Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра “Компьютерные технологии”

Д.С. Белешко

Построение визуализатора алгоритма генерации кодов Грея без циклов 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

2005

Содержание

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.

В данной работе строится визуализатор алгоритма генерации кодов Грея без циклов на базе технологии *Vizi*.

Коды Грея — двоичные коды, в которых каждый следующий набор битов отличается от предыдущего только в одном разряде.

1. Анализ литературы

Информация о рассматриваемом алгоритме содержится в электронной книге Д. Кнута [1]. Описание алгоритма, предложенное Д. Кнутом, достаточно сухо и кратко. В данной работе приведено полностью переработанное и дополненное описание алгоритма и его доказательство.

2. Описание и анализ алгоритма генерации кодов Грея без циклов

Код Грея – это такой способ кодирования целых чисел, при котором закодированное число и число, следующее за ним, отличаются только одним битом. С помощью этих кодов можно перебрать все 2^n комбинаций из n битов, изменяя при каждой итерации только один бит. Эта концепция может быть обобщена для любого основания системы счисления, но в дальнейшем будут рассматриваться только бинарные коды Грея.

Коды Грея можно определить двумя рекурсивными правилами:

1) $G_0 = \epsilon$.
2) $G_{n+1} = 0G_n, 1G_n^R$.

Здесь G_n представляет собой последовательность кодов Грея, состоящих из n битов; ϵ – пустую строку; G_n^R – последовательность G_n в обратном порядке следования кодов. При этом запись $0G_n$ указывает на последовательность G_n с дополнительным 0 перед каждым кодом. Аналогично определяется $1G_n^R$.

Так как последний код G_n совпадает с первым кодом G_n^R, то, в соответствии с двумя выше определенными правилами, получим, что каждый следующий код последовательности G_{n+1} отличается от предыдущего только одним битом. Таким образом, получается, что данное определение удовлетворяет основному свойству кодов Грея.

Для $n = 3$ последовательность кодов Грея имеет вид:

000
001
011
010
110
111
101
100

Обратим внимание, что такой код вручную строится с использованием зеркальных отображений соответствующих разрядов.

Определим рекурсивно еще одну последовательность строк, но в десятичной системе счисления:

1. $F_0 = 0$.
2. $F_n = F_{n-1}\ n\ F_{n-1}$.

Здесь F_n представляет собой соответствующую строку десятичных чисел.

Например, для $n = 3$ последовательность строк будет иметь вид:

0

0 1 0

0 1 0 2 0 1 0

Рассмотрим n-ю строку: *0102010*. С помощью этой строки можно сформировать последовательность кодов Грея, состоящих из трех битов. Это выполняется следующим образом:

1. Выводим начальный код, состоящий из n нулей.
2. Берем очередное (сначала – первое) число j из строки F_n (выборка производится слева направо) и изменяем j-й бит кода (нумерация битов начинается справа налево).
3. Выводим код и переходим ко второму шагу.

В результате получается последовательность n-битных кодов Грея с использованием только одной строки чисел.

Отметим важное свойство строки F_n: если взять k-е число строки (пусть значение числа равно j), то 2^j будет наибольшим четным делителем числа k. Используя это свойство, можно получить строку F_n без предварительных рекурсивных вычислений.

Для этого необходимо представить двоичное число k специальным образом. Для удобства будем использовать массив `q[0..n]` двоичного представления числа k.

Для перехода к следующему числу $k+1$, нужно выполнить две операции с массивом q (прибавление единицы):

1. Выбираем такой наименьший индекс i, что `q[i] = 0` и для всех $m < i$ устанавливаем `q[m] = 0`.
2. Устанавливаем `q[i] = 1`.

Для оптимизации этих операций введем массив чисел `f[0..n]`, который будет неявно определять k. Этот массив может быть интерпретирован, как набор указателей на элементы массива `q`. Значения массива `f` формируются по следующим правилам:

1. Если `q[i] = 1`, а `q[i - 1] = 0`, то `f[i] = m`, где m – такой наименьший индекс, что `q[m] = 0` и $m > i$.
2. Иначе `f[i] = i`.

Тогда операция прибавления единицы к числу k может быть записана с помощью масиива `f` таким образом (действия аналогичны операциям с `q`):

1. `j = f[0], f[0] = 0`.
2. `f[j] = f[j + 1], f[j + 1] = j + 1`.

Кроме того, что массив `f` представляет число k, он также позволяет сразу получить необходимое число строки F_n. Это число содержит нулевой элемент массива. Действительно, в элементе массива `f[0]` содержится наименьший индекс j нулевого элемента двоичного разложения числа k. Таким образом, j указывает число разрядов, на которые можно сдвинуть вправо двоичное разложение числа k – на сколько степеней двойки можно разделить число k.

В итоге, определен способ получения индекса j, с помощью которого можно сформировать очередной код Грея.

Таким образом, предложен алгоритм получения последовательности кодов Грея без циклов. Основу алгоритма составляет выбор числа j, который осуществляется с помощью массива `f`, как обсуждалось выше.

Для удобства введем массив `a[0..n-1]`, где будет храниться текущий код Грея, а также уже рассмотренный массив `f[0..n]`. Приведем предложенный алгоритм генерации кодов Грея без циклов:

1. **Инициализация**. Устанавливаем `a[j] = 0` и `f[j] = j` для $0 \le j < n$; `f[n] = n`.
2. **Вывод кода Грея**. Выводим массив `a`.
3. **Выбор j**. Устанавливаем `j = f[0], f[0] = 0`. Если $j = n$, то алгоритм заканчивается, иначе устанавливаются `f[j] = f[j + 1]` и `f[j + 1] = j + 1`.
4. **Изменение кода Грея**. Устанавливается `a[j] = 1 – a[j]` и выполняется переход ко второму шагу.

Так как на каждом шаге выполняется константное число операций, то общее время работы алгоритма будет пропорционально 2^n.

3. Реализация визуализатора

План программирования визуализатора:

1. Представление структур данных.
2. Описание алгоритма и модели данных в формате *XML / Java.*
3. Разработка интерфейса.

Для описания кода Грея используется одномерный массив *a*. Он реализует представление текущего кода на каждом шаге алгоритма. Для генерации последующих кодов используется вспомогательный массив указателей *f*. С помощью этих двух массивов реализуется работа с кодами Грея в алгоритме.

На втором этапе модель данных и алгоритм описываются в формате *XML/Java*. Созданные модель данных и алгоритм приведены в приложении 1.

Разработка интерфейса определяет отображение шагов алгоритма на экране и описание соответствующих функций реализации шагов.

4. Описание модели данных

Класс, содержащий все необходимые алгоритму переменные и структуры данных, называется моделью данных. Для реализации алгоритма генерации кодов Грея без циклов требуется модель данных, содержащая:

1. Массив для кодов Грея *a*.
2. Вспомогательный массив для фокусных элементов *f*.
3. Переменная цикла *i*.
4. Экземпляр апплета *LooplessGrayVisualizer*.

5. Описание визуализатора как конечного автомата

В приложении 1 приведен текст программы, написанной в формате *XML / Java*. Из нее можно выделить модель данных, содержащую следующие конструкции:

1. Последовательности операторов.
2. Оператор ветвления (*if-then-else*).
3. Цикл с предусловием (*while*).

Такая реализация позволяет представить алгоритм в виде конечного автомата, как это сделано в работе [2]. На основе этого представления пакет *Vizi* [3] генерирует *java*-файл. Полученный таким образом файл `LoopleesGray.java` (приложение 2), представляет собой реализацию алгоритма генерации кодов Грея без циклов в виде двух конечных автоматов (прямного и обратного), каждый из которых содержит по восемь состояний.

6. Описание интерфейса визуализатора

На рис. 1 приведен скриншот визуализатора.

Рис. 1. Внешний вид аплета

Опишем аплет.

1. *Область визуализации*. В этой области изображен код Грея на текущем шаге визуализации.

2. *Область комментариев*. Здесь появляются комментарии к текущему шагу.

3. *Стандартная панель управления*. При помощи кнопок "<<" и ">>" в левой части можно переходить к следующему шагу алгоритма или возвращаться к предыдущему. Кнопка "`Рестарт`" позволяет начать все с самого начала. Кнопка "`Авто`" запускает визуализацию в автоматическом режиме. Пауза между шагами выставляется элементом управления "`Задержка`". Кнопка "?" выводит информацию об авторах и используемой технологии.

4. *Панель изменения размера кода и загрузки/сохранения визуализатора*. Здесь можно выбрать количество бит кода. При нажатии на кнопку "`Сохранить / Загрузить`" можно сохранить/загрузить текущий код Грея и состояние визуализатора.

7. Конфигурация визуализатора

Конфигурация визуализатора описывается в файле `LooplessGray-Configuration.xml` (приложение 1). Все параметры конфигурации указываются в теге `configuration`.

Таблица, приведенная ниже, содержит описание параметров визуализатора.

Название параметра	**Описание параметра**
comment-lines	Количество строк для комментариев
elements	Панель изменения размера кода Грея
array	Три возможные схемы отображения элементов визуализатора
SaveLoadDialog	Конфигурация окна сохранения/загрузки состояния
ArrayLengthComment	Комментарий для длины массива в сохраняемом файле
ElementsComment	Комментарий для элементов массива в сохраняемом файле
StepComment	Комментарий для номера текущего шага в сохраняемом файле

Заключение

В заключении отметим удобство использования технологии *Vizi* при разработке визуализаторов алгоритмов. Она заметно облегчает программирование визуализатора. Это достигается за счет формального описания логики алгоритма и скрытия многих технических деталей.

В основе технологии *Vizi* лежит автоматное программирование [2]. Именно представление алгоритма в виде взаимосвязанной системы конечных автоматов упрощает сам процесс построения логики и программирования визуализатора. Нет необходимости каждый раз придумывать новые оптимальные способы построения визуализации.

В будущем хотелось бы увидеть весь процесс создания визуализаторов более удобным. Это может быть некоторая графическая оболочка или некая автоматизированная среда, походящая на визуальную среду разработки программ. Это сделало бы технологию *Vizi* незаменимой и самой удобной базой для разработки программ, связанных с визуализированием различных алгоритмов и процессов.

Источники

1. *Knuth D.* The Art of Computer Programming. Generating All n-Tuples. *Addison-Wesley.* http://sunburn.stanford.edu/~knuth/fasc2a.ps.gz

2. *Казаков М.А., Корнеев Г.А., Шалыто А.А.* Разработка логики визуализаторов алгоритмов на основе конечных автоматов // Телекоммуникации и информатизация образования. 2003. № 6, с.27-58. http://is.ifmo.ru/works/vis/

3. *Vizi Home Page.* http://ctddev.ifmo.ru/vizi/

Приложение 1. *XML*-описание визуализатора

Файл LooplessGray.xml (основные параметры)

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

<!--
    "LooplessGray" visualizer description (example)
    Version: $Id: LooplessGray.xml,v 1.1 2004/02/24$
-->

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

<visualizer
    id="LooplessGray"
    package="ru.ifmo.vizi.loopless_gray"
    main-class="LooplessGrayVisualizer"

    preferred-width="400"
    preferred-height="250"

    name-ru="Генерация кодов Грея без циклов (пример)"
    name-en="Loopless Gray binary generation (example)"

    author-ru="Дмитрий Белешко"
    author-en="Dmitri Beleshko"
    author-email="beleshko@rain.ifmo.ru"

    supervisor-ru="Георгий Корнеев"
    supervisor-en="Georgy 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 LooplessGray-Algorithm.xml (описание алгоритма)

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

<!--
    "LooplessGray" algoritm description (example)
    Version: $Id: LooplessGray-Algorithm.xml,v 1.0 2004/08/09 13:00:04 geo Exp $
-->

<algorithm>
    <data>
        <variable description="Массив для кодов Грея">int a[] = new int[]{ 0, 0, 0,
                                                                            0};</variable>
        <variable description="Вспомогательный массив">int f[] = new int[]{ 0, 1, 2, 3,
                                                                            4};</variable>
        <variable description="Переменная цикла">int i;</variable>
        <variable description="Экземпляр апплета">LooplessGrayVisualizer
                                                                    visualizer;</variable>
```

```
        <toString>
            return new String();
        </toString>
    </data>

    <auto id="Main" description="Генерирует коды Грея">
        <start
            comment-ru="На экране изображен массив, который будет представлять собой текущий
                                                                                код Грея"
            comment-en="There is an array on the display, which will show the current Gray
                                                                                code"
        >
            <draw>
                d.visualizer.updateArray(0, 0);
            </draw>
        </start>
        <step
            id="Initialization"
            description="Инициализация"
            comment-ru="Инициализация. Получаем первый код Грея."
            comment-en="Initialization. We get the first Gray code."
        >
            <draw>
                d.visualizer.updateArray(0, 0);
            </draw>
            <action>
                for ( d.i = 0; d.i &lt; d.a.length; d.i++) {
                   d.a[d.i] = 0;
                   d.f[d.i] = d.i;
                }
                d.f[d.a.length] = d.a.length;
                d.i = 0;
            </action>
        </step>
        <while
            id="Loop"
            description="Цикл"
            test="d.i &lt; d.a.length"
            level="-1"
        >
            <if
                id="Cond"
                description="Условие"
                test="d.f[0] != d.a.length"
                true-comment-ru="В массиве есть невыделенные элементы."
                true-comment-en="There are non-selected elements in the array."
                false-comment-ru="В массиве больше нет невыделенных элементов. Завершение
                                                                                программы."
                false-comment-en="There aren't non-selected elements in the array.
                                                                Termination of the programm."
            >
                <draw>
                    d.visualizer.updateArray(d.a.length, 2);
                </draw>
                <then>
                    <step
                        id="SimpleStep"
                        description="Выбор элемента"
                        comment-ru="Выбираем невыделенный элемент, находящийся как можно
                 левее. Все выделенные элементы, левее выбранного, становятся невыделенными."
                        comment-en="Choose non-selected element, which has the most left
location. All selected elements, situated to the right of the chosen one, become unselected."
                    >
                        <draw>
                            d.visualizer.updateArray(d.i, 1);
                        </draw>
                        <action>
                            d.i @= d.f[0];
                            d.f[0] @= 0;
                        </action>
                    </step>
                    <step
                        id="ChangeF"
                        description="Выделение элемента."
```

```
                    comment-ru="Выделение и изменение выбранного элемента."
                    comment-en="Selection and changing of the chosen element."
                >
                    <draw>
                        d.visualizer.updateArray(d.i, 2);
                    </draw>
                    <action>
                        d.f[d.i] @= d.f[d.i + 1];
                      d.f[d.i + 1] @= d.i + 1;
                        d.a[d.i] @= 1 - d.a[d.i];
                    </action>
                </step>
                <step
                    id="DrawGrayCode"
                    description="Текущий код Грея."
                    comment-ru="Получаем текущий код Грея."
                    comment-en="We get the current Gray code."
                    level="1"
                >
                <draw>
                    d.visualizer.updateArray(0, 0);
                </draw>
                <action>
                </action>
            </step>
            </then>
            <else>
            <step
                    id="ElseStep"
                    description="Последний шаг алгоритма."
                    level="-1"
                >
                    <action>
                        d.i @= d.f[0];
                        d.f[0] @= 0;
                    </action>
                </step>
            </else>
        </if>
    </while>
    <finish
        comment-ru="Получен последний код Грея."
        comment-en="We get the last Gray code."
    >
        <draw>
            d.visualizer.updateArray(0, 0);
        </draw>
    </finish>
</auto>
</algorithm>
```

Файл LooplessGray-Configuration.xml (конфигурация)

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

<!--
    "LooplessGray" visualizer configuration (example).
    Shared for LooplessGray.xml
    Version: $Id: LooplessGray-Configuration.xml,v 1.2 2003/12/25 $
-->

<configuration>
    <property
        description = "Comment pane height"
        param       = "comment-height"
        value       = "40"
    />
    <spin-panel
        description = "Number of elements in the array"
        param       = "elements"
```

```
        caption-ru  = "Элементов: {0,number,####}"
        caption-en  = "Elements: {0,number,####}"
        hint-ru     = "Количество элементов в массиве"
        hint-en     = "Number of elements in the array"
        value       = "4"
        min-value   = "4"
        max-value   = "20"
        step        = "1"
    >
        <button
            param       = "button-less"
            caption-ru  = "&lt;&lt;"
            caption-en  = "&lt;&lt;"
            hint-ru     = "Уменьшить количество элементов"
            hint-en     = "Decrease number of elements"
        />
        <button
            param       = "button-more"
            caption-ru  = ">>"
            caption-en  = ">>"
            hint-ru     = "Увеличить количество элементов"
            hint-en     = "Increase number of elements"
        />
    </spin-panel>
    <styleset
        description = "Array style set"
        param       = "array"
    >
        <style
            description     = "Ordinary cell"
            text-color      = "000000"
            text-align      = "0.5"
            border-color    = "000000"
            border-status   = "true"
            fill-color      = "8080ff"
            fill-status     = "true"
            aspect-status   = "false"
            padding         = "0.2"
        >
            <font
                face        = "Serif"
                size        = "12"
                style       = "plain"
            />
        </style>
        <style
            description     = "Selected cell"
            fill-color      = "80ff80"
        />
        <style
            description     = "Changed cell"
            fill-color      = "ff8080"
        />
    </styleset>
</configuration>
```

Приложение 2. Сгенерированный код автомата

Файл LooplessGray.java

```
package ru.ifmo.vizi.loopless_gray;

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

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

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

    /**
      * Данные.
      */
    public final class Data {
        /**
          * Массив для кодов Грея.
          */
        public int a[] = new int[]{ 0, 0, 0, 0};

        /**
          * Вспомогательный массив.
          */
        public int f[] = new int[]{ 0, 1, 2, 3, 4};

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

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

        public String toString() {
            return new String();
        }
    }

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

        /**
          * Конечное состояние автомата.
          */
        private final int END_STATE = 9;

        /**
```

```
    * Описания состояний.
    */
  private final String[] descriptions = new String[]{"Начальное состояние",
"Инициализация", "Цикл", "Условие", "Условие (окончание)", "Выбор элемента", "Выделение
    элемента.", "Текущий код Грея.", "Последний шаг алгоритма.", "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
                  stack.pushBoolean(false);
                  state = 2; // Цикл
                  break;
              }
              case 2: { // Цикл
                  if (d.i < d.a.length) {
                      state = 3; // Условие
                  } else {
```

```
                state = END_STATE;
            }
            break;
        }
        case 3: { // Условие
            if (d.f[0] != d.a.length) {
                state = 5; // Выбор элемента
            } else {
                state = 8; // Последний шаг алгоритма.
            }
            break;
        }
        case 4: { // Условие (окончание)
            stack.pushBoolean(true);
            state = 2; // Цикл
            break;
        }
        case 5: { // Выбор элемента
            state = 6; // Выделение элемента.
            break;
        }
        case 6: { // Выделение элемента.
            state = 7; // Текущий код Грея.
            break;
        }
        case 7: { // Текущий код Грея.
            stack.pushBoolean(true);
            state = 4; // Условие (окончание)
            break;
        }
        case 8: { // Последний шаг алгоритма.
            stack.pushBoolean(false);
            state = 4; // Условие (окончание)
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // Инициализация
            startSection();
                            for ( d.i = 0; d.i < d.a.length; d.i++) {
                             d.a[d.i] = 0;
                             d.f[d.i] = d.i;
                        }
                        d.f[d.a.length] = d.a.length;
                           d.i = 0;
            break;
        }
        case 2: { // Цикл
            break;
        }
        case 3: { // Условие
            break;
        }
        case 4: { // Условие (окончание)
            break;
        }
        case 5: { // Выбор элемента
            startSection();
            storeField(d, "i");
            d.i = d.f[0];
            storeArray(d.f, 0);
            d.f[0] = 0;
            break;
        }
        case 6: { // Выделение элемента.
            startSection();
                                        storeArray(d.f, d.i);
                                        d.f[d.i] = d.f[d.i + 1];
            storeArray(d.f, d.i + 1);
                                    d.f[d.i + 1] = d.i + 1;
                                        storeArray(d.a, d.i);
                                        d.a[d.i] = 1 - d.a[d.i];
```

```
                break;
            }
            case 7: { // Текущий код Грея.
                startSection();
                break;
            }
            case 8: { // Последний шаг алгоритма.
                startSection();
                storeField(d, "i");
                d.i = d.f[0];
                storeArray(d.f, 0);
                d.f[0] = 0;
                break;
            }
        }
    } while (!isInteresting(level));
}

/**
  * Сделать шаг в назад.
  */
public void stepBackward(int level) {
    do {
        // Обращение действия в текущем состоянии
        switch (state) {
            case 1: { // Инициализация
                restoreSection();
                break;
            }
            case 2: { // Цикл
                break;
            }
            case 3: { // Условие
                break;
            }
            case 4: { // Условие (окончание)
                break;
            }
            case 5: { // Выбор элемента
                restoreSection();
                break;
            }
            case 6: { // Выделение элемента.
                restoreSection();
                break;
            }
            case 7: { // Текущий код Грея.
                restoreSection();
                break;
            }
            case 8: { // Последний шаг алгоритма.
                restoreSection();
                break;
            }
        }

        // Переход в предыдущее состояние
        switch (state) {
            case 1: { // Инициализация
                state = START_STATE;
                break;
            }
            case 2: { // Цикл
                if (stack.popBoolean()) {
                    state = 4; // Условие (окончание)
                } else {
                    state = 1; // Инициализация
                }
                break;
            }
            case 3: { // Условие
                state = 2; // Цикл
                break;
            }
```

```
            case 4: { // Условие (окончание)
                if (stack.popBoolean()) {
                    state = 7; // Текущий код Грея.
                } else {
                    state = 8; // Последний шаг алгоритма.
                }
                break;
            }
            case 5: { // Выбор элемента
                state = 3; // Условие
                break;
            }
            case 6: { // Выделение элемента.
                state = 5; // Выбор элемента
                break;
            }
            case 7: { // Текущий код Грея.
                state = 6; // Выделение элемента.
                break;
            }
            case 8: { // Последний шаг алгоритма.
                state = 3; // Условие
                break;
            }
            case END_STATE: { // Начальное состояние
                state = 2; // 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: // Цикл
            return level <= -1;
        case 3: // Условие
            return level <= 0;
        case 4: // Условие (окончание)
            return level <= -1;
        case 5: // Выбор элемента
            return level <= 0;
        case 6: // Выделение элемента.
            return level <= 0;
        case 7: // Текущий код Грея.
            return level <= 1;
        case 8: // Последний шаг алгоритма.
            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 = LooplessGray.this.getComment("Main.START_STATE");
            break;
        }
        case 1: { // Инициализация
            comment = LooplessGray.this.getComment("Main.Initialization");
            break;
        }
        case 3: { // Условие
            if (d.f[0] != d.a.length) {
                comment = LooplessGray.this.getComment("Main.Cond.true");
            } else {
                comment = LooplessGray.this.getComment("Main.Cond.false");
            }
            break;
        }
        case 5: { // Выбор элемента
            comment = LooplessGray.this.getComment("Main.SimpleStep");
            break;
        }
        case 6: { // Выделение элемента.
            comment = LooplessGray.this.getComment("Main.ChangeF");
            break;
        }
        case 7: { // Текущий код Грея.
            comment = LooplessGray.this.getComment("Main.DrawGrayCode");
            break;
        }
        case END_STATE: { // Конечное состояние
            comment = LooplessGray.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.updateArray(0, 0);
            break;
        }
        case 1: { // Инициализация
            d.visualizer.updateArray(0, 0);
            break;
        }
        case 3: { // Условие
            d.visualizer.updateArray(d.a.length, 2);
            break;
        }
        case 5: { // Выбор элемента
            d.visualizer.updateArray(d.i, 1);
            break;
        }
        case 6: { // Выделение элемента.
            d.visualizer.updateArray(d.i, 2);
            break;
        }
        case 7: { // Текущий код Грея.
            d.visualizer.updateArray(0, 0);
            break;
        }
        case END_STATE: { // Конечное состояние
            d.visualizer.updateArray(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. Исходные коды интерфейса визуализатора

Файл LooplessGrayVisualizer.java

```
package ru.ifmo.vizi.loopless_gray;

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;

/**
 * LooplessGray applet.
 *
 * @author  Beleshko Dmitri
 * @version $Id: LooplessGrayVisualizer.java,v 1.0 $
 */
public final class LooplessGrayVisualizer extends Base {
    /**
     * LooplessGray automata instance.
     */
    private final LooplessGray auto;

    /**
     * LooplessGray automata data.
     */
    private final LooplessGray.Data data;

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

    /**
     * Number of elements in array.
     */
    private final SpinPanel elements;
    /**
     * Array shape style set.
     */
    private final ShapeStyle[] styleSet;
    /**
     * Array shape style for all cells.
     */
    private int cellStyle[] = new int[] {0, 0, 0, 0 };
    /**
     * Creates a new LooplessGray visualizer.
     *
     * @param parameters visualizer parameters.
     */
    public LooplessGrayVisualizer(VisualizerParameters parameters) {
        super(parameters);
        auto = new LooplessGray(locale);
        data = auto.d;
        data.visualizer = this;
        cells = new Stack();

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

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

    setArraySize(elements.getIntValue());

    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.CENTER);

    Panel bottomPanel = new Panel();
    bottomPanel.add(elements);
    panel.add(bottomPanel, BorderLayout.SOUTH);

    return panel;
}

/**
 * Adjusts array size to match current model size.
 */
private void adjustArraySize() {
    int size = auto.d.a.length;

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

/**
 * Sets new array size.
 *
 * @param size new array size.
 */
private void setArraySize(int size) {
    auto.d.a = new int[size];
    auto.d.f = new int[size + 1];
    cellStyle = new int[size];
    resetArray();
    adjustArraySize();
}

/**
 * Reset array values.
 *
 */
private void resetArray() {
    for (int i = 0; i < data.a.length; i++) {
        data.a[i] = 0;
    }
    rewind(0);
}
```

```
    /**
     * Rewinds algorithm to the specified step.
     *
     * @param step spet of the algorith to rewind to.
     */
    private void rewind(int step) {
        adjustArraySize();
        auto.toStart();
        updateArray(0, 0);
        while (!auto.isAtEnd() && auto.getStep() < step) {
            auto.stepForward(0);
        }
    }

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

        color = 0;
        for (i = 0; i < data.a.length; i++) {
            if ( i == activeCell) {
                cellStyle[i] = activeStyle;
            }
            else {
                cellStyle[i] = color;
            }
        }

        if (activeStyle > 0){
        for (i = 0; i < data.a.length; i++) {
            for (j = data.f[i] - 1; j >= i; j--) {
                cellStyle[j] = 2;
            }
        }
        }

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

    /**
     * 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, (clientHeight) * 10 / 13);
        int y = (clientHeight - height) / 2 ;
        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();
        }
    }

}
```