

**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**  
**Вариант 2**

Студент гр. 8383

Преподаватель

\_\_\_\_\_

\_\_\_\_\_

Степанов В.Д.

Фирсов М. А.

Санкт-Петербург

2020

### **Цель работы.**

Изучить работу и реализовать алгоритм Кнута-Морриса-Пратта для нахождения подстроки в строке.

### **Постановка задачи.**

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

Базовая часть лаб. работы № 4 состоит в выполнении последних трёх заданий на Stepik из раздела 5. При этом реализация алгоритма КМП должна удовлетворять индивидуализации. Для обоих заданий на программирование должны быть версии кода с выводом промежуточных данных.

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.

### Описание алгоритма.

Даны образец (строка)  $S$  и строка  $T$ . Рассмотрим сравнение строк на позиции  $i$ , где образец  $S[0, m-1]$  сопоставляется с частью текста  $T[i, i+m-1]$ . Предположим, что первое несовпадение произошло между  $T[i+j]$  и  $S[j]$ , где  $1 < j < m$ . Тогда  $T[i, i+j-1] = S[0, j-1] = P$  и  $a = T[i+j] \neq S[j] = b$ .

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

Это приводит нас к следующему алгоритму: пусть  $\pi[j]$  — значение префикс-функции от строки  $S[0, m-1]$  для индекса  $j$ . Тогда после сдвига мы можем возобновить сравнения с места  $T[i+j]$  и  $S[\pi[j]]$  без потери возможного местонахождения образца. Можно показать, что таблица  $\pi$  может быть вычислена (амортизационно) за  $\Theta(m)$  сравнений перед началом поиска. А поскольку строка  $T$  будет пройдена ровно один раз, суммарное время работы алгоритма будет равно  $\Theta(m+n)$ , где  $n$  — длина текста  $T$ . По памяти мы можем оценить как  $\Theta(m)$ , так как хранится только строка  $T$ . Для поиска циклического сдвига одна из строк склеивается и в полученной строке ищется вторая.

### **Описание основных функций.**

```
void computeLPSArray(std::string &pat, int M, int* lps)
```

`std::string &pat` – строка обработки, `int M` – длина строки `pat`, `int* lps` – массив префиксов для строки `pat`

Функция заполняет массив префиксов для строки `pat`.

### **Программа поиска подстроки в строке:**

```
void KMPSearch(std::string &pat, std::vector <int> &answer)
```

`std::string &pat` – образец, `std::vector <int> &answer` – вектор, для записи индексов вхождения `pat`

Функция считывает посимвольно текст и ищет в нем вхождения строки образца, если совпадение найдено, то сохраняется индекс вхождения. Функция завершается, когда будет обработан весь текст.

### **Программа поиска циклического сдвига:**

```
void KMPSearch(std::string &pat, std::string &txt)
```

`std::string &pat` – образец, `std::string &txt` – текст

Функция ищет вхождение образца в тексте, при нахождении выводит индекс и завершает работу. Если образец не найден, то выводится -1.

### **Тестирование**

Тестирование программы 1 (поиска подстроки в строке) приведено на рисунке 1. Тестирование программы 2 (поиска циклического сдвига) приведено на рисунке 2.

### **Выводы.**

В ходе лабораторной работы был реализован на языке C++ алгоритм Кнута-Морриса-Практа для нахождения подстроки в строке.

```
PiAA_lab4 — -zsh — 70x48

lab4_1

Test #1 SUCCESS
Input:
ab
abab
Expected result:
0,2
Result:
0,2

Test #2 SUCCESS
Input:
a
abcdabaa
Expected result:
0,4,6,7
Result:
0,4,6,7

Test #3 SUCCESS
Input:
bcm
abcdmcma
Expected result:
-1
Result:
-1

Test #4 SUCCESS
Input:
abcd
abc
Expected result:
-1
Result:
-1

Test #5 SUCCESS
Input:
a
a
Expected result:
0
Result:
0
vladislavstepanov@MacBook-Pro--Vladislav PiAA_lab4 %
```

Рисунок 1 – Тестирование программы 1



A terminal window titled "PiAA\_lab4 — -zsh — 70x55" displays the output of a program. The program has executed six tests, all of which are successful. Each test shows the input, the expected result, and the actual result, which match perfectly. The tests involve string processing and counting characters.

```
lab4_2

Test #1 SUCCESS
Input:
defabc
abcdef
Expected result:
3
Result:
3

Test #2 SUCCESS
Input:
abcdefghijk
abcdefghijk
Expected result:
0
Result:
0

Test #3 SUCCESS
Input:
qwert
ertqw
Expected result:
2
Result:
2

Test #4 SUCCESS
Input:
ioewuropiqweruqwoei
ioewquopqwieruoew
Expected result:
-1
Result:
-1

Test #5 SUCCESS
Input:
aaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaa
Expected result:
0
Result:
0

Test #6 SUCCESS
Input:
z
z
Expected result:
0
Result:
0
```

Рисунок 2 – Тестирование программы 2

## ПРИЛОЖЕНИЕ А

### КОД ПРОГРАММЫ 1

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

void computeLPSArray(std::string &pat, int M, int* lps);

void KMPSearch(std::string &pat, std::vector <int> &answer)    ///
КМП алгоритм
{
    int M = (int) pat.length();

    int lps[M];          /// массив длин
префиксов

    computeLPSArray(pat, M, lps);    /// заполнение массива
длин префиксов

    std::cin.get();
    char c = std::cin.get();

    int i = 0;          /// индекс, для
перемещения по строке поиска
    int j = 0;          /// индекс, для
перемещению по подстроке

    while (c != '\n' && c != EOF) {    /// считываем строку
поиска посимвольно, пока не конец

        if (pat[j] == c) {    /// если символы i
символ подстроки и j символ подстроки равны
            j++;    /// то увеличиваем j и
i и считываем следующий символ
            i++;
            c = std::cin.get();
        }

        if (j == M) {    /// если j равна
длине подстроки
            answer.push_back(i-j);
```

```

        j = lps[j - 1];

        } else if (c != '\n' && c != EOF && pat[j] != c) { ///
иначе если не конец строки поиска и символы не совпадают

                if (j != 0) ///
если j не 0, то задаем j значения предыдущего значения массива
длин префикса
                        j = lps[j - 1];
                else{ ///
иначе считываем следующий символ
                        i = i + 1;
                        c = std::cin.get();
                }
        }
}

void computeLPSArray(std::string &pat, int M, int* lps) { ///
заполнение массива длин префиксов

        int len = 0;

        lps[0] = 0; /// первый
элемент всегда 0

        int i = 1;

        while (i < M) { ///
заполняем от 1 до размер массива -1
                if (pat[i] == pat[len]) {
                        len++;
                        lps[i] = len; /// задаем
i элементу значение len
                        i++;

                } else { /// (pat[i]
!= pat[len])

                        if (len != 0) {
                                len = lps[len - 1]; /// если
len не 0, то задаем len значения предыдущего значения массива длин
префикса

```



```

        } else {                                     // if (len
== 0)
                lps[i] = 0;                           /// задаем
i элементу массиву 0
                i++;
        }
    }
}

int main()
{

    std::string pat;
    std::vector<int> answer;
    std::ofstream outFile;
    outFile.open("./outProg");

    std::cin >> pat;

    KMPSearch(pat, answer);

    for (int k = 0; k < answer.size(); k++) {         /// ВЫВОД
результатов работы лагоритма

        std::cout << answer[k];
        outFile << answer[k];

        if (k != answer.size()-1){
            std::cout << ",";
            outFile << ",";
        }
    }

    if (answer.empty()){
        std::cout << -1;
        outFile << -1;
    }
    return 0;
}

```

## ПРИЛОЖЕНИЕ Б

### КОД ПРОГРАММЫ 2

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

void computeLPSArray(std::string &pat, int M, int* lps);

void KMPSearch(std::string &pat, std::string &txt){    /// КМП
алгоритм

    std::ofstream outFile;
    outFile.open("./outProg");

    int M = (int) pat.length();
    int N = (int )txt.length();

    int *lps = new int [M];                /// массив длин
префиксов

    computeLPSArray(pat, M, lps);          /// заполнение массива
длин префиксов

    int i = 0;                            /// индекс, для
перемещения по строке поиска
    int j = 0;                            /// индекс, для
перемещению по подстроке

    while (i < N) {                        /// считываем строку поиска
посимвольно, пока не конец

        if (pat[j] == txt[i]) {           /// если символы
i символ подстроки и j смвол подстроки равны
            j++;                          /// то увеличиваем j и
i и считываем следующий символ
            i++;
        }

        if (j == M) {                     /// если j равна
длине подстроки, то строка является циклическим сдвигом
```

```

        std::cout << i - j;                /// выводим индекс
начала строки
        outFile << i - j;
        return;

    } else if (i < N && pat[j] != txt[i]) {    ///
иначе если не конец строки поиска и символы не совпадают

        if (j != 0)                            ///
если j не 0, то задаем j значения предыдущего значения массива
длин префикса
            j = lps[j - 1];
        else{                                    ///
иначе считываем следующий символ
            i++;
        }
    }
}

std::cout << -1;                /// если подстрока не найдена, то
строка не является циклическим сдвигом
    outFile << -1;
}

void computeLPSArray(std::string &pat, int M, int* lps) { ///
заполнение массива длин префиксов

    int len = 0;

    lps[0] = 0;                            /// первый
элемент всегда 0

    int i = 1;

    while (i < M) {                            ///
заполняем от 1 до размер массива -1
        if (pat[i] == pat[len]) {
            len++;
            lps[i] = len;                    /// задаем
i элементу значение len
            i++;
        }
    }
}

```

```

        } else {
            // (pat[i]
            != pat[len])

            if (len != 0) {
                len = lps[len - 1];
                /// если
                len не 0, то задаем len значения предыдущего значения массива длин
                префикса
            } else {
                // if (len
                == 0)

                lps[i] = 0;
                /// задаем
                i элементу массиву 0
                i++;
            }
        }
    }
}

```

```

int main()
{

    std::string pat;

    std::string txt;

    std::cin >> txt;
    std::cin >> pat;

    if (txt.length() != pat.length()) {
        /// если длины строк
        не равны, то строка не является циклическим сдвигом
        std::cout << -1;
        return 0;
    }

    txt += txt;
    /// склеиваем строку
    поиска
    KMPSearch(pat, txt);
    return 0;
}

```