Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

В. Ю. Лоторейчик

Построение визуализатора алгоритма генерации всех простых строк и цикла де Брюина на базе технологии *Vizi*

Программирование с явным выделением состояний
Проектная документация

Санкт-Петербург
2004

Содержание

Введение

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

Визуализатор – это программа, в процессе работы которой на экране компьютера динамически демонстрируется применение алгоритма к заданному набору данных и/или параметров. Простые строки применяются в процедуре сравнения с шаблоном и для построения цикла де Брюина. Алгоритм их генерации сходен с алгоритмами генерации перестановок, сочетаний и т. д.

В данной работе строится логика и реализация визуализатора алгоритма генерации всех простых строк и построения цикла де Брюина на базе технологии *Vizi.*

1. Анализ литературы

Информация о рассматриваемом алгоритме была взята из англоязычных черновиков очередного тома из серии «Искусство программирования» Дональда Кнута. Этот том посвящен комбинаторным алгоритмам.

Алгоритм описан по шагам (достаточно детально) для того, чтобы его можно было запрограммировать. В тексте приведены определения всех необходимых понятий, а также две теоремы, которые полностью обосновывают изложенный алгоритм. Доказательство первой из них приведено полностью. Кроме того, у Кнута описаны разные математические свойства генерируемого комбинаторного объекта.

В данной работе осуществлен перевод изложения Кнута по данному вопросу с английского языка на русский, построен визуализатор и разработана проектная документация.

2. Введение в теорию простых строк и описание алгоритма их генерации

Рассмотрим $\gamma = \alpha\beta$ – конкатенация двух строк; α называется *префиксом* γ, а β называется *суффиксом.* Суффикс или префикс строки γ называется *собственным,* если его длина положительна, но меньше длины γ. Таким образом, β собственный суффикс $\alpha\beta$ тогда и только тогда, когда $\alpha \neq \varepsilon$ и $\beta \neq \varepsilon$ (ε – пустая строка).

Определение А. С*трочка называется простой, если она не пуста и (лексикографически) меньше всех своих собственных суффиксов.*

Например, 01101 – не простая строка, потому что она больше, чем 01. Однако строка 01102 – простая, потому что она меньше, чем 1102, 102, 02, и 2. Предположим, что строки состоят из букв, цифр или других символов из линейно упорядоченного алфавита.

Лексикографический или словарный порядок – обычный способ сравнения строк. При этом можно записать $\alpha < \beta$ и сказать "α меньше чем β", когда α лексикографически меньше, чем β. Отметим, что всегда верно следующее: $\alpha \leq \alpha\beta$, а $\alpha < \alpha\beta$ тогда и только тогда, когда $\beta \neq \varepsilon$.

Простые строки часто называют *словами Линдона*, так как они были изобретены R. C. Lyndon'ом в 1954 г. Линдон называл их «стандартными последовательностями». Более простой термин «простая строка» оправдывается фундаментальной теоремой о факторизации, формулировка, которой будет приведена ниже в данной работе. Отдавая дань уважения Линдону, будем обозначать простые строки греческой буквой λ.

Несколько важных свойств простых строк были установлены Ченом, Фоксом и Линдоном в работе по теории групп в 1958 г. Среди этих свойств был и следующий важный результат.

Теорема А. *Непустая строка, которая меньше всех своих циклических сдвигов является простой.*

(Циклические сдвиги строки $a_1 \ldots a_n$ – это $a_2 \ldots a_n a_1$, $a_3 \ldots a_n a_1 a_2$, …, $a_n a_1 \ldots a_{n-1}$.)

Доказательство. Будем строить доказательство от противного. Рассмотрим строку γ, которая не будет простой, но будет меньше всех своих циклических сдвигов. Пусть γ имеет вид $\alpha\beta$. Строка γ не является простой, по причине того, что $\alpha \neq \varepsilon$ и $\gamma \geq \beta \neq \varepsilon$, и при этом γ меньше своего циклического сдвига $\beta\alpha$. Из условий $\beta \leq \gamma < \beta\alpha$ следует, что $\gamma = \beta\theta$ для некоторого $\theta < \alpha$. Однако, γ меньше своего циклического сдвига $\theta\beta$, поэтому $\theta < \alpha < \alpha\beta < \theta\beta$. Пришли к противоречию, так как α и θ одинаковой длины.

Пусть $L_m(n)$ – количество простых строк длины n над m-арным алфавитом. Каждая строка $a_1 \ldots a_n$, вместе со своими циклическими сдвигами образует d различных строк, где d некоторый делитель n. При этом можно показать, что все эти строки будут соответствовать одной простой строке длины d. Например, из строки 010010 с помощью циклических сдвигов можно получить строки 100100 и 001001, содержащие периодические части {010, 100, 001}, наименьшая из которых – 001. Можно показать, что

$$\sum_{d \backslash n} dL_m(d) = m^n, \forall m, n \geq 1.$$

Это семейство уравнений можно разрешить относительно $L_m(n)$. При этом получится

$$L_m(n) = \frac{1}{n} \sum_{d \backslash n} \mu(d) m^{n/d}.$$

Здесь $\mu(m)$ – *функция Мёбиуса,* которая определяется при всех $m \geq 1$ уравнением $\sum_{d \backslash m} \mu(d) = [m = 1]$.

В 70-х годах Г. Фредриксен и Дж. Майорана открыли изящный способ генерации всех m-арных (над алфавитом из m символов) простых строк длины n или меньше при возрастающем лексикографическом порядке. До описания этого алгоритма рассмотрим понятие n-расширения непустой строки λ. Оно представляет собой первые n символов бесконечной строки $\lambda\lambda\lambda\ldots$. Например, 10-расшиерение строки 123 – это строка 1231231231. Вообще говоря, если $|\lambda| = k$, то n-расширение это $\lambda^{[n/k]}\,\lambda'$, где λ' – префикс λ длины $n \bmod k$.

Определение В. *Строка называется предпростой, если она является непустым префиксом простой строки.*

Теорема В: *Строка длины $n > 0$ является предпростой, если и только если она является n-расширением простой строки λ длины $k \leq n$. Причем простая строка λ определяется единственным образом.*

Доказательство. Нетривиальное следствие из теоремы о факторизации, формулировка которой будет дана ниже.

Теорема В, кроме того, устанавливает, что существует взаимно однозначное соответствие между простыми строками длины меньшей или равной n и предпростыми строками длины n. Следующий алгоритм позволяет генерировать все m-арные предпростые строки в возрастающем порядке.

Алгоритм А *(Генерация простых и предпростых строк).* Этот алгоритм обходит все такие m-арные кортежи длины n вида $(a_1,\ldots,a_n)$, где $a_1\ldots a_n$ – предпростая строка. Он также определяет индекс j такой, что $a_1\ldots a_n$ является n-расширением простой строки $a_1\ldots a_j$.

- **Шаг 1:** [Инициализация.] Установим $a_1 \leftarrow \ldots \leftarrow a_n \leftarrow 0$, $j \leftarrow 1$; установим $a_1 \leftarrow -1$.
- **Шаг 2:** [Посещение.] Посетить $(a_1,\ldots,a_n)$ с индексом j.
- **Шаг 3:** [Подготовка к увеличению.] Установить $j \leftarrow n$. Затем, если $a_j = m - 1$, то уменьшать j до тех пор, пока не встретиться $a_j < m - 1$.
- **Шаг 4:** [Увеличение.] Завершить работу, если $j = 0$. В противном случае установить $a_j \leftarrow a_j + 1$. Таким образом, получим простую строку $a_1\ldots a_j$
- **Шаг 5:** [Построение n-расширения.] В цикле по $k \leftarrow j + 1, \ldots, n$ установить $a_k = a_{k-j}$. Перейти к шагу 2.

Рассмотрим пример работы Алгоритма А на примере генерации 32 четырехразрядных предпростых строк при $m = 3$ и $n = 4$ (таблица).

Таблица. Предпростые строки

0000	**0011**	**0022**	**0111**	**0122**	**0212**	**1**111	**12**12
0001	**0012**	**01**01	**0112**	**02**02	**022**0	**1112**	**122**1

0002	**002**0	**0102**	**012**0	**021**0	**0221**	**112**1	**1222**
0010	**0021**	**011**0	**0121**	**0211**	**0222**	**1122**	**2**222

(Жирным шрифтом выделены простые строки, соответствующие предпростым).

Теорема B объясняет, почему алгоритм правилен. Шаги 3-4 находят наименьшую *m*-арную простую строку длины меньше или равной n, которая превосходит предыдущую предпростую $a_1 \ldots a_n$. Отметим, что после увеличения a_1 с нуля до единицы, алгоритм осуществляет перебор всех (m – 1)-арных простых и предпростых строк, увеличенных на строку 1…1.

Алгоритм A лежит в основе построения цикла де Брюина. Если цифры $a_1, \ldots, a_j$ выводить на втором шаге алгоритма в тех случаях, когда j будет делителем n, то последовательность всех этих цифр сформирует цикл де Брюина. Например, в случае $m = 3$, а $n = 4$, следующая восемьдесят одна цифра будет выведена, и это будет циклом де Брюина:

0 0001 0002 0011 0012 0021 0022 01 0102 0111 0112 0121 0122 02 0211 0212 0221 0222 1 1112 1122 12 1222 2.

Здесь пропущены строки 001, 002, 011, …, 122, так как их длина не является делителем четырех.

Из анализа алгоритма A следует, что:

- среднее значение величины $n - j$ в шагах 3 и 5 примерно $m / (m - 1)^2$;
- общее время работы алгоритма и построения цикла де Брюина $O(m^n)$.

Приведем без доказательства следующую теорему:

Теорема C: *(о факторизации) Любая строка α может быть представлена в виде* $\alpha = \lambda_1 \lambda_2 \ldots \lambda_t$, $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_t$ *, где каждая* λ_j *– простая строка.*

3. Описание модели данных

Моделью данных называется класс, содержащий все необходимые алгоритму переменные и структуры данных. Для реализации алгоритма генерации всех простых строк и построения цикла де Брюина требуется модель данных содержащая:

- массив для генерации строк `a`;
- переменная цикла перехода `j`;
- переменная цикла дополнения `k`;
- размер алфавита `alph`;
- количество сгенерированных строк на текущий момент `num`;
- длина строки (цикла) де Брюина `dBlen`;
- массив для генерации строки цикла де Брюина `dB`;

- экземпляр аплета `visualizer`.

В последних версиях инструментального средства *Vizi* построение модели данных осуществляется автоматически на основе определенных в XML-представлении алгоритма глобальных и локальных переменных.

4. Описание алгоритма на языке *XML*

Описание алгоритма на языке *XML*, используемом в технологии *Vizi,* изоморфно строится по его реализации. Полученное XML-описание алгоритма, дополненное комментариями (текстовой информацией о визуализируемых шагах) и информацией о действиях по отрисовке состояний, помещается в файл `PrimeStrings-Algorithm.xml`, приведенный в приложении 1.

Определение каждой переменной производится с помощью тега `variable`. У этого тега имеется атрибут description, содержащий описание соответствующей переменной. Переменные, описанные внутри тегов `auto`, становятся локальными переменными соответствующей процедуры, а описанные вне данных тегов – глобальными переменными. В модель данных включаются все локальные и глобальные переменные. При этом к имени каждой локальной переменной добавляется префикс, состоящий из имени процедуры, в которой они определены, и символа подчеркивания.

В описании реализации алгоритма оператор ветвления представляется тегом `if`, а цикл с предусловием – тегом `while`. У перечисленных тегов есть атрибуты, которые, в частности, содержат текстовые описания вызываемых операторов, используемые при составлении комментариев в автоматически генерируемых файлах реализации, а также комментарии для понимания выполняемого шага алгоритма.

Действия, которые необходимо выполнять при работе алгоритма, локализуются внутри тегов `action`**.** Они, в свою очередь, размещаются в тегах `step`, отмечающих шаги алгоритма, которые требуется визуализировать, заостряя на них внимание пользователя. Важнейшим атрибутом этого тега является атрибут `level`, хранящий целую величину, называемую «интересностью» данного шага. В текущей реализации *Vizi* интересность может принимать значения –1, 0, 1.

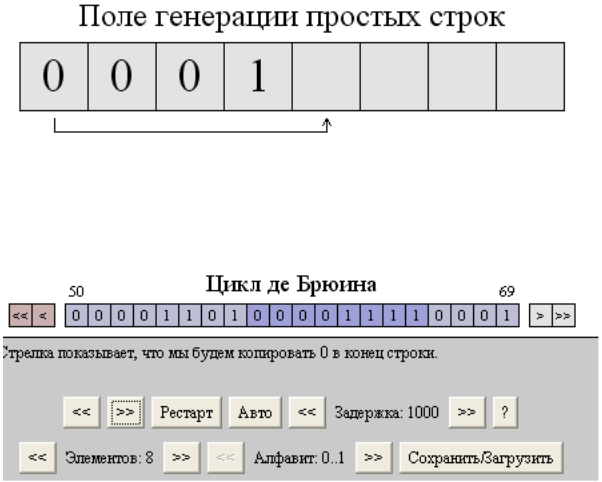
В случае если «интересность» имеет значение -1, то данный шаг не будет выделяться визуализатором, и программа сразу перейдет к содержимому следующего тега `step`. Если «интересность» имеет нулевое значение, то данный шаг будет выделяться визуализатором только при последовательном просмотре шагов в неавтоматическом режиме. Если же «интересность» имеет значение единица, то данный шаг будет всегда выделяться визуализатором.

В XML-описании алгоритма также должен использоваться тег `draw`, содержащий вызов метода визуализатора, выполняющего перерисовку. Так как дальнейшая реализация алгоритма осуществляется на основе автоматного подхода, все теги, которые отвечают за функционирование одного и того же автомата, собираются в одном теге `auto`. Эти теги, вместе с тегом `toString`, вкладываются в тег `algorithm`.

Кроме того, строится еще два файла. Первый из них (`PrimeStrings.xml`) содержит тег `visualizer`, хранящий общую информацию, такую как название визуализатора, имя автора и так далее, а также ссылки на два других файла – `PrimeStrings-Algorithm.xml` и `PrimeStrings-Configuration.xml`.

Третий файл (`PrimeStrings-Configuration.xml`) содержит параметры административного интерфейса визуализатора.

5. Описание интерфейса визуализатора



Рассмотрим пользовательский интерфейс визуализатора (рисунок).

Управление процессом визуализации пользователь осуществляет посредством нажатия на кнопки. Рассмотрим верхний ряд кнопок (слева направо). Первая и вторая кнопки, соответственно, осуществляют переход к следующему (предыдущему) шагу визуализации. Кнопка «Рестарт» позволяет запустить процесс визуализации сначала. Кнопка «Авто» включает автоматический режим. В автоматическом режиме переход к следующему шагу визуализации осуществляется автоматически, с задержкой, длительность которой устанавливается при помощи пятой и шестой кнопок и отображается в текстовой области, расположенной между ними. Седьмая кнопка (на которой изображен вопросительный знак) выводит краткую информацию о визуализаторе – название алгоритма, имена и адреса электронной почты автора, руководителя и создателя технологии.

Аналогично рассмотрим нижний ряд кнопок. Первая и вторая кнопки уменьшают и увеличивают соответственно максимальную длину генерируемых простых строк. Третья и четвертая кнопки определяют размерность алфавита, над которым генерируются строки.

Кнопка «Сохранить/Загрузить» выводит диалоговое окно, в котором можно задать параметры визуализатора и установить шаг алгоритма. Нажатие любой из кнопок нижнего ряда приводит к изменению параметров алгоритма, поэтому процесс визуализации его работы начинается сначала.

Пользователю на каждом визуализируемом шаге работы алгоритма предоставляется комментарий, а также графическое изображение текущего состояния генерируемой строки и текущего состояния генерируемого цикла де Брюина. Когда простая строка сгенерированна, то она подсвечивается (в данной конфигурации) красным цветом. После этого строится предпростое расширение, где стрелкой показывается, как осуществляется копирование символа. При построении по текущей предпростой строке простой строки рассматриваемый символ подсвечивается синим. Цикл де Брюина может достигать очень большой длины, поэтому он визуализируется фрагментарно с возможностью прокрутки. Кнопки «<» и «>» прокручивают цикл де Брюина на один символ, а кнопки «<<» и «>>» – на всю длину визуализируемого фрагмента. На рисунке показан цикл де Брюина с 50-го по 69-ый символ.

6. Автоматически сгенерированный исходный код, реализующий логику визуализатора

Рассмотрим исходный код, сгенерированный с помощью технологии *Vizi* (Приложение 2). Этот код содержит класс `PrimeStrings`, содержащий, в свою очередь, классы `Data` – модель данных, `Main` – реализация пары автоматов для движения по алгоритму в «прямом» и «обратных» направлениях. Эти автоматы имеют по двадцать одинаковых состояний.

Каждый автомат реализуется двумя операторами `switch`. В автомате для прямого прохода первый оператор обеспечивает переход в следующее состояние, а второй – действия в текущем состоянии. В автомате для обратного прохода первый оператор обеспечивает обращение действий в текущем состоянии, а второй переход в предыдущее состояние.

Основными методами класса `Main` являются методы `stepForward` и `stepBackward`. Первый из них обеспечивает переход в следующее состояние автомата с выполнением соответствующих действий, а второй – переход в обратном направлении. При этом каждый их этих методов может совершить несколько шагов за один вызов, в зависимости от того, считается ли данное состояние «интересным». «Интересность» состояния определяется методом `isInteresting`**,** использующим значение атрибута `level` тега `step` (разд. 4). Вывод комментариев к состояниям осуществляется методом `getComment`**,** а отрисовка состояний методом `drawState`. Привязка к состояниям комментариев и действий по отрисовке также осуществляется автоматически с помощью двух дополнительных операторов `switch`.

7. Исходный код пользовательского интерфейса визуализатора

В приложении 3 приведен исходный код пользовательского интерфейса визуализатора, написанный вручную на языке *Java* с использованием библиотеки *Vizi*.

Пользовательский интерфейс визуализатора реализован финальным классом `PrimeStringsVisualizer`, являющимся наследником класса `Base` библиотеки *Vizi*.

Данный класс имеет следующие поля:

- `auto` – экземпляр автомата;
- `data` – экземпляр модели данных автомата;
- `cells` – вектор (массив), хранящий прямоугольники для отрисовки строки;
- `elements` – спин-панель, задающая максимальную длину генерируемых простых строк;
- `alphabet` – спин-панель, задающая размер алфавита;
- `styleSet` – набор стилей для изображения массивов (строк);
- `cptstyleSet` – набор стилей для изображения надписей;
- `arrow` – массив линий, составляющих стрелку;
- `isArrow` – флаг, указывающий на присутствие стрелки на экране;
- `begL, endL` – задают начало и конец стрелки;
- `dBsz` – размер окна для цикла де Брюина;
- `dBwnd` – массив, хранящий прямоугольники для отрисовки цикла де Брюина;
- `dBstyles` – массив номеров стилей для изображения цикла де Брюина ;
- `dBmaxsize` – максимальный размер цикла де Брюина;
- `dBwndpos` – текущая позиция в цикле де Брюина;
- `dBoverfull` – флаг, указывающий, что цикл де Брюина достиг `dBmaxsize`;
- `dBcurrentstyle` – номер текущего стиля для изображения цикла де Брюина;
- `dBcpt` – надпись «Цикл де Брюина» (в текущей конфигурации);
- `Primecpt` – надпись «Поле генерации простых строк» (в текущей конфигурации);
- `dBbegcpt, dBendcpt` – надписи над циклом де Брюина, показывающие номера первого и последнего визуализируемого символа цикла;
- `saveLoadDialog` – диалоговое окно Сохранить/Загрузить;
- `LeftScroll, RightScroll` – кнопки для скроллинга цикла де Брюина;
- `FastLeftScroll, FastRightScroll` – кнопки для быстрого скроллинга цикла де Брюина.

Конструктор данного класса загружает данные конфигурации визуализатора, выделяет память для структур данных пользовательского интерфейса и создает пользовательский интерфейс.

Переопределенный метод `createControlsPane` добавляет две спин-панели «Элементов» и «Алфавит» и добавляет кнопку «Сохранить/Загрузить» к стандартной панели управления библиотеки *Vizi*.

Метод `layoutClientPane` располагает графические объекты в области визуализации. Данный метод использует ряд вспомогательных математических функций.

В приложении 4 приведены исходные коды дополнительных файлов визуализации. В них реализованы классы: `Line` – наследник класса `Shape`**,** `AdvancedRect` – наследник класса `Component`**.** Класс `Line` реализует графический примитив – линия и методы по его отрисовке. Класс `AdvancedRect` реализует элемент управления – кнопка. В этом классе хранится образ кнопки и содержится метод обработки нажатия.

8. Описание административного интерфейса (конфигурации) визуализатора

Конфигурирование визуализатора производится посредством изменения файла `PrimeStrings-Configuration.xml` (приложение 1). Все параметры конфигурации указываются в секции `configuration`. Используя XML-тег `property`**,** можно изменить такие параметры конфигурации, как высоту области для комментария (параметр `comment-height`), размер окошка, в котором визуализируется цикл де Брюина (параметр `dBsz`), максимальный допустимый размер цикла де Брюина (параметр `dBmaxsize`).

Для изменения максимальной длины генерируемых строк создается спин-панель (совокупность текстовой области, отображающей значение определенной величины, и кнопок для её изменения) `elements`, в XML-описании которой указываются все названия кнопок, подсказки (hints), минимальное и максимальное целое число, которое панель позволяет ввести. Для создания спин-панели используется XML-тег `spin-panel`**,** формат которого интуитивно ясен. Аналогично создается спин-панель `alphabet` для задания алфавита, над которым будут генерироваться простые строки.

Для предоставления возможности изменить цветовую гамму визуализатора без перекомпиляции аплета в *Vizi* существует механизм таблиц стилей (`stylesets`). В данном аплете используется две таблицы стилей: `array` и `caption`. Таблица стилей `array` содержит стили для отображения массивов (строк). Таблица стилей `caption` используется для отображения различных надписей в поле визуализации. Таблицы стилей создаются при помощи XML-тега `styleset`**,** их элементы при помощи XML-тега – `style`.

С помощью тега `message` задаются константные текстовые строки, а также строки с параметрами. XML-тег `group` задает диалоговое окно `SaveLoadDialog`, где тегами `property` описываются различные свойства этого окна.

Заключение

Отметим ряд преимуществ технологии *Vizi* перед реализациями, написанными вручную:

- построение по XML-описанию «прямого» алгоритма визуализатора не только его прямого прохода, но и обратного (при написании данного визуализатора эта возможность была использована частично);
- логика алгоритма реализована с помощью четырех автоматов, по два («прямой» и «обратный») для каждой процедуры. Общее число состояний автоматов равно 40, учитывая тот факт, что в прямом и обратном автомате используются одни и те же состояния. Каждый автомат реализуется двумя операторами `switch`;
- описание алгоритма при помощи языка *XML* позволяет автоматически вводить комментарии в код (отличающиеся от комментариев, вводимых для визуализации), а особенность валидации XML-кода технологией *Vizi* делают написание этих комментариев неизбежным, что значительно повышает читабельность кода, а, следовательно, упрощает его дальнейшее сопровождение;
- привязка к состояниям комментариев и действий по отрисовке также осуществляется автоматически с помощью дополнительных операторов `switch`;
- использование для построения визуализаторов единой технологии стандартизирует процесс разработки и позволяет во многих случаях избежать дублирования кода, уменьшая вероятность появления ошибок;
- стандартный эргономичный интерфейс визуализаторов имеет большое значение и обуславливает удобство изучения коллекций визуализаторов алгоритмов;
- компактность XML-описания.

Литература

1. *Удов Г. Г., Шалыто А. А.* Построение визуализатора алгоритма пирамидальной сортировки набора чисел на базе технологии *VIZI* // http://is.ifmo.ru
2. *Корнеев Г. А., Шалыто А. А.* Требования к визуализаторам алгоритмов, выполняемых на базе технологии *Vizi,* версия 4.0. // http://is.ifmo.ru
3. *Donald E. Knuth,* The art of computer programming (a draft of section 7.2.1.1: generating all n-tuples) // Stanford University.

Приложение 1. XML-описание визуализатора

Файл PrimeStrings.xml (основные параметры визуализатора)

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

<!--
    "PrimeStrings" visualizer description (example)
    Version: $Id: PrimeStrings.xml,v 1.3 2003/08/09 13:00:04 geo Exp $
-->

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

<visualizer

    id="PrimeStrings"
    package="ru.ifmo.vizi.prime_strings"
    main-class="PrimeStringsVisualizer"

    preferred-width="460"
    preferred-height="370"

    name-ru="Генерация всех простых строк и построение цикла де Брюина"
    name-en="Generating of all prime strings and constructing de Bruijn cycle"

    author-ru="Лоторейчик Владимир Юрьевич"
    author-en="Vladimir J. Lotoreichik"
    author-email="lotor@rain.ifmo.ru"

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

    copyright-ru="Copyright \u00A9 Кафедра КТ, СПб ГИТМО (ТУ), 2004"
    copyright-en="Copyright \u00A9 Computer Technologies Department, SPb IFMO,
2004"
>
    &algorithm;
    &configuration;
</visualizer>
```

Файл PrimeStrings-Algorithm.xml (xml-представление алгоритма)

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

<algorithm>
    <data>
        <variable description="Массив для генерации">int a[] = new int[]{1, 2, 3,
1, 3, 5, 6};</variable>
        <variable description="Переменная цикла">int j;</variable>
        <variable description="Переменная цикла">int k;</variable>
        <variable description="Размер алфавита">int alph;</variable>
        <variable description="Длина строки (цикла) де Брюина">int
dBlen;</variable>
        <variable description="Количество сгенерированных строк">int
num;</variable>
        <variable description="Цикл де Брюина">int dB[];</variable>
        <variable description="Экземпляр апплета">PrimeStringsVisualizer
visualizer;</variable>

        <toString>
```

```
        StringBuffer s = new StringBuffer();

        return s.toString();
    </toString>

</data>

<auto id="Main" description="Генерирует простые строки">

  <start
            comment-ru="Визуализатор алгоритма генерации всех простых строк и
                                                    построения цикла де Брюина."
       comment-en="Visualizer of algorythm of generating all prime strings and
                                                    constructing de Bruijn cycle."
    >
        <draw>
        </draw>
    </start>

  <step
        id="Initialization"
        description="Инициализация"
            comment-ru="Будут сгенерированы все простые строки длины не больше
                                                                        { 0}
                    символов и построен соответствующий цикл де Брюина."
            comment-en="There will be generated all prime strings of length no
                                                            more then { 0}
                                symbols and according de Bruijn cycle will be
                                                            constructed."
        comment-args="new Integer(d.a.length)"
    >
        <draw>
            d.visualizer.updateArrayEnhanced(0, -1, 0, 0);
            d.visualizer.updatedB(0, 0);
        </draw>
        <direct>
            d.num = 0;
            d.dBlen = 0;
        </direct>
    </step>

    <step
        id="MainLoopInit"
        description="Начало главного цикла"
                    comment-ru="Строка называется простой, если она не пустая и
                                                        (лексикографически)
                                                меньше всех своих суффиксов."
                            comment-en="A string is prime if it is nonempty and
                                                        (lexicographically)
                                                    less than all of its suffixes."
    >
        <direct>
            d.j = 0;
        </direct>
        <reverse>
        </reverse>
    </step>

  <while
        id="MainLoop"
        description="Главный Цикл"
        test="d.a[ 0]  &lt; d.alph"
        level="-1"
    >

      <step
            id="CurrentPrime"
```

```
                description="Извещение о том, что очередная простая строка
                                                    наконец-то сгенерирована"
             comment-ru="Сгенерирована текущая {0}-ая в лексикографическом
                                                    порядке простая строка."
             comment-en="Generated current {0}-th in lexicographical order
                                                          prime string."
        comment-args="new Integer(d.num)"
    >
        <draw>
            d.visualizer.updateArrayEnhanced(0, d.j, 3, 0);
        </draw>
        <direct>
            d.num++;
        </direct>
        <reverse>
            d.num--;
        </reverse>
</step>

    <if
        id="deBruijnAdvanceCondition"
        description="Условие расширение цикла де Брюина"
        test="(d.a.length - (d.a.length / (d.j + 1)) * (d.j + 1) ) == 0"
          true-comment-ru="Длина построенной простой строки ({0}) является
                                                делителем числа элементов ({1}),
                                        поэтому цикл де Брюина будет расширен."
           true-comment-en="Length of the generated prime string ({0}) is
                                                the divisor of number of
                                    elements ({1}), so de Bruijn cycle will be
                                                              enhanced."
             false-comment-ru="Длина построенной простой строки ({0}) не
                                        является делителем числа элементов ({1});
                                          цикл де Брюина остается без изменений."
            false-comment-en="Length of the generated prime string ({0})
                                                  isn't  the divisor of number of
                                elements ({1}), so de Bruijn cycle will be left
                                                        without changes."
            comment-args="new Integer(d.j + 1), new Integer(d.a.length)"
    >
        <draw>
            d.visualizer.updateArrayEnhanced(0, d.j, 3, 0);
            d.visualizer.updatedB(0, 0);
        </draw>
        <then>
            <step
                id="deBruijnAdvance"
                description="Расширение цикла де Брюина"

                        comment-en="Append current prime to the end of the
                                    connstructed part of the de Bruijn cycle."
            >
                <draw>
                    d.visualizer.updatedB(d.j + 1, 3);
                    d.visualizer.updateArrayEnhanced(0, d.j, 3, 0);
                </draw>
                <direct>
                    d.visualizer.dBadd(d.j + 1);
                </direct>
                <reverse>
                    d.visualizer.dBremove(d.j + 1);
                </reverse>
            </step>
        </then>
    </if>

<step
        id="Loop2Init"
                              description="Инициализация цикла дополнения"
```

```
        comment-ru="Начинаем процесс дополнения текущей простой строки до
                                                                предпростой.
                    Копируем 1-ый символ в конец строки, затем 2-ой в
                                            конец получившейся строки и т.д."
        comment-en="Let's begin the process of expanding current prime to
                                                                the preprime.
                    We should copy the 1-st symbol to the end of the
                                        string; than the 2nd to the end and so on."

    >
        <draw>
            d.visualizer.removeLink();
            d.visualizer.updatedB(0, 0);
        </draw>
        <direct>
            d.k = d.j + 1;
        </direct>
        <reverse>
            d.k = d.a.length;
        </reverse>
</step>

    <while
          id="Loop2"
          description="Цикл дополнения"
          test="d.k &lt; d.a.length"
        level="-1"
    >
        <step
            id="BeforeCopy"
            description=""
             comment-ru="Стрелка показывает, что мы будем копировать { 0}  в
                                                            конец строки."
               comment-en="Arrow shows that we will copy { 0}  to the end of
                                                             the string."
            comment-args="new Integer(d.a[ d.k - d.j - 1] )"
        >
            <draw>
                d.visualizer.updateArrayEnhanced(0, d.k - 1, 0, 0);
                d.visualizer.drawLink(d.k - d.j - 1, d.k);
            </draw>
            <direct>
            </direct>
            <reverse>
            </reverse>
        </step>

        <step
            id="Copy"
            description="Копирование"
            comment-ru="Скопировали."
            comment-en="Copied."
        >
            <draw>
                d.visualizer.removeLink();
                d.visualizer.updateArrayEnhanced(0, d.k, 0, 0);
            </draw>

            <direct>
                d.a[ d.k]  = d.a[ d.k - d.j - 1] ;
            </direct>
          <reverse>
              d.a[ d.k]  = d.alph;
            </reverse>
        </step>


    <step
```

```
            id="Inc"
            description="Инкремент"
            level="-1"
        >
            <direct>
                d.k++;
            </direct>
        <reverse>
                d.k--;
        </reverse>
        </step>
    </while>


 <step
        id="CurrentPrePrime"
      description="Показываем результат расширения"
        comment-ru="Построили предпростое расширение простой строки."
        comment-en="Preprime expansion of prime is built."
    >
        <draw>
            d.visualizer.updateArray(0, 0);
        </draw>
        <direct>
        </direct>
        <reverse>
        </reverse>
    </step>

     <step
        id="Loop1Init"
             description="Инициализация цикла перехода к следующей строке"
               comment-ru="Начинаем построение очередной простой строки из
                                                предпростой. Двигаемся с конца,
                                  пока не встретим не последний в алфавите символ,
                                                заменим его на следующий."
         comment-en="Let's generate next prime from the given preprime. We
                                                should move from the end,
                                till we meet not the last symbol in alphabet order
                                          and then we exchange it on the next one."
    >
        <draw>
            d.visualizer.updateArray(0, 0);
        </draw>

        <direct>
        stack.pushInteger(d.j);
            d.j = d.a.length - 1;

        </direct>
        <reverse>
            d.j = stack.popInteger();
        </reverse>
    </step>

<while
          id="Loop1"
          description="Цикл перехода"
          test="d.a[ d.j]  == d.alph"
          level="-1"
    >
        <step
            id="Move"
            description="Сдвиг окошка просмотра"
                  comment-ru="Текущий символ последний в алфавите, поэтому
                                                сдвигаем окошко влево."
             comment-en="Current symbol is the last in the alphabet, so we
                                            move the window to the left."
```

```
        >
            <draw>
                d.visualizer.updateArray(d.j, 1);
            </draw>

            <direct>
            </direct>
            <reverse>
            </reverse>
        </step>

        <step
            id="Dec"
            description="Декремент"
            level="-1"
        >
            <direct>
                d.j--;
            </direct>
            <reverse>
                d.j++;
            </reverse>
        </step>

    </while>

    <step
        id="BeforeModify"
        description="До модификации"
        comment-ru="Заменяем символ на следующий в алфавите."
        comment-en="Let's exchange symbol on the next in the alphabet."
    >
        <draw>
            d.visualizer.updateArray(d.j, 1);
        </draw>
      <direct>
        </direct>
        <reverse>
        </reverse>
    </step>

     <step
        id="Modify"
        description="После модификации"
        comment-ru="Заменили."
        comment-en="Exchange is done."
    >
        <draw>
            d.visualizer.updateArray(d.j, 1);
            d.visualizer.updatedB(0, 0);
        </draw>
      <direct>
            d.a[ d.j] ++;
        </direct>
      <reverse>
          d.a[ d.j] --;
      </reverse>
    </step>



</while>

<step
    id="Last"
    description="Последний шаг"
```

```
                    comment-ru="Сгенерирована последняя {0}-ая в лексикографическом
                                                                порядке строка
                          ее длина (1) делитель числа элементов ({1}), добавляем ее
                                                                в конец цикла."
                     comment-en="Generated last {0}-th in the lexicographical order
                                                                string, it's length
                          (1) is the divisor of number of elements, append it to the
                                                                end of cycle."
            comment-args="new Integer(d.num), new Integer(d.a.length)"
          >
            <draw>
                d.visualizer.updatedB(1, 3);
                d.visualizer.updateArrayEnhanced(0, 0, 3, 0);
            </draw>
            <direct>
                d.num++;
                d.visualizer.dBadd(d.j + 1);
            </direct>
            <reverse>
                d.num--;
                d.visualizer.dBremove(d.j + 1);
            </reverse>
        </step>

        <finish
            comment-ru="Цикл де Брюина построен."
            comment-en="de Bruijn cycle is built."
        >
            <draw>
                d.visualizer.updateArrayEnhanced(0, 0, 3, 0);
                d.visualizer.updatedB(0, 0);
            </draw>

        </finish>
    </auto>
</algorithm>
```

Файл PrimeStrings-Configuration.xml (конфигурация визуализатора)

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


<configuration>
    <property
        description = "de Bruijn window size"
        param       = "dBsz"
        value       = "20"
    />
    <property
        description = "de Bruijn cycle maximal size"
        param       = "dBmaxsize"
        value       = "5000"
    />

    <property
        description = "Comment pane height"
        param       = "comment-height"
        value       = "40"
    />
    <spin-panel
        param       = "elements"
        caption-ru  = "Элементов: {0,number,####}"
```

```
    caption-en  = "Elements: {0,number,####}"
    hint-ru     = "Длина предпростой строки"
    hint-en     = "Length of the preprime string"
    value       = "5"
    min-value   = "3"
    max-value   = "10"
    step        = "1"
>
    <button
        description = "Decrease length"
        param       = "button-less"
        caption-ru  = "&lt;&lt;"
        caption-en  = "&lt;&lt;"
        hint-ru     = "Уменьшить количество элементов"
        hint-en     = "Decrease length"
    />
    <button
        description = "Increase length"
        param       = "button-more"
        caption-ru  = ">>"
        caption-en  = ">>"
        hint-ru     = "Увеличить количество элементов"
        hint-en     = "Increase length"
    />
</spin-panel>


<spin-panel
    description = "SpinPanel to change alphabet size"
    param       = "alphabet"
    caption-ru  = "Алфавит: 0..{0,number,####}"
    caption-en  = "Alphabet: 0..{0,number,####}"
    hint-ru     = "Размер алфавита"
    hint-en     = "Alphabet size"
    value       = "1"
    min-value   = "1"
    max-value   = "9"
    step        = "1"
>
    <button
        description = "Decrease alphabet"
        param       = "button-less"
        caption-ru  = "&lt;&lt;"
        caption-en  = "&lt;&lt;"
        hint-ru     = "Уменьшить  размер алфавита"
        hint-en     = "Decrease alphabet"
    />
    <button
        description = "Increase alphabet"
        param       = "button-more"
        caption-ru  = ">>"
        caption-en  = ">>"
        hint-ru     = "Увеличить размер алфавита"
        hint-en     = "Increase alphabet"
    />
</spin-panel>

<styleset
    description = "Array style set"
    param       = "array"
>
    <style
        description     = "Inactive Style"
        text-color      = "000000"
        text-align      = "0.5"
        border-color    = "000000"
        border-status   = "true"
        fill-color      = "e0e0e0"
```

```
        fill-status     = "true"
        aspect-status   = "false"
        padding         = "0.2"
    >
        <font
            face        = "Serif"
            size        = "12"
            style       = "plain"
        />
    </style>
    <style
        description     = "Style for moving window"
        fill-color      = "a0a0d0"
    />
    <style
        description     = "de Bruijn cycle 1st style"
        fill-color      = "9090d0"
    />
    <style
        description     = "Active style (please attention smth. is done)"
        fill-color      = "c0a0a0"
    />
    <style
        description     = "de Bruijn cycle 2nd style"
        fill-color      = "b0b0d0"
    />


</styleset>

<styleset
    description = "Captions style set"
    param       = "caption"
>
    <style
        description    = "Standard style for captions"
        text-color     = "000000"
        text-align     = "0.5"
        border-color   = "000000"
        border-status  = "false"
        fill-color     = "000000"
        fill-status    = "false"
        aspect-status  = "false"
        padding        = "0.2"
    >
        <font
            face       = "Serif"
            size       = "14"
            style      = "plain"
        />
    </style>
</styleset>

<message
    description = "de Bruijn cycle caption"
    param       = "deBruijnCaption"
    message-ru  = "Цикл де Брюина"
    message-en  = "de Bruijn Cycle"
/>
<message
    description = "Prime strings generation field caption"
    param       = "PrimeCaption"
    message-ru  = "Поле генерации простых строк"
    message-en  = "Field of prime strings generation"
/>
<message
```

```
            description = 'Comment for "NumberOfElements" parameter in the output
                                                                                file'
        param       = "NumberElementsComment"
        message-ru  = "Длина предпростой строки ({0}  ... {1})"
        message-en  = "Length of the preprime ({0}  ... {1})"
    />
    <message
        description = 'Comment for "Alphabet" parameter in the output file'
        param       = "AlphabetComment"
        message-ru  = "Размер алфавита 0.. ({0}  ... {1})"
        message-en  = "Alphabet size 0.. ({0}  ... {1})"
    />
    <message
        description = 'Comment for "Step" parameter in the output file'
        param       = "StepComment"
        message-ru  = "Номер шага"
        message-en  = "Current step"
    />

    <group
        description = "Save/Load dialog configuration"
        param       = "SaveLoadDialog"
    >
        <property
            description = "Height of the comment pane"
            param       = "CommentPane-lines"
            value       = "2"
        />
        <property
            description = "Width of the text area"
            param       = "columns"
            value       = "40"
        />
        <property
            description = "Height of the text area"
            param       = "rows"
            value       = "7"
        />
    </group>


</configuration>
```

Приложение 2. Сгенерированный код автомата

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

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

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

    /**
      * Данные.
      */
    public final class Data {
```

```
        /**
          * Массив для генерации.
          */
        public int a[] = new int[]{1, 2, 3, 1, 3, 5, 6};

        /**
          * Переменная цикла.
          */
        public int j;

        /**
          * Переменная цикла.
          */
        public int k;

        /**
          * Размер алфавита.
          */
        public int alph;

        /**
          * Длина строки (цикла) де Брюина.
          */
        public int dBlen;

        /**
          * Количество сгенерированных строк.
          */
        public int num;

        /**
          * Цикл де Брюина.
          */
        public int dB[];

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

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

            return s.toString();
        }
    }

    /**
      * Генерирует простые строки.
      */
    private final class Main implements Automata {
        /**
          * Начальное состояние автомата.
          */
        private final int START_STATE = 0;

        /**
          * Конечное состояние автомата.
          */
        private final int END_STATE = 21;

        /**
          * Описания состояний.
          */
        private final String[] descriptions = new String[]{"Начальное состояние",
"Инициализация", "Начало главного цикла", "Главный Цикл", "Извещение о том, что
очередная простая строка наконец-то сгенерирована", "Условие расширение цикла де
Брюина", "Условие расширение цикла де Брюина (окончание)", "Расширение цикла де
```

```
Брюина", "Инициализация цикла дополнения", "Цикл дополнения", "", "Копирование",
"Инкремент", "Показываем результат расширения", "Инициализация цикла перехода к
следующей строке", "Цикл перехода", "Сдвиг окошка просмотра", "Декремент", "До
модификации", "После модификации", "Последний шаг", "Конечное состояние"};
```

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

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

        /**
          * Переход в начальнное 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 в перед.
          */
        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 главного цикла
                        break;
                    }
```

```
case 2: {  // Начало главного цикла
    stack.pushBoolean(false);
    state = 3; // Главный Цикл
    break;
}
case 3: {  // Главный Цикл
    if (d.a[ 0]  < d.alph) {
          state = 4; // Извещение о том, что очередная простая
                                    строка наконец-то сгенерирована
    } else {
        state = 20; // Последний шаг
    }
    break;
}
    case 4: {  // Извещение о том, что очередная простая строка
                                        наконец-то сгенерирована
    state = 5; // Условие расширение цикла де Брюина
    break;
}
case 5: {  // Условие расширение цикла де Брюина
     if ((d.a.length - (d.a.length / (d.j + 1)) * (d.j + 1) )
                                                        == 0) {
        state = 7; // Расширение цикла де Брюина
    } else {
        stack.pushBoolean(false);
             state = 6; // Условие расширение цикла де Брюина
                                                    (окончание)
    }
    break;
}
case 6: {  // Условие расширение цикла де Брюина (окончание)
    state = 8; // Инициализация цикла дополнения
    break;
}
case 7: {  // Расширение цикла де Брюина
    stack.pushBoolean(true);
             state = 6; // Условие расширение цикла де Брюина
                                                    (окончание)
    break;
}
case 8: {  // Инициализация цикла дополнения
    stack.pushBoolean(false);
    state = 9; // Цикл дополнения
    break;
}
case 9: {  // Цикл дополнения
    if (d.k < d.a.length) {
        state = 10;
    } else {
        state = 13; // Показываем результат расширения
    }
    break;
}
case 10: {
    state = 11; // Копирование
    break;
}
case 11: {  // Копирование
    state = 12; // Инкремент
    break;
}
case 12: {  // Инкремент
    stack.pushBoolean(true);
    state = 9; // Цикл дополнения
    break;
}
case 13: {  // Показываем результат расширения
```

```
                state = 14; // Инициализация цикла перехода к следующей
                                                                строке
            break;
        }
        case 14: {  // Инициализация цикла перехода к следующей строке
            stack.pushBoolean(false);
            state = 15; // Цикл перехода
            break;
        }
        case 15: {  // Цикл перехода
            if (d.a[ d.j] == d.alph) {
                state = 16; // Сдвиг окошка просмотра
            } else {
                state = 18; // До модификации
            }
            break;
        }
        case 16: {  // Сдвиг окошка просмотра
            state = 17; // Декремент
            break;
        }
        case 17: {  // Декремент
            stack.pushBoolean(true);
            state = 15; // Цикл перехода
            break;
        }
        case 18: {  // До модификации
            state = 19; // После модификации
            break;
        }
        case 19: {  // После модификации
            stack.pushBoolean(true);
            state = 3; // Главный Цикл
            break;
        }
        case 20: {  // Последний шаг
            state = END_STATE;
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: {  // Инициализация
            d.num = 0;
            d.dBlen = 0;
            break;
        }
        case 2: {  // Начало главного цикла
            d.j = 0;
            break;
        }
        case 3: {  // Главный Цикл
            break;
        }
            case 4: {  // Извещение о том, что очередная простая строка
                                                    наконец-то сгенерирована
            d.num++;
            break;
        }
        case 5: {  // Условие расширение цикла де Брюина
            break;
        }
        case 6: {  // Условие расширение цикла де Брюина (окончание)
            break;
        }
        case 7: {  // Расширение цикла де Брюина
            d.visualizer.dBadd(d.j + 1);
```

```
                    break;
                }
                case 8: {  // Инициализация цикла дополнения
                    d.k = d.j + 1;
                    break;
                }
                case 9: {  // Цикл дополнения
                    break;
                }
                case 10: {
                    break;
                }
                case 11: {  // Копирование
                    d.a[ d.k]  = d.a[ d.k - d.j  - 1] ;
                    break;
                }
                case 12: {  // Инкремент
                    d.k++;
                    break;
                }
                case 13: {  // Показываем результат расширения
                    break;
                }
                case 14: {  // Инициализация цикла перехода к следующей строке
                                    stack.pushInteger(d.j);
                                        d.j = d.a.length - 1;
                    break;
                }
                case 15: {  // Цикл перехода
                    break;
                }
                case 16: {  // Сдвиг окошка просмотра
                    break;
                }
                case 17: {  // Декремент
                    d.j--;
                    break;
                }
                case 18: {  // До модификации
                    break;
                }
                case 19: {  // После модификации
                    d.a[ d.j] ++;
                    break;
                }
                case 20: {  // Последний шаг
                    d.num++;
                    d.visualizer.dBadd(d.j + 1);
                    break;
                }
            }
        } while (!isInteresting(level));
}

/**
  * Сделать шаг в назад.
  */
public void stepBackward(int level) {
    do {
        // Обращение действия в текущем состоянии
        switch (state) {
            case 1: {  // Инициализация
                break;
            }
            case 2: {  // Начало главного цикла
                break;
            }
            case 3: {  // Главный Цикл
```

```
            break;
        }
            case 4: {  // Извещение о том, что очередная простая строка
                                                наконец-то сгенерирована
            d.num--;
            break;
        }
        case 5: {  // Условие расширение цикла де Брюина
            break;
        }
        case 6: {  // Условие расширение цикла де Брюина (окончание)
            break;
        }
        case 7: {  // Расширение цикла де Брюина
            d.visualizer.dBremove(d.j + 1);
            break;
        }
        case 8: {  // Инициализация цикла дополнения
            d.k = d.a.length;
            break;
        }
        case 9: {  // Цикл дополнения
            break;
        }
        case 10: {
            break;
        }
        case 11: {  // Копирование
                                    d.a[ d.k]  = d.alph;
            break;
        }
        case 12: {  // Инкремент
            d.k--;
            break;
        }
        case 13: {  // Показываем результат расширения
            break;
        }
        case 14: {  // Инициализация цикла перехода к следующей строке
            d.j = stack.popInteger();
            break;
        }
        case 15: {  // Цикл перехода
            break;
        }
        case 16: {  // Сдвиг окошка просмотра
            break;
        }
        case 17: {  // Декремент
            d.j++;
            break;
        }
        case 18: {  // До модификации
            break;
        }
        case 19: {  // После модификации
                                d.a[ d.j] --;
            break;
        }
        case 20: {  // Последний шаг
            d.num--;
            d.visualizer.dBremove(d.j + 1);
            break;
        }
    }

    // Переход в предыдущее состояние
    switch (state) {
```

```
case 1: {  // Инициализация
    state = START_STATE;
    break;
}
case 2: {  // Начало главного цикла
    state = 1; // Инициализация
    break;
}
case 3: {  // Главный Цикл
    if (stack.popBoolean()) {
        state = 19; // После модификации
    } else {
        state = 2; // Начало главного цикла
    }
    break;
}
    case 4: {  // Извещение о том, что очередная простая строка
                                          наконец-то сгенерирована
    state = 3; // Главный Цикл
    break;
}
case 5: {  // Условие расширение цикла де Брюина
          state = 4; // Извещение о том, что очередная простая
                                    строка наконец-то сгенерирована
    break;
}
case 6: {  // Условие расширение цикла де Брюина (окончание)
    if (stack.popBoolean()) {
        state = 7; // Расширение цикла де Брюина
    } else {
        state = 5; // Условие расширение цикла де Брюина
    }
    break;
}
case 7: {  // Расширение цикла де Брюина
    state = 5; // Условие расширение цикла де Брюина
    break;
}
case 8: {  // Инициализация цикла дополнения
              state = 6; // Условие расширение цикла де Брюина
                                                    (окончание)
    break;
}
case 9: {  // Цикл дополнения
    if (stack.popBoolean()) {
        state = 12; // Инкремент
    } else {
        state = 8; // Инициализация цикла дополнения
    }
    break;
}
case 10: {
    state = 9; // Цикл дополнения
    break;
}
case 11: {  // Копирование
    state = 10;
    break;
}
case 12: {  // Инкремент
    state = 11; // Копирование
    break;
}
case 13: {  // Показываем результат расширения
    state = 9; // Цикл дополнения
    break;
}
case 14: {  // Инициализация цикла перехода к следующей строке
```

```
                    state = 13; // Показываем результат расширения
                    break;
                }
                case 15: {  // Цикл перехода
                    if (stack.popBoolean()) {
                        state = 17; // Декремент
                    } else {
                                state = 14; // Инициализация цикла перехода к
                                                                    следующей строке
                    }
                    break;
                }
                case 16: {  // Сдвиг окошка просмотра
                    state = 15; // Цикл перехода
                    break;
                }
                case 17: {  // Декремент
                    state = 16; // Сдвиг окошка просмотра
                    break;
                }
                case 18: {  // До модификации
                    state = 15; // Цикл перехода
                    break;
                }
                case 19: {  // После модификации
                    state = 18; // До модификации
                    break;
                }
                case 20: {  // Последний шаг
                    state = 3; // Главный Цикл
                    break;
                }
                case END_STATE: {  // Начальное состояние
                    state = 20; // Последний шаг
                    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 главного цикла
                return level <= 0;
            case 3: // Главный Цикл
                return level <= -1;
             case 4: // Извещение о том, что очередная простая строка наконец-
                                                                    то сгенерирована
                return level <= 0;
            case 5: // Условие расширение цикла де Брюина
                return level <= 0;
            case 6: // Условие расширение цикла де Брюина (окончание)
                return level <= -1;
            case 7: // Расширение цикла де Брюина
                return level <= 0;
            case 8: // Инициализация цикла дополнения
                return level <= 0;
            case 9: // Цикл дополнения
```

```
                    return level <= -1;
                case 10:
                    return level <= 0;
                case 11: // Копирование
                    return level <= 0;
                case 12: // Инкремент
                    return level <= -1;
                case 13: // Показываем результат расширения
                    return level <= 0;
                case 14: // Инициализация цикла перехода к следующей строке
                    return level <= 0;
                case 15: // Цикл перехода
                    return level <= -1;
                case 16: // Сдвиг окошка просмотра
                    return level <= 0;
                case 17: // Декремент
                    return level <= -1;
                case 18: // До модификации
                    return level <= 0;
                case 19: // После модификации
                    return level <= 0;
                case 20: // Последний шаг
                    return level <= 0;
                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 = PrimeStrings.this.getComment("Main.START_STATE");
                    break;
                }
                case 1: {  // Инициализация
                    comment =
PrimeStrings.this.getComment("Main.Initialization");
                    args = new Object[]{new Integer(d.a.length)};
                    break;
                }
                case 2: {  // Начало главного цикла
                    comment = PrimeStrings.this.getComment("Main.MainLoopInit");
                    break;
                }
                        case 4: {  // Извещение о том, что очередная простая строка
                                                                наконец-то сгенерирована
                    comment = PrimeStrings.this.getComment("Main.CurrentPrime");
                    args = new Object[]{new Integer(d.num)};
                    break;
                }
                case 5: {  // Условие расширение цикла де Брюина
                       if ((d.a.length - (d.a.length / (d.j + 1)) * (d.j + 1) ) ==
                                                                            0) {
                        comment =
PrimeStrings.this.getComment("Main.deBruijnAdvanceCondition.true");
                    } else {
                        comment =
PrimeStrings.this.getComment("Main.deBruijnAdvanceCondition.false");
                    }
```

```
                    args = new Object[]{new Integer(d.j + 1), new
Integer(d.a.length)};
                    break;
                }
                case 7: {  // Расширение цикла де Брюина
                    comment =
PrimeStrings.this.getComment("Main.deBruijnAdvance");
                    break;
                }
                case 8: {  // Инициализация цикла дополнения
                    comment = PrimeStrings.this.getComment("Main.Loop2Init");
                    break;
                }
                case 10: {
                    comment = PrimeStrings.this.getComment("Main.BeforeCopy");
                    args = new Object[]{new Integer(d.a[ d.k - d.j - 1])};
                    break;
                }
                case 11: {  // Копирование
                    comment = PrimeStrings.this.getComment("Main.Copy");
                    break;
                }
                case 13: {  // Показываем результат расширения
                    comment =
PrimeStrings.this.getComment("Main.CurrentPrePrime");
                    break;
                }
                case 14: {  // Инициализация цикла перехода к следующей строке
                    comment = PrimeStrings.this.getComment("Main.Loop1Init");
                    break;
                }
                case 16: {  // Сдвиг окошка просмотра
                    comment = PrimeStrings.this.getComment("Main.Move");
                    break;
                }
                case 18: {  // До модификации
                    comment = PrimeStrings.this.getComment("Main.BeforeModify");
                    break;
                }
                case 19: {  // После модификации
                    comment = PrimeStrings.this.getComment("Main.Modify");
                    break;
                }
                case 20: {  // Последний шаг
                    comment = PrimeStrings.this.getComment("Main.Last");
                                args = new Object[]{new Integer(d.num), new
                                                    Integer(d.a.length)};
                    break;
                }
                case END_STATE: {  // Конечное состояние
                    comment = PrimeStrings.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
                    break;
                }
                case 1: {  // Инициализация
                    d.visualizer.updateArrayEnhanced(0, -1, 0, 0);
```

```
                d.visualizer.updatedB(0, 0);
                break;
            }
                case 4: {  // Извещение о том, что очередная простая строка
                                                наконец-то сгенерирована
                d.visualizer.updateArrayEnhanced(0, d.j, 3, 0);
                break;
            }
            case 5: {  // Условие расширение цикла де Брюина
                d.visualizer.updateArrayEnhanced(0, d.j, 3, 0);
                d.visualizer.updatedB(0, 0);
                break;
            }
            case 7: {  // Расширение цикла де Брюина
                d.visualizer.updatedB(d.j + 1, 3);
                d.visualizer.updateArrayEnhanced(0, d.j, 3, 0);
                break;
            }
            case 8: {  // Инициализация цикла дополнения
                d.visualizer.removeLink();
                d.visualizer.updatedB(0, 0);
                break;
            }
            case 10: {
                d.visualizer.updateArrayEnhanced(0, d.k - 1, 0, 0);
                d.visualizer.drawLink(d.k - d.j - 1, d.k);
                break;
            }
            case 11: {  // Копирование
                d.visualizer.removeLink();
                d.visualizer.updateArrayEnhanced(0, d.k, 0, 0);
                break;
            }
            case 13: {  // Показываем результат расширения
                d.visualizer.updateArray(0, 0);
                break;
            }
            case 14: {  // Инициализация цикла перехода к следующей строке
                d.visualizer.updateArray(0, 0);
                break;
            }
            case 16: {  // Сдвиг окошка просмотра
                d.visualizer.updateArray(d.j, 1);
                break;
            }
            case 18: {  // До модификации
                d.visualizer.updateArray(d.j, 1);
                break;
            }
            case 19: {  // После модификации
                d.visualizer.updateArray(d.j, 1);
                d.visualizer.updatedB(0, 0);
                break;
            }
            case 20: {  // Последний шаг
                d.visualizer.updatedB(1, 3);
                d.visualizer.updateArrayEnhanced(0, 0, 3, 0);
                break;
            }
            case END_STATE: {  // Конечное состояние
                d.visualizer.updateArrayEnhanced(0, 0, 3, 0);
                d.visualizer.updatedB(0, 0);
                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 3. Исходный код интерфейса визуализатора

```
package ru.ifmo.vizi.prime_strings;

import ru.ifmo.vizi.base.ui.*;
import ru.ifmo.vizi.base.*;
import ru.ifmo.vizi.base.widgets.Rect;
import ru.ifmo.vizi.base.widgets.ShapeStyle;

import java.awt.*;
import java.util.Stack;

/**
 * PrimeStrings applet.
 *
 * @author  Lotoreichik Vladimir
 * @version $Id: PrimeStringsVisualizer.java,v 0.1 2004/01/10 11:41:22
 */
public final class PrimeStringsVisualizer extends Base {
    /**
     * PrimeStrings automata instance.
     */
    private final PrimeStrings auto;

    /**
     * Primestrings automata data.
     */
    private final PrimeStrings.Data data;

    /**
     * Cells with array elements.
     * Vector of {@link Rect}.
     */
    private final Stack cells;

    /**
     * Number of elements in array.
     */
    private final SpinPanel elements;

    /**
     * Alphabet size.
     */
    private final SpinPanel alphabet;


   /**
     * Array shape style set.
    */
    private final ShapeStyle[] styleSet;

    /**
```

```
 * Captions shape style set.
 */
private final ShapeStyle[] cptstyleSet;

private final Frame forefather;

/*
 * Arrow lines.
 */
private final Line[] arrow = new Line[5];

/*
 * Flag which to show arrow or not.
 */
private boolean isArrow;

/*
 * de Bruijn window size.
 */
private int dBsz;
/*
 * de Bruijn window.
 */
private final Rect[] dBwnd;

/*
 * de Bruijn shape styles.
 */
private int[] dBstyles;

/*
 * Maximal size of de Bruijn cycle.
 */
private int dBmaxSize;

/*
 * Current position on the de Bruijn cycle.
 */
private int dBwndPos;

/*
 * Flag which indicates that de Bruijn cycle is overfulled.
 */
private boolean dBoverfull;

/*
 *  It indicates current style for de Bruijn cycle.
 */
private int dBcurrentstyle;

/*
 * de Bruijn caption.
 */
private final Rect dBcpt;

/*
 *  Prime Strings caption.
 */
private final Rect Primecpt;

/*
 *  de Bruijn Begin and End captions.
 */
private final Rect dBbegcpt, dBendcpt;
/**
 * Save/load dialog.
 */
private SaveLoadDialog saveLoadDialog;
```

```
 private int begL, endL;

/**
  * Scrolling Buttons.
  */

 private final AdvancedRect LeftScroll, RightScroll;

/**
  * Fast scrolling Buttons.
  */

 private AdvancedRect FastLeftScroll, FastRightScroll;

 /**
  * Creates a new Prime Strings visualizer.
  *
  * @param parameters visualizer parameters.
  */
 public PrimeStringsVisualizer(VisualizerParameters parameters) {
     super(parameters);
     this.forefather = parameters.getForefather();
     auto = new PrimeStrings(locale);
     data = auto.d;
     data.visualizer = this;
     cells = new Stack();

     styleSet     = ShapeStyle.loadStyleSet(config, "array");
     cptstyleSet  = ShapeStyle.loadStyleSet(config, "caption");

     elements = new SpinPanel(config, "elements") {
         protected void click(double value) {
             setArraySize(getIntValue());
         }
     };

     dBsz = config.getInteger("dBsz");
     dBwnd = new Rect[ dBsz] ;
     for (int i = 0; i < dBsz; i++) {
         dBwnd[ i]  = new Rect(styleSet);
         dBwnd[ i] .setStyle(0);
         dBwnd[ i] .setMessage("");
         clientPane.add(dBwnd[ i] );
     }

     dBcpt = new Rect(cptstyleSet);
     dBcpt.setStyle(0);
     dBcpt.setMessage(config.getParameter("deBruijnCaption"));
     clientPane.add(dBcpt);

     Primecpt = new Rect(cptstyleSet);
     Primecpt.setStyle(0);
     Primecpt.setMessage(config.getParameter("PrimeCaption"));
     clientPane.add(Primecpt);

     dBbegcpt = new Rect(cptstyleSet);
     dBbegcpt.setStyle(0);
     dBbegcpt.setMessage(config.getParameter("deBruijnCaption"));
     clientPane.add(dBbegcpt);

     dBendcpt = new Rect(cptstyleSet);
     dBendcpt.setStyle(0);
     dBendcpt.setMessage(config.getParameter("deBruijnCaption"));
     clientPane.add(dBendcpt);

     LeftScroll = new AdvancedRect(styleSet, "<", 0, 3, 3) {
         protected void proceed() {
```

```
                if (dBwndPos > 0) setdBwndPos(dBwndPos - 1);
            }
        };

        RightScroll = new AdvancedRect(styleSet, ">", 0, 3, 3) {
            protected void proceed() {
                if (dBwndPos + dBsz < data.dBlen) setdBwndPos(dBwndPos + 1);
            }
        };

        clientPane.add(LeftScroll);
        clientPane.add(RightScroll);
        clientPane.add(LeftScroll.rect);
        clientPane.add(RightScroll.rect);

        FastLeftScroll = new AdvancedRect(styleSet, "<<", 0, 3, 3) {
            protected void proceed() {
                if (dBwndPos > 0) setdBwndPos(Math.max(0, dBwndPos - dBsz));
            }
        };

        FastRightScroll = new AdvancedRect(styleSet, ">>", 0, 3, 3) {
            protected void proceed() {
                 if (dBwndPos + dBsz < data.dBlen) setdBwndPos(Math.min(data.dBlen
                                                        - dBsz, dBwndPos + dBsz));
            }
        };

        clientPane.add(FastLeftScroll);
        clientPane.add(FastRightScroll);
        clientPane.add(FastLeftScroll.rect);
        clientPane.add(FastRightScroll.rect);


        setArraySize(elements.getIntValue());

      alphabet = new SpinPanel(config, "alphabet") {
            protected void click(double value) {
            setAlphabetSize(getIntValue());
            }
        };

        setAlphabetSize(alphabet.getIntValue());


        for (int i = 0; i < 5; i++) {
            arrow[ i]  = new Line(styleSet, -1, -1, -1, -1);
        }

        isArrow = false;

        dBwndPos       = 0;
        dBmaxSize      = config.getInteger("dBmaxsize");
        dBoverfull     = false;
        data.dB        = new int[ dBmaxSize] ;
        dBstyles       = new int[ dBmaxSize] ;
        dBcurrentstyle = 2;

        for (int i = 0; i < dBmaxSize; i++) dBstyles[ i]  = 0;

        createInterface(auto);

    }

    /**
     * This method creates panel with visualizer controls.
     *
     * @return controls pane.
```

```
 */
public Component createControlsPane() {
    Panel panel = new Panel(new BorderLayout());

              panel.add(new AutoControlsPane(config, auto, forefather, false),
                                                              BorderLayout.NORTH);

    Panel centerPanel = new Panel();
    centerPanel.add(elements);
    centerPanel.add(alphabet);


    //Panel bottomPanel = new Panel();

    if (config.getBoolean("button-ShowSaveLoad")) {
        centerPanel.add(new HintedButton(config, "button-SaveLoad") {
            protected void click() {
                saveLoadDialog.center();
                StringBuffer buffer = new StringBuffer();

                buffer.append("/* ").append(
                    I18n.message(
                        config.getParameter("NumberElementsComment"),
                        new Double(elements.getMinValue()),
                        new Double(elements.getMaxValue())
                    )
                ).append(" */\n");

                                                buffer.append("PreprimeLength =
                                          ").append(data.a.length).append("\n");

                buffer.append("/* ").append(
                    I18n.message(
                        config.getParameter("AlphabetComment"),
                        new Double(alphabet.getMinValue()),
                        new Double(alphabet.getMaxValue())
                    )
                ).append(" */\n");

                buffer.append("Alphabet = ").append(data.alph).append("\n");

                buffer.append("/* ").append(
                    config.getParameter("StepComment")
                ).append(" */\n");
                buffer.append("Step = ").append(auto.getStep());
                saveLoadDialog.show(buffer.toString());
            }
        });
    }
    //panel.add(bottomPanel, BorderLayout.SOUTH);
    panel.add(centerPanel, BorderLayout.CENTER);
    saveLoadDialog = new SaveLoadDialog(config, forefather) {
        public boolean load(String text) throws Exception {
            SmartTokenizer tokenizer = new SmartTokenizer(text, config);
            tokenizer.expect("PreprimeLength");
            tokenizer.expect("=");
            setArraySize(tokenizer.nextInt(
                (int) elements.getMinValue(),
                (int) elements.getMaxValue()
            ));
            elements.setValue(data.a.length);

            tokenizer.expect("Alphabet");
            tokenizer.expect("=");
            data.alph = tokenizer.nextInt(
                (int) alphabet.getMinValue(),
                (int) alphabet.getMaxValue()
            );
```

```
                alphabet.setValue(data.alph);


                tokenizer.expect("Step");
                tokenizer.expect("=");
                rewind(tokenizer.nextInt());

                tokenizer.expectEOF();

                return true;
            }
        };


        return panel;
    }

    /**
     * Sets new array size.
     *
     * @param size new array size.
     */
    private void setArraySize(int size) {
        data.a = new int[ size];
        while (cells.size() < size) {
            Rect rect = new Rect(styleSet);
            cells.push(rect);
            clientPane.add(rect);
        }
        while (cells.size() > size) {
            clientPane.remove((Component) cells.pop());
        }
        clientPane.doLayout();


        init();

    }

     /**
     * Sets new alphabet size.
     *
     * @param size new alphabet size.
     */
    private void setAlphabetSize(int size) {
        data.alph = size;
        init();
    }

    /**
     * Initializes.
     */
    public void init() {
        auto.toStart();

        for (int i = 0; i < data.a.length; i++) {
            data.a[ i]  = 0;
        }

        data.dBlen = 0;
        dBwndPos   = 0;

        updateArrayEnhanced(0, -1, 0, 0);
        updatedB(0, 0);
        removeLink();

    }
```

```
    /**
     * Updates array view.
     *
     * @param activeCell current active cell.
     * @param activeStyle style of active cell.
     */
    public void updateArray(int activeCell, int activeStyle) {

        for (int i = 0; i < data.a.length; i++) {
            Rect rect = (Rect) cells.elementAt(i);
            rect.setMessage(Integer.toString(data.a[ i] ));
            rect.setStyle(i == activeCell ? activeStyle : 0);
        }
        update(true);
    }

    /**
     * Updates array view.
     *
     * @param activeCell current active cell.
     * @param activeStyle style of active cell.
     */
    public void updateArrayEnhanced(int begCell, int endCell, int activeStyle,
int passiveStyle) {

        for (int i = 0; i < data.a.length; i++) {
            Rect rect = (Rect) cells.elementAt(i);
            if ((i >= begCell) && (i <= endCell)) {
                rect.setMessage(Integer.toString(data.a[ i] ));
                rect.setStyle(activeStyle);
            }
            else {
                rect.setMessage("");
                rect.setStyle(passiveStyle);
            }

        }
        update(true);
    }

    /**
     * Draws link.
     *
     * @param begLink Link begin.
     * @param endLink Link end.
     */
    public void drawLink(int begLink, int endLink) {
        Rect rect1 = (Rect) cells.elementAt(begLink);
        Rect rect2 = (Rect) cells.elementAt(endLink);
        Point p1 = rect1.getLocation();
        Point p2 = rect2.getLocation();
        Dimension d1 = rect1.getSize();
        Dimension d2 = rect2.getSize();
        p1.x = p1.x + d1.width  / 2;
        p2.x = p2.x + d2.width  / 2;
        p1.y = p1.y + d1.height / 2;
        p2.y = p2.y + d2.height / 2;

         arrow[ 0] .SetLine(p1.x,  p1.y + d1.height * 6 / 10, p1.x, p1.y + d1.height
                                                                  * 8 / 10);
         arrow[ 1] .SetLine(p1.x,  p1.y + d1.height * 8 / 10, p2.x, p2.y + d2.height
                                                                  * 8 / 10);
         arrow[ 2] .SetLine(p2.x,  p2.y + d2.height * 8 / 10, p2.x, p2.y + d2.height
                                                                  * 6 / 10);
          arrow[ 3] .SetLine(p2.x,  p2.y + d2.height * 6 / 10, p2.x - d2.width / 15,
                                                               p2.y + d2.height * 7 / 10);
          arrow[ 4] .SetLine(p2.x,  p2.y + d2.height * 6 / 10, p2.x + d2.width / 15,
                                                               p2.y + d2.height * 7 / 10);
```

```
        for (int i = 0; i < 5; i++) {
           clientPane.add(arrow[ i] );
        }

        isArrow = true;

        begL = begLink;
        endL = endLink;

        update(true);
    }

   /**
     * Removes link.
     *
     */
    public void removeLink() {
        if (isArrow)
        {
            for (int i = 0; i < 5; i++) {
                clientPane.remove(arrow[ i] );
            }
        }
        isArrow = false;
        update(true);
    }

   /**
     * Updates de Bruijn cycle.
     * @param lastlentgh
     * @param laststyle
     */

    public void updatedB(int lastlength, int laststyle) {

        for (int i = 0; i < Math.min(dBsz, data.dBlen - dBwndPos); i++) {
            dBwnd[ i] .setMessage(Integer.toString(data.dB[ i + dBwndPos] ));
            if (data.dB[ i + dBwndPos] > 9) dBwnd[ i] .setMessage("...");
            if ((data.dBlen -  i - dBwndPos <= lastlength) && (!dBoverfull))
dBwnd[ i] .setStyle(laststyle);
            else dBwnd[ i] .setStyle(dBstyles[ dBwndPos + i] );
        }

        for (int i = Math.min(dBsz, data.dBlen - dBwndPos); i < dBsz; i++) {
             dBwnd[ i] .setMessage("");
             dBwnd[ i] .setStyle(0);
        }

        if (dBwndPos > 0) LeftScroll.setActive(); else LeftScroll.setPassive();
        if (dBwndPos + dBsz < data.dBlen) RightScroll.setActive(); else
RightScroll.setPassive();

        if (dBwndPos > 0) FastLeftScroll.setActive(); else
FastLeftScroll.setPassive();
        if (dBwndPos + dBsz < data.dBlen) FastRightScroll.setActive(); else
FastRightScroll.setPassive();

        dBbegcpt.setMessage(Integer.toString(dBwndPos + 1));
        dBendcpt.setMessage(Integer.toString(dBwndPos + dBsz));

        update(true);
    }

    /**
     * Adds current prime to the de Bruijn cycle.
     * @param primelength Length of the current prime.
     */
```

```
public void dBadd(int primelength) {

    if (data.dBlen < dBmaxSize - dBsz) {

        dBoverfull = false;
      for (int i = 0; i < primelength; i++) {
            data.dB[ data.dBlen + i]   = data.a[ i] ;
            dBstyles[ data.dBlen + i]  = dBcurrentstyle;
        }

        data.dBlen = data.dBlen + primelength;

    }
    else if (!dBoverfull) {
        data.dB[ data.dBlen]  = data.num;
        dBstyles[ data.dBlen]  = dBcurrentstyle;
        data.dBlen++;
        dBoverfull = true;
    }
    if (data.dBlen <= dBsz) dBwndPos = 0;
    if (data.dBlen >  dBsz) dBwndPos = data.dBlen - dBsz;

    if (dBcurrentstyle == 2) dBcurrentstyle = 4; else dBcurrentstyle = 2;
}

/**
 *
 *
 */

public void dBremove(int primelength) {
    if ((dBoverfull) && (data.num == data.dB[ data.dBlen - 1])) data.dBlen--;

    if ((!dBoverfull) || (data.num < data.dB[ data.dBlen])) {
        data.dBlen = data.dBlen - primelength;
        if (data.dBlen <= dBsz) dBwndPos = 0;
        if (data.dBlen >  dBsz) dBwndPos = data.dBlen - dBsz;

        dBoverfull = false;
    }

    if (dBcurrentstyle == 2) dBcurrentstyle = 4; else dBcurrentstyle = 2;


}

/**
 * Set Position on the de Bruijn cycle.
 * @param wndPos
 */

public void setdBwndPos(int wndPos) {
    dBwndPos = wndPos;
    updatedB(0, 0);
}
/**
 * Rewinds algorithm to the specified step.
 *
 * @param step step of the algorith to rewind to.
 */

private void rewind(int step) {

    init();

    while (!auto.isAtEnd() && auto.getStep() < step) {
        auto.stepForward(0);
```

```
        }
    }

    /**
     * Invoked when client pane shoud be layouted.
     *
     * @param clientWidth client pane width.
     * @param clientHeight client pane height.
     */
    protected void layoutClientPane(int clientWidth, int clientHeight) {

        int n = cells.size();

        int width = Math.round(clientWidth / (n + 1));
        int height = Math.min(width,  7 * clientHeight /  18);
        int y = Math.min(clientWidth / 11, clientHeight / 6);
        int x = (clientWidth - width * n) / 2;



        for (int i = 0; i < n; i++) {
            Rect rect = (Rect) cells.elementAt(i);
            rect.setBounds(x + i * width, y, width + 1, height + 1);
            rect.adjustFontSize("00");
        }

        int clientWidth0 = dBsz * clientWidth / (dBsz + 5);
        int dBwidth  = Math.round(clientWidth0 / (dBsz + 1));
        int dBheight = Math.min(dBwidth, clientHeight * 2 / 21);
        int dBx      = (clientWidth - clientWidth0) / 2 + (clientWidth0 - dBwidth
* dBsz) / 2;
        int dBy      = clientHeight - dBheight * 3 / 2;

        for (int i = 0; i < dBsz; i++) {
           dBwnd[ i] .setBounds(dBx + i * dBwidth, dBy, dBwidth + 1, dBheight + 1);
           dBwnd[ i] .adjustFontSize("0");
        }

                dBbegcpt.setBounds(dBx, dBy - dBheight ,  dBwidth + 1, dBheight );
                                dBbegcpt.adjustFontSize(dBbegcpt.getMessage());

          dBendcpt.setBounds(dBx + (dBsz - 1) * dBwidth, dBy - dBheight ,  dBwidth
                                                          + 1, dBheight );
                                dBendcpt.adjustFontSize(dBendcpt.getMessage());

              LeftScroll.rect.setBounds(dBx -  3 * dBwidth / 2,  dBy, dBwidth + 1,
                                                            dBheight + 1);
                                           LeftScroll.rect.adjustFontSize("0");

               RightScroll.rect.setBounds(dBx + (2 * dBsz + 1) * dBwidth / 2, dBy,
                                                    dBwidth + 1, dBheight + 1);
                                          RightScroll.rect.adjustFontSize("0");

                                  LeftScroll.setBounds(LeftScroll.rect.getBounds());
                                RightScroll.setBounds(RightScroll.rect.getBounds());

          FastLeftScroll.rect.setBounds(dBx -  5 * dBwidth / 2,  dBy, dBwidth + 1,
                                                            dBheight + 1);
                                       FastLeftScroll.rect.adjustFontSize("0");

           FastRightScroll.rect.setBounds(dBx + (2 * dBsz + 3) * dBwidth / 2, dBy,
                                                    dBwidth + 1, dBheight + 1);
                                     FastRightScroll.rect.adjustFontSize("0");

                         FastLeftScroll.setBounds(FastLeftScroll.rect.getBounds());
                       FastRightScroll.setBounds(FastRightScroll.rect.getBounds());
```

```
            dBcpt.setBounds(dBx, dBy - 16 * dBheight / 10,  clientWidth - 2 * dBx, 3
                                                                    * dBheight / 2);
                                                dBcpt.adjustFontSize(dBcpt.getMessage());

                Primecpt.setBounds(x, 5 * y / 35,  clientWidth - 2 * x, 30 * y / 35);
            Primecpt.adjustFontSize(Primecpt.getMessage());

            //Made so because earlier there were no double bufferisation
            if (isArrow) {  removeLink(); drawLink(begL, endL); }

        }
}
```

Приложение 4. Исходный код дополнительных компонент

Класс, реализующий линию (Line.java)

```
package ru.ifmo.vizi.prime_strings;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import ru.ifmo.vizi.base.widgets.*;

/**
 * Line shape.
 *
 * @author  Alex Kotov (modified by Lotoreichik Vladimir)
 * @version $Id: line.java,v 1.0 2003/12/05 23:50:00 cat Exp $
 */
public final class Line extends Shape {
    int x1;
    int y1;
    int x2;
    int y2;

    public static final int SIMPLE = 0;
    public static final int LINKED = 1;
    public static final int REPRESENTED = 2;


    /**
     * Creates a new rectangle with specified style set, empty message
     * and square corneres.
     *
     * @param styleSet shape's style set.
     */
    public Line(ShapeStyle styleSet[], int out_x1, int out_y1, int out_x2, int
out_y2) {
      this(styleSet, out_x1, out_y1, out_x2, out_y2, SIMPLE);
    }


    /**
     * Creates a new rectangle with specified style set, empty message
     * and square corneres.
     *
     * @param styleSet shape's style set.
     */
    public Line(ShapeStyle styleSet[], int out_x1, int out_y1, int out_x2, int
out_y2, int styleNumber) {
      super(styleSet);
      setStyle(styleNumber);
      x1 = (out_x1 > out_x2) ? (out_x1 - out_x2) : 0;
      y1 = (out_y1 > out_y2) ? (out_y1 - out_y2) : 0;
```

```
      x2 = (out_x1 < out_x2) ? (out_x2 - out_x1) : 0;
      y2 = (out_y1 < out_y2) ? (out_y2 - out_y1) : 0;
      setLocation(out_x1 - x1, out_y1 - y1);
      setSize(Math.abs(out_x2 - out_x1) + 1, Math.abs(out_y2 - out_y1) + 1);
    }

    public Dimension fit(Dimension size) {
      return size;
    }

    /**
     * Change rectangle.
     */
    public void SetLine(int out_x1, int out_y1, int out_x2, int out_y2) {
        x1 = (out_x1 > out_x2) ? (out_x1 - out_x2) : 0;
      y1 = (out_y1 > out_y2) ? (out_y1 - out_y2) : 0;
      x2 = (out_x1 < out_x2) ? (out_x2 - out_x1) : 0;
      y2 = (out_y1 < out_y2) ? (out_y2 - out_y1) : 0;
      setLocation(out_x1 - x1, out_y1 - y1);
      setSize(Math.abs(out_x2 - out_x1) + 1, Math.abs(out_y2 - out_y1) + 1);
    }


    /**
     * Paints this component.
     *
     * @param g graphics context for painting.
     */
    public void paint(Graphics g) {
      g.setColor(look.getBorderColor(style));
      g.drawLine(x1, y1, x2, y2);
    }
}
```

Класс, реализующий кнопку (AdvanceRect.java)

```
package ru.ifmo.vizi.prime_strings;

import java.awt.*;
import java.awt.event.*;

import ru.ifmo.vizi.base.widgets.*;

/**
 * Rect with MouseListener..
 *
 * @author  Lotoreichik Vladimir
 * @version $Id: advancedrect.java,v 0.99 22.01.04
 */
public abstract class AdvancedRect extends Component implements MouseListener{

    private int inactive;

    private int exited;

    private int entered;

    public Rect rect;

    private boolean active;

    AdvancedRect(ShapeStyle[] styleSet, String message, int style0, int style1,
int style2) {
      addMouseListener(this);
        inactive = style0;
        entered  = style1;
```

```
        exited   = style2;
        rect     = new Rect(styleSet, message);
        rect.setStyle(inactive);
    }

  /*
    private boolean getState(int x, int y) {

      boolean   res;
      Point     p = rect.getLocation();
      Dimension d = rect.getSize();

      if ( ( x >= p.x ) && ( x <= p.x + d.width ) &&
           ( y >= p.y ) && ( y <= p.y + d.height) ) res = true;
      else res = false;

      return res;
    }
    */
    public void setActive() {
        active = true;
        rect.setStyle(exited);
    }

    public void setPassive() {
        active = false;
        rect.setStyle(inactive);
    }

    public void mouseClicked(MouseEvent e) {
              proceed();

    }

    public void mousePressed(MouseEvent e) {
     //System.out.println(e.getX());
    }

    public void mouseReleased(MouseEvent e) {

    }

    public void mouseEntered(MouseEvent e) {
    }

    public void mouseExited(MouseEvent e) {
    }

    public void mouseDragged(MouseEvent e) {
    }

    public void mouseMoved(MouseEvent e) {

    }

    protected abstract void proceed();

}
```