

**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: Поиск подстроки в строке**

Студент гр. 3343

\_\_\_\_\_

Пименов П.В.

Преподаватель

\_\_\_\_\_

Жангиров Т.Р.

Санкт-Петербург

2025

### **Цель работы.**

Изучить и реализовать алгоритм Кнута-Морриса-Пратта для поиска подстроки в строке. Применить алгоритм на практике, определив, является ли одна строка циклическим сдвигом другой.

### **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  $-1$

Sample Input:

ab

abab

Sample Output:

0,2

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

### **Выполнение работы.**

Реализован алгоритм КМП (Кнута-Морриса-Пратта).

Описание реализованных функций:

- `vector<int> prefix_function(const string& pattern)` – вычисляет значения префикс функции для каждой позиции  $i$  в строке `pattern`. Возвращает вектор полученных значений.
- `vector<int> find(const string& text, const string& pattern)` – функция поиска подстроки в строке. Использует алгоритм КМП. Осуществляем проход по тексту с использованием значений префикс-функции. Это позволяет не сравнивать символы, которые уже были сравнены до этого и совпали. То есть, если `lps[j-1] = k`, это значит: первые  $k$  символов уже гарантированно совпали, мы можем начать следующую попытку с позиции  $k$  в шаблоне, а не с начала.

Для решения задачи определения, является ли одна строка (A) циклическим сдвигом другой (B), можно с помощью алгоритма КМП проверить, является ли строка B подстрокой строки  $A + A$ .

### **Сложность алгоритма**

- По времени: алгоритм работает за  $O(n+m)$ , где  $n$  – длина текста,  $m$  – длина шаблона.  $O(m)$  – получение значений префикс функции для всех позиций  $i$  шаблона,  $O(n)$  – поиск по исходному тексту (с учетом использования значений префикс-функции).
- По памяти: алгоритм работает за  $O(m)$ , так как необходимо хранить  $m$  значений префикс функции.

## Тестирование

Входные данные	Выходные данные	Комментарий
abc bcabca	2	Поиск, успех
aa aaaaa	0,1,2,3	Поиск, успех
asd gra	-1	Поиск, успех
abcabc bcabca	1	Цикл, успех
aaa aaa	0	Цикл, успех
gra arr	-1	Цикл, успех

## Выводы.

Изучен и реализован алгоритм Кнута-Морриса-Пратта для поиска подстроки в строке. Алгоритм применен на практике, написана программа, определяющая, является ли одна строка циклическим сдвигом другой.

## ПРИЛОЖЕНИЕ А

### Файл main.cpp:

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

using namespace std;

vector<int> prefix_function(const string& pattern) {
    cout << "Calculating prefix function for string: " << pattern << endl
         << endl;
    int m = pattern.size();
    vector<int> lps(m, 0);
    int len = 0;
    for (int i = 1; i < m; i++) {
        cout << "i: " << i << ", len: " << len << endl;
        cout << "Comparing " << pattern[i] << " and " << pattern[len] << endl;
        if (pattern[i] == pattern[len]) {
            cout << "Symbols match" << endl;
            lps[i++] = ++len;
            cout << "lps[" << i - 1 << "] = " << len << endl;
            cout << "Increase i, i = " << i << endl;
        } else {
            cout << "Symbols do not match" << endl;
            if (len) {
                cout << "len = " << lps[len - 1] << endl;
                len = lps[len - 1];
            } else {
                cout << "lps[" << i << "] = 0" << endl;
                lps[i++] = 0;
                cout << "Increase i, i = " << i << endl;
            }
        }
        cout << endl;
    }

    cout << "Prefix function result:" << endl;
    for (int i = 0; i < m; i++)
        cout << lps[i] << " ";
    cout << endl << endl;

    return lps;
}

vector<int> find(const string& text, const string& pattern) {
    cout << "Searching for: '" << pattern << "' in: '" << text << "'" << endl
         << endl;
    vector<int> result;
    int n = text.size(), m = pattern.size();
    if (m > n) {
        cout << "Text size is less than pattern size" << endl;
        return result;
    }
    vector<int> lps = prefix_function(pattern);

    int i = 0, j = 0;
    while (i < n) {
        cout << "i: " << i << ", j: " << j << endl;
        cout << "Comparing " << text[i] << " and " << pattern[j] << endl;
        if (text[i] == pattern[j]) {
            cout << "Symbols match" << endl;
            i++, j++;
            cout << "Increase i and j, i = " << i << ", j = " << j << endl;
        }
    }
}
```

```

        if (j == m) {
            cout << "Substring found at " << i - j << endl;
            result.push_back(i - j);
            cout << "Reset j, j = " << lps[j - 1] << endl;
            j = lps[j - 1];
        }
    } else {
        cout << "Symbols do not match" << endl;
        if (j) {
            cout << "Reset j, j = " << lps[j - 1] << endl;
            j = lps[j - 1];
        } else {
            i++;
            cout << "j = 0, Increase i, i = " << i << endl;
        }
    }
    cout << endl;
}
return result;
}

int main() {
    string pattern, text;
    cin >> pattern >> text;

    vector<int> matches = find(text, pattern);
    if (matches.empty()) {
        cout << "-1" << endl;
    } else {
        for (size_t i = 0; i < matches.size(); i++) {
            if (i > 0)
                cout << ",";
            cout << matches[i];
        }
        cout << endl;
    }

    // if (text.size() != pattern.size()) {
    //     cout << -1 << endl;
    //     return 0;
    // }

    // vector<int> matches = find(pattern + pattern, text);
    // if (matches.empty()) {
    //     cout << -1 << endl;
    //     return 0;
    // }
    // cout << matches[0] << endl;
    return 0;
}

```