

**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. 8382

\_\_\_\_\_

Кобенко В.П.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2020

### **Цель работы.**

Изучить принцип работы алгоритма Кнута-Морриса-Пратта для поиска подстроки в строке.

### **Постановка задачи.**

1) Вар. 2. Оптимизация по памяти: программа должна требовать  $O(m)$  памяти, где  $m$  - длина образца.

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

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

Sample Input:

defabc

abcdef

Sample Output:

3

### **Описание алгоритма КМП.**

На вход алгоритма передается строка-образец, вхождения которой нужно найти, и строка-текст, в которой нужно найти вхождения. Алгоритм сначала вычисляет префикс-функцию (длина наибольшего собственного префикса подстроки, который совпадает с суффиксом этой подстроки) строки-образца.

Далее переменная-счетчик  $k$  приравнивается к 0. Переменная-позиция в образце изначально  $l = 0$ . При каждом совпадении  $k$ -го символа образца и  $l$ -го символа текста, переменные увеличиваются на 1. Если  $k = \text{размер образца}$ , значит вхождение найдено. Если очередной символ текста не совпал с  $k$ -ым символом образца, то обращаемся к массиву с префикс-функциями, смотрим значение префикс-функции для символа, предшествующего не совпавшему, приравниваем индекс символа в образце этому значению.

Функция вычисления префикс-функции проходится по строке-образцу 1 раз, поэтому требует  $O(m)$  времени, где  $m$  – длина строки-образца. Процесс поиска вхождений строки-образца в строке-текста выполняется, пока не закончится строка-текст т.е. требует  $O(n)$  времени, где  $n$  - длина строки-текста. Итого общая оценка сложности **по времени** алгоритма Кнута-Морриса-Пратта составляет  $O(m + n)$ .

Для работы, алгоритм вычисляет значение префикс функции для каждого символа строки-образца и хранит эти значения в массиве, следовательно

сложность алгоритма **по памяти** составляет  $O(m)$ , где  $m$  - длина строки-образца.

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

Для поиска циклического сдвига было решено использовать функцию поиска префикс-функции. Но для начала проверяется равенство длин строк, если длина разная, строка  $A$  - не циклический сдвиг строки  $B$ . Строки проверяются на равенство, если строки идентичны, то строка  $A$  является циклическим сдвигом  $B$  со сдвигом  $0$ .

Если длина строк совпадает, но строки не совпадают, то составляется строка, представляющая собой склейку строки  $A$  и двух строк  $B$ . Две склеенных строки  $B$  обязательно содержат в себе все возможные комбинации циклических сдвигов строки  $A$ . Запускается поиск префикс-функции для строки-склейки. Если в результате работы функции для какого-либо символа значение префикс-функции оказывается равно длине изначальной строки, это фактически означает, что в строке, состоящей из двух строк  $B$  была найдена строка  $A$ , значит  $A$  - циклический сдвиг  $B$ .

Сложность алгоритма **по времени** составляет  $O(n)$ , где  $n$  - длина строки  $A$ , т.к. алгоритм в худшем случае будет искать префикс-функцию для всех символов склеенной строки т.е. пройдет по всем  $3n$  символам.

Сложность алгоритма **по памяти** составляет  $O(n)$ , т.к. в процессе к исходной строке добавляется 2 строки длиной  $n$  и в результате выполнения функции нахождения функции-префикса в худшем случае хранятся значения префикс-функции для  $3n$  символов.

### **Описание функций.**

`void prFunc(std::string& p, std::vector<int>& pf)`- функция нахождения префикс-функции, прекращает работу, когда значение префикс-функции для какого-либо становится равной длине исходно введенной строки,

выводит на экран индекс начала зацикливания, ничего не возвращает и прекращает работу, если это произошло, если нет печатает -1.

### Промежуточные результаты.

Пример вывода промежуточных результатов для алгоритма КМР представлены на рис. 1.

```
ab
abab
Поиск ПФ
[1], [0] не совпали b, a.
ПФ для b-[1] - 0.

Начало алгоритма
В [0], [0] - a.

Индексы увеличены на 1: j = 1, i = 1.
В [1], [1] - b.
[2] - равен длине искомой строки 2
Индексы увеличены на 1: j = 2, i = 2.
[2], [2] не равны a, .
Индексы: j = 2, i = 0.
В [0], [2] - a.

Индексы увеличены на 1: j = 3, i = 1.
В [1], [3] - b.
[2] - равен длине искомой строки 2
Индексы увеличены на 1: j = 4, i = 2.
0,2
vlad@vlad-GL62M-7RDX:~/PAA/Lab4$
```

Рисунок 1 - Вывод промежуточных результатов алгоритма КМР.

Пример вывода промежуточных результатов алгоритма поиска циклического сдвига представлен на рис. 2.

```

abcdef
defabc
Длины строк одинаковы.
Создание строки, содержащей 1-ую и две 2-ых строки
Получилась строка: defabcabcdefabcdef
Поиск ПФ
[6], [0] не совпали а, d.
ПФ для а-[6] - 0.
[7], [0] не совпали b, d.
ПФ для b-[7] - 0.
[8], [0] не совпали c, d.
ПФ для c-[8] - 0.
[9] и [0] - это d.Индексы увеличены на 1
ПФ для d[9] - 1.
[10] и [1] - это e.Индексы увеличены на 1
ПФ для e[10] - 2.
[11] и [2] - это f.Индексы увеличены на 1
ПФ для f[11] - 3.
[12] и [3] - это а.Индексы увеличены на 1
ПФ для а[12] - 4.
[13] и [4] - это b.Индексы увеличены на 1
ПФ для b[13] - 5.
[14] и [5] - это c.Индексы увеличены на 1
ПФ для c[14] - 6.
6 - начало строки, дальше закидывается с началом в:
3

```

Рисунок 2 — Вывод промежуточных результатов алгоритма нахождения циклического сдвига

### Тестирование.

Тестирование алгоритма КМР.

№	Input	Output
1	ab abab	0,2
2	ab abcabca	0,3
3	def abcabcab	-1

Тестирование алгоритма поиска циклического сдвига.

№	Input	Output
1	def def	0

2	abcde deabc	3
3	aa aaa	-1
4	a b	-1

Код программ приведен в приложении А.

### **Вывод.**

В ходе лабораторной работы был реализован на языке C++ алгоритм Кнутта-Морриса-Пратта для поиска подстроки в строке, а также алгоритм для поиска циклического сдвига. Алгоритм КМП имеет лучшую сложность по времени, чем наивный поиск подстроки  $O(n+m)$  против  $O(n*m)$ , поэтому предпочтителен в использовании.

## ПРИЛОЖЕНИЕ А

### st\_4\_1.cpp

```
#include <iostream>

#include <string>
#include <vector>

#define d

int main() {
    int i = 0, j = 0, res_counter = 0, isFound = 0, count = 1, counter = 0;
    std::string p, t;
    std::vector<int> pf, res;
    std::cin >> p;
    std::cin >> t;
    pf.resize(p.length());
    pf[0] = 0;
    #ifndef d
    std::cout << "Поиск ПФ" << std::endl;
    #endif
    while (count < (int)p.length()) {
        if (p[count] == p[counter]) {
            #ifndef d
            std::cout << "[" << count << "] и [" << counter << "] - это " << p[count] << ".Индексы  
увеличены на 1\n";
            #endif
            pf[count] = counter + 1;
            #ifndef d
            std::cout << "ПФ для " << p[count] << "[" << count << "] - " << pf[count] << ".\n";
            #endif
            count++;
            counter++;
        }
        else {
            #ifndef d
            std::cout << "[" << count << "], [" << counter << "] не совпали " << p[count] << ", " << p[counter] << ".\n";
            std::cout << "ПФ для " << p[count] << "-[" << count << "]" << " - 0.\n";
            #endif
            if (counter == 0) {
                pf[count] = 0;
                count++;
            }
            else {
                #ifndef d
                std::cout << "Индекс counter теперь равен значению ПФ " << pf[counter - 1] << ".\n";
                #endif
                counter = pf[counter - 1];
            }
        }
    }
}
```



```

}
}
}
#ifdef d
std::cout << "\nНачало алгоритма\n";
#endif
while (j != t.length()) {
if (t[j] == p[i]) {
#ifdef d
std::cout << "B [" << i << "], [" << j << "] - " << t[j] << ".\n";
#endif
j++;
i++;
if (i == p.length()) {
#ifdef d
std::cout << "[" << i << "] - равен длине искомой строки " << i;
#endif
isFound = 1;
res.push_back(j - p.length());
res_counter++;
}
#ifdef d
std::cout << "\nИндексы увеличены на 1: j = " << j << ", i = " << i << ".\n";
#endif
}
else {
#ifdef d
std::cout << "[" << i << "], [" << j << "] не равны " << t[j] << ", " << p[i] << ".\n";
#endif
if (i == 0) {
j++;
}
else {
i = pf[i - 1];
}
#ifdef d
std::cout << "Индексы: j = " << j << ", i = " << i << ".\n";
#endif
}
}
if (!isFound) {
#ifdef d
std::cout << "Совпадений не найдено!\n";
#endif
}
if(res.empty()){
std::cout << "-1";
return 0;
}
for (int k = 0; k < res.size(); k++){

```

```
if (k != 0)
std::cout << "," << res[k];
else std::cout << res[k];
}
std::cout << std::endl;
}
```

## st\_4\_2.cpp

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

#define d

void prFunc(std::string& p, std::vector<int>& pf) {
    pf.resize(p.length());
    pf[0] = 0;
    int count = p.length() / 3, counter = 0, res;
    while (count < (int)p.length()) {
        if (p[count] == p[counter]) {
            #ifndef d
            std::cout << "[" << count << "] и [" << counter << "] - это " << p[count] << ".Индексы
            увеличены на 1\n";
            #endif
            pf[count] = counter + 1;
            #ifndef d
            std::cout << "ПФ для " << p[count] << "[" << count << "] - " << pf[count] << ".\n";
            #endif
            if (pf[count] == (p.length() / 3)) {
                #ifndef d
                std::cout << pf[count] << " - начало строки, дальше зацикливается с началом в:\n";
                #endif
                res = count - 2 * ((p.length()) / 3) + 1;
                std::cout << res << std::endl;
                return;
            }
            count++;
            counter++;
        }
        else {
            #ifndef d
            std::cout << "[" << count << "], [" << counter << "] не совпали " << p[count] << ", " <<
            p[counter] << ".\n";
            std::cout << "ПФ для " << p[count] << "-[" << count << "]" << " - 0.\n";
            #endif
            if (counter == 0) {
                pf[count] = 0;
                count++;
            }
            else {
                #ifndef d
                std::cout << "Индекс counter теперь равен значению ПФ " << pf[counter - 1] << ".\n";
                #endif
                counter = pf[counter - 1];
            }
        }
    }
}
```

```

}
}
std::cout << "-1";
}

```

```

int main() {
std::vector<int> pf;
std::string p, t;
std::cin >> p;
std::cin >> t;
if (p.length() != t.length()) {
#ifdef d
std::cout << "Длины строк разные\n";
#endif
std::cout << "-1";
return 0;
}
#ifdef d
std::cout << "Длины строк одинаковы.\n";
#endif
if (p == t) {
#ifdef d
std::cout << "Строки одинаковые\n";
#endif
std::cout << "0";
return 0;
}
#ifdef d
std::cout << "Создание строки, содержащей 1-ую и две 2-ых строки\n";
#endif
t = t + p + p;
#ifdef d
std::cout << "Получилась строка: " << t << "\n";
std::cout << "Поиск ПФ\n";
#endif
prFunc(t, pf);
return 0;
}

```