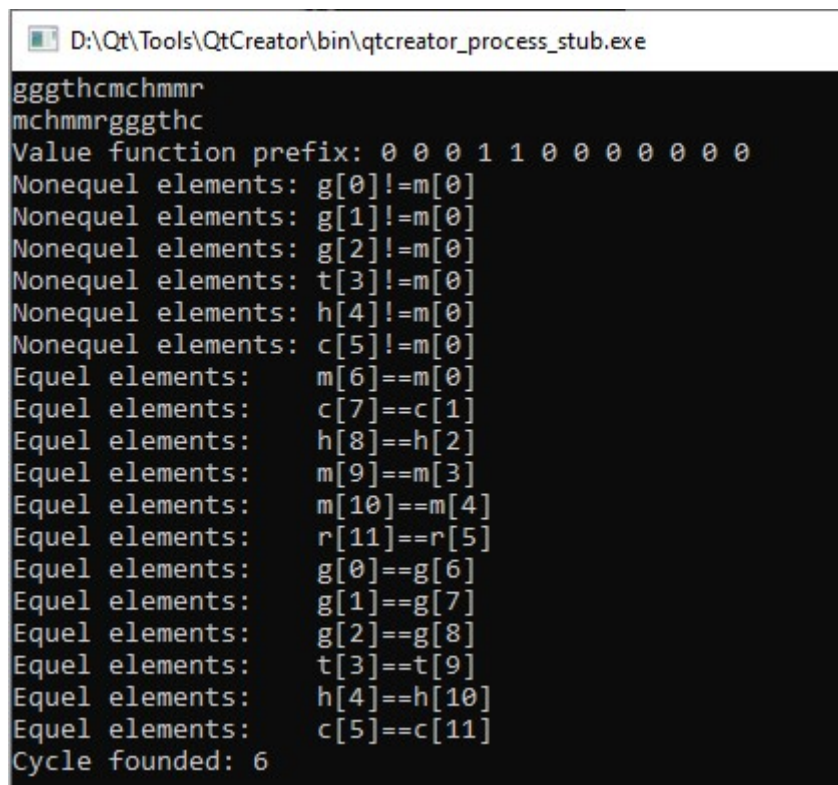
```
D:\Qt\Tools\QtCreator\bin\
defabc
abcdec
-1
```

#### 5. Тестирования при не одном циклическом сдвиге



```
D:\Qt\Tools\QtCreator\I
abcabcabcabcabc
bcabcabcabcabca
1
```

6.



```
D:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
gggthcmchmmr
mchmmrgggthc
Value function prefix: 0 0 0 1 1 0 0 0 0 0 0 0
Nonequel elements: g[0]!=m[0]
Nonequel elements: g[1]!=m[0]
Nonequel elements: g[2]!=m[0]
Nonequel elements: t[3]!=m[0]
Nonequel elements: h[4]!=m[0]
Nonequel elements: c[5]!=m[0]
Equal elements: m[6]==m[0]
Equal elements: c[7]==c[1]
Equal elements: h[8]==h[2]
Equal elements: m[9]==m[3]
Equal elements: m[10]==m[4]
Equal elements: r[11]==r[5]
Equal elements: g[0]==g[6]
Equal elements: g[1]==g[7]
Equal elements: g[2]==g[8]
Equal elements: t[3]==t[9]
Equal elements: h[4]==h[10]
Equal elements: c[5]==c[11]
Cycle founded: 6
```

Для второй строки значение префикс функции: 000110000000. Алгоритм ищет соответствие между  $i$ -ым (изначально 0) символом первой и 0-ым символом второй строки, как видно по скриншоту, проходясь по первой строке (увеличивая  $i$ ), в данном случае первое совпадение обнаруживается на 6. На всех последующих итерациях, счётчики для первой и второй строки увеличиваются, до тех пор, пока счётчик второй строки не станет равным её длине. Это значит, что цикл найден и происходит вывод начала второй строки в первой строке.

## Выводы

В ходе выполнения лабораторной работы были изучены и реализованы на языке программирования C++ алгоритм КМП нахождения подстроки в строке и алгоритм нахождения циклического сдвига. Также алгоритм КМП был модифицирован в соответствии с индивидуализацией, для этого хранились только подстрока и массив со значениями префикс функции для подстроки, а строка, в которой необходимо было производить поиск, считывалась

ПОСИМВОЛЬНО.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД

#### КМП.

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

class SubStr
{
    std::vector <int> ind_lengths;
    std::string input_string;
public:
    SubStr(){}
    void prefix_func()
    {
        //std::cout << "Value function prefix: 0 ";
        for (size_t i=1; i<input_string.length(); ++i)
        {
            //ищем, какой префикс можно расширить
            size_t j = ind_lengths[i-1]; //длина предыдущего префикса, может быть нулевой
            while ((j > 0) && (input_string[i] != input_string[j])) //если нельзя расширить
                j = ind_lengths[j-1]; //уменьшаем значение префикса

            if (input_string[i] == input_string[j])
                ++j; // расширяем найденный префикс
            ind_lengths[i] = j;
            //std::cout << j << ' ';

        }
        //std::cout << "\n";
    }
}

void input_data()
{
    std::cin >> input_string;
    std::vector <int> tmp(input_string.length()); //инициализируем нулями вектор длин префиксов
    ind_lengths = tmp;
}

void KMP()
{
    input_data();
    char c;
    int j = 0;
    int i = 0;
    bool no_one_digit = false;
    prefix_func(); //считается префикс функция для образца
    std::cin >> c;
    while(!std::cin.fail()) //пока возможно считать символ
    {
        if(input_string[j] == c) //соответствие найдено
        {
            //std::cout << "Equal elements:  " << input_string[j] << "[" << j << "]" << "==" << c << "[" << i << "]"
```

```

<< "\n";
    j++;
    i++;
    std::cin >> c;
}
if(j == input_string.length())//подстрока найдена
{
    //std::cout << "Sub string founded, position: ";
    if(no_one_digit)//если более одной цифры, то между ними ставится запятая
        std::cout << ',';
    std::cout << i - j << "\n";
    j = ind_lengths[j-1];
    no_one_digit = true;
}
else if(input_string[j] != c && !std::cin.fail())//совпадения не обнаружено или конец ввода
{
    //std::cout << "Non equal:      " << input_string[j] << "[" <<j << "]" << "!=" << c << "[" << i << "]" <<
"\n";
    if(j != 0)
        j = ind_lengths[j - 1];
    else
    {
        i++;
        std::cin >> c;
    }
}

}
if(!no_one_digit)//если подстроки найдено не было
    std::cout << -1;
}
};

int main()
{
    SubStr tmp;
    tmp.KMP();
}

```

## Алгоритм нахождения циклического сдвига.

```

#include <iostream>
#include <algorithm>
#include <vector>
#include <string>

class Cycle
{
    std::vector<int> ind_lengths;
    std::string first_string;
    std::string second_string;
public:
    Cycle() {}
    void prefix_func()
    {

```

```

//std::cout << "Value function prefix: 0 ";
for (size_t i=1; i<second_string.length(); ++i)
{
    // ищем, какой префикс можно расширить
    size_t j = ind_lengths[i-1]; //длина предыдущего префикса, может быть нулевой
    while ((j > 0) && (second_string[i] != second_string[j])) //если нельзя расширить,
        j = ind_lengths[j-1]; //уменьшаем значение префикса

    if (second_string[i] == second_string[j])
        ++j; //расширяем найденный префикс
    ind_lengths[i] = j;
    //std::cout << j << ' ';
}
//std::cout << "\n";
}
void input_data()
{
    std::cin >> first_string;
    std::cin >> second_string;
}
void cycle()
{
    input_data();
    if(first_string.length() != second_string.length())//если длины строк не равны - выход
    {
        //std::cout << "Non-equal lengths: " << first_string.length() << "!=" << second_string.length() << "\n";
        std::cout << -1;
        return;
    }
    std::vector<int> tmp(second_string.length());
    ind_lengths = tmp;
    this->prefix_func();// вычисление префикс функции для second_string
    int laps = 0;
    for (int ind_f = 0, ind_s = 0;;)
    {
        if(first_string[ind_f] == second_string[ind_s])
        {
            //std::cout << "Equal elements:  " << first_string[ind_f] << "[" << ind_f << "]" << "==" <<
second_string[ind_s] << "[" << ind_s << "]" << "\n";
            ind_f++;
            ind_s++;
        }
        if(ind_f == first_string.length())
        {
            ind_f = 0;
            laps++;
        }
        if(ind_s == first_string.length())
        {
            //std::cout << "Cycle founded: ";
            std::cout << ind_f;
            return;
        }
        else if(first_string[ind_f] != second_string[ind_s] && ind_s < first_string.length())
        {
            //std::cout << "Nonequal elements: " << first_string[ind_f] << "[" << ind_f << "]" << "!=" <<
second_string[ind_s] << "[" << ind_s << "]" << "\n";

```

```
        if(ind_s == 0)
            ind_f++;
        else
            ind_s = ind_lenghts[ind_s - 1];
    }
    if(laps > 1)
        break;
    }
    std::cout << -1;
}
};
```

```
int main()
{
    Cycle tmp;
    tmp.cycle();
}
```