

Федеральное государственное бюджетное образовательное учреждение  
высшего образования «Московский государственный технический университет  
имени Н. Э. Баумана» (национальный исследовательский университет)

Дисциплина: «Анализ алгоритмов»

Отчет по лабораторной работе №7

Тема работы:

«Алгоритмы поиска подстроки в строке»

Студент: Васюков А. В.

Группа: ИУ7-52Б

Преподаватели: Волкова Л. Л.,

Строганов Ю. В.

Москва, 2019 г.

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Аналитический раздел</b>	<b>4</b>
1.1 Описание алгоритмов . . . . .	4
1.1.1 Алгоритм Кнута-Морриса-Пратта . . . . .	4
1.1.2 Алгоритм Бойера-Мура . . . . .	4
Выводы . . . . .	5
<b>2 Конструкторский раздел</b>	<b>6</b>
2.1 Разработка алгоритмов . . . . .	6
2.2 Сравнительный анализ алгоритмов . . . . .	9
2.2.1 Пример работы алгоритма Кнута-Морриса-Пратта . . . . .	9
2.2.2 Пример работы алгоритма Бойера-Мура . . . . .	10
2.2.3 Оценка времени работы алгоритма Кнута-Морриса-Пратта	11
2.2.4 Оценка времени работы алгоритма Бойера-Мура . . . . .	11
Выводы . . . . .	11
<b>3 Технологический раздел</b>	<b>13</b>
3.1 Требования к программному обеспечению . . . . .	13
3.2 Средства реализации . . . . .	13
3.3 Листинг программы . . . . .	14
3.4 Тестовые данные . . . . .	16
Выводы . . . . .	16
<b>4 Исследовательский раздел</b>	<b>17</b>
4.1 Примеры работы . . . . .	17
4.2 Постановка эксперимента . . . . .	19
4.3 Сравнительный анализ на основе эксперимента . . . . .	19
Выводы . . . . .	20
<b>Заключение</b>	<b>21</b>
<b>Список литературы</b>	<b>22</b>

# Введение

**Поиск подстроки в строке** — важная задача поиска информации. Применяется в виде встроенной функции в текстовых редакторах, СУБД, поисковых машинах и языках программирования.

Такой поиск приходится проводить довольно часто, поэтому необходимо, чтобы он осуществлялся как можно быстрее. Становится ясно, что наивный алгоритм с полным перебором всех частей строки, длины которых соответствуют длине шаблона, является не самым эффективным способом решения такой задачи.

Существует немалое количество алгоритмов, справляющихся с поиском подстроки, лучше, чем примитивный перебор. Особенно красиво реализуются алгоритмы, основанные на применении конечных автоматов.

**Цель лабораторной работы:** изучение метода динамического программирования на материале алгоритмов поиска подстроки в строке.

## **Задачи работы:**

- 1) изучить алгоритмы поиска подстроки в строке Кнута-Морриса-Пратта и Бойера-Мура;
- 2) применить метод динамического программирования для реализации указанных алгоритмов;
- 3) провести сравнительный анализ алгоритмов по времени выполнения;
- 4) экспериментально подтвердить различия во временной эффективности алгоритмов при помощи разработанного программного обеспечения на материале замеров процессорного времени на строках различных размеров;
- 5) описать и обосновать полученные результаты в отчете о лабораторной работе, выполненного как расчётно-пояснительная записка.

# 1 Аналитический раздел

В данном разделе будут рассмотрены выбранные алгоритмы поиска подстроки в строке.

## 1.1 Описание алгоритмов

### 1.1.1 Алгоритм Кнута-Морриса-Пратта

Алгоритм Кнута-Морриса-Пратта основан на принципе конечного автомата. В этом алгоритме состояния помечаются символами, совпадение с которыми должно в данный момент произойти. Из каждого состояния имеется два перехода: один соответствует успешному сравнению, другой — несовпадению. Успешное сравнение переводит нас в следующий узел автомата, а в случае несовпадения мы попадаем в предыдущий узел, отвечающий образцу. Пример автомата Кнута-Морриса-Пратта для подстроки *ababcb* приведен на рис. 1. [1]

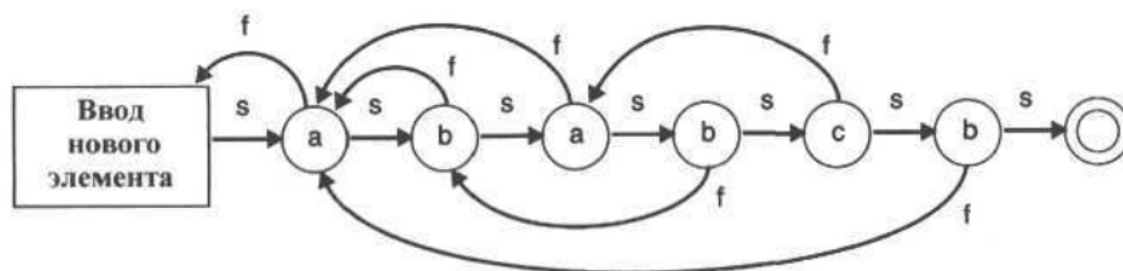


Рис. 1: Полный автомат Кнута-Морриса-Пратта для подстроки *ababcb*

При всяком переходе по успешному сравнению в конечном автомате Кнута-Морриса-Пратта происходит выборка нового символа из текста. Переходы, отвечающие неудачному сравнению, не приводят к выборке нового символа; вместо этого они повторно используют последний выбранный символ. Если мы перешли в конечное состояние, то это означает, что искомая подстрока найдена.

### 1.1.2 Алгоритм Бойера-Мура

Алгоритм Бойера-Мура осуществляет сравнение с образцом справа налево, а не слева направо, как алгоритм Кнута-Морриса-Пратта. Исследуя искомый образец, можно осуществлять более эффективные прыжки в тексте при обнаружении несовпадения. [1]

Рассмотрим строку *there they are* и подстроку *they*. Здесь сначала сравнивается *y* с *r* и обнаруживается несовпадение. Поскольку известно, что буква *r*

вообще не входит в образец, можно сдвинуться в тексте на целых четыре буквы (т. е. на длину образца) вправо. Затем сравнивается буква  $y$  с  $h$  и вновь обнаруживается несовпадение. Однако поскольку на этот раз  $p$  входит в образец, есть возможность сдвинуться вправо только на две буквы так, чтобы буквы  $h$  совпали. Затем начинается сравнение справа и обнаруживается полное совпадение кусочка текста с образцом. В алгоритме Бойера—Мура делается 6 сравнений вместо 13 сравнений в примитивном алгоритме, в котором рассматриваются все подстроки исходной строки, совпадающие по длине с шаблоном.

Пример проиллюстрирован в таблице 1.

Таблица 1: Поиск образца *they* в тексте *there they are*

	t	h	e	r	e		t	h	e	y		a	r	e
1	t	h	e	y										
2						t	h	e	y					
3							t	h	e	y				

Алгоритм Бойера-Мура обрабатывает образец двумя способами. Во-первых, можно вычислить величину возможного сдвига при несовпадении очередного символа. Во-вторых, вычисляется величина прыжка. Эти действия помогают эффективно подстраивать шаблон под новую подстроку.

## Выводы

Рассмотрены алгоритмы поиска подстроки в строке, предоставляющие, очевидно, более эффективное решение в сравнении с наивным алгоритмом. Обоснованы различия между приведенными алгоритмами и выделены ключевые моменты каждого из них.

## 2 Конструкторский раздел

В разделе приводятся схемы выбранных алгоритмов поиска подстроки в строке, производится их теоретический сравнительный анализ.

### 2.1 Разработка алгоритмов

На рис. 2-4 приведены схемы алгоритмов поиска подстроки.

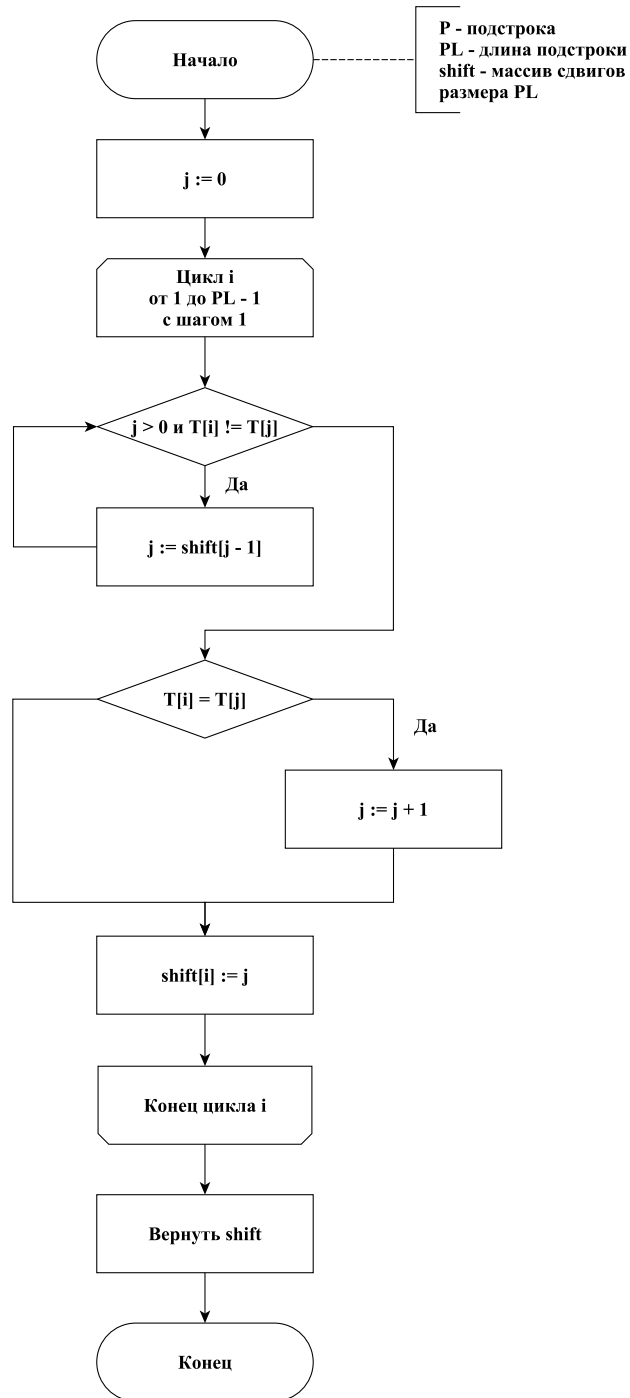


Рис. 2: Заполнение массива сдвигов

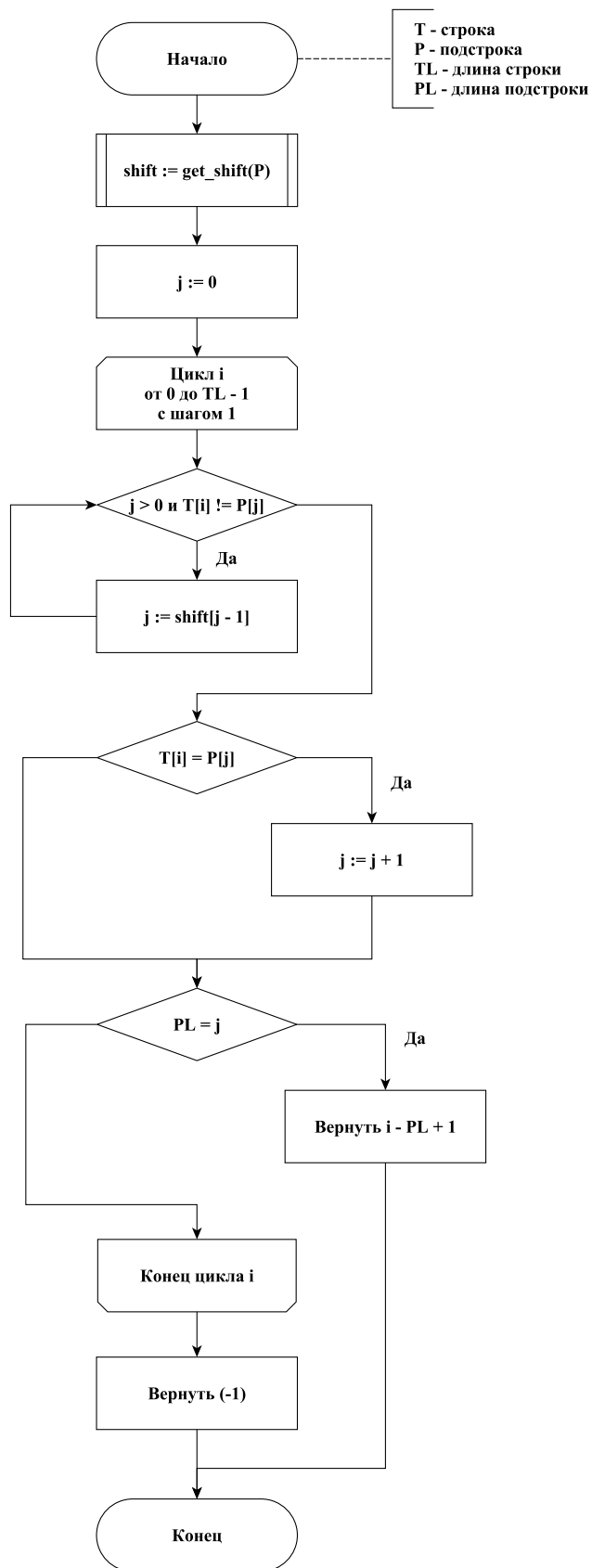


Рис. 3: Алгоритм Кнута-Морриса-Пратта

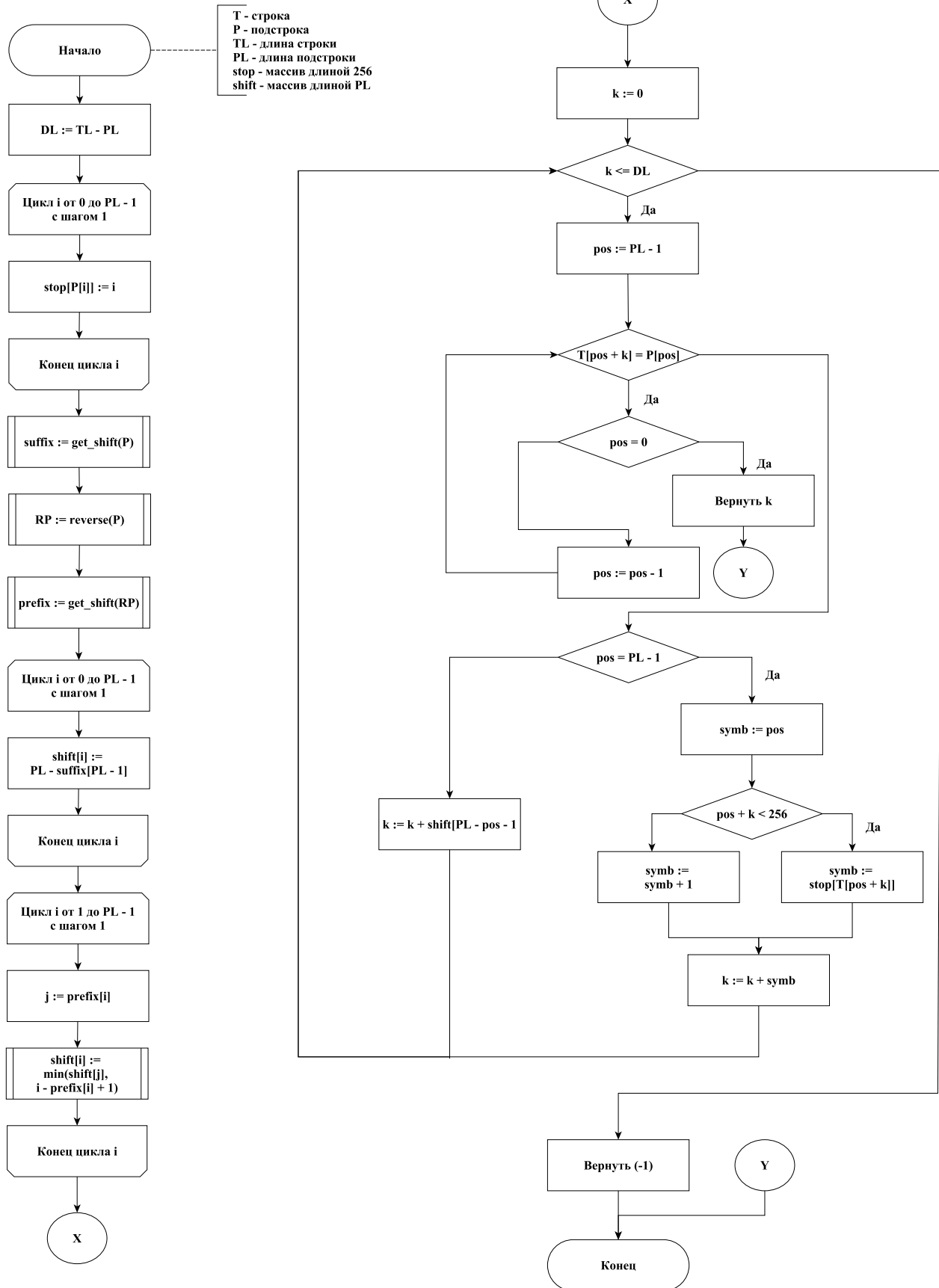


Рис. 4: Алгоритм Бойера-Мура



## 2.2 Сравнительный анализ алгоритмов

### 2.2.1 Пример работы алгоритма Кнута-Морриса-Пратта

Рассмотрим пример работы алгоритмов на строке *ababqcabababakababacqw* длиной 22 символа и подстроке *ababac* длиной 6 символов.

Сперва необходимо построить префикс-функцию для подстроки. Для этого последовательно пройдем по подстроке в поисках очередного максимального суффикса, равного префиксу. Пошаговый разбор представлен в таблице 2: слева отмечен индекс нового входящего символа, посередине — рассматриваемая строка, справа — значение префикс-функции. *Курсивом* и **жирным** выделены совпадающие максимальные *префиксы* и **суффиксы**.

Таблица 2: Нахождение префикс-функции

Индекс	Строка	Префикс-функция
0	a	0
1	a b	0
2	a b <b>a</b>	1
3	a <i>b</i> <b>a</b> <b>b</b>	2
4	a <i>b</i> <b>a</b> <b>b</b> <b>a</b>	3
5	a b a b a c	0

Далее уже применим похожий подход для поиска подстроки. При несовпадении символов, будем смещать подстроку таким образом, чтобы префикс строки (до нового рассматриваемого символа этой строки) был равен некоторому префиксу подстроки (табл. 3).

Таблица 3: Пример работы алгоритма Кнута-Морриса-Пратта

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	a	b	a	b	<b>q</b>	c	a	b	a	b	a	<b>b</b>	a	<b>k</b>	a	b	a	b	a	c	q	w
1	<b>a</b>	<b>b</b>	<b>a</b>	<b>b</b>	<b>a</b>	c																
2					<b>a</b>	b	a	b	a	c												
3						<b>a</b>	<b>b</b>	<b>a</b>	<b>b</b>	<b>a</b>	c											
4								<b>a</b>	<b>b</b>	<b>a</b>	<b>b</b>	a	c									
5															<b>a</b>	<b>b</b>	<b>a</b>	<b>b</b>	<b>a</b>	c		

- 1) несовпадение происходит на символе с индексом 4, подходящего префикса нет, поэтому следующее сравнение начинаем с 5 символа;
- 2) аналогичная ситуация;
- 3) несовпадение на 11 символе, суффикс *abab* равен префиксу подстроки, значение префикс-функции для данной части подстроки равно 2, поэтому имеем соответствующее смещение;

- 4) аналогичный случай в 1-2 этапах;
- 5) полное совпадение.

Для нахождения потребовалось 5 сравнений на первом этапе, 1 — на втором, по 6 — на остальных этапах. Итого, 24 сравнения. В наивном алгоритме потребовалось бы 38 сравнений.

### 2.2.2 Пример работы алгоритма Бойера-Мура

Посмотрим, как работает алгоритм Бойера-Мура на тех же входных данных.

В алгоритме Бойера-Мура необходимо дополнительно найти массив стоп-символов и массив сдвигов. Длина массива стоп-символов равна мощности алфавита, над которым производятся действия. Для каждого уникального символа подстроки вычислим индекс самого последнего вхождения и занесем его в этот массив, для остальных символов поставим значение равное -1. Итак, для  $a - 4$ ,  $b - 3$ ,  $c - 5$ .

Для каждого возможного суффикса шаблона указываем наименьшую величину, на которую нужно сдвинуть вправо шаблон, чтобы он снова совпал с префиксом и при этом символ, предшествующий этому вхождению, не совпал бы с символом, предшествующим суффиксу. Если такой сдвиг невозможен, ставится длина подстроки (табл. 4).

Таблица 4: Массив сдвигов в алгоритме Бойера-Мура

Индекс	0	1	2	3	4	5
Строка	a	b	a	b	a	c
Сдвиг	6	2	3	4	5	6

Теперь можно перейти непосредственно к поиску подстроки (табл. 5).

Таблица 5: Пример работы алгоритма Бойера-Мура

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	a	b	a	b	<b>q</b>	c	a	<b>b</b>	a	<b>b</b>	a	<b>b</b>	a	<b>k</b>	a	b	a	b	a	c	q	w
1	a	b	a	b	<b>a</b>	<b>c</b>																
2			a	b	a	b	a	<b>c</b>														
3					a	b	a	b	a	<b>c</b>												
4							a	b	a	b	a	<b>c</b>										
5									a	b	a	b	a	<b>c</b>								
6															<b>a</b>	<b>b</b>	<b>a</b>	<b>b</b>	<b>a</b>	<b>c</b>		

- 1) несовпадение происходит на символе с индексом 4, префикс  $ab$  совпадает с суффиксом, поэтому смещаемся на 2 в соответствии с массивом сдвигов;

- 2-5) аналогичная ситуация несовпадения, только смещение происходит по стоп-таблице, так как нет подходящего суффикса;
- 6) полное совпадение.

Для нахождения потребовалось 2 сравнения на первом этапе, 4 — на последнем этапе, по 1 — на всех остальных. Всего 12 сравнений.

### 2.2.3 Оценка времени работы алгоритма Кнута-Морриса-Пратта

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. Можно показать, что таблица  $\pi$  может быть вычислена (амортизационно) за  $O(m)$  сравнений перед началом поиска. А поскольку строка  $T$  будет пройдена ровно один раз, суммарное время работы алгоритма будет равно  $O(m+n)$ , где  $n$  — длина текста  $T$ . [2]

### 2.2.4 Оценка времени работы алгоритма Бойера-Мура

Ниже мы обозначаем через  $A$  — число символов в алфавите.

При подсчете массива сдвигов по одному присваиванию приходится на каждый элемент массива и еще по одному — на каждый символ образца. Поэтому общее число присваиваний равно  $O(m+A)$ . При вычислении массива прыжков в худшем случае каждый символ образца будет сравниваться со всеми последующими его символами.

Можно подсчитать, что число сравнений в худшем случае будет  $O(mn)$ . В лучшем случае асимптотика равна  $O(m+n)$ . [6]

## Выводы

В разделе представлены схемы алгоритмов поиска подстроки в строке Кнута-Морриса-Пратта и Бойера-Мура. Произведен теоретический анализ временной

сложности, обозначены различия в работе алгоритмов, пояснена их эффективность.

## 3 Технологический раздел

Здесь описываются требования к программному обеспечению и средства реализации, приводятся листинги программы и тестовые данные.

### 3.1 Требования к программному обеспечению

**Входные данные:**

- *text* — строка, в которой ведется поиск;
- *pattern* — искомая подстрока.

**Выходные данные:** индекс первого вхождения подстроки *pattern* в строке *text*; при отсутствии подстроки либо неверных данных (длина подстроки превышает длину строки) — сообщение об этом.

На рис. 5 приведена функциональная схема программы поиска подстроки в строке.

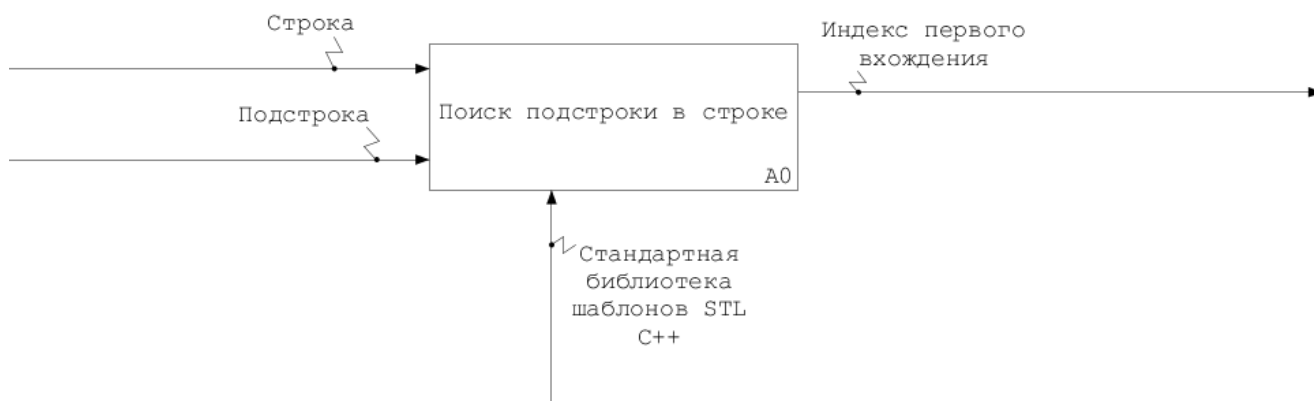


Рис. 5: Функциональная схема программы поиска подстроки в строке

### 3.2 Средства реализации

Программа написана на языке C++ (стандарт C++17), который предоставляет программисту мощные инструменты для реализации различных алгоритмов и является достаточно надежным и эффективным. [7] Для написания использовался редактор исходного кода *Visual Studio Code*, для сборки проекта применена система автоматизации *CMake* (флаги компиляции `-std=c++17 -Wall -Werror -pedantic`).

Для хранения массивов и строк применяются контейнерные классы из стандартной библиотеки шаблонов STL — `std::vector` и `std::string` соответственно.

Замер времени выполнения программы производится с помощью библиотеки *chrono*, функционал которой позволяет подсчитывать процессорное время в тиках, а затем конвертировать полученный результат в реальное время. [8]

### 3.3 Листинг программы

Реализованная программа представлена в листингах 1, 2 и 3.

Листинг 1: Реализация заполнения массива сдвигов

```
1 std::vector<int> get_shift(const std::string &str) {
2     int len = str.length();
3     std::vector<int> shift(len, 0);
4
5     for (int i = 1, j = 0; i < len; ++i) {
6         while (j > 0 && str[i] != str[j]) {
7             j = shift[j - 1];
8         }
9
10        if (str[i] == str[j]) {
11            ++j;
12        }
13
14        shift[i] = j;
15    }
16
17    return shift;
18 }
```

Листинг 2: Реализация алгоритма Кнута-Морриса-Пратта

```
1 int kmp_search(const std::string &text, const std::string &pattern) {
2     if (text.length() < pattern.length()) {
3         return WRONG_LEN;
4     }
5
6     const int text_len = text.length();
7     const int pattern_len = pattern.length();
8
9     auto shift = get_shift(pattern);
10
11    for (int i = 0, j = 0; i < text_len; ++i) {
12        while (j > 0 && text[i] != pattern[j]) {
13            j = shift[j - 1];
14        }
15
16        if (text[i] == pattern[j]) {
17            ++j;
18        }
19
20        if (pattern_len == j) {
21            return i - pattern_len + 1;
22        }
23    }
```

```

24
25     return NOT_FOUND;
26 }

```

### Листинг 3: Реализация алгоритма Бойера-Мура

```

1  int bm_search(const std::string &text, const std::string &pattern) {
2      if (text.length() < pattern.length()) {
3          return WRONG_LEN;
4      }
5
6      const int text_len = text.length();
7      const int pattern_len = pattern.length();
8      const int dif_len = text_len - pattern_len;
9
10     const int ascii_num = 256;
11     std::vector<int> stop_table(ascii_num, -1);
12     std::vector<int> shift_table(pattern_len);
13
14     for (int i = 0; i < pattern_len; ++i) {
15         stop_table[pattern[i]] = i;
16     }
17
18     std::string rev_pattern(pattern.rbegin(), pattern.rend());
19     auto suffix_table = get_shift(pattern);
20     auto prefix_table = get_shift(rev_pattern);
21
22     for (int i = 0; i < pattern_len + 1; ++i) {
23         shift_table[i] = pattern_len - suffix_table.back();
24     }
25
26     for (int i = 1; i < pattern_len; ++i) {
27         int j = prefix_table[i];
28         shift_table[i] = std::min(shift_table[j], i - prefix_table[i] + 1);
29     }
30
31     int shift = 0;
32     while (shift <= dif_len) {
33         int pos = pattern_len - 1;
34
35         while (text[pos + shift] == pattern[pos]) {
36             if (0 == pos) {
37                 return shift;
38             }
39
40             --pos;
41         }
42
43         if (pattern_len - 1 == pos) {
44             int stop_symb = pos;
45
46             if (pos + shift < ascii_num) {
47                 stop_symb -= stop_table[text[pos + shift]];
48             } else {
49                 ++stop_symb;
50             }

```

```

51         shift += stop_symb;
52     } else {
53         shift += shift_table[pattern_len - pos - 1];
54     }
55 }
56
57
58 return NOT_FOUND;
59 }

```

### 3.4 Тестовые данные

Для тестирования выделены следующие случаи, представленные в табл. 6.

Таблица 6: Тестовые данные

№	Строка	Подстрока	Результат
1	a	abc	Неверная длина
2	abc	abcd	Неверная длина
3	abc	def	Не найдено
4	abc	aa	Не найдено
5	abbaabbab	aaab	Не найдено
6	abc	a	0
7	abbaabbab	abba	0
8	abc	b	1
9	abbaabbab	baab	2
10	there they are	they	6
11	abc	c	2
12	abbaabbab	bbab	5
13	there they are	are	11
14	abcabc	a	0
15	abcabc	c	2
16	abbaabbabbbab	bbab	5
17	there they are are here are	are	11
18	a	a	0
19	abc	abc	0
20	abbaabbabbbab	abbaabbabbbab	0

## Выводы

В данном разделе были рассмотрены требования к программному обеспечению, обоснован выбор средств реализации, приведены листинги программы и тестовые данные.



## 4 Исследовательский раздел

В разделе представлены примеры выполнения программы, а также результаты исследования эффективности сортировок.

### 4.1 Примеры работы

На рис. 6-12 приведены примеры работы программы.

```
~~~~~
Input text : abc
Input pattern : abcd
~~~~~
Text = "abc"
Pattern = "abcd"
~~~~~
KMP = WRONG_LEN
BM = WRONG_LEN
~~~~~
```

Рис. 6: Некорректная длина

```
~~~~~
Input text : abc
Input pattern : aa
~~~~~
Text = "abc"
Pattern = "aa"
~~~~~
KMP = NOT_FOUND
BM = NOT_FOUND
~~~~~
```

Рис. 7: Подстрока не найдена

```
~~~~~
Input text : abbaabbab
Input pattern : abba
~~~~~
Text = "abbaabbab"
Pattern = "abba"
~~~~~
KMP = 0
BM = 0
~~~~~
```

Рис. 8: Подстрока в начале

```
~~~~~
Input text : abbaabbab
Input pattern : baab
~~~~~
Text = "abbaabbab"
Pattern = "baab"
~~~~~
KMP = 2
BM = 2
~~~~~
```

Рис. 9: Подстрока в середине

```
~~~~~
Input text : there they are
Input pattern : are
~~~~~
Text = "there they are"
Pattern = "are"
~~~~~
KMP = 11
BM = 11
~~~~~
```

Рис. 10: Подстрока в конце

```
~~~~~
Input text : abcabc
Input pattern : c
~~~~~
Text = "abcabc"
Pattern = "c"
~~~~~
KMP = 2
BM = 2
~~~~~
```

Рис. 11: Повтор подстроки

```
~~~~~
Input text : abc
Input pattern : abc
~~~~~
Text = "abc"
Pattern = "abc"
~~~~~
KMP = 0
BM = 0
~~~~~
```

Рис. 12: Строка равна подстроке

## 4.2 Постановка эксперимента

Необходимо сравнить время работы реализованных алгоритмов поиска подстроки в строке. Размер строки меняется от 100 тыс. до 1 млн. символов, длина шаблона – 100. Строка содержит только символы *a*, в подстроке символы *b* и *a* чередуются (реализуется худший случай для алгоритма Бойера-Мура).

## 4.3 Сравнительный анализ на основе эксперимента

Замеры произведены на 4-ядерном процессоре *Intel Core i5-8250U* с тактовой частотой 1,8 ГГц, оперативная память — 8 ГБ.

На рис. 13 приводятся графики сравнения времени выполнения выбранных алгоритмов.

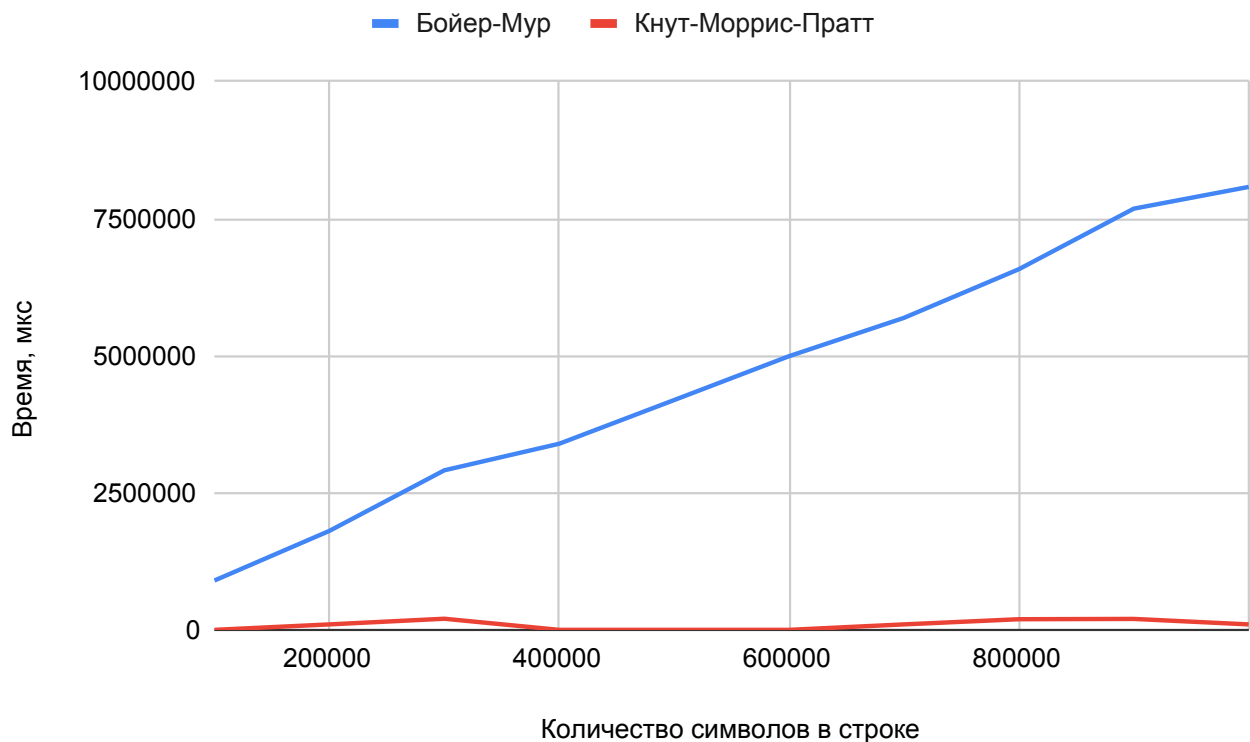


Рис. 13: Сравнение времени выполнения алгоритмов поиска подстроки в строке

Как видно, подтверждены теоретические расчеты. В худшем случае асимптотика алгоритма Бойера-Мура квадратична, а Кнута-Морриса-Пратта – линейна. Из-за того, что все символы *a* из текста повторяются в шаблоне *m* раз, эвристика хорошего суффикса будет пытаться сопоставить шаблон в каждой позиции (суммарно, *n* раз), а эвристика плохого символа в каждой позиции будет двигать строку *m* раз. Итого,  $O(mn)$ . [3]

## **Выводы**

Программа успешно прошла все заявленные тесты. Эксперименты замера времени подтвердили предположения и теоретические расчеты из раздела 2.2.

## Заключение

В ходе работы выполнено следующее:

- 1) изучены алгоритмы поиска подстроки в строке;
- 2) применен метод динамического программирования для реализации указанных алгоритмов;
- 3) проведен сравнительный анализ алгоритмов по времени выполнения;
- 4) экспериментально подтверждены различия во временной эффективности алгоритмов при помощи разработанного программного обеспечения на материале замеров процессорного времени на строках различных размеров;
- 5) описаны и обоснованы полученные результаты в отчете о лабораторной работе, выполненного как расчётно-пояснительная записка.

## Список литературы

- [1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.- М.:Техносфера, 2009.
- [2] Д. Кнут. Искусство программирования, М., Мир, 1978
- [3] Алгоритмы поиска в строке [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/111449/>, свободный – (26.11.2019)
- [4] Поиск подстроки. Алгоритм Кнута–Морриса–Пратта [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/191454/>, свободный – (21.11.2019)
- [5] Алгоритм Бойера-Мура для поиска по шаблону [Электронный ресурс]. – Режим доступа: <http://espressocode.top/boyer-moore-algorithm-for-pattern-searching/>, свободный – (26.11.2019)
- [6] Алгоритм Бойера-Мура для поиска по шаблону [Электронный ресурс]. – Режим доступа: <http://www-igm.univ-mlv.fr/lecroq/string/node14.html>, свободный – (26.11.2019)
- [7] ISO/IEC 14882:2017 [Электронный ресурс]. – Режим доступа: <https://www.iso.org/standard/68564.html>, свободный – (27.11.2019)
- [8] <chrono> [Электронный ресурс]. – Режим доступа: <http://www.cplusplus.com/reference/chrono/>, свободный – (20.11.2019)