

ФГБОУ ВПО  
ПЕТЕРБУРГСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ  
ИМПЕРАТОРА АЛЕКСАНДРА I

Кафедра "Информационные и вычислительные системы"

ОТЧЕТ  
по лабораторной работе №1  
**Поиск подстроки в строке**

Выполнил студент

Группа ИВБ-811

\_\_\_\_\_  
(подпись)

Зайцев Л.А.

Отчет принял

\_\_\_\_\_  
(подпись)

Баушев А.Н.

Санкт-Петербург, 2020 г.

# 1 Введение

**Поиск подстроки в строке** (англ. *String searching algorithm*) - класс алгоритмов над строками, которые позволяют найти паттерн (*pattern*) в тексте (*str*).

**Задача 1** Дана строка  $str[1..n]$  и паттерн  $pattern[1..m]$  такие, что  $n \geq m$  и элементы этих строк — символы из конечного алфавита  $\Sigma$ . Требуется проверить, входит ли *pattern* в *str*.

Эти алгоритмы подразделяются на несколько групп:

- **Сравнение — «чёрный ящик»**

Во всех этих алгоритмах сравнение строк является «чёрным ящиком». К этим алгоритмам относится **примитивный алгоритм**.

- **Основанные на сравнении с начала**

Это семейство алгоритмов страдает невысокой скоростью на «хороших» данных, что компенсируется отсутствием регрессии на «плохих». К этим алгоритмам относятся **алгоритм Рабина-Карпа** и **алгоритм Кнута-Морриса-Пратта**.

- **Основанные на сравнении с конца**

Сравнение строк друг с другом проводится справа налево.

- **Проводящие сравнение в необычном порядке**

## 2 Описание алгоритмов и их реализация

### 2.1 Примитивный алгоритм

#### 2.1.1 Описание алгоритма

В примитивном алгоритме поиск всех допустимых сдвигов производится с помощью цикла, в котором проверяется условие  $str[i .. i + m - 1] = pattern$  для каждого из  $n - m + 1$  возможных значений  $i$ .

#### 2.1.2 Код программы

Листинг 1: Примитивный алгоритм поиска подстроки в строке

```
function [index] = Pos(str, pattern)

n = strlength(str);
m = strlength(pattern);
for i = 1 : n - m + 1
    if str(i : i + m - 1) == pattern
        index = i;
        return;
    end
end

index = -1;
end % End of 'Pos.m' function
```

### 2.1.3 Сложность алгоритма

$$O((n - m) * m)$$

### 2.1.4 Результаты работы

$m = [5, 10, 20]$  (красный, зеленый, синий)

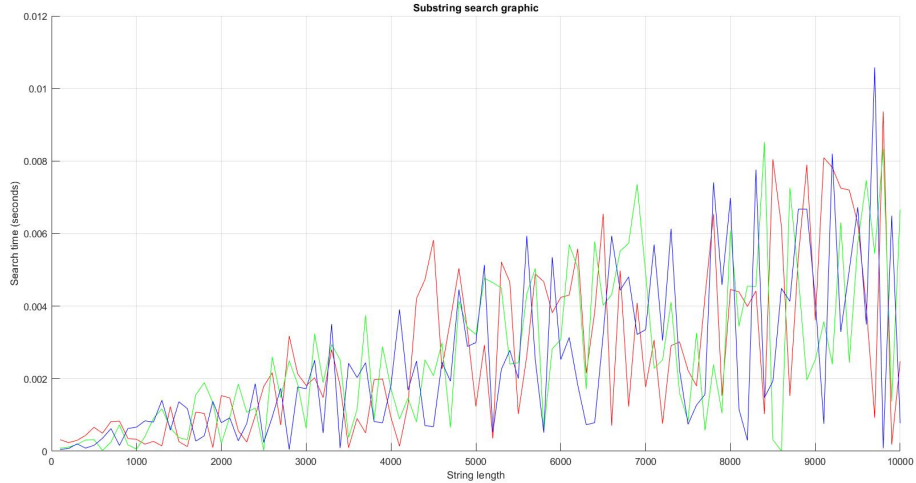


Рис. 1: Результат работы примитивного алгоритма

## 2.2 Алгоритм Рабина-Карпа

### 2.2.1 Описание алгоритма

Для реализации данного алгоритма нам понадобится хэш-функция. Воспользуемся полиномиальным хэшем:

$$h = \text{hash}(s[1..n]) = (p^n s_1 + p^{n-1} s_2 + \dots + p^0 s_n) \bmod q, \quad (1)$$

где  $p$  и  $q$  заранее заданные числа.

Отсюда следует, что:

$$h[i + 1] = h[i] \cdot p + s[i] \quad (2)$$

Исходя из (2) можно сделать вывод, что для того, чтобы пересчитать хэш-код за  $O(1)$  нам необходимо воспользоваться формулой:

$$\text{hash}(s[i..i + m - 1]) = (p \cdot \text{hash}(s[i - 1..i + m - 2]) - p^m s[i - 1] + s[i + m - 1]) \bmod q \quad (3)$$

Значения для  $p$  и  $q$  следует выбрать таким образом, чтобы уменьшить вероятность коллизий. В частности, мы хотим минимизировать количество таких остатков, которые не могут быть хэшем никакой строки. Тогда возьмём взаимнопростые числа.

Алгоритм:

1. Вычисляем значение хэш-кода для паттерна и для первых  $m$  символов в тексте.

2. Сравниваем хэш код для паттерна и текущей подстроки. Если они равны, то сохраняем индекс и выходим.
3. Считаем значение хэш-кода для следующей подстроки, переходим к шагу 2.

### 2.2.2 Код программы

Листинг 2: Алгоритм Рабина-Карпа

```
function [index] = RabinKarp(str, pattern)
n = strlen(str);
m = strlen(pattern);
q = 433494437;
p = 29;
h = 1;

% Evaluate p^m
for i = 2 : m + 1
    h = ModuloMult(h, p, q);
end
hStr = 0;
hPattern = 0;
% Evaluate hash value for pattern string and the first window
for i = 1 : m
    hPattern = ModuloAdd(pattern(i), ModuloMult(p, hPattern, q), q);
    hStr = ModuloAdd(str(i), ModuloMult(p, hStr, q), q);
end

if hStr == hPattern
    index = 1;
    return;
end

for i = 2 : n - m + 1
    hStr = ModuloAdd(ModuloMult(p, hStr, q), ModuloAdd(-ModuloMult(h, str(i - 1), q), hStr, q), q);

    if hStr < 0
        hStr = hStr + q;
    end

    if hStr == hPattern
        index = i;
        for j = index : index + m - 1
            if str(j) ~= pattern(j - index + 1)
                break;
            end
        end
        return;
    end
end
```

```

    end
end
index = -1;
end % End of 'RabinKarp' function

function res = ModuloAdd(x, y, q)
res = rem(rem(x, q) + rem(y, q), q);
end % End of 'ModuloAdd' function

function res = ModuloMult(x, y, q)
res = rem(rem(x, q) * rem(y, q), q);
end % End of 'ModuloMult' function

```

### 2.2.3 Сложность алгоритма

$O(n)$  в среднем, в худшем  $O(nm)$

### 2.2.4 Результаты работы

$m = [5, 10, 20]$  (красный, зеленый, синий)

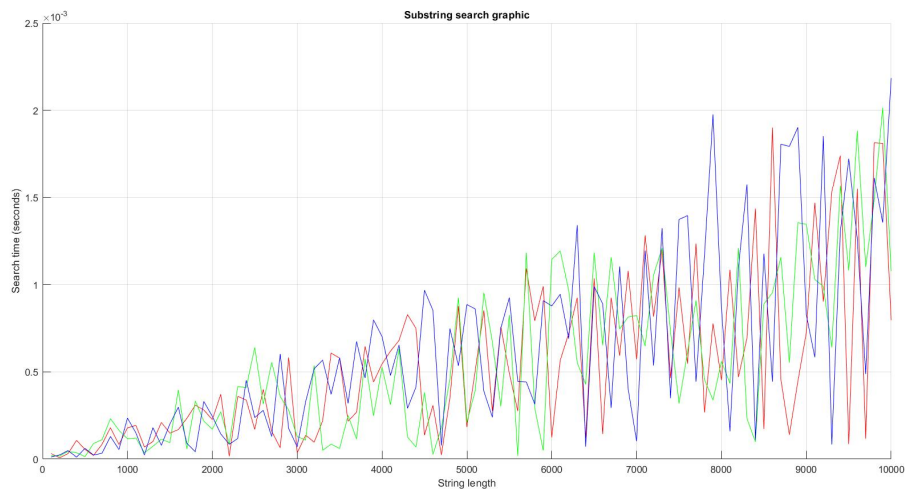


Рис. 2: Результат работы алгоритма Рабина-Карпа

## 2.3 Алгоритм Кнута-Морриса-Пратта

### 2.3.1 Описание алгоритма

Дана цепочка  $T$  и образец  $pattern$ . Требуется найти все позиции, начиная с которых  $pattern$  входит в  $T$ . Построим строку  $S = pattern\#str$ , где  $\#$  — любой символ, не входящий в алфавит  $pattern$  и  $str$ . Посчитаем на ней значение префикс-функции  $p$ . Заметим, что по определению префикс-функции при  $i > |pattern|$  и  $p[i] = |pattern|$  подстроки длины  $pattern$ , начинающиеся с позиций 0 и  $i - |pattern| + 1$ , совпадают. Если в какой-то позиции  $i$  выполняется условие  $p[i] = |pattern|$ , то в этой позиции начинается очередное вхождение образа в цепочку.

### 2.3.2 Код программы

Листинг 3: Алгоритм Рабина-Карпа

```
function [index] = KnuthMorrisPratt(str, pattern)

n = length(str);
m = length(pattern);

a = [pattern, '#', str];
pref = PrefixFuction(a);

for i = 1 : n
    if pref(m + i) == m
        index = i - m;
        return;
    end
end
end % End of 'KnuthMorrisPratt' function

function pref = PrefixFuction(str)
n = length(str);
pref = zeros(1, n);
for i = 2 : n
    k = pref(i - 1);
    while k > 0 && str(i) ~= str(k + 1)
        k = pref(k);
    end
    if str(i) == str(k + 1)
        k = k + 1;
    end
    pref(i) = k;
end
end % End of 'PrefixFuction' function
```

### 2.3.3 Сложность алгоритма

$O(n + m)$

### 2.3.4 Результаты работы

$m = [10, 30, 70]$  (красный, зеленый, синий)

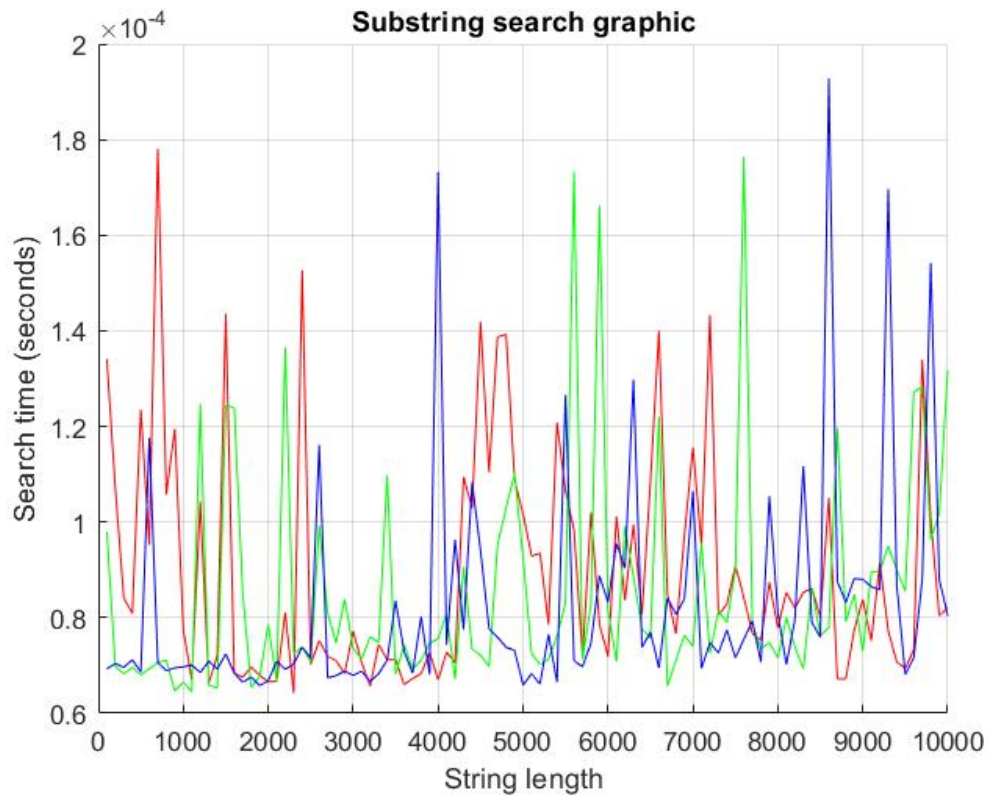


Рис. 3: Результат работы алгоритма Кнута-Морриса-Пратта

### 3 Вывод

Мы сравнили все 3 алгоритма. Тестовый код:

Листинг 4: Тестовый скрипт

```
n = 100 : 100 : 10000;
m = 5 : 5 : 20;
t1 = zeros(length(m), length(n));
t2 = zeros(length(m), length(n));
t3 = zeros(length(m), length(n));
array = 'abcdefghijklmnopqrstuvwxyz';

for i = 1 : length(m)
    for j = 1 : length(n)
        for k = 1 : n(j)
            str(k) = array(floor(length(array) * rand(1, 1)) + 1);
        end

        index = floor((n(j) - m(i)) * rand(1, 1)) + 1;
        substr = str(index : index + m(i) - 1);

        tic
        resIndex = Pos(str, substr);
```

```

    t1(i, j) = t1(i, j) + toc;

    if (resIndex ~= index)
        display("Wrong result.");
    end

    tic
    resIndex = RabinKarp(str, substr);
    t2(i, j) = t2(i, j) + toc;

    if (resIndex ~= index)
        display("Wrong result.");
    end

    tic
    resIndex = KnuthMorrisPratt(str, substr);
    t3(i, j) = t3(i, j) + toc;

    if (resIndex ~= index)
        display("Wrong result.");
    end
end
end
figure;
hold on;

grid on;
title('Substring search graphic');
xlabel('String length');
ylabel('Search time (seconds)');

for i = 1 : length(n)
    tmp(i) = t1(2, i);
end
plot(n, tmp, 'r')

for i = 1 : length(n)
    tmp(i) = t2(2, i);
end
plot(n, tmp, 'g')

for i = 1 : length(n)
    tmp(i) = t3(2, i);
end
plot(n, tmp, 'b')

```



Проведем сравнение при  $m = 10$ . Результаты представлены ниже (красный - **примитивный** алгоритм, зеленый - **алгоритм Рабина-Карпа**, **красный** - **алгоритм Кнута-Морриса-Пратта**):

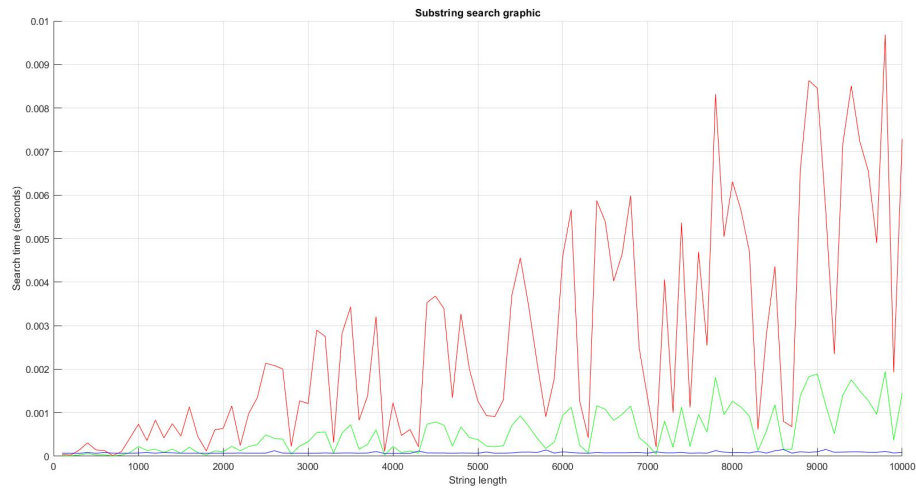


Рис. 4: Результат сравнения всех трех алгоритмов

Результат сравнения скоростей алгоритмов:

1. **Алгоритм Кнута-Морриса-Пратта**
2. **Алгоритм Рабина-Карпа**
3. **Примитивный алгоритм**

## 4 Библиографический список

1. <https://neerc.ifmo.ru>
2. <https://ru.wikipedia.org/>
3. Дж. Макконелл "Анализ алгоритмов"