

**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. 9382

Преподаватель

\_\_\_\_\_

\_\_\_\_\_

Савельев И.С.

Фирсов М.А.

Санкт-Петербург

2021

**Цель работы.**

На практике ознакомиться с алгоритмом Кнут-Моррис-Пратт.

**Задание.****Stepik 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  $-1$

**Sample Input:**

ab  
abab

**Sample Output:**

0,2

**Stepik 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.

**Sample Input:**

```
defabc  
abcdef
```

**Sample Output:**

3

**Описание алгоритма.**

Stepic 1

Алгоритм на вход получает две строки, нужно найти вхождение первой строки во вторую. Алгоритм начинается с вычисления префикс функции для первой строки. Префикс функция для  $i$ -го символа показывает максимальную длину совпадающего префикса и суффикса подстроки, которая заканчивается  $i$ -м символом. Затем мы посимвольно начинаем сравнивать две строки, если при сравнении все символы первой строки совпали со второй, то мы нашли вхождение записываем индекс начала вхождения в массив. Если при сравнении первый символ первой строки не совпал с первым символом второй строки, то мы сравниваем первый символ первой строки со вторым символом второй строки и так далее. Если при сравнении не первый символ первой строки не совпадает с символом второй строки, то следующим символом с которым мы начнем сравнивать первую и вторую строку будет символ первой строки под индексом префикс функции предыдущего символа. Алгоритм закончится когда мы сравним все символы строки.

Stepic 2

Во втором задании происходит проверка того, является ли первая строка циклическим сдвигом второй. Для этого используется тот же алгоритм, что и для stepic 1, только поиск ведется в удвоенной первой строке, так как при

сложении строк первая будет содержать в себе вторую строку, если она является циклическим сдвигом. Алгоритм заканчивается при нахождении первого вхождения или при рассмотрении всех символом удвоенной первой строки или если строки изначально не равны.

### **Функции и структуры данных.**

`std::vector<int> create_prefix(std::string text)` - принимает на вход строку, по которой формирует массив значений префикс функции символов входной строки. Возвращает этот массив

#### **Stepik 1**

`void kmp(std::string sample, std::string text, std::vector<int> prefix_arr, std::vector<int>& answer)` - функция реализует алгоритм Кнута-Морриса-Пратта, на вход принимает строку шаблон - `sample`, текст - `text`, массив значений префикс функции - `prefix_arr` и массив в котором будет храниться ответ - `answer`. Функция ничего не возвращает.

#### **Stepik 2**

`int is_cycle(std::string string_a, std::string string_b)` - функция проверяет является ли строка `string_a`, циклическим сдвигом строки `string_b`. Если `string_a` является циклическим сдвигом `string_b`, функция возвращает индекс начала строки `string_b` в `string_a`, иначе возвращает `-1`.

### **Оценка сложности по памяти и времени.**

В обеих программах сложность по памяти составляем  $O(m + n)$ , где  $m$  - длина первой строки,  $n$  - длина второй строки.

Сложность по времени также будет равна  $O(m + n)$ , где  $m$  - длина первой строки,  $n$  - длина второй строки.

### **Тестирование.**

Таблица 1 результаты тестирования программы для поиска вхождений одной строки в другую(stepik 1).

Входные данные	Вывод программы
ab abab	0,2
fgd gfdasdbfs	-1
zxc afafazxcdgqzxc	5,11

Таблица 2 результаты тестирования программы определяющий циклический сдвиг(stepik 2).

Входные данные	Вывод программы
asdfgh ghasdf	4
asfs afag	-1

### **Выводы.**

В ходе выполнения данной лабораторной работы был реализован алгоритм Кнута-Морриса-Пратта на языке программирования C++.

## ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ.

code lab4\_step1.cpp

```
#include <iostream>
#include <vector>

// Вычисление префикс функции
std::vector<int> create_prefix(std::string text) {
    int text_length = text.length();
    std::vector<int> prefix_arr(text_length);
    prefix_arr[0] = 0;
    std::cout << "\nЗначение префикс-функции для символа под номером: "
    << 0
        << " (" << text[0] << ")"
        << " = " << 0 << '\n';
    for (int i = 1; i < text_length; i++) {
        int current_length = prefix_arr[i - 1];
        std::cout
        << "\nВычисление значения префикс-функции для символа под
номером: "
        << i << " (" << text[i] << ")" << '\n';
        // Если предыдущий суффикс нельзя расширить, нужно рассмотреть
суффикс
        // меньшего размера prefix_arr[current_length - 1]
        while (current_length > 0 && (text[current_length] != text[i])) {
            std::cout << "Предыдущий суффикс размера " << current_length
                << " нельзя расширить" << '\n';
            current_length = prefix_arr[current_length - 1];
            std::cout << "Рассмотрим новый суффикс меньшего размера: "
                << current_length << '\n';
        }
        // Проверяем можно ли расширить
        if (text[current_length] == text[i]) {
            std::cout << "Суффикс длины " << current_length << " можно
расширить"
                << '\n';
            current_length++;
            std::cout << "Новый размер суффикса: " << current_length << '\n';
        }
        std::cout << "Значения префикс-функции равно " << current_length <<
'\n';
        // заносим соответствующие значение в массив
        prefix_arr[i] = current_length;
    }

    std::cout << "\nЗначения префикс-функции:" << '\n';
    // Выводим символы текста
    for (auto j : text) {
        std::cout << j;
        std::cout.width(3);
    }
}
```

```

    }
    std::cout << '\n';
    // Выводим соответствующие значения
    for (auto i : prefix_arr) {
        std::cout << i;
        std::cout.width(3);
    }
    std::cout << "\n\n";
    return prefix_arr;
}

void kmp(std::string sample, std::string text, std::vector<int>
prefix_arr,
        std::vector<int>& answer) {
    int text_size = text.size();
    int sample_i = 0;
    int text_i = 0;
    std::cout << "Алгоритм Кнута-Морриса-Пратта"
        << "\n\n";
    while (text_i < text_size) {
        // если символы совпали
        if (text[text_i] == sample[sample_i]) {
            std::cout << "Найдено совпадение " << sample_i << " символа образца "
                << "(" << sample[sample_i] << ") "
                << "и " << text_i << " символа текста "
                << "(" << text[text_i] << ") " << '\n';
            sample_i++;
            text_i++;
            // если нашли вхождение
            if (sample_i == sample.size()) {
                std::cout << "Нашли вхождение, индекс равен " << text_i - sample_i
                    << "\n\n";
                answer.push_back(text_i - sample_i);
                // переходим на позицию равную предпоследнему значению префикс
                // функции
                sample_i = prefix_arr[sample_i - 1];
            }
            // если сравнение было с первым символом
            else if (sample_i == 0) {
                text_i++;
            }
            // если по образцу продвинулись
            else {
                sample_i = prefix_arr[sample_i - 1];
            }
        }
    }
}

int main() {

```

```

std::vector<int> answer;
std::string string_a;
std::string string_b;
std::cout << "Введите строку A" << '\n';
std::cin >> string_a;
std::cout << "Введите строку B" << '\n';
std::cin >> string_b;

kmp(string_a, string_b, create_prefix(string_a), answer);
std::cout << "" << '\n';
bool flag = true;
if (!answer.size()) {
    std::cout << -1;
} else {
    // ВЫВОДИМ ОТВЕТ
    for (auto a : answer) {
        // если не первый символ из массива
        if (flag) {
            flag = false;
        } else {
            std::cout << ",";
        }
        std::cout << a;
    }
}

return 0;
}

code lab4_step2.cpp

#include <iostream>
#include <vector>

// Вычисление префикс функции
std::vector<int> create_prefix(std::string text) {
    int text_length = text.length();
    std::vector<int> prefix_arr(text_length);
    prefix_arr[0] = 0;
    std::cout << "\nЗначение префикс-функции для символа под номером: "
    << 0
        << " (" << text[0] << ")"
        << " = " << 0 << '\n';
    for (int i = 1; i < text_length; i++) {
        int current_length = prefix_arr[i - 1];
        std::cout
        << "\nВычисление значения префикс-функции для символа под
номером: "
        << i << " (" << text[i] << ")" << '\n';
        // Если предыдущий суффикс нельзя расширить, нужно рассмотреть
суффикс

```



```

        // меньшего размера prefix_arr[current_length - 1]
        while (current_length > 0 && (text[current_length] != text[i])) {
            std::cout << "Предыдущий суффикс размера " << current_length
                << " нельзя расширить" << '\n';
            current_length = prefix_arr[current_length - 1];
            std::cout << "Рассмотрим новый суффикс меньшего размера: "
                << current_length << '\n';
        }
        // Проверяем можно ли расширить
        if (text[current_length] == text[i]) {
            std::cout << "Суффикс длинны " << current_length << " можно
расширить"
                << '\n';
            current_length++;
            std::cout << "Новый размер суффикса: " << current_length << '\n';
        }
        std::cout << "Значения префикс-функции равно " << current_length <<
'\n';
        // заносим соответствующие значение в массив
        prefix_arr[i] = current_length;
    }

    std::cout << "\nЗначения префикс-функции:" << '\n';
    // Выводим символы текста
    for (auto j : text) {
        std::cout << j;
        std::cout.width(3);
    }
    std::cout << '\n';
    // Выводим соответствующие значения
    for (auto i : prefix_arr) {
        std::cout << i;
        std::cout.width(3);
    }
    std::cout << "\n\n";
    return prefix_arr;
}

// проверка на циклический сдвиг
int is_cycle(std::string string_a, std::string string_b) {
    // проверяем длину
    if (string_a.length() != string_b.length()) {
        std::cout << "Длины не равны => не циклический сдвиг" << '\n';
        return -1;
    }
    // вызываем функция для создания массива префиксоф
    std::vector<int> prefix_arr_b = create_prefix(string_b);
    int string_a_length = string_a.length();
    int string_a_length_2 = string_a_length * 2;
    int cur_len_b = 0;
    for (int i = 0; i < string_a_length_2; i++) {

```

```

// так как индексов в два раза больше
int j = i % string_a_length;
// если символы не совпали
if (string_b[cur_len_b] != string_a[j]) {
std::cout << "Не совпали " << j << " символа строки A "
    << "(" << string_a[j] << ") "
    << "и " << cur_len_b << " символа строки B "
    << "(" << string_b[cur_len_b] << ") "
    << "\n\n";
}
while (cur_len_b > 0 && string_b[cur_len_b] != string_a[j]) {
cur_len_b = prefix_arr_b[cur_len_b - 1];
}
// если совпали символы
if (string_b[cur_len_b] == string_a[j]) {
std::cout << "Найдено совпадение " << j << " символа строки A "
    << "(" << string_a[j] << ") "
    << "и " << cur_len_b << " символа строки B "
    << "(" << string_b[cur_len_b] << ") " << '\n';
cur_len_b++;
}
// если нашли вхождение
if (cur_len_b == string_a_length) {
std::cout << "\nНашли вхождение, индекс равен " << i - cur_len_b +
1
    << "\n\n";
return i - cur_len_b + 1;
}
}
std::cout << "Не циклический сдвиг" << '\n';
return -1;
}

int main() {
std::vector<int> answer;
std::string string_a;
std::string string_b;

std::cout << "Введите строку A" << '\n';
std::cin >> string_a;
std::cout << "Введите строку B" << '\n';
std::cin >> string_b;

std::cout << is_cycle(string_a, string_b) << '\n';
return 0;
}

```