

**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. 8303

\_\_\_\_\_

Рудько Д.Ю.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2020

## Цель работы

Изучение алгоритма Кнута-Морриса-Пратта поиска образца в строке.

## Задание

### • Задание 1

Реализуйте алгоритм КМП и с его помощью для заданных образца  $P$  ( $|P| \leq 15000$ ) 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

### • 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.

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

### **Индивидуализация 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 алгоритмов**

#### **КМП**

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.

#### **Циклический сдвиг**

В данном алгоритме можно обойтись без удваивания строки. В самом начале происходит проверка на соответствие длин строк. Если соответствия не было обнаружено, то выводится -1. Создаются два счётчика для первой и второй строки. Далее сравниваются символы первой и второй строки, если символы совпадают переход к следующим, счётчики увеличиваются, если совпадения не обнаружено, счётчик для второй строки уменьшается. В том случае, если счётчик второй строки равен её длине, то сдвиг найден, а если

счётчик первой строки равен её длине, то происходит его обнуление, таким образом строка зацикливается.

### **Префикс функция**

Префикс-функция от строки и позиции в ней — длина наибольшего собственного префикса подстроки, который одновременно является суффиксом этой подстроки. То есть, в начале подстроки длины нужно найти такой префикс максимальной длины, который был бы суффиксом данной подстроки.

### **Разделение исходного текста**

Получаем исходный текст (строку) и число частей, на которое нужно разделить строку. Анализируя длину строки и число частей получаем длину каждой части строки. Далее сохраняем части строки в массив. Отдельно проверяем не раздели ли мы строку на месте образца. Для этого получаем подстроку с разрезом строки по середине и обрабатываем данную подстроку отдельно, ее длина  $2 \cdot n - 2$ , где  $n$  - длина образца.

Пример:

abacaba | cabaaba

обрабатываемая подстрока баса.

Сложность алгоритма КМП :

$O(n + m)$ ,  $n$  — длина подстроки,  $m$  — длина строки.

Сложность алгоритма поиска циклического сдвига:

$O(n + n) = O(n)$ .

### **Описание функций**

1) `vector<int> prefix_function (string s)`

Возвращает значение префикс функции для строки.

2) `vector<int> КМР(string t, string p, vector<int> &pi)`

Функция нахождения образца в тексте алгоритмом Кнута-Морриса-Пратта.

`string t` — исходный текст

`string p` - образец

`vector<int> &pi` — ссылка на вектор значений префикс-функции.

3) `void split(string t, string p, int k, vector<string> &str, vector<int> &ans_current, vector<int> &ans, vector<int> &pi)`

Функция разделения исходного текста на части.

`string t` — исходный текст

`string p` — образец

`int k` — число частей исходного текста

`vector<int> &str` — ссылка на вектор хранящий части строк исходного текста

`vector<int> &ans_current` — ссылка на вектор ответов для текущей части исходного текста

`vector<int> &ans` — ссылка на вектор ответов для всего текста

`vector<int> &pi` — ссылка на вектор значений префикс-функции.

## Тестирование

### КМР

№	Входные данные	Выходные данные
1	abacabacabaaba aba	0, 4, 8, 11
2	Dance for me, dance for me, dance for me, oh, oh, oh I've never seen anybody do the things you do before They say move for me, move for me, move for me, ay, ay, ay And when you're done I'll make you do it all again for	6, 20, 34, 100, 119, 132, 145
3	qwertyuilkjhgfdszxcvbn ,mn	-1
4	ababasssababasssababa ababa	0, 8, 16
5	fffffffffff f	0, 1, 2, 3, 4, 5, 6, 7, 8, 9

```
Справка
Чтобы запустить программу введите номер задачи или ее название.
Найдите все вхождения образца в тексте:
Номер задачи - 1
Название - КМР или kmp
Определить, является ли строка 1 циклическим сдвигом строки 2:
Номер задачи - 2
Название - Rotation или rotation

Введите номер задачи или название алгоритма
1
Введите текст
qwertyqwertyqwerty
Введите образец (искомую подстроку)
qwerty
Введите число от 1 до 2
2
-----
Строка будет разделена на 2 частей
Максимальная длина части исходного текста - 9
-----
Подстрака: центр на месте разреза - tyqwertyqw
Префикс-функция для образца qwerty
0 0 0 0 0 0
Несовпадение: t!=q index: 0 0
Несовпадение: y!=q index: 1 0
Совпадение: q==q index: 2 0
Совпадение: w==w index: 3 1
Совпадение: e==e index: 4 2
Совпадение: r==r index: 5 3
Совпадение: t==t index: 6 4
Совпадение: y==y index: 7 5
Найдена подстрака
-----
Несовпадение: q!= index: 8 6
Совпадение: q==q index: 8 0
Совпадение: w==w index: 9 1
-----
```

```
Часть исходного текста qwertyqwe
Префикс-функция для образца qwerty
0 0 0 0 0 0
Совпадение: q==q index: 0 0
Совпадение: w==w index: 1 1
Совпадение: e==e index: 2 2
Совпадение: r==r index: 3 3
Совпадение: t==t index: 4 4
Совпадение: y==y index: 5 5
Найдена подстрака
-----
Несовпадение: q!= index: 6 6
Совпадение: q==q index: 6 0
Совпадение: w==w index: 7 1
Совпадение: e==e index: 8 2
-----
Часть исходного текста rtyqwerty
Префикс-функция для образца qwerty
0 0 0 0 0 0
Несовпадение: r!=q index: 0 0
Несовпадение: t!=q index: 1 0
Несовпадение: y!=q index: 2 0
Совпадение: q==q index: 3 0
Совпадение: w==w index: 4 1
Совпадение: e==e index: 5 2
Совпадение: r==r index: 6 3
Совпадение: t==t index: 7 4
Совпадение: y==y index: 8 5
Найдена подстрака
-----
0,6,12
Для закрытия данного окна нажмите <ВВОД>...
```

## Rotation

№	Входные данные	Выходные данные
1	defabc abcdef	3
2	defabc abcder	-1
3	foobar foobar	0
4	abaa aaba	3

```

Справка
Чтобы запустить программу введите номер задачи или ее название.
Найдите все вхождения обзарца в тексте:
Номер задачи - 1
Название - КМР или kmp
    Определить, является ли строка 1 циклическим сдвигом строки 2:
Номер задачи - 2
Название - Rotation или rotation

Введите номер задачи или название алгоритма
2
Введите строки 1 и 2
ghjklpoiuyt
poiuytghjkl
Префикс-функция для строки 2
0 0 0 0 0 0 0 0 0 0
Несовпадение: g!=p index: 0 0
Несовпадение: h!=p index: 1 0
Несовпадение: j!=p index: 2 0
Несовпадение: k!=p index: 3 0
Несовпадение: l!=p index: 4 0
Совпадение:  p==p index: 5 0
Совпадение:  o==o index: 6 1
Совпадение:  i==i index: 7 2
Совпадение:  u==u index: 8 3
Совпадение:  y==y index: 9 4
Совпадение:  t==t index: 10 5
Совпадение:  g==g index: 0 6
Совпадение:  h==h index: 1 7
Совпадение:  j==j index: 2 8
Совпадение:  k==k index: 3 9
Совпадение:  l==l index: 4 10
Цикл: 5
Для закрытия данного окна нажмите <ВВОД>...

```

## Вывод

В ходе выполнения лабораторной работы был изучен и реализован алгоритм Кнута-Морриса-Пратта для поиска подстроки в строке, результатом которого является набор индексов вхождения подстроки. Для работы алгоритма также реализована префикс-функция. Помимо основного алгоритма, так же реализован механизм распараллеливания строки, для запуска алгоритма сразу в нескольких местах.

## ПРИЛОЖЕНИЕ А.

### ИСХОДНЫЙ КОД ПРОГРАММЫ

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

using namespace std;

vector<int> prefix_function (string s) {
    int n = (int) s.length();
    vector<int> pi(n);
    for (int i=1; i<n; ++i) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j]) ++j;
        pi[i] = j;
    }
    return pi;
}

vector<int> KMP(string t, string p, vector<int> &pi){
    vector<int> ans;
    cout << "Префикс-функция для образца " << p << endl;
    for(int i = 0; i < pi.size(); i++)
        cout << pi[i] << ' ';
    cout << endl;
    int n = t.length();
    int m = p.length();
    int k = 0, l = 0;
    while(k < n){
        if(t[k] == p[l]){
            cout << "Совпадение:   " << t[k] << "==" << p[l] <<" index: "<< k <<
            " " << l << endl;
            k++; l++;
            if(l == m){ans.push_back(k-l); cout << "Найдена подстрока\
n-----" << endl; }
        }
        else {
            if(l == 0){
                cout << "Несовпадение: " << t[k] << "!=" << p[l] <<" index: "<<
            k << " " << l << endl;
            k++;
        }
        else {
            cout << "Несовпадение: " << t[k] << "!=" << p[l] <<" index: "<<
            k << " " << l << endl;
            l = pi[l-1];
        }
    }
    }
    return ans;
}

void split(string t, string p, int k, vector<string> &str, vector<int>
&ans_current, vector<int> &ans, vector<int> &pi){
    int len_parts, flag = 0;
    int k1;
    //-----
    //определяем длину каждой части
    if(t.length() % k){
        len_parts = int(t.length()/k)+1; //длина части строки
        flag = 1;
        k1 = k-1;
    }
}
```



```

else {
    k1 = k;
    len_parts = t.length()/k;
}
//-----
int begin = 0;
string part = "";
//цикл для получения массива подстрок из текста
while(k1 > 0){
    part = "";
    part.append(t, begin, len_parts);
    str.push_back(part);
    begin += len_parts;
    k1--;
}
if(flag){
    part = "";
    part.append(t, begin, (t.length()-(len_parts*(k-1))));
    str.push_back(part);
}

//цикл для получения и проверки подстрок на стыках на каждом стыке
проверяется 2 стрки
k1 = 1;
while(k1 < k){
    part = "";
    part.append(t, (len_parts*k1)-p.length()+1, 2*p.length()-2);
    int top = (len_parts*k1)-p.length()+1;
    cout << "Подстрака с центром на месте разреза - " << part << endl;
    ans_current = KMP(part, p, pi);
    if(ans_current.size() > 0){
        ans_current[0] += top; //определяем номер символа начала подстроки в
исходном тексте
        ans.insert(ans.end(), ans_current.begin(), ans_current.end());
    }
    k1++;
}

int main()
{
    cout << "\tСправка\nЧтобы запустить программу введите номер задачи или ее
название.\n"
        "\tНайдите все вхождения образца в тексте:\nНомер задачи - 1\
nНазвание - KMP или kmp\n"
        "\tОпределить, является ли строка 1 циклическим сдвигом строки 2:\
nНомер задачи - 2\nНазвание - Rotation или rotation"<< endl;
    cout << endl;
    string task;
    cout << "Введите номер задачи или название алгоритма" << endl;
    getline(cin, task);
    if(task == "KMP" or task == "kmp" or task == "1"){
        string p, t;
        cout << "Введите текст" << endl;
        getline(cin, t);
        cout << "Введите образец (искомую подстроку)" << endl;
        getline(cin, p);
        int max_threads = sizeof(thread); // определяем максимально возможное
число потоков
//-----
// определяем на сколько частей можно разделить строку
double alpha = (double)t.length()/(double)p.length();
max_threads = min(max_threads, int(alpha)-1);
if(max_threads == 0)
    max_threads = 1;
int k ;//= max_threads;
if(max_threads == 1){
    cout << "Длина исходного текста недостаточна для деления строки" <<
endl;
    k = 1;
}
else{

```

```

cout << "Введите число от 1 до " << max_threads << endl;
cin >> k;
while(k < 1 or k > max_threads){
    cout << "Введите число от 1 до " << max_threads << endl;
    cin >> k;
}
}
//-----
vector<int> pi = prefix_function(p);
vector<int> ans, ans_current;
vector<string> str;
cout << "-----" << endl;
cout << "Строка будет разделена на " << k << " частей" << endl;
if(k == 1)
    ans = KMP(t, p, pi);
else {
    //-----
    // определяем длину каждой части
    int len_parts;
    if(t.length() % k){
        len_parts = int(t.length()/k)+1; //длина части строки
    }
    else {
        len_parts = t.length()/k;
    }
    cout << "Максимальная длина части исходного текста - " << len_parts
<< endl;
    cout << "-----" << endl;
    cout << endl;
    split(t, p, k, str, ans_current, ans, pi);
    //-----
    //заполняем исходный массив ответов
    for(int i = 0; i < str.size(); i++){
        cout << "-----\nЧасть исходного текста " <<
str[i] << endl;
        ans_current = KMP(str[i], p, pi);
        if(ans_current.size() > 0){
            for(int j = 0; j < ans_current.size(); j++){
                ans_current[j] += (len_parts*i); // определяем номер
символа начала образца в исходном тексте
                ans.insert(ans.end(), ans_current.begin(),
ans_current.end());
            }
        }
    }
    // Вывод ответа
    if(ans.size() == 0)
        cout << "-1";
    else {
        sort(ans.begin(), ans.end()); //сортируем массив ответов для
читабельности
        for(int i = 0; i < ans.size(); i++){
            if(i == ans.size()-1)
                cout << ans[i];
            else {
                cout << ans[i] << ',';
            }
        }
        cout << endl;
    }
}
}
else{
    if(task == "Rotation" or task == "rotation" or task == "2"){
        string a,b;
        cout << "Введите строки 1 и 2" << endl;
        cin >> a >> b;
        vector<int> pi = prefix_function(b);
        cout << "Префикс-функция для строки 2" << endl;
        for(int i = 0; i < pi.size(); i++){
            cout << pi[i] << ' ';
        }
        cout << endl;
    }
}
}

```

```

    if(b.length() != a.length())
    {
        cout << "-1" << endl;
        return 0;
    }
    if(a == b){
        cout << 0 << endl;
        return 0;
    }
    int it_a = 0, it_b = 0;
    int cikle = 0;
    int al = a.length();
    while(true){
        if(a[it_a] == b[it_b]){
            cout << "Совпадение: " << a[it_a] << "==" << b[it_b] << "
index: " << it_a << " " << it_b << endl;
            it_a++;
            it_b++;
        }
        if(it_a == al){
            it_a = 0;
            cikle++;
        }
        if(it_b == al){
            cout << "Цикл: ";
            cout << it_a << endl;
            return 0;
        }
        else{
            if(a[it_a] != b[it_b]){
                cout << "Несовпадение: " << a[it_a] << "!=" << b[it_b]
<< " index: " << it_a << " " << it_b << endl;
                if(it_b == 0)
                    it_a++;
                else {
                    it_b = pi[it_b-1];
                }
            }
        }
        if(cikle > 1){
            cout << -1 << endl;
            return 0;
        }
    }
}
return 0;
}

```