

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**  
**по практической работе №4**  
**по дисциплине «Построение и Анализ Алгоритмов»**  
**Тема: Алгоритм Кнута-Морриса-Пратта**

Студент гр. 8382

\_\_\_\_\_

Вербин К.М.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2020

### **Цель работы.**

Ознакомиться с алгоритмом Кнута-Морриса-Пратта для поиска подстроки в строке и реализовать его.

### **Задание 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  $T$ .

#### **Входные данные**

Первая строка -  $P$ .

Вторая строка -  $T$ .

#### **Выходные данные**

индексы начал вхождений  $P$  в  $T$ , разделенных запятой, если  $P$  не входит в  $T$ , то вывести -1.

#### **Пример входных данных**

ab

abab

#### **Соответствующие выходные данные**

0,2

### **Задание 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.

#### **Входные данные**

Первая строка -  $A$ .

Вторая строка -  $B$ .

#### **Выходные данные**

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

#### **Пример входных данных**

defabc

abcdef

## **Соответствующие выходные данные**

3

### **Вариант 1**

Подготовка к распараллеливанию: работа по поиску разделяется на  $k$  равных частей, пригодных для обработки  $k$  потоками (при этом длина образца гораздо меньше длины строки поиска).

### **Описание алгоритма**

#### **Жадный алгоритм**

Алгоритм считывает две строки: образец и текст для поиска. После этого "строится" (физически новая строка не строится в целях экономии памяти, для этого используется структура, описанная ниже) строка вида  $S \ P \ T = \#$ , где  $\#$  - символ-разделитель, текст  $T$  и образец  $P$  (по условию сказано, что символы строк принадлежат латинскому алфавиту, поэтому можно взять любой другой в качестве разделителя). Для полученной строки рассчитывается префиксфункция. Префикс-функция для  $i$ -го символа возвращает значение, равное максимальной длине совпадающих префикса и суффикса подстроки, которая заканчивается  $i$ -м символом. Благодаря символу-разделителю выполняется условие  $\forall i \leq |P| \ p_i \neq \#$ , где  $p_i$  – значение префикс-функции для символа  $i$ , т.к. благодаря разделителю префиксом будет образец.

После того, как префикс-функция была построена, вызывается функция, которая обходит строку за заданное число "потоков", которое введено. Каждый поток обрабатывает свою часть строки. Если значение префикс функции текущего символа равно длине образца, то это означает, что алгоритм нашел вхождение. Индекс вхождения (смещение в строке для поиска) равен текущему индексу за вычетом двух длин образца. Если ни одного вхождения не было найдено, то выводится -1.

### **Сложность алгоритма по памяти**

Сложность алгоритма по памяти будет  $O(P + T)$ , т.к. префикс-функция считается для "склеенной" строки  $P\#T$

### **Временная сложность алгоритма**

Сложность по времени алгоритма будет  $O(P + T)$ , т.к. вычисление префикс-функции для "склеенной" строки  $P\#T$  выполняется за линейное время (то есть за  $O(P + T)$ ), а поиск вхождений образца в строке в цикле выполняется также за линейное время  $O(P + T)$ , поэтому общая сложность будет  $O(P + T + P + T) = O(P + T)$ .

### **Алгоритм A\***

В начале алгоритма считываются две строки одинаковой длины  $A$  и  $B$ , если длины не совпадают, то алгоритм сразу завершает работу и выводит -1, т.к. в таком случае  $A$  не является циклическим сдвигом. Затем с помощью той же функции вычисляется префикс-функция для строки  $B$ . После этого строка  $A$  удваивается, чтобы найти индекс строки  $B$  в строке  $A$  (если является циклическим сдвигом). После алгоритм заводит два индекса, которые указывают на символы в удвоенной строке  $A$  и строке  $B$ , после чего алгоритм проходит и сравнивает символы. Если произошло очередное совпадение, то индексы увеличиваются, если происходит несовпадение, то с помощью вектора префикс-функций для  $B$  осуществляется "сдвиг" образа (не на 1, как в "наивном" алгоритме, т.к. по префикс-функции можно определить, что часть символов уже совпала). Когда индекс указывающий на символ в образе  $B$  равен длине этого образа, это значит, что вхождение найдено. Т.к.  $A$  - это удвоенная строка, то мы определили что исходная строка  $A$  является циклическим сдвигом  $B$  и алгоритм завершает работу. В противном случае выводится -1.

### **Сложность алгоритма по памяти**

Сложность по памяти будет  $O(B + A)$ , т.к. в этом случае алгоритм строит префикс-функцию для строки  $B$  и на ее хранение требуется  $B$  памяти, а удвоение строки требует дополнительно  $A$  памяти

## Временная сложность алгоритма

Сложность по времени будет  $O(2|A| + |B|) = O(3|A|) = O(|A| + |B|)$  т.к. для заполнения префикс-функции нужно  $|B|$  операций, а при поиске цикл проходит за линейное время по удвоенной строке  $A$ .

## Описание структур данных

Splice – хранит в себе строки, обрабатываемые алгоритмом

prefixFunction - вычисляет префикс-функцию для каждого символа переданной строки splice и сохраняет значения в векторе prefix.

findEntry - получает на вход текст, образец, вектор префикс-функции, число переданных потоков и вычисляет индексы вхождений образца.

printInterInfo - выводит промежуточные данные - состояние префикс-функции для текущей части строки (часть строки передается через n).

## Тестирование

Input	Output
ab abab 1	Индексы: 0,2
a ababfaabfaaabaafhaaa 5	Индексы: 0,2,5,6,9,10,11,13,14,17,18,19
aaaaaaaaaaaaa aaaaaaaaaaaaa	Является сдвигом, позиция:0
Aaa aaaa	Не является: -1

## **Вывод**

В ходе работы был разобран и реализован алгоритм Кнута-МоррисаПратта для поиска вхождений подстроки в строку

## ПРИЛОЖЕНИЕ

### КОД ПРОГРАММЫ

#### Поиск подстроки

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <cmath>
#include <climits>

bool INTER = true;

//перегружены оператор вывода вектора в поток
std::ostream& operator<<(std::ostream& stream, std::vector<int> vect) {
    stream << vect[0];
    for (int i = 1; i < vect.size(); i++) {
        stream << "," << vect[i];
    }
    return stream;
}

void printInterInfo(std::string& splice, std::vector<int> vect, int n) {
    std::cout << "\nСостояние префикс-функции для строки: " <<
std::endl;
    for (int i = 0; i <= n; i++)
        std::cout << splice[i] << " ";
    std::cout << std::endl;
    for (int i = 0; i <= n; i++)
        std::cout << vect[i] << " ";
    std::cout << std::endl;
}

void prefixFunction(std::string& sample, std::vector<int>& prefix) {
    prefix[0] = 0;
    std::cout << "Начало расчета префикс-функции" << std::endl;
    int n = 1;
    for (int i = 1; i < sample.length(); i++)
    { // поиск какой префикс-суффикс можно расширить
        int k = prefix[i - 1]; // длина предыдущего префикса
        while (k > 0 && sample[i] != sample[k]) // этот нельзя расширить,
```

```

        k = prefix[k - 1];           // берем ранее рассчитанное значение

        if (sample[k] == sample[i])
            k = k + 1;               // расширяем найденный префикс-
суффикс

        prefix[i] = k;

        if (INTER)
            printInterInfo(sample, prefix, n++);
    }
}

```

```

//функция определения сдвига
bool findEntry(std::string& text, std::string& sample, std::vector<int>&
prefix) {
    int k = 0; //индекс сравниваемого символа в sample
    bool flag = false;
    if (text.length() <= sample.length()) {
        std::cout << "\nНе является циклическим сдвигом: ";
        std::cout << "-1";
        return false;
    }

    for (int i = 0; i < text.length(); i++) {
        while (k > 0 && sample[k] != text[i])
            k = prefix[k - 1];

        if (sample[k] == text[i])
            k = k + 1;
        else if (k != 0) k = prefix[k - 1];
        if (k == sample.length()) {
            std::cout << "Является сдвигом, позиция: ";
            std::cout << i - sample.length() + 1 << std::endl;
            flag = true;
        }
    }

    if (!flag) std::cout << "\nНе является циклическим сдвигом: " << -1;
    return flag;
}

```



```
}
```

```
int main()
{
    setlocale(LC_ALL, "Russian");
    std::string text;
    std::string sample;
    std::cin >> sample; std::cin >> text;
    std::vector<int> prefix(sample.length());
    prefixFunction(sample, prefix);
    findEntry(text, sample, prefix);
    return 0;
}
```

### **Код поиска циклического сдвига**

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <cmath>
#include <climits>
```

```
bool INTER = true;
```

```
//перегружены оператор вывода вектора в поток
std::ostream& operator<<(std::ostream& stream, std::vector<int> vect) {
    stream << vect[0];
    for (int i = 1; i < vect.size(); i++) {
        stream << "," << vect[i];
    } return stream;
}
```

```
void printInterInfo(std::string& splice, std::vector<int> vect, int n) {
    std::cout << "\nСостояние префикс-функции для строки: " <<
std::endl;
    for (int i = 0; i <= n; i++)
        std::cout << splice[i] << " ";
    std::cout << std::endl;
    for (int i = 0; i <= n; i++)
        std::cout << vect[i] << " ";
    std::cout << std::endl;
```

```
}
```

```
void prefixFunction(std::string& sample, std::vector<int>& prefix) {  
    prefix[0] = 0;  
    std::cout << "Начало расчета префикс-функции" << std::endl;  
    int n = 1;  
    for (int i = 1; i < sample.length(); i++)  
    { // поиск какой префикс-суффикс можно расширить  
        int k = prefix[i - 1]; // длина предыдущего префикса  
        while (k > 0 && sample[i] != sample[k]) // этот нельзя расширить,  
            k = prefix[k - 1]; // берем ранее рассчитанное значение  
  
        if (sample[k] == sample[i])  
            k = k + 1; // расширяем найденный префикс-  
суффикс  
  
        prefix[i] = k;  
  
        if (INTER)  
            printInterInfo(sample, prefix, n++);  
    }  
}
```

```
//функция определения сдвига  
bool findEntry(std::string& text, std::string& sample, std::vector<int>&  
prefix) {  
    int k = 0; //индекс сравниваемого символа в sample  
    if (text.length() != sample.length()) {  
        std::cout << "\nНе является циклическим сдвигом: ";  
        std::cout << "-1";  
        return false;  
    }  
  
    text = text + text; //удваиваем строку  
  
    for (int i = 0; i < text.length(); i++) {  
        while (k > 0 && sample[k] != text[i])  
            k = prefix[k - 1];  
  
        if (sample[k] == text[i])  
            k = k + 1;  
    }  
}
```

```

        if (k == sample.length()) {
            std::cout << "Является сдвигом, позиция: ";
            std::cout << i - sample.length() + 1;
            return true;
        }

    }

    std::cout << "\nНе является циклическим сдвигом: ";
    std::cout << "-1";
    return false;

}

int main()
{
    setlocale(LC_ALL, "Russian");
    std::string text;
    std::string sample;
    std::cin >> text;
    std::cin >> sample;
    std::vector<int> prefix(sample.length());
    prefixFunction(sample, prefix);
    findEntry(text, sample, prefix);
    return 0;
}

```