

Отчет по лабораторной работе №7  
по курсу "Анализ алгоритмов"  
по теме "Методы поиска подстроки в строке"

Студент: Барсуков Н.М. ИУ7-56  
Преподаватель: Волкова Л.Л., Строганов Ю.В.

# Содержание

<b>1</b>	<b>Аналитический раздел</b>	<b>3</b>
1.1	Цель и задачи . . . . .	3
1.2	Алгоритмы . . . . .	3
1.2.1	Обзор алгоритма Кнута-Мориса-Прата . . . . .	3
1.3	Обзор алгоритма Бойера - Мура . . . . .	5
1.3.1	Сканирование слева направо, сравнение справа налево . . . . .	5
1.3.2	Эвристика стоп-символа . . . . .	5
1.3.3	Эвристика совпавшего суффикса . . . . .	6
1.4	Вывод . . . . .	6
<b>2</b>	<b>Конструкторский раздел</b>	<b>7</b>
<b>3</b>	<b>Технологический раздел</b>	<b>11</b>
3.1	Требования к программному обеспечению . . . . .	11
3.2	Средства реализации . . . . .	11
3.3	Интерфейс . . . . .	11
3.4	Листинг . . . . .	11
<b>4</b>	<b>Исследовательский раздел</b>	<b>13</b>
4.1	Характеристики оборудования . . . . .	13
4.2	Замеры . . . . .	13
<b>5</b>	<b>Заключение</b>	<b>15</b>
	<b>Список использованных источников</b>	<b>16</b>

## Введение

Поиск подстроки в строку - одна из простейших задач поиска информации. Применяется в виде встроенной функции в текстовых редакторах, СУБД, поисковых машинах, языках программирования и т. п. В задачах поиска традиционно принято обозначать шаблон поиска как *needle* (с англ. игла), а строку, в которой ведется поиск как *haystack* (с англ - стог сена). Обычно через  $\Sigma$  обозначается алфавит, на котором проводится поиск.

# 1 Аналитический раздел

В данном разделе описана цель и задачи необходимые для выполнения он-ной. Дано полное описание алгоритма Кнута - Морриса - Прата и Бойера - Мура и их математические описания.

## 1.1 Цель и задачи

Целью данной работы является изучение способов поиска подстроки в строке с помощью алгоритмов Кнута-Морриса-Прата и Бойера-Мура. Для достижения поставленной цели требуется решить следующие задачи:

1. изучить алгоритмы:
  - (a) стандартный;
  - (b) Кнута-Морриса-Прата;
  - (c) Бойера-Мура;
2. математически описать решение задачи поиска подстроки на основании выше указанных алгоритмов;
3. реализовать выше указанные алгоритмы;
4. сравнить;
5. сделать выводы.

## 1.2 Алгоритмы

### 1.2.1 Обзор алгоритма Кнута-Морриса-Прата

В задачах поиска информации одной из важнейших задач является поиск точно заданной подстроки в строке. Примитивный алгоритм поиска подстроки в строке основан на переборе всех подстрок, длина которых равна длине шаблона поиска, и посимвольном сравнении таких подстрок с шаблоном поиска.

Алгоритм Кнута-Морриса-Пратта является одним из первых алгоритмов с линейной оценкой в худшем случае.

Обозначим через:

$$n = |heystack|, m = |needle| \quad (1)$$

где  $|word|$ , обозначает длину слова  $word$

Префикс-функция строки  $\pi(S, i)$  – это длина наибольшего префикса строки  $S[1..i]$ , который не совпадает с этой строкой и одновременно является ее суффиксом. Проще говоря, это длина наиболее длинного начала строки,

Таблица 1: Пример префикс функции для строки abcdabca

$S[i]$	a	b	c	d	a	b	c	a
$\pi(S, i)$	0	0	0	0	1	2	3	1

являющегося также и ее концом. Для строки  $S$  удобно представлять префикс функцию в виде вектора длиной  $|S|-1$ . Можно рассматривать префикс-функцию длины  $|S|$ , положив  $\pi(S, 1) = 0$ . Пример приведен на таблице 1.2.1

Наиболее полный алгоритм вычисления префикс-функции на псевдокоде показан в листинге 1

---

**Algorithm 1:** Псевдокод функции поиска префикса

---

```

1 алг префикс_функция (арг строка  $S[1 \dots i]$ ) begin
2   Предположим что  $\pi(S, i) = k$ ;
3   if  $S[i + 1] = S[k + 1]$ , то then
4     |  $\pi(S, i + 1) = k + 1$ ;
5   else
6     | if  $k = 0$  then
7       |  $\pi(S, i + 1) = 0$ .;
8     | else
9       | Положить  $k_i = \pi(S, k)$  и перейти к шагу 3;
```

---

Рассмотрим алгоритм Кнута-Морриса-Пратта. Пусть  $S[0 \dots m-1]$  – образец,  $T[0 \dots n-1]$  – строка, в которой ведется поиск. Рассмотрим сравнение строк на позиции  $i$ , то есть образец  $S[0 \dots m-1]$  сопоставляется с частью строки  $T[i \dots i+m-1]$ . Предположим, первое несовпадение произошло между символами  $S[j]$  и  $T[i+j]$ , где  $i < j < m$ . Обозначим  $P = S[0 \dots j-1] = T[i \dots i+j-1]$ . При сдвиге можно ожидать, что префикс  $S$  сойдется с каким-либо суффиксом строки  $P$ . Поскольку длина наиболее длинного префикса, являющегося одновременно суффиксом, есть префикс-функция от строки  $S$  для индекса  $j$ , приходим к следующему алгоритму.

- 1) шаг 1: построить префикс-функцию образца  $S$ , обозначим ее  $F$ ;
- 2) шаг 2: положить  $k = 0, i = 0$ ;
- 3) шаг 3: сравнить символы  $S[k]$  и  $T[i]$ . Если символы равны, увеличить  $k$  на 1. Если при этом  $k$  стало равно длине образца, то вхождение образца  $S$  в строку  $T$  найдено, индекс вхождения равен  $i - |S| + 1$ . Алгоритм завершается. Пока  $k > 0$ , присвоим  $k = F[k-1]$  и переходим в начало шага 3;
- 4) шаг 4: Пока  $i < |T|$ , увеличиваем  $i$  на 1 и переходим к шагу 3.

### 1.3 Обзор алгоритма Бойера - Мура

Алгоритм сравнивает символы шаблона  $x$  справа налево, начиная с самого правого, один за другим с символами исходной строки  $y$ . Если символы совпадают, производится сравнение предпоследнего символа шаблона и так до конца. Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и поиск окончен. В случае несовпадения какого-либо символа (или полного совпадения всего шаблона) он использует две предварительно вычисляемых эвристических функции, чтобы сдвинуть позицию для начала сравнения вправо.

Алфавит обозначим буквой  $\Sigma$ . Пусть  $|y| = n, |x| = m$  и  $|\Sigma| = \sigma$ . Предположим, что в процессе сравнения возникает несовпадение между символом  $x[i] = a$  шаблона и символом  $y[i+j] = b$  исходного текста при проверке на позиции  $j$ . Тогда:

$$x[i+1 \dots m-1] = y[i+j+1 \dots j+m-1] = u \quad (2)$$

и

$$x[i] \neq y[i+j] \quad (3)$$

тогда  $m - i - 1$  символом шаблона не совпало (оставшийся “хвост” подстроки). В целом алгоритм можно описать с помощью трех ключевых положений.

#### 1.3.1 Сканирование слева направо, сравнение справа налево

Совмещается начало текста (строки) и шаблона, проверка начинается с последнего символа шаблона. Если символы совпадают, производится сравнение предпоследнего символа шаблона и т. д. Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и выполняется поиск следующего вхождения подстроки. Если же какой-то символ шаблона не совпадает с соответствующим символом строки, шаблон сдвигается на несколько символов вправо, и проверка снова начинается с последнего символа.

#### 1.3.2 Эвристика стоп-символа

Эвристика стоп-символа присутствует в большинстве описаний алгоритма Бойера — Мура, включая оригинальную статью Бойера и Мура, но не является необходимой для достижения оценки  $O(n+m)$  времени работы. Предположим, что мы производим поиск слова «колокол». Первая же буква не совпала — «к» (назовём эту букву стоп-символом). Тогда можно сдвинуть шаблон вправо до последней его буквы «к», что показано в таблице 2

Таблица 2: Поиск слова колокол в строке к

Строка	*	*	*	*	*	*	к	*	*	*	*	*	*
Шаблон	к	о	л	о	к	о	л						
След. шаг			к	о	л	о	к	о	л				

Если стоп-символ «к» оказался за другой буквой «к», эвристика стоп-символа не работает. В таких ситуациях может быть полезна третья идея алгоритма Бойера — Мура — эвристика совпавшего суффикса.

### 1.3.3 Эвристика совпавшего суффикса

Если при чтении шаблона справа налево совпал суффикс  $S$ , а символ  $b$ , стоящий перед  $S$  в шаблоне (т. е. шаблон имеет вид  $PbS$ ), не совпал, то эвристика совпавшего суффикса сдвигает шаблон на наименьшее число позиций вправо так, чтобы строка  $S$  совпала с шаблоном, а символ, предшествующий в шаблоне данному совпадению  $S$ , отличался бы от  $b$  (если такой символ вообще есть). Для данного шаблона  $s[0 \dots m-1]$  считается целочисленный массив  $\text{suffshift}[0 \dots m]$ , в котором  $\text{suffshift}[i]$  равно минимальному числу  $j > 0$ , такому, что  $s[i-j] \neq s[i-1]$  и  $s[i-j+k] = s[i-1+k]$  для любого  $k > 0$ , для которого выполняется  $0 \leq i-j+k < m$  и  $0 \leq i-1+k < m$ . 3

Таблица 3: Поиск слова “скалкала” в строке, рассмотрение суффикса “рка”

Строка	*	*	*	*	*	*	р	к	а	*	*	*	*	*	
Шаблон	с	к	а	л	к	а	л	к	а						
След. шаг							с	к	а	л	к	а	л	к	а

В данном случае совпал суффикс «ка», и шаблон сдвигается вправо до ближайшего «ка», перед которым нет буквы «л».

## 1.4 Вывод

В данном разделе было приведено общее и алгоритмическое описание алгоритмов Кнута-Морриса-Пратта и Бойера-Мура, приведены примеры.

## 2 Конструкторский раздел

В данном разделе приведены схемы алгоритмов:

- 1) Кнута-Морриса-Пратта
- 2) Бойера Мура



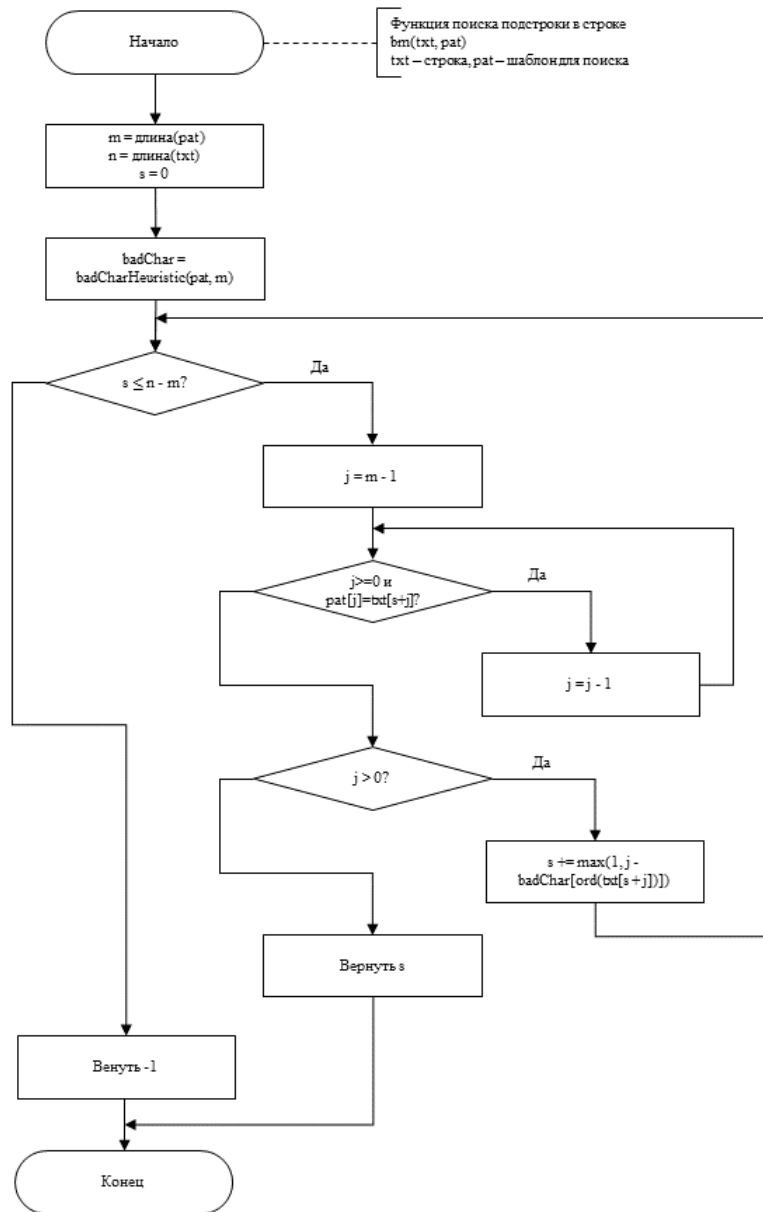


Рис. 1:

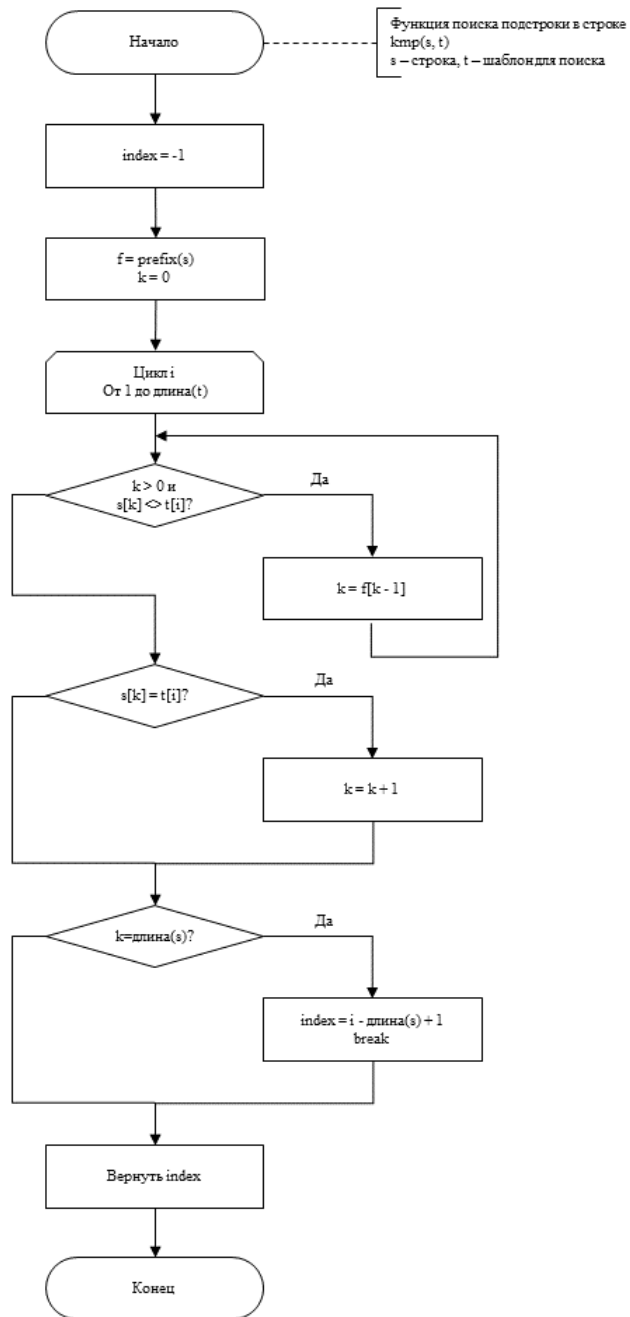


Рис. 2:

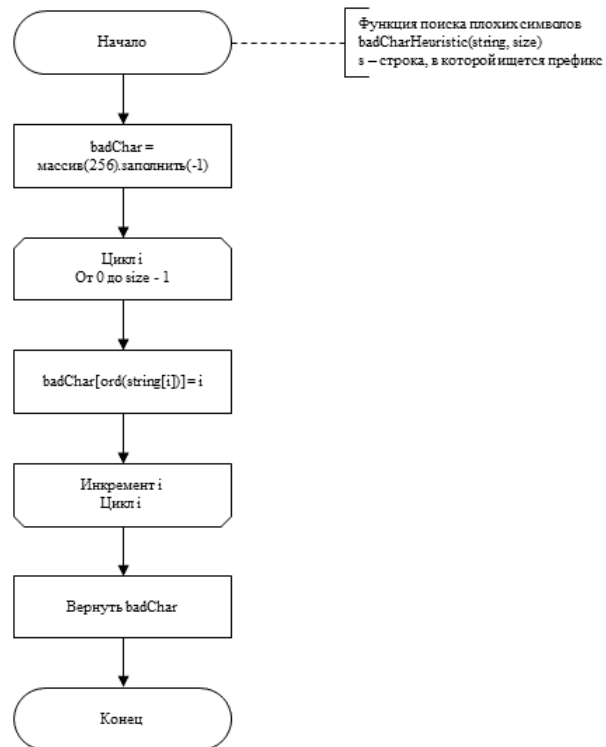


Рис. 3:

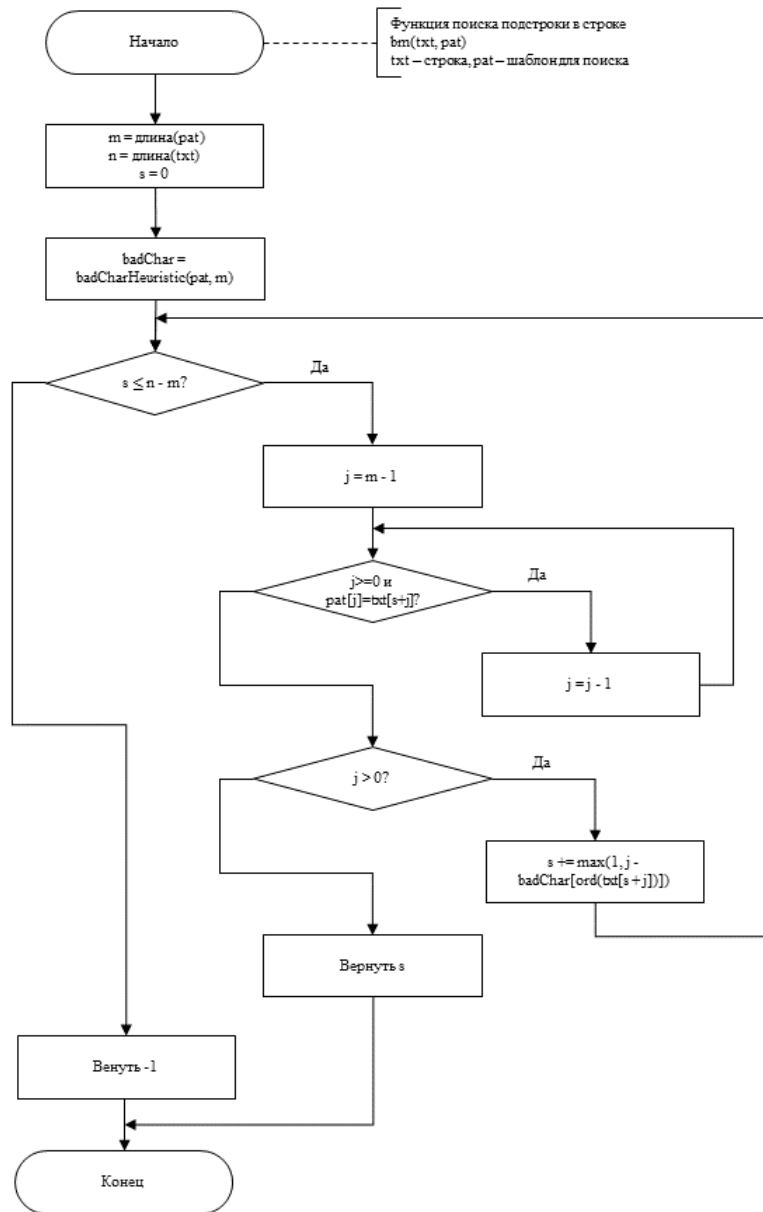


Рис. 4:

В данном разделе были приведены схемы алгоритмов поиска подстроки

в строке Кнута-Морриса-Пратта и Бойера Мура

## 3 Технологический раздел

### 3.1 Требования к программному обеспечению

Минимальные системные требования: PC с операционной системой Windows версии XP/Vista/7/8/10. Требуются устройства ввода: клавиатура, мышь.

### 3.2 Средства реализации

Для выполнения работы был выбран язык программирования Python ввиду его простоты. И SublimeText3 text

### 3.3 Интерфейс

Интерфейс из себя представляет простую консоль, где пользователю ничего делать не нужно.

### 3.4 Листинг

В данном подразделе приведены листинги программ.

1. алгоритм Кнута - Мориса - Прата 1
2. алгоритм Бойера - Мура 2

Листинг 1: Алгоритм Кнута-Мориса-Прата

```
1
2     def prefix(s):
3         v = [0] * len(s)
4         for i in range(1, len(s)):
5             k = v[i - 1]
6             while k > 0 and s[k] != s[i]:
7                 k = v[k - 1]
8             if s[k] == s[i]:
9                 k = k + 1
10            v[i] = k
11        return v
12
13    def kmp(s, t):
14        index = -1
15        f = prefix(s)
16        k = 0
17        for i in range(len(t)):
18            while k > 0 and s[k] != t[i]:
19                k = f[k - 1]
20            if s[k] == t[i]:
21                k = k + 1
22            if k == len(s):
```

```

23         index = i - len(s) + 1
24         break
25     return index

```

Листинг 2: Алгоритма Бойера-Мура

```

1  def badCharHeuristic(string, size):
2      badChar = [-1] * 256
3
4      for i in range(size):
5          badChar[ord(string[i])] = i
6
7      return badChar
8
9
10 def search(txt, pat):
11     m = len(pat)
12     n = len(txt)
13     badChar = badCharHeuristic(pat, m)
14     s = 0
15     while s <= n - m:
16         j = m - 1
17
18         while j >= 0 and pat[j] == txt[s + j]:
19             j -= 1
20
21         if j < 0:
22             return s # Return only first entry
23
24         else:
25             s += max(1, j - badChar[ord(txt[s + j])])
26
27     return -1

```

В данном разделе были приведены листинги алгоритмов Кнута-Мориса-Прата и Бойера-Мура на языке программирования Python, а также была приведена функция тестирования этих алгоритмов.

## 4 Исследовательский раздел

### 4.1 Характеристики оборудования

1) Компьютер:

- (a) Тип компьютера Компьютер с ACPI на базе x64;
- (b) Операционная система Microsoft Windows 10 Pro.

2) Системная плата:

- (a) тип ЦП DualCore Intel Core i5-6200U, 2700 MHz (27 x 100);
- (b) системная плата HP 8079;
- (c) чипсет системной платы Intel Sunrise Point-LP, Intel Skylake-U;
- (d) системная память 8072 МБ (DDR4 SDRAM).

### 4.2 Замеры

Замеры для случайно сгенерированных слов размером от 10,000 до 1,000,000 с шагом 10,000. Случайным образом в слове  $x$  выбирался срез длиной 1,000 символом  $y$ , далее  $x$  и  $y$  подаются на вход алгоритмам поиска подстроки в строке. Алгоритмы тестировались на больших длинах строк, так как на строках в диапазоне от 1 до 1,000 среднее время работы алгоритмов было 0,00017 секунд. Эксперимент с замерами времени работы алгоритмов в этом диапазон длин строк может оказаться некорректным, так как из-за быстрой работы на замеры времени могут влиять системные прерывания и другие процессы, усложнив поиск типа зависимости скорости работы от длины строк. На момент замера времени работало в среднем 76 активных процессов. Результаты эксперимента являются воспроизводимыми. Скорость работы обоих алгоритмов линейно зависит от количества символов в строке, в которой ведется поиск подстроки. 5



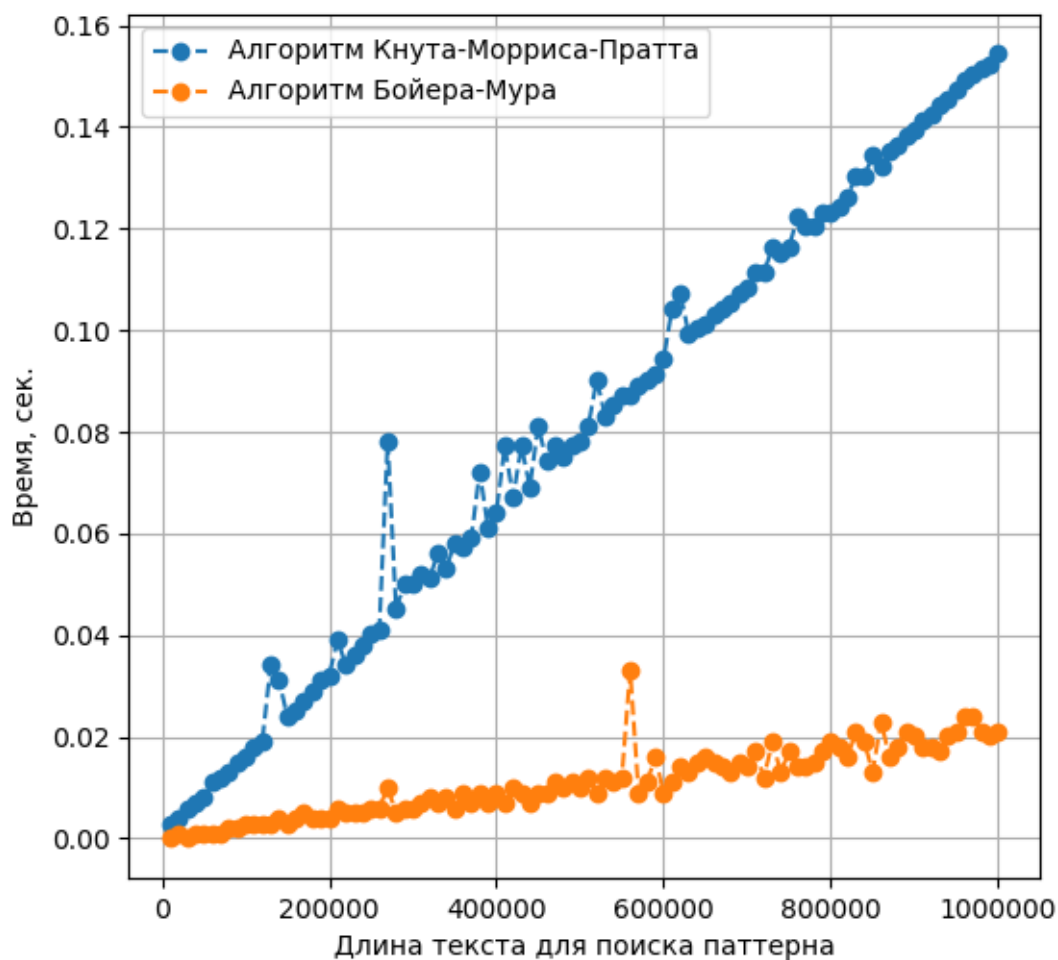


Рис. 5:

В данном разделе было проведено экспериментальное сравнение алгоритмов Кнута-Морриса-Пратта и Бойера-Мура поиска подстроки в строке по времени работы, была найдена зависимость алгоритмов от длин строк и проведено тестирование на корректность работы. Опыт выявил, что алгоритм Бойера-Мура работает в среднем в 6 раз быстрее алгоритма Кнута-Морриса-Пратта. Оба алгоритма имеют линейную сложность в зависимости от суммы длин строки и подстроки.

## 5 Заключение

В результате выполнения данной работы были рассмотрены способы решения задачи поиска подстроки в строке с помощью алгоритмов Кнута-Мориса-Прата и Бойера-Мура. Опыт показал, что рекомендуется использовать алгоритм Бойера-Мура, так как он показал большую производительность по времени в сравнении с алгоритмом Кнута-Мориса-Прата. Было изучено применение алгоритмов Кнута-Мориса-Прата и Бойера-Мура к задаче поиска подстроки в строке. Математически описано решение задачи поиска подстроки в строке на основании алгоритмов Кнута-Мориса-Прата и Бойера-Мура. Реализованы алгоритмы Кнута-Мориса-Прата и Бойера-Мура. Сравнены алгоритмы Кнута-Мориса-Прата и Бойера-Мура по скорости работы. Сделаны выводы о применимости алгоритмов Кнута-Мориса-Прата и Бойера-Мура к решению задачи поиска подстроки в строке.

## Список литературы