МГТУ им. Баумана

Лабораторная работа №7

По курсу: "Анализ алгоритмов"

Поиск подстроки в строке

Работу выполнила: Овчинникова Анастасия, ИУ7-55Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Оглавление

Введение

Целью данной лабораторной работы является изучение алгоритмов поиска подстроки в строке, в частности, алгоритма Кнута-Морриса-Пратта и алгоритма Бойера-Мура.

Задачи лабораторной работы:

1. реализовать алгоритм Кнута-Морриса-Пратта;
2. реализовать алгоритм Бойера-Мура.

Аналитическая часть

Поиск подстроки в строке — одна из простейших задач поиска информации. Поиск подстроки в строке применяется в виде встроенной функции в текстовых редакторах, СУБД, поисковых машинах, языках программирования и т. п.

Пусть дана некоторая строка T (текст) и подстрока W (шаблон). Задача поиска подстроки сводится к поиску вхождения шаблона W в указанной строке T. Строго задача формулируется следующим образом: пусть задан массив T из N элементов и массив W из M элементов, $0 < M \leqslant N$. Если алгоритм поиска подстроки обнаруживает вхождение W в T, то возвращается индекс, указывающий на первое совпадение подстроки со строкой.

Простой алгоритм

Стандартный алгоритм начинает со сравнения первого символа текста с первым символом подстроки. Если они совпадают, то происходит переход ко второму символу текста и подстроки. При совпадении сравниваются следующие символы. Так продолжается до тех пор, пока не окажется, что подстрока целиком совпала с отрезком текста, или пока не встретятся несовпадающие символы. Впервом случае задача решена, во втором мы сдвигаем указатель текущего положения в тексте на один символ и заново начинаем сравнение с подстрокой.

Проблема стандартного алгоритма заключается в том, что он затрачивает много усилий впустую. Если сравнение начала подстроки уже произведено, то полученную информацию можно использовать для того, чтобы определить начало следующего сравниваемого отрезка текста.

Алгоритм Кнута-Морриса-Пратта

Идея алгоритма в том, что при каждом несовпадении T[I] и W[J] мы сдвигаемся не на единицу, а на J, так как меньшие сдвиги не приведут к полному совпадению. К сожалению, этот алгоритм поиска дает выигрыш только тогда, когда несовпадению предшествовало некоторое число совпадений, иначе алгоритм работает как примитивный. Так как совпадения встречаются реже, чем несовпадения, выигрыш в большинстве случаев незначителен.

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 успешно-му сравнению, другой — несовпадению. Успешное сравнение переводит нас в следующий узел автомата, а в случае несовпадения мы попадаемв предыдущий узел, отвечающий образцу.

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.

Заметим, что при совпадении ничего особенного делать не надо: происходит переход к следующему узлу. Напротив, переходы по несовпадению определяются тем, как искомая подстрока соотносится сама с собой.

Метод КМП использует предобработку искомой строки, а именно: на ее основе создается префикс-функция. Префикс-функция от строки S и позиции i в ней — длина k наибольшего собственного (не равного всей подстроке) префикса подстроки $S[1..i]$, который одновременно является суффиксом этой подстроки. То есть, в начале подстроки $S[1..i]$ длины i нужно найти такой префикс максимальной длины $k < i$, который был бы суффиксом данной подстроки $S[1..k] = S[(i-k+1)..i]$.

Например, для строки "abcdabscabcdabia"префикс-функция будет такой:
[0 ,0 ,0 ,0 ,1 ,2 ,0 ,0 ,1 ,2 ,3 ,4 ,5 ,6 ,0 ,1].

Значения префикс-фукнции для каждого символа шаблона вычисляются перед началом поиска подстроки в строке и затем используются для сдвига.

Алгоритм Бойера-Мура

Алгоритм поиска строки Бойера — Мура считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке.

Преимущество этого алгоритма в том, что ценой некоторого количества предварительных вычислений над шаблоном (но не над строкой, в которой ведётся поиск) шаблон сравнивается с исходным текстом не во всех позициях — часть проверок пропускаются как заведомо не дающие результата.

Идея БМ-поиска – сравнение символов начинается с конца образца, а не с начала, то есть сравнение отдельных символов происходит справа налево. Затем с помощью некоторой эвристической процедуры вычисляется величина сдвига вправо s. И снова производится сравнение символов, начиная с конца образца.

Простейший вариант алгоритма Бойера-Мура состоит из следующих шагов. На первом шаге мы строим таблицу смещений для искомого образца. Процесс построения таблицы будет описан ниже. Далее мы совмещаем начало строки и образца и начинаем проверку с последнего символа образца. Если последний символ образца и соответствующий ему при наложении символ строки не совпадают, образец сдвигается относительно строки на величину, полученную из таблицы смещений, и снова проводится сравнение, начиная с последнего символа

образца. Если же символы совпадают, производится сравнение предпоследнего символа образца и т. д. Если все символы образца совпали с наложенными символами строки, значит мы нашли подстроку и поиск окончен. Если же какой-то (не последний) символ образца не совпадает с соответствующим символом строки, мы сдвигаем образец на один символ вправо и снова начинаем проверку с последнего символа. Весь алгоритм выполняется до тех пор, пока либо не будет найдено вхождение искомого образца, либо не будет достигнут конец строки.

Таблица смещений строится следующим образом. Каждому символу ставится в соответствие величина, равная разности длины шаблона и порядкового номера символа (если символ повторяется, то берется самое правое вхождение).

Величина смещения для каждого символа образца зависит только от порядка символов в образце, поэтому смещения удобно вычислить заранее и хранить в виде одномерного массива, где каждому символу алфавита соответствует смещение относительно последнего символа образца.

Конструкторская часть

В данном разделе будут описаны принципы работы выбранных решений и их блоксхемы.

Требования к программе

Требования к вводу:

Программе на вход подается две строки: текст и шаблон. Алгоритм Бойера-Мура работает только с 128 символами ASCII-таблицы.

Требования к программе:
Программа должна находить первое вхождение шаблона в текст и его индекс (индексация строк начинается с нуля). Если вхождение шаблона не найдено, индекс считается равным -1.

Схемы алгоритмов

На рисунках 1-2 представлена схема алгоритма Бойера-Мура. На рисунке 3 представлена схема функции slides(), которая вычисляет сдвиги для алгоритма Бойера-Мура. На рисунке 4 представлена схема алгоритма Кнута-Морриса-Пратта. На рисунке 5 представлена схема функции preffix(), которая используется в алгоритме КМП.

Следует подчеркнуть, что под длиной строки подразумевается количество символов в строке.

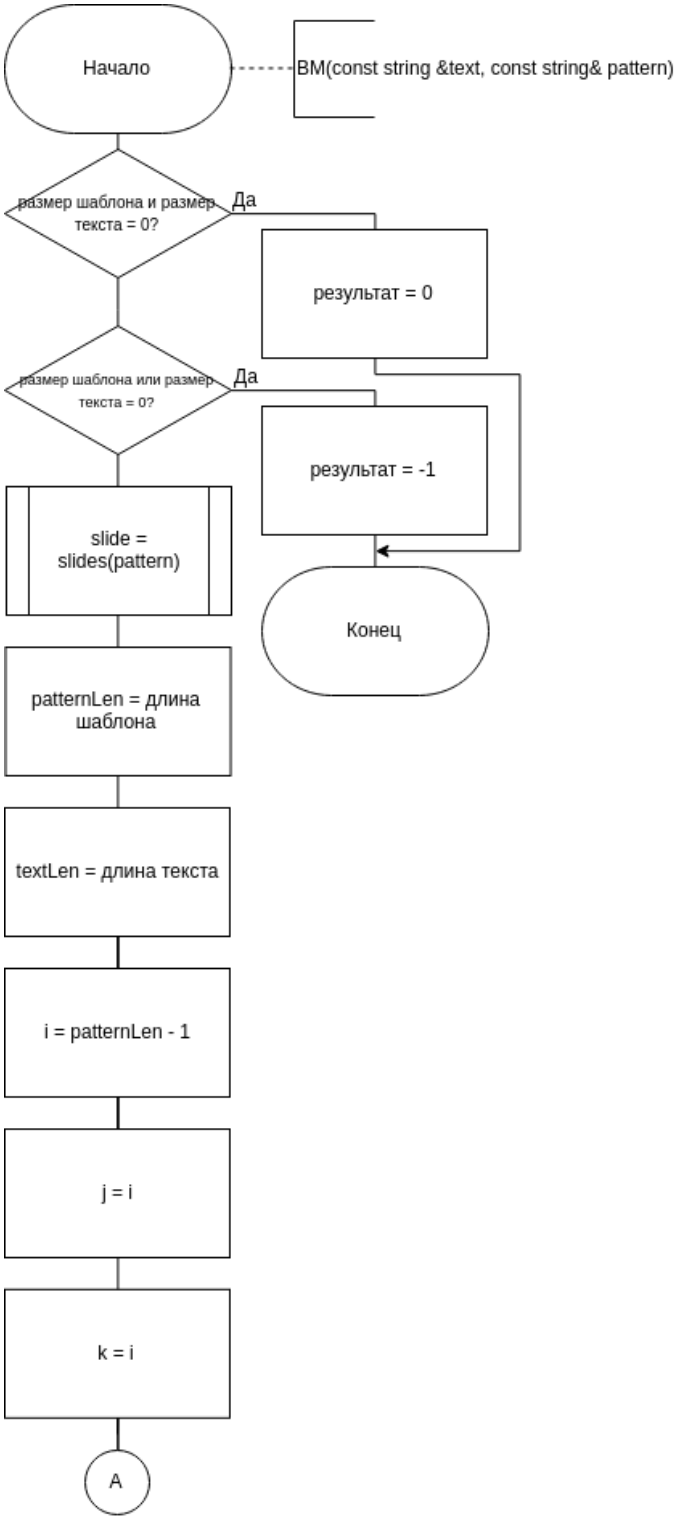

Рис. 1: Схема алгоритма Бойера-Мура

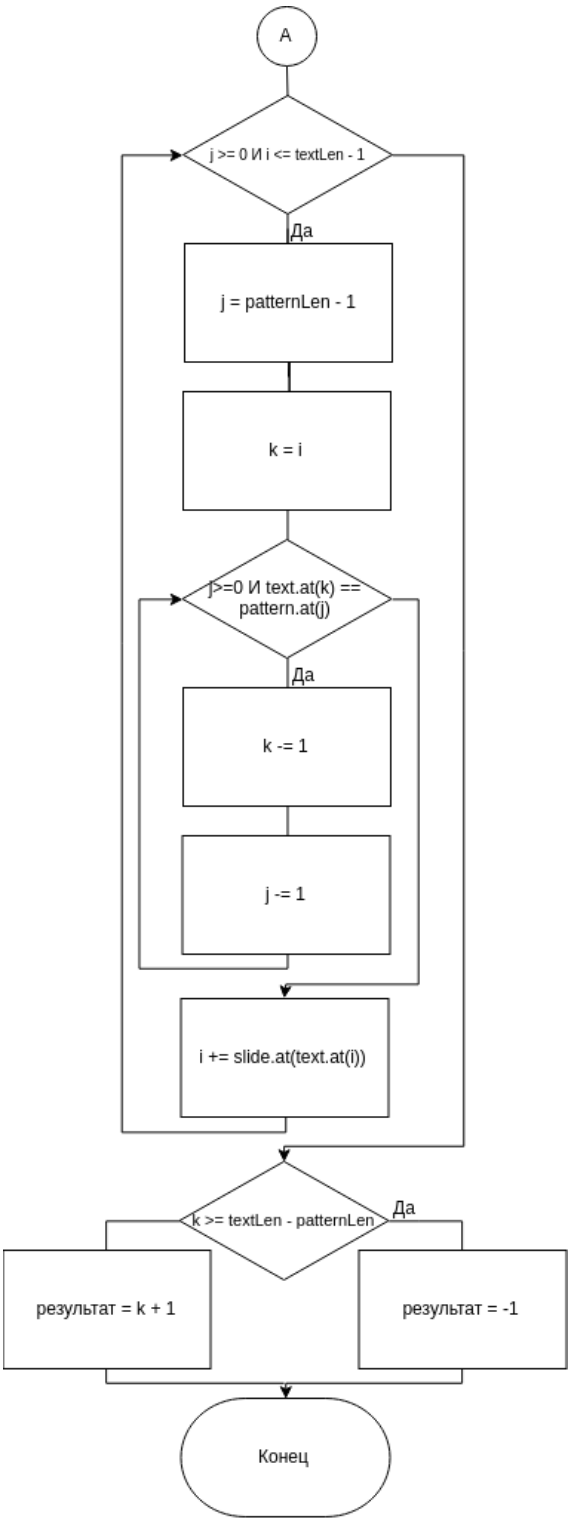

Рис. 2: Схема алгоритма Бойера-Мура (продолжение)

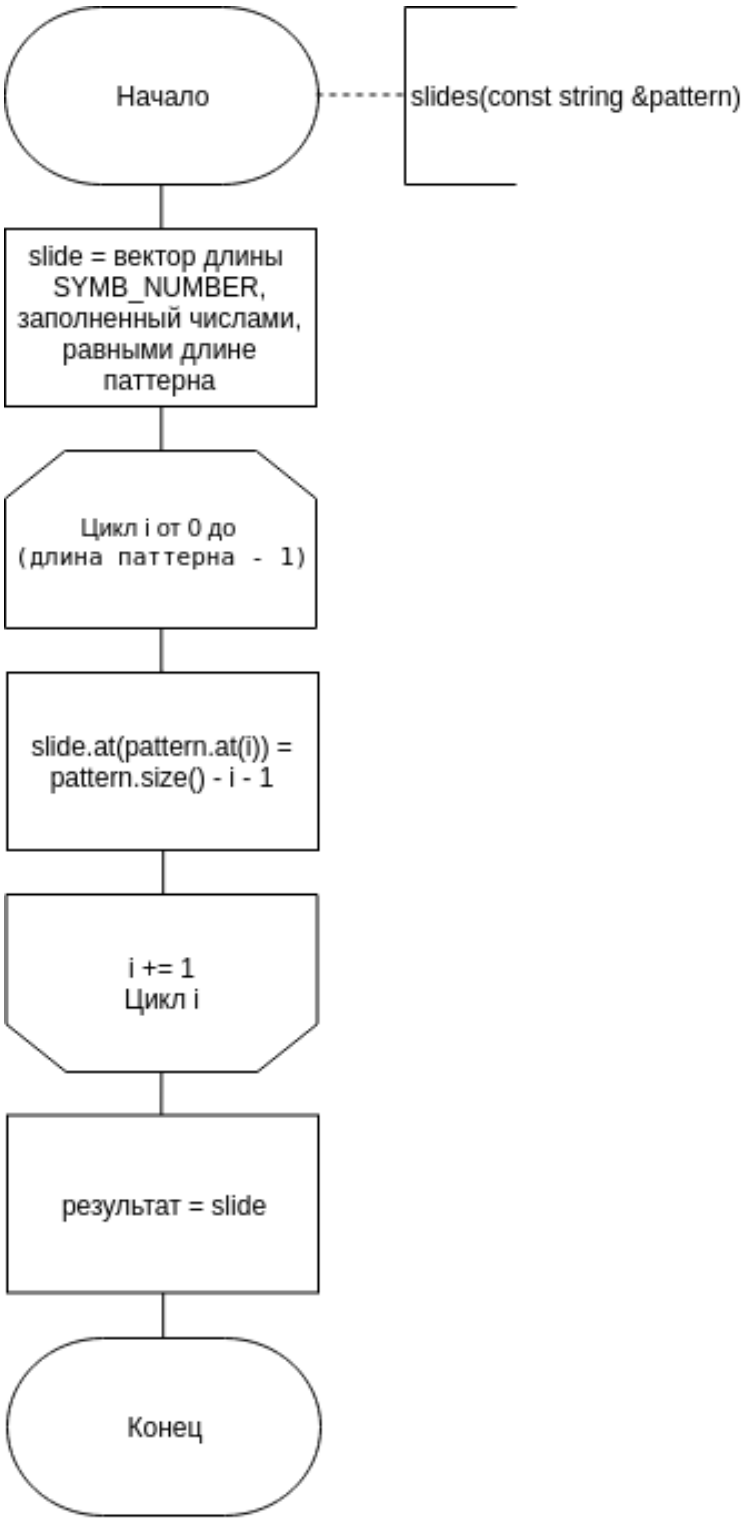

Рис. 3: Схема метода slides()

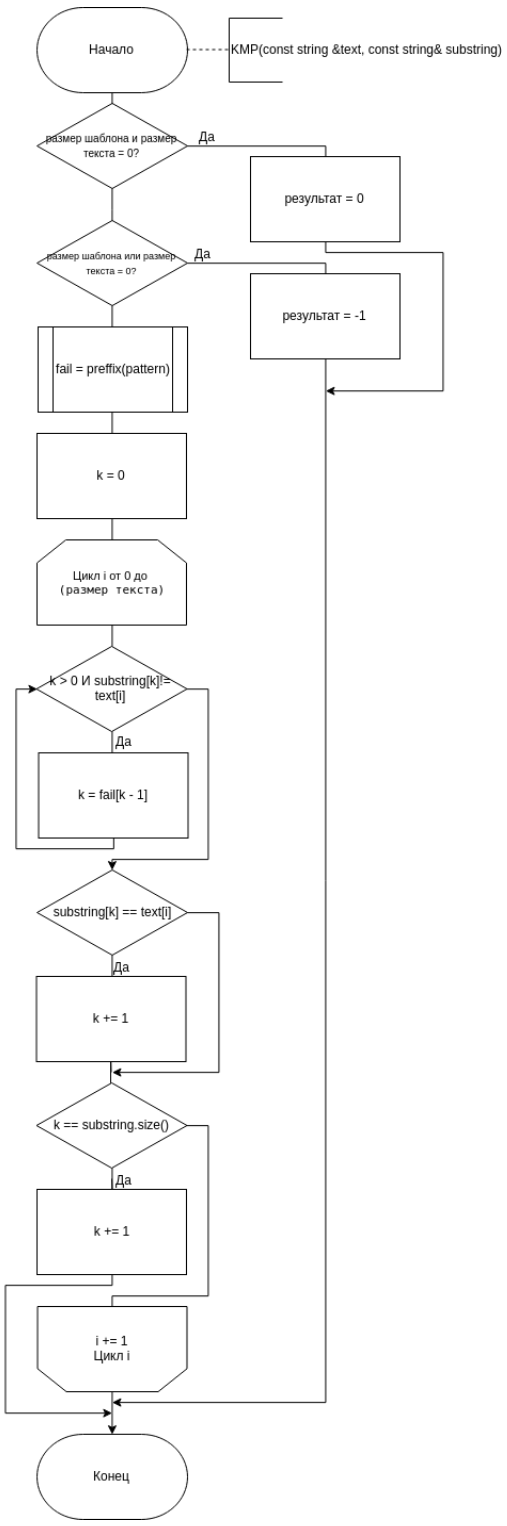

Рис. 4: Схема алгоритма Кнута-Морриса-Пратта

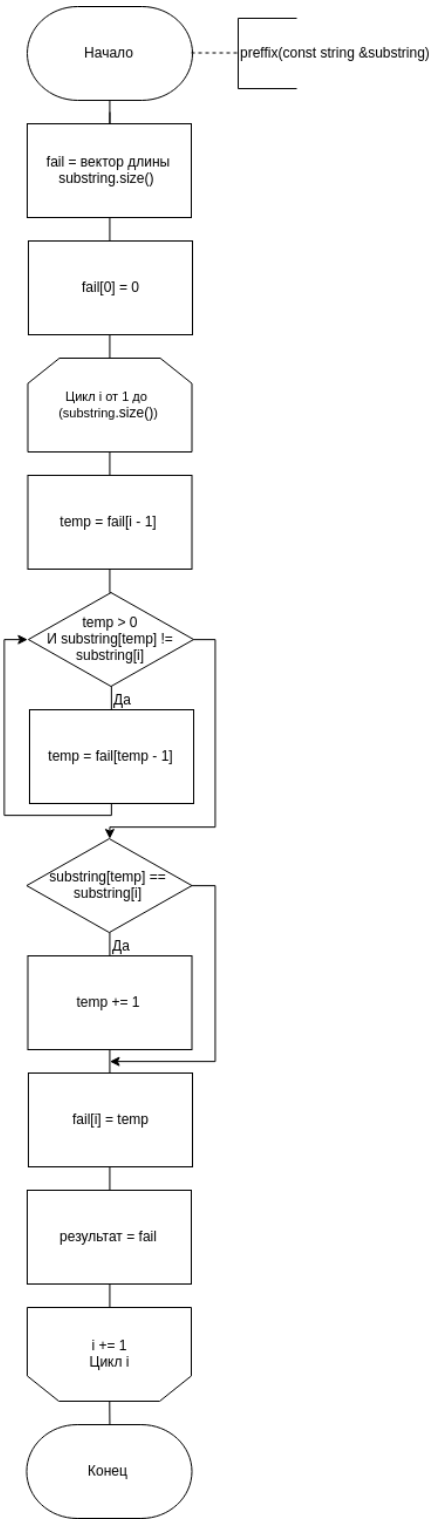

Рис. 5: Схема функции preffix()

Технологическая часть

В данном разделе будут определены средства реализации и приведен листинг кода.

Выбор языка программирования

В качестве языка программирования для реализации программы был выбран язык C++ и фреймворк Qt, потому что:

- язык C++ имеет высокую вычислительную производительность;
- язык C++ поддерживает различные стили программирования;
- в Qt существует удобный инструмент для тестирования - QtTest - который позволяет собирать тесты в группы, собирать результаты выполнения тестов, а также уменьшить дублирование кода при схожих объектах тестирования.

Сведения о модулях программы

Программа состоит из следующих файлов:

- main.cpp - главный файл программы;
- search.h, search.cpp - файл и заголовочный файл, в котором расположена реализация алгоритма КМП и алгоритма БМ;
- testsearch.h, testsearch.cpp - файл и заголовочный файл, в котором расположены тесты.

Листинги кода алгоритмов

В листинге 1 приведена реализация алгоритма Кнута-Морриса-Пратта. В листинге 2 приведена реализация алгоритма Бойера-Мура. В листинге 3 приведена реализация функции slides(), которая используется в алгоритме Бойера-Мура для вычисления сдвигов. В алгоритме Бойера-Мура используется константа, которая равна количеству символов в Ascii-таблице (SYMB_NUMBER = 128).

Листинг 1: Агоритм Кнута-Морриса-Пратта

```
int KMP(const string &text, const string& substring)
{
    if (text.size() == 0 && substring.size() == 0)
        return 0;
    else if(text.size() == 0 || substring.size() == 0)
        return -1;

    // prefix-function
    vector<size_t> fail(substring.size());
    fail[0] = 0;
    for (unsigned int i = 1; i < substring.size(); ++i)
    {
        size_t temp = fail[i - 1];
        while ((temp > 0) && (substring[temp] != substring[i]))
        {
            temp = fail[temp - 1];
        }
        if (substring[temp] == substring[i])
        {
            ++temp;
        }
        fail[i] = temp;
    }

    // Search
    for (unsigned long k = 0, i = 0; i < text.size(); ++i)
    {
        while ((k  > 0) && (substring[k] != text[i]))
        {
            k = fail[k - 1];
        }
        if (substring[k] == text[i])
        {
            ++k;
        }
        if (k == substring.size())
        {
            return static_cast<int>(i - substring.size() + 1);
        }
    }
    return -1;
}
```

Листинг 2: Агоритм Бойера-Мура

```
int BM(const string &text, const string& pattern)
{
    if (text.size() == 0 && pattern.size() == 0)
        return 0;
    else if(text.size() == 0 || pattern.size() == 0)
        return -1;

    vector<size_t> slide = slides(pattern);
```

```
9
10      int patternLen = static_cast<int>(pattern.size());
11      int textLen = static_cast<int>(text.size());
12
13      int i = patternLen - 1;
14      int j = i;
15      int k = i;
16      while (j >= 0 && i <= textLen - 1)
17      {
18          j = patternLen - 1;
19          k = i;
20          while (j >= 0 && text.at(k) == pattern.at(j))
21          {
22              k--;
23              j--;
24          }
25          i += slide.at(text.at(i));
26      }
27      if (k >= textLen - patternLen)
28      {
29          return -1;
30      }
31      else
32      {
33          return k + 1;
34      }
35  }
```

Листинг 3: Функция slides()

```
1  vector<size_t> slides(const string &pattern)
2  {
3      vector<size_t> slide(SYMB_NUMBER, pattern.size());
4      for (unsigned int i = 0; i < pattern.size() - 1; ++i)
5      {
6          slide.at(pattern.at(i)) = pattern.size() - i - 1;
7      }
8      return slide;
9  }
```

Тесты

Тестирование проводилось с помощью модуля QtTest. Для этого был написан класс TestSearch. Каждый алгоритм тестировался на заранее заготовленном наборе тестовых данных. Использованный набор тестовых данных приведен в таблице 1.

Все написанные тесты были пройдены.

Таблица 1: Набор тестовых данных

Текст	Шаблон	Ожидаемый индекс
“there they are “	“they “	6
“there they are “	“there “	0
“there they are “	“are “	11
“there they are “	“hh “	-1
“there they are “	“there they are “	0
““	“dsd “	-1
““	““	0
“abc “	““	-1
“there they are “	“tt “	-1
“there they are “	“ee “	-1
“there they are “	“aa “	-1

Примеры работы алгоритмов

Рассмотрим на конкретных примерах работу алгоритма Кнута-Морриса-Пратта и алгоритма Бойера-Мура.

Алгоритм Кнута-Морриса-Пратта

Пусть у нас есть алфавит из пяти символов: a, b, c, d, e и мы хотим найти вхождение образца “abbad” в строке “abeccacbadbabbad”.

Таблица преффиксов будет выглядеть так.

a	b	b	a	d
0	0	0	1	0

Начало поиска.

a	b	e	c	c	a	c	b	a	d	b	a	b	b	a	d
a	b	b	a	d											
		a	b	b	a	d									
			a	b	b	a	d								
				a	b	b	a	d							
					a	b	b	a	d						
						a	b	b	a	d					
							a	b	b	a	d				
								a	b	b	a	d			
									a	b	b	a	d		
										a	b	b	a	d	
											a	b	b	a	d

Совпадение найдено.

a	b	e	c	c	a	c	b	a	d	b	a	b	b	a	d
		a	b	b	a	d									

Конечный автомат представлен на рисунке 6.

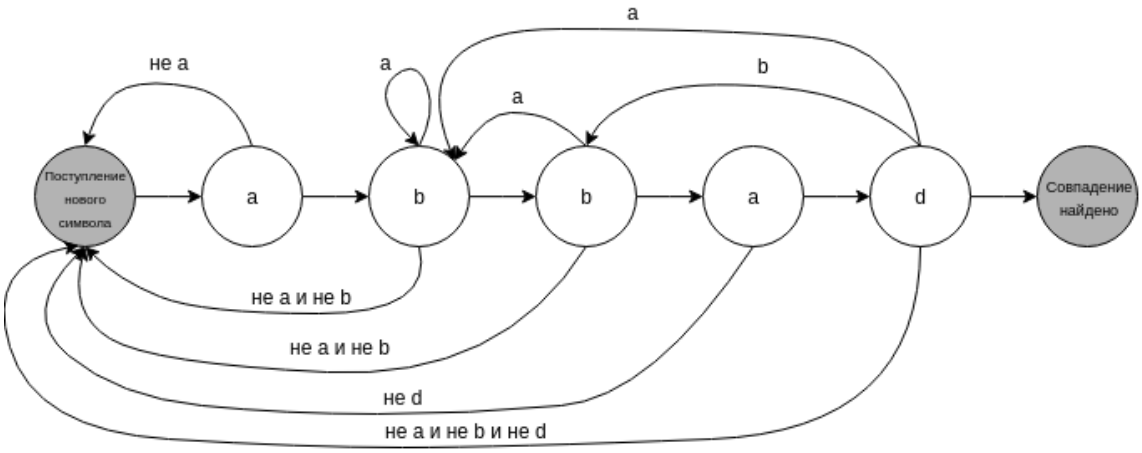

Рис. 6: Конечный автомат в алгоритме КМП

Алгоритм Бойера-Мура

Пусть у нас есть алфавит из пяти символов: a, b, c, d, e и мы хотим найти вхождение образца “abbad” в строке “abeccacbadbabbad”.

Таблица смещений будет выглядеть так.

a	b	c	d	e
1	2	5	5	5

Начало поиска.

a	b	e	c	c	a	c	b	a	d	b	a	b	b	a	d
a	b	b	a	d											

Последний символ образца не совпадает с наложенным символом строки. Сдвигаем образец вправо на 5 позиций.

a	b	e	c	c	a	c	b	a	d	b	a	b	b	a	d
					a	b	b	a	d						

Второй символ не совпадает, сдвигаем на 5 символов вправо.

a	b	e	c	c	a	c	b	a	d	b	a	b	b	a	d
										a	b	b	a	d	

Последний символ не совпадает, сдвигаем на один символ вправо.

a	b	e	c	c	a	c	b	a	d	b	a	b	b	a	d
											a	b	b	a	d

Совпадение найдено.
Конечный автомат представлен на рисунке 7.

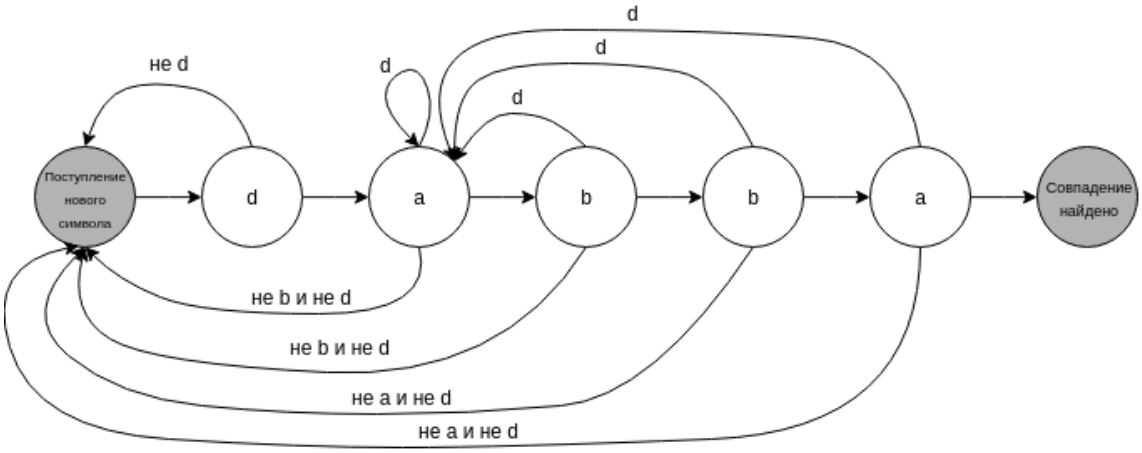

Рис. 7: Конечный автомат в алгоритме КМП

Заключение

Таким образом, в ходе данной лабораторной работы были изучены и реализованы алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура. Кроме того, была рассмотрена работа этих алгоритмов на конкретных примерах.

Список литературы

1. Дж. Макконнелл. Основы современных алгоритмов. 2-е дополненное издание. Москва: Техносфера, 2004. - 368с.