Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

И. А. Зверев, А. А. Шалыто

Построение визуализатора алгоритма поиска подстроки Кнута-Морриса-Пратта на базе технологии *Vizi*

Программирование с явным выделением состояний
Проектная документация

Проект создан в рамках
«Движения за открытую проектную документацию»
http://is.ifmo.ru/

Санкт-Петербург
2004

Содержание

Введение ... 3
1. Описание алгоритма ... 3
1.1. Вспомогательные утверждения ... 3
1.2. Алгоритм ... 3
2. Реализация визуализируемого алгоритма ... 4
3. Описание модели данных ... 4
4. Преобразование программы ... 5
4.1. Преобразование реализации алгоритма ... 5
4.2. Построение XML-файлов с описанием визуализатора ... 5
5. Описание интерфейса визуализатора ... 7
6. Описание конфигурации визуализатора ... 8
7. Предложения по развитию технологии *Vizi* ... 9
Заключение ... 10
Литература ... 11
Приложение 1. Исходный текст реализации алгоритма Кнута-Морриса-Пратта ... 12
Приложение 2. Исходный текст преобразованной реализации алгоритма ... 13
Приложение 3. XML-описания визуализатора ... 14
Файл SearchKMP.xml ... 14
Файл SearchKMP-Configuration.xml ... 14
Файл SearchKMP-Algorithm.xml ... 17
Приложение 4. Автоматически сгенерированный код ... 21
Приложение 5. Исходный код визуализатора ... 30

Введение

На кафедре «Компьютерные технологии» СПбГУ ИТМО для разработки и реализации визуализаторов алгоритмов на основе конечных автоматов была предложена технология *Vizi* [1, 2].

Визуализатор — это программа, в процессе работы которой на экране компьютера динамически демонстрируется применение алгоритма к выбранному набору данных, что позволяет наглядно представить суть алгоритма.

В данной работе логика и реализация визуализатора алгоритма Кнута-Морриса-Пратта [3] строятся на базе технологии *Vizi*.

1. Описание алгоритма

1.1. Вспомогательные утверждения

Алгоритм поиска подстроки базируется на специальной функции, которую обозначим n(x). Примененная на слове x, она возвращает самое длинное начало этого слова, одновременно являющееся его концом. Например, для слова «abbaobabba» значение функции n(x) будет равно «abba», значение функции n(n(x)) — «a». И наконец, значение n(n(n(x))) — пустая строка.

Очевидно, что все слова n(x), n(n(x)), n(n(n(x))) являются началами слова x, так как каждое из них является началом предыдущего. Также они являются концами слова x. Указанная последовательность обрывается, заканчиваясь пустой строкой, поскольку каждое следующее слово строго короче предыдущего. Можно также доказать, что любая последовательность, являющаяся одновременно началом и концов слова x, равна $n^k(x)$ для некоего k.

1.2. Алгоритм

Алгоритм Кнута-Морриса-Пратта (КМП) получает на вход слово X = x[1]x[2] … x[n] и просматривает его буква за буквой, заполняя при этом массив натуральных чисел L = l[1]…l[n] так, что l[i] = длина слова n(x[1] … x[i]). Другими словами, значение l[i] — длина наибольшего начала слова x[1] … x[i], одновременно являющегося его концом. Заметим, что если составить X как A#B (A – искомая подстрока, B – строка, в которой производится поиск, # - символ, не встречающийся ни в одной из этих строк), то после применения алгоритма КМП элементы массива L, равные длине слова A, будут отмечать последние позиции входа подстроки в строку B. Если же такие элементы отсутствуют, то A в B не входит.

Массив L заполняется динамически. Первый элемент всегда равен нулю, так как строка единичной длины может иметь только пустые подстроки. Пусть заполнены элементы l[1]...l[i]. Мы читаем очередной символ x[i+1] и должны вычислить l[i+1]. Другими словами, требуется определить начала слова x[1] ... x[i+1], одновременно являющиеся его концами. Откуда берутся эти начала? Каждое из них получается из некоего начала слова x[1] ... x[i] путем приписывания символа x[i+1]. Логично, что не любое слово, являющееся началом и концом слова x[1]...x[i], годится — требуется, чтобы за ним следовала буква x[i+1]. Таким образом, суть алгоритма в следующем: берем все начала слова x[1]...x[i] в порядке убывания длины и пробуем дописать к ним символ x[i+1]. Если полученное слово является началом слова x[1] ... x[i+1], то оно подходит, и его длина заносится в ячейку l[i+1]. Остается отметить, что все длины начал слов x[1]...x[i] можно получить, последовательно вычисляя l[i], l[l[i]] и т.д., пока очередной элемент не станет равным нулю.

Сложность алгоритма не превышает C(n+m), где n — длина строки, m — длина подстроки.

2. Реализация визуализируемого алгоритма

Программа визуализации реализует описанный выше алгоритм. Ее исходный текст приведен в приложении 1.

3. Описание модели данных

Модель данных формировалась только с учетом используемых в вышеприведенном алгоритме переменных. Текущее начало слова из цикла поиска подходящего начала представляется в модели своей длиной, так как само слово можно найти как первые *len* символов слова X. Таким образом, модель данных содержит следующие переменные:

- слово *X*, представляющее собой объединение подстроки и строки с разделительным символом;
- массив *L*;
- длина обработанной части слова *i*;
- длина текущего проверяемого начала слова x[1] ... x[i] *len*;
- ссылка на экземпляр визуализатора *visualizer*.

4. Преобразование программы

4.1. Преобразование реализации алгоритма

Поскольку алгоритм не задействует сложных языковых конструкций (лишь два цикла while и одно условие), преобразование алгоритма к виду, пригодному для формирования автомата, свелось к введению модели данных (приложение 2). В случае использования более сложных операторов требуется свести их к следующим семи конструкциям:

- оператор присваивания;
- последовательность операторов (составной оператор);
- укороченный вариант ветвления (`if-then`);
- полный оператор ветвления (`if-then-else`);
- цикл с предусловием (`while`);
- вызов процедуры.

4.2. Построение XML-файлов с описанием визуализатора

Основным файлом, содержащим информацию о визуализаторе, является *SearchKMP.xml*. В нем содержится информация о классах, размерах окна визуализатора и текстовая информация: название, разработчик, руководитель проекта, лицензия. Кроме того, в конце файла включаются два других, содержащих описание конфигурации визуализатора и описание модели данных и алгоритма. Все три *XML*-файла на текущем этапе развития технологии *Vizi* подготавливаются вручную. Их исходные тексты приведены в приложении 3.

Файл описания конфигурации *SearchKMP-Configuration.xml* состоит из последовательности свойств, которые можно разделить на параметры (тэг `property`), сообщения (`message`), стили (`styleset`, `style`, `font`, `color`) и свойства компонентов (`panel`, `button` и т.п.). Каждое свойство адресуется с помощью идентификатора, указываемого в атрибуте `param`, и имеет описание, задаваемое в атрибуте `description`. Кроме того, для отображаемого текста, встречающегося в сообщениях и свойствах компонентов, должны задаваться как русский, так и английский варианты. Использование жестко заданных строк в самом визуализаторе **недопустимо**, так как это делает невозможным его полную локализацию. Для удобства разделения визуализатора на составляющие (в случае визуализации сложного алгоритма) элементы конфигурации можно объединять в группы с помощью тэгов `group`.

Наконец, описание алгоритма содержится в файле *SearchKMP-Algorithm.xml*. Вероятно, в будущем структура данного файла будет генерироваться автоматически с помощью специ-

альной программы, сейчас же, по сути, необходимо вручную превратить преобразованную реализацию алгоритма в *XML*-вид.

Логически этот файл можно разделить на три части: перечисление импортируемых модулей, описание модели данных, описание автоматов, определение вспомогательных методов. Модель данных описывается в начале файла внутри тэга `data`. В нем может содержаться любое количество переменных модели, каждая из которых задается и инициализируется внутри тэга `variable`, имеющего обязательный атрибут `description` — описание переменной. Все описания появляются только в сгенерированном *java*-файле, поэтому при их написании допустим любой удобный разработчику язык. При необходимости в модель данных можно добавить метод ее преобразования в строку с помощью тэга `toString`.

Каждый автомат описывается внутри тэгов `auto`. Автомат и все состояния внутри него должны иметь уникальные идентификаторы, по которым сценарий адресует комментарии к каждому шагу визуализации. Очевидно, что каждому шагу визуализации соответствует одно состояние автомата, но не наоборот: могут встречаться состояния, которые недостаточно важны для визуализации. Важность состояния определяется атрибутом `level` в тэгах соответствующих операторов. Если значение атрибута меньше нуля, то визуализация состояния не происходит; если больше нуля — визуализатор всегда останавливается на состоянии. Если значение этого атрибута нулевое, то визуализатор останавливается на данном состоянии только в режиме пошагового исполнения алгоритма.

Содержание описания автомата таково: сначала идет состояние «начало» (тэг `start`), затем — последовательность операторов (тэги `step`, `while`, `if` и `call-auto`), и в конце — состояние «конец» (`finish`). Все эти состояния, кроме вызова вложенного автомата (`call-auto`, соответствует вложенным функциям в программе) имеют атрибуты комментария на русском и английском языке (`comment-ru` и `comment-en`). Стоит отметить, что комментарий может содержать произвольные параметры, задающиеся как список объектов (атомарные типы недопустимы) в атрибуте `comment-args`, а в строках комментариев заменяющие последовательности вида «{i}», где i — номер аргумента, начиная с нулевого. В операторах с условием (`while` и `if`) следует задавать две пары комментариев: для случая, когда условие (задаваемое атрибутом `test`) выполнено и когда — нет. Параметры для обоих случаев используются одни и те же.

Операторы `then` и `else`, встречающиеся внутри `if`, и оператор `while` содержат другие операторы, оператор `call-auto` через атрибут `id` ссылается на другой автомат, а все действия происходят внутри тэга `step`, представляющего операторы присваивания и составные операторы. Он может содержать четыре типа фрагментов кода: обновление отобра-

женного состояния визуализатора (в тэге draw), действия при прямом проходе алгоритма в данном операторе (direct), действия при обратном проходе (при нажатии кнопки «предыдущий шаг», тэг reverse) и автоматически обращаемый фрагмент кода (тэг action, более подробно описан в [1]).

При составлении описания реализации алгоритма Кнута-Морриса-Пратта возникло единственное затруднение — во время выполнения внутреннего цикла уменьшение длины не обратимо без использования стека, но его введение излишне усложнило бы реализацию. Поэтому возвращение происходит в начало цикла: в масштабах визуализации это не слишком большой шаг. Сгенерированный на основе описания *java*-файл приведен в приложении 4. Алгоритм генерации подробно описан в работе [1].

Обратим внимание, что автоматически построено два автомата визуализации: один для прямого прохода, а другой — для обратного. Каждый из автоматов содержит по 13 состояний.

5. Описание интерфейса визуализатора

Типичный вид окна визуализатора приведен на рис. 1. Интерфейс состоит из трех частей: область визуализации, комментарий к текущему шагу алгоритма и панель управления. Прежде всего, рассмотрим область визуализации и то, как отражается в ней текущее состояние алгоритма.

В упомянутой области отображено два массива: первый — обрабатываемая строка X, второй — массив L. В целях наглядности из второго массива отображаются только заполненные элементы l[1] ... l[i], а значение переменной i можно определить, во-первых, по коли-

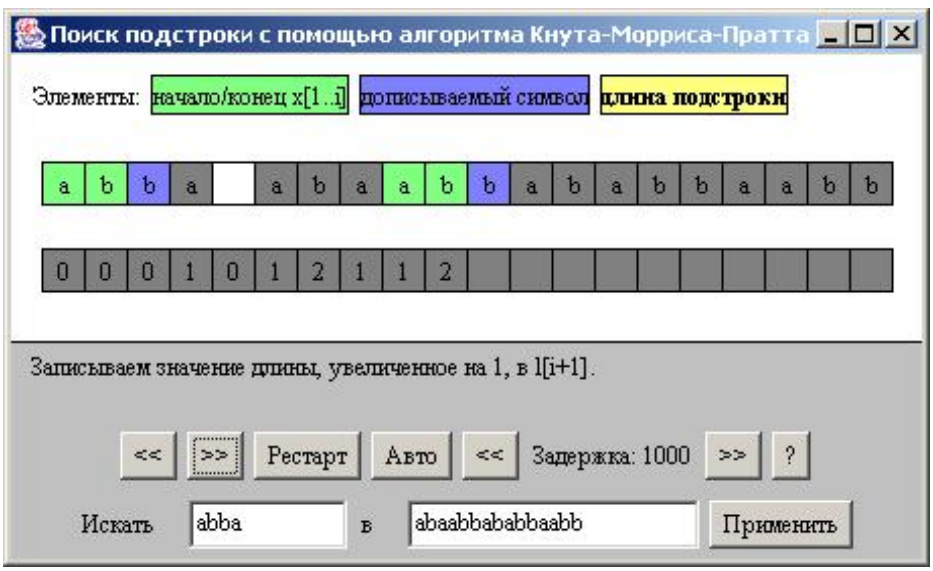

Рис. 1. Внешний вид визуализатора

честву отображаемых значений массива L, и во-вторых, по позиции крайнего правого выделенного синим элемента x[i+1]. Проверяемые начало и конец последовательности x[1] ... x[i] выделяются зеленым. Часто встречается ситуация, когда длина начала нулевая, поэтому ни одна клетка не раскрашена в зеленый цвет.

Когда в процессе работы алгоритма обнаруживается полное вхождение подстроки в строку, иными словами, значение некоего элемента l[i] равно длине подстроки, соответствующая ячейка на экране раскрашивается в желтый цвет. После окончания работы алгоритма из массива L оставляются только такие ячейки. Вполне вероятна ситуация, когда такие ячейки отсутствуют.

Кратко перечислим назначение кнопок на панели управления процессом визуализации:

- «>>»: перейти к следующему шагу визуализации;
- «<<»: вернуться к предыдущему шагу визуализации;
- «Рестарт»: вернуться к начальному состоянию;
- «Авто»: включить или выключить режим автоматической работы визуализатора;
- «Задержка»: с помощью кнопок слева и справа от этой надписи можно регулировать величину задержки между шагами визуализации во время автоматической работы.
- «?»: посмотреть информацию о разработчиках визуализатора и авторских правах;
- «Применить»: использовать введенные слева от кнопки подстроку и строку для демонстрации работы на них алгоритма. Нажатие кнопки возвращает визуализирующий автомат в начальное состояние. Заметим, что кнопка работает только когда длины обеих строк лежат в допустимых пределах (задаваемых в конфигурации).

6. Описание конфигурации визуализатора

Как уже было упомянуто, конфигурация визуализатора хранится в файле *SearchKMP-Configuration.xml*. Одним из важных атрибутов всех тэгов данного XML-файла является атрибут `description`, хранящий описание элемента. Таким образом, большинство настраиваемых параметров уже описаны в файле конфигурации. Однако некоторые элементы требуют отдельного пояснения. Так, сообщения «`label-legend`», «`label-seq`», «`label-i1`» и «`label-found`» выводятся в область визуализации как пояснения к обозначениям элементов массивов. При их редактировании стоит учитывать, что все четыре строки должны помещаться в ряд в окне визуализатора, не выходя за его пределы по ширине. Это

связано с тем, что при внедрении визуализатора в страницу HTML его ширину изменить уже невозможно. Стиль же надписи «`label-legend`» определяется пятым по номеру (шестым по счету) в списке стилей «`array`». По сути, это прямоугольник с текстом, но без рамки и заливки.

7. Предложения по развитию технологии *Vizi*

В ходе работы над проектом и его описанием были отмечены некоторые перспективные пути улучшения реализации технологии. Прежде всего, теоретически возможно и на практике сравнительно легко осуществимо создание программы, автоматически преобразующей текст реализации алгоритма в *XML*-вид. После такой обработки пользователю системы потребуется только выбрать важность каждого шага и сформулировать их описания.

Что касается деталей реализации, некоторое неудобство было замечено в том, что символ «©» вставляется в описание визуализатора с помощью его кода в системе *Unicode*. Поскольку он присутствует всегда в строках, содержащий информацию об авторских правах, логично было бы создать специальный символ, его обозначающий (по примеру *HTML* — `©`).

Заметим, что для изменения конфигурации путем редактирования *XML*-файла требуется последующая пересборка проекта, поскольку в готовом пакете визуализатора конфигурация хранится во внутреннем формате. Было бы удобнее сделать загрузку конфигурации в исходном виде, чтобы дать возможность редактировать ее в уже готовом пакете (например, помещая соответствующий отредактированный файл рядом с файлом пакета).

Основная и главная недоработка *Vizi* как проекта — отсутствие проектной документации. Существует работа [1], посвященная рассмотрению *Vizi* в контексте алгоритма преобразования процедурной реализации алгоритма в автоматную, но что касается описания проекта как инструмента для создания визуализаторов (заметим, основное его назначение) — тут наблюдается почти полное отсутствие информации. Прилагаемый пример (визуализатор алгоритма поиска максимума) не в полной мере раскрывает потенциал *Vizi*, а комментарии внутри файлов описания структуры XML (*scripts/*.dtd*) слишком поверхностны и не описывают варианты использования соответствующих структур. Автор хотел бы порекомендовать как пример формат комментариев к библиотеке Apache Tapestry [5]. То же самое касается библиотеки, на основе которой строится класс визуализатора. К примеру, способ вернуть автомат в начальное состояние удалось узнать только благодаря исходным кодам работ сокурсников.

Заключение

Несмотря на некоторые замеченные недостатки, технология *Vizi* существенно улучшила процесс создания визуализаторов как для простых, так и для сложных для визуализации алгоритмов. Большей частью, это достигается благодаря частичной автоматизации процесса построения автоматной логики [6] визуализатора по процедурному алгоритму. Это не только устраняет необходимость вручную эвристически определять важные для визуализации шаги алгоритма, но и позволяет автоматически строить автомат для реализации обратного прохода. Кроме того, обеспечивается возможность легко программировать визуализирующую часть, разделяя логику алгоритма, способ визуализации и модель данных.

При реализации данного проекта была отмечена высокая эффективность и польза от проектной документации [4]. Программная документация, автоматически генерируемая программой *javadoc*, увы, далеко не достаточна для полноценного понимания и использования задокументированного модуля, что можно выяснить на примере текущей (на момент разработки) версии библиотеки *BaseApplet*. Остается выразить надежду, что в будущем система *Vizi* будет документирована полнее.

Литература

1. *Казаков М. А., Корнеев Г. А., Шалыто А. А.* Разработка логики визуализаторов алгоритмов на основе конечных автоматов // Телекоммуникации и информатизация образования. 2003. № 6. http://is.ifmo.ru/, раздел «Статьи».
2. *Корнеев Г. А., Шалыто А. А.* Требования к визаулизаторам алгоритмов, выполняемых на базе технологии *Vizi*. http://is.ifmo.ru/vis/_requirements.pdf
3. *Шень А.* Программирование: теоремы и задачи. М.: МЦНМО, 1995, 264 с. ftp://ftp.mccme.ru/users/shen/progbook/
4. *Шалыто А. А.* Новая инициатива в программировании. Движение за открытую проектную документацию // Мир ПК. 2003. № 9. http://is.ifmo.ru/, раздел «Статьи».
5. http://jakarta.apache.org/tapestry/dtd/Tapestry_3_0.dtd
6. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления СПб.: Наука, 1998, 628 с

Приложение 1. Исходный текст реализации алгоритма Кнута-Морриса-Пратта

```
package orig;

public class CorrectKMP_step1 {
    private static char[] x;
    private static int[] l;

    private static void kmp() {
        int i = 1;
        l[0] = 0;
        while( i < x.length ) {
            int len = l[i-1];
            while( (x[len] != x[i]) && (len > 0) )
                len = l[len-1];
            if(x[len] == x[i])
                l[i] = len + 1;
            else
                l[i] = 0;
            i++;

        }
    }

    public static void main(String[] args) {
        String str = "abaabbaababaabbbabaaa";
        String substr = "abbaa";

        x = (substr+"@"+str).toCharArray();
        l = new int[x.length];
        kmp();

        for( int i = 0; i < l.length; ++i )
            System.out.print(x[i]);
        System.out.println();
        for( int i = 0; i < l.length; ++i )
            System.out.print(l[i]);
        System.out.println();
    }
}
```

Приложение 2. Исходный текст преобразованной реализации алгоритма

```
package orig;

public class CorrectKMP_step2 {
    public final static class Data {
        public char[] x;    // строка [0..n-1]
        public int[] l;     // длины [1..i]
        public int i;       // длина обработанной строки
        public int len;     // длина начала слова [1..i], равной его концу.
    }

    private static Data d = new Data();

    private static void kmp() {
        d.i = 1;
        d.l[0] = 0;
        while( d.i < d.x.length ) {
            d.len = d.l[d.i-1];
            while( (d.x[d.len] != d.x[d.i]) && (d.len > 0) )
                d.len = d.l[d.len-1];
            if(d.x[d.len] == d.x[d.i])
                d.l[d.i] = d.len + 1;
            else
                d.l[d.i] = 0;
            d.i++;
        }
    }

    public static void main(String[] args) {
        String str = "abaabbaababaabbbabaaa";
        String substr = "abbaa";

        d.x = (substr+"@"+str).toCharArray();
        d.l = new int[d.x.length];
        kmp();

        for( int i = 0; i < d.l.length; ++i )
            System.out.print(d.x[i]);
        System.out.println();
        for( int i = 0; i < d.l.length; ++i )
            System.out.print(d.l[i]);
        System.out.println();
    }
}
```

Приложение 3. XML-описания визуализатора

Файл SearchKMP.xml

```
<?xml version="1.0" encoding="WINDOWS-1251"?>

<!--
    "SearchKMP" visualizer description
-->

<!DOCTYPE visualizer PUBLIC
    "-//IFMO Vizi//Visualizer description"
    "http://ips.ifmo.ru/vizi/dtd/visualizer.dtd"
[
    <!ENTITY algorithm SYSTEM "SearchKMP-Algorithm.xml">
    <!ENTITY configuration SYSTEM "SearchKMP-Configuration.xml">
]>

<visualizer
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="../scripts/visualizer.xsd"

    id="SearchKMP"
    package="ru.ifmo.vizi.kmp"
    main-class="SearchKMPVisualizer"

    preferred-width="450"
    preferred-height="250"

    name-ru="Поиск подстроки с помощью\nалгоритма Кнута-Морриса-Пратта"
    name-en="Substring search using\nKnuth-Morris-Pratt algorithm"

    author-ru="Илья Зверев"
    author-en="Ilya Zverev"
    author-email="zverik@rbcmail.ru"

    supervisor-ru="Георгий Корнеев"
    supervisor-en="Georgiy Korneev"
    supervisor-email="kgeorgiy@rain.ifmo.ru"

    copyright-ru="Copyright \u00A9 Кафедра КТ, СПб ГИТМО (ТУ), 2003"
    copyright-en="Copyright \u00A9 Computer Technologies Department, SPb IFMO,
2003"
>
    &algorithm;
    &configuration;
</visualizer>
```

Файл SearchKMP-Configuration.xml

```
<?xml version="1.0" encoding="WINDOWS-1251"?>

<!--
    "SearchKMP" visualizer configuration.
-->

<configuration>
```

```
<property
    description = "Comment pane height"
    param       = "comment-height"
    value       = "40"
/>
<button
    param       = "button-apply"
    description = "Кнопка использования введенных строк для демонстрации"
    caption-ru  = "Применить"
    caption-en  = "Apply"
    hint-ru     = "Использовать строки, введенную в поля"
    hint-en     = "Use entered strings for demonstration"
/>
<styleset
    description = "Array style set"
    param       = "array"
>
    <style
        description     = "0: стиль всех неиспользуемых квадратов"
        text-color      = "000000"
        text-align      = "0.5"
        border-color    = "000000"
        border-status   = "true"
        fill-color      = "808080"
        fill-status     = "true"
        aspect-status   = "false"
        padding         = "0.2"
    >
        <font
            face        = "Serif"
            size        = "12"
            style       = "plain"
        />
    </style>
    <style
        description     = "1: стиль квадрата с разделителем строк"
        fill-status     = "false"
    />
    <style
        description     = "2: стиль квадратов, входящих в подстроку"
        fill-color      = "80ff80"
    />
    <style
        description     = "3: стиль квадрата в позиции i+1"
        fill-color      = "8080ff"
    />
    <style
        description     = "4: стиль квадрата с числом = длине подстроки"
        fill-color      = "ffff80"
    >
        <font
            style       = "bold"
        />
    </style>
    <style
        description     = "5: строка 'legend:'"
        fill-status     = "false"
        border-status   = "false"
    />
</styleset>
<property
    param       = "max-substr-length"
    description = "Максимальная длина подстроки"
    value       = "9"
```

```
    />
    <property
        param       = "max-str-length"
        description = "Максимальная длина строки"
        value       = "20"
    />
    <property
        param       = "default-substring"
        description = "Подстрока по умолчанию"
        value       = "abba"
    />
    <property
        param       = "default-string"
        description = "Строка по умолчанию"
        value       = "abaabbababbaabb"
    />
    <property
        param       = "max-value-string"
        description = "Число наибольшей ширины, которое может встречаться в
ячейке массива длин подстрок."
        value       = "8"
    />
    <message
        param       = "label-search"
        description = "Текст на метке перед полем для ввода подстроки"
        message-ru  = "Искать"
        message-en  = "Search for"
    />
    <message
        param       = "label-in"
        description = "Текст на метке между полями для ввода подстроки и строки"
        message-ru  = "в"
        message-en  = "in"
    />
    <!-- легенда -->
    <message
        param       = "label-legend"
        description = "Метка легенды"
        message-ru  = "Элементы:"
        message-en  = "Legend:"
    />
    <message
        param       = "label-seq"
        description = "Обозначение символа, входящего в выделенные подстроки"
        message-ru  = "начало/конец x[1..i]"
        message-en  = "head/tail of x[1..i]"
    />
    <message
        param       = "label-i1"
        description = "Обозначение символа в позиции i+1"
        message-ru  = "дописываемый символ"
        message-en  = "added character"
    />
    <message
        param       = "label-found"
        description = "Обозначение элемента массива длин с длиной подстроки"
        message-ru  = "длина подстроки"
        message-en  = "substring length"
    />

</configuration>
```

Файл SearchKMP-Algorithm.xml

```
<?xml version="1.0" encoding="WINDOWS-1251"?>

<!--
    "SearchKMP" algoritm description
-->

<algorithm>
    <data>
        <variable description="Соединенные подстрока и строка">char x[] =
"abbaa@abaabbaabababaabbbabaaa".toCharArray();</variable>
        <variable description="Массив длин совпадений">int[] l = new
int[x.length];</variable>
        <variable description="Максимальная длина совпадения">int
len;</variable>
        <variable description="Длина обработанной строки">int i;</variable>
        <variable description="Экземпляр апплета">SearchKMPVisualizer visual-
izer;</variable>
        <toString>
            StringBuffer s = new StringBuffer();

            s.append("l = ");
            for( int li = 0; li &lt; l.length; ++ li ) {
                s.append(l[li]);
            }
            s.append("\n");

            s.append("i = ");
            s.append(i);
            s.append("\n");

            s.append("len = ");
            s.append(len);
            s.append("\n");


            return s.toString();
        </toString>
    </data>

    <auto id="Main" description="Ищет подстроку в строке">
        <start
            comment-ru="На экране изображена строка, в которой будет произво-
диться поиск, и искомая подстрока."
            comment-en="On the screen there is a string displayed and the sub-
string that needs to be found."
        >
            <draw>
                d.visualizer.updateState();
            </draw>
        </start>
        <step
            id="Initialization"
            description="Инициализация"
            comment-ru="l[1] = 0 по определению. Инвариант: таблица l[1]..l[i]
заполнена правильно. Выделен следующий элемент (i+1)."
            comment-en="l[1] = 0 by definition. The invariant is that table
l[1]..l[i] is filled correctly. Selected element has index i+1."
        >
            <draw>
                d.visualizer.updateState();
            </draw>
```

```
            <direct>
                d.l[0] = 0;
                d.i = 1;
                d.len = -1;
            </direct>
            <reverse>
                d.i = 0;
            </reverse>
        </step>
        <step
            id="LoopInit"
            description="Начало цикла"
            level="-1"
        >
            <direct>
                d.i = 1;
            </direct>
        </step>
        <while
            id="Loop"
            description="Цикл"
            test="d.i &lt; d.x.length"
            level="-1"
        >
            <step
                id="InitLen"
                description="Инициализация len"
                comment-ru="Выделенные начало и конец строки x[1]..x[i] совпадают или имеют нулевую длину. Пробуем приписать справа к подстроке текущий символ."
                comment-en="Selected head and tail of string x[1]..x[i] are equal or have zero length. Now trying to add the next character."
            >
                <draw>
                    d.visualizer.updateState();
                </draw>
                <direct>
                    d.len = d.l[d.i-1];
                </direct>
                <reverse>
                    d.len = -1;
                </reverse>
            </step>
            <while
                id="lenLoop"
                description="Цикл уменьшения длины"
                test="(d.x[d.len] != d.x[d.i]) &amp;&amp; (d.len &gt; 0)"
                level="-1"
                true-comment-ru="Символы справа от общей области не совпадают."
                true-comment-en="Characters to the right of common substring do not match."
                false-comment-ru="Символы справа от общей области совпадают, либо длина подстроки нулевая, поэтому ее не урезаем."
                false-comment-en="Characters to the right of common area match, or the area's length is 0, so it's not trimmed."
            >
                <step
                    id="decreaseLen"
                    description="Подстрока не подходит. Примененяем функцию n(len)"
                    comment-ru="При добавлении по символу справа расширенные начало и конец не совпадают. Выбираем более короткие одинаковые подстроки в начале и конце."
```

```
                    comment-en="Extended head and tail of x[1]..x[i+1] do not
match. Choosing shorter substrings that are head and tail of x[1]..x[i]."
                >
                    <draw>
                        d.visualizer.updateState();
                    </draw>
                    <direct>
                        d.len = d.l[d.len-1];
                    </direct>
                    <reverse>
                        d.len = d.l[d.i-1];
                    </reverse>
                </step>
            </while>
            <if
                id="Cond"
                description="Условие"

                test="d.x[d.len] == d.x[d.i]"
                true-comment-ru="Начало и конец с дописанным символом x[i+1]
совпадают."
                true-comment-en="Extended head and tail of x[1]..x[i+1] are
equal."
                false-comment-ru="Не удалось найти начало и конец строки
x[1]..x[i], остающиеся равными при добавлении символа x[i+1]."
                false-comment-en="Could not find head and tail that remain equal
with addition of a new character x[i+1]."
                level="1"
            >
                <then>
                    <step
                        id="setNewLen"
                        description="Заносим найденную длину подстроки"
                        comment-ru="Записываем значение длины, увеличенное на 1,
в l[i+1]."
                        comment-en="Storing incremented length value to l[i+1]."
                    >
                        <draw>
                            d.visualizer.updateState();
                        </draw>
                        <direct>
                            d.l[d.i] = d.len + 1;
                        </direct>
                    </step>
                </then>
                <else>
                    <step
                        id="setZeroLen"
                        description="Заносим нулевую длину подстроки"
                        comment-ru="Заносим 0 в l[i+1]."
                        comment-en="Storing zero length value to l[i+1]."
                    >
                        <draw>
                            d.visualizer.updateState();
                        </draw>
                        <direct>
                            d.l[d.i] = 0;
                        </direct>
                    </step>
                </else>
            </if>
            <step
                id="inc"
                description="Инкремент"
```

```
                level="-1"
            >
                <direct>
                    d.i++;
                    d.len = -1;
                </direct>
                <reverse>
                    d.i--;
                    d.len = (d.l[d.i] == 0 ) ? 0 : d.l[d.i] - 1;
                </reverse>
            </step>
        </while>
        <finish
            comment-ru="Алгоритм завершил работу. Выбираем ячейки массива длин
со значениями, равными длине подстроки. Символы в этих позициях - последние сим-
волы найденных подстрок."
            comment-en="The algorithm has finished. Cells of l array are left
which value is equal to the substring length. Characters in those positions are
the last ones in found substrings."
        >
            <draw>
                d.visualizer.updateState();
            </draw>
        </finish>
    </auto>
</algorithm>
```

Приложение 4. Автоматически сгенерированный код

```
package ru.ifmo.vizi.kmp;

import ru.ifmo.vizi.base.auto.*;
import java.util.Locale;

public final class SearchKMP extends BaseAutomataWithListener {
    /**
      * Модель.
      */
    public final Data d = new Data();

    /**
      * Конструктор для языка
      */
    public SearchKMP(Locale locale) {
        super("ru.ifmo.vizi.kmp.Comments", locale);
        init(new Main(), d);
    }

    /**
      * Данные.
      */
    public final class Data {
        /**
          * Соединенные подстрока и строка.
          */
        public char x[] = "abbaa@abaabbaababaabbbabaaa".toCharArray();

        /**
          * Массив длин совпадений.
          */
        public int[] l = new int[x.length];

        /**
          * Максимальная длина совпадения.
          */
        public int len;

        /**
          * Длина обработанной строки.
          */
        public int i;

        /**
          * Экземпляр апплета.
          */
        public SearchKMPVisualizer visualizer;

        public String toString() {
            StringBuffer s = new StringBuffer();

            s.append("l = ");
            for( int li = 0; li < l.length; ++ li ) {
                s.append(l[li]);
            }
            s.append("\n");

            s.append("i = ");
            s.append(i);
```

```
        s.append("\n");

        s.append("len = ");
        s.append(len);
        s.append("\n");

        return s.toString();
    }
}

/**
  * Ищет подстроку в строке.
  */
private final class Main implements Automata {
    /**
      * Начальное состояние автомата.
      */
    private final int START_STATE = 0;

    /**
      * Конечное состояние автомата.
      */
    private final int END_STATE = 12;

    /**
      * Описания состояний.
      */
    private final String[] descriptions = new String[]{
        "Начальное состояние",
        "Инициализация",
        "Начало цикла",
        "Цикл",
        "Инициализация len",
        "Цикл уменьшения длины",
        "Подстрока не подходит. Примененяем функцию n(len)",
        "Условие",
        "Условие (окончание)",
        "Заносим найденную длину подстроки",
        "Заносим нулевую длину подстроки",
        "Инкремент",
        "Конечное состояние"
    };

    /**
      * Текущее состояние автомата.
      */
    private int state;

    /**
      * Текущий вложенный автомат.
      */
    private Automata child;

    /**
      * Переход в начальное состояние.
      */
    public void toStart() {
        state = START_STATE;
        child = null;
    }

    /**
      * Переход в конечное состояние.
      */
```

```
public void toEnd() {
    state = END_STATE;
    child = null;
}


/**
  * Находится ли автомат в начальном состоянии.
  */
public boolean isAtStart() {
    return state == START_STATE;
}

/**
  * Находится ли автомат в конечном состоянии.
  */
public boolean isAtEnd() {
    return state == END_STATE;
}

/**
  * Номер текущего шага.
  */
public int getStep() {
    return step;
}

/**
  * Сделать шаг вперед.
  */
public void stepForward(int level) {
    do {
        step++;
        // Переход в следующее состояние
        switch (state) {
            case START_STATE: { // Начальное состояние
                state = 1; // Инициализация
                break;
            }
            case 1: { // Инициализация
                state = 2; // Начало цикла
                break;
            }
            case 2: { // Начало цикла
                stack.pushBoolean(false);
                state = 3; // Цикл
                break;
            }
            case 3: { // Цикл
                if (d.i < d.x.length) {
                    state = 4; // Инициализация len
                } else {
                    state = END_STATE;
                }
                break;
            }
            case 4: { // Инициализация len
                stack.pushBoolean(false);
                state = 5; // Цикл уменьшения длины
                break;
            }
            case 5: { // Цикл уменьшения длины
                if ((d.x[d.len] != d.x[d.i]) && (d.len > 0)) {
                    state = 6; // Подстрока не подходит.
```

```
                        // Примененяем функцию n(len)
            } else {
                state = 7; // Условие
            }
            break;
        }
        case 6: { // Подстрока не подходит.
                  //  Примененяем функцию n(len)
            stack.pushBoolean(true);
            state = 5; // Цикл уменьшения длины
            break;
        }
        case 7: { // Условие
            if (d.x[d.len] == d.x[d.i]) {
                state = 9; // Заносим найденную длину подстроки
            } else {
                state = 10; // Заносим нулевую длину подстроки
            }
            break;
        }
        case 8: { // Условие (окончание)
            state = 11; // Инкремент
            break;
        }
        case 9: { // Заносим найденную длину подстроки
            stack.pushBoolean(true);
            state = 8; // Условие (окончание)
            break;
        }
        case 10: { // Заносим нулевую длину подстроки
            stack.pushBoolean(false);
            state = 8; // Условие (окончание)
            break;
        }
        case 11: { // Инкремент
            stack.pushBoolean(true);
            state = 3; // Цикл
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // Инициализация
            d.l[0] = 0;
            d.i = 1;
            d.len = -1;
            break;
        }
        case 2: { // Начало цикла
            d.i = 1;
            break;
        }
        case 3: { // Цикл
            break;
        }
        case 4: { // Инициализация len
            d.len = d.l[d.i-1];
            break;
        }
        case 5: { // Цикл уменьшения длины
            break;
        }
        case 6: { // Подстрока не подходит.
```

```
                    //  Примененяем функцию n(len)

                d.len = d.l[d.len-1];
                break;
            }
            case 7: { // Условие
                break;
            }
            case 8: { // Условие (окончание)
                break;
            }
            case 9: { // Заносим найденную длину подстроки
                d.l[d.i] = d.len + 1;
                break;
            }
            case 10: { // Заносим нулевую длину подстроки
                d.l[d.i] = 0;
                break;
            }
            case 11: { // Инкремент
                d.i++;
                d.len = -1;
                break;
            }
        }
    } while (!isInteresting(level));
}

/**
  * Сделать шаг назад.
  */
public void stepBackward(int level) {
    do {
        // Обращение действия в текущем состоянии
        switch (state) {
            case 1: { // Инициализация
                d.i = 0;
                break;
            }
            case 2: { // Начало цикла
                break;
            }
            case 3: { // Цикл
                break;
            }
            case 4: { // Инициализация len
                d.len = -1;
                break;
            }
            case 5: { // Цикл уменьшения длины
                break;
            }
            case 6: { // Подстрока не подходит.
                      //  Примененяем функцию n(len)
                d.len = d.l[d.i-1];
                break;
            }
            case 7: { // Условие
                break;
            }
            case 8: { // Условие (окончание)
                break;
            }
            case 9: { // Заносим найденную длину подстроки
```

```
            break;
        }
        case 10: { // Заносим нулевую длину подстроки
            break;
        }
        case 11: { // Инкремент
            d.i--;
            d.len = (d.l[d.i] == 0 ) ? 0 : d.l[d.i] - 1;
            break;
        }
    }

    // Переход в предыдущее состояние
    switch (state) {
        case 1: { // Инициализация
            state = START_STATE;
            break;
        }
        case 2: { // Начало цикла
            state = 1; // Инициализация
            break;
        }
        case 3: { // Цикл
            if (stack.popBoolean()) {
                state = 11; // Инкремент
            } else {
                state = 2; // Начало цикла
            }
            break;
        }
        case 4: { // Инициализация len
            state = 3; // Цикл
            break;
        }
        case 5: { // Цикл уменьшения длины
            if (stack.popBoolean()) {
                state = 6; // Подстрока не подходит.
                           // Примененяем функцию n(len)
            } else {
                state = 4; // Инициализация len
            }
            break;

        }
        case 6: { // Подстрока не подходит.
                  //  Примененяем функцию n(len)
            state = 5; // Цикл уменьшения длины
            break;
        }
        case 7: { // Условие
            state = 5; // Цикл уменьшения длины
            break;
        }
        case 8: { // Условие (окончание)
            if (stack.popBoolean()) {
                state = 9; // Заносим найденную длину подстроки
            } else {
                state = 10; // Заносим нулевую длину подстроки
            }
            break;
        }
        case 9: { // Заносим найденную длину подстроки
            state = 7; // Условие
            break;
```

```
            }
            case 10: { // Заносим нулевую длину подстроки
                state = 7; // Условие
                break;
            }
            case 11: { // Инкремент
                state = 8; // Условие (окончание)
                break;
            }
            case END_STATE: { // Начальное состояние
                state = 3; // Цикл
                break;
            }
        }

        step--;
    } while (!isInteresting(level));
}

/**
  * Интересно ли текущее состояние.
  */
public boolean isInteresting(int level) {
    // Интересность
    switch (state) {
        case START_STATE: // Начальное состояние
            return true;
        case 1: // Инициализация
            return level <= 0;
        case 2: // Начало цикла
            return level <= -1;
        case 3: // Цикл
            return level <= -1;
        case 4: // Инициализация len
            return level <= 0;
        case 5: // Цикл уменьшения длины
            return level <= -1;
        case 6: // Подстрока не подходит. Примененяем функцию n(len)
            return level <= 0;
        case 7: // Условие
            return level <= 1;
        case 8: // Условие (окончание)
            return level <= -1;
        case 9: // Заносим найденную длину подстроки
            return level <= 0;
        case 10: // Заносим нулевую длину подстроки
            return level <= 0;
        case 11: // Инкремент
            return level <= -1;
        case END_STATE: // Конечное состояние
            return true;
    }

    throw new RuntimeException("isInterest");
}

/**
  * Комментарий к текущему состоянию
  */
public String getComment() {
    String comment = "";
    Object[] args = null;
    // Выбор комментария
    switch (state) {
```

```
            case START_STATE: { // Начальное состояние
                comment = SearchKMP.this.getComment("Main.START_STATE");
                break;
            }
            case 1: { // Инициализация
                comment = SearchKMP.this.getComment("Main.Initialization");
                break;

            }
            case 4: { // Инициализация len
                comment = SearchKMP.this.getComment("Main.InitLen");
                break;
            }
            case 5: { // Цикл уменьшения длины
                if ((d.x[d.len] != d.x[d.i]) && (d.len > 0)) {
                    comment =
                       SearchKMP.this.getComment("Main.lenLoop.true");
                } else {
                    comment =
                       SearchKMP.this.getComment("Main.lenLoop.false");
                }
                break;
            }
            case 6: { // Подстрока не подходит. Примененяем функцию n(len)
                comment = SearchKMP.this.getComment("Main.decreaseLen");
                break;
            }
            case 7: { // Условие
                if (d.x[d.len] == d.x[d.i]) {
                    comment = SearchKMP.this.getComment("Main.Cond.true");
                } else {
                    comment = SearchKMP.this.getComment("Main.Cond.false");
                }
                break;
            }
            case 9: { // Заносим найденную длину подстроки
                comment = SearchKMP.this.getComment("Main.setNewLen");
                break;
            }
            case 10: { // Заносим нулевую длину подстроки
                comment = SearchKMP.this.getComment("Main.setZeroLen");
                break;
            }
            case END_STATE: { // Конечное состояние
                comment = SearchKMP.this.getComment("Main.END_STATE");
                break;
            }
        }

        return java.text.MessageFormat.format(comment, args);
    }

    /**
      * Выполняет действия по отрисовке состояния
      */
    public void drawState() {
        switch (state) {
            case START_STATE: { // Начальное состояние
                d.visualizer.updateState();
                break;
            }
            case 1: { // Инициализация
                d.visualizer.updateState();
                break;
```

```
                }
                case 4: { // Инициализация len
                    d.visualizer.updateState();
                    break;
                }
                case 6: { // Подстрока не подходит. Примененяем функцию n(len)
                    d.visualizer.updateState();
                    break;
                }
                case 9: { // Заносим найденную длину подстроки
                    d.visualizer.updateState();
                    break;
                }
                case 10: { // Заносим нулевую длину подстроки
                    d.visualizer.updateState();
                    break;
                }
                case END_STATE: { // Конечное состояние
                    d.visualizer.updateState();
                    break;
                }
            }
        }

        public StringBuffer toString(StringBuffer s) {
            s.append("Main ").append(state).append(" ");
            s.append('(');
            s.append(descriptions[state]);
            s.append(")\n");
            if (child != null && !child.isAtStart() && !child.isAtEnd()) {
                child.toString(s);
            }
            return s;
        }
    }
}
```

Приложение 5. Исходный код визуализатора

```
package ru.ifmo.vizi.kmp;

import ru.ifmo.vizi.base.Base;
import ru.ifmo.vizi.base.VisualizerParameters;
import ru.ifmo.vizi.base.ui.AutoControlsPane;
import ru.ifmo.vizi.base.ui.HintedButton;
import ru.ifmo.vizi.base.widgets.ShapeStyle;
import ru.ifmo.vizi.base.widgets.Rect;

import java.awt.*;
import java.awt.event.TextListener;
import java.awt.event.TextEvent;
import java.util.Stack;

/**
 * Аплет, визуализирующий алгоритм поиска подстроки Кнута-Морриса-Пратта.
 *
 * @author Ilya Zverev

 * @version 04.06.2004
 */
public class SearchKMPVisualizer extends Base {
    /** Разделитель подстроки и строки. */
    private static final char separator = '\1';
    /** Позиция разделителя в массиве <code>x</code> (длина подстроки). */
    private int separatorPos = -1;

    /** Автомат, реализующий визуализатор. */
    private final SearchKMP auto;
    /** Модель данных автомата. */
    private final SearchKMP.Data data;

    /** Стек элементов массива <code>x</code> */
    private final Stack str;
    /** Стек элементов массива <code>l</code> */
    private final Stack lengths;
    /** Строка максимальной ширины, которая может быть элементом массива l. */
    private final String maxValueString;

    /** Поле ввода подстроки */
    private final TextField subString;
    /** Поле ввода строки. */
    private final TextField string;
    /**
     * Кнопка применения введенных {@link #subString подстроки}
     * и {@link #string строки}.
     */
    private final HintedButton applyButton;
    /** Максимальная допустимая длина {@link #subString подстроки}. */
    private final int maxSubstrLength;
    /** Максимальная допустимая длина {@link #string строки}. */
    private final int maxStrLength;

    /** Набор стилей отображения элементов массивов. */
    private final ShapeStyle[] styleSet;
    /** Ссылка на окно приложения или аплета */
    private final Frame forefather;

    /** Надпись "Элементы:" */
```

```
private final Rect legendRect;
/** Обозначение начала и конца последовательности x[1]..x[i]. */
private final Rect lGreenRect;
/** Обозначение элементов x[len+1] и x[i+1]. */
private final Rect lBlueRect;
/** Обозначение элементов l[i], равных длине подстроки. */
private final Rect lYellowRect;

/**
 * Класс, принимащий сообщения от полей ввода подстроки и строки,
 * проверяющий длины строк и в зависимости от этого включающий
 * или выключащий кнопку применения изменений.
 */
private class ValidString implements TextListener {
    public void textValueChanged( TextEvent e ) {
        TextField field = (TextField)e.getSource();
        String txt = (field).getText();

        applyButton.setEnabled(
                txt.length() > 0 &&
                txt.length() <=
                    (field == string ? maxStrLength : maxSubstrLength)
        );
    }
}
/** Экземпляр класса {@link ValidString}, общий для обоих полей ввода. */
private final ValidString validString = new ValidString();

/**
 * Создает экземпляр визуализатора.
 * @param parameters параметры визуализатора.
 */
public SearchKMPVisualizer( VisualizerParameters parameters ) {
    super(parameters);
    this.forefather = parameters.getForefather();
    auto = new SearchKMP(locale);
    data = auto.d;
    data.visualizer = this;
    str = new Stack();
    lengths = new Stack();

    styleSet = ShapeStyle.loadStyleSet(config, "array");
    maxSubstrLength = config.getInteger("max-substr-length", 9);
    maxStrLength = config.getInteger("max-str-length", 30);
    maxValueString = config.getParameter("max-value-string",
                                         Integer.toString(maxSubstrLength));
    subString = new TextField(
            config.getParameter("default-substring", "abba"),
            maxSubstrLength
    );
    subString.addTextListener(validString);
    string = new TextField(
            config.getParameter("default-substring", "abaabbababbaabb"),
            maxStrLength
    );
    string.addTextListener(validString);
    applyButton = new HintedButton(config, "button-apply"){
        protected void click() {
            retrieveValue();
        }
    };

    legendRect = new Rect(styleSet, config.getParameter("label-legend"));
    lGreenRect = new Rect(styleSet, config.getParameter("label-seq"));
```

```
    lBlueRect = new Rect(styleSet, config.getParameter("label-i1"));
    lYellowRect = new Rect(styleSet, config.getParameter("label-found"));

    int lgap = 10;
    int lx = lgap;
    int ly = lgap;

    lx = adjustLegendRect(legendRect, 5, lx, ly );
    lx = adjustLegendRect(lGreenRect, 2, lx + lgap / 2, ly );
    lx = adjustLegendRect(lBlueRect, 3, lx + lgap / 2, ly );
    lx = adjustLegendRect(lYellowRect, 4, lx + lgap / 2, ly );

    retrieveValue();
    createInterface(auto);
}

/**
 * Вспомогательный метод для унификации процесса размещения
 * пояснений в окне приложения. Устанавливает у переданного
 * элемента заданный стиль, перемещает его в точку <code>(x,y)</code>,
 * настраивает размер в зависимости от его сообщения и
 * добавляет элемент в клиентскую область визуализатора.
 * @param r элемент строки обозначений.
 * @param s номер стиля элемента.
 * @param x позиция левой границы элемента.
 * @param y позиция верхней границы элемента.
 * @return позиция правой границы элемента.
 */
private int adjustLegendRect( Rect r, int s, int x, int y ) {
    r.setStyle(s);
    r.setLocation(x, y);
    r.adjustSize();
    clientPane.add(r);
    return x + r.getBounds().width;
}

/**
 * Создает панель управления визуализатором.
 * Дополнительно к основным элементам управления размещает поля ввода
 * подстроки и строки с метками и кнопку применения изменений.
 * @return созданная панель управления.
 */
protected Component createControlsPane() {
    Panel panel = new Panel(new BorderLayout());

    panel.add(new AutoControlsPane(config, auto, forefather, false),
              BorderLayout.CENTER);
    Panel bottomPanel = new Panel();
    bottomPanel.add(new Label(config.getParameter("label-search")));
    bottomPanel.add(subString);
    bottomPanel.add(new Label(config.getParameter("label-in")));
    bottomPanel.add(string);
    bottomPanel.add(applyButton);
    panel.add(bottomPanel, BorderLayout.SOUTH);

    return panel;
}

/**
 * Читает введенные значения из полей ввода, объединяет в
 * одну строку и вызывает {@link #setStringValue(java.lang.String)}.
 */
private void retrieveValue() {
    separatorPos = subString.getText().length();
```

```
    setStringValue(subString.getText() + separator + string.getText());
}

/**
 * Устанавливает новые данные массива <code>x</code> для визуализатора.
 * @param value строка для визуализатора.
 */
private void setStringValue(String value) {
    auto.toStart();
    auto.d.x = value.toCharArray();
    auto.d.l = new int[value.length()];
    auto.d.i = 0;
    while (str.size() < value.length()) {
        Rect rect = new Rect(styleSet);
        str.push(rect);
        clientPane.add(rect);
        Rect rect2 = new Rect(styleSet);
        lengths.push(rect2);
        clientPane.add(rect2);
    }
    while (str.size() > value.length()) {
        clientPane.remove((Component) str.pop());
        clientPane.remove((Component) lengths.pop());
    }
    clientPane.doLayout();
    updateState();
}

/**
 * Устанавливает стили элементов массивов в соответствии с
 * текущим состоянием визуализатора.
 */
public void updateState() {
    // зависим от i.
    // в строке l рисуем ровно i элементов.
    // если i == x.length, то конец алгоритма: выделяем ячейки с
    // l[i] == sub.length.
    for (int i = 0; i < data.x.length; i++) {
        Rect rect = (Rect) str.elementAt(i);
        if( data.x[i] == separator ) {
            rect.setMessage("");
        } else {
            StringBuffer s = new StringBuffer();
            s.append(data.x[i]);
            rect.setMessage(s.toString());
        }
        if( i + 1 == data.i + 1)
            rect.setStyle(3);
        else if( (i + 1 == data.len + 1) && data.i < data.x.length )
            rect.setStyle(3);
        else if( i + 1 <= data.len ||
                (i + 1 <= data.i && i + 1 > data.i - data.len) )
            rect.setStyle(2);
        else if( data.x[i] == separator )
            rect.setStyle(1);
        else
            rect.setStyle(0);

        Rect rect2 = (Rect) lengths.elementAt(i);
        if( data.i >= data.x.length && data.l[i] < separatorPos )
            rect2.setMessage("");
        else
            rect2.setMessage(
                    i+1 <= data.i ? Integer.toString(data.l[i]) : "");
```

```
                rect2.setStyle((data.l[i] == separatorPos) ? 4 : 0);
            }
            update(true);
        }

        /**
         * Размещает в клиентской области визуализатора элементы массивов.
         *
         * @param clientWidth ширина клиентской области.
         * @param clientHeight высота клиенской области.
         */
        protected void layoutClientPane(int clientWidth, int clientHeight) {
            int n = str.size();

            Rectangle mb = legendRect.getBounds();
            int x0 = mb.x + mb.height;

            int width = Math.round(clientWidth / (n + 1));
            int height = Math.min(width, (clientHeight - x0) / 4);
            int x = (clientWidth - width * n) / 2;
            int y = (clientHeight + x0 - height) / 2 - height - 1;

            for (int i = 0; i < n; i++) {
                Rect rect = (Rect) str.elementAt(i);
                rect.setBounds(x + i * width, y, width + 1, height + 1);
                rect.adjustFontSize(maxValueString);

                Rect rect2 = (Rect) lengths.elementAt(i);
                rect2.setBounds(x + i * width, y + height + height + 1,
                                width + 1, height + 1);
                rect2.adjustFontSize(maxValueString);
            }
        }
    }
```