Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра “Компьютерные технологии”

А.В. Смаль

Построение визуализатора алгоритма Винограда вычисления коротких сверток с использованием технологии *Vizi*

Санкт-Петербург
2005

Оглавление

Введение

Теории приходят и уходят, а примеры остаются.

И.М. Гельфант

Современное развитие мультимедийных средств требует все более быстрой и качественной обработки сигналов. Один из методов, который используется в этой области, преобразования Фурье. Существует много разновидностей этого преобразования, одна из них — быстрое преобразование Фурье (БПФ).

Алгоритм, рассматриваемый в настоящей работе, является частью алгоритма Винограда вычисления БПФ, который является одним из лучших алгоритмов в этой области. Тем не менее алгоритм Винограда вычисления коротких сверток сам по себе представляет большой интерес, так как показывает, что при переходе с одного поля на другое есть возможность сократить количество операций.

Схема создания визуализатора на основе технологии *Vizi*

На рисунке 1 представлена схема создания визуализатора на основе технологии *Vizi*.

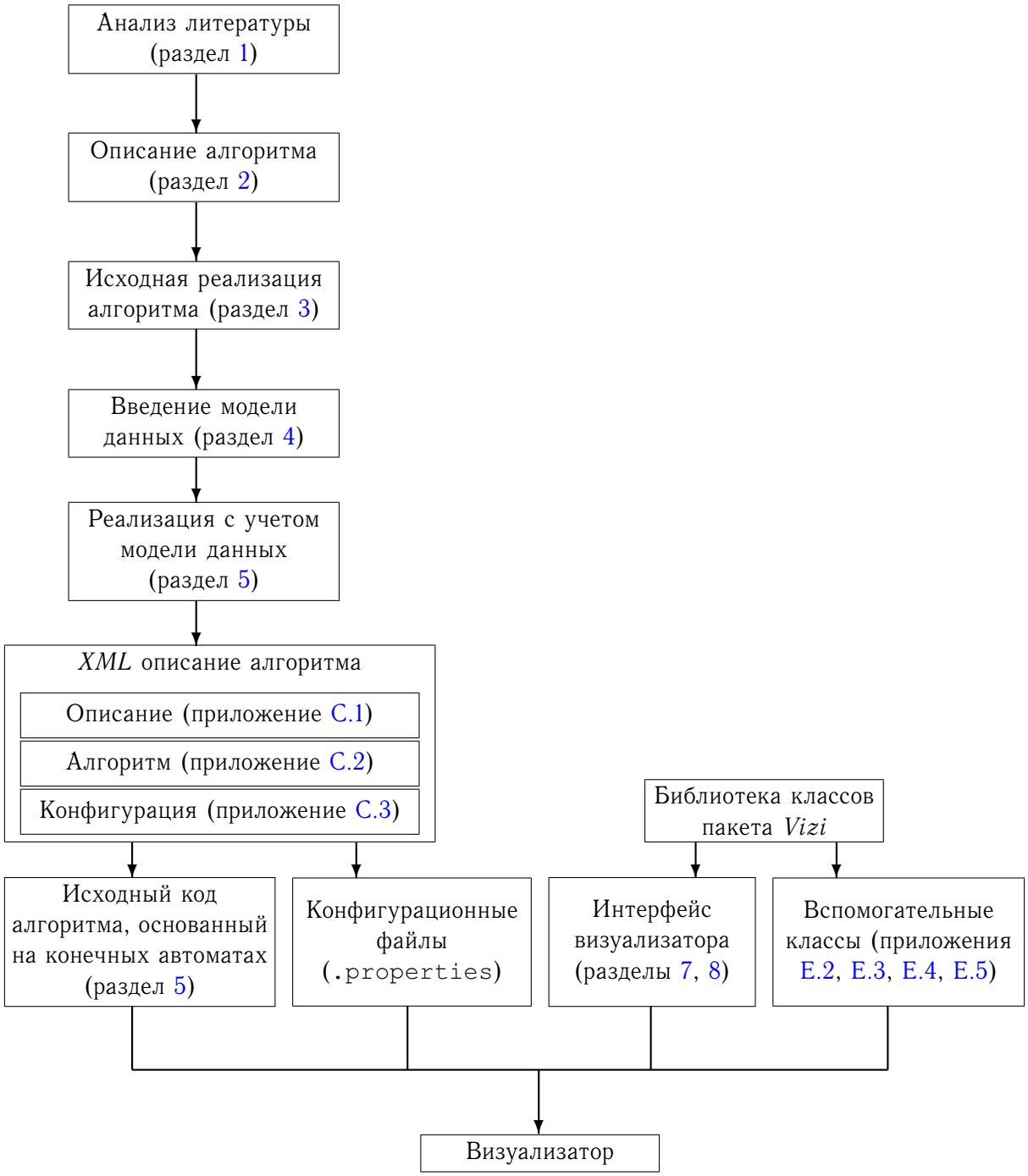

Рис. 1. Схема создания визуализатора на основе технологии *Vizi*

1. Анализ литературы

Автором был выполнен анализ описания алгоритма Винограда по трем изданиям.

1. **Блейхут Р.** Быстрые алгоритмы цифровой обработки сигналов. М.: Мир, 1989. –448 с.

   Данная книга дает наиболее исчерпывающую информацию по рассматриваемому вопросу. Именно на основе этого описания был реализован данный визуализатор.

2. **Гольденберг Л.М., Матюшкин Б.Д., Поляк М.Н.** Цифровая обработка сигналов. М.: Радио и связь, 1985. –312 с.

   Данная книга — справочник. Алгоритм Винограда вычисления коротких сверток описан достаточно понятно.

3. **Корман М.В.** О быстром преобразовании Фурье и некоторых его применениях. Дипломная работа. СПбГУ, 1997. –80 с.

   Данная работа представляет сжатое, конспективное описание рассматриваемого алгоритма и является трудной для понимания.

2. Описание алгоритма

При вычислении БПФ возникает потребность в вычислении произведения небольших полиномов — коротких сверток. Основная идея этого алгоритма — переход с поля многочленов любой степени P^{∞} на поле многочленов P^n степени не больше n, где перемножение многочленов проводится по некоторому модулю.

Предположим, что требуется вычислить линейную свертку

$$s(x) = a(x)b(x).$$

Эту задачу можно решить через другую:

$$s(x) = a(x)b(x) \pmod{m(x)},$$

где $m(x)$ — фиксированный многочлен степени n, такой, что

$$\deg a(x), \deg b(x) < n.$$

Если выбрать $m(x)$ таким, что

$$\deg m(x) > \deg s(x) = \deg a(x) + \deg b(x),$$

то выигрыша получить не удается.

Намного интереснее рассмотреть случай $\deg m(x) = \deg a(x) + \deg b(x)$[1]. Эту задачу можно заменить некоторым множеством задач с меньшим общим числом вычислений[2]. Для этого разложим $m(x)$ на простые над некоторым полем P многочлены:

$$m(x) = m^{(1)}(x)m^{(2)}(x)\dots m^{(n)}(x). \tag{1}$$

Далее вычисляются полиномы $a^{(1)}(x)$, $a^{(2)}(x), \dots,$ $a^{(n)}(x)$ и $b^{(1)}(x), b^{(2)}(x), \dots, b^{(n)}(x)$:

$$a^{(i)}(x) = a(x) \pmod{m^{(i)}(x)}, \qquad b^{(i)}(x) = b(x) \pmod{m^{(i)}(x)}. \tag{2}$$

Введем вспомогательные полиномы $s^{(k)}(x)$:

$$s^{(k)}(x) = a^{(k)}(x)b^{(k)}(x). \tag{3}$$

По китайской теореме об остатках [1, с. 77]

$$s(x) \equiv s^{(1)}(x)c^{(1)}(x) + s^{(2)}(x)c^{(2)}(x) + \dots + s^{(n)}(x)c^{(n)}(x) \pmod{m(x)}, \tag{4}$$

где $c^{(k)}(x)$ могут быть найдены из решений Диофантовых уравнений:

$$m^{(k)}(x)n^{(k)}(x) + \frac{m(x)}{m^{(k)}(x)}N^{(k)}(x) = 1, \quad \text{тогда} \quad c^{(k)}(x) = \frac{m(x)}{m^{(k)}(x)}N^{(k)}(x). \tag{5}$$

Так как $\deg m(x) = \deg a(x) + \deg b(x)$, то

$$s(x) = (s(x) \bmod m(x)) + m(x)\frac{s_n}{m_n},$$

где s_n и m_n — старшие коэффициенты полиномов $s(x)$ и $m(x)$ соответственно. Коэффициент s_n можно вычислить следующим образом:

$$s_n = a_p b_q, \tag{6}$$

где a_p и b_q — старшие коэффициенты полиномов $a(x)$ и $b(x)$ соответственно.

Теперь посмотрим, где происходят вещественные умножения. Будем считать, что необходимо вычислить много однотипных сверток (именно такие задачи возникают

[1] В [1,2] рассмотрен так же случай $\deg m(x) < \deg a(x) + \deg b(x)$.

[2] Здесь и далее под количеством вычислений будем понимать количество вещественных умножений — умножение вещественного числа на вещественное. Умножение вещественного числа на малое целое учитываться не будет.

при вычислении БПФ). Выбрать полином $m(x)$ (соотношение (1)), а точнее $m^{(k)}(x)$, по которым вычисляется $m(x)$, необходимо только один раз.

При вычислении $a^{(k)}(x)$ и $b^{(k)}(x)$ по соотношению (2) происходят умножения только на малые целые числа — коэффициенты $m(x)$. При вычислении $s^{(k)}(x)$ происходят умножения двух вещественных полиномов, следовательно здесь потребуются вещественные умножения. Вычислить $c(x)$ по соотношению (5) необходимо только один раз (заметим, что $c(x)$ — полином с целыми коэффициентами). При вычислении $s(x) \bmod m(x)$ по соотношению (4) происходят умножения только на малые целые числа. Еще одно вещественное умножение кроется в вычислении старшего коэффициента $s(x)$ по соотношению (6).

Можно еще упростить вычисления — преобразовать все вычисления к матричной форме:

$$s = PAQB,$$

где s — вектор из коэффициентов полинома $s(x)$, матрицы A и B содержат линейные выражения из коэффициентов $a(x)$ и $b(x)$ соответственно, а матрицы P и Q состоят из целых чисел и производят все необходимые преобразования. Подробнее об этом в работе [1, с.90].

3. Реализация визуализируемого алгоритма

Для эффективной реализации визуализируемого алгоритма используется отдельный класс `dVector` (приложение E.2), который инкапсулирует все основные операции с полиномами (сложение, умножение на полином, умножение на число, вычисление остатка по модулю), а также решает Диофантовы уравнения. Эти операции не будут визуализироваться, поэтому их удобно вынести в отдельный класс. Сам алгоритм записывается достаточно кратко (приложение A) и по структуре повторяет описание алгоритма.

4. Описание модели данных

В модель данных (класс `Data` приложения B) выносятся все переменные, которые понадобятся для визуализации. Во-первых, это начальные данные и результат: полиномы $a(x)$ и $b(x)$, количество взаимно простых делителей у $m(x)$, полиномы

$m^{(k)}(x)$ (сам полином $m(x)$ может быть посчитан, как произведение $m^{(k)}(x)$), полином результата $s(x)$ и количество вещественных умножений.

Во-вторых, все промежуточные данные, которые необходимо визуализировать: полиномы $a^{(k)}(x)$, $b^{(k)}(x)$, $s^{(k)}(x)$, $c^{(k)}(x)$, $cs^{(k)}(x) = c^{(k)}(x)s^{(k)}(x) \bmod m^{(k)}(x)$.

Все остальные переменные будут определяться локально, так как они непосредственно не участвуют в визуализации (например, переменная `M` типа `dVector` (приложение A строка 61)).

5. Приведение программы к виду удобному для преобразования в автомат

Первое, что необходимо сделать — разбить на шаги визуализации (в приложении B это сделано при помощи комментариев). Все, что оказывается внутри шага, будет просчитываться за один шаг.

Циклы, которые оказались вне шагов, необходимо преобразовать в `while`. Если в цикле определяется переменная цикла (например, цикл `for`), то ее требуется вынести в модель данных (заметим, что для нескольких невложенных циклов достаточно вынести одну переменную цикла). Циклы, которые целиком содержатся внутри некоторого шага, нет необходимости преобразовывать (к примеру, цикл `for` на строке 56 приложения A остался без изменений в приложении B).

После этих операций можно переходить к созданию *XML*-описания алгоритма.

6. Создание описания алгоритма на *XML*

Описание алгоритма на *XML* состоит из трех частей, каждая из которых выделена в отдельный файл.

Первый из них содержит некоторые данные о визуализаторе и его создателе (приложение C.1 файл `Vinograd.xml`). Некоторые параметры доступны в двух формах: с постфиксом “-en” и с постфиксом “-ru” — это значения параметров для англо- и русскоязычной версий (к примеру, `author-en` и `author-ru`). Значения всех параметров соответствуют их названиям.

Второй файл содержит описание модифицированной реализации алгоритма. Это описание получается в результате преобразования модифицированной реализации

алгоритма из приложения В в *XML*, соблюдая следующие правила.

- Реализация должна находится внутри тега `algorithm`.
- Элементы модели данных преобразуются в теги `variable`.
- Добавляется тег `toString`, который преобразуется в метод `toString` класса `Vinograd`, возвращающий информацию об этом классе.
- Внутри тега `auto` записывается алгоритм.
- В начале алгоритма размещается тег `start`, содержащий комментарии, которые высвечиваются при старте алгоритма.
- В конце алгоритма размещается тег `finish`, содержащий комментарии, которые высвечиваются при окончании работы алгоритма.
- Циклы, которые находятся вне шагов, преобразуются в тег `while`. Этот тег имеет следующие параметры:

  `description` — описание того, что происходит в этом цикле;

  `test` — условие, которое проверяется на каждой итерации цикла;

  `comment-ru, comment-en` — русские и английские комментарии, которые будут отображаться в начале каждой итерации;

  `comment-args` — перечисленные через запятую параметры, которые будут подставляться в комментарии вместо {(номер параметра)};

  `level` — если этот параметр меньше нуля, то итерации визуализироваться не будут — произойдет переход к первому шагу внутри цикла.
- Шаги преобразуются в тег `step`. Тег имеет следующие параметры:

  `description` — описание того, что происходит на этом шаге;

  `comment-ru, comment-en` — русские и английские комментарии, которые будут высвечиваться на этом шаге;

  `comment-args` — перечисленные через запятую параметры, которые будут подставляться в комментарии вместо {(номер параметра)};

  `level` — если этот параметр меньше нуля, то этот шаг визуализироваться не будет — произойдет переход к следующему шагу.
- Внутри тегов `while, step, start, finish` может быть тег `draw`, внутри которого можно обращаться к интерфейсу. Этот тег требуется для того, чтобы отображать изменения визуализатора на новом шаге.
- Внутри тегов `step, start` могут быть теги `direct` и `reverse`. В этих тегах описываются действия, которые совершаются при выполнении этого шага и при выполнении этого шага в обратном направлении соответственно.

- Вместо тегов `direct` и `reverse` может быть тег `action`. Такая замена возможна, если на этом шаге выполняются только присваивания. Тогда все символы '=' требуется заменить на '`@=`'. При этом все сокращенные операторы типа '+=', '-=' и т. д. нужно преобразовать в обычную форму.
- Все символы, которые предусматриваются стандартом *XML*, преобразовываются в специальную форму. Например, символ '>' преобразовывается в '`>`'.
- Обращение к экземпляру класса `Data` заменяется на `@`.

Третий файл содержит конфигурацию визуализатора — описание всех параметров, которые можно изменять у интерфейса (размеры, литеральные строки, начальные значение, цвета, шрифты и т. д.). Этот файл описан в разделе 9.

7. Описание интерфейса визуализатора

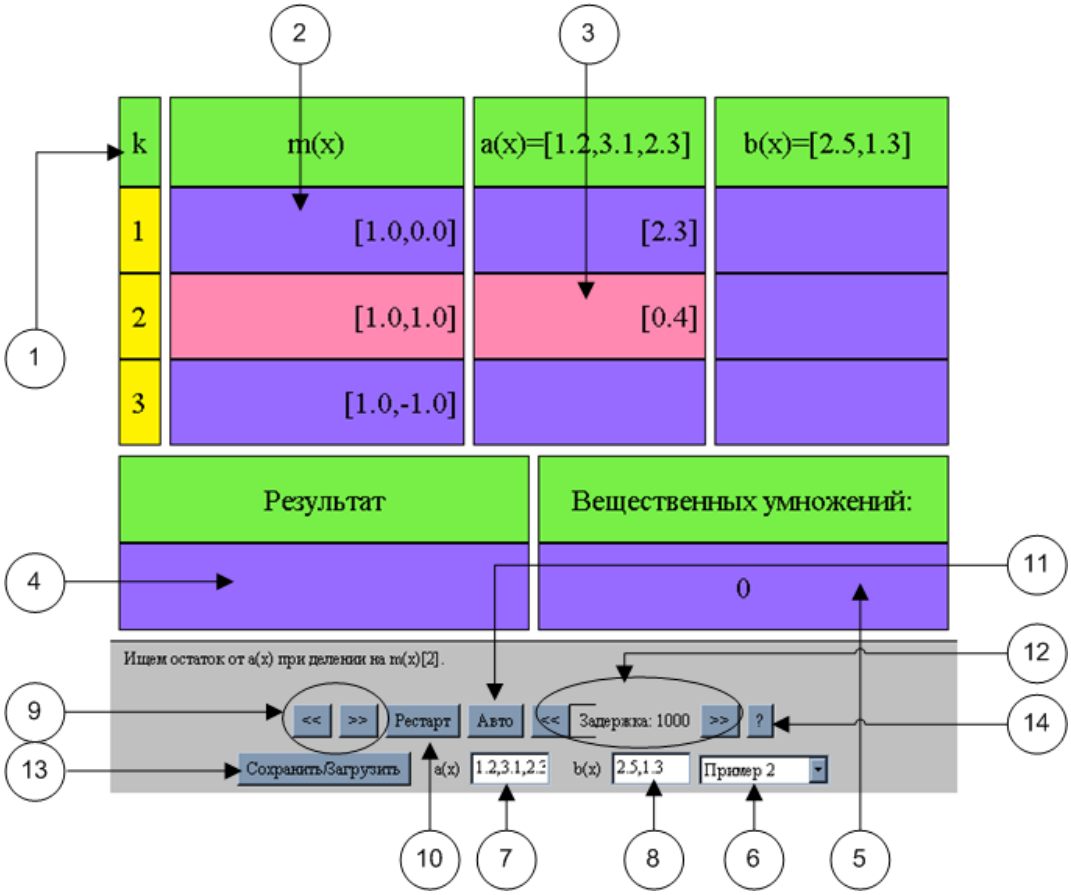

Рис. 2. Вид визуализатора.

На рисунке 2 представлен интерфейс визуализатора. Интерфейс состоит из двух частей. Первая (верхняя) используется для визуализуализации алгоритма. Слева находится колонка с номерами полиномов 1. Остальные колонки используются для

хранения полиномов. Полином представляется в виде набора коэффициентов от старшего к младшему 2. Полиномы, участвующие в вычислениях на данном шаге, подсвечиваются 3. В ячейку в левом нижнем 4 углу будет выведен результат. Ячейка в правом нижнем углу предназначена для отображения количества вещественных умножений 5.

Нижняя часть визуализатора содержит элементы управления. В списке 6 можно выбрать один из пяти предустановленных примеров. Есть возможность самому ввести пример. Для этого нужно ввести исходные полиномы $a(x)$ и $b(x)$ в окна 7 и 8 соответственно (степени не должны превышать трех). Кнопки 9 позволяют продвигаться на шаг вперед или назад в алгоритме. Кнопка 10 возвращает визуализатор в исходное состояние. Кнопка 11 запускает визуализацию в автоматическом режиме. Кнопки 12 позволяют выбрать задержку между шагами в автоматическом режиме. Кнопка 13 открывает окно сохранения/загрузки состояния визуализатора. Кнопка 14 высвечивает информацию о визуализаторе.

8. Работа с визуализатором

Для того, чтобы начать работу с визуализатором, нужно ввести начальные данные. При помощи списка 6 можно выбрать один из пяти предустановленных примеров. Можно ввести начальные данные вручную: в окошки 7 и 8 нужно ввести два набора чисел, коэффициентов вектора, разделенных запятыми (к примеру, ‘2.3,-1.4,0.4’ и ‘1.4,-3,0’). Если формат данных окажется некорректным (к примеру, “1..4,-3,0’ — две точки подряд), то будет выведено сообщение об ошибке, а данные преобразованы в ‘0.0’. При изменении начальных данных визуализатор автоматически перейдет в начальное состояние.

При помощи кнопок 9 можно продвигаться на шаг вперед и на шаг назад в алгоритме. При помощи кнопки 10 можно вернуть визуализатор в исходное состояние. Можно запустить автоматический режим (визуализатор будет сам продвигаться вперед, делая на каждом шаге задержку). Время задержки можно регулировать при помощи кнопок 12.

При желании сохранить состояние визуализатора нужно воспользоваться кнопкой 13. В появившемся окне конфигурация визуализатора (рис. 3) записана в несложном формате: все числа и идентификаторы разделены пробелами. Текст заключенный в ‘/*’, ‘*/’ — комментарии. Кнопка 1 сохраняет в файл, кнопка 2 — загружает

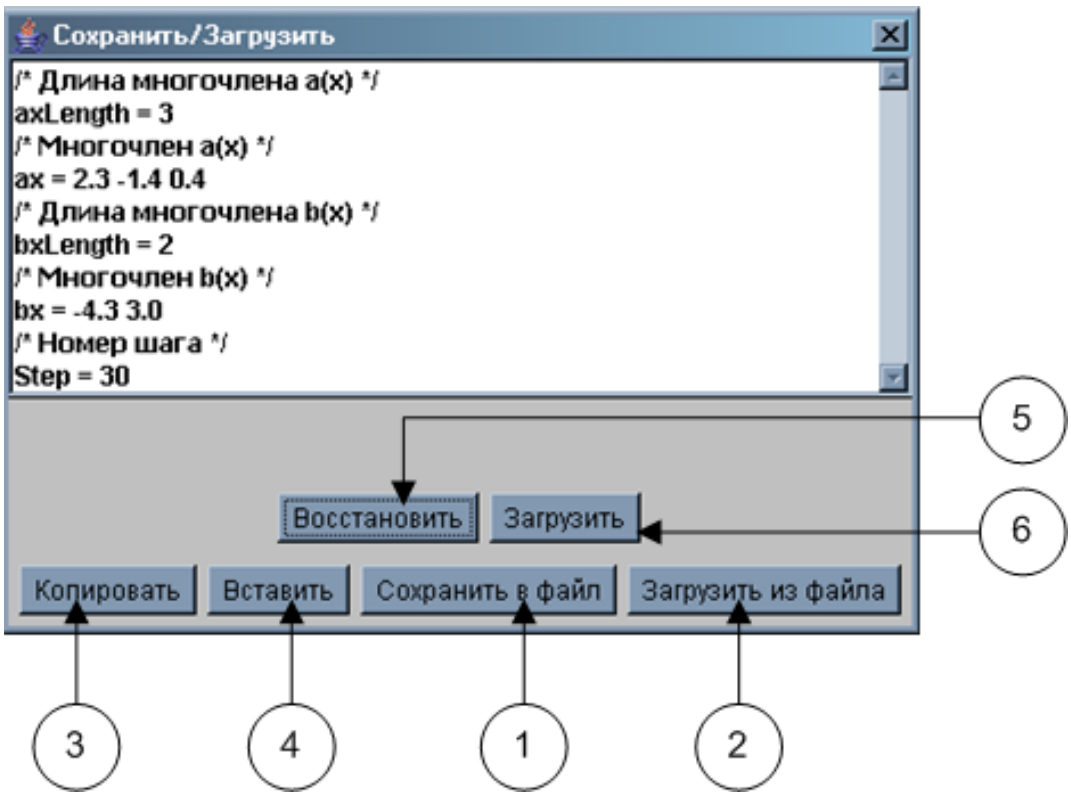

Рис. 3. Вид окна сохранения состояния/загрузки состояния визуализатора.

из файла. При помощи кнопки 3 можно скопировать в буфер обмена, а при помощи кнопки 4 — вставить из буфера обмена. Кнопка 5 восстанавливает текст текущей конфигурации в окне. Кнопка 6 устанавливает конфигурацию из окна в визуализатор.

9. Описание конфигурации визуализатора

В файле `Vinograd-Configuration.xml` (C.3) представлена конфигурация визуализатора. В нем описываются все элементы интерфейса, литеральные строки, цвета, размеры и другое.

- Все теги находятся внутри тега `configuration`.
- Все теги имеют общие параметры:

  `description` — описание параметра;

  `param` — имя параметра (через него значение параметра будет доступно в коде).
- Параметры можно группировать при помощи тега `group` (тогда к имени параметра через дефис добавляется имя группы).
- Тег `message` позволяет задавать строку. Имеет параметры `message-ru` и

`message-en` — значение строки на русском и на английском языках соответственно.

- Тег `property` позволяет задавать число. Имеет параметр `value` — значение (не зависит от языка).
- Тег `styleset` позволяет задавать множество стилей, которые задаются тегами `style` (описание параметров см. документация к *Vizi*). Тег `style` может содержать в себе тег `font`, определяющий шрифт. Необходимо заметить, что параметры наследуются в пределах `styleset`: значения параметров следующего стиля по умолчанию равны значениям предыдущего.

Заключение

Данный визуализатор может быть использован в учебных целях, как наглядное пособие для изучении теории преобразований Фурье. Подробное описание процесса создания визуализатора в данной документации позволяет использовать ее для написания подобных визуализаторов с использованием технологии *Vizi*.

Литература

1. **Блейхут Р.** Быстрые алгоритмы цифровой обработки сигналов. М.: Мир, 1989. 448 с.

2. **Гольденберг Л.М., Матюшкин Б.Д., Поляк М.Н.** Цифровая обработка сигналов. М.: Радио и связь, 1985. 312 с.

3. **Корман М.В.** О быстром преобразовании Фурье и некоторых его применениях. Дипломная работа. СПбГУ, 1997. 80 с.

A. Исходный текст реализации алгоритма Винограда

```
1  package ru.ifmo.vizi.vinograd;
2
3  import ru.ifmo.vizi.vinograd.dVector;
4
5  public class Vinograd {
6      public void run() {
7          //Столбец полинома a(x)
8          dVector[] ax;
9          //Столбец полинома b(x)
10         dVector[] bx;
11         //Столбец полинома m(x)
12         dVector[] mx;
13         //Столбец полинома s(x)
14         dVector[] sx;
15         //Столбец коэффициентов c(x)
16         dVector[] cx;
17         //Столбец полинома разложения свертки
18         dVector[] csx;
19         //Вектор результата
20         dVector rez;
21         //Вектор a(x)
22         dVector a;
23         //Вектор b(x)
24         dVector b;
25         //Число вещественных умножений
26         int multCount = 0;
27         //Количество взаимнопростых делителей у m(x)
28         int k;
29         //Инициализация
30         k = 3;
31         ax = new dVector[k];
32         bx = new dVector[k];
33         cx = new dVector[k];
34         mx = new dVector[k];
35         sx = new dVector[k];
36         csx = new dVector[k];
37         a = new dVector("1.2,2.3");
38         b = new dVector("2.5,-5.4,1.3");
39         mx[0] = new dVector("1.0,0.0");
40         mx[2] = new dVector("1.0,1.0");
41         mx[1] = new dVector("1.0,-1.0");
42         //Расчет остатков при делении a(x) на m(x)[i]
43         for (int j = 0; j < k ; j++) {
44             ax[j] = a.mod(mx[j]);
45         }
46         //Расчет остатков при делении b(x) на m(x)[i]
47         for (int j = 0; j < k ; j++) {
48             bx[j] = b.mod(mx[j]);
49         }
50         //Расчет s(x)[i] = a(x)[i]*b(x)[i]
51         for (int j = 0; j < k ; j++) {
52             sx[j] = ax[j].multVec(bx[j]).mod(mx[j]);
53             multCount = multCount + ax[j].size() * bx[j].size();
54         }
55         //Считаем все c(x)[i]
56         for (int i = 0; i < k; i++) {
57             //Считаем m(x)/m(x)[i] = M(x)
58                     dVector M = new dVector("1.0");
59             for (int j = 0; j < k; j++) {
60                 if (j == i) continue;
61                 M = M.multVec(mx[j]);
62             }
63             //Решаем диофантово уравнение и решение умножаем на M
64             cx[i] = mx[i].getSolve(M).multVec(M);
65         }
66         //Расчет c(x)[i]*s(x)[i]
67         for (int i = 0; i < k; i++) {
68             csx[j] = sx[j].multVec(cx[j]);
69         }
70         //Получаем результат
71         dVector rez = new dVector("0.0");
72         multCount = multCount + 1;
73         dVector m = new dVector("1.0");
74
```

```
75          for (int i = 0; i < k; i++) {
76              rez = rez.addVec(csx[i]);
77              m = m.multVec(mx[i]);
78          }
79          m = m.multNum(a.getCoef(a.size() - 1) * b.getCoef(b.size() - 1));
80          rez = rez.mod(m).addVec(m);
81          //Конец
82      }
83  }
```

B. Преобразованный исходный текст реализации алгоритма с использованием выбранной модели данных

```
1  package ru.ifmo.vizi.vinograd;
2
3  import ru.ifmo.vizi.vinograd.dVector;
4
5  class Data {
6      //Экземпляр апплета
7      public VinogradVisualizer applet = null;
8      //Столбец полинома a(x)
9      public dVector[] ax = null;
10     //Столбец полинома b(x)
11     public dVector[] bx = null;
12     //Столбец полинома m(x)
13     public dVector[] mx = null;
14     //Столбец полинома s(x)
15     public dVector[] sx = null;
16     //Столбец коэффициентов c(x)
17     public dVector[] cx = null;
18     //Столбец полинома разложения свертки
19     public dVector[] csx = null;
20     //Вектор результата
21     public dVector rez = null;
22     //Вектор a(x)
23     public dVector a = null;
24     //Вектор b(x)
25     public dVector b = null;
26     //Количество взаимнопростых делителей у m(x)
27     public int k = 4;
28     //Локальный индекс
29     public int j = 0;
30     //Число вещественных умножений
31     public int multCount = 0;
32 }
33
34
35 public class Vinograd {
36     public void run() {
37
38 //-------------------------- Шаг --------------------------------
39         //Инициализация
40         Data d = new Data();
41         d.ax = new dVector[d.k];
42         d.bx = new dVector[d.k];
43         d.cx = new dVector[d.k];
44         d.mx = new dVector[d.k];
45         d.sx = new dVector[d.k];
46         d.csx = new dVector[d.k];
47
48         d.k = 3;
49         d.a = new dVector("1.2,2.3");
50                 d.b = new dVector("2.5,-5.4,1.3");
51         d.mx[0] = new dVector("1.0,0.0");
52         d.mx[2] = new dVector("1.0,1.0");
53         d.mx[1] = new dVector("1.0,-1.0");
54 //---------------------- Конец шага ----------------------------
55
56         //Расчет остатков при делении a(x) на m(x)[i]
57         d.j = 0;
58         while (d.j < d.k) {
59
60 //-------------------------- Шаг --------------------------------
61             d.ax[d.j] = d.a.mod(d.mx[d.j]);
62             d.j = d.j + 1;
63 //---------------------- Конец шага ----------------------------
64
65         }
66
67         //Расчет остатков при делении b(x) на m(x)[i]
68         d.j = 0;
69         while (d.j < d.k) {
```

```
70
71 //--------------------------- Шаг --------------------------------
72             d.bx[d.j] = d.b.mod(d.mx[d.j]);
73             d.j = d.j + 1;
74 //----------------------- Конец шага ----------------------------
75
76         }
77         //Расчет s(x)[i] = a(x)[i]*b(x)[i]
78         d.j = 0;
79         while (d.j < d.k) {
80
81 //--------------------------- Шаг --------------------------------
82             d.sx[d.j] = d.ax[d.j].multVec(d.bx[d.j]).mod(d.mx[d.j]);
83             d.multCount = d.multCount + d.ax[d.j].size() * d.bx[d.j].size();
84             d.j = d.j + 1;
85 //----------------------- Конец шага ----------------------------
86
87         }
88
89 //--------------------------- Шаг --------------------------------
90         //Считаем все c(x)[i]
91         d.j = 0;
92         for (int i = 0; i < d.k; i++) {
93             dVector M = new dVector("1.0");
94             for (int j = 0; j < d.k; j++) {
95                 if (j == i) continue;
96                 M = M.multVec(d.mx[j]);
97             }
98             d.cx[i] = d.mx[i].getSolve(M).multVec(M);
99         }
100 //----------------------- Конец шага ----------------------------
101
102         //Расчет c(x)[i]*s(x)[i]
103         d.j = 0;
104         while (d.j < d.k) {
105
106 //--------------------------- Шаг --------------------------------
107             d.csx[d.j] = d.sx[d.j].multVec(d.cx[d.j]);
108             d.j = d.j + 1;
109 //----------------------- Конец шага ----------------------------
110
111         }
112
113 //--------------------------- Шаг --------------------------------
114         //Получаем результат
115         dVector rez = new dVector("0.0");
116         d.multCount = d.multCount + 1;
117         dVector m = new dVector("1.0");
118         for (int i = 0; i < d.k; i++) {
119             rez = rez.addVec(d.csx[i]);
120             m = m.multVec(d.mx[i]);
121         }
122         m = m.multNum(d.a.getCoef(d.a.size() - 1) * d.b.getCoef(d.b.size() - 1));
123         d.rez = rez.mod(m).addVec(m);
124         //Конец
125 //----------------------- Конец шага ----------------------------
126
127     }
128 }
```

C. XML–описание визуализатора

C.1. Исходный текст файла Vinograd.xml

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

<!--
    "InfPost" visualizer description (example)
    Version: $Id: Vinograd.xml,v 1.1 2003/12/24 10:50:42 geo Exp $
-->

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

<visualizer
    id="VinogradAlgorithm"
    package="ru.ifmo.vizi.vinograd"
    main-class="VinogradVisualizer"

    preferred-width="600"
    preferred-height="500"

    name-ru="Алгоритм Винограда вычисления коротких сверток"
    name-en="Vinograd's algorithm for calculate short foldings"

    author-ru="Александр Смаль"
    author-en="Alexandr Smal"
    author-email="asmal@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>
```

C.2. Исходный текст файла Vinograd-Algorithm.xml

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

<algorithm>

    <!--                D A T A    -->
    <!--        <variable description="Переменная цикла">int i;</variable>-->
    <variable description="Экземпляр апплета"
        type="VinogradVisualizer"
        name="applet"
        value="null"/>
    <variable description="Столбец полинома a(x)"
        type="dVector []"
        name="ax"
        value="null"/>
    <variable description="Столбец полинома b(x)"
        type="dVector []"
        name="bx"
        value="null"/>
    <variable description="Столбец полинома m(x)"
        type="dVector[]"
        name="mx"
        value="null"/>
    <variable description="Столбец полинома s(x)"
        type="dVector []"
        name="sx"
        value="null"/>
    <variable description="Столбец коэффициентов c(x)"
        type="dVector []"
```

```
        name="cx"
        value="null"/>
    <variable description="Столбец полинома разложения свертки"
        type="dVector []"
        name="csx"
        value="null"/>
    <variable description="Вектор результата"
        type="dVector"
        name="rez"
        value="null"/>
    <variable description="Вектор a(x)"
        type="dVector"
        name="a"
        value="null"/>
    <variable description="Вектор b(x)"
        type="dVector"
        name="b"
        value="null"/>
    <variable description="Количество взаимнопростых делителей у m(x)"
        type="int"
        name="k"
        value="4"/>
    <variable description="Локальный индекс"
        type="int"
        name="j"
        value="0"/>
    <variable description="Число вещественных умножений"
        type="int"
        name="multCount"
        value="0"/>
    <toString>
        StringBuffer s = new StringBuffer();
        s.append("step = ");
        s.append(step);
        s.append("\n");
        return s.toString();
    </toString>
    <!--                A B T O M A T    -->

    <auto id="Main" description="Собственно сам алгоритм">
        <start
            comment-ru="Это визуализатор алгоритма Винограда для вычисления коротких сверток.
                Выбирите один из примеров или введите свой."
            comment-en="This is visualizer of Vinograd's algorithm for calculating shrot
                foldings. Chose one of examples or enter your own."
            >
            <draw>
                if(@ax != null) {
                    @applet.updateColumns();
                    @applet.aCol.unHighlightAll();
                    @applet.bCol.unHighlightAll();
                    @applet.cCol.unHighlightAll();
                    @applet.kCol.unHighlightAll();
                    @applet.rezCol.unHighlightAll();
                    @applet.multCol.unHighlightAll();
                }
            </draw>
        </start>
        <step  id="InitialStep"  description="Инициализация автомата"
            comment-ru="Считаем коэффициенты полиномов в память в виде векторов."
            comment-en="Read polynoms coefficient in memory as vectors."
            level="0"
            >
            <draw>
               @applet.init();
               @applet.updateColumns();
            </draw>
            <direct>
               @ax = new dVector[@k];
               @bx = new dVector[@k];
               @cx = new dVector[@k];
               @mx = new dVector[@k];
               @sx = new dVector[@k];
               @csx = new dVector[@k];

               if (@a==null) {
```

```
                @k=3;
                @a = new dVector("1.2,2.3");
                @b = new dVector("2.5,-5.4,1.3");
                @mx[0] = new dVector("1.0,0.0");
                @mx[2] = new dVector("1.0,1.0");
                @mx[1] = new dVector("1.0,-1.0");
            }
        </direct>
    </step>
    <step  id="DegreeChanging"  description="Выбираем степень полинома m(x)"
        comment-ru="Степени полиномов a(x) и b(x) соответственно равны {0} и {1},
            следовательно степень результирующего равна {2}."
        comment-en="Degrees of polynoms a(x) and b(x) are {0} and {1}, so degree of result
             must be {2}."
        comment-args="new Integer(@a.getDeg()),new Integer(@b.getDeg()),new Integer(@a.
            getDeg()+@b.getDeg())"
        level="0">
        <draw>
            @applet.aCol.unHighlightAll();
            @applet.updateColumns();
        </draw>
        <direct>

        </direct>
    </step>
    <step  id="ModuleChoose"    description="Подбираем полином, являющийся произведением
        взаимнопростых"
        comment-ru="Выберем полином m(x), который является произведением взамнопростых с
            малыми целыми коэффициентами, и степень которого равна степени результата."
        comment-en="Choose polynom m(x) that is multiplication of coprime polynoms with
            the same degree as degree of resulting polynom and tiny integer coefficients."
        level="0"
        >
        <draw>
            if(@mx[0]==null) @applet.readModul();
            @applet.updateColumns();
            @applet.aCol.highlightAll();
        </draw>
        <direct>
        </direct>
        <reverse>
            @mx = new dVector[@k];
        </reverse>
    </step>
    <step  id="LetCalcAmodM"
        description="Убираем подсветку, обнуляет переменную цикла"
        comment-ru="Посчитаем a(x) mod m(x)[i] для i=1{0}."
        comment-en="Let's calculate a(x) mod m(x)[i] for i=1{0}."
        comment-args="new Integer(@k)"
        level="0"
        >
        <draw>
            @applet.aCol.unHighlightAll();
            @applet.bCol.unHighlightAll();
            @applet.updateColumns();
        </draw>
        <action>
            @j @= 0;
        </action>
    </step>
    <while id="WhileAmodMi"
        description="Расчет остатков при делении a(x) на m(x)[i]"
        test="@j &lt; @k"
        level="-1"
        >
        <step id="FindModA"
            description="Считает соответствующий остаток"
            comment-ru="Ищем остаток от a(x) при делении на m(x)[{0}]."
            comment-en="Finding a(x) mod m(x)[{0}]."
            comment-args="new Integer(@j)"
            level="0"
            >
            <draw>
                @applet.aCol.unHighlightAll();
                @applet.bCol.unHighlightAll();
                @applet.aCol.highlight(@j);
```

```
                @applet.bCol.highlight(@j);
                @applet.updateColumns();
            </draw>
            <action>
                @ax[@j] @= @a.mod(@mx[@j]);
                @j @= @j+1;
            </action>
        </step>
    </while>
    <step  id="LetCalcBmodM"
        description="Убираем подсветку, обнуляет переменную цикла"
        comment-ru="Посчитаем b(x) mod m(x)[i] для i=1{0}."
        comment-en="Let's calculate b(x) mod m(x)[i] for i=1{0}."
        comment-args="new Integer(@k)"
        level="0"
        >
        <draw>
            @applet.aCol.unHighlightAll();
            @applet.bCol.unHighlightAll();
            @applet.cCol.unHighlightAll();
            @applet.updateColumns();
        </draw>
        <action>
            @j @= 0;
        </action>
    </step>
    <while id="WhileBmodMi"
        description="Расчет остатков при делении b(x) на m(x)[i]"
        test="@j &lt; @k"
        level="-1"
        >
        <step id="FindModB"
            description="Считает соответствующий остаток"
            comment-ru="Ищем остаток от b(x) при делении на m(x)[{0}]."
            comment-en="Finding b(x) mod m(x)[{0}]."
            comment-args="new Integer(@j)"
            level="0"
            >
            <draw>
                @applet.aCol.unHighlightAll();
                @applet.cCol.unHighlightAll();
                @applet.aCol.highlight(@j);
                @applet.cCol.highlight(@j);
                @applet.updateColumns();
            </draw>
            <action>
                @bx[@j] @= @b.mod(@mx[@j]);
                @j @= @j+1;
            </action>
        </step>
    </while>
    <step  id="LetCalcSx"
        description="Убираем подсветку, обнуляет переменную цикла"
        comment-ru="Посчитаем s(x)[i] = a(x)[i]*b(x)[i] для i=1{0}."
        comment-en="Let's calculate s(x)[i] = a(x)[i]*b(x)[i] for i=1{0}."
        comment-args="new Integer(@k)"
        level="0"
        >

        <draw>
            @applet.aCol.unHighlightAll();
            @applet.bCol.unHighlightAll();
            @applet.cCol.unHighlightAll();
            @applet.updateColumns();
        </draw>
        <action>
            @j @= 0;
        </action>

    </step>
    <while id="WhileSx"
        description="Расчет s(x)[i]"
        test="@j &lt; @k"
        level="-1"
        >
        <step id="FindSxModM"
```

```
            description="Считает соответствующий s(x)[i]"
            comment-ru="Посчитаем s(x)[{0}] = a(x)[{0}]*b(x)[{0}]. Нам понадобится {1}
                вещественных(ое) умножений(ие)."
            comment-en="Let's calculate s(x)[{0}] = a(x)[{0}]*b(x)[{0}]. We need {1}
                rational multiplication(s)."
            comment-args="new Integer(@j), new Integer(@ax[@j-1].size()*@bx[@j-1].size())"
            level="0"
            >
            <draw>
                @applet.aCol.unHighlightAll();
                @applet.bCol.unHighlightAll();
                @applet.cCol.unHighlightAll();

                @applet.aCol.highlight(@j);
                @applet.bCol.highlight(@j);
                @applet.cCol.highlight(@j);

                @applet.updateColumns();
            </draw>
            <action>
                @sx[@j] @= @ax[@j].multVec(@bx[@j]).mod(@mx[@j]);
                @multCount @= @multCount + @ax[@j].size()*@bx[@j].size();
                @j @= @j+1;
            </action>
        </step>
    </while>
    <step  id="LetCalcCx"
        description="Убираем подсветку, обнуляет переменную цикла"
        comment-ru="Посчитаем c(x)[i] - соответствующие множители в Китайской теореме об
            остатках."
        comment-en="Let's calculate c(x)[i] - multipliers from China theorem about
            remainders."
        level="0"
        >
        <draw>
            @applet.aCol.unHighlightAll();
            @applet.bCol.unHighlightAll();
            @applet.cCol.unHighlightAll();
            @applet.updateColumns();
        </draw>
        <action>
            @j @= 0;
        </action>

    </step>
    <step  id="FindCx"
            description="Считает соответствующий c(x)[i]"
            comment-ru="Посчитаем все c(x)[i] - решим {0} диофантовых уравнений вида m(x)[
                i]*n(x)[i]+N(x)[i]*m(x)/m(x)[i]=1. Тогда c(x)[i]=m(x)[i]*N(x)[i]."
            comment-en="Let's calculate every c(x)[i] - solve {0} diofants equations m(x)[
                i]*n(x)[i]+N(x)[i]*m(x)/m(x)[i]=1. So c(x)[i]=m(x)[i]*N(x)[i]."
            comment-args="new Integer(@k)"
            level="0"
            >
          <draw>
               @applet.aCol.unHighlightAll();
               @applet.bCol.unHighlightAll();
               @applet.cCol.unHighlightAll();
               @applet.bCol.highlightAll();
               @applet.updateColumns();
          </draw>
          <action>
                for(int i=0; i&lt;@k; i++) {
                    dVector M = new dVector("1.0");
                    for(int j=0; j&lt;@k; j++) {
                        if(j==i) continue;
                        M = M.multVec(@mx[j]);
                    }
                    @cx[i] @= @mx[i].getSolve(M).multVec(M);
                }
                @j @= @j+1;
     </action>
    </step>
    <step  id="LetCalcCSx"
        description="Убираем подсветку, обнуляет переменную цикла"
        comment-ru="Посчитаем cs(x)[i] = c(x)[i]\u00d7s(x)[i]."
```

```
319             comment-en="Let's calculate cs(x)[i] = c(x)[i]\u00d7s(x)[i]."
320             comment-args="new Integer(@k)"
321             level="0"
322             >
323             <draw>
324                 @applet.aCol.unHighlightAll();
325                 @applet.bCol.unHighlightAll();
326                 @applet.cCol.unHighlightAll();
327                 @applet.updateColumns();
328             </draw>
329             <action>
330                 @j @= 0;
331             </action>
332
333         </step>
334         <while id="WhileCSx"
335             description="Расчет s(x)[i]"
336             test="@j &lt; @k"
337             level="-1"
338             >
339             <step id="FindSxCx"
340                 description="Считает s(x)[i]\u00d7c(x)[i]"
341                 comment-ru="Посчитаем c(x)[{0}]\u00d7s(x)[{0}]."
342                 comment-en="Let's calculate c(x)[{0}]*s(x)[{0}]."
343                 comment-args="new Integer(@j)"
344                 level="0"
345                 >
346                 <draw>
347                     @applet.aCol.unHighlightAll();
348                     @applet.bCol.unHighlightAll();
349                     @applet.cCol.unHighlightAll();
350
351                     @applet.aCol.highlight(@j);
352                     @applet.bCol.highlight(@j);
353                     @applet.cCol.highlight(@j);
354
355                     @applet.updateColumns();
356                 </draw>
357                 <action>
358                     @csx[@j] @= @sx[@j].multVec(@cx[@j]);
359                     @j @= @j+1;
360                 </action>
361             </step>
362         </while>
363         <step  id="LetCalcRezult"
364             description="Убираем подсветку, обнуляет переменную цикла"
365             comment-ru="Осталось сложить все cs(x)[i] по модулю m(x) и прибавить m(x)\u00d7s
                {0} - старший коэффициент s(x)."
366             comment-en="Finally we can get answer - sum of every cs(x)[i] mod m(x) + m(x)\
                u00d7s{0} - main coefficient of s(x)."
367             comment-args="new Integer(@a.getDeg()+@b.getDeg())"
368             level="0"
369             >
370             <draw>
371                 @applet.aCol.unHighlightAll();
372                 @applet.bCol.unHighlightAll();
373                 @applet.cCol.highlightAll();
374                 @applet.rezCol.unHighlightAll();
375                 @applet.updateColumns();
376             </draw>
377             <action>
378                 @j @= 0;
379             </action>
380         </step>
381
382         <step  id="Rezult"
383                 description="Выводит результат"
384                 comment-ru="s{1} = a{2}\u00d7b{3} = {4}. Это потребует еще одно вещественное
                    умножение. Посчитаем s(x)."
385                 comment-en="s{1} = a{2}\u00d7b{3} = {4}. It takes one real multiplication. Let
                    's calculate s(x)."
386                 comment-args="@rez.toStr(), new Integer(@a.getDeg()+@b.getDeg()), new Integer(
                    @a.getDeg()), new Integer(@b.getDeg()), new Double(Math.round(@a.getCoef(
                    @a.size()-1)*@b.getCoef(@b.size()-1)*100)/100.0)"
387                 level="0"
388                 >
```

```
389                 <draw>
390                      @applet.rezCol.highlight(1);
391                      @applet.cCol.unHighlightAll();
392                      @applet.multCol.unHighlightAll();
393                      @applet.updateColumns();
394                 </draw>
395                 <action>
396                      dVector rez = new dVector("0.0");
397                      @multCount @= @multCount + 1;
398                      dVector m = new dVector("1.0");
399                      for(int i=0; i&lt;@k; i++) {
400                          rez = rez.addVec(@csx[i]);
401                          m = m.multVec(@mx[i]);
402                      }
403                      m = m.multNum(@a.getCoef(@a.size()-1)*@b.getCoef(@b.size()-1));
404                      @rez @= rez.mod(m).addVec(m);
405                </action>
406           </step>
407           <step  id="Finish"
408                   description="Говорит о преимуществах алгоритма"
409                   comment-ru="Таким образом свертка была посчитана за {0} вещественных умножений
                           вместо {1}, если считать в лоб."
410                   comment-en="So we have got answer in {0} real multiplications instead of {1}
                          if we were calculating without this algorithm."
411                   comment-args="new Integer(@multCount), new Integer(@a.size()*@b.size())"
412                   level="0"
413                   >
414                 <draw>
415                      @applet.rezCol.highlight(1);
416                      @applet.multCol.highlight(1);
417                      @applet.cCol.unHighlightAll();
418                      @applet.updateColumns();
419                 </draw>
420                 <action>
421
422                </action>
423           </step>
424           <finish
425               comment-ru="Конец."
426               comment-en="Finish."
427               >
428               <draw>
429                  @applet.rezCol.unHighlightAll();
430                  @applet.multCol.unHighlightAll();
431               </draw>
432               </finish>
433       </auto>
434
435 </algorithm>
```

C.3. Исходный текст файла Vinograd-Configuration.xml

```
1  <?xml version="1.0" encoding="WINDOWS-1251"?>
2
3  <!--
4      "InfPost" visualizer configuration.
5      Shared for Vinograd.xml
6      Version: $Id: Vinograd-Configuration.xml v 1.0$
7  -->
8
9  <configuration>
10
11     <!-- Labels -->
12
13
14
15     <group
16         description="Диалог с сообщением об ошибке ввода"
17         param="error">
18     <message
19         param="input"
20         description="Надпись, которая будет высвечиваться при неправильном вводе данных"
21         message-ru="Неправильный ввод данных:"
22         message-en="Error input:"
23         />
24     <message
```

```
25          param="title"
26          description="Надпись, которая будет высвечиваться при неправильном вводе данных в
                заголовке окна"
27          message-ru="Ошибка"
28          message-en="Error"
29          />
30      <message
31          param="ax"
32          description="Название поля для a(x),которое будет высвечиваться при неправильном вводе
                данных"
33          message-ru="a(x)"
34          message-en="a(x)"
35          />
36      <message
37          param="bx"
38          description="Название поля для b(x),которое будет высвечиваться при неправильном вводе
                данных"
39          message-ru="b(x)"
40          message-en="b(x)"
41          />
42      <message
43          param="message"
44          description="Комментарий к ошибке"
45          message-ru="Коэффициенты полиномов записываются через запятую. Разделитель дробной и
                целой части - точка. Количество коэффициентов не должно превышать {0}."
46          message-en="All factors are splited with comas. Integer and fractional parts are
                splited with dot. Count of factors must be no more then {0}."
47          />
48          </group>
49      <group
50          description="Надписи в заголовках колонок"
51          param="label"
52      >
53      <message
54          param="ax"
55          description="Надпись, которая будет высвечиваться в заголовке колонки с a(x)"
56          message-ru="a(x)"
57          message-en="a(x)"
58          />
59      <message
60          param="bx"
61          description="Надпись, которая будет высвечиваться в заголовке колонки с b(x)"
62          message-ru="b(x)"
63          message-en="b(x)"
64          />
65      <message
66          param="rez"
67          description="Надпись, которая будет высвечиваться в заголовке клетки с результатом"
68          message-ru="Результат"
69          message-en="Result"
70          />
71      <message
72          param="mult"
73          description="Надпись, которая будет высвечиваться в заголовке клетки с количеством
                вещественных умножений"
74          message-ru="Число вещественныз умножений"
75          message-en="Count of real multiplications"
76          />
77      <message
78          param="mx"
79          description="Надпись, которая будет высвечиваться в заголовке колонки с m(x)"
80          message-ru="m(x)"
81          message-en="m(x)"
82          />
83      <message
84          param="sx"
85          description="Надпись, которая будет высвечиваться в заголовке колонки с s(x)"
86          message-ru="s(x)=a(x)\u00d7b(x)"
87          message-en="s(x)=a(x)\u00d7b(x)"
88          />
89      <message
90          param="k"
91          description="Надпись, которая будет высвечиваться в заголовке колонки с номерами k"
92          message-ru="k"
93          message-en="k"
94          />
```

```
95      <message
96          param="cx"
97          description="Надпись, которая будет высвечиваться в заголовке колонки с c(x)"
98          message-ru="c(x)"
99          message-en="c(x)"
100         />
101     <message
102         param="csx"
103         description="Надпись, которая будет высвечиваться в заголовке колонки с c(x)*s(x)"
104         message-ru="c(x)\u00d7s(x)"
105         message-en="c(x)\u00d7s(x)"
106         />
107     </group>
108     <property
109         param="max-polynom-deg"
110         description="Максимально возможная степень исходного полинома (a(x) и b(x))"
111         value="3"
112         />
113     <group
114         param="Examples"
115         description="ListBox с примерами"
116         >
117         <property
118             param="size"
119             description="Количество элементво"
120             value="6"
121             />
122         <message
123             param="label-0"
124             description="Подпись произвольного примера"
125             message-ru="Ваш вариант"
126             message-en="Your variant"
127             />
128         <message
129             param="label-1"
130             description="Подпись примера 1"
131             message-ru="Пример 1"
132             message-en="Example 1"
133             />
134         <message
135             param="label-2"
136             description="Подпись примера 2"
137             message-ru="Пример 2"
138             message-en="Example 2"
139             />
140         <message
141             param="label-3"
142             description="Подпись примера 3"
143             message-ru="Пример 3"
144             message-en="Example 3"
145             />
146         <message
147             param="label-4"
148             description="Подпись примера 4"
149             message-ru="Пример 4"
150             message-en="Example 4"
151             />
152         <message
153             param="label-5"
154             description="Подпись примера 5"
155             message-ru="Пример 5"
156             message-en="Example 5"
157             />
158         <message
159             param="hint"
160             description="Хинт"
161             message-ru="Примеры полиномов"
162             message-en="Examples of polynoms"
163             />
164         <message
165             param="example-1"
166             description="Пример 1"
167             message-ru="2.3,3.4;4.3,3.0"
168             message-en="2.3,3.4;4.3,3.0"
169             />
170         <message
```

```
            param="example-2"
            description="Пример 2"
            message-ru="1.2,3.1,2.3;2.5,1.3"
            message-en="1.2,3.1,2.3;2.5,1.3"
            />
        <message
            param="example-3"
            description="Пример 3"
            message-ru="2.3,1.3,3.1;4.0,2.4,4.1"
            message-en="2.3,1.3,3.1;4.0,2.4,4.1"
            />
        <message
            param="example-4"
            description="Пример 4"
            message-ru="3.5,4.2,6.1,3.4;2.4,7.3,4.2"
            message-en="3.5,4.2,6.1,3.4;2.4,7.3,4.2"
            />
        <message
            param="example-5"
            description="Пример 5"
            message-ru="2.9,4.2,4.5,6.3;6.2,5.4,10.1,4.7"
            message-en="2.9,4.2,4.5,6.3;6.2,5.4,10.1,4.7"
            />
    </group>
    <group
        param="Modul"
        description="m(x)[i] для соответствующих степеней результирующих полиномов"
        >
        <message
            param="0"
            description="m(x)[i] для степени результирующего полинома 0"
            message-ru="1.0,0.0"
            message-en="1.0,0.0"
            />
        <message
            param="1"
            description="m(x)[i] для степени результирующего полинома 1"
            message-ru="1.0,0.0"
            message-en="1.0,0.0"
            />
        <message
            param="2"
            description="m(x)[i] для степени результирующего полинома 2"
            message-ru="1.0,0.0;1.0,1.0"
            message-en="1.0,0.0;1.0,1.0"
            />
        <message
            param="3"
            description="m(x)[i] для степени результирующего полинома 3"
            message-ru="1.0,0.0;1.0,1.0;1.0,-1.0"
            message-en="1.0,0.0;1.0,1.0;1.0,-1.0"
            />
        <message
            param="4"
            description="m(x)[i] для степени результирующего полинома 4"
            message-ru="1.0,0.0;1.0,2.0;1.0,-2.0;1.0,1.0"
            message-en="1.0,0.0;1.0,2.0;1.0,-2.0;1.0,1.0"
            />
        <message
            param="5"
            description="m(x)[i] для степени результирующего полинома 5"
            message-ru="1.0,0.0;1.0,1.0;1.0,0.0,1.0;1.0,-1.0"
            message-en="1.0,0.0;1.0,1.0;1.0,0.0,1.0;1.0,-1.0"
            />
        <message
            param="6"
            description="m(x)[i] для степени результирующего полинома 6"
            message-ru="1.0,0.0;1.0,1.0;1.0,0.0,1.0;1.0,0.0,2.0"
            message-en="1.0,0.0;1.0,1.0;1.0,0.0,1.0;1.0,0.0,2.0"
            />
    </group>
    <group
        param="InputFielda"
        description="Поле ввода полинома a(x)"
        >
        <message
```

```
            param="hint"
            description="Хинт для поля ввода полинома a(x)"
            message-ru="Входное выражение: полином a(x)"
            message-en="Input expression: polynom a(x)"
            />
        <message
            param="label"
            description="Подпись для поля ввода для полинома a(x)"
            message-ru="a(x)"
            message-en="a(x)"
            />
    </group>
    <group
        param="InputFieldb"
        description="Поле ввода полинома b(x)"
        >

        <message
            param="hint"
            description="Хинт для поля ввода полинома b(x)"
            message-ru="Входное выражение: полином b(x)"
            message-en="Input expression: polynom b(x)"
            />
        <message
            param="label"
            description="Подпись для поля ввода для полинома b(x)"
            message-ru="b(x)"
            message-en="b(x)"
            />
    </group>
    <styleset
        description="Стиль для колонок с векторами (a(x),b(x),c(x),s(x),m(x),c(x)*s(x))"
        param="arrayCol"
        >
        <style
            description="Plain element"
            fill-color="976bff"
            text-color="000000"
            text-align="0.5"
            message-align="1.0"
            border-color="000000"
            border-status="true"
            fill-status="true"
            aspect-status="false"
            padding="0.5"
            >
            <font
                face="Serif"
                size="12"
                style="plain"
                />
        </style>
        <style
            description="Header element"
            fill-color="77ef47"
            message-align="0.5"
            />
        <style
            description="Selected element"
            fill-color="ff89b0"
            />
    </styleset>
    <styleset
        description="Стиль для клетки с результатом"
        param="rezCol"
        >
        <style
            description="Plain element"
            fill-color="976bff"
            text-color="000000"
            text-align="0.5"
            border-color="000000"
            border-status="true"
            fill-status="true"
            aspect-status="false"
            padding="0.5"
```

```
            >
            <font
                face="Serif"
                size="12"
                style="plain"
                />
        </style>
        <style
            description="Header element"
            fill-color="77ef47"
            />
        <style
            description="Selected element"
            fill-color="ff89b0"
            />
    </styleset>
    <styleset
        description="Стиль для клеткм вещественных умножений"
        param="multCol"
        >
        <style
            description="Plain element"
            fill-color="976bff"
            text-color="000000"
            text-align="0.5"
            border-color="000000"
            border-status="true"
            fill-status="true"
            aspect-status="false"
            padding="0.5"
            >
            <font
                face="Serif"
                size="12"
                style="plain"
                />
        </style>
        <style
            description="Header element"
            fill-color="77ef47"
            />
        <style
            description="Selected element"
            fill-color="ff89b0"
            />
    </styleset>
    <styleset
        description="Стиль для колонки с номерами"
        param="kCol"
        >
        <style
            description="Plain element"
            fill-color="fff200"
            text-color="000000"
            text-align="0.5"
            border-color="000000"
            border-status="true"
            fill-status="true"
            aspect-status="false"
            padding="0.5"
            >
            <font
                face="Serif"
                size="12"
                style="plain"
                />
        </style>
        <style
            description="Header element"
            fill-color="77ef47"
            />
        <style
            description="Selected element"
            fill-color="ff89b0"
            />
    </styleset>
```

```
399     <property
400         description="Comment pane height"
401         param="comment-height"
402         value="40"
403         />
404 
405     <!-- Save/Load dialog -->
406 
407     <group
408         description="Save/Load dialog configuration"
409         param="SaveLoadDialog"
410         >
411         <property
412             description="Height of the comment pane"
413             param="CommentPane-lines"
414             value="2"
415             />
416         <property
417             description="Width of the text area"
418             param="columns"
419             value="40"
420             />
421         <property
422             description="Height of the text area"
423             param="rows"
424             value="9"
425             />
426         <group
427             description="Комментарии в Save файле"
428             param="comments">
429             <message
430                 description="Комментарий к номеру шага"
431                 param="step"
432                 message-ru="Номер шага"
433                 message-en="Current step"
434                 />
435             <message
436                 param="axLength"
437                 description="Комментарий к длине полинома a(x)"
438                 message-ru="Длина многочлена a(x)"
439                 message-en="Length of polynot a(x)"
440                 />
441             <message
442                 param="ax"
443                 description="Комментарий к полиному a(x)"
444                 message-ru="Многочлен a(x)"
445                 message-en="Polynot a(x)"
446                 />
447             <message
448                 param="bxLength"
449                 description="Комментарий к длине полинома a(x)"
450                 message-ru="Длина многочлена b(x)"
451                 message-en="Length of polynot b(x)"
452                 />
453             <message
454                 param="bx"
455                 description="Комментарий к полиному b(x)"
456                 message-ru="Многочлен b(x)"
457                 message-en="Polynot b(x)"
458                 />
459         </group>
460     </group>
461 </configuration>
```

D. Исходные коды автомата, сгенерированные автоматически

D.1. Исходный текст файла VinogradAlgorithm.java

```
1  package ru.ifmo.vizi.vinograd;
2
3  import ru.ifmo.vizi.base.auto.*;
4  import java.util.Locale;
5
6  public final class VinogradAlgorithm extends BaseAutoReverseAutomata {
7      /**
8        * Модель данных.
9        */
10     public final Data d = new Data();
11
12     /**
13       * Конструктор для языка
14       */
15     public VinogradAlgorithm(Locale locale) {
16         super("ru.ifmo.vizi.vinograd.Comments", locale);
17         init(new Main(), d);
18     }
19
20     /**
21       * Данные.
22       */
23     public final class Data {
24         /**
25           * Экземпляр апплета.
26           */
27         public VinogradVisualizer applet = null;
28
29         /**
30           * Столбец полинома a(x).
31           */
32         public dVector [] ax = null;
33
34         /**
35           * Столбец полинома b(x).
36           */
37         public dVector [] bx = null;
38
39         /**
40           * Столбец полинома m(x).
41           */
42         public dVector[] mx = null;
43
44         /**
45           * Столбец полинома s(x).
46           */
47         public dVector [] sx = null;
48
49         /**
50           * Столбец коэффициентов c(x).
51           */
52         public dVector [] cx = null;
53
54         /**
55           * Столбец полинома разложения свертки.
56           */
57         public dVector [] csx = null;
58
59         /**
60           * Вектор результата.
61           */
62         public dVector rez = null;
63
64         /**
65           * Вектор a(x).
66           */
67         public dVector a = null;
68
69         /**
```

```
70          * Вектор b(x).
71          */
72         public dVector b = null;
73
74         /**
75          * Количество взаимнопростых делителей у m(x).
76          */
77         public int k = 4;
78
79         /**
80          * Локальный индекс.
81          */
82         public int j = 0;
83
84         /**
85          * Число вещественных умножений.
86          */
87         public int multCount = 0;
88
89         public String toString() {
90             StringBuffer s = new StringBuffer();
91             s.append("step = ");
92             s.append(step);
93             s.append("\n");
94             return s.toString();
95         }
96     }
97
98     /**
99      * Собственно сам алгоритм.
100     */
101    private final class Main extends BaseAutomata implements Automata {
102        /**
103         * Начальное состояние автомата.
104         */
105        private final int START_STATE = 0;
106
107        /**
108         * Конечное состояние автомата.
109         */
110        private final int END_STATE = 21;
111
112        /**
113         * Конструктор.
114         */
115        public Main() {
116            super(
117                "Main",
118                0, // Номер начального состояния
119                21, // Номер конечного состояния
120                new String[]{
121                    "Начальное состояние",
122                    "Инициализация автомата",
123                    "Выбираем степень полинома m(x)",
124                    "Подбираем полином, являющийся произведением взаимнопростых",
125                    "Убираем подсветку, обнуляет переменную цикла",
126                    "Расчет остатков при делении a(x) на m(x)[i]",
127                    "Считает соответствующий остаток",
128                    "Убираем подсветку, обнуляет переменную цикла",
129                    "Расчет остатков при делении b(x) на m(x)[i]",
130                    "Считает соответствующий остаток",
131                    "Убираем подсветку, обнуляет переменную цикла",
132                    "Расчет s(x)[i]",
133                    "Считает соответствующий s(x)[i]",
134                    "Убираем подсветку, обнуляет переменную цикла",
135                    "Считает соответствующий c(x)[i]",
136                    "Убираем подсветку, обнуляет переменную цикла",
137                    "Расчет s(x)[i]",
138                    "Считает s(x)[i]\u00d7c(x)[i]",
139                    "Убираем подсветку, обнуляет переменную цикла",
140                    "Выводит результат",
141                    "Говорит о преимуществах алгоритма",
142                    "Конечное состояние"
143                }, new int[]{
144                    Integer.MAX_VALUE, // Начальное состояние,
145                    0, // Инициализация автомата
```

```
                0, // Выбираем степень полинома m(x)
                0, // Подбираем полином, являющийся произведением взаимнопростых
                0, // Убираем подсветку, обнуляет переменную цикла
                -1, // Расчет остатков при делении a(x) на m(x)[i]
                0, // Считает соответствующий остаток
                0, // Убираем подсветку, обнуляет переменную цикла
                -1, // Расчет остатков при делении b(x) на m(x)[i]
                0, // Считает соответствующий остаток
                0, // Убираем подсветку, обнуляет переменную цикла
                -1, // Расчет s(x)[i]
                0, // Считает соответствующий s(x)[i]
                0, // Убираем подсветку, обнуляет переменную цикла
                0, // Считает соответствующий c(x)[i]
                0, // Убираем подсветку, обнуляет переменную цикла
                -1, // Расчет s(x)[i]
                0, // Считает s(x)[i]\u00d7c(x)[i]
                0, // Убираем подсветку, обнуляет переменную цикла
                0, // Выводит результат
                0, // Говорит о преимуществах алгоритма
                Integer.MAX_VALUE, // Конечное состояние
            }
        );
    }

    /**
      * Сделать один шаг автомата в перед.
      */
    protected void doStepForward(int level) {
        // Переход в следующее состояние
        switch (state) {
            case START_STATE: { // Начальное состояние
                state = 1; // Инициализация автомата
                break;
            }
            case 1: { // Инициализация автомата
                state = 2; // Выбираем степень полинома m(x)
                break;
            }
            case 2: { // Выбираем степень полинома m(x)
                state = 3; // Подбираем полином, являющийся произведением взаимнопростых
                break;
            }
            case 3: { // Подбираем полином, являющийся произведением взаимнопростых
                state = 4; // Убираем подсветку, обнуляет переменную цикла
                break;
            }
            case 4: { // Убираем подсветку, обнуляет переменную цикла
                stack.pushBoolean(false);
                state = 5; // Расчет остатков при делении a(x) на m(x)[i]
                break;
            }
            case 5: { // Расчет остатков при делении a(x) на m(x)[i]
                if (d.j < d.k) {
                    state = 6; // Считает соответствующий остаток
                } else {
                    state = 7; // Убираем подсветку, обнуляет переменную цикла
                }
                break;
            }
            case 6: { // Считает соответствующий остаток
                stack.pushBoolean(true);
                state = 5; // Расчет остатков при делении a(x) на m(x)[i]
                break;
            }
            case 7: { // Убираем подсветку, обнуляет переменную цикла
                stack.pushBoolean(false);
                state = 8; // Расчет остатков при делении b(x) на m(x)[i]
                break;
            }
            case 8: { // Расчет остатков при делении b(x) на m(x)[i]
                if (d.j < d.k) {
                    state = 9; // Считает соответствующий остаток
                } else {
                    state = 10; // Убираем подсветку, обнуляет переменную цикла
                }
                break;
```

```
222                 }
223                 case 9: { // Считает соответствующий остаток
224                     stack.pushBoolean(true);
225                     state = 8; // Расчет остатков при делении b(x) на m(x)[i]
226                     break;
227                 }
228                 case 10: { // Убираем подсветку, обнуляет переменную цикла
229                     stack.pushBoolean(false);
230                     state = 11; // Расчет s(x)[i]
231                     break;
232                 }
233                 case 11: { // Расчет s(x)[i]
234                     if (d.j < d.k) {
235                         state = 12; // Считает соответствующий s(x)[i]
236                     } else {
237                         state = 13; // Убираем подсветку, обнуляет переменную цикла
238                     }
239                     break;
240                 }
241                 case 12: { // Считает соответствующий s(x)[i]
242                     stack.pushBoolean(true);
243                     state = 11; // Расчет s(x)[i]
244                     break;
245                 }
246                 case 13: { // Убираем подсветку, обнуляет переменную цикла
247                     state = 14; // Считает соответствующий c(x)[i]
248                     break;
249                 }
250                 case 14: { // Считает соответствующий c(x)[i]
251                     state = 15; // Убираем подсветку, обнуляет переменную цикла
252                     break;
253                 }
254                 case 15: { // Убираем подсветку, обнуляет переменную цикла
255                     stack.pushBoolean(false);
256                     state = 16; // Расчет s(x)[i]
257                     break;
258                 }
259                 case 16: { // Расчет s(x)[i]
260                     if (d.j < d.k) {
261                         state = 17; // Считает s(x)[i]\u00d7c(x)[i]
262                     } else {
263                         state = 18; // Убираем подсветку, обнуляет переменную цикла
264                     }
265                     break;
266                 }
267                 case 17: { // Считает s(x)[i]\u00d7c(x)[i]
268                     stack.pushBoolean(true);
269                     state = 16; // Расчет s(x)[i]
270                     break;
271                 }
272                 case 18: { // Убираем подсветку, обнуляет переменную цикла
273                     state = 19; // Выводит результат
274                     break;
275                 }
276                 case 19: { // Выводит результат
277                     state = 20; // Говорит о преимуществах алгоритма
278                     break;
279                 }
280                 case 20: { // Говорит о преимуществах алгоритма
281                     state = END_STATE;
282                     break;
283                 }
284             }
285 
286             // Действие в текущем состоянии
287             switch (state) {
288                 case 1: { // Инициализация автомата
289                     d.ax = new dVector[d.k];
290                     d.bx = new dVector[d.k];
291                     d.cx = new dVector[d.k];
292                     d.mx = new dVector[d.k];
293                     d.sx = new dVector[d.k];
294                     d.csx = new dVector[d.k];
295 
296                     if (d.a==null) {
297                         d.k=3;
```

```
298                     d.a = new dVector("1.2,2.3");
299                     d.b = new dVector("2.5,-5.4,1.3");
300                     d.mx[0] = new dVector("1.0,0.0");
301                     d.mx[2] = new dVector("1.0,1.0");
302                     d.mx[1] = new dVector("1.0,-1.0");
303                  }
304                 break;
305             }
306             case 2: { // Выбираем степень полинома m(x)
307                 break;
308             }
309             case 3: { // Подбираем полином, являющийся произведением взаимнопростых
310                 break;
311             }
312             case 4: { // Убираем подсветку, обнуляет переменную цикла
313                 startSection();
314                 storeField(d, "j");
315                 d.j = 0;
316                 break;
317             }
318             case 5: { // Расчет остатков при делении a(x) на m(x)[i]
319                 break;
320             }
321             case 6: { // Считает соответствующий остаток
322                 startSection();
323                 storeArray(d.ax, d.j);
324                 d.ax[d.j] = d.a.mod(d.mx[d.j]);
325                 storeField(d, "j");
326                 d.j = d.j+1;
327                 break;
328             }
329             case 7: { // Убираем подсветку, обнуляет переменную цикла
330                 startSection();
331                 storeField(d, "j");
332                 d.j = 0;
333                 break;
334             }
335             case 8: { // Расчет остатков при делении b(x) на m(x)[i]
336                 break;
337             }
338             case 9: { // Считает соответствующий остаток
339                 startSection();
340                 storeArray(d.bx, d.j);
341                 d.bx[d.j] = d.b.mod(d.mx[d.j]);
342                 storeField(d, "j");
343                 d.j = d.j+1;
344                 break;
345             }
346             case 10: { // Убираем подсветку, обнуляет переменную цикла
347                 startSection();
348                 storeField(d, "j");
349                 d.j = 0;
350                 break;
351             }
352             case 11: { // Расчет s(x)[i]
353                 break;
354             }
355             case 12: { // Считает соответствующий s(x)[i]
356                 startSection();
357                 storeArray(d.sx, d.j);
358                 d.sx[d.j] = d.ax[d.j].multVec(d.bx[d.j]).mod(d.mx[d.j]);
359                 storeField(d, "multCount");
360                 d.multCount = d.multCount + d.ax[d.j].size()*d.bx[d.j].size();
361                 storeField(d, "j");
362                 d.j = d.j+1;
363                 break;
364             }
365             case 13: { // Убираем подсветку, обнуляет переменную цикла
366                 startSection();
367                 storeField(d, "j");
368                 d.j = 0;
369                 break;
370             }
371             case 14: { // Считает соответствующий c(x)[i]
372                 startSection();
373                 for(int i=0; i<d.k; i++) {
```

```
374                         dVector M = new dVector("1.0");
375                         for(int j=0; j<d.k; j++) {
376                             if(j==i) continue;
377                             M = M.multVec(d.mx[j]);
378                         }
379                         storeArray(d.cx, i);
380                         d.cx[i] = d.mx[i].getSolve(M).multVec(M);
381                     }
382                     storeField(d, "j");
383                     d.j = d.j+1;
384                     break;
385                 }
386                 case 15: { // Убираем подсветку, обнуляет переменную цикла
387                     startSection();
388                     storeField(d, "j");
389                     d.j = 0;
390                     break;
391                 }
392                 case 16: { // Расчет s(x)[i]
393                     break;
394                 }
395                 case 17: { // Считает s(x)[i]\u00d7c(x)[i]
396                     startSection();
397                     storeArray(d.csx, d.j);
398                     d.csx[d.j] = d.sx[d.j].multVec(d.cx[d.j]);
399                     storeField(d, "j");
400                     d.j = d.j+1;
401                     break;
402                 }
403                 case 18: { // Убираем подсветку, обнуляет переменную цикла
404                     startSection();
405                     storeField(d, "j");
406                     d.j = 0;
407                     break;
408                 }
409                 case 19: { // Выводит результат
410                     startSection();
411                     dVector rez = new dVector("0.0");
412                     storeField(d, "multCount");
413                     d.multCount = d.multCount + 1;
414                     dVector m = new dVector("1.0");
415                     for(int i=0; i<d.k; i++) {
416                         rez = rez.addVec(d.csx[i]);
417                         m = m.multVec(d.mx[i]);
418                     }
419                     m = m.multNum(d.a.getCoef(d.a.size()-1)*d.b.getCoef(d.b.size()-1));
420                     storeField(d, "rez");
421                     d.rez = rez.mod(m).addVec(m);
422                     break;
423                 }
424                 case 20: { // Говорит о преимуществах алгоритма
425                     startSection();
426                     break;
427                 }
428             }
429         }
430 
431         /**
432           * Сделать один шаг автомата назад.
433           */
434         protected void doStepBackward(int level) {
435             // Обращение действия в текущем состоянии
436             switch (state) {
437                 case 1: { // Инициализация автомата
438                     break;
439                 }
440                 case 2: { // Выбираем степень полинома m(x)
441                     break;
442                 }
443                 case 3: { // Подбираем полином, являющийся произведением взаимнопростых
444                     d.mx = new dVector[d.k];
445                     break;
446                 }
447                 case 4: { // Убираем подсветку, обнуляет переменную цикла
448                     restoreSection();
449                     break;
```

```
450                 }
451                 case 5: { // Расчет остатков при делении a(x) на m(x)[i]
452                     break;
453                 }
454                 case 6: { // Считает соответствующий остаток
455                     restoreSection();
456                     break;
457                 }
458                 case 7: { // Убираем подсветку, обнуляет переменную цикла
459                     restoreSection();
460                     break;
461                 }
462                 case 8: { // Расчет остатков при делении b(x) на m(x)[i]
463                     break;
464                 }
465                 case 9: { // Считает соответствующий остаток
466                     restoreSection();
467                     break;
468                 }
469                 case 10: { // Убираем подсветку, обнуляет переменную цикла
470                     restoreSection();
471                     break;
472                 }
473                 case 11: { // Расчет s(x)[i]
474                     break;
475                 }
476                 case 12: { // Считает соответствующий s(x)[i]
477                     restoreSection();
478                     break;
479                 }
480                 case 13: { // Убираем подсветку, обнуляет переменную цикла
481                     restoreSection();
482                     break;
483                 }
484                 case 14: { // Считает соответствующий c(x)[i]
485                     restoreSection();
486                     break;
487                 }
488                 case 15: { // Убираем подсветку, обнуляет переменную цикла
489                     restoreSection();
490                     break;
491                 }
492                 case 16: { // Расчет s(x)[i]
493                     break;
494                 }
495                 case 17: { // Считает s(x)[i]\u00d7c(x)[i]
496                     restoreSection();
497                     break;
498                 }
499                 case 18: { // Убираем подсветку, обнуляет переменную цикла
500                     restoreSection();
501                     break;
502                 }
503                 case 19: { // Выводит результат
504                     restoreSection();
505                     break;
506                 }
507                 case 20: { // Говорит о преимуществах алгоритма
508                     restoreSection();
509                     break;
510                 }
511             }
512 
513             // Переход в предыдущее состояние
514             switch (state) {
515                 case 1: { // Инициализация автомата
516                     state = START_STATE;
517                     break;
518                 }
519                 case 2: { // Выбираем степень полинома m(x)
520                     state = 1; // Инициализация автомата
521                     break;
522                 }
523                 case 3: { // Подбираем полином, являющийся произведением взаимнопростых
524                     state = 2; // Выбираем степень полинома m(x)
525                     break;
```

```
526                 }
527                 case 4: { // Убираем подсветку, обнуляет переменную цикла
528                     state = 3; // Подбираем полином, являющийся произведением взаимнопростых
529                     break;
530                 }
531                 case 5: { // Расчет остатков при делении a(x) на m(x)[i]
532                     if (stack.popBoolean()) {
533                         state = 6; // Считает соответствующий остаток
534                     } else {
535                         state = 4; // Убираем подсветку, обнуляет переменную цикла
536                     }
537                     break;
538                 }
539                 case 6: { // Считает соответствующий остаток
540                     state = 5; // Расчет остатков при делении a(x) на m(x)[i]
541                     break;
542                 }
543                 case 7: { // Убираем подсветку, обнуляет переменную цикла
544                     state = 5; // Расчет остатков при делении a(x) на m(x)[i]
545                     break;
546                 }
547                 case 8: { // Расчет остатков при делении b(x) на m(x)[i]
548                     if (stack.popBoolean()) {
549                         state = 9; // Считает соответствующий остаток
550                     } else {
551                         state = 7; // Убираем подсветку, обнуляет переменную цикла
552                     }
553                     break;
554                 }
555                 case 9: { // Считает соответствующий остаток
556                     state = 8; // Расчет остатков при делении b(x) на m(x)[i]
557                     break;
558                 }
559                 case 10: { // Убираем подсветку, обнуляет переменную цикла
560                     state = 8; // Расчет остатков при делении b(x) на m(x)[i]
561                     break;
562                 }
563                 case 11: { // Расчет s(x)[i]
564                     if (stack.popBoolean()) {
565                         state = 12; // Считает соответствующий s(x)[i]
566                     } else {
567                         state = 10; // Убираем подсветку, обнуляет переменную цикла
568                     }
569                     break;
570                 }
571                 case 12: { // Считает соответствующий s(x)[i]
572                     state = 11; // Расчет s(x)[i]
573                     break;
574                 }
575                 case 13: { // Убираем подсветку, обнуляет переменную цикла
576                     state = 11; // Расчет s(x)[i]
577                     break;
578                 }
579                 case 14: { // Считает соответствующий c(x)[i]
580                     state = 13; // Убираем подсветку, обнуляет переменную цикла
581                     break;
582                 }
583                 case 15: { // Убираем подсветку, обнуляет переменную цикла
584                     state = 14; // Считает соответствующий c(x)[i]
585                     break;
586                 }
587                 case 16: { // Расчет s(x)[i]
588                     if (stack.popBoolean()) {
589                         state = 17; // Считает s(x)[i]\u00d7c(x)[i]
590                     } else {
591                         state = 15; // Убираем подсветку, обнуляет переменную цикла
592                     }
593                     break;
594                 }
595                 case 17: { // Считает s(x)[i]\u00d7c(x)[i]
596                     state = 16; // Расчет s(x)[i]
597                     break;
598                 }
599                 case 18: { // Убираем подсветку, обнуляет переменную цикла
600                     state = 16; // Расчет s(x)[i]
601                     break;
```

```
602                 }
603                 case 19: { // Выводит результат
604                     state = 18; // Убираем подсветку, обнуляет переменную цикла
605                     break;
606                 }
607                 case 20: { // Говорит о преимуществах алгоритма
608                     state = 19; // Выводит результат
609                     break;
610                 }
611                 case END_STATE: { // Начальное состояние
612                     state = 20; // Говорит о преимуществах алгоритма
613                     break;
614                 }
615             }
616         }
617 
618         /**
619           * Комментарий к текущему состоянию
620           */
621         public String getComment() {
622             String comment = "";
623             Object[] args = null;
624             // Выбор комментария
625             switch (state) {
626                 case START_STATE: { // Начальное состояние
627                     comment = VinogradAlgorithm.this.getComment("Main.START_STATE");
628                     break;
629                 }
630                 case 1: { // Инициализация автомата
631                     comment = VinogradAlgorithm.this.getComment("Main.InitialStep");
632                     break;
633                 }
634                 case 2: { // Выбираем степень полинома m(x)
635                     comment = VinogradAlgorithm.this.getComment("Main.DegreeChanging");
636                     args = new Object[]{new Integer(d.a.getDeg()),new Integer(d.b.getDeg()),
                            new Integer(d.a.getDeg()+d.b.getDeg())};
637                     break;
638                 }
639                 case 3: { // Подбираем полином, являющийся произведением взаимнопростых
640                     comment = VinogradAlgorithm.this.getComment("Main.ModuleChoose");
641                     break;
642                 }
643                 case 4: { // Убираем подсветку, обнуляет переменную цикла
644                     comment = VinogradAlgorithm.this.getComment("Main.LetCalcAmodM");
645                     args = new Object[]{new Integer(d.k)};
646                     break;
647                 }
648                 case 6: { // Считает соответствующий остаток
649                     comment = VinogradAlgorithm.this.getComment("Main.FindModA");
650                     args = new Object[]{new Integer(d.j)};
651                     break;
652                 }
653                 case 7: { // Убираем подсветку, обнуляет переменную цикла
654                     comment = VinogradAlgorithm.this.getComment("Main.LetCalcBmodM");
655                     args = new Object[]{new Integer(d.k)};
656                     break;
657                 }
658                 case 9: { // Считает соответствующий остаток
659                     comment = VinogradAlgorithm.this.getComment("Main.FindModB");
660                     args = new Object[]{new Integer(d.j)};
661                     break;
662                 }
663                 case 10: { // Убираем подсветку, обнуляет переменную цикла
664                     comment = VinogradAlgorithm.this.getComment("Main.LetCalcSx");
665                     args = new Object[]{new Integer(d.k)};
666                     break;
667                 }
668                 case 12: { // Считает соответствующий s(x)[i]
669                     comment = VinogradAlgorithm.this.getComment("Main.FindSxModM");
670                     args = new Object[]{new Integer(d.j), new Integer(d.ax[d.j-1].size()*d.bx[
                            d.j-1].size())};
671                     break;
672                 }
673                 case 13: { // Убираем подсветку, обнуляет переменную цикла
674                     comment = VinogradAlgorithm.this.getComment("Main.LetCalcCx");
675                     break;
```

```
676                 }
677                 case 14: { // Считает соответствующий c(x)[i]
678                     comment = VinogradAlgorithm.this.getComment("Main.FindCx");
679                     args = new Object[]{new Integer(d.k)};
680                     break;
681                 }
682                 case 15: { // Убираем подсветку, обнуляет переменную цикла
683                     comment = VinogradAlgorithm.this.getComment("Main.LetCalcCSx");
684                     args = new Object[]{new Integer(d.k)};
685                     break;
686                 }
687                 case 17: { // Считает s(x)[i]\u00d7c(x)[i]
688                     comment = VinogradAlgorithm.this.getComment("Main.FindSxCx");
689                     args = new Object[]{new Integer(d.j)};
690                     break;
691                 }
692                 case 18: { // Убираем подсветку, обнуляет переменную цикла
693                     comment = VinogradAlgorithm.this.getComment("Main.LetCalcRezult");
694                     args = new Object[]{new Integer(d.a.getDeg()+d.b.getDeg())};
695                     break;
696                 }
697                 case 19: { // Выводит результат
698                     comment = VinogradAlgorithm.this.getComment("Main.Rezult");
699                     args = new Object[]{d.rez.toStr(), new Integer(d.a.getDeg()+d.b.getDeg()),
                          new Integer(d.a.getDeg()), new Integer(d.b.getDeg()), new Double(Math
                         .round(d.a.getCoef(d.a.size()-1)*d.b.getCoef(d.b.size()-1)*100)/100.0)
                         };
700                     break;
701                 }
702                 case 20: { // Говорит о преимуществах алгоритма
703                     comment = VinogradAlgorithm.this.getComment("Main.Finish");
704                     args = new Object[]{new Integer(d.multCount), new Integer(d.a.size()*d.b.
                         size())};
705                     break;
706                 }
707                 case END_STATE: { // Конечное состояние
708                     comment = VinogradAlgorithm.this.getComment("Main.END_STATE");
709                     break;
710                 }
711             }
712 
713             return java.text.MessageFormat.format(comment, args);
714         }
715 
716         /**
717           * Выполняет действия по отрисовке состояния
718           */
719         public void drawState() {
720             switch (state) {
721                 case START_STATE: { // Начальное состояние
722                     if(d.ax != null) {
723                         d.applet.updateColumns();
724                         d.applet.aCol.unHighlightAll();
725                         d.applet.bCol.unHighlightAll();
726                         d.applet.cCol.unHighlightAll();
727                         d.applet.kCol.unHighlightAll();
728                         d.applet.rezCol.unHighlightAll();
729                         d.applet.multCol.unHighlightAll();
730                     }
731                     break;
732                 }
733                 case 1: { // Инициализация автомата
734                     d.applet.init();
735                     d.applet.updateColumns();
736                     break;
737                 }
738                 case 2: { // Выбираем степень полинома m(x)
739                     d.applet.aCol.unHighlightAll();
740                     d.applet.updateColumns();
741                     break;
742                 }
743                 case 3: { // Подбираем полином, являющийся произведением взаимнопростых
744                     if(d.mx[0]==null) d.applet.readModul();
745                     d.applet.updateColumns();
746                     d.applet.aCol.highlightAll();
747                     break;
```

```
748                 }
749                 case 4: { // Убираем подсветку, обнуляет переменную цикла
750                     d.applet.aCol.unHighlightAll();
751                     d.applet.bCol.unHighlightAll();
752                     d.applet.updateColumns();
753                     break;
754                 }
755                 case 6: { // Считает соответствующий остаток
756                     d.applet.aCol.unHighlightAll();
757                     d.applet.bCol.unHighlightAll();
758                     d.applet.aCol.highlight(d.j);
759                     d.applet.bCol.highlight(d.j);
760                     d.applet.updateColumns();
761                     break;
762                 }
763                 case 7: { // Убираем подсветку, обнуляет переменную цикла
764                     d.applet.aCol.unHighlightAll();
765                     d.applet.bCol.unHighlightAll();
766                     d.applet.cCol.unHighlightAll();
767                     d.applet.updateColumns();
768                     break;
769                 }
770                 case 9: { // Считает соответствующий остаток
771                     d.applet.aCol.unHighlightAll();
772                     d.applet.cCol.unHighlightAll();
773                     d.applet.aCol.highlight(d.j);
774                     d.applet.cCol.highlight(d.j);
775                     d.applet.updateColumns();
776                     break;
777                 }
778                 case 10: { // Убираем подсветку, обнуляет переменную цикла
779                     d.applet.aCol.unHighlightAll();
780                     d.applet.bCol.unHighlightAll();
781                     d.applet.cCol.unHighlightAll();
782                     d.applet.updateColumns();
783                     break;
784                 }
785                 case 12: { // Считает соответствующий s(x)[i]
786                     d.applet.aCol.unHighlightAll();
787                     d.applet.bCol.unHighlightAll();
788                     d.applet.cCol.unHighlightAll();
789 
790                     d.applet.aCol.highlight(d.j);
791                     d.applet.bCol.highlight(d.j);
792                     d.applet.cCol.highlight(d.j);
793 
794                     d.applet.updateColumns();
795                     break;
796                 }
797                 case 13: { // Убираем подсветку, обнуляет переменную цикла
798                     d.applet.aCol.unHighlightAll();
799                     d.applet.bCol.unHighlightAll();
800                     d.applet.cCol.unHighlightAll();
801                     d.applet.updateColumns();
802                     break;
803                 }
804                 case 14: { // Считает соответствующий c(x)[i]
805                     d.applet.aCol.unHighlightAll();
806                     d.applet.bCol.unHighlightAll();
807                     d.applet.cCol.unHighlightAll();
808                     d.applet.bCol.highlightAll();
809                     d.applet.updateColumns();
810                     break;
811                 }
812                 case 15: { // Убираем подсветку, обнуляет переменную цикла
813                     d.applet.aCol.unHighlightAll();
814                     d.applet.bCol.unHighlightAll();
815                     d.applet.cCol.unHighlightAll();
816                     d.applet.updateColumns();
817                     break;
818                 }
819                 case 17: { // Считает s(x)[i]\u00d7c(x)[i]
820                     d.applet.aCol.unHighlightAll();
821                     d.applet.bCol.unHighlightAll();
822                     d.applet.cCol.unHighlightAll();
823 
```

```
824                     d.applet.aCol.highlight(d.j);
825                     d.applet.bCol.highlight(d.j);
826                     d.applet.cCol.highlight(d.j);
827 
828                     d.applet.updateColumns();
829                     break;
830                 }
831                 case 18: { // Убираем подсветку, обнуляет переменную цикла
832                     d.applet.aCol.unHighlightAll();
833                     d.applet.bCol.unHighlightAll();
834                     d.applet.cCol.highlightAll();
835                     d.applet.rezCol.unHighlightAll();
836                     d.applet.updateColumns();
837                     break;
838                 }
839                 case 19: { // Выводит результат
840                     d.applet.rezCol.highlight(1);
841                     d.applet.cCol.unHighlightAll();
842                     d.applet.multCol.unHighlightAll();
843                     d.applet.updateColumns();
844                     break;
845                 }
846                 case 20: { // Говорит о преимуществах алгоритма
847                     d.applet.rezCol.highlight(1);
848                     d.applet.multCol.highlight(1);
849                     d.applet.cCol.unHighlightAll();
850                     d.applet.updateColumns();
851                     break;
852                 }
853                 case END_STATE: { // Конечное состояние
854                     d.applet.rezCol.unHighlightAll();
855                     d.applet.multCol.unHighlightAll();
856                     break;
857                 }
858             }
859         }
860     }
861 }
```

Е. Исходные коды интерфейса визуализатора

Е.1. Исходный текст файла `VinogradVisualizer.java`

```
1  package ru.ifmo.vizi.vinograd;
2
3  import ru.ifmo.vizi.base.Base;
4  import ru.ifmo.vizi.base.SmartTokenizer;
5  import ru.ifmo.vizi.base.VisualizerParameters;
6  import ru.ifmo.vizi.base.auto.AutomataController;
7  import ru.ifmo.vizi.base.ui.AutoControlsPane;
8  import ru.ifmo.vizi.base.ui.HintedButton;
9  import ru.ifmo.vizi.base.ui.SaveLoadDialog;
10 import ru.ifmo.vizi.base.widgets.Rect;
11 import ru.ifmo.vizi.base.widgets.ShapeLook;
12 import ru.ifmo.vizi.base.widgets.ShapeStyle;
13
14 import java.awt.*;
15 import java.awt.event.*;
16 import java.util.StringTokenizer;
17 import java.util.Vector;
18
19 /**
20  * Реализует визуализацию алгоритма Винограда.
21  *
22  * @author Alexandr Smal
23  * @version $Id: 0.0$
24  */
25 public final class VinogradVisualizer extends Base implements ActionListener, ItemListener,
        KeyListener, FocusListener {
26
27     /**
28      * Отвечает за реализацию и отображение колонок.
29      */
30     public final class Column extends Vector {
31         /**
32          * Координата левого верхнего угла.
33          */
34         private Point luCorn;
35
36         /**
37          * Координата правого нижнего угла.
38          */
39         private Point rdCorn;
40
41         /**
42          * Текст метки.
43          */
44         private String Label = "";
45
46         /**
47          * Возвращает текст метки.
48          *
49          * @return текст метки.
50          */
51         public String getLabel() {
52             return Label;
53         }
54
55         /**
56          * Утановить текст метки.
57          *
58          * @param label текст метки.
59          */
60         public void setLabel(String label) {
61             Label = label;
62             if (size() > 0) {
63                 element(0).setMessage(Label);
64             }
65         }
66
67         /**
68          * Установить количество ячеек.
69          *
70          * @param length количество ячеек.
```

```
71          */
72         public void setLength(int length) {
73             removeAllElements();
74             for (int i = 0; i <= length; i++) push("");
75             element(0).setStyle(1);
76             element(0).setMessage(Label);
77         }
78 
79         /**
80          * Конструктор по умолчанию.
81          *
82          * @param style стили прямоугольников.
83          */
84         public Column(ShapeStyle[] style) {
85             super();
86             styleSet = style;
87             push(Label);
88             element(0).setStyle(1);
89         }
90 
91         /**
92          * Конструктор по умолчанию.
93          *
94          * @param style  стили прямоугольников.
95          * @param length количество ячеек.
96          */
97         public Column(ShapeStyle[] style, int length) {
98             this(style);
99             for (int i = 0; i < length; i++) push("");
100        }
101
102        /**
103         * Стили прямоугольников.
104         */
105        private final ShapeStyle[] styleSet;
106
107        /**
108         * Возвращает ячейку.
109         *
110         * @param i номер ячейки.
111         */
112        public Rect element(int i) {
113            if (i >= size()) {
114                return null;
115            } else {
116                return (Rect) elementAt(i);
117            }
118        }
119
120        /**
121         * Устанавливает координаты.
122         *
123         * @param lu координата левого верхнего угла.
124         * @param rd координата правого нижнего угла.
125         */
126        public void setCoords(Point lu, Point rd) {
127            element(0).setMessage(Label);
128            luCorn = lu;
129            rdCorn = rd;
130            renewcoords();
131        }
132
133        /**
134         * Обновляет координаты всех Rect-ов.
135         */
136        public void renewcoords() {
137            if (size() > 0) {
138                int rWidth = rdCorn.x - luCorn.x;
139                int rHeight = (int) ((rdCorn.y - luCorn.y) / (size() + 0.0));
140                for (int i = size() - 1; i > 0; i--) {
141                    element(i).setSize(rWidth, rHeight);
142                    element(i).setLocation(luCorn.x, rdCorn.y - (size() - i) * rHeight);
143                }
144                element(0).setSize(rWidth, (rdCorn.y - luCorn.y) - (size() - 1) * rHeight);
145                element(0).setLocation(luCorn);
146            }
```

```
147             clientPane.repaint();
148         }
149 
150         /**
151          * Втолкнуть новую ячейку с строкой s.
152          *
153          * @param s текст на этой ячейке.
154          */
155         private void push(String s) {
156             Rect r = new Rect(styleSet, s);
157             insertElementAt(r, 0);
158             r.setStyle(0);
159             clientPane.add(r);
160         }
161 
162         /**
163          * Подсветить i-й элемент.
164          *
165          * @param i номер элемента.
166          */
167         public int highlight(int i) {
168             if (size() <= i) return 1;
169             element(i).setStyle((i == 0) ? 3 : 2);
170             return 0;
171         }
172 
173         /**
174          * Убрать подсветку i-го элемента.
175          *
176          * @param i номер элемента.
177          */
178         public int unHighlight(int i) {
179             if (size() < i) return 1;
180             ((Rect) elementAt(i)).setStyle(0);
181             return 0;
182         }
183 
184         /**
185          * Выделяет все элементы (кроме шапки).
186          */
187         public void highlightAll() {
188             for (int i = 1; i < size(); i++) {
189                 highlight(i);
190             }
191         }
192 
193         /**
194          * Снимает выделение всех элементов.
195          */
196         public void unHighlightAll() {
197             for (int i = 1; i < size(); i++) {
198                 unHighlight(i);
199             }
200         }
201 
202         /**
203          * Установить вектор в ячейку.
204          *
205          * @param i   - номер ячейки.
206          * @param vec вектор
207          */
208         public void setVector(dVector vec, int i) {
209             element(i).setMessage((vec == null) ? "" : vec.toStr());
210         }
211 
212         /**
213          * Возвращает размер шрифта ячейки с номером i.
214          *
215          * @param i номер ячейки.
216          * @return размер шрифта.
217          */
218         public int getFontSize(int i) {
219             Rect r = new Rect(this.styleSet);
220             r.setStyle(element(i).getStyle());
221             r.setSize(element(i).getSize());
222             r.setMessage(element(i).getMessage());
```

```
223             r.adjustFontSize();
224             return r.getLook().getTextFont(element(i).getStyleSet()[element(i).getStyle()]).
                    getSize();
225         }
226 
227         /**
228          * Устанавливает размер шрифта у i-ой ячейки.
229          *
230          * @param i    номер ячейки.
231          * @param size размер шрифта.
232          */
233         public void setFontSize(int i, int size) {
234             ShapeLook look = element(i).getLook();
235             Font font = look.getTextFont(element(i).getStyleSet()[element(i).getStyle()]);
236             look.setTextFont(new Font(font.getName(), font.getStyle(), size));
237         }
238 
239         /**
240          * Устанавливает размер шрифта всех ячеек, кроме первой (там, где метка).
241          *
242          * @param size размер шрифта.
243          */
244         public void setFontSize(int size) {
245             for (int i = 1; i < size(); i++) {
246                 setFontSize(i, size);
247             }
248         }
249     }
250 
251     /**
252      * Экземпляр автомата.
253      */
254     private VinogradAlgorithm auto;
255 
256     /**
257      * Экземпляр класса хранящего всю информацию.
258      */
259     private final VinogradAlgorithm.Data data;
260 
261     /**
262      * Save/load dialog.
263      */
264     private SaveLoadDialog saveLoadDialog;
265 
266     /**
267      * Поле для входного выражения a(x).
268      */
269     private HintedTextField InputFielda;
270 
271     /**
272      * Поле для входного выражения b(x).
273      */
274     private HintedTextField InputFieldb;
275 
276     /**
277      * Choice список (с хинтами), в котором выбираются примеры.
278      */
279     private HintedChoice Examples;
280 
281     /**
282      * Верхний столбец номеров.
283      */
284     public Column kCol;
285 
286     /**
287      * Колонка для счетчика числа умножений.
288      */
289     public Column multCol;
290 
291     /**
292      * Столбец полинома a(x).
293      */
294     public Column aCol;
295 
296     /**
297      * Столбец полинома b(x).
```

```
298         */
299        public Column bCol;
300
301        /**
302         * Столбец полинома m(x).
303         */
304        public Column cCol;
305
306        /**
307         * Место для результата.
308         */
309        public Column rezCol;
310
311        /**
312         * Массив, содержащий колонки aCol, bCol, cCol и kCol.
313         */
314        public Column Columns[] = new Column[4];
315
316        /**
317         * Массив, содержащий вообще все колонки.
318         */
319        public Column AllColumns[] = new Column[6];
320
321        /**
322         * Deault contructor.
323         */
324        public VinogradVisualizer(VisualizerParameters parameters) {
325            super(parameters);
326            auto = new VinogradAlgorithm(locale);
327            createInterface(auto);
328            data = auto.d;
329            data.applet = this;
330            aCol = new Column(ShapeStyle.loadStyleSet(config, "arrayCol"));
331            bCol = new Column(ShapeStyle.loadStyleSet(config, "arrayCol"));
332            cCol = new Column(ShapeStyle.loadStyleSet(config, "arrayCol"));
333            kCol = new Column(ShapeStyle.loadStyleSet(config, "kCol"));
334            rezCol = new Column(ShapeStyle.loadStyleSet(config, "rezCol"), 1);
335            multCol = new Column(ShapeStyle.loadStyleSet(config, "multCol"), 1);
336
337            aCol.setLabel(config.getParameter("label-ax", "a(x)"));
338            bCol.setLabel(config.getParameter("label-bx", "b(x)"));
339            cCol.setLabel(config.getParameter("label-mx", "m(x)"));
340            kCol.setLabel(config.getParameter("label-k", "k"));
341            rezCol.setLabel(config.getParameter("label-rez", "Result:"));
342            multCol.setLabel(config.getParameter("label-mult", "Real multiplications:"));
343
344            Columns[0] = aCol;
345            Columns[1] = bCol;
346            Columns[2] = cCol;
347            Columns[3] = kCol;
348
349            AllColumns[0] = aCol;
350            AllColumns[1] = bCol;
351            AllColumns[2] = cCol;
352            AllColumns[3] = kCol;
353            AllColumns[4] = rezCol;
354            AllColumns[5] = multCol;
355
356            Examples.select(1);
357            init();
358            updateColumns();
359        }
360
361        /**
362         * Установить кол-во ячеек во всех колонках равным k.
363         *
364         * @param k кол-во ячеек во всех колонках.
365         */
366        private void setK(int k) {
367            clientPane.removeAll();
368            rezCol.setLength(1);
369            multCol.setLength(1);
370            aCol.setLength(k);
371            bCol.setLength(k);
372            cCol.setLength(k);
373            kCol.setLength(k);
```

```
374         data.k = k;
375         for (int i = 1; i <= k; i++) {
376             kCol.element(i).setMessage("" + i);
377         }
378         layoutClientPane(clientPane.getSize().width, clientPane.getSize().height);
379         data.ax = new dVector[k];
380         data.bx = new dVector[k];
381         data.sx = new dVector[k];
382         data.cx = new dVector[k];
383         data.csx = new dVector[k];
384         data.mx = new dVector[k];
385     }
386 
387     /**
388      * Инициализирует начальные вектора.
389      */
390     public void init() {
391         if (Examples.getSelectedIndex() == 0) {
392             init(InputFielda.getText(), InputFieldb.getText());
393         } else {
394             init(config.getParameter("Examples-example-" + Examples.getSelectedIndex()));
395         }
396     }
397 
398     /**
399      * Инициализирует начальные вектора из строки, где они
400      * записанны через ';'.
401      *
402      * @param s строка с векторами.
403      */
404     private void init(String s) {
405         StringTokenizer tok = new StringTokenizer(s, ";");
406         init(tok.nextToken(), tok.nextToken());
407     }
408 
409     /**
410      * Инициализирует начальные вектора из строк.
411      *
412      * @param a строка с вектором a(x).
413      * @param b строка с вектором b(x).
414      */
415     private void init(String a, String b) {
416         data.a = new dVector(a);
417         InputFielda.setText(data.a.toCSVStr());
418         data.b = new dVector(b);
419         InputFieldb.setText(data.b.toCSVStr());
420         StringTokenizer tok = new StringTokenizer(config.getParameter("Modul-" +
421                                                                       (data.a.getDeg() + data.
                                                                           b.getDeg())), ";");
422         data.k = tok.countTokens();
423         setK(data.k);
424     }
425 
426     /**
427      * This method must create panel with applet's controls.
428      *
429      * @return created pane.
430      */
431     public Component createControlsPane() {
432         Panel panel = new Panel(new BorderLayout());
433         panel.add(new AutoControlsPane(config, auto, forefather, false), BorderLayout.CENTER);
434         Panel bottomPanel = new Panel();
435         if (config.getBoolean("button-ShowSaveLoad")) {
436             bottomPanel.add(new HintedButton(config, "button-SaveLoad") {
437                 protected void click() {
438                     saveLoadDialog.center();
439                     StringBuffer buffer = new StringBuffer();
440                     buffer.append("/* ").append(config.getParameter("SaveLoadDialog-comments-
                           axLength")).append(" */\n");
441                     buffer.append("axLength = ");
442                     buffer.append(data.a.size());
443 
444                     buffer.append("\n/* ").append(config.getParameter("SaveLoadDialog-comments
                           -ax")).append(" */\n");
445                     buffer.append("ax = ");
446                     for (int i = data.a.size() - 1; i >= 0; i--) {
```

```
447                     buffer.append(data.a.elementAt(i)).append(" ");
448                 }
449                 buffer.append("\n/* ").append(config.getParameter("SaveLoadDialog-comments
                        -bxLength")).append(" */\n");
450                 buffer.append("bxLength = ");
451                 buffer.append(data.b.size());
452 
453                 buffer.append("\n/* ").append(config.getParameter("SaveLoadDialog-comments
                        -bx")).append(" */\n");
454                 buffer.append("bx = ");
455                 for (int i = data.b.size() - 1; i >= 0; i--) {
456                     buffer.append(data.b.elementAt(i)).append(" ");
457                 }
458 
459                 buffer.append("\n/* ").append(config.getParameter("SaveLoadDialog-comments
                        -step")).append(" */\n");
460                 buffer.append("Step = ").append(auto.getStep());
461                 saveLoadDialog.show(buffer.toString());
462             }
463         });
464     }
465     panel.add(bottomPanel, BorderLayout.SOUTH);
466 
467     saveLoadDialog = new SaveLoadDialog(config, forefather) {
468         public boolean load(String text) throws Exception {
469             SmartTokenizer tokenizer = new SmartTokenizer(text, config);
470             while (!auto.isAtStart()) auto.stepBackward(1);
471             tokenizer.expect("axLength");
472             tokenizer.expect("=");
473             int aLength = tokenizer.nextInt(1, config.getInteger("max-polynom-deg") + 1);
474 
475             tokenizer.expect("ax");
476             tokenizer.expect("=");
477             data.a = new dVector();
478             for (int i = 0; i < aLength; i++) {
479                 data.a.insertElementAt(new Double(tokenizer.nextDouble(-20, +20)), 0);
480             }
481             tokenizer.expect("bxLength");
482             tokenizer.expect("=");
483             int bLength = tokenizer.nextInt(1, config.getInteger("max-polynom-deg") + 1);
484 
485             tokenizer.expect("bx");
486             tokenizer.expect("=");
487             data.b = new dVector();
488             for (int i = 0; i < bLength; i++) {
489                 data.b.insertElementAt(new Double(tokenizer.nextDouble(-20, +20)), 0);
490             }
491             tokenizer.expect("Step");
492             tokenizer.expect("=");
493             int step = tokenizer.nextInt();
494             tokenizer.expectEOF();
495 
496             Examples.select(0);
497             data.applet.init(data.a.toCSVStr(), data.b.toCSVStr());
498             while (!auto.isAtEnd() && auto.getStep() < step) auto.stepForward(0);
499             return true;
500         }
501     };
502     Label axLabel = new Label(config.getParameter("InputFielda-label"));
503     Label bxLabel = new Label(config.getParameter("InputFieldb-label"));
504     axLabel.setAlignment(2);
505     bxLabel.setAlignment(2);
506     bottomPanel.add(axLabel);
507     InputFielda = new HintedTextField(config, "InputFielda", 10);
508     InputFielda.addActionListener(this);
509     InputFielda.addKeyListener(this);
510     InputFielda.addFocusListener(this);
511     bottomPanel.add(InputFielda);
512     bottomPanel.add(bxLabel);
513     InputFieldb = new HintedTextField(config, "InputFieldb", 10);
514     InputFieldb.addActionListener(this);
515     InputFieldb.addKeyListener(this);
516     InputFieldb.addFocusListener(this);
517     bottomPanel.add(InputFieldb);
518     Examples = new HintedChoice(config, "Examples");
519     Examples.addItemListener(this);
```

```
520         bottomPanel.add(Examples);
521         panel.add(bottomPanel, BorderLayout.SOUTH);
522         return panel;
523     }
524 
525     /**
526      * Вызывается, если произойдет действие с <CODE>InputFielda</CODE>
527      * или <CODE>InputFieldb</CODE>.
528      *
529      * @param e событие.
530      */
531     public void actionPerformed(ActionEvent e) {
532         if (isInputCorrect()) {
533             while (!auto.isAtStart()) auto.stepBackward(1);
534             init();
535             updateColumns();
536         } else {
537             showErrorMessage();
538             if (!isCorrectVector(InputFielda.getText())) InputFielda.setText("0.0");
539             if (!isCorrectVector(InputFieldb.getText())) InputFieldb.setText("0.0");
540         }
541     }
542 
543     /**
544      * Вызывается, если изменится выбранный элемент в
545      * <CODE>Examples</CODE>.
546      */
547     public void itemStateChanged(ItemEvent e) {
548         while (!auto.isAtStart()) auto.stepBackward(1);
549         init();
550         updateColumns();
551     }
552 
553     /**
554      * Вызывается, когда в <CODE>InputFielda</CODE> или <CODE>InputFieldb</CODE>
555      * нажимают кнопку.
556      *
557      * @param ke кнопочное событие.
558      */
559     public void keyPressed(KeyEvent ke) {
560         char ch = ke.getKeyChar();
561         if (!((ch >= '0' && ch <= '9') || (ch == '.') || (ch == ',') || (ch == 8) || (ch ==
                10) || (ch == 65535) || (ch == 127) || (ch == '+') || ch == '-')) {
562             ke.consume();
563         }
564     }
565 
566     /**
567      * Вызывается, когда в <CODE>InputFielda</CODE> или <CODE>InputFieldb</CODE>
568      * отпускают кнопку.
569      *
570      * @param ke кнопочное событие.
571      */
572     public void keyReleased(KeyEvent ke) {
573     }
574 
575     /**
576      * Вызывается, когда в <CODE>InputFielda</CODE> или <CODE>InputFieldb</CODE>
577      * уже набран символ.
578      *
579      * @param ke кнопочное событие.
580      */
581     public void keyTyped(KeyEvent ke) {
582     }
583 
584     /**
585      * Вызывается, когда <CODE>InputFielda</CODE> или <CODE>InputFieldb</CODE>
586      * получает фокус.
587      *
588      * @param fe фокусное событие.
589      */
590     public void focusGained(FocusEvent fe) {
591         auto.getController().setEnabled(false);
592         Examples.select(0);
593     }
594 
```

```
595     /**
596      * Вызывается, когда <CODE>InputFielda</CODE> или <CODE>InputFieldb</CODE>
597      * теряет фокус.
598      *
599      * @param fe фокусное событие.
600      */
601     public void focusLost(FocusEvent fe) {
602         if (isInputCorrect()) {
603             auto.getController().setEnabled(true);
604             while (!auto.isAtStart()) auto.stepBackward(1);
605             init();
606             updateColumns();
607         } else if (!fe.isTemporary()) {
608             showErrorMessage();
609             ((HintedTextField) (fe.getComponent())).requestFocus();
610         }
611     }
612 
613     /**
614      * Проверяет правильность ввода данных. В случае их некорректности возвращает <CODE>false
           </CODE>.
615      *
616      * @return правильные ли даные.
617      */
618     private boolean isInputCorrect() {
619         String a = InputFielda.getText();
620         String b = InputFieldb.getText();
621         int max = config.getInteger("max-polynom-deg");
622         if (a.length() == 0) {
623             InputFielda.setText("0.0");
624             a = "0.0";
625         }
626         if (b.length() == 0) {
627             InputFieldb.setText("0.0");
628             b = "0.0";
629         }
630         if (!isCorrectVector(a) || !isCorrectVector(b))
631             return false;
632         if ((new dVector(a)).getDeg() > max || (new dVector(b)).getDeg() > max)
633             return false;
634         return true;
635     }
636 
637     /**
638      * Проверяет, может ли данная строка быть интерпретирована как полином.
639      *
640      * @param s строка с коэффициентами.
641      * @return правильна ли строка.
642      */
643     private boolean isCorrectVector(String s) {
644         if (s.length() == 0) return false;
645         char cf = s.charAt(0);
646         char cl = s.charAt(s.length() - 1);
647         if (cl == ',' || cf == ',' || cl == '.' || cf == '.' || cl == '-')
648             return false;
649         for (int i = 1; i < s.length(); i++) {
650             if (s.charAt(i) == '-' && s.charAt(i - 1) != ',')
651                 return false;
652             if ((s.charAt(i) == '.' || s.charAt(i) == ',') && (s.charAt(i - 1) == ',' || s.
                    charAt(i - 1) == '.' || s.charAt(i - 1) == '-'))
653                 return false;
654         }
655         return true;
656     }
657 
658     /**
659      * Показывает диалоговое окно с сообщением об ошибке ввода.
660      */
661     private void showErrorMessage() {
662         auto.getController().setExecutionMode(AutomataController.MANUAL_MODE);
663         StringBuffer s = new StringBuffer();
664         if (!isCorrectVector(InputFielda.getText())) {
665             s.append(config.getParameter("error-ax"));
666         }
667         if (!isCorrectVector(InputFieldb.getText())) {
668             if (s.length() == 0) {
```

```
669                 s.append(config.getParameter("error-bx"));
670             } else {
671                 s.append(", ").append(config.getParameter("error-bx"));
672             }
673         }
674         ErrorDialog errDlg = new ErrorDialog(config, forefather, s.toString());
675         Thread t = new Thread(errDlg);
676         t.start();
677     }
678 
679     /**
680      * Invoked when client pane shoud be layouted.
681      *
682      * @param clientWidth  client pane width.
683      * @param clientHeight client pane height.
684      */
685     protected void layoutClientPane(int clientWidth, int clientHeight) {
686         int hspace = clientWidth / 100;
687         int vspace = clientHeight / 50;
688         kCol.setCoords(new Point(hspace, vspace), new Point(clientWidth / 14 - hspace, (data.k
                + 1) * clientHeight / (data.k + 3) - vspace));
689         rezCol.setCoords(new Point(hspace, (data.k + 1) * clientHeight / (data.k + 3)), new
                Point(7 * clientWidth / 14 - hspace / 2, clientHeight - vspace));
690         multCol.setCoords(new Point(7 * clientWidth / 14 + hspace / 2, (data.k + 1) *
                clientHeight / (data.k + 3)), new Point(14 * clientWidth / 14 - hspace,
                clientHeight - vspace));
691 
692         if (data.sx != null && data.sx[0] != null) {
693             if (data.cx != null && data.cx[0] != null) {
694                 aCol.setCoords(new Point(clientWidth / 14, vspace), new Point(5 * clientWidth
                        / 14 - hspace, (data.k + 1) * clientHeight / (data.k + 3) - vspace));
695                 bCol.setCoords(new Point(5 * clientWidth / 14, vspace), new Point(9 *
                        clientWidth / 14 - hspace, (data.k + 1) * clientHeight / (data.k + 3) -
                        vspace));
696                 cCol.setCoords(new Point(9 * clientWidth / 14, vspace), new Point(14 *
                        clientWidth / 14 - hspace, (data.k + 1) * clientHeight / (data.k + 3) -
                        vspace));
697             } else {
698                 aCol.setCoords(new Point(clientWidth / 14, vspace), new Point(5 * clientWidth
                        / 14 - hspace, (data.k + 1) * clientHeight / (data.k + 3) - vspace));
699                 bCol.setCoords(new Point(5 * clientWidth / 14, vspace), new Point(9 *
                        clientWidth / 14 - hspace, (data.k + 1) * clientHeight / (data.k + 3) -
                        vspace));
700                 cCol.setCoords(new Point(9 * clientWidth / 14, vspace), new Point(14 *
                        clientWidth / 14 - hspace, (data.k + 1) * clientHeight / (data.k + 3) -
                        vspace));
701             }
702         } else {
703             aCol.setCoords(new Point(clientWidth / 14, vspace), new Point(6 * clientWidth / 14
                     - hspace, (data.k + 1) * clientHeight / (data.k + 3) - vspace));
704             bCol.setCoords(new Point(6 * clientWidth / 14, vspace), new Point(10 * clientWidth
                     / 14 - hspace, (data.k + 1) * clientHeight / (data.k + 3) - vspace));
705             cCol.setCoords(new Point(10 * clientWidth / 14, vspace), new Point(14 *
                    clientWidth / 14 - hspace, (data.k + 1) * clientHeight / (data.k + 3) - vspace
                    ));
706         }
707         int minHeadFontSize = rezCol.getFontSize(0);
708         for (int i = 0; i < 6; i++)
709             if (minHeadFontSize > AllColumns[i].getFontSize(0)) minHeadFontSize = AllColumns[i
                    ].getFontSize(0);
710         for (int i = 0; i < 6; i++) AllColumns[i].setFontSize(0, minHeadFontSize);
711         int minVecFontSize = aCol.getFontSize(1);
712         for (int i = 0; i < 4; i++)
713             for (int j = 1; j <= data.k; j++) {
714                 if (minVecFontSize > Columns[i].getFontSize(j)) minVecFontSize = Columns[i].
                        getFontSize(j);
715             }
716         if (minVecFontSize > rezCol.getFontSize(1)) minVecFontSize = rezCol.getFontSize(1);
717         for (int i = 0; i < 4; i++) Columns[i].setFontSize((minVecFontSize > minHeadFontSize)
                ? minHeadFontSize : minVecFontSize);
718         rezCol.setFontSize((minVecFontSize > minHeadFontSize) ? minHeadFontSize :
                minVecFontSize);
719         multCol.setFontSize((minVecFontSize > minHeadFontSize) ? minHeadFontSize :
                minVecFontSize);
720     }
721 
```

```
722     /**
723      * Считывает m(x)[i] для данной степени результирующего полинома
724      * в <CODE>data.mx</CODE> из <CODE>config</CODE>.
725      */
726     public void readModul() {
727         StringTokenizer tok = new StringTokenizer(config.getParameter("Modul-" +
728                                                                       (data.a.getDeg() + data.
                                                                           b.getDeg())), ";");
729         data.mx = new dVector[data.k];
730         for (int i = 0; i < data.k; i++) {
731             data.mx[i] = new dVector(tok.nextToken());
732         }
733     }
734 
735     /**
736      * Обновляет надписи во всех колонках
737      * в зависимости от <CODE>data</CODE>.
738      */
739     public void updateColumns() {
740         multCol.element(1).setMessage("" + data.multCount);
741         for (int i = 1; i <= data.k; i++) {
742             if (data.sx != null && data.sx[0] != null) {
743                 if (data.cx != null && data.cx[0] != null) {
744 
745                     aCol.setVector(data.sx[i - 1], i);
746                     bCol.setVector(data.cx[i - 1], i);
747                     cCol.setVector(data.csx[i - 1], i);
748                     aCol.setLabel(config.getParameter("label-sx"));
749                     bCol.setLabel(config.getParameter("label-cx"));
750                     cCol.setLabel(config.getParameter("label-csx"));
751 
752                 } else {
753                     aCol.setLabel(config.getParameter("label-ax") + "=" + data.a.toStr());
754                     bCol.setLabel(config.getParameter("label-bx") + "=" + data.b.toStr());
755                     cCol.setLabel(config.getParameter("label-sx"));
756                     aCol.setVector(data.ax[i - 1], i);
757                     bCol.setVector(data.bx[i - 1], i);
758                     cCol.setVector(data.sx[i - 1], i);
759                 }
760             } else {
761                 aCol.setLabel(config.getParameter("label-mx"));
762                 bCol.setLabel(config.getParameter("label-ax") + "=" + data.a.toStr());
763                 cCol.setLabel(config.getParameter("label-bx") + "=" + data.b.toStr());
764 
765                 aCol.setVector(data.mx[i - 1], i);
766                 bCol.setVector(data.ax[i - 1], i);
767                 cCol.setVector(data.bx[i - 1], i);
768             }
769         }
770         rezCol.setVector(data.rez, 1);
771         layoutClientPane(clientPane.getSize().width, clientPane.getSize().height);
772 
773     }
774 
775 
776 }
```

E.2. Исходный текст файла dVector.java

```
1  package ru.ifmo.vizi.vinograd;
2 
3  import java.util.StringTokenizer;
4  import java.util.Vector;
5 
6  /**
7   * Created by IntelliJ IDEA.
8   * User: Alexandr V. Smal
9   * Date: 11.06.2004
10  * Time: 5:42:35
11  */
12 
13 /**
14  * Класс, инкапсулирующий вектор коэффициентов и операции с ним.
15  */
16 public final class dVector extends Vector {
17 
```

```
18      /**
19       * Конструктор по умолчанию.
20       */
21      public dVector() {
22          super();
23      }
24
25      /**
26       * Конструктор копирования.
27       *
28       * @param vec образ копирования.
29       */
30      public dVector(dVector vec) {
31          for (int i = 0; i < vec.size(); i++) {
32              addElement(new Double(vec.getCoef(i)));
33          }
34          killZeros();
35      }
36
37      /**
38       * Конструктор, создающий полином из строки с коэффициентами, разделенными
39       * запятыми.
40       *
41       * @param s строка с коэффициентами.
42       */
43      public dVector(String s) {
44          String st = s;
45          s = s.replace('[', ' ').replace(']', ' ');
46          StringTokenizer tokenizer = new StringTokenizer(st, ",");
47          double d = 0.0;
48          while (tokenizer.hasMoreTokens()) {
49              try {
50                  d = (new Double(tokenizer.nextToken())).doubleValue();
51              } catch (Exception e) {
52                  d = 0.0;
53              }
54              insertElementAt(new Double(d), 0);
55          }
56          killZeros();
57      }
58
59      /**
60       * Равен ли вектор нулю.
61       */
62      public boolean isZero() {
63          this.killZeros();
64          return (size() == 1 && Math.round(getCoef(0) * 100) / 100.0 == 0.0);
65      }
66
67
68      /**
69       * Установить коэффициент.
70       *
71       * @param x коэффициент.
72       * @param i номер коэффициента.
73       */
74      public void setCoef(double x, int i) {
75          setElementAt(new Double(x), i);
76      }
77
78      /**
79       * Возвращает значение коэффициента.
80       *
81       * @param i номер коэффициента.
82       * @return коэффициент.
83       */
84      public double getCoef(int i) {
85          if (i < size() && i >= 0) {
86              return ((Double) (elementAt(i))).doubleValue();
87          } else {
88              return 0.0;
89          }
90      }
91
92      /**
93       * Прибавляет к данному вектору другой и возвращает сумму.
```

```
94       *
95       * @param vec прибавляемый вектор.
96       * @return сумма этого вектора и vec.
97       */
98      public dVector addVec(final dVector vec) {
99          dVector rez = new dVector();
100         dVector a = toLenght(Math.max(size(), vec.size()));
101         dVector b = vec.toLenght(Math.max(size(), vec.size()));
102         for (int i = 0; i < a.size(); i++) {
103             rez.addElement(new Double(a.getCoef(i) + b.getCoef(i)));
104         }
105         return rez.killZeros();
106     }
107
108     /**
109      * Вычитает из данного вектора другой и возвращает разность.
110      *
111      * @param vec прибавляемый вектор.
112      * @return сумма этого вектора и vec.
113      */
114     public dVector subVec(final dVector vec) {
115         dVector rez = new dVector();
116         dVector a = toLenght(Math.max(size(), vec.size()));
117         dVector b = vec.toLenght(Math.max(size(), vec.size()));
118         for (int i = 0; i < a.size(); i++) {
119             rez.addElement(new Double(a.getCoef(i) - b.getCoef(i)));
120         }
121         return rez.killZeros();
122     }
123
124     /**
125      * Умножает данный вектор на другой как вектор коэффициентов и возвращает произведение.
126      *
127      * @param vec прибавляемый вектор.
128      * @return сумма этого вектора и vec.
129      */
130     public dVector multVec(dVector vec) {
131         dVector rez = new dVector();
132         for (int i = 0; i < (size() + vec.size() - 1); i++) {
133             rez.addElement(new Double(0.0));
134         }
135         for (int i = 0; i < size(); i++) {
136             for (int j = 0; j < vec.size(); j++) {
137                 rez.setCoef(rez.getCoef(i + j) + (getCoef(i) * vec.getCoef(j)), i + j);
138             }
139         }
140         return rez.killZeros();
141     }
142
143     /**
144      * Умножает вектор на число.
145      *
146      * @param x множитель.
147      * @return вектор, умноженный на число.
148      */
149     public dVector multNum(double x) {
150         dVector rez = new dVector();
151         for (int i = 0; i < size(); i++) {
152             rez.addElement(new Double(getCoef(i) * x));
153         }
154         return rez.killZeros();
155     }
156
157     /**
158      * Убирает нули ведущие нули, т.е. нули при старших коэффициенах.
159      *
160      * @return указатель на сам вектор.
161      */
162     public dVector killZeros() {
163         while (size() > 1 && Math.round(getCoef(size() - 1) * 100) / 100.0 == 0.0) {
164             removeElementAt(size() - 1);
165         }
166         return this;
167     }
168
169     /**
```

```
170      * Увеличивает длину вектора ведущими нулями. Если нужная
171      * длина меньше реальной, то с вектором ничего не происходит.
172      *
173      * @param i нужная длина.
174      * @return результат.
175      */
176     private dVector toLenght(int i) {
177         dVector rez = new dVector(this);
178         if (rez.size() < i) {
179             while (rez.size() < i) {
180                 rez.addElement(new Double(0.0));
181             }
182         }
183         return rez;
184     }
185 
186     /**
187      * Сдвигает вектор за счет младших разрядов, что бы достичь
188      * нужной длины (по сути домножает полином на x^k, что бы
189      * его степень стала равной i-1).
190      * Если нужная длина меньше, то с вектором ничего
191      * не происходит.
192      *
193      * @param i новая длина вектора.
194      * @return резилутат.
195      */
196     private dVector shift(int i) {
197         dVector rez = new dVector(this);
198         if (rez.size() < i) {
199             while (rez.size() < i) {
200                 rez.insertElementAt(new Double(0.0), 0);
201             }
202         }
203         return rez;
204     }
205 
206     /**
207      * Ищет остаток по модулю.
208      *
209      * @param mod вектор коэффициентов модуля.
210      * @return остаток.
211      */
212     public dVector mod(final dVector mod) {
213         mod.killZeros();
214         if (mod.size() > size()) {
215             return new dVector(this);
216         } else {
217             dVector a = new dVector(this);
218             while ((a.size() >= mod.size()) && !(a.isZero())) {
219                 a = a.subVec((mod.shift(a.size())).multNum(a.getCoef(a.size() - 1) / mod.
                        getCoef(mod.size() - 1)));
220                 a.killZeros();
221             }
222             return a;
223         }
224     }
225 
226     /**
227      * Ищет частное от деления.
228      *
229      * @param div вектор делителя.
230      * @return частное.
231      */
232     public dVector div(final dVector div) {
233         div.killZeros();
234         if (div.size() > size()) {
235             return new dVector("0.0");
236         } else {
237             dVector a = new dVector(this);
238             dVector rez = new dVector();
239             while (a.size() >= div.size() && !(a.isZero())) {
240                 double x = (1.0 * a.getCoef(a.size() - 1)) / div.getCoef(div.size() - 1);
241                 rez.insertElementAt(new Double(x), 0);
242                 a = a.subVec((div.shift(a.size())).multNum(x));
243                 a.killZeros();
244             }
```

```
245             while (rez.size() < (size() - div.size() + 1)) rez.insertElementAt(new Double(0.0)
                    , 0);
246             return rez;
247         }
248     }
249
250
251     /**
252      * Преобразует в строку CSV, заключенную в квадратные скобки.
253      *
254      * @return строка с вектором.
255      */
256     public String toStr() {
257         return "[" + toCSVStr() + "] ";
258     }
259
260     /**
261      * Преобразует в строку CSV (coma separated value).
262      *
263      * @return строка с вектором.
264      */
265     public String toCSVStr() {
266         String s = new String("");
267         if (size() == 0) return "0.0";
268         for (int i = size() - 1; i > 0; i--) {
269             double x = Math.round(getCoef(i) * 100) / 100.0;
270             s = s + x + ",";
271         }
272         double x = Math.round(getCoef(0) * 100) / 100.0;
273         return s + x;
274     }
275
276     /**
277      * Степень старшего коэффициента.
278      *
279      * @return степень старшего коэффициента.
280      */
281     public int getDeg() {
282         return size() - 1;
283     }
284
285     /**
286      * Выдает второй коэффициент в решении в решении диофантового уравнения.
287      *
288      * @param a второй вектор (первый вектор - сам вектор, т.е. this).
289      * @return собственно сам коэффициент.
290      */
291     public dVector getSolve(dVector a) {
292         diofantSolution ds = new diofantSolution(this, a);
293         return ds.diofant().B;
294     }
295
296     /**
297      * Класс содержащий два вектора (нужен, что бы возвращать решение)
298      * диофантового уравнения.
299      */
300     private class diofantSolution {
301
302         /**
303          * Первый вектор.
304          */
305         public dVector A;
306
307         /**
308          * Второй вектор.
309          */
310         public dVector B;
311
312         /**
313          * Конструктор.
314          */
315         public diofantSolution(dVector a, dVector b) {
316             A = a;
317             B = b;
318         }
319
```

```
320         /**
321          * Решает диофантовое уравнение xA+yB=1.
322          *
323          * @return решение (вектора x и y).
324          */
325         public diofantSolution diofant() {
326             dVector a, b, c, d, q, x, y, temp;
327             a = new dVector("1.0");
328             b = new dVector("0.0");
329             c = new dVector("0.0");
330             d = new dVector("1.0");
331             x = A;
332             y = B;
333             while (!x.mod(y).isZero()) {
334                 q = x.div(y);
335                 temp = new dVector(a);
336                 a = new dVector(b);
337                 b = temp.subVec(b.multVec(q));
338                 temp = new dVector(c);
339                 c = new dVector(d);
340                 d = temp.subVec(d.multVec(q));
341                 temp = new dVector(x);
342                 x = new dVector(y);
343                 y = temp.mod(y);
344             }
345             double z = b.multVec(A).addVec(d.multVec(B)).getCoef(0);
346             if (z == 0) z = 1.0;
347             return new diofantSolution(b.multNum(1 / z), d.multNum(1 / z));
348         }
349 
350     }
351 
352 
353 }
```

E.3. Исходный текст файла ErrorDialog.java

```
1  package ru.ifmo.vizi.vinograd;
2  
3  import ru.ifmo.vizi.base.Configuration;
4  import ru.ifmo.vizi.base.I18n;
5  import ru.ifmo.vizi.base.ui.HintedButton;
6  import ru.ifmo.vizi.base.ui.Hinter;
7  import ru.ifmo.vizi.base.ui.ModalDialog;
8  
9  import java.awt.*;
10 import java.util.StringTokenizer;
11 
12 /**
13  * ErrorDialog - это класс, которой высвечивает сообщение об ошибке
14  * в отдельном потоке.
15  */
16 public class ErrorDialog implements Runnable {
17 
18     /**
19      * Конфигурация, из которой читаются все параметры.
20      */
21     private Configuration config;
22 
23     /**
24      * Родительский фрейм для dlg.
25      */
26     private Frame owner;
27 
28     /**
29      * Ширина диалога.
30      */
31     private final static int width = 300;
32 
33     /**
34      * Высота диалога.
35      */
36     private final static int heigth = 200;
37 
38     /**
39      * Само окошко с ошибкой.
```

```
40       */
41      private ModalDialog dlg;
42
43      /**
44       * Название колонки с ошибкой.
45       */
46      private String errorFields;
47
48      /**
49       * Конструктор.
50       *
51       * @param config      конфигурация.
52       * @param owner       родительский фрейм.
53       * @param errorFields строка, содержащая имя TextField с ошибочными данными.
54       */
55      public ErrorDialog(Configuration config, Frame owner, String errorFields) {
56          this.config = config;
57          this.owner = owner;
58          this.errorFields = errorFields;
59      }
60
61      /**
62       * Метод вызываемый потоком, вызывающий окошко с ошибкой.
63       */
64      public void run() {
65          dlg = new ModalDialog(owner, config.getParameter("error-title"));
66          TextArea lMessage = new TextArea();
67          lMessage.setEditable(false);
68          Hinter hinter = new Hinter(config);
69          dlg.add(hinter);
70          Container panel = hinter.getContentPane();
71          panel.setLayout(new BorderLayout());
72          panel.add(lMessage, BorderLayout.CENTER);
73          panel.add(new HintedButton(config, "about-button-ok") {
74              protected void click() {
75                  dlg.setVisible(false);
76              }
77          }, BorderLayout.SOUTH);
78          Font font = config.getFont("about-font");
79          dlg.setFont(font);
80          Font captionFont = config.getFont("about-caption-font", font);
81          lMessage.setFont(captionFont);
82          StringBuffer text = new StringBuffer();
83          text.append(config.getParameter("error-input")).append(" ").append(errorFields).append
                ("\n");
84          int maxlen = (int) ((width * 0.9 * 1.5) / captionFont.getSize());
85          text.append(wrap(I18n.message(config.getParameter("error-message"), "" + (new Integer(
                config.getParameter("max-polynom-deg")).intValue() + 1)), maxlen));
86          lMessage.setText(text.toString());
87          dlg.setResizable(false);
88          dlg.pack();
89          dlg.setSize(width, heigth);
90          dlg.center();
91          dlg.setVisible(true);
92      }
93
94      /**
95       * Форматирует текст таким образом, что бы его ширина не была больше чем
96       * width.
97       *
98       * @param s     строка с текстом, который нужно форматировать.
99       * @param width ширина форматированного текста.
100      * @return форматированного текста.
101      */
102     private String wrap(String s, int width) {
103         StringBuffer sb = new StringBuffer();
104         StringTokenizer st = new StringTokenizer(s, " ");
105         int len = 0;
106         String tok;
107         while (st.hasMoreTokens()) {
108             tok = st.nextToken();
109             if (len + tok.length() + 1 > width) {
110                 if (len == 0) {
111                     sb.append(tok).append("\n");
112                 } else {
113                     sb.append("\n").append(tok);
```

```
                    len = tok.length();
                }
            } else {
                sb.append(" ").append(tok);
                len += tok.length() + 1;
            }
        }
        return sb.toString();
    }
}
```

E.4. Исходный текст файла HintedChoice.java

```
package ru.ifmo.vizi.vinograd;

import ru.ifmo.vizi.base.Configuration;
import ru.ifmo.vizi.base.ui.Hinter;

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

/**
 * HintedChoice это {@link java.awt.Choice},
 * на который повешен хинт.
 *
 * @author Alexandr Smal
 */
public final class HintedChoice extends Choice {

    /**
     * Строчка, содержащая хинт.
     */
    private final String hint;

    /**
     * Размер списка.
     */
    public final int size;

    /**
     * Конструктор.
     *
     * @param config конфигурация.
     * @param name   параметр из конфигурации.
     */
    public HintedChoice(Configuration config, String name) {
        hint = config.getParameter(name + "-hint");
        size = config.getInteger(name + "-size", 1);

        for (int i = 0; i < size; i++)
            this.addItem(config.getParameter(name + "-label-" + i, "--empty--"));
        enableEvents(AWTEvent.COMPONENT_EVENT_MASK);
    }

    /**
     * Показывает/убирает хинт.
     *
     * @param e событие компонента.
     */
    protected final void processComponentEvent(ComponentEvent e) {
        super.processComponentEvent(e);
        if (isVisible()) {
            Hinter.applyHint(this, hint);
        } else {
            Hinter.applyHint(this, null);
        }
    }
}
```

E.5. Исходный текст файла HintedTextField.java

```
package ru.ifmo.vizi.vinograd;

import ru.ifmo.vizi.base.Configuration;
import ru.ifmo.vizi.base.ui.Hinter;
```

```
5
6  import java.awt.*;
7  import java.awt.event.ComponentEvent;
8
9  /**
10  * HintedTextField это {@link java.awt.TextField},
11  * на который повешен хинт.
12  *
13  * @author Alexandr Smal
14  */
15 public final class HintedTextField extends TextField {
16
17     /**
18      * Строчка, содержащая хинт.
19      */
20     private final String hint;
21
22     /**
23      * Конструктор.
24      *
25      * @param config конфигурация.
26      * @param name   параметр из конфигурации.
27      * @param length длина.
28      */
29     public HintedTextField(Configuration config, String name, int length) {
30         super(length);
31         hint = config.getParameter(name + "-hint");
32         enableEvents(AWTEvent.COMPONENT_EVENT_MASK);
33     }
34
35     /**
36      * Конструктор.
37      *
38      * @param config конфигурация.
39      * @param name   параметр из конфигурации.
40      */
41     public HintedTextField(Configuration config, String name) {
42         hint = config.getParameter(name + "-hint");
43         enableEvents(AWTEvent.COMPONENT_EVENT_MASK);
44     }
45
46
47     /**
48      * Показывает/убирает хинт.
49      *
50      * @param e событие компонента.
51      */
52     protected final void processComponentEvent(ComponentEvent e) {
53         super.processComponentEvent(e);
54         if (isVisible()) {
55             Hinter.applyHint(this, hint);
56         } else {
57             Hinter.applyHint(this, null);
58         }
59     }
60 }
```