Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра “Компьютерные технологии”

Н.Н. Красильников

Построение визуализатора *2-3 дерева* на базе технологии *Vizi*

Программирование с явным выделением состояний
Проектная документация

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

Санкт-Петербург

2005

Оглавление

Введение

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

Визуализатор — программа, в процессе работы которой на экране компьютера динамически демонстрируется применение алгоритма к выбранному набору данных.

2-3 дерево [3, 4] – дерево, для которого операции поиска, вставки и удаления осуществляются за время $O(\log n)$, где n — число элементов в *2-3 дереве*.

В данной работе строится визуализатор алгоритмов поиска, вставки и удаления для *2-3 дерева* на базе технологии *Vizi*.

1. Анализ литературы

Информацию о *2-3 дереве* можно найти в книгах [3, 4].

Наиболее подробно эта структура данных описана в первой книге. Помимо ее описания, там также приведен пример реализации *2-3 дерева*.

В книге Т. Кормена основное внимание уделено *B-деревьям*, а о *2-3 деревьях* говориться только как об их частном случае.

2. Описание и анализ *2-3 дерева*

Существует целый ряд структур данных, для которых даже в самом худшем случае время выполнения операторов имеет порядок $O(\log n)$. Рассмотрим одну из таких реализаций, которая называется *2-3 дерево* и имеет следующие свойства:

- каждый внутренний узел имеет два или три сына;
- все пути от корня до любого листа имеют одинаковую длину.

Будем считать, что пустое дерево и дерево с одним узлом также являются *2-3 деревьями*.

На рис. 1 изображен пример *2-3 дерева*.

Рассмотрим представления множеств, элементы которых упорядочены посредством некоторого отношения линейного порядка, обозначаемого символом "<". Элементы множества располагаются в листьях дерева. Если элемент a располагается левее элемента b, то справедливо отношение $a < b$. Предполагаем, что упорядочивание элементов по используемому отношению линейного порядка основывается только на значениях одного поля из всех полей записи. Это поле назовем *ключом*.

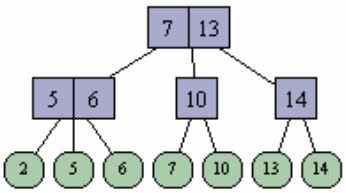

Рис.1. Пример *2-3 дерева*

В каждый внутренний узел записываются ключ наименьшего элемента, являющегося потомком второго сына, и ключ наименьшего элемента — потомка третьего сына, если он есть.

Для *поиска* в *2-3 дереве* необходимо последовательно просматривать ключи, хранящиеся во внутренних ячейках, спускаясь от корня к листьям. Вначале ключ искомого элемента сравнивается с первым ключом ячейки и, если искомый ключ не больше первого, то осуществляется переход в левое поддерево. Иначе, сравниваем искомый ключ со вторым ключом в ячейке (если второго ключа нет — поддерева всего два, то сразу переходим во второе поддерево) и если наш ключ не превосходит второй ключ, то осуществляется переход в среднее поддерево, а если превосходит, то идем в правое поддерево.

Так осуществляется спуск до искомого элемента.

Для *вставки* сначала реализуется алгоритм поиска для определения ячейки, сыном которой будет являться вставляемый элемент. Далее элемент вставляется в качестве сына

найденной ячейки. Ячейки сортируются. Если же в результате вставки у ячейки стало четыре сына, что является нарушением первого свойства *2-3 дерева*, то ячейка делится на две и выясняется, не нарушились ли свойства дерева для отца образовавшихся ячеек. В случае, когда первое свойство *2-3 дерева* не выполняется в корне дерева, то корень делится на две ячейки и для них создается отец, который и будет новым корнем дерева.

Для *удаления* реализуем алгоритм поиска удаляемого элемента в дереве. После этого удаляем найденный элемент и проверяем выполнение свойств *2-3 дерева* для его отца. Если свойства выполняются, то алгоритм удаления завершен. Если нет, то анализируются братья этой ячейки. Если у левого или правого брата три сына, то один из сыновей передается ячейке, из которой было произведено удаление. Если же у братьев по два сына, то после удаления единственный сын ячейки передается одному из братьев, а “бездетную” ячейку удаляем. В последнем случае необходимо опять же проверить выполнение свойств *2-3 дерева* для отца, рассмотренных братьев. Если у корня дерева в ходе этих перемещений остался только один сын, то удаляем корень и обозначаем новым корнем его единственного сына.

Высота *2-3 дерева* лежит между $log3\ n$ и $log2\ n$, где n - количество элементов в дереве. Поэтому и описанные операции над ним выполняются за требуемое время — $O(log\ n)$.

3. Реализация визуализатора *2-3 дерева* и операций над ним

Этапы создания визуализатора:

- разработка структур данных;
- запись модели данных в формате *XML/Java*;
- запись описанных алгоритмов в формате *XML/Java*;
- создание интерфейса.

Для хранения дерева и работы с ним созданы классы *Tree* и *Node*. Они находятся в файлах `Tree.java` и `Node.java` соответственно.

При этом получившаяся структура данных отличается от *2-3 дерева*. В указанных выше классах реализованы элементарные функции над деревом и функции по его отображению. Это значительно облегчило работу с деревом и позволило уменьшить "нагрузку" на класс *Tree23Applet* — главный класс визуализации. В связи с этим в данные классы были добавлены дополнительные поля, связанные с отображением. Более того, в связи с тем, что на некоторых шагах визуализации требуется отображать не *2-3 дерево*, так как при вставке на некоторых шагах у узла может быть по четыре сына, классы дерева были дополнительно расширены (добавлен четвертый сын и величина наименьшего ключа в нем).

Классы с "правильной" реализацией *2-3 дерева* приведены в приложении 1.

На втором и третьем этапах модель данных и алгоритмы писались в формате *XML/Java*. Алгоритмы в этом формате описывается почти также как и на привычных языках программирования. Здесь же пишутся и комментарии, которые выводятся при работе визуализатора. Созданные модель данных и алгоритмы приведены в приложении 2.

Процесс разработки интерфейса заключался в определении того, что должно быть отображено на экране, и добавлении соответствующих функций отображения в классы *Tree23Applet*, *Tree* и *Node*.

Исходные коды этих классов приведены в приложении 4.

4. Описание модели данных

Моделью данных называется класс, содержащий все необходимые переменные и структуры данных.

Созданная модель данных приведена ниже.

Экземпляр апплета	*Tree23Applet visualizer*
Стек	*AutoStack myStack*
2-3 дерево	*Tree tree*
Текущая ячейка	*Node currentNode*
Флаг алгоритма вставки	*Boolean insert*
Добавляемый элемент	*int insertingElement*
Флаг окончания алгоритма вставки	*Boolean insertEnd*
Флаг алгоритма удаления	*Boolean delete*
Удаляемый элемент	*int deletingElement*
Флаг окончания алгоритма удаления	*Boolean deleteEnd*
Флаг алгоритма поиска	*Boolean find*
Элемент для поиска	*int serchingElement*

5. Описание алгоритма в формате XML/Java

В приложении 2 приведен текст программы с выделенной моделью данных. Программа содержит следующие конструкции:

- последовательности операторов;
- условные операторы (*if-then-else*);
- вызовы функций (*call-auto*).

Данные конструкции являются стандартными и не затрудняют понимание написанного в данном формате текста.

6. Анализ получившихся автоматов

С помощью средств технологии *Vizi* из созданного описания алгоритма в формате *XML/Java* автоматически был сгенерирован *Java*-код. Этот код содержит уже готовые автоматы, реализующие алгоритмы работы с *2-3 деревом*.

В данном случае было сгенерировано семь пар автоматов. Каждая пара соответствует выделенной в алгоритме функции и обеспечивает прямой и обратный ход алгоритма. Ниже приведено количество состояний для каждого автомата.

Функция	*Количество состояний в паре автоматов*
`Main`	12
`Find`	9
`FindInSubtree`	26
`Insert`	30
`InsertIntoSubtree`	43
`Delete`	17
`DeleteFromSubtree`	64

Большое число состояний автоматов объясняется сложностью логики реализованных алгоритмов вставки, удаления и поиска алгоритмов и детальной их визуализацией.

Автоматы не являются идеально оптимизированными. При их создании вручную может получиться и меньшее число состояний. Однако в этом случае при создании визуализатора может потребоваться гораздо больше времени.

7. Описание визуализатора как *Java*-апплета

Визуализатор *2-3 дерева* на базе технологии *Vizi* представляет собой *Java*-апплет. Он может быть встроен в любой *HTML*-документ и запушен из под любого *Web*-браузера, поддерживающего *Java*-апплеты. Если браузер (*Internt Explorer*, *Opera*, *Mozilla*) по какой-то причине не отображает *Java*-апплеты, то следует скачать виртуальную машину *Java* (http://java.sun.com).

Визуализатор отображает произвольное *2-3 дерево* и позволяет осуществлять с ним операции поиска, вставки и удаления выбранных элементов.

При выполнении выбранной операции отображается каждое изменение, произведенное над *2-3 деревом*, и выводятся соответствующие комментарии.

8. Описание интерфейса визуализатора

На рис. 2 изображен интерфейс визуализатора.

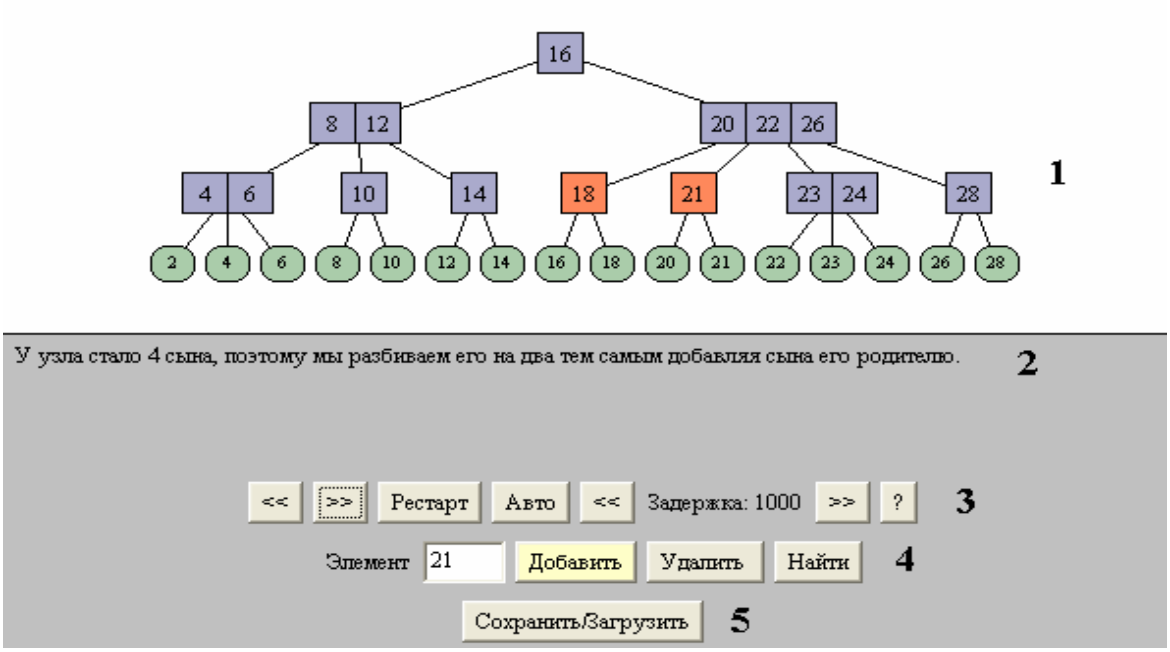

Рис. 2. Интерфейс визуализатора *2-3 дерева*

Ниже приведены основные элементы интерфейса.

1. *Область визуализации*. В этой области изображено дерево на текущем шаге визуализации.

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

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

4. *Панель выбора элемента дерева и операций над ним*. Здесь можно выбрать элемент и задать операцию.

5. *Панель загрузки/сохранения*. При нажатии на кнопку “Сохранить/Загрузить” можно сохранить/загрузить дерево и состояние визуализатора.

9. Конфигурация визуализатора

Параметры визуализатора изменяются в файле конфигурации `Tree23-Configuration` (приложение 2).

Параметры визуализатора и их описания приведены в таблице.

Параметры	**Описание**
Стили	
-style0	Стиль линий
-style1	Стиль листа
-style2	Стиль внутреннего узла
-style3	Стиль текущего узла
Дополнительные параметры	
-comment-height	Высота области комментария
-label-element	Надпись "Элемент"
-button-add	Параметры кнопки добавления
-button-remove	Параметры кнопки удаления
-button-find	Параметры кнопки поиска
-text-field-element-hint	Параметры поля ввода элемента
-text-field-element-rows	Параметры поля ввода элемента
-not-number-error	Ошибка ввода - некорректный номер элемента
-max-point-number-error	Ошибка ввода - предел количества элементов
-line-style	Стиль линий
-leaf-default-style	Стиль листа
-interior-default-style	Стиль внутреннего узла
-current-node-style	Стиль текущего узла
-mark-out-button-color-r	Цвет подсвеченной кнопки
-mark-out-button-color-g	Цвет подсвеченной кнопки

-mark-out-button-color-b	Цвет подсвеченной кнопки
-leaf-width	Параметры листа
-leaf-height	Параметры листа
-interior-width	Параметры внутреннего элемента
-interior-height	Параметры внутреннего элемента
-min-element	Минимальный элемент
-max-element	Максимальный элемент
-min-element-number	Минимальный номер элемента
-max-element-number	Максимальный номер элемента
-example-of-tree	Пример
-save-load-mode	Параметры кнопки и диалога сохранения-загрузки
-save-load-width	Параметры кнопки и диалога сохранения-загрузки
-save-load-height	Параметры кнопки и диалога сохранения-загрузки
-find	Название алгоритма
-insert	Название алгоритма
-delete	Название алгоритма
-load-info-0	Ошибка ввода при загрузке
-load-info-1	Ошибка ввода при загрузке
-load-info-2	Ошибка ввода при загрузке
-load-info-3	Ошибка ввода при загрузке
-load-info-4	Ошибка ввода при загрузке
-load-info-5	Ошибка ввода при загрузке
-load-info-6	Ошибка ввода при загрузке
-load-info-7	Ошибка ввода при загрузке
-load-info-8	Ошибка ввода при загрузке
-load-info-9	Ошибка ввода при загрузке

-load-info-10	Ошибка ввода при загрузке
-save-load	Параметры кнопки и диалога сохранения-загрузки
-SaveLoadDialog-CommentPane-height	Параметры кнопки и диалога сохранения-загрузки

Заключение

Технология *Vizi* имеет ряд положительных сторон.

При ее использовании достаточно строго определен технологический процесс создания визуализатора, что уменьшает количество эвристических решений.

Многие необходимые элементы визуализатора существуют в уже готовом виде, что облегчает написание визуализатора в целом.

Также в результате получается визуализатор со стандартизованным интерфейсом, что особенно важно при разработке визуализаторов большим числом программистов для одного *web*-ресурса.

К сожалению, есть и недостатки.

При использовании данной технологии довольно велико количество усилий, уходящих на написание *XML*-файлов, так как данный формат сильно увеличивает количество написанного текста в символах. Так для данного визуализатора размер *XML*-описания составляет примерно 128 Kb.

В заключение отметим то, что документация занимает 183 станицы, их которых 167 страниц кода (в том числе сгенерированного автоматически)!

Источники

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

2. Vizi Home Page. http://ctddev.ifmo.ru/vizi/

3. Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. М.: Вильямс. 2001, с. 158-167.

4. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ. М.: МЦМНО, 1999, с. 359-375.

Приложение 1. Исходный текст приложения на *Java*

Tree23.java

```
/**
 * Tree 2-3
 */
public class Tree23
{
    /**
     * Children
     */
    public Node root;

    /**
     * Finds node by key
     *
     * @param key - key
     * @return searching node
     */
    Node find(int key)
    {
        return root.find(key);
    }

    /**
     * Inserts node
     *
     * @param element - new element
     */
    void insert(int element)
    {
        int low;
        Node tempNode;
        Node backNode;
        backNode = null;

        low = root.insert(element, backNode);

        if (backNode != null) // т.е. стало ли четыре сына
        {
            // разбиваем корень на два узла и создаем им родителя
            // этот новый элемент и станет корнем
            tempNode = new Node();
            tempNode.child[0] = root;
            tempNode.child[1] = backNode;
            tempNode.low2 = low;

            root = tempNode;
        }
    }

    /**
     * Deletes node
     *
     * @param element - element
     */
    void delete(int element)
    {
        if (root.delete(element)) {
            // у корня один сын, этот сын и станет новым корнем
            root = root.child[0];
        }
    }
}
```

Node.java

```
/**
 * Node for tree 2-3
 */
public class Node
{
    /**
     * Shows is this node leaf
     */
    boolean isLeaf;

    /**
     * Children
     */
    public Node child[];

    /**
     * Node's keys
     */
    public int low2, low3;

    /**
     * Element (actual if this node is leaf and could be large then int)
     */
    public int element;

    /**
     * Finds node by key
     *
     * @param key - key
     * @return searching node
     */
    Node find(int key)
    {
        if (isLeaf) {
            if (key == element) {
                return this; // нашли!
            } else {
                return null; // такого элемента в дереве нет
            }
        } else {
            // выберем подддерево для продолжения поиска
            if (key < low2) {
                return child[0].find(key);
            } else {
                if (child[2] == null) {
                    return child[1].find(key);
                } else {
                    if (key < low3) {
                        return child[1].find(key);
                    } else {
                        return child[2].find(key);
                    }
                }
            }
        }
    }

    /**
     * Inserts node
     *
     * @param element - new element
     * @param newNode - new node
     * @return low
     */
    int insert(int element, Node newNode)
    {
        int childToInsert;
        int low;
```

```
Node backNode;
backNode = null;
newNode = null;

if (isLeaf) {
    if (element != this.element) {
        // создание нового листа и
        // передача на верх нового минимального значения
        if (element < this.element) {
            newNode = new Node();
            newNode.isLeaf = true;
            newNode.element = this.element;
            this.element = element;
            return element;
        } else {
            newNode = new Node();
            newNode.isLeaf = true;
            newNode.element = element;
            return element;
        }
    } else {
        // такой элемент в дереве уже есть
        return 0;
    }
} else {
    // поиск поддерева в котором произведем вставку листа
    if (element < low2) {
        low = child[0].insert(element, backNode);
        childToInsert = 1;
    } else {
        if (child[2] == null) {
            low = child[1].insert(element, backNode);
            childToInsert = 2;
        } else {
            if (element < low3) {
                low = child[1].insert(element, backNode);
                childToInsert = 2;
            } else {
                low = child[2].insert(element, backNode);
                childToInsert = 3;
            }
        }
    }

    if (backNode != null) // т.е. стало ли 4 сына
    {
        // стало ...
        // теперь лишнего отпрыска нужно куда-нибудь пристроить
        if (child[2] == null) {
            // сделаем его третьим сыном
            if (childToInsert == 2) {
                child[2] = backNode;
                low3 = low;
            } else {
                child[2] = child[1];
                low3 = low2;
                child[1] = backNode;
                low2 = low;
            }
        } else {
            // разобьем узел на 2
            newNode = new Node();
            if (childToInsert == 3) {
                newNode.child[0] = child[2];
                newNode.child[1] = backNode;
                newNode.low2 = low;
                return low3;
            } else {
                newNode.child[1] = child[2];
                newNode.low2 = low3;
                if (childToInsert == 2) {
                    newNode.child[0] = backNode;

                    return low;
```

```
                    } else // childToInsert == 1
                    {
                        newNode.child[0] = child[1];
                        child[1] = backNode;
                        low2 = low;

                        return low2;
                    }
                }
            }
        }
    }
    return 0;
}

/**
 * Deletes node
 *
 * @param element - element
 * @return is only one child
 */
boolean delete(int element)
{
    int childToDelete;
    boolean onlyone = false;

    if (child[0].isLeaf) {
        // мы находимся на уровне над листьями
        // найдем (или не найдем) среди детей нужный элемент и
        //  удалим его
        if (child[2] == null) {
            if (child[0].element == element) {
                child[0] = child[1];
                child[1] = null;
            }
            if (child[1].element == element) {
                child[1] = null;
            }

            return true;
        } else {
            if (child[0].element == element) {
                child[0] = child[1];
                child[1] = child[2];
                child[2] = null;
            }
            if (child[1].element == element) {
                child[1] = child[2];
                child[2] = null;
            }
            if (child[2].element == element) {
                child[2] = null;
            }
        }
    } else {
        // поиск поддерева в котором произведем удаление листа
        if (element < low2) {
            onlyone = child[0].delete(element);
            childToDelete = 1;
        } else {
            if (child[2] == null) {
                onlyone = child[1].delete(element);
                childToDelete = 2;
            } else {
                if (element < low3) {
                    onlyone = child[1].delete(element);
                    childToDelete = 2;
                } else {
                    onlyone = child[2].delete(element);
                    childToDelete = 3;
                }
            }
        }
```

```
            if (onlyone) // остался ли ровно один сын
            {
                // остался ...
                // додавим его к братьям или
                // добавим ему брата из соседних поддеревьев

                if (childToDelete == 1) {
                    if (child[1].child[2] != null) {
                        child[0].child[1] = child[1].child[0];
                        child[1].child[0] = child[1].child[1];
                        child[1].child[1] = child[1].child[2];
                        child[1].child[2] = null;
                    } else {
                        child[0].child[1] = child[1].child[0];
                        child[0].child[2] = child[1].child[1];

                        if (child[2] != null) {
                            child[1] = child[2];
                            child[2] = null;
                        } else {
                            child[1] = null;

                            return true;
                        }
                    }
                }
                if (childToDelete == 2) {
                    if (child[0].child[2] != null) {
                        child[1].child[1] = child[1].child[0];
                        child[1].child[0] = child[0].child[2];
                        child[0].child[2] = null;
                    } else {
                        if (child[2] != null) {
                            if (child[2].child[2] != null) {
                                child[1].child[1] = child[2].child[0];
                                child[2].child[0] = child[2].child[1];
                                child[2].child[1] = child[2].child[2];
                                child[2].child[2] = null;
                            } else {
                                child[0].child[2] = child[1].child[0];
                                child[1] = child[2];
                                child[2] = null;
                            }
                        } else {
                            child[0].child[2] = child[1].child[0];
                            child[1] = null;
                            return true;
                        }
                    }
                }
                if (childToDelete == 3) {
                    if (child[1].child[2] != null) {
                        child[2].child[1] = child[2].child[0];
                        child[2].child[0] = child[1].child[2];
                        child[1].child[2] = null;
                    } else {
                        child[1].child[2] = child[2].child[0];
                        child[2] = null;
                    }
                }
            }
        }

        return false;
    }
}
```

Приложение 2. *XML*-описание визуализатора

Tree23.xml (основные параметры)

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

<!--
"Tree23" visualizer description
Version: $Id: Tree23.xml,v 1.3 2004/01/12 23:55:04 Kot Exp $
-->

<!DOCTYPE visualizer PUBLIC
    "-//IFMO Vizi//Visualizer description"
    "http://ips.ifmo.ru/vizi/dtd/visualizer.dtd"
[
    <!ENTITY algorithm SYSTEM "Tree23-Algorithm.xml">
    <!ENTITY configuration SYSTEM "Tree23-Configuration.xml">
]>

<visualizer
    id="Tree23"
    package="ru.ifmo.vizi.Tree23"
    main-class="Tree23Applet"

    preferred-width="600"
    preferred-height="400"

    name-ru="2-3 Деревья"
    name-en="2-3 Trees"

    author-ru="Красильников Николай"
    author-en="Nick Krasilnikov"
    author-email="krasilnikov@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), 2003"
    copyright-en="Copyright \u00A9
Computer Technologies Department, SPb IFMO, 2003"
>
    &algorithm;
    &configuration;
</visualizer>
```

Tree23-Algorithm.xml (описание алгоритма)

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

<algorithm>

    <data>
        <variable description="Applet">Tree23Applet visualizer;</variable>

        <variable description="2-3 Tree">Tree tree;</variable>

        <variable description="Current node">Node currentNode;</variable>

        <variable description="Stack">AutoStack myStack = new AutoStack();</variable>

        <variable description="Insert">boolean insert = false;</variable>
        <variable description="Inserting element">int insertingElement;</variable>
```

```
    <variable description="End of insert auto">boolean insertEnd = false;</variable>

    <variable description="Delete">boolean delete = false;</variable>
    <variable description="Deleting element">int deletingElement;</variable>
    <variable description="End of delete auto">boolean deleteEnd = false;</variable>

    <variable description="Find">boolean find = false;</variable>
    <variable description="Serching element">int serchingElement;</variable>

    <toString>
        return "";
    </toString>
</data>

<auto id="Main" description="main auto">

    <start
        comment-ru="2-3 дерево - это сбалансированное дерево со следующими свойствами:\n
                    1) Каждый внутренний узел имеет два или три сына.\n
                    2) Все пути от корня до листа имеют одинаковую длинну.\n
                    В каждом внутреннем узле хранятся значения наименьших
                    элементов в поддеревьях (начиная со второго поддерева).
                    Для начала выберите действие над деревом,
                    а затем для начала визуализации нажмите &quot;&gt;&gt;&quot;."
        comment-en="2-3 tree is balanced tree with the folowing properties:\n
                    1) Every interior node have 2 or 3 children.\n
                    2) All ways from root to leaves have equal length.\n
                    In every interior node(beginning from second subtree)
                    least values of subtrees are stored.
                    Select an operation to begin. To start visualization press
                    &quot;&gt;&gt;&quot;."
        />

    <if
        id="isinsert"
        description="is insert"
        level="-1"
        test="d.insert"
        >
        <then>
            <call-auto id="insert"/>
        </then>
        <else>
            <if
                id="isdelete"
                description="is delete"
                level="-1"
                test="d.delete"
                >
                <then>
                    <call-auto id="delete"/>
                </then>
                <else>
                    <if
                        id="isfind"
                        description="is find"
                        level="-1"
                        test="d.find"
                        >
                        <then>
                            <call-auto id="find"/>
                        </then>
                        <else>
                            <step
                                id="nocommand"
                                description="no command"
                                comment-ru="Не выбрано действие."
                                comment-en="Command is not chosen."
                                level="1"
                                >
                                <draw/>
                                <direct/>
                                <reverse/>
                            </step>
```

```
                    </else>
                </if>
            </else>
        </if>
    </else>
</if>

</auto>

<auto id="find" description="find auto">

    <if
        id="findistreeempty"
        description="is tree empty"
        level="-1"
        test="d.tree.childrenNumber == 0"
        >
        <then>
            <step
                id="findtreeempty"
                description="tree is empty"
                comment-ru="Дерево пусто, и искомого элемента в нем соответственно нет."
                comment-en="Tree is empty. So required element is certainly missing."
                level="1"
                >
                <draw>
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                </draw>
                <direct/>
                <reverse/>
            </step>
        </then>
        <else>
            <step
                id="findcurrentelementinit1"
                description="init"
                level="-1"
                >
                <draw/>
                <direct>
                    d.currentNode = d.tree.rootNode1;
                    d.currentNode.SetCurrentNodeStyle();
                </direct>
                <reverse>
                    d.currentNode.SetOldStyle();
                </reverse>
            </step>
            <step
                id="findbeginsfromroot"
                description="begins from root"
                comment-ru="В поисках искомого элемента ({0})
                            мы спустимся от корня к листьям,
                            на каждом шаге выбирая в каком
                            поддереве продолжать поиски.
                            Начинаем с корня."
                comment-en="Searching for suoght element {{0}},
                            we'll decend from root to leaves,
                            deciding in wich subtree should we continue or searching."
                comment-args="new Integer(d.serchingElement)"
                level="1"
                >
                <draw>
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                </draw>
                <direct/>
                <reverse/>
            </step>
            <call-auto id="findinsubtree"/>
        </else>
    </if>
    <step
        id="findreturnssetings"
```

```
        description="returns setings"
        level="-1"
        >
        <draw/>
        <direct>
            d.currentNode.SetOldStyle();
        </direct>
        <reverse>
            d.currentNode.SetCurrentNodeStyle();
        </reverse>
    </step>

</auto>

<auto id="findinsubtree" description="find auto">

    <if
        id="findisleaf"
        description="is leaf"
        level="-1"
        test="d.currentNode.isLeaf"
        >
        <then>
            <step
                id="findyesitisleaf"
                description="yes it is"
                comment-ru="Поскольку текущий элемент является листом,
                            надо только проверить, {0} ли он."
                comment-en="Current element is leaf.
                            So we should only compare it with {0}."
                comment-args="new Integer(d.serchingElement)"
                level="1"
                >
                <draw>
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                </draw>
                <direct/>
                <reverse/>
            </step>
            <if
                id="findisitserchingelement"
                description="is it serching element"
                level="-1"
                test="d.currentNode.element == d.serchingElement"
                >
                <then>
                    <step
                        id="findyesitisserchingelement"
                        description="yes it is"
                        comment-ru="Элемент {0} в дереве есть."
                        comment-en="There is element {0} in tree"
                        comment-args="new Integer(d.serchingElement)"
                        level="1"
                        >
                        <draw>
                            d.visualizer.DrawTree();
                            d.visualizer.UpdateScreen();
                        </draw>
                        <direct/>
                        <reverse/>
                    </step>
                </then>
                <else>
                    <step
                        id="findnoitisnotserchingelement"
                        description="no it is not"
                        comment-ru="Элементa {0} в дереве нет."
                        comment-en="There is no element {0} in tree."
                        comment-args="new Integer(d.serchingElement)"
                        level="1"
                        >
                        <draw>
                            d.visualizer.DrawTree();
```

```
                    d.visualizer.UpdateScreen();
                </draw>
                <direct/>
                <reverse/>
            </step>
        </else>
    </if>
</then>
<else>
    <if
        id="findisserchingelementin1subtree"
        description="is serching element in first subtree"
        level="-1"
        test="d.currentNode.low2 &gt; d.serchingElement"
        >
        <then>
            <step
                id="findltlow2"
                description="lt low2"
                comment-ru="Искомый элемент меньше наименьшего элемента
                            во втором поддереве ({0}),
                            поэтому поиски будем продолжать в первом поддереве."
                comment-en="Sought element is less than least element
                            of second subtree ({0}),
                            so our search we'll continue in first subtree."
                comment-args="new Integer(d.currentNode.low2)"
                level="1"
                >
                <draw>
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                </draw>
                <direct/>
                <reverse/>
            </step>
            <step
                id="findcurrentelementinit2"
                description="init"
                level="-1"
                >
                <draw/>
                <direct>
                    d.currentNode.SetOldStyle();
                    d.currentNode = d.currentNode.child1;
                    d.currentNode.SetCurrentNodeStyle();
                </direct>
                <reverse>
                    d.currentNode.SetOldStyle();
                    d.currentNode = d.currentNode.parent;
                    d.currentNode.SetCurrentNodeStyle();
                </reverse>
            </step>
            <call-auto id="findinsubtree"/>
        </then>
        <else>
            <if
                id="findischildrennumber2ornot"
                description="is children number 2"
                level="-1"
                test="d.currentNode.childrenNumber == 2"
                >
                <then>
                    <step
                        id="findgtlow2"
                        description="gt low2"
                        comment-ru="Искомый элемент больше наименьшего элемента
                                    во втором поддереве ({0}),
                                    поэтому поиски будем продолжать
                                    во втором поддереве."
                        comment-en="Sought element is greater than least element
                                    of second subtree ({0}),
                                    so our search we'll continue
                                    in second subtree."
                        comment-args="new Integer(d.currentNode.low2)"
```

```
            level="1"
            >
            <draw>
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
            </draw>
            <direct/>
            <reverse/>
        </step>
        <step
            id="findcurrentelementinit3"
            description="init"
            level="-1"
            >
            <draw/>
            <direct>
                d.currentNode.SetOldStyle();
                d.currentNode = d.currentNode.child2;
                d.currentNode.SetCurrentNodeStyle();
            </direct>
            <reverse>
                d.currentNode.SetOldStyle();
                d.currentNode = d.currentNode.parent;
                d.currentNode.SetCurrentNodeStyle();
            </reverse>
        </step>
        <call-auto id="findinsubtree"/>
    </then>
    <else>
        <if
            id="findisserchingelementin2subtree"
            description="is serching element in second subtree"
            level="-1"
            test="d.currentNode.low3 &gt; d.serchingElement"
            >
            <then>
                <step
                    id="findltlow3"
                    description="lt low3"
                    comment-ru="Искомый элемент больше
                                наименьшего элемента
                                во втором поддереве ({0}) и
                                меньше наименьшего элемента
                                в третьем ({1}),
                                поэтому поиски будем продолжать
                                во втором поддереве."
                    comment-en="Sought element is greater than
                                least element of second subtree ({0})
                                and less than least element
                                of the third subtree ({1}),
                                so our search we'll continue
                                in second subtree."
                    comment-args="new Integer(d.currentNode.low2),
                                  new Integer(d.currentNode.low3)"
                    level="1"
                    >
                    <draw>
                        d.visualizer.DrawTree();
                        d.visualizer.UpdateScreen();
                    </draw>
                    <direct/>
                    <reverse/>
                </step>
                <step
                    id="findcurrentelementinit4"
                    description="init"
                    level="-1"
                    >
                    <draw/>
                    <direct>
                        d.currentNode.SetOldStyle();
                        d.currentNode = d.currentNode.child2;
                        d.currentNode.SetCurrentNodeStyle();
                    </direct>
```

```
                                        <reverse>
                                            d.currentNode.SetOldStyle();
                                            d.currentNode = d.currentNode.parent;
                                            d.currentNode.SetCurrentNodeStyle();
                                        </reverse>
                                    </step>
                                    <call-auto id="findinsubtree"/>
                                </then>
                                <else>
                                    <step
                                        id="findgtlow3"
                                        description="gt low3"
                                        comment-ru="Искомый элемент больше
                                                    наименьшего элемента
                                                    в третьем поддереве ({0}),
                                                    поэтому поиски будем продолжать
                                                    в третьем поддереве."
                                        comment-en="Sought element is greater than
                                                    least element of third subtree ({0}),
                                                    so our search we'll continue
                                                    in third subtree."
                                        comment-args="new Integer(d.currentNode.low3)"
                                        level="1"
                                        >
                                        <draw>
                                            d.visualizer.DrawTree();
                                            d.visualizer.UpdateScreen();
                                        </draw>
                                        <direct/>
                                        <reverse/>
                                    </step>
                                    <step
                                        id="findcurrentelementinit5"
                                        description="init"
                                        level="1"
                                        >
                                        <draw>
                                            d.visualizer.DrawTree();
                                            d.visualizer.UpdateScreen();
                                        </draw>
                                        <direct>
                                            d.currentNode.SetOldStyle();
                                            d.currentNode = d.currentNode.child3;
                                            d.currentNode.SetCurrentNodeStyle();
                                        </direct>
                                        <reverse>
                                            d.currentNode.SetOldStyle();
                                            d.currentNode = d.currentNode.parent;
                                            d.currentNode.SetCurrentNodeStyle();
                                        </reverse>
                                    </step>
                                    <call-auto id="findinsubtree"/>
                                </else>
                            </if>
                        </else>
                    </if>
                </else>
            </if>

        </auto>

        <auto id="insert" description="insert auto">

            <if
                id="insertistreeempty"
                description="is tree empty"
                level="-1"
                test="d.tree.childrenNumber == 0"
                >
                <then>
                    <step
                        id="inserttreeempty"
```

```
            description="tree is empty"
            comment-ru="Дерево пусто, поэтому вставляемый элемент становится корнем."
            comment-en="Tree is empty, so inserted element becoms root."
            level="1"
            >
            <draw>
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
            </draw>
            <direct>
                d.tree.NewChildLeaf(d.insertingElement);
                d.tree.rootNode1.SetCurrentNodeStyle();
                d.currentNode = d.tree.rootNode1;
            </direct>
            <reverse>
                d.tree.DeleteChild(1);
            </reverse>
        </step>
    </then>
    <else>
        <if
            id="insertisrootleaf"
            description="is root leaf"
            level="-1"
            test="d.tree.rootNode1.isLeaf"
            >
            <then>
                <if
                    id="insertisthiselementintree"
                    description="is this element in tree"
                    level="-1"
                    test="d.tree.rootNode1.element == d.insertingElement"
                    >
                    <then>
                        <step
                            id="insertthiselementisintree"
                            description="this element is in tree"
                            comment-ru="Такой элемент в дереве уже есть."
                            comment-en="This element is already exists."
                            level="1"
                            >
                            <draw>
                                d.visualizer.DrawTree();
                                d.visualizer.UpdateScreen();
                            </draw>
                            <direct>
                                d.tree.rootNode1.SetCurrentNodeStyle();
                            </direct>
                            <reverse>
                                d.tree.rootNode1.SetOldStyle();
                            </reverse>
                        </step>
                    </then>
                    <else>
                        <step
                            id="insertrootisleaf"
                            description="root is leaf"
                            comment-ru="Корень является листом.
                                        Создаем еще одно дерево
                                        из вставляемого элемента."
                            comment-en="Root is leaf.
                                        Create another tree from inserted element."
                            level="1"
                            >
                            <draw>
                                d.visualizer.DrawTree();
                                d.visualizer.UpdateScreen();
                            </draw>
                            <direct>
                                int returned =
                                    d.tree.NewChildLeaf(d.insertingElement);
                                if(returned == 1)
                                    d.tree.rootNode1.SetCurrentNodeStyle();
                                if(returned == 2)
```

```
                        d.tree.rootNode2.SetCurrentNodeStyle();
                    </direct>
                    <reverse>
                        d.tree.rootNode1.SetOldStyle();
                        d.tree.rootNode2.SetOldStyle();
                        d.tree.DeleteChild(
                            d.tree.FindChild(d.insertingElement));
                    </reverse>
                </step>
                <step
                    id="insertnewroot0"
                    description="to new node"
                    comment-ru="Все дерево разделилось на два,
                                объединяем их, создавая новый корень."
                    comment-en="Tree divided in two, we'll unite them,
                                spawning a new root."
                    level="1"
                    >
                    <draw>
                        d.visualizer.DrawTree();
                        d.visualizer.UpdateScreen();
                    </draw>
                    <direct>
                        d.tree.rootNode1.SetOldStyle();
                        d.tree.rootNode2.SetOldStyle();
                        d.tree.DeleteChild(2);
                        d.tree.DeleteChild(1);
                        d.tree.NewChildInterior(d.tree.rootNode1,
                                                d.tree.rootNode2);
                        d.tree.rootNode1.SetCurrentNodeStyle();
                    </direct>
                    <reverse>
                        d.tree.rootNode1.SetOldStyle();
                        d.tree.DeleteChild(1);
                        Node tn = d.tree.rootNode1;
                        d.tree.AddChild(tn.child1);
                        d.tree.AddChild(tn.child2);
                        int returned = d.tree.FindChild(d.insertingElement);
                        if(returned == 1)
                            d.tree.rootNode1.SetCurrentNodeStyle();
                        if(returned == 2)
                            d.tree.rootNode2.SetCurrentNodeStyle();
                    </reverse>
                </step>
            </else>
        </if>
        <step
            id="insertreturnssetings0"
            description="returns setings"
            level="-1"
            >
            <draw/>
            <direct>
                d.tree.rootNode1.SetOldStyle();
            </direct>
            <reverse>
                d.tree.rootNode1.SetCurrentNodeStyle();
            </reverse>
        </step>
    </then>
    <else>
        <step
            id="insertcurrentelementinit1"
            description="init"
            level="-1"
            >
            <draw/>
            <direct>
                d.currentNode = d.tree.rootNode1;
                d.currentNode.SetCurrentNodeStyle();
            </direct>
            <reverse>
                d.currentNode.SetOldStyle();
            </reverse>
```

```
                    </step>
                    <step
                        id="insertbeginsfromroot"
                        description="begins from root"
                        comment-ru="Сначала найдем место куда вставить {0}.
                                    При этом повторяются действия алгоритма поиска."
                        comment-en="At first we'll find a place to insert {0}.
                                    And search algorithm's actions are repeated."
                        comment-args="new Integer(d.insertingElement)"
                        level="1"
                        >
                        <draw>
                            d.visualizer.DrawTree();
                            d.visualizer.UpdateScreen();
                        </draw>
                        <direct/>
                        <reverse/>
                    </step>
                    <call-auto id="insertintosubtree"/>
                </else>
            </if>
        </else>
    </if>
    <if id="insertis40"
        description="is 4 children"
        level="-1"
        test="d.tree.rootNode1.childrenNumber == 4">
        <then>
            <step
                id="insertinsertelement0"
                description="insert"
                comment-ru="У узла стало четыре сына, поэтому мы разбиваем его на два
                            тем самым добавляя сына его родителю,
                            или разделяя дерево на два,
                            если текущий этемент является корнем."
                comment-en="Current node have four children, so we divide this node in
                            two adding by this operation child to his parent,
                            or divide tree in two subtrees if current node is root."
                level="1"
                >
                <draw>
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                </draw>
                <direct>
                    d.currentNode.SetOldStyle();
                    d.tree.rootNode1.DeleteChild(4);
                    d.tree.rootNode1.DeleteChild(3);
                    d.tree.NewChildInterior(d.tree.rootNode1.child3,
                    d.tree.rootNode1.child4);
                    d.tree.rootNode1.SetCurrentNodeStyle();
                    d.tree.rootNode2.SetCurrentNodeStyle();
                </direct>
                <reverse>
                    d.currentNode.SetCurrentNodeStyle();
                    d.tree.rootNode1.SetOldStyle();
                    d.tree.rootNode2.SetOldStyle();
                    Node tempNode = d.tree.rootNode2;
                    d.tree.DeleteChild(2);
                    d.tree.rootNode1.AddChild(tempNode.child1);
                    d.tree.rootNode1.AddChild(tempNode.child2);
                </reverse>
            </step>
            <step
                id="insertnewroot1"
                description="to new node"
                comment-ru="Дерево разделилось на два, объединяем их,
                            создавая новый корень."
                comment-en="The tree is diveded in two, so we unite 'em,
                            creating a new root."
                level="1"
                >
                <draw>
                    d.visualizer.DrawTree();
```

```
            d.visualizer.UpdateScreen();
        </draw>
        <direct>
            d.tree.rootNode1.SetOldStyle();
            d.tree.rootNode2.SetOldStyle();
            Node tempNode1 = d.tree.rootNode1;
            Node tempNode2 = d.tree.rootNode2;
            d.tree.DeleteChild(2);
            d.tree.DeleteChild(1);
            d.tree.NewChildInterior(tempNode1, tempNode2);
            d.tree.rootNode1.SetCurrentNodeStyle();
        </direct>
        <reverse>
            d.tree.rootNode1.SetOldStyle();
            d.tree.DeleteChild(1);
            Node tn = d.tree.rootNode1;
            d.tree.AddChild(tn.child1);
            d.tree.AddChild(tn.child2);
        </reverse>
    </step>
    <step
        id="insertreturnssetings1"
        description="returns setings"
        level="-1"
        >
        <draw/>
        <direct>
            d.tree.rootNode1.SetOldStyle();
        </direct>
        <reverse>
            d.tree.rootNode1.SetCurrentNodeStyle();
        </reverse>
    </step>
    <if id="insertisnot40"
        description="is not 4 children"
        level="-1"
        test="d.currentNode.parent.childrenNumber == 3">
        <then>
            <step
                id="insertend0"
                description="end"
                comment-ru="После вставки стало три сына.
                            Осталось поднятся по дереву и,
                            возможно, исправить в узлах значения
                            наименьших элементов в поддеревьях."
                comment-en="There are three sons after isertion.
                            We should climb the tree and, maybe,
                            correct node least element info values of subtrees."
                level="1"
                >
                <draw>
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                </draw>
                <direct>
                    d.currentNode.parent.SetCurrentNodeStyle();
                </direct>
                <reverse>
                    d.currentNode.parent.SetOldStyle();
                </reverse>
            </step>
            <step
                id="insertreturnsettings0"
                description="retuns settings"
                level="-1"
                >
                <draw/>
                <direct>
                    d.currentNode.parent.SetOldStyle();
                </direct>
                <reverse>
                    d.currentNode.parent.SetCurrentNodeStyle();
                </reverse>
            </step>
```

```
        </then>
    </if>
</then>
<else>
    <if
        id="insertisroot0"
        description="is root"
        level="-1"
        test="d.currentNode != null &amp;amp;&amp;amp; !d.currentNode.isRoot">
        <then>
            <step
                id="insertgotoparent0"
                description="to parent"
                comment-ru="Сверяем значения наименьших элементов в поддеревьях."
                comment-en="Compare least values of subtrees."
                level="-1"
                >
                <draw>
                </draw>
                <direct>
                    d.currentNode.SetOldStyle();
                    d.myStack.pushInteger(d.currentNode.thisChildNumber);
                    d.currentNode = d.currentNode.parent;
                    d.currentNode.SetCurrentNodeStyle();
                    d.myStack.pushInteger(d.currentNode.low1);
                    d.myStack.pushInteger(d.currentNode.low2);
                    d.myStack.pushInteger(d.currentNode.low3);
                    d.myStack.pushInteger(d.currentNode.low4);
                    if(d.currentNode.childrenNumber == 1)
                    {
                        d.currentNode.low1 = d.currentNode.child1.low1;
                    }
                    if(d.currentNode.childrenNumber == 2)
                    {
                        d.currentNode.low1 = d.currentNode.child1.low1;
                        d.currentNode.low2 = d.currentNode.child2.low1;
                    }
                    if(d.currentNode.childrenNumber == 3)
                    {
                        d.currentNode.low1 = d.currentNode.child1.low1;
                        d.currentNode.low2 = d.currentNode.child2.low1;
                        d.currentNode.low3 = d.currentNode.child3.low1;
                    }
                </direct>
                <reverse>
                    d.currentNode.SetOldStyle();
                    d.currentNode.low4 = d.myStack.popInteger();
                    d.currentNode.low3 = d.myStack.popInteger();
                    d.currentNode.low2 = d.myStack.popInteger();
                    d.currentNode.low1 = d.myStack.popInteger();
                    int returned = d.myStack.popInteger();
                    if(returned == 1)
                        d.currentNode = d.currentNode.child1;
                    if(returned == 2)
                        d.currentNode = d.currentNode.child2;
                    if(returned == 3)
                        d.currentNode = d.currentNode.child3;
                    if(returned == 4)
                        d.currentNode = d.currentNode.child4;
                    d.currentNode.SetCurrentNodeStyle();
                </reverse>
            </step>
        </then>
    </if>
    <step
        id="insertreturnsettings4"
        description="retuns settings"
        level="1"
        >
        <draw>
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
        </draw>
        <direct>
```

```
                    d.currentNode.SetOldStyle();
                </direct>
                <reverse>
                    d.currentNode.SetCurrentNodeStyle();
                </reverse>
            </step>
        </else>
    </if>
    <step
        id="insertend"
        description="end"
        level="-1"
        >
        <draw/>
        <direct>
            d.insertEnd = true;
        </direct>
        <reverse>
            d.insertEnd = false;
        </reverse>
    </step>

</auto>

<auto id="insertintosubtree" description="insert auto">

    <if
        id="insertischildleaf"
        description="is child leaf"
        level="-1"
        test="d.currentNode.child1.isLeaf"
        >
        <then>
            <step
                id="insertyesitisleaf"
                description="yes it is"
                comment-ru="Поскольку текущий элемент находится на уровне,
                            предшествующем листьям, мы вставляем наш элемент
                            {0} среди его сыновей (если такого среди них нет)."
                comment-en="Becose current element is positioned on level, preceding
                            the level of leaves,
                            we insert our element {0} in-between it's sons."
                comment-args="new Integer(d.insertingElement)"
                level="1"
                >
                <draw>
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                </draw>
                <direct/>
                <reverse/>
            </step>
            <if
                id="insertischildrennumber2ornot"
                description="is children number 2"
                level="-1"
                test="d.currentNode.childrenNumber == 2"
                >
                <then>
                    <if
                        id="insertisinsertingelementintree1"
                        description="is inserting element in tree"
                        level="-1"
                        test="d.currentNode.child1.element == d.insertingElement ||
                              d.currentNode.child2.element == d.insertingElement"
                        >
                        <then>
                            <step
                                id="insertyesitisinsertingelementintree1"
                                description="yes it is"
                                comment-ru="Элемент {0} в дереве уже есть."
                                comment-en="Element {0} already exists."
                                comment-args="new Integer(d.insertingElement)"
                                level="1"
```

```
                    >
                    <draw>
                        d.visualizer.DrawTree();
                        d.visualizer.UpdateScreen();
                    </draw>
                    <direct/>
                    <reverse/>
                </step>
            </then>
            <else>
                <step
                    id="insertinsertelement1"
                    description="insert"
                    comment-ru="После вставки стало три сына,
                                и алгоритм вставки завершается."
                    comment-en="There are three sons after isertion.
                                Algorithm ends."
                    level="1"
                    >
                    <draw>
                        d.visualizer.DrawTree();
                        d.visualizer.UpdateScreen();
                    </draw>
                    <direct>
                        d.currentNode.SetOldStyle();
                        int returned =
                            d.currentNode.NewChildLeaf(d.insertingElement);
                        if(returned == 1)
                            d.currentNode = d.currentNode.child1;
                        if(returned == 2)
                            d.currentNode = d.currentNode.child2;
                        if(returned == 3)
                            d.currentNode = d.currentNode.child3;
                        d.currentNode.SetCurrentNodeStyle();
                    </direct>
                    <reverse>
                        d.currentNode.SetOldStyle();
                        d.currentNode.parent.DeleteChild(
                                            d.currentNode.thisChildNumber);
                        d.currentNode = d.currentNode.parent;
                        d.currentNode.SetCurrentNodeStyle();
                    </reverse>
                </step>
            </else>
        </if>
    </then>
    <else>
        <if
            id="insertisinsertingelementintree2"
            description="is inserting element in tree"
            level="-1"
            test="d.currentNode.child1.element == d.insertingElement ||
                  d.currentNode.child2.element == d.insertingElement ||
                  d.currentNode.child3.element == d.insertingElement"
            >
            <then>
                <step
                    id="insertyesitisinsertingelementintree2"
                    description="yes it is"
                    comment-ru="Элемент {0} в дереве уже есть."
                    comment-en="There is element {0} in tree."
                    comment-args="new Integer(d.insertingElement)"
                    level="1"
                    >
                    <draw>
                        d.visualizer.DrawTree();
                        d.visualizer.UpdateScreen();
                    </draw>
                    <direct/>
                    <reverse/>
                </step>
            </then>
            <else>
                <step
```

```
                            id="insertinsertelement2"
                            description="insert"
                            comment-ru="После вставки у родителя стало четыре сына."
                            comment-en="There are four sons after isertion."
                            level="1"
                            >
                            <draw>
                                d.visualizer.DrawTree();
                                d.visualizer.UpdateScreen();
                            </draw>
                            <direct>
                                d.currentNode.SetOldStyle();
                                int returned =
                                    d.currentNode.NewChildLeaf(d.insertingElement);
                                if(returned == 1)
                                    d.currentNode = d.currentNode.child1;
                                if(returned == 2)
                                    d.currentNode = d.currentNode.child2;
                                if(returned == 3)
                                    d.currentNode = d.currentNode.child3;
                                if(returned == 4)
                                    d.currentNode = d.currentNode.child4;
                                d.currentNode.SetCurrentNodeStyle();
                            </direct>
                            <reverse>
                                d.currentNode.SetOldStyle();
                                d.currentNode.parent.DeleteChild(
                                                    d.currentNode.thisChildNumber);
                                d.currentNode = d.currentNode.parent;
                                d.currentNode.SetCurrentNodeStyle();
                            </reverse>
                        </step>
                        <step
                            id="insertinsertelement2"
                            description="insert"
                            level="-1"
                            >
                            <draw/>
                            <direct>
                            </direct>
                            <reverse>
                                d.currentNode.parent.SetOldStyle();
                            </reverse>
                        </step>
                    </else>
                </if>
            </else>
        </if>
    </then>
    <else>
        <if
            id="insertisserchingelementin1subtree"
            description="is serching element in first subtree"
            level="-1"
            test="d.currentNode.low2 &gt; d.insertingElement"
            >
            <then>
                <step
                    id="insertltlow2"
                    description="lt low2"
                    comment-ru="Искомый элемент меньше наименьшего элемента
                                во втором поддереве ({0}),
                                поэтому поиски будем продолжать в первом поддереве."
                    comment-en="Sought element is less than least element
                                of second subtree ({0}),
                                so our search we'll continue in first subtree."
                    comment-args="new Integer(d.currentNode.low2)"
                    level="1"
                    >
                    <draw>
                        d.visualizer.DrawTree();
                        d.visualizer.UpdateScreen();
                    </draw>
                    <direct/>
```

```
                <reverse/>
            </step>
            <step
                id="insertcurrentelementinit2"
                description="init"
                level="-1"
                >
                <draw/>
                <direct>
                    d.currentNode.SetOldStyle();
                    d.currentNode = d.currentNode.child1;
                    d.currentNode.SetCurrentNodeStyle();
                </direct>
                <reverse>
                    d.currentNode.SetOldStyle();
                    d.currentNode = d.currentNode.parent;
                    d.currentNode.SetCurrentNodeStyle();
                </reverse>
            </step>
            <call-auto id="insertintosubtree"/>
        </then>
        <else>
            <if
                id="insertischildrennumber2ornot"
                description="is children number 2"
                level="-1"
                test="d.currentNode.childrenNumber == 2"
                >
                <then>
                    <step
                        id="insertgtlow2"
                        description="gt low2"
                        comment-ru="Искомый элемент, больше наименьшего элемента
                                    во втором поддереве ({0}),
                                    поэтому поиски будем продолжать
                                    во втором поддереве."
                        comment-en="Sought element is greater than least element
                                    of second subtree ({0}),
                                    so our search we'll continue
                                    in second subtree."
                        comment-args="new Integer(d.currentNode.low2)"
                        level="1"
                        >
                        <draw>
                            d.visualizer.DrawTree();
                            d.visualizer.UpdateScreen();
                        </draw>
                        <direct/>
                        <reverse/>
                    </step>
                    <step
                        id="insertcurrentelementinit3"
                        description="init"
                        level="-1"
                        >
                        <draw/>
                        <direct>
                            d.currentNode.SetOldStyle();
                            d.currentNode = d.currentNode.child2;
                            d.currentNode.SetCurrentNodeStyle();
                        </direct>
                        <reverse>
                            d.currentNode.SetOldStyle();
                            d.currentNode = d.currentNode.parent;
                            d.currentNode.SetCurrentNodeStyle();
                        </reverse>
                    </step>
                    <call-auto id="insertintosubtree"/>
                </then>
                <else>
                    <if
                        id="insertisserchingelementin2subtree"
                        description="is serching element in second subtree"
                        level="-1"
```

```
test="d.currentNode.low3 &gt; d.insertingElement"
>
<then>
    <step
        id="insertltlow3"
        description="lt low3"
        comment-ru="Искомый элемент больше
                    наименьшего элемента
                    во втором поддереве ({0}) и
                    меньше наименьшего элемента
                    в третьем ({1}),
                    поэтому поиски будем продолжать
                    во втором поддереве."
        comment-en="Sought element is greater
                    than least element
                    of second subtree ({0})
                    and less than least element
                    of the third subtree ({1}),
                    so our search we'll continue
                    in second subtree."
        comment-args="new Integer(d.currentNode.low2),
                      new Integer(d.currentNode.low3)"
        level="1"
        >
        <draw>
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
        </draw>
        <direct/>
        <reverse/>
    </step>
    <step
        id="insertcurrentelementinit4"
        description="init"
        level="-1"
        >
        <draw/>
        <direct>
            d.currentNode.SetOldStyle();
            d.currentNode = d.currentNode.child2;
            d.currentNode.SetCurrentNodeStyle();
        </direct>
        <reverse>
            d.currentNode.SetOldStyle();
            d.currentNode = d.currentNode.parent;
            d.currentNode.SetCurrentNodeStyle();
        </reverse>
    </step>
    <call-auto id="insertintosubtree"/>
</then>
<else>
    <step
        id="insertgtlow3"
        description="gt low3"
        comment-ru="Искомый элемент больше
                    наименьшего элемента
                    в третьем поддереве ({0}),
                    поэтому поиски будем продолжать
                    в  третьем поддереве."
        comment-en="Sought element is greater
                    than least element
                    of third subtree ({0}),
                    so our search we'll continue
                    in third subtree."
        comment-args="new Integer(d.currentNode.low3)"
        level="1"
        >
        <draw>
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
        </draw>
        <direct/>
        <reverse/>
    </step>
```

```
                            <step
                                id="insertcurrentelementinit5"
                                description="init"
                                level="-1"
                                >
                                <draw/>
                                <direct>
                                    d.currentNode.SetOldStyle();
                                    d.currentNode = d.currentNode.child3;
                                    d.currentNode.SetCurrentNodeStyle();
                                </direct>
                                <reverse>
                                    d.currentNode.SetOldStyle();
                                    d.currentNode = d.currentNode.parent;
                                    d.currentNode.SetCurrentNodeStyle();
                                </reverse>
                            </step>
                            <call-auto id="insertintosubtree"/>
                        </else>
                    </if>
                </else>
            </if>
        </else>
    </if>
    <if
        id="insertis4"
        description="is 4 children"
        level="-1"
        test="d.currentNode.parent.childrenNumber == 4">
        <then>
            <step
                id="insertgotoparent1"
                description="to parent"
                level="-1"
                >
                <draw/>
                <direct>
                    d.tree.SetDefaultStyles();
                    d.myStack.pushInteger(d.currentNode.thisChildNumber);
                    d.currentNode = d.currentNode.parent;
                    d.currentNode.SetCurrentNodeStyle();
                </direct>
                <reverse>
                    d.currentNode.SetOldStyle();
                    int returned = d.myStack.popInteger();
                    if(returned == 1)
                        d.currentNode = d.currentNode.child1;
                    if(returned == 2)
                        d.currentNode = d.currentNode.child2;
                    if(returned == 3)
                        d.currentNode = d.currentNode.child3;
                    if(returned == 4)
                        d.currentNode = d.currentNode.child4;
                    d.currentNode.SetCurrentNodeStyle();
                </reverse>
            </step>
            <step
                id="insertinsertelement3"
                description="insert"
                comment-ru="У узла стало 4 сына, поэтому мы разбиваем его на два
                            тем самым добавляя сына его родителю."
                comment-en="Current node have four children,
                            so we divide this node in
                            two adding by this operation child to his parent."
                level="1"
                >
                <draw>
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                </draw>
                <direct>
                    d.currentNode.DeleteChild(4);
                    d.currentNode.DeleteChild(3);
                    int returned =
```

```
                d.currentNode.
                    parent.NewChildInterior(d.currentNode.child3,
                                            d.currentNode.child4);
            if(returned == 2)
                d.currentNode.parent.child2.SetCurrentNodeStyle();
            if(returned == 3)
                d.currentNode.parent.child3.SetCurrentNodeStyle();
            if(returned == 4)
                d.currentNode.parent.child4.SetCurrentNodeStyle();
        </direct>
        <reverse>
            Node tempNode;
            if(d.currentNode.thisChildNumber == 1)
            {
                tempNode = d.currentNode.parent.child2;
                d.currentNode.parent.DeleteChild(2);
                d.currentNode.AddChild(tempNode.child1);
                d.currentNode.AddChild(tempNode.child2);
            }
            if(d.currentNode.thisChildNumber == 2)
            {
                tempNode = d.currentNode.parent.child3;
                d.currentNode.parent.DeleteChild(3);
                d.currentNode.AddChild(tempNode.child1);
                d.currentNode.AddChild(tempNode.child2);
            }
            if(d.currentNode.thisChildNumber == 3)
            {
                tempNode = d.currentNode.parent.child4;
                d.currentNode.parent.DeleteChild(4);
                d.currentNode.AddChild(tempNode.child1);
                d.currentNode.AddChild(tempNode.child2);
            }
            if(d.currentNode.thisChildNumber == 4)
            {
                tempNode = d.currentNode.parent.child3;
                d.currentNode.parent.DeleteChild(3);
                d.currentNode.AddChild(tempNode.child1);
                d.currentNode.AddChild(tempNode.child2);
            }
            d.currentNode.SetOldStyle();
            int returned = d.currentNode.FindChild(d.insertingElement);
            if(returned == 1)
                d.currentNode.child1.SetCurrentNodeStyle();
            if(returned == 2)
                d.currentNode.child2.SetCurrentNodeStyle();
            if(returned == 3)
                d.currentNode.child3.SetCurrentNodeStyle();
            if(returned == 4)
                d.currentNode.child4.SetCurrentNodeStyle();

            d.currentNode.SetCurrentNodeStyle();
        </reverse>
    </step>
    <step
        id="insertgonewnode"
        description="to new node"
        level="-1"
        >
        <draw/>
        <direct>
            if(d.currentNode.thisChildNumber == 1)
            {
                d.currentNode.SetOldStyle();
                d.currentNode.parent.child2.SetOldStyle();
            }
            if(d.currentNode.thisChildNumber == 2)
            {
                d.currentNode.SetOldStyle();
                d.currentNode.parent.child3.SetOldStyle();
            }
            if(d.currentNode.thisChildNumber == 3)
            {
                d.currentNode.SetOldStyle();
```

```
                    d.currentNode.parent.child4.SetOldStyle();
                }
            </direct>
            <reverse>
                if(d.currentNode.thisChildNumber == 1)
                {
                    d.currentNode.SetCurrentNodeStyle();
                    d.currentNode.parent.child2.SetCurrentNodeStyle();
                }
                if(d.currentNode.thisChildNumber == 2)
                {
                    d.currentNode.SetCurrentNodeStyle();
                    d.currentNode.parent.child3.SetCurrentNodeStyle();
                }
                if(d.currentNode.thisChildNumber == 3)
                {
                    d.currentNode.SetCurrentNodeStyle();
                    d.currentNode.parent.child4.SetCurrentNodeStyle();
                }
            </reverse>
        </step>
        <if id="insertisnot42"
            description="is not 4 children"
            level="-1"
            test="d.currentNode.parent.childrenNumber != 4">
            <then>
            </then>
        </if>
    </then>
    <else>
        <step
            id="insertgotoparent2"
            description="to parent"
            comment-ru="Сверяем значения наименьших элементов в поддеревьях."
            comment-en="Compare least values of subtrees."
            level="-1"
            >
            <draw/>
            <direct>
                d.currentNode.SetOldStyle();
                d.myStack.pushInteger(d.currentNode.thisChildNumber);
                d.currentNode = d.currentNode.parent;
                d.currentNode.SetCurrentNodeStyle();
                d.myStack.pushInteger(d.currentNode.low1);
                d.myStack.pushInteger(d.currentNode.low2);
                d.myStack.pushInteger(d.currentNode.low3);
                d.myStack.pushInteger(d.currentNode.low4);
                if(d.currentNode.childrenNumber == 1)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                }
                if(d.currentNode.childrenNumber == 2)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
                }
                if(d.currentNode.childrenNumber == 3)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
                    d.currentNode.low3 = d.currentNode.child3.low1;
                }
            </direct>
            <reverse>
                d.currentNode.SetOldStyle();
                d.currentNode.low4 = d.myStack.popInteger();
                d.currentNode.low3 = d.myStack.popInteger();
                d.currentNode.low2 = d.myStack.popInteger();
                d.currentNode.low1 = d.myStack.popInteger();
                int returned = d.myStack.popInteger();
                if(returned == 1)
                    d.currentNode = d.currentNode.child1;
                if(returned == 2)
                    d.currentNode = d.currentNode.child2;
```

```
                            if(returned == 3)
                                d.currentNode = d.currentNode.child3;
                            if(returned == 4)
                                d.currentNode = d.currentNode.child4;
                            d.currentNode.SetCurrentNodeStyle();
                        </reverse>
                    </step>
                    <step
                        id="insertreturnsettings3"
                        description="retuns settings"
                        level="-1"
                        >
                        <draw/>
                        <direct>
                            d.currentNode.SetOldStyle();
                        </direct>
                        <reverse>
                            d.currentNode.SetCurrentNodeStyle();
                        </reverse>
                    </step>
                </else>
            </if>
        </else>
    </if>

</auto>

<auto id="delete" description="delete auto">

    <if
        id="deleteistreeempty"
        description="is tree empty"
        level="-1"
        test="d.tree.childrenNumber == 0"
        >
        <then>
            <step
                id="deletetreeempty"
                description="tree is empty"
                comment-ru="Дерево пусто, и удаляемого элемента в нем нет."
                comment-en="Tree is empty.  So required element is certainly missing."
                level="1"
                >
                <draw>
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                </draw>
                <direct/>
                <reverse/>
            </step>
        </then>
        <else>
            <step
                id="deletecurrentelementinit1"
                description="init"
                level="-1"
                >
                <draw>
                </draw>
                <direct>
                    d.currentNode = d.tree.rootNode1;
                    d.currentNode.SetCurrentNodeStyle();
                </direct>
                <reverse>
                    d.currentNode.SetOldStyle();
                </reverse>
            </step>
            <step
                id="deletebeginsfromroot"
                description="begins from root"
                comment-ru="Сначала найдем {0}.
                            При этом повторяются действия алгоритма поиска."
                comment-en="At first we'll find {0}.
                            And search algorithm's actions are repeated."
```

```
            comment-args="new Integer(d.deletingElement)"
            level="1"
            >
            <draw>
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
            </draw>
            <direct/>
            <reverse/>
        </step>
        <call-auto id="deletefromsubtree"/>
    </else>
</if>
<if id="deleteis10"
    description="is 1 children"
    level="-1"
    test="d.tree.rootNode1.childrenNumber == 1">
    <then>
        <step
            id="deleteinsertelement0"
            description="insert"
            comment-ru="У узла остался один сын, поэтому мы его удаляем,
                        а его сын становится корнем."
            comment-en="Node have only one son, so we'll remove him,
                        and his son will become the root."
            level="1"
            >
            <draw>
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
            </draw>
            <direct>
                d.tree.DeleteChild(1);
                d.tree.AddChild(d.tree.rootNode1.child1);
                d.tree.rootNode1.SetCurrentNodeStyle();
            </direct>
            <reverse>
                d.tree.rootNode1.SetOldStyle();
                d.tree.DeleteChild(1);
                d.tree.NewChildInterior(d.tree.rootNode1);
            </reverse>
        </step>
        <step
            id="deletereturnssetings0"
            description="returns setings"
            level="-1"
            >
            <draw>
            </draw>
            <direct>
                d.tree.rootNode1.SetOldStyle();
            </direct>
            <reverse>
                d.tree.rootNode1.SetCurrentNodeStyle();
            </reverse>
        </step>
    </then>
    <else>
        <if
            id="deleteisroot0"
            description="is root"
            level="-1"
            test="d.currentNode != null  &amp;&amp; !d.currentNode.isRoot">
            <then>
                <step
                    id="deletegotoparent2"
                    description="to parent"
                    comment-ru="Сверяем значения наименьших элементов в поддеревьях.
                                Если был удален первый элемент в некотором поддереве,
                                то мы возвращаем новое наименьшее значение,
                                поднимаясь вверх по дереву
                                до тех пор пока поддерево, из которого мы вернулись,
                                является первым.
                                Как только это стало не так,
```

```
                мы для данного элемента меняем наименьшее значение
                того поддерева из
                которого мы вернулись на новое."
            comment-en="Compare least values of subtrees."
            level="1"
            >
            <draw>
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
            </draw>
            <direct>
                d.currentNode.SetOldStyle();
                d.currentNode = d.currentNode.parent;
                d.currentNode.SetCurrentNodeStyle();
                d.myStack.pushInteger(d.currentNode.low1);
                d.myStack.pushInteger(d.currentNode.low2);
                d.myStack.pushInteger(d.currentNode.low3);
                d.myStack.pushInteger(d.currentNode.low4);
                if(d.currentNode.childrenNumber == 1)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                }
                if(d.currentNode.childrenNumber == 2)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
                }
                if(d.currentNode.childrenNumber == 3)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
                    d.currentNode.low3 = d.currentNode.child3.low1;
                }
            </direct>
            <reverse>
                d.currentNode.SetOldStyle();
                d.currentNode.low4 = d.myStack.popInteger();
                d.currentNode.low3 = d.myStack.popInteger();
                d.currentNode.low2 = d.myStack.popInteger();
                d.currentNode.low1 = d.myStack.popInteger();
                int returned = d.currentNode.FindChild(d.deletingElement);
                if(returned == 1)
                    d.currentNode = d.currentNode.child1;
                if(returned == 2)
                    d.currentNode = d.currentNode.child2;
                if(returned == 3)
                    d.currentNode = d.currentNode.child3;
                d.currentNode.SetCurrentNodeStyle();
            </reverse>
        </step>
    </then>
</if>
<step
    id="deletereturnssetings1"
    description="returns setings"
    level="1"
    >
    <draw>
        d.visualizer.DrawTree();
        d.visualizer.UpdateScreen();
    </draw>
    <direct>
        d.tree.rootNode1.SetOldStyle();
    </direct>
    <reverse>
        d.tree.rootNode1.SetCurrentNodeStyle();
    </reverse>
</step>
</else>
</if>
<step
    id="deleteend"
    description="end"
    level="-1"
```

```
        >
        <draw/>
        <direct>
            d.deleteEnd = true;
        </direct>
        <reverse>
            d.deleteEnd = false;
        </reverse>
    </step>

</auto>

<auto id="deletefromsubtree" description="delete auto">

    <if
        id="deleteischildleaf"
        description="is child leaf"
        level="-1"
        test="d.currentNode.child1.isLeaf"
        >
        <then>
            <step
                id="deleteyesitisleaf"
                description="yes it is"
                comment-ru="Поскольку текущий элемент находится на уровне,
                            предшествующем листьям, мы удаляем наш элемент {0}
                            из его сыновей,
                            если он среди них есть"
                comment-en="Becose current element is positioned on level, preceding
                            the level of leaves, we delete our element {0}
                            in-between it's sons."
                comment-args="new Integer(d.deletingElement)"
                level="1"
                >
                <draw>
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                </draw>
                <direct/>
                <reverse/>
            </step>
            <if
                id="deleteischildrennumber2ornot"
                description="is children number 2"
                level="-1"
                test="d.currentNode.childrenNumber == 2"
                >
                <then>
                    <if
                        id="deleteisdeletingelementintree1"
                        description="is deleting element in tree"
                        level="-1"
                        test="d.currentNode.child1.element != d.deletingElement
                              &amp;&amp;
                              d.currentNode.child2.element != d.deletingElement"
                        >
                        <then>
                            <step
                                id="deletenoitisnotdeletingelementintree1"
                                description="no it is not"
                                comment-ru="Элемента {0} в дереве нет"
                                comment-en="There is element {0} in tree"
                                comment-args="new Integer(d.deletingElement)"
                                level="1"
                                >
                                <draw>
                                    d.visualizer.DrawTree();
                                    d.visualizer.UpdateScreen();
                                </draw>
                                <direct/>
                                <reverse/>
                            </step>
                        </then>
                        <else>
```

```
                <step
                    id="deleteinsertelement1"
                    description="insert"
                    comment-ru="После удаления остался один сын."
                    comment-en="There is only one son
                                in tree after removing."
                    level="1"
                    >
                    <draw>
                        d.visualizer.DrawTree();
                        d.visualizer.UpdateScreen();
                    </draw>
                    <direct>
                        int returned =
                            d.currentNode.FindChild(d.deletingElement);
                        if(returned == 1)
                            d.currentNode.DeleteChild(1);
                        if(returned == 2)
                            d.currentNode.DeleteChild(2);
                    </direct>
                    <reverse>
                        d.currentNode.NewChildLeaf(d.deletingElement);
                        d.visualizer.DrawTree();
                        d.visualizer.UpdateScreen();
                    </reverse>
                </step>
            </else>
        </if>
    </then>
    <else>
        <if
            id="deleteisinsertingelementintree2"
            description="is inserting element in tree"
            level="-1"
            test="d.currentNode.child1.element != d.deletingElement
                                &amp;&amp;
                  d.currentNode.child2.element != d.deletingElement
                                &amp;&amp;
                  d.currentNode.child3.element != d.deletingElement"
            >
            <then>
                <step
                    id="deleteyesitisinsertingelementintree2"
                    description="yes it is"
                    comment-ru="Элемента {0} в дереве нет."
                    comment-en="There is element {0} in tree."
                    comment-args="new Integer(d.deletingElement)"
                    level="1"
                    >
                    <draw>
                        d.visualizer.DrawTree();
                        d.visualizer.UpdateScreen();
                    </draw>
                    <direct/>
                    <reverse/>
                </step>
            </then>
            <else>
                <step
                    id="deleteinsertelement2"
                    description="insert"
                    comment-ru="Поскольку после вставки
                                стало два сына в данном поддереве,
                                осталось сверить в узлах значения
                                наименьших элементов в поддеревьях."
                    comment-en="There are two sons after removing.
                                We should climb the tree and, maybe,
                                correct node least element
                                info values of subtrees."
                    level="1"
                    >
                    <draw>
                        d.visualizer.DrawTree();
                        d.visualizer.UpdateScreen();
```

```
                        </draw>
                        <direct>
                            int returned =
                                d.currentNode.FindChild(d.deletingElement);
                            if(returned == 1)
                                d.currentNode.DeleteChild(1);
                            if(returned == 2)
                                d.currentNode.DeleteChild(2);
                            if(returned == 3)
                                d.currentNode.DeleteChild(3);
                            d.currentNode.SetCurrentNodeStyle();
                        </direct>
                        <reverse>
                            d.currentNode.NewChildLeaf(d.deletingElement);
                        </reverse>
                    </step>
                </else>
            </if>
        </else>
    </if>
</then>
<else>
    <if
        id="deleteisdeletingelementin1subtree"
        description="is deleting element in first subtree"
        level="-1"
        test="d.currentNode.low2 &gt; d.deletingElement"
        >
        <then>
            <step
                id="deleteltlow2"
                description="lt low2"
                comment-ru="Искомый элемент меньше наименьшего элемента
                            во втором поддереве ({0}),
                            поэтому поиски будем продолжать в первом."
                comment-en="Sought element is less than least element
                            of second subtree ({0}),
                            so our search we'll continue in first subtree."
                comment-args="new Integer(d.currentNode.low2)"
                level="1"
                >
                <draw>
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                </draw>
                <direct/>
                <reverse/>
            </step>
            <step
                id="deletecurrentelementinit2"
                description="init"
                level="-1"
                >
                <draw>
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                </draw>
                <direct>
                    d.currentNode.SetOldStyle();
                    d.currentNode = d.currentNode.child1;
                    d.currentNode.SetCurrentNodeStyle();
                </direct>
                <reverse>
                    d.currentNode.SetOldStyle();
                    d.currentNode = d.currentNode.parent;
                    d.currentNode.SetCurrentNodeStyle();
                </reverse>
            </step>
            <call-auto id="deletefromsubtree"/>
        </then>
        <else>
            <if
                id="deleteischildrennumber2ornot"
                description="is children number 2"
```

```
level="-1"
test="d.currentNode.childrenNumber == 2"
>
<then>
    <step
        id="insertgtlow2"
        description="gt low2"
        comment-ru="Искомый элемент, больше наименьшего элемента
                    во втором поддереве ({0}),
                    поэтому поиски будем продолжать
                    во втором поддереве."
        comment-en="Sought element is greater than least element
                    of second subtree ({0}),
                    so our search we'll continue
                    in second subtree."
        comment-args="new Integer(d.currentNode.low2)"
        level="1"
        >
        <draw>
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
        </draw>
        <direct/>
        <reverse/>
    </step>
    <step
        id="deletecurrentelementinit3"
        description="init"
        level="-1"
        >
        <draw>
        </draw>
        <direct>
            d.currentNode.SetOldStyle();
            d.currentNode = d.currentNode.child2;
            d.currentNode.SetCurrentNodeStyle();
        </direct>
        <reverse>
            d.currentNode.SetOldStyle();
            d.currentNode = d.currentNode.parent;
            d.currentNode.SetCurrentNodeStyle();
        </reverse>
    </step>
    <call-auto id="deletefromsubtree"/>
</then>
<else>
    <if
        id="deleteisserchingelementin2subtree"
        description="is serching element in second subtree"
        level="-1"
        test="d.currentNode.low3 &gt; d.deletingElement"
        >
        <then>
            <step
                id="deleteltlow3"
                description="lt low3"
                comment-ru="Искомый элемент больше
                            наименьшего элемента
                            во втором поддереве ({0}) и
                            меньше наименьшего элемента
                            в третьем ({1}),
                            поэтому поиски будем продолжать
                            во втором поддереве."
                comment-en="Sought element is greater than
                            least element of second subtree ({0})
                            and less than least element
                            of the third subtree ({1}),
                            so our search we'll continue
                            in second subtree."
                comment-args="new Integer(d.currentNode.low2),
                              new Integer(d.currentNode.low3)"
                level="1"
                >
                <draw>
```

```
                                    d.visualizer.DrawTree();
                                    d.visualizer.UpdateScreen();
                                </draw>
                                <direct/>
                                <reverse/>
                            </step>
                            <step
                                id="deletecurrentelementinit4"
                                description="init"
                                level="-1"
                                >
                                <draw>
                                </draw>
                                <direct>
                                    d.currentNode.SetOldStyle();
                                    d.currentNode = d.currentNode.child2;
                                    d.currentNode.SetCurrentNodeStyle();
                                </direct>
                                <reverse>
                                    d.currentNode.SetOldStyle();
                                    d.currentNode = d.currentNode.parent;
                                    d.currentNode.SetCurrentNodeStyle();
                                </reverse>
                            </step>
                            <call-auto id="deletefromsubtree"/>
                        </then>
                        <else>
                            <step
                                id="deletegtlow3"
                                description="gt low3"
                                comment-ru="Искомый элемент больше наименьшего
                                            элемента в третьем поддереве ({0}),
                                            поэтому поиски будем продолжать
                                            в третьем поддереве."
                                comment-en="Sought element is greater than least
                                            element of third subtree ({0}),
                                            so our search we'll continue
                                            in third subtree."
                                comment-args="new Integer(d.currentNode.low3)"
                                level="1"
                                >
                                <draw>
                                    d.visualizer.DrawTree();
                                    d.visualizer.UpdateScreen();
                                </draw>
                                <direct/>
                                <reverse/>
                            </step>
                            <step
                                id="deletecurrentelementinit5"
                                description="init"
                                level="-1"
                                >
                                <draw>
                                </draw>
                                <direct>
                                    d.currentNode.SetOldStyle();
                                    d.currentNode = d.currentNode.child3;
                                    d.currentNode.SetCurrentNodeStyle();
                                </direct>
                                <reverse>
                                    d.currentNode.SetOldStyle();
                                    d.currentNode = d.currentNode.parent;
                                    d.currentNode.SetCurrentNodeStyle();
                                </reverse>
                            </step>
                            <call-auto id="deletefromsubtree"/>
                        </else>
                    </if>
                </else>
            </if>
        </else>
    </if>
    <if id="deleteis1"
```

```
description="is 1 children"
level="-1"
test="d.currentNode.childrenNumber == 1">
<then>
    <if
        id="deleteisit1child"
        description="is it 1 child"
        level="-1"
        test="d.currentNode.thisChildNumber == 1"
        >
        <then>
            <if
                id="deleteis3childrenin2child"
                description="is it 1 child"
                level="-1"
                test="d.currentNode.parent.child2.childrenNumber == 3"
                >
                <then>
                    <step
                        id="delete3childrenin2child"
                        description="insert"
                        comment-ru="Отнимаем один элемент у брата
                                    (у него их три)."
                        comment-en="We take one element from brother."
                        level="1"
                        >
                        <draw>
                            d.visualizer.DrawTree();
                            d.visualizer.UpdateScreen();
                        </draw>
                        <direct>
                            Node tempNode =
                                d.currentNode.parent.child2.child1;
                            d.currentNode.parent.child2.DeleteChild(1);
                            d.currentNode.AddChild(tempNode);
                        </direct>
                        <reverse>
                            Node tempNode = d.currentNode.child2;
                            d.currentNode.DeleteChild(2);
                             d.currentNode.parent.
                                 child2.AddChild(tempNode);
                        </reverse>
                    </step>
                </then>
                <else>
                    <step id="delete2childrenin2child"
                        description="insert"
                        comment-ru="Объединяем данный узел с его братом."
                        comment-en="We combine this node and
                                    it's brother."
                        level="1"
                        >
                        <draw>
                            d.visualizer.DrawTree();
                            d.visualizer.UpdateScreen();
                        </draw>
                        <direct>
                            Node tempNode = d.currentNode.child1;
                            d.currentNode.parent.DeleteChild(1);
                            d.currentNode.parent.
                                child1.AddChild(tempNode);
                            d.currentNode = d.currentNode.parent.child1;
                            d.currentNode.SetCurrentNodeStyle();
                        </direct>
                        <reverse>
                            d.currentNode.SetOldStyle();
                            Node tempNode = d.currentNode.child1;
                            d.currentNode.DeleteChild(1);
                            d.currentNode.parent.
                                NewChildInterior(tempNode);
                            d.currentNode = d.currentNode.parent.child1;
                            d.currentNode.SetCurrentNodeStyle();
                        </reverse>
                    </step>
```

```
            </else>
        </if>
    </then>
</if>
<if
    id="deleteisit2child"
    description="is it 2 child"
    level="-1"
    test="d.currentNode.thisChildNumber == 2"
    >
    <then>
        <if
            id="deleteis2childreninparent"
            description="is it 2 child"
            level="-1"
            test="d.currentNode.parent.childrenNumber == 2"
            >
            <then>
                <if
                    id="deleteis3childrenin1child"
                    description="is it 1 child"
                    level="-1"
                    test="d.currentNode.parent.child1.childrenNumber
                          == 3"
                    >
                    <then>
                        <step
                            id="delete3childrenin1child"
                            description="insert"
                            comment-ru="Отнимаем один элемент у брата
                                        (у него их три)."
                            comment-en="We take one element
                                        from brother."
                            level="1"
                            >
                            <draw>
                                d.visualizer.DrawTree();
                                d.visualizer.UpdateScreen();
                            </draw>
                            <direct>
                                Node tempNode =
                                    d.currentNode.
                                        parent.child1.child3;
                                d.currentNode.parent.child1.
                                DeleteChild(3);
                                d.currentNode.AddChild(tempNode);
                            </direct>
                            <reverse>
                                Node tempNode = d.currentNode.child1;
                                d.currentNode.DeleteChild(1);
                                d.currentNode.parent.child1.
                                AddChild(tempNode);
                            </reverse>
                        </step>
                    </then>
                    <else>
                        <step id="delete2childrenin1child"
                            description="insert"
                            comment-ru="Объединяем данный узел
                                        с его братом."
                            comment-en="We combine this node and
                                        it's brother."
                            level="1"
                            >
                            <draw>
                                d.visualizer.DrawTree();
                                d.visualizer.UpdateScreen();
                            </draw>
                            <direct>
                                Node tempNode = d.currentNode.child1;
                                d.currentNode.parent.DeleteChild(2);
                                d.currentNode.parent.child1.
                                AddChild(tempNode);
                                d.currentNode =
```

```
                    d.currentNode.parent.child1;
                    d.currentNode.SetCurrentNodeStyle();
                </direct>
                <reverse>
                    d.currentNode.SetOldStyle();
                    Node tempNode = d.currentNode.child3;
                    d.currentNode.parent.child1.
                    DeleteChild(3);
                    d.currentNode.parent.
                    NewChildInterior(tempNode);
                    d.currentNode =
                    d.currentNode.parent.child2;
                    d.currentNode.SetCurrentNodeStyle();
                </reverse>
            </step>
        </else>
    </if>
</then>
<else>
    <if
        id="deleteis3childrenin1child1"
        description="is it 1 child"
        level="-1"
        test="d.currentNode.parent.child1.childrenNumber
                == 3"
        >
        <then>
            <step
                id="delete3childrenin1child1"
                description="insert"
                comment-ru="Отнимаем один элемент у брата
                            (у него их три)."
                comment-en="We take one element
                     from brother."
                level="1"
                >
                <draw>
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                </draw>
                <direct>
                    Node tempNode =
                        d.currentNode.
                            parent.child1.child3;
                    d.currentNode.parent.child1.
                    DeleteChild(3);
                    d.currentNode.AddChild(tempNode);
                </direct>
                <reverse>
                    Node tempNode = d.currentNode.child1;
                    d.currentNode.DeleteChild(1);
                    d.currentNode.parent.child1.
                    AddChild(tempNode);
                </reverse>
            </step>
        </then>
        <else>
            <if
                id="deleteis3childrenin3child"
                description="is it 1 child"
                level="-1"
                test="d.currentNode.parent.child3.
                    childrenNumber == 3"
                >
                <then>
                    <step
                        id="delete3childrenin3child"
                        description="insert"
                        comment-ru="Отнимаем один элемент
                                    у брата
                                    (у него их три)."
                        comment-en="We take one element
                                    from brother."
                        level="1"
```

```
                                        >
                                        <draw>
                                            d.visualizer.DrawTree();
                                            d.visualizer.UpdateScreen();
                                        </draw>
                                        <direct>
                                            Node tempNode =
                                                d.currentNode.parent.
                                            child3.child1;
                                            d.currentNode.parent.
                                            child3.DeleteChild(1);
                                            d.currentNode.
                                                AddChild(tempNode);
                                        </direct>
                                        <reverse>
                                            Node tempNode =
                                            d.currentNode.child2;
                                            d.currentNode.DeleteChild(2);
                                            d.currentNode.parent.child3.
                                            AddChild(tempNode);
                                        </reverse>
                                    </step>
                                </then>
                                <else>
                                    <step id=
                                          "delete2childrenin1and2child"
                                        description="insert"
                                        comment-ru="Объединяем данный
                                                    узел
                                                    с его братом."
                                        comment-en="We combine this node
                                                    and it's brother."
                                        level="1"
                                        >
                                        <draw>
                                            d.visualizer.DrawTree();
                                            d.visualizer.UpdateScreen();
                                        </draw>
                                        <direct>
                                            Node tempNode =
                                                 d.currentNode.child1;
                                            d.currentNode.parent.
                                                 DeleteChild(2);
                                            d.currentNode.parent.child1.
                                                 AddChild(tempNode);
                                            d.currentNode =
                                                d.currentNode.
                                                    parent.child1;
                                            d.currentNode.
                                                 SetCurrentNodeStyle();
                                        </direct>
                                        <reverse>
                                            d.currentNode.SetOldStyle();
                                            Node tempNode =
                                                d.currentNode.child3;
                                            d.currentNode.parent.child1.
                                                DeleteChild(3);
                                            d.currentNode.parent.
                                                NewChildInterior(
                                                    tempNode);
                                            d.currentNode =
                                                d.currentNode.
                                                    parent.child2;
                                            d.currentNode.
                                                SetCurrentNodeStyle();
                                        </reverse>
                                    </step>
                                </else>
                            </if>
                        </else>
                    </if>
                </else>
            </if>
        </then>
```

```
</if>
<if
    id="deleteisit3child"
    description="is it 3 child"
    level="-1"
    test="d.currentNode.thisChildNumber == 3"
    >
    <then>
        <if
            id="deleteis3childrenin2child1"
            description="is it 1 child"
            level="-1"
            test="d.currentNode.parent.child2.childrenNumber == 3"
            >
            <then>
                <step
                    id="delete3childrenin2child1"
                    description="insert"
                    comment-ru="Отнимаем один элемент у брата
                                (у него их три)."
                    comment-en="We take one element from brother."
                    level="1"
                    >
                    <draw>
                        d.visualizer.DrawTree();
                        d.visualizer.UpdateScreen();
                    </draw>
                    <direct>
                        Node tempNode =
                             d.currentNode.parent.child2.child3;
                        d.currentNode.parent.child2.DeleteChild(3);
                        d.currentNode.AddChild(tempNode);
                    </direct>
                    <reverse>
                        Node tempNode = d.currentNode.child1;
                        d.currentNode.DeleteChild(1);
                        d.currentNode.parent.
                            child2.AddChild(tempNode);
                    </reverse>
                </step>
            </then>
            <else>
                <step id="delete2childrenin2child1"
                    description="insert"
                    comment-ru="Объединяем данный узел с его братом."
                    comment-en="We combine this node and
                                it's brother."
                    level="1"
                    >
                    <draw>
                        d.visualizer.DrawTree();
                        d.visualizer.UpdateScreen();
                    </draw>
                    <direct>
                        Node tempNode = d.currentNode.child1;
                        d.currentNode.parent.DeleteChild(3);
                        d.currentNode.parent.
                            child2.AddChild(tempNode);
                        d.currentNode = d.currentNode.parent.child2;
                        d.currentNode.SetCurrentNodeStyle();
                    </direct>
                    <reverse>
                        d.currentNode.SetOldStyle();
                        Node tempNode = d.currentNode.child3;
                        d.currentNode.parent.child2.DeleteChild(3);
                        d.currentNode.parent.
                            NewChildInterior(tempNode);
                        d.currentNode = d.currentNode.parent.child3;
                        d.currentNode.SetCurrentNodeStyle();
                    </reverse>
                </step>
            </else>
        </if>
    </then>
```

```
        </if>
        <step
            id="deletegotoparent"
            description="to parent"
            level="-1"
            >
            <draw/>
            <direct>
                if(d.currentNode.parent.childrenNumber == 1)
                {
                    d.currentNode.SetOldStyle();
                    d.currentNode = d.currentNode.parent;
                    d.currentNode.SetCurrentNodeStyle();
                }
            </direct>
            <reverse>
                if(d.currentNode.childrenNumber == 1)
                {
                    d.currentNode.SetOldStyle();
                    int returned = d.currentNode.
                                   FindChild(d.deletingElement);
                    if(returned == 1)
                        d.currentNode = d.currentNode.child1;
                    if(returned == 2)
                        d.currentNode = d.currentNode.child2;
                    if(returned == 3)
                        d.currentNode = d.currentNode.child3;
                    d.currentNode.SetCurrentNodeStyle();
                }
            </reverse>
        </step>
    </then>
    <else>
        <step
            id="deletegotoparent1"
            description="to parent"
            comment-ru="Сверяем значения наименьших элементов в поддеревьях.
                        Если был удален первый элемент в некотором поддереве,
                        то мы возвращаем новое наименьшее значение поднимаясь
                        вверх по дереву
                        до тех пор пока поддерево из которого мы вернулись
                        является первым.
                        Как только это стало не так,
                        мы для данного элемента меняем наименьшее значение
                        того поддерева из
                        которого мы вернулись на новое."
            comment-en="Compare least values of subtrees."
            level="1"
            >
            <draw>
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
            </draw>
            <direct>
                d.currentNode.SetOldStyle();
                d.currentNode = d.currentNode.parent;
                d.currentNode.SetCurrentNodeStyle();
                d.myStack.pushInteger(d.currentNode.low1);
                d.myStack.pushInteger(d.currentNode.low2);
                d.myStack.pushInteger(d.currentNode.low3);
                d.myStack.pushInteger(d.currentNode.low4);
                if(d.currentNode.childrenNumber == 1)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                }
                if(d.currentNode.childrenNumber == 2)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
                }
                if(d.currentNode.childrenNumber == 3)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
```

```
                                d.currentNode.low3 = d.currentNode.child3.low1;
                            }
                        </direct>
                        <reverse>
                            d.currentNode.SetOldStyle();
                            d.currentNode.low4 = d.myStack.popInteger();
                            d.currentNode.low3 = d.myStack.popInteger();
                            d.currentNode.low2 = d.myStack.popInteger();
                            d.currentNode.low1 = d.myStack.popInteger();
                            int returned = d.currentNode.FindChild(d.deletingElement);
                            if(returned == 1)
                                d.currentNode = d.currentNode.child1;
                            if(returned == 2)
                                d.currentNode = d.currentNode.child2;
                            if(returned == 3)
                                d.currentNode = d.currentNode.child3;
                            d.currentNode.SetCurrentNodeStyle();
                        </reverse>
                    </step>
                </else>
            </if>
        </else>
    </if>

    </auto>

</algorithm>
```

Tree23-Configuration.xml (конфигурация)

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

<configuration>

    <!-- Параметры панели управления -->

    <property description="Высота области комментария" param="comment-height"
        value="80"
        />

    <message description="Надпись &quot;Элемент&quot;" param="label-element"
        message-ru="Элемент"
        message-en="Element"
        />

    <button description="Параметры кнопки добавления" param="button-add"
        caption-ru="Добавить"
        caption-en="Insert"
        hint-ru="Добавить элемент"
        hint-en="Add element"
        />

    <button description="Параметры кнопки удаления" param="button-remove"
        caption-ru="Удалить "
        caption-en="Delete"
        hint-ru="Удалить элемент"
        hint-en="Remove element"
        />

    <button description="Параметры кнопки поиска" param="button-find"
        caption-ru="Найти"
        caption-en="Find"
        hint-ru="Найти элемент"
        hint-en="Find element"
        />

    <message description="Параметры поля ввода элемента" param="text-field-element-hint"
        message-ru="Ввод номера элемента"
        message-en="Element number input"
        />
```

```
<property description="Параметры поля ввода элемента" param="text-field-element-rows"
    value="3"
    />

<message description="Ошибка ввода - некоректный номер элемента" param="not-number-error"
    message-ru="Номер элемента должен быть целым числом от {0} до {1}"
    message-en="Element number should be integer between {0} and {1}"
    />

<message description="Ошибка ввода - предел количества элементов"
    param="max-point-number-error"
    message-ru="Достигнуто максимальное число элементов.
                Число элементов должно быть целым числом от {0} до {1}"
    message-en="There are maximum number of elements in the tree.
                Number of elements should be integer between {0} and {1}"
    />


<!-- Стили -->

<styleset description="Styles"
    param="styles"
    >
    <style
        description="Стиль линий"
        text-color="000000"
        text-align="0.5"
        border-color="000000"
        border-status="true"
        fill-color="cccccc"
        fill-status="true"
        aspect-status="false"
        padding="0.3"
        >
        <font
            face="Serif"
            size="20"
            style="plain"
            />
    </style>
    <style
        description="Стиль листа"
        fill-color="aaaacc"
        />
    <style
        description="Стиль внутреннего узла"
        fill-color="aaccaa"
        />
    <style
        description="Стиль текущего узла"
        fill-color="ff885b"
        />
</styleset>

<property description="Стиль линий" param="line-style"
    value="0"
    />

<property description="Стиль листа" param="leaf-default-style"
    value="2"
    />

<property description="Стиль внутреннего узла" param="interior-default-style"
    value="1"
    />

<property description="Стиль текущего узла" param="current-node-style"
    value="3"
    />

<property description="Цвет подсвеченной кнопки" param="mark-out-button-color-r"
    value="255"/>
<property description="Цвет подсвеченной кнопки" param="mark-out-button-color-g"
```

```
    value="255"/>
<property description="Цвет подсвеченной кнопки" param="mark-out-button-color-b"
    value="200"/>

<!-- Параметры дерева -->

<property description="Параметры листа" param="leaf-width"
    value="23"
    />

<property description="Параметры листа" param="leaf-height"
    value="23"
    />


<property description="Параметры внутреннего элемента" param="interior-width"
    value="24"
    />

<property description="Параметры внутреннего элемента" param="interior-height"
    value="24"
    />

<property description="Минимальный элемент" param="min-element"
    value="1"
    />

<property description="Максимальный элемент" param="max-element"
    value="99"
    />

<property description="Минимальный номер элемента " param="min-element-number"
    value="0"
    />

<property description="Максимальный номер элемента" param="max-element-number"
    value="20"
    />

<message description="Пример" param="example-of-tree"
  message-ru="((((2)(4)(6))((8)(10))((12)(14)))(((16)(18)(20))((22)(23)(24))((26)(28))))"
  message-en="((((2)(4)(6))((8)(10))((12)(14)))(((16)(18)(20))((22)(23)(24))((26)(28))))"
    />

<!-- Параметры сохранения/загрузки -->

<property
    description="Параметры кнопки и диалога сохранения-загрузки" param="save-load-mode"
    value="true"
    />

<property description="Параметры кнопки и диалога сохранения-загрузки"
        param="save-load-width"
    value="400"
    />

<property description="Параметры кнопки и диалога сохранения-загрузки"
    param="save-load-height"
    value="200"
    />

<message description="Название алгоритма" param="find"
    message-ru="поиск"
    message-en="find"
    />

<message description="Название алгоритма" param="insert"
    message-ru="добавление"
    message-en="insert"
    />

<message description="Название алгоритма" param="delete"
    message-ru="удаление"
    message-en="delete"
```

```
    />

<message description="Ошибка ввода при загрузке" param="load-info-0"
    message-ru="У узла 4 сына."
    message-en="Node has 4 children."
    />

<message description="Ошибка ввода при загрузке" param="load-info-1"
    message-ru="Нет упорядоченности элементов."
    message-en="There is not order on elements."
    />

<message description="Ошибка ввода при загрузке" param="load-info-2"
    message-ru="Не все листья находятся на одном уровне."
    message-en="There are not all leaves in one level."
    />

<message description="Ошибка ввода при загрузке" param="load-info-3"
    message-ru="Присутствует недопустимый символ вместо ( или ), или всего 1 сын."
    message-en="Bab simbol (Should be ( or ) ), or only 1 son."
    />

<message description="Ошибка ввода при загрузке" param="load-info-4"
    message-ru="Дерево должно начинаться со (."
    message-en="Tree should begins from (."
    />

<message description="Ошибка ввода при загрузке" param="load-info-5"
    message-ru="За описанием дерева следует лишний текст."
    message-en="There is bad text in the end."
    />

<message description="Ошибка ввода при загрузке" param="load-info-6"
    message-ru="Допустимые значения алгоритма: {0},{1} и {2}."
    message-en="Algerithm name should be {0},{1} or {2}."
    />

<message description="Ошибка ввода при загрузке" param="load-info-7"
    message-ru="Пропущен элемент."
    message-en="There is not element."
    />

<message description="Ошибка ввода при загрузке" param="load-info-8"
    message-ru="Пропущен шаг."
    message-en="There is not step."
    />

<message description="Ошибка ввода при загрузке" param="load-info-9"
    message-ru="Число элементов должно быть от {0} до {1}."
    message-en="Elements number should be between {0} and {1}."
    />

<message description="Ошибка ввода при загрузке" param="load-info-10"
    message-ru="Элементы должны быть от {0} до {1}."
    message-en="Elements should be between {0} and {1}."
    />

<button description="Параметры кнопки и диалога сохранения-загрузки" param="save-load"
    caption-ru="Сохранить/Загрузить"
    caption-en="Save/Load"
    hint-ru="Сохранить или загрузить входные данные"
    hint-en="Save or load input data"
    />

<property description="Параметры кнопки и диалога сохранения-загрузки"
    param="SaveLoadDialog-CommentPane-height"
    value="20"
    />

<message description="Комментарий" param="comment-alg"
    message-ru="Название алгоритма"
    message-en="Algorithm name"
    />
```

```
    <message description="Комментарий" param="comment-step"
        message-ru="Номер шага"
        message-en="Step"
        />

    <message description="Комментарий" param="comment-element"
        message-ru="Элемент"
        message-en="Element"
        />

    <message description="Комментарий" param="comment-tree"
        message-ru="Дерево"
        message-en="Tree"
        />

</configuration>
```

Приложение 3. Сгенерированный код автомата

Tree23.java

```
package ru.ifmo.vizi.Tree23;

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

public final class Tree23 extends BaseAutomataWithListener {
    /**
      * Модель.
      */
    public final Data d = new Data();

    /**
      * Конструктор для языка
      */
    public Tree23(Locale locale) {
        super("ru.ifmo.vizi.Tree23.Comments", locale);
        init(new Main(), d);
    }

    /**
      * Данные.
      */
    public final class Data {
        /**
          * Applet.
          */
        public Tree23Applet visualizer;

        /**
          * 2-3 Tree.
          */
        public Tree tree;

        /**
          * Current node.
          */
        public Node currentNode;

        /**
          * Stack.
          */
        public AutoStack myStack = new AutoStack();

        /**
          * Insert.
          */
        public boolean insert = false;

        /**
          * Inserting element.
          */
        public int insertingElement;

        /**
          * End of insert auto.
          */
        public boolean insertEnd = false;

        /**
          * Delete.
          */
        public boolean delete = false;
```

```
            /**
              * Deleting element.
              */
            public int deletingElement;

            /**
              * End of delete auto.
              */
            public boolean deleteEnd = false;

            /**
              * Find.
              */
            public boolean find = false;

            /**
              * Serching element.
              */
            public int serchingElement;

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

        /**
          * main auto.
          */
        private final class Main implements Automata {
            /**
              * Начальное состояние автомата.
              */
            private final int START_STATE = 0;

            /**
              * Конечное состояние автомата.
              */
            private final int END_STATE = 11;

            /**
              * Описания состояний.
              */
            private final String[] descriptions = new String[]{"Начальное состояние", "is
insert", "is insert (окончание)", "insert auto (автомат)", "is delete", "is delete (окончание)",
"delete auto (автомат)", "is find", "is find (окончание)", "find auto (автомат)", "no command",
"Конечное состояние"};

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

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

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

            /**
              * Переход в конечное состояние.
              */
            public void toEnd() {
                state = END_STATE;
                child = null;
            }

            /**
```

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

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

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

/**
  * Сделать шаг в перед.
  */
public void stepForward(int level) {
    do {
        step++;
        // Переход в следующее состояние
        switch (state) {
            case START_STATE: { // Начальное состояние
                state = 1; // is insert
                break;
            }
            case 1: { // is insert
                if (d.insert) {
                    state = 3; // insert auto (автомат)
                } else {
                    state = 4; // is delete
                }
                break;
            }
            case 2: { // is insert (окончание)
                state = END_STATE;
                break;
            }
            case 3: { // insert auto (автомат)
                if (child.isAtEnd()) {
                    child = null;
                    stack.pushBoolean(true);
                    state = 2; // is insert (окончание)
                }
                break;
            }
            case 4: { // is delete
                if (d.delete) {
                    state = 6; // delete auto (автомат)
                } else {
                    state = 7; // is find
                }
                break;
            }
            case 5: { // is delete (окончание)
                stack.pushBoolean(false);
                state = 2; // is insert (окончание)
                break;
            }
            case 6: { // delete auto (автомат)
                if (child.isAtEnd()) {
                    child = null;
                    stack.pushBoolean(true);
                    state = 5; // is delete (окончание)
                }
                break;
            }
            case 7: { // is find
```

```
            if (d.find) {
                state = 9; // find auto (автомат)
            } else {
                state = 10; // no command
            }
            break;
        }
        case 8: { // is find (окончание)
            stack.pushBoolean(false);
            state = 5; // is delete (окончание)
            break;
        }
        case 9: { // find auto (автомат)
            if (child.isAtEnd()) {
                child = null;
                stack.pushBoolean(true);
                state = 8; // is find (окончание)
            }
            break;
        }
        case 10: { // no command
            stack.pushBoolean(false);
            state = 8; // is find (окончание)
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // is insert
            break;
        }
        case 2: { // is insert (окончание)
            break;
        }
        case 3: { // insert auto (автомат)
            if (child == null) {
                child = new insert();
                child.toStart();
            }
            child.stepForward(level);
            step--;
            break;
        }
        case 4: { // is delete
            break;
        }
        case 5: { // is delete (окончание)
            break;
        }
        case 6: { // delete auto (автомат)
            if (child == null) {
                child = new delete();
                child.toStart();
            }
            child.stepForward(level);
            step--;
            break;
        }
        case 7: { // is find
            break;
        }
        case 8: { // is find (окончание)
            break;
        }
        case 9: { // find auto (автомат)
            if (child == null) {
                child = new find();
                child.toStart();
            }
            child.stepForward(level);
            step--;
            break;
        }
```

```
            case 10: { // no command
                break;
            }
        }
    } while (!isInteresting(level));
}

/**
  * Сделать шаг в назад.
  */
public void stepBackward(int level) {
    do {
        // Обращение действия в текущем состоянии
        switch (state) {
            case 1: { // is insert
                break;
            }
            case 2: { // is insert (окончание)
                break;
            }
            case 3: { // insert auto (автомат)
                if (child == null) {
                    child = new insert();
                    child.toEnd();
                }
                child.stepBackward(level);
                step++;
                break;
            }
            case 4: { // is delete
                break;
            }
            case 5: { // is delete (окончание)
                break;
            }
            case 6: { // delete auto (автомат)
                if (child == null) {
                    child = new delete();
                    child.toEnd();
                }
                child.stepBackward(level);
                step++;
                break;
            }
            case 7: { // is find
                break;
            }
            case 8: { // is find (окончание)
                break;
            }
            case 9: { // find auto (автомат)
                if (child == null) {
                    child = new find();
                    child.toEnd();
                }
                child.stepBackward(level);
                step++;
                break;
            }
            case 10: { // no command
                break;
            }
        }

        // Переход в предыдущее состояние
        switch (state) {
            case 1: { // is insert
                state = START_STATE;
                break;
            }
            case 2: { // is insert (окончание)
                if (stack.popBoolean()) {
                    state = 3; // insert auto (автомат)
                } else {
```

```
                    state = 5; // is delete (окончание)
                }
                break;
            }
            case 3: { // insert auto (автомат)
                if (child.isAtStart()) {
                    child = null;
                    state = 1; // is insert
                }
                break;
            }
            case 4: { // is delete
                state = 1; // is insert
                break;
            }
            case 5: { // is delete (окончание)
                if (stack.popBoolean()) {
                    state = 6; // delete auto (автомат)
                } else {
                    state = 8; // is find (окончание)
                }
                break;
            }
            case 6: { // delete auto (автомат)
                if (child.isAtStart()) {
                    child = null;
                    state = 4; // is delete
                }
                break;
            }
            case 7: { // is find
                state = 4; // is delete
                break;
            }
            case 8: { // is find (окончание)
                if (stack.popBoolean()) {
                    state = 9; // find auto (автомат)
                } else {
                    state = 10; // no command
                }
                break;
            }
            case 9: { // find auto (автомат)
                if (child.isAtStart()) {
                    child = null;
                    state = 7; // is find
                }
                break;
            }
            case 10: { // no command
                state = 7; // is find
                break;
            }
            case END_STATE: { // Начальное состояние
                state = 2; // is insert (окончание)
                break;
            }
        }

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

/**
  * Интересно ли текущее состояние.
  */
public boolean isInteresting(int level) {
    // Интересность
    switch (state) {
        case START_STATE: // Начальное состояние
            return true;
        case 1: // is insert
            return level <= -1;
        case 2: // is insert (окончание)
```

```
            return level <= -1;
        case 3: // insert auto (автомат)
            return (child != null) && !child.isAtEnd();
        case 4: // is delete
            return level <= -1;
        case 5: // is delete (окончание)
            return level <= -1;
        case 6: // delete auto (автомат)
            return (child != null) && !child.isAtEnd();
        case 7: // is find
            return level <= -1;
        case 8: // is find (окончание)
            return level <= -1;
        case 9: // find auto (автомат)
            return (child != null) && !child.isAtEnd();
        case 10: // no command
            return level <= 1;
        case END_STATE: // Конечное состояние
            return true;
    }

    throw new RuntimeException("isInterest");
}

/**
  * Комментарий к текущему состоянию
  */
public String getComment() {
    String comment = "";
    Object[] args = null;
    // Выбор комментария
    switch (state) {
        case START_STATE: { // Начальное состояние
            comment = Tree23.this.getComment("Main.START_STATE");
            break;
        }
        case 3: { // insert auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 6: { // delete auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 9: { // find auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 10: { // no command
            comment = Tree23.this.getComment("Main.nocommand");
            break;
        }
    }

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

/**
  * Выполняет действия по отрисовке состояния
  */
public void drawState() {
    switch (state) {
        case 3: { // insert auto (автомат)
            child.drawState();
            break;
        }
        case 6: { // delete auto (автомат)
            child.drawState();
            break;
        }
        case 9: { // find auto (автомат)
```

```
                        child.drawState();
                        break;
                    }
                    case 10: { // no command
                        break;
                    }
                }
            }

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

        /**
          * find auto.
          */
        private final class find implements Automata {
            /**
              * Начальное состояние автомата.
              */
            private final int START_STATE = 0;

            /**
              * Конечное состояние автомата.
              */
            private final int END_STATE = 8;

            /**
              * Описания состояний.
              */
            private final String[] descriptions = new String[]{"Начальное состояние", "is tree
empty", "is tree empty (окончание)", "tree is empty", "init", "begins from root", "find auto
(автомат)", "returns setings", "Конечное состояние"};

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

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

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

            /**
              * Переход в конечное состояние.
              */
            public void toEnd() {
                state = END_STATE;
                child = null;
            }

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

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

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

/**
  * Сделать шаг в перед.
  */
public void stepForward(int level) {
    do {
        step++;
        // Переход в следующее состояние
        switch (state) {
            case START_STATE: { // Начальное состояние
                state = 1; // is tree empty
                break;
            }
            case 1: { // is tree empty
                if (d.tree.childrenNumber == 0) {
                    state = 3; // tree is empty
                } else {
                    state = 4; // init
                }
                break;
            }
            case 2: { // is tree empty (окончание)
                state = 7; // returns setings
                break;
            }
            case 3: { // tree is empty
                stack.pushBoolean(true);
                state = 2; // is tree empty (окончание)
                break;
            }
            case 4: { // init
                state = 5; // begins from root
                break;
            }
            case 5: { // begins from root
                state = 6; // find auto (автомат)
                break;
            }
            case 6: { // find auto (автомат)
                if (child.isAtEnd()) {
                    child = null;
                    stack.pushBoolean(false);
                    state = 2; // is tree empty (окончание)
                }
                break;
            }
            case 7: { // returns setings
                state = END_STATE;
                break;
            }
        }

        // Действие в текущем состоянии
        switch (state) {
            case 1: { // is tree empty
                break;
            }
            case 2: { // is tree empty (окончание)
                break;
            }
```

```
            case 3: { // tree is empty
                break;
            }
            case 4: { // init
                d.currentNode = d.tree.rootNode1;
                d.currentNode.SetCurrentNodeStyle();
                break;
            }
            case 5: { // begins from root
                break;
            }
            case 6: { // find auto (автомат)
                if (child == null) {
                    child = new findinsubtree();
                    child.toStart();
                }
                child.stepForward(level);
                step--;
                break;
            }
            case 7: { // returns setings
                d.currentNode.SetOldStyle();
                break;
            }
        }
    } while (!isInteresting(level));
}

/**
  * Сделать шаг в назад.
  */
public void stepBackward(int level) {
    do {
        // Обращение действия в текущем состоянии
        switch (state) {
            case 1: { // is tree empty
                break;
            }
            case 2: { // is tree empty (окончание)
                break;
            }
            case 3: { // tree is empty
                break;
            }
            case 4: { // init
                d.currentNode.SetOldStyle();
                break;
            }
            case 5: { // begins from root
                break;
            }
            case 6: { // find auto (автомат)
                if (child == null) {
                    child = new findinsubtree();
                    child.toEnd();
                }
                child.stepBackward(level);
                step++;
                break;
            }
            case 7: { // returns setings
                d.currentNode.SetCurrentNodeStyle();
                break;
            }
        }

        // Переход в предыдущее состояние
        switch (state) {
            case 1: { // is tree empty
                state = START_STATE;
                break;
            }
            case 2: { // is tree empty (окончание)
                if (stack.popBoolean()) {
```

```
                    state = 3; // tree is empty
                } else {
                    state = 6; // find auto (автомат)
                }
                break;
            }
            case 3: { // tree is empty
                state = 1; // is tree empty
                break;
            }
            case 4: { // init
                state = 1; // is tree empty
                break;
            }
            case 5: { // begins from root
                state = 4; // init
                break;
            }
            case 6: { // find auto (автомат)
                if (child.isAtStart()) {
                    child = null;
                    state = 5; // begins from root
                }
                break;
            }
            case 7: { // returns setings
                state = 2; // is tree empty (окончание)
                break;
            }
            case END_STATE: { // Начальное состояние
                state = 7; // returns setings
                break;
            }
        }

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

/**
  * Интересно ли текущее состояние.
  */
public boolean isInteresting(int level) {
    // Интересность
    switch (state) {
        case START_STATE: // Начальное состояние
            return true;
        case 1: // is tree empty
            return level <= -1;
        case 2: // is tree empty (окончание)
            return level <= -1;
        case 3: // tree is empty
            return level <= 1;
        case 4: // init
            return level <= -1;
        case 5: // begins from root
            return level <= 1;
        case 6: // find auto (автомат)
            return (child != null) && !child.isAtEnd();
        case 7: // returns setings
            return level <= -1;
        case END_STATE: // Конечное состояние
            return true;
    }

    throw new RuntimeException("isInterest");
}

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

```
        // Выбор комментария
        switch (state) {
            case 3: { // tree is empty
                comment = Tree23.this.getComment("find.findtreeempty");
                break;
            }
            case 5: { // begins from root
                comment = Tree23.this.getComment("find.findbeginsfromroot");
                args = new Object[]{new Integer(d.serchingElement)};
                break;
            }
            case 6: { // find auto (автомат)
                comment = child.getComment();
                args = new Object[0];
                break;
            }
        }

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

    /**
      * Выполняет действия по отрисовке состояния
      */
    public void drawState() {
        switch (state) {
            case 3: { // tree is empty
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
                break;
            }
            case 4: { // init
                break;
            }
            case 5: { // begins from root
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
                break;
            }
            case 6: { // find auto (автомат)
                child.drawState();
                break;
            }
            case 7: { // returns setings
                break;
            }
        }
    }

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

/**
  * find auto.
  */
private final class findinsubtree implements Automata {
    /**
      * Начальное состояние автомата.
      */
    private final int START_STATE = 0;

    /**
      * Конечное состояние автомата.
      */
    private final int END_STATE = 26;
```

```
                /**
                  * Описания состояний.
                  */
                private final String[] descriptions = new String[]{"Начальное состояние", "is leaf",
"is leaf (окончание)", "yes it is", "is it serching element", "is it serching element (окончание)",
"yes it is", "no it is not", "is serching element in first subtree", "is serching element in first
subtree (окончание)", "lt low2", "init", "find auto (автомат)", "is children number 2", "is children
number 2 (окончание)", "gt low2", "init", "find auto (автомат)", "is serching element in second
subtree", "is serching element in second subtree (окончание)", "lt low3", "init", "find auto
(автомат)", "gt low3", "init", "find auto (автомат)", "Конечное состояние"};

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

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

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

                /**
                  * Переход в конечное состояние.
                  */
                public void toEnd() {
                    state = END_STATE;
                    child = null;
                }

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

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

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

                /**
                  * Сделать шаг в перед.
                  */
                public void stepForward(int level) {
                    do {
                        step++;
                        // Переход в следующее состояние
                        switch (state) {
                            case START_STATE: { // Начальное состояние
                                state = 1; // is leaf
                                break;
                            }
                            case 1: { // is leaf
                                if (d.currentNode.isLeaf) {
                                    state = 3; // yes it is
```

```
    } else {
        state = 8; // is serching element in first subtree
    }
    break;
}
case 2: { // is leaf (окончание)
    state = END_STATE;
    break;
}
case 3: { // yes it is
    state = 4; // is it serching element
    break;
}
case 4: { // is it serching element
    if (d.currentNode.element == d.serchingElement) {
        state = 6; // yes it is
    } else {
        state = 7; // no it is not
    }
    break;
}
case 5: { // is it serching element (окончание)
    stack.pushBoolean(true);
    state = 2; // is leaf (окончание)
    break;
}
case 6: { // yes it is
    stack.pushBoolean(true);
    state = 5; // is it serching element (окончание)
    break;
}
case 7: { // no it is not
    stack.pushBoolean(false);
    state = 5; // is it serching element (окончание)
    break;
}
case 8: { // is serching element in first subtree
    if (d.currentNode.low2 > d.serchingElement) {
        state = 10; // lt low2
    } else {
        state = 13; // is children number 2
    }
    break;
}
case 9: { // is serching element in first subtree (окончание)
    stack.pushBoolean(false);
    state = 2; // is leaf (окончание)
    break;
}
case 10: { // lt low2
    state = 11; // init
    break;
}
case 11: { // init
    state = 12; // find auto (автомат)
    break;
}
case 12: { // find auto (автомат)
    if (child.isAtEnd()) {
        child = null;
        stack.pushBoolean(true);
        state = 9; // is serching element in first subtree (окончание)
    }
    break;
}
case 13: { // is children number 2
    if (d.currentNode.childrenNumber == 2) {
        state = 15; // gt low2
    } else {
        state = 18; // is serching element in second subtree
    }
    break;
}
case 14: { // is children number 2 (окончание)
```

```
            stack.pushBoolean(false);
            state = 9; // is serching element in first subtree (окончание)
            break;
        }
        case 15: { // gt low2
            state = 16; // init
            break;
        }
        case 16: { // init
            state = 17; // find auto (автомат)
            break;
        }
        case 17: { // find auto (автомат)
            if (child.isAtEnd()) {
                child = null;
                stack.pushBoolean(true);
                state = 14; // is children number 2 (окончание)
            }
            break;
        }
        case 18: { // is serching element in second subtree
            if (d.currentNode.low3 > d.serchingElement) {
                state = 20; // lt low3
            } else {
                state = 23; // gt low3
            }
            break;
        }
        case 19: { // is serching element in second subtree (окончание)
            stack.pushBoolean(false);
            state = 14; // is children number 2 (окончание)
            break;
        }
        case 20: { // lt low3
            state = 21; // init
            break;
        }
        case 21: { // init
            state = 22; // find auto (автомат)
            break;
        }
        case 22: { // find auto (автомат)
            if (child.isAtEnd()) {
                child = null;
                stack.pushBoolean(true);
                state = 19; // is serching element in second subtree (окончание)
            }
            break;
        }
        case 23: { // gt low3
            state = 24; // init
            break;
        }
        case 24: { // init
            state = 25; // find auto (автомат)
            break;
        }
        case 25: { // find auto (автомат)
            if (child.isAtEnd()) {
                child = null;
                stack.pushBoolean(false);
                state = 19; // is serching element in second subtree (окончание)
            }
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // is leaf
            break;
        }
        case 2: { // is leaf (окончание)
            break;
```

```
}
case 3: { // yes it is
    break;
}
case 4: { // is it serching element
    break;
}
case 5: { // is it serching element (окончание)
    break;
}
case 6: { // yes it is
    break;
}
case 7: { // no it is not
    break;
}
case 8: { // is serching element in first subtree
    break;
}
case 9: { // is serching element in first subtree (окончание)
    break;
}
case 10: { // lt low2
    break;
}
case 11: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.child1;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 12: { // find auto (автомат)
    if (child == null) {
        child = new findinsubtree();
        child.toStart();
    }
    child.stepForward(level);
    step--;
    break;
}
case 13: { // is children number 2
    break;
}
case 14: { // is children number 2 (окончание)
    break;
}
case 15: { // gt low2
    break;
}
case 16: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.child2;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 17: { // find auto (автомат)
    if (child == null) {
        child = new findinsubtree();
        child.toStart();
    }
    child.stepForward(level);
    step--;
    break;
}
case 18: { // is serching element in second subtree
    break;
}
case 19: { // is serching element in second subtree (окончание)
    break;
}
case 20: { // lt low3
    break;
}
case 21: { // init
```

```
                d.currentNode.SetOldStyle();
                d.currentNode = d.currentNode.child2;
                d.currentNode.SetCurrentNodeStyle();
                break;
            }
            case 22: { // find auto (автомат)
                if (child == null) {
                    child = new findinsubtree();
                    child.toStart();
                }
                child.stepForward(level);
                step--;
                break;
            }
            case 23: { // gt low3
                break;
            }
            case 24: { // init
                d.currentNode.SetOldStyle();
                d.currentNode = d.currentNode.child3;
                d.currentNode.SetCurrentNodeStyle();
                break;
            }
            case 25: { // find auto (автомат)
                if (child == null) {
                    child = new findinsubtree();
                    child.toStart();
                }
                child.stepForward(level);
                step--;
                break;
            }
        }
    } while (!isInteresting(level));
}

/**
  * Сделать шаг в назад.
  */
public void stepBackward(int level) {
    do {
        // Обращение действия в текущем состоянии
        switch (state) {
            case 1: { // is leaf
                break;
            }
            case 2: { // is leaf (окончание)
                break;
            }
            case 3: { // yes it is
                break;
            }
            case 4: { // is it serching element
                break;
            }
            case 5: { // is it serching element (окончание)
                break;
            }
            case 6: { // yes it is
                break;
            }
            case 7: { // no it is not
                break;
            }
            case 8: { // is serching element in first subtree
                break;
            }
            case 9: { // is serching element in first subtree (окончание)
                break;
            }
            case 10: { // lt low2
                break;
            }
            case 11: { // init
```

```
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 12: { // find auto (автомат)
    if (child == null) {
        child = new findinsubtree();
        child.toEnd();
    }
    child.stepBackward(level);
    step++;
    break;
}
case 13: { // is children number 2
    break;
}
case 14: { // is children number 2 (окончание)
    break;
}
case 15: { // gt low2
    break;
}
case 16: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 17: { // find auto (автомат)
    if (child == null) {
        child = new findinsubtree();
        child.toEnd();
    }
    child.stepBackward(level);
    step++;
    break;
}
case 18: { // is serching element in second subtree
    break;
}
case 19: { // is serching element in second subtree (окончание)
    break;
}
case 20: { // lt low3
    break;
}
case 21: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 22: { // find auto (автомат)
    if (child == null) {
        child = new findinsubtree();
        child.toEnd();
    }
    child.stepBackward(level);
    step++;
    break;
}
case 23: { // gt low3
    break;
}
case 24: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 25: { // find auto (автомат)
    if (child == null) {
        child = new findinsubtree();
```

```
            child.toEnd();
        }
        child.stepBackward(level);
        step++;
        break;
    }
}

// Переход в предыдущее состояние
switch (state) {
    case 1: { // is leaf
        state = START_STATE;
        break;
    }
    case 2: { // is leaf (окончание)
        if (stack.popBoolean()) {
            state = 5; // is it serching element (окончание)
        } else {
            state = 9; // is serching element in first subtree (окончание)
        }
        break;
    }
    case 3: { // yes it is
        state = 1; // is leaf
        break;
    }
    case 4: { // is it serching element
        state = 3; // yes it is
        break;
    }
    case 5: { // is it serching element (окончание)
        if (stack.popBoolean()) {
            state = 6; // yes it is
        } else {
            state = 7; // no it is not
        }
        break;
    }
    case 6: { // yes it is
        state = 4; // is it serching element
        break;
    }
    case 7: { // no it is not
        state = 4; // is it serching element
        break;
    }
    case 8: { // is serching element in first subtree
        state = 1; // is leaf
        break;
    }
    case 9: { // is serching element in first subtree (окончание)
        if (stack.popBoolean()) {
            state = 12; // find auto (автомат)
        } else {
            state = 14; // is children number 2 (окончание)
        }
        break;
    }
    case 10: { // lt low2
        state = 8; // is serching element in first subtree
        break;
    }
    case 11: { // init
        state = 10; // lt low2
        break;
    }
    case 12: { // find auto (автомат)
        if (child.isAtStart()) {
            child = null;
            state = 11; // init
        }
        break;
    }
    case 13: { // is children number 2
```

```
                state = 8; // is serching element in first subtree
                break;
            }
            case 14: { // is children number 2 (окончание)
                if (stack.popBoolean()) {
                    state = 17; // find auto (автомат)
                } else {
                    state = 19; // is serching element in second subtree (окончание)
                }
                break;
            }
            case 15: { // gt low2
                state = 13; // is children number 2
                break;
            }
            case 16: { // init
                state = 15; // gt low2
                break;
            }
            case 17: { // find auto (автомат)
                if (child.isAtStart()) {
                    child = null;
                    state = 16; // init
                }
                break;
            }
            case 18: { // is serching element in second subtree
                state = 13; // is children number 2
                break;
            }
            case 19: { // is serching element in second subtree (окончание)
                if (stack.popBoolean()) {
                    state = 22; // find auto (автомат)
                } else {
                    state = 25; // find auto (автомат)
                }
                break;
            }
            case 20: { // lt low3
                state = 18; // is serching element in second subtree
                break;
            }
            case 21: { // init
                state = 20; // lt low3
                break;
            }
            case 22: { // find auto (автомат)
                if (child.isAtStart()) {
                    child = null;
                    state = 21; // init
                }
                break;
            }
            case 23: { // gt low3
                state = 18; // is serching element in second subtree
                break;
            }
            case 24: { // init
                state = 23; // gt low3
                break;
            }
            case 25: { // find auto (автомат)
                if (child.isAtStart()) {
                    child = null;
                    state = 24; // init
                }
                break;
            }
            case END_STATE: { // Начальное состояние
                state = 2; // is leaf (окончание)
                break;
            }
        }
```

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

/**
  * Интересно ли текущее состояние.
  */
public boolean isInteresting(int level) {
    // Интересность
    switch (state) {
        case START_STATE: // Начальное состояние
            return true;
        case 1: // is leaf
            return level <= -1;
        case 2: // is leaf (окончание)
            return level <= -1;
        case 3: // yes it is
            return level <= 1;
        case 4: // is it serching element
            return level <= -1;
        case 5: // is it serching element (окончание)
            return level <= -1;
        case 6: // yes it is
            return level <= 1;
        case 7: // no it is not
            return level <= 1;
        case 8: // is serching element in first subtree
            return level <= -1;
        case 9: // is serching element in first subtree (окончание)
            return level <= -1;
        case 10: // lt low2
            return level <= 1;
        case 11: // init
            return level <= -1;
        case 12: // find auto (автомат)
            return (child != null) && !child.isAtEnd();
        case 13: // is children number 2
            return level <= -1;
        case 14: // is children number 2 (окончание)
            return level <= -1;
        case 15: // gt low2
            return level <= 1;
        case 16: // init
            return level <= -1;
        case 17: // find auto (автомат)
            return (child != null) && !child.isAtEnd();
        case 18: // is serching element in second subtree
            return level <= -1;
        case 19: // is serching element in second subtree (окончание)
            return level <= -1;
        case 20: // lt low3
            return level <= 1;
        case 21: // init
            return level <= -1;
        case 22: // find auto (автомат)
            return (child != null) && !child.isAtEnd();
        case 23: // gt low3
            return level <= 1;
        case 24: // init
            return level <= 1;
        case 25: // find auto (автомат)
            return (child != null) && !child.isAtEnd();
        case END_STATE: // Конечное состояние
            return true;
    }

    throw new RuntimeException("isInterest");
}

/**
  * Комментарий к текущему состоянию
  */
public String getComment() {
    String comment = "";
```

```
    Object[] args = null;
    // Выбор комментария
    switch (state) {
        case 3: { // yes it is
            comment = Tree23.this.getComment("findinsubtree.findyesitisleaf");
            args = new Object[]{new Integer(d.serchingElement)};
            break;
        }
        case 6: { // yes it is
            comment =
        Tree23.this.getComment("findinsubtree.findyesitisserchingelement");
            args = new Object[]{new Integer(d.serchingElement)};
            break;
        }
        case 7: { // no it is not
            comment =
        Tree23.this.getComment("findinsubtree.findnoitisnotserchingelement");
            args = new Object[]{new Integer(d.serchingElement)};
            break;
        }
        case 10: { // lt low2
            comment = Tree23.this.getComment("findinsubtree.findltlow2");
            args = new Object[]{new Integer(d.currentNode.low2)};
            break;
        }
        case 12: { // find auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 15: { // gt low2
            comment = Tree23.this.getComment("findinsubtree.findgtlow2");
            args = new Object[]{new Integer(d.currentNode.low2)};
            break;
        }
        case 17: { // find auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 20: { // lt low3
            comment = Tree23.this.getComment("findinsubtree.findltlow3");
            args = new Object[]{new Integer(d.currentNode.low2),new
                                Integer(d.currentNode.low3)};
            break;
        }
        case 22: { // find auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 23: { // gt low3
            comment = Tree23.this.getComment("findinsubtree.findgtlow3");
            args = new Object[]{new Integer(d.currentNode.low3)};
            break;
        }
        case 25: { // find auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
    }

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

/**
  * Выполняет действия по отрисовке состояния
  */
public void drawState() {
    switch (state) {
        case 3: { // yes it is
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
```

```
            break;
        }
        case 6: { // yes it is
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 7: { // no it is not
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 10: { // lt low2
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 11: { // init
            break;
        }
        case 12: { // find auto (автомат)
            child.drawState();
            break;
        }
        case 15: { // gt low2
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 16: { // init
            break;
        }
        case 17: { // find auto (автомат)
            child.drawState();
            break;
        }
        case 20: { // lt low3
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 21: { // init
            break;
        }
        case 22: { // find auto (автомат)
            child.drawState();
            break;
        }
        case 23: { // gt low3
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 24: { // init
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 25: { // find auto (автомат)
            child.drawState();
            break;
        }
    }
}

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

```
        }
    }

    /**
      * insert auto.
      */
    private final class insert implements Automata {
        /**
          * Начальное состояние автомата.
          */
        private final int START_STATE = 0;

        /**
          * Конечное состояние автомата.
          */
        private final int END_STATE = 29;

        /**
          * Описания состояний.
          */
        private final String[] descriptions = new String[]{"Начальное состояние", "is tree
empty", "is tree empty (окончание)", "tree is empty", "is root leaf", "is root leaf (окончание)",
"is this element in tree", "is this element in tree (окончание)", "this element is in tree", "root
is leaf", "to new node", "returns setings", "init", "begins from root", "insert auto (автомат)", "is
4 children", "is 4 children (окончание)", "insert", "to new node", "returns setings", "is not 4
children", "is not 4 children (окончание)", "end", "retuns settings", "is root", "is root
(окончание)", "to parent", "retuns settings", "end", "Конечное состояние"};

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

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

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

        /**
          * Переход в конечное состояние.
          */
        public void toEnd() {
            state = END_STATE;
            child = null;
        }

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

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

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

```
/**
  * Сделать шаг в перед.
  */
public void stepForward(int level) {
    do {
        step++;
        // Переход в следующее состояние
        switch (state) {
            case START_STATE: { // Начальное состояние
                state = 1; // is tree empty
                break;
            }
            case 1: { // is tree empty
                if (d.tree.childrenNumber == 0) {
                    state = 3; // tree is empty
                } else {
                    state = 4; // is root leaf
                }
                break;
            }
            case 2: { // is tree empty (окончание)
                state = 15; // is 4 children
                break;
            }
            case 3: { // tree is empty
                stack.pushBoolean(true);
                state = 2; // is tree empty (окончание)
                break;
            }
            case 4: { // is root leaf
                if (d.tree.rootNode1.isLeaf) {
                    state = 6; // is this element in tree
                } else {
                    state = 12; // init
                }
                break;
            }
            case 5: { // is root leaf (окончание)
                stack.pushBoolean(false);
                state = 2; // is tree empty (окончание)
                break;
            }
            case 6: { // is this element in tree
                if (d.tree.rootNode1.element == d.insertingElement) {
                    state = 8; // this element is in tree
                } else {
                    state = 9; // root is leaf
                }
                break;
            }
            case 7: { // is this element in tree (окончание)
                state = 11; // returns setings
                break;
            }
            case 8: { // this element is in tree
                stack.pushBoolean(true);
                state = 7; // is this element in tree (окончание)
                break;
            }
            case 9: { // root is leaf
                state = 10; // to new node
                break;
            }
            case 10: { // to new node
                stack.pushBoolean(false);
                state = 7; // is this element in tree (окончание)
                break;
            }
            case 11: { // returns setings
                stack.pushBoolean(true);
                state = 5; // is root leaf (окончание)
                break;
            }
```

```
case 12: { // init
    state = 13; // begins from root
    break;
}
case 13: { // begins from root
    state = 14; // insert auto (автомат)
    break;
}
case 14: { // insert auto (автомат)
    if (child.isAtEnd()) {
        child = null;
        stack.pushBoolean(false);
        state = 5; // is root leaf (окончание)
    }
    break;
}
case 15: { // is 4 children
    if (d.tree.rootNode1.childrenNumber == 4) {
        state = 17; // insert
    } else {
        state = 24; // is root
    }
    break;
}
case 16: { // is 4 children (окончание)
    state = 28; // end
    break;
}
case 17: { // insert
    state = 18; // to new node
    break;
}
case 18: { // to new node
    state = 19; // returns setings
    break;
}
case 19: { // returns setings
    state = 20; // is not 4 children
    break;
}
case 20: { // is not 4 children
    if (d.currentNode.parent.childrenNumber == 3) {
        state = 22; // end
    } else {
        stack.pushBoolean(false);
        state = 21; // is not 4 children (окончание)
    }
    break;
}
case 21: { // is not 4 children (окончание)
    stack.pushBoolean(true);
    state = 16; // is 4 children (окончание)
    break;
}
case 22: { // end
    state = 23; // retuns settings
    break;
}
case 23: { // retuns settings
    stack.pushBoolean(true);
    state = 21; // is not 4 children (окончание)
    break;
}
case 24: { // is root
    if (d.currentNode != null && !d.currentNode.isRoot) {
        state = 26; // to parent
    } else {
        stack.pushBoolean(false);
        state = 25; // is root (окончание)
    }
    break;
}
case 25: { // is root (окончание)
    state = 27; // retuns settings
```

```
            break;
        }
        case 26: { // to parent
            stack.pushBoolean(true);
            state = 25; // is root (окончание)
            break;
        }
        case 27: { // retuns settings
            stack.pushBoolean(false);
            state = 16; // is 4 children (окончание)
            break;
        }
        case 28: { // end
            state = END_STATE;
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // is tree empty
            break;
        }
        case 2: { // is tree empty (окончание)
            break;
        }
        case 3: { // tree is empty
            d.tree.NewChildLeaf(d.insertingElement);
            d.tree.rootNode1.SetCurrentNodeStyle();
            d.currentNode = d.tree.rootNode1;
            break;
        }
        case 4: { // is root leaf
            break;
        }
        case 5: { // is root leaf (окончание)
            break;
        }
        case 6: { // is this element in tree
            break;
        }
        case 7: { // is this element in tree (окончание)
            break;
        }
        case 8: { // this element is in tree
            d.tree.rootNode1.SetCurrentNodeStyle();
            break;
        }
        case 9: { // root is leaf
            int returned = d.tree.NewChildLeaf(d.insertingElement);
            if(returned == 1)
                d.tree.rootNode1.SetCurrentNodeStyle();
            if(returned == 2)
                d.tree.rootNode2.SetCurrentNodeStyle();
            break;
        }
        case 10: { // to new node
            d.tree.rootNode1.SetOldStyle();
            d.tree.rootNode2.SetOldStyle();
            d.tree.DeleteChild(2);
            d.tree.DeleteChild(1);
            d.tree.NewChildInterior(d.tree.rootNode1, d.tree.rootNode2);
            d.tree.rootNode1.SetCurrentNodeStyle();
            break;
        }
        case 11: { // returns setings
            d.tree.rootNode1.SetOldStyle();
            break;
        }
        case 12: { // init
            d.currentNode = d.tree.rootNode1;
            d.currentNode.SetCurrentNodeStyle();
            break;
        }
```

```
case 13: { // begins from root
    break;
}
case 14: { // insert auto (автомат)
    if (child == null) {
        child = new insertintosubtree();
        child.toStart();
    }
    child.stepForward(level);
    step--;
    break;
}
case 15: { // is 4 children
    break;
}
case 16: { // is 4 children (окончание)
    break;
}
case 17: { // insert
    d.currentNode.SetOldStyle();
    d.tree.rootNode1.DeleteChild(4);
    d.tree.rootNode1.DeleteChild(3);
    d.tree.NewChildInterior(d.tree.rootNode1.child3,
                            d.tree.rootNode1.child4);
    d.tree.rootNode1.SetCurrentNodeStyle();
    d.tree.rootNode2.SetCurrentNodeStyle();
    break;
}
case 18: { // to new node
    d.tree.rootNode1.SetOldStyle();
    d.tree.rootNode2.SetOldStyle();
    Node tempNode1 = d.tree.rootNode1;
    Node tempNode2 = d.tree.rootNode2;
    d.tree.DeleteChild(2);
    d.tree.DeleteChild(1);
    d.tree.NewChildInterior(tempNode1, tempNode2);
    d.tree.rootNode1.SetCurrentNodeStyle();
    break;
}
case 19: { // returns setings
    d.tree.rootNode1.SetOldStyle();
    break;
}
case 20: { // is not 4 children
    break;
}
case 21: { // is not 4 children (окончание)
    break;
}
case 22: { // end
    d.currentNode.parent.SetCurrentNodeStyle();
    break;
}
case 23: { // retuns settings
    d.currentNode.parent.SetOldStyle();
    break;
}
case 24: { // is root
    break;
}
case 25: { // is root (окончание)
    break;
}
case 26: { // to parent
    d.currentNode.SetOldStyle();
    d.myStack.pushInteger(d.currentNode.thisChildNumber);
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    d.myStack.pushInteger(d.currentNode.low1);
    d.myStack.pushInteger(d.currentNode.low2);
    d.myStack.pushInteger(d.currentNode.low3);
    d.myStack.pushInteger(d.currentNode.low4);
    if(d.currentNode.childrenNumber == 1)
    {
```

```
                    d.currentNode.low1 = d.currentNode.child1.low1;
                }
                if(d.currentNode.childrenNumber == 2)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
                }
                if(d.currentNode.childrenNumber == 3)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
                    d.currentNode.low3 = d.currentNode.child3.low1;
                }
                break;
            }
            case 27: { // retuns settings
                d.currentNode.SetOldStyle();
                break;
            }
            case 28: { // end
                d.insertEnd = true;
                break;
            }
        }
    } while (!isInteresting(level));
}

/**
  * Сделать шаг в назад.
  */
public void stepBackward(int level) {
    do {
        // Обращение действия в текущем состоянии
        switch (state) {
            case 1: { // is tree empty
                break;
            }
            case 2: { // is tree empty (окончание)
                break;
            }
            case 3: { // tree is empty
                d.tree.DeleteChild(1);
                break;
            }
            case 4: { // is root leaf
                break;
            }
            case 5: { // is root leaf (окончание)
                break;
            }
            case 6: { // is this element in tree
                break;
            }
            case 7: { // is this element in tree (окончание)
                break;
            }
            case 8: { // this element is in tree
                d.tree.rootNode1.SetOldStyle();
                break;
            }
            case 9: { // root is leaf
                d.tree.rootNode1.SetOldStyle();
                d.tree.rootNode2.SetOldStyle();
                d.tree.DeleteChild(d.tree.FindChild(d.insertingElement));
                break;
            }
            case 10: { // to new node
                d.tree.rootNode1.SetOldStyle();
                d.tree.DeleteChild(1);
                Node tn = d.tree.rootNode1;
                d.tree.AddChild(tn.child1);
                d.tree.AddChild(tn.child2);
                int returned = d.tree.FindChild(d.insertingElement);
                if(returned == 1)
```

```
            d.tree.rootNode1.SetCurrentNodeStyle();
        if(returned == 2)
            d.tree.rootNode2.SetCurrentNodeStyle();
        break;
    }
    case 11: { // returns setings
        d.tree.rootNode1.SetCurrentNodeStyle();
        break;
    }
    case 12: { // init
        d.currentNode.SetOldStyle();
        break;
    }
    case 13: { // begins from root
        break;
    }
    case 14: { // insert auto (автомат)
        if (child == null) {
            child = new insertintosubtree();
            child.toEnd();
        }
        child.stepBackward(level);
        step++;
        break;
    }
    case 15: { // is 4 children
        break;
    }
    case 16: { // is 4 children (окончание)
        break;
    }
    case 17: { // insert
        d.currentNode.SetCurrentNodeStyle();
        d.tree.rootNode1.SetOldStyle();
        d.tree.rootNode2.SetOldStyle();
        Node tempNode = d.tree.rootNode2;
        d.tree.DeleteChild(2);
        d.tree.rootNode1.AddChild(tempNode.child1);
        d.tree.rootNode1.AddChild(tempNode.child2);
        break;
    }
    case 18: { // to new node
        d.tree.rootNode1.SetOldStyle();
        d.tree.DeleteChild(1);
        Node tn = d.tree.rootNode1;
        d.tree.AddChild(tn.child1);
        d.tree.AddChild(tn.child2);
        break;
    }
    case 19: { // returns setings
        d.tree.rootNode1.SetCurrentNodeStyle();
        break;
    }
    case 20: { // is not 4 children
        break;
    }
    case 21: { // is not 4 children (окончание)
        break;
    }
    case 22: { // end
        d.currentNode.parent.SetOldStyle();
        break;
    }
    case 23: { // retuns settings
        d.currentNode.parent.SetCurrentNodeStyle();
        break;
    }
    case 24: { // is root
        break;
    }
    case 25: { // is root (окончание)
        break;
    }
    case 26: { // to parent
```

```
            d.currentNode.SetOldStyle();
            d.currentNode.low4 = d.myStack.popInteger();
            d.currentNode.low3 = d.myStack.popInteger();
            d.currentNode.low2 = d.myStack.popInteger();
            d.currentNode.low1 = d.myStack.popInteger();
            int returned = d.myStack.popInteger();
            if(returned == 1)
                d.currentNode = d.currentNode.child1;
            if(returned == 2)
                d.currentNode = d.currentNode.child2;
            if(returned == 3)
                d.currentNode = d.currentNode.child3;
            if(returned == 4)
                d.currentNode = d.currentNode.child4;
            d.currentNode.SetCurrentNodeStyle();
            break;
        }
        case 27: { // retuns settings
            d.currentNode.SetCurrentNodeStyle();
            break;
        }
        case 28: { // end
            d.insertEnd = false;
            break;
        }
    }

    // Переход в предыдущее состояние
    switch (state) {
        case 1: { // is tree empty
            state = START_STATE;
            break;
        }
        case 2: { // is tree empty (окончание)
            if (stack.popBoolean()) {
                state = 3; // tree is empty
            } else {
                state = 5; // is root leaf (окончание)
            }
            break;
        }
        case 3: { // tree is empty
            state = 1; // is tree empty
            break;
        }
        case 4: { // is root leaf
            state = 1; // is tree empty
            break;
        }
        case 5: { // is root leaf (окончание)
            if (stack.popBoolean()) {
                state = 11; // returns setings
            } else {
                state = 14; // insert auto (автомат)
            }
            break;
        }
        case 6: { // is this element in tree
            state = 4; // is root leaf
            break;
        }
        case 7: { // is this element in tree (окончание)
            if (stack.popBoolean()) {
                state = 8; // this element is in tree
            } else {
                state = 10; // to new node
            }
            break;
        }
        case 8: { // this element is in tree
            state = 6; // is this element in tree
            break;
        }
        case 9: { // root is leaf
```

```
    state = 6; // is this element in tree
    break;
}
case 10: { // to new node
    state = 9; // root is leaf
    break;
}
case 11: { // returns setings
    state = 7; // is this element in tree (окончание)
    break;
}
case 12: { // init
    state = 4; // is root leaf
    break;
}
case 13: { // begins from root
    state = 12; // init
    break;
}
case 14: { // insert auto (автомат)
    if (child.isAtStart()) {
        child = null;
        state = 13; // begins from root
    }
    break;
}
case 15: { // is 4 children
    state = 2; // is tree empty (окончание)
    break;
}
case 16: { // is 4 children (окончание)
    if (stack.popBoolean()) {
        state = 21; // is not 4 children (окончание)
    } else {
        state = 27; // retuns settings
    }
    break;
}
case 17: { // insert
    state = 15; // is 4 children
    break;
}
case 18: { // to new node
    state = 17; // insert
    break;
}
case 19: { // returns setings
    state = 18; // to new node
    break;
}
case 20: { // is not 4 children
    state = 19; // returns setings
    break;
}
case 21: { // is not 4 children (окончание)
    if (stack.popBoolean()) {
        state = 23; // retuns settings
    } else {
        state = 20; // is not 4 children
    }
    break;
}
case 22: { // end
    state = 20; // is not 4 children
    break;
}
case 23: { // retuns settings
    state = 22; // end
    break;
}
case 24: { // is root
    state = 15; // is 4 children
    break;
}
```

```
            case 25: { // is root (окончание)
                if (stack.popBoolean()) {
                    state = 26; // to parent
                } else {
                    state = 24; // is root
                }
                break;
            }
            case 26: { // to parent
                state = 24; // is root
                break;
            }
            case 27: { // retuns settings
                state = 25; // is root (окончание)
                break;
            }
            case 28: { // end
                state = 16; // is 4 children (окончание)
                break;
            }
            case END_STATE: { // Начальное состояние
                state = 28; // end
                break;
            }
        }

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

/**
  * Интересно ли текущее состояние.
  */
public boolean isInteresting(int level) {
    // Интересность
    switch (state) {
        case START_STATE: // Начальное состояние
            return true;
        case 1: // is tree empty
            return level <= -1;
        case 2: // is tree empty (окончание)
            return level <= -1;
        case 3: // tree is empty
            return level <= 1;
        case 4: // is root leaf
            return level <= -1;
        case 5: // is root leaf (окончание)
            return level <= -1;
        case 6: // is this element in tree
            return level <= -1;
        case 7: // is this element in tree (окончание)
            return level <= -1;
        case 8: // this element is in tree
            return level <= 1;
        case 9: // root is leaf
            return level <= 1;
        case 10: // to new node
            return level <= 1;
        case 11: // returns setings
            return level <= -1;
        case 12: // init
            return level <= -1;
        case 13: // begins from root
            return level <= 1;
        case 14: // insert auto (автомат)
            return (child != null) && !child.isAtEnd();
        case 15: // is 4 children
            return level <= -1;
        case 16: // is 4 children (окончание)
            return level <= -1;
        case 17: // insert
            return level <= 1;
        case 18: // to new node
            return level <= 1;
```

```
            case 19: // returns setings
                return level <= -1;
            case 20: // is not 4 children
                return level <= -1;
            case 21: // is not 4 children (окончание)
                return level <= -1;
            case 22: // end
                return level <= 1;
            case 23: // retuns settings
                return level <= -1;
            case 24: // is root
                return level <= -1;
            case 25: // is root (окончание)
                return level <= -1;
            case 26: // to parent
                return level <= -1;
            case 27: // retuns settings
                return level <= 1;
            case 28: // end
                return level <= -1;
            case END_STATE: // Конечное состояние
                return true;
        }

        throw new RuntimeException("isInterest");
    }

    /**
      * Комментарий к текущему состоянию
      */
    public String getComment() {
        String comment = "";
        Object[] args = null;
        // Выбор комментария
        switch (state) {
            case 3: { // tree is empty
                comment = Tree23.this.getComment("insert.inserttreeempty");
                break;
            }
            case 8: { // this element is in tree
                comment = Tree23.this.getComment("insert.insertthiselementisintree");
                break;
            }
            case 9: { // root is leaf
                comment = Tree23.this.getComment("insert.insertrootisleaf");
                break;
            }
            case 10: { // to new node
                comment = Tree23.this.getComment("insert.insertnewroot0");
                break;
            }
            case 13: { // begins from root
                comment = Tree23.this.getComment("insert.insertbeginsfromroot");
                args = new Object[]{new Integer(d.insertingElement)};
                break;
            }
            case 14: { // insert auto (автомат)
                comment = child.getComment();
                args = new Object[0];
                break;
            }
            case 17: { // insert
                comment = Tree23.this.getComment("insert.insertinsertelement0");
                break;
            }
            case 18: { // to new node
                comment = Tree23.this.getComment("insert.insertnewroot1");
                break;
            }
            case 22: { // end
                comment = Tree23.this.getComment("insert.insertend0");
                break;
            }
            case 26: { // to parent
```

```
            comment = Tree23.this.getComment("insert.insertgotoparent0");
            break;
        }
    }

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

/**
  * Выполняет действия по отрисовке состояния
  */
public void drawState() {
    switch (state) {
        case 3: { // tree is empty
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 8: { // this element is in tree
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 9: { // root is leaf
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 10: { // to new node
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 11: { // returns setings
            break;
        }
        case 12: { // init
            break;
        }
        case 13: { // begins from root
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 14: { // insert auto (автомат)
            child.drawState();
            break;
        }
        case 17: { // insert
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 18: { // to new node
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 19: { // returns setings
            break;
        }
        case 22: { // end
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 23: { // retuns settings
            break;
        }
        case 26: { // to parent
            break;
        }
        case 27: { // retuns settings
            d.visualizer.DrawTree();
```

```
                        d.visualizer.UpdateScreen();
                        break;
                    }
                    case 28: { // end
                        break;
                    }
                }
            }

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

        /**
          * insert auto.
          */
        private final class insertintosubtree implements Automata {
            /**
              * Начальное состояние автомата.
              */
            private final int START_STATE = 0;

            /**
              * Конечное состояние автомата.
              */
            private final int END_STATE = 42;

            /**
              * Описания состояний.
              */
            private final String[] descriptions = new String[]{"Начальное состояние", "is child
leaf", "is child leaf (окончание)", "yes it is", "is children number 2", "is children number 2
(окончание)", "is inserting element in tree", "is inserting element in tree (окончание)", "yes it
is", "insert", "is inserting element in tree", "is inserting element in tree (окончание)", "yes it
is", "insert", "insert", "is serching element in first subtree", "is serching element in first
subtree (окончание)", "lt low2", "init", "insert auto (автомат)", "is children number 2", "is
children number 2 (окончание)", "gt low2", "init", "insert auto (автомат)", "is serching element in
second subtree", "is serching element in second subtree (окончание)", "lt low3", "init", "insert
auto (автомат)", "gt low3", "init", "insert auto (автомат)", "is 4 children", "is 4 children
(окончание)", "to parent", "insert", "to new node", "is not 4 children", "is not 4 children
(окончание)", "to parent", "retuns settings", "Конечное состояние"};


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

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

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

            /**
              * Переход в конечное состояние.
              */
            public void toEnd() {
                state = END_STATE;
```

```
    child = null;
}

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

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

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

/**
  * Сделать шаг в перед.
  */
public void stepForward(int level) {
    do {
        step++;
        // Переход в следующее состояние
        switch (state) {
            case START_STATE: { // Начальное состояние
                state = 1; // is child leaf
                break;
            }
            case 1: { // is child leaf
                if (d.currentNode.child1.isLeaf) {
                    state = 3; // yes it is
                } else {
                    state = 15; // is serching element in first subtree
                }
                break;
            }
            case 2: { // is child leaf (окончание)
                state = END_STATE;
                break;
            }
            case 3: { // yes it is
                state = 4; // is children number 2
                break;
            }
            case 4: { // is children number 2
                if (d.currentNode.childrenNumber == 2) {
                    state = 6; // is inserting element in tree
                } else {
                    state = 10; // is inserting element in tree
                }
                break;
            }
            case 5: { // is children number 2 (окончание)
                stack.pushBoolean(true);
                state = 2; // is child leaf (окончание)
                break;
            }
            case 6: { // is inserting element in tree
                if (d.currentNode.child1.element == d.insertingElement ||
                            d.currentNode.child2.element == d.insertingElement) {
                    state = 8; // yes it is
                } else {
                    state = 9; // insert
                }
                break;
            }
```

```
case 7: { // is inserting element in tree (окончание)
    stack.pushBoolean(true);
    state = 5; // is children number 2 (окончание)
    break;
}
case 8: { // yes it is
    stack.pushBoolean(true);
    state = 7; // is inserting element in tree (окончание)
    break;
}
case 9: { // insert
    stack.pushBoolean(false);
    state = 7; // is inserting element in tree (окончание)
    break;
}
case 10: { // is inserting element in tree
    if (d.currentNode.child1.element == d.insertingElement ||
              d.currentNode.child2.element == d.insertingElement ||
              d.currentNode.child3.element == d.insertingElement) {
        state = 12; // yes it is
    } else {
        state = 13; // insert
    }
    break;
}
case 11: { // is inserting element in tree (окончание)
    stack.pushBoolean(false);
    state = 5; // is children number 2 (окончание)
    break;
}
case 12: { // yes it is
    stack.pushBoolean(true);
    state = 11; // is inserting element in tree (окончание)
    break;
}
case 13: { // insert
    state = 14; // insert
    break;
}
case 14: { // insert
    stack.pushBoolean(false);
    state = 11; // is inserting element in tree (окончание)
    break;
}
case 15: { // is serching element in first subtree
    if (d.currentNode.low2 > d.insertingElement) {
        state = 17; // lt low2
    } else {
        state = 20; // is children number 2
    }
    break;
}
case 16: { // is serching element in first subtree (окончание)
    state = 33; // is 4 children
    break;
}
case 17: { // lt low2
    state = 18; // init
    break;
}
case 18: { // init
    state = 19; // insert auto (автомат)
    break;
}
case 19: { // insert auto (автомат)
    if (child.isAtEnd()) {
        child = null;
        stack.pushBoolean(true);
        state = 16; // is serching element in first subtree (окончание)
    }
    break;
}
case 20: { // is children number 2
    if (d.currentNode.childrenNumber == 2) {
```

```
            state = 22; // gt low2
        } else {
            state = 25; // is serching element in second subtree
        }
        break;
    }
    case 21: { // is children number 2 (окончание)
        stack.pushBoolean(false);
        state = 16; // is serching element in first subtree (окончание)
        break;
    }
    case 22: { // gt low2
        state = 23; // init
        break;
    }
    case 23: { // init
        state = 24; // insert auto (автомат)
        break;
    }
    case 24: { // insert auto (автомат)
        if (child.isAtEnd()) {
            child = null;
            stack.pushBoolean(true);
            state = 21; // is children number 2 (окончание)
        }
        break;
    }
    case 25: { // is serching element in second subtree
        if (d.currentNode.low3 > d.insertingElement) {
            state = 27; // lt low3
        } else {
            state = 30; // gt low3
        }
        break;
    }
    case 26: { // is serching element in second subtree (окончание)
        stack.pushBoolean(false);
        state = 21; // is children number 2 (окончание)
        break;
    }
    case 27: { // lt low3
        state = 28; // init
        break;
    }
    case 28: { // init
        state = 29; // insert auto (автомат)
        break;
    }
    case 29: { // insert auto (автомат)
        if (child.isAtEnd()) {
            child = null;
            stack.pushBoolean(true);
            state = 26; // is serching element in second subtree (окончание)
        }
        break;
    }
    case 30: { // gt low3
        state = 31; // init
        break;
    }
    case 31: { // init
        state = 32; // insert auto (автомат)
        break;
    }
    case 32: { // insert auto (автомат)
        if (child.isAtEnd()) {
            child = null;
            stack.pushBoolean(false);
            state = 26; // is serching element in second subtree (окончание)
        }
        break;
    }
    case 33: { // is 4 children
        if (d.currentNode.parent.childrenNumber == 4) {
```

```
                state = 35; // to parent
            } else {
                state = 40; // to parent
            }
            break;
        }
        case 34: { // is 4 children (окончание)
            stack.pushBoolean(false);
            state = 2; // is child leaf (окончание)
            break;
        }
        case 35: { // to parent
            state = 36; // insert
            break;
        }
        case 36: { // insert
            state = 37; // to new node
            break;
        }
        case 37: { // to new node
            state = 38; // is not 4 children
            break;
        }
        case 38: { // is not 4 children
            if (d.currentNode.parent.childrenNumber != 4) {
                stack.pushBoolean(true);
                state = 39; // is not 4 children (окончание)
            } else {
                stack.pushBoolean(false);
                state = 39; // is not 4 children (окончание)
            }
            break;
        }
        case 39: { // is not 4 children (окончание)
            stack.pushBoolean(true);
            state = 34; // is 4 children (окончание)
            break;
        }
        case 40: { // to parent
            state = 41; // retuns settings
            break;
        }
        case 41: { // retuns settings
            stack.pushBoolean(false);
            state = 34; // is 4 children (окончание)
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // is child leaf
            break;
        }
        case 2: { // is child leaf (окончание)
            break;
        }
        case 3: { // yes it is
            break;
        }
        case 4: { // is children number 2
            break;
        }
        case 5: { // is children number 2 (окончание)
            break;
        }
        case 6: { // is inserting element in tree
            break;
        }
        case 7: { // is inserting element in tree (окончание)
            break;
        }
        case 8: { // yes it is
            break;
```

```
}
case 9: { // insert
    d.currentNode.SetOldStyle();
    int returned = d.currentNode.NewChildLeaf(d.insertingElement);
    if(returned == 1)
        d.currentNode = d.currentNode.child1;
    if(returned == 2)
        d.currentNode = d.currentNode.child2;
    if(returned == 3)
        d.currentNode = d.currentNode.child3;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 10: { // is inserting element in tree
    break;
}
case 11: { // is inserting element in tree (окончание)
    break;
}
case 12: { // yes it is
    break;
}
case 13: { // insert
    d.currentNode.SetOldStyle();
    int returned = d.currentNode.NewChildLeaf(d.insertingElement);
    if(returned == 1)
        d.currentNode = d.currentNode.child1;
    if(returned == 2)
        d.currentNode = d.currentNode.child2;
    if(returned == 3)
        d.currentNode = d.currentNode.child3;
    if(returned == 4)
        d.currentNode = d.currentNode.child4;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 14: { // insert
    break;
}
case 15: { // is serching element in first subtree
    break;
}
case 16: { // is serching element in first subtree (окончание)
    break;
}
case 17: { // lt low2
    break;
}
case 18: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.child1;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 19: { // insert auto (автомат)
    if (child == null) {
        child = new insertintosubtree();
        child.toStart();
    }
    child.stepForward(level);
    step--;
    break;
}
case 20: { // is children number 2
    break;
}
case 21: { // is children number 2 (окончание)
    break;
}
case 22: { // gt low2
    break;
}
case 23: { // init
    d.currentNode.SetOldStyle();
```

```
    d.currentNode = d.currentNode.child2;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 24: { // insert auto (автомат)
    if (child == null) {
        child = new insertintosubtree();
        child.toStart();
    }
    child.stepForward(level);
    step--;
    break;
}
case 25: { // is serching element in second subtree
    break;
}
case 26: { // is serching element in second subtree (окончание)
    break;
}
case 27: { // lt low3
    break;
}
case 28: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.child2;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 29: { // insert auto (автомат)
    if (child == null) {
        child = new insertintosubtree();
        child.toStart();
    }
    child.stepForward(level);
    step--;
    break;
}
case 30: { // gt low3
    break;
}
case 31: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.child3;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 32: { // insert auto (автомат)
    if (child == null) {
        child = new insertintosubtree();
        child.toStart();
    }
    child.stepForward(level);
    step--;
    break;
}
case 33: { // is 4 children
    break;
}
case 34: { // is 4 children (окончание)
    break;
}
case 35: { // to parent
    d.tree.SetDefaultStyles();
    d.myStack.pushInteger(d.currentNode.thisChildNumber);
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 36: { // insert
    d.currentNode.DeleteChild(4);
    d.currentNode.DeleteChild(3);
    int returned = d.currentNode.parent.
        NewChildInterior(d.currentNode.child3, d.currentNode.child4);
    if(returned == 2)
```

```
                    d.currentNode.parent.child2.SetCurrentNodeStyle();
                if(returned == 3)
                    d.currentNode.parent.child3.SetCurrentNodeStyle();
                if(returned == 4)
                    d.currentNode.parent.child4.SetCurrentNodeStyle();
                break;
            }
            case 37: { // to new node
                if(d.currentNode.thisChildNumber == 1)
                {
                    d.currentNode.SetOldStyle();
                    d.currentNode.parent.child2.SetOldStyle();
                }
                if(d.currentNode.thisChildNumber == 2)
                {
                    d.currentNode.SetOldStyle();
                    d.currentNode.parent.child3.SetOldStyle();
                }
                if(d.currentNode.thisChildNumber == 3)
                {
                    d.currentNode.SetOldStyle();
                    d.currentNode.parent.child4.SetOldStyle();
                }
                break;
            }
            case 38: { // is not 4 children
                break;
            }
            case 39: { // is not 4 children (окончание)
                break;
            }
            case 40: { // to parent
                d.currentNode.SetOldStyle();
                d.myStack.pushInteger(d.currentNode.thisChildNumber);
                d.currentNode = d.currentNode.parent;
                d.currentNode.SetCurrentNodeStyle();
                d.myStack.pushInteger(d.currentNode.low1);
                d.myStack.pushInteger(d.currentNode.low2);
                d.myStack.pushInteger(d.currentNode.low3);
                d.myStack.pushInteger(d.currentNode.low4);
                if(d.currentNode.childrenNumber == 1)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                }
                if(d.currentNode.childrenNumber == 2)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
                }
                if(d.currentNode.childrenNumber == 3)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
                    d.currentNode.low3 = d.currentNode.child3.low1;
                }
                break;
            }
            case 41: { // retuns settings
                d.currentNode.SetOldStyle();
                break;
            }
        }
    } while (!isInteresting(level));
}

/**
  * Сделать шаг в назад.
  */
public void stepBackward(int level) {
    do {
        // Обращение действия в текущем состоянии
        switch (state) {
            case 1: { // is child leaf
                break;
```

```
}
case 2: { // is child leaf (окончание)
    break;
}
case 3: { // yes it is
    break;
}
case 4: { // is children number 2
    break;
}
case 5: { // is children number 2 (окончание)
    break;
}
case 6: { // is inserting element in tree
    break;
}
case 7: { // is inserting element in tree (окончание)
    break;
}
case 8: { // yes it is
    break;
}
case 9: { // insert
    d.currentNode.SetOldStyle();
    d.currentNode.parent.DeleteChild(d.currentNode.thisChildNumber);
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 10: { // is inserting element in tree
    break;
}
case 11: { // is inserting element in tree (окончание)
    break;
}
case 12: { // yes it is
    break;
}
case 13: { // insert
    d.currentNode.SetOldStyle();
    d.currentNode.parent.DeleteChild(d.currentNode.thisChildNumber);
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 14: { // insert
    d.currentNode.parent.SetOldStyle();
    break;
}
case 15: { // is serching element in first subtree
    break;
}
case 16: { // is serching element in first subtree (окончание)
    break;
}
case 17: { // lt low2
    break;
}
case 18: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 19: { // insert auto (автомат)
    if (child == null) {
        child = new insertintosubtree();
        child.toEnd();
    }
    child.stepBackward(level);
    step++;
    break;
}
case 20: { // is children number 2
```

```
    break;
}
case 21: { // is children number 2 (окончание)
    break;
}
case 22: { // gt low2
    break;
}
case 23: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 24: { // insert auto (автомат)
    if (child == null) {
        child = new insertintosubtree();
        child.toEnd();
    }
    child.stepBackward(level);
    step++;
    break;
}
case 25: { // is serching element in second subtree
    break;
}
case 26: { // is serching element in second subtree (окончание)
    break;
}
case 27: { // lt low3
    break;
}
case 28: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 29: { // insert auto (автомат)
    if (child == null) {
        child = new insertintosubtree();
        child.toEnd();
    }
    child.stepBackward(level);
    step++;
    break;
}
case 30: { // gt low3
    break;
}
case 31: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 32: { // insert auto (автомат)
    if (child == null) {
        child = new insertintosubtree();
        child.toEnd();
    }
    child.stepBackward(level);
    step++;
    break;
}
case 33: { // is 4 children
    break;
}
case 34: { // is 4 children (окончание)
    break;
}
case 35: { // to parent
    d.currentNode.SetOldStyle();
    int returned = d.myStack.popInteger();
```

```
    if(returned == 1)
        d.currentNode = d.currentNode.child1;
    if(returned == 2)
        d.currentNode = d.currentNode.child2;
    if(returned == 3)
        d.currentNode = d.currentNode.child3;
    if(returned == 4)
        d.currentNode = d.currentNode.child4;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 36: { // insert
    Node tempNode;
    if(d.currentNode.thisChildNumber == 1)
    {
        tempNode = d.currentNode.parent.child2;
        d.currentNode.parent.DeleteChild(2);
        d.currentNode.AddChild(tempNode.child1);
        d.currentNode.AddChild(tempNode.child2);
    }
    if(d.currentNode.thisChildNumber == 2)
    {
        tempNode = d.currentNode.parent.child3;
        d.currentNode.parent.DeleteChild(3);
        d.currentNode.AddChild(tempNode.child1);
        d.currentNode.AddChild(tempNode.child2);
    }
    if(d.currentNode.thisChildNumber == 3)
    {
        tempNode = d.currentNode.parent.child4;
        d.currentNode.parent.DeleteChild(4);
        d.currentNode.AddChild(tempNode.child1);
        d.currentNode.AddChild(tempNode.child2);
    }
    if(d.currentNode.thisChildNumber == 4)
    {
        tempNode = d.currentNode.parent.child3;
        d.currentNode.parent.DeleteChild(3);
        d.currentNode.AddChild(tempNode.child1);
        d.currentNode.AddChild(tempNode.child2);
    }
    d.currentNode.SetOldStyle();
    int returned = d.currentNode.FindChild(d.insertingElement);
    if(returned == 1)
        d.currentNode.child1.SetCurrentNodeStyle();
    if(returned == 2)
        d.currentNode.child2.SetCurrentNodeStyle();
    if(returned == 3)
        d.currentNode.child3.SetCurrentNodeStyle();
    if(returned == 4)
        d.currentNode.child4.SetCurrentNodeStyle();

    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 37: { // to new node
    if(d.currentNode.thisChildNumber == 1)
    {
        d.currentNode.SetCurrentNodeStyle();
        d.currentNode.parent.child2.SetCurrentNodeStyle();
    }
    if(d.currentNode.thisChildNumber == 2)
    {
        d.currentNode.SetCurrentNodeStyle();
        d.currentNode.parent.child3.SetCurrentNodeStyle();
    }
    if(d.currentNode.thisChildNumber == 3)
    {
        d.currentNode.SetCurrentNodeStyle();
        d.currentNode.parent.child4.SetCurrentNodeStyle();
    }
    break;
}
case 38: { // is not 4 children
```

```
            break;
        }
        case 39: { // is not 4 children (окончание)
            break;
        }
        case 40: { // to parent
            d.currentNode.SetOldStyle();
            d.currentNode.low4 = d.myStack.popInteger();
            d.currentNode.low3 = d.myStack.popInteger();
            d.currentNode.low2 = d.myStack.popInteger();
            d.currentNode.low1 = d.myStack.popInteger();
            int returned = d.myStack.popInteger();
            if(returned == 1)
                d.currentNode = d.currentNode.child1;
            if(returned == 2)
                d.currentNode = d.currentNode.child2;
            if(returned == 3)
                d.currentNode = d.currentNode.child3;
            if(returned == 4)
                d.currentNode = d.currentNode.child4;
            d.currentNode.SetCurrentNodeStyle();
            break;
        }
        case 41: { // retuns settings
            d.currentNode.SetCurrentNodeStyle();
            break;
        }
    }

    // Переход в предыдущее состояние
    switch (state) {
        case 1: { // is child leaf
            state = START_STATE;
            break;
        }
        case 2: { // is child leaf (окончание)
            if (stack.popBoolean()) {
                state = 5; // is children number 2 (окончание)
            } else {
                state = 34; // is 4 children (окончание)
            }
            break;
        }
        case 3: { // yes it is
            state = 1; // is child leaf
            break;
        }
        case 4: { // is children number 2
            state = 3; // yes it is
            break;
        }
        case 5: { // is children number 2 (окончание)
            if (stack.popBoolean()) {
                state = 7; // is inserting element in tree (окончание)
            } else {
                state = 11; // is inserting element in tree (окончание)
            }
            break;
        }
        case 6: { // is inserting element in tree
            state = 4; // is children number 2
            break;
        }
        case 7: { // is inserting element in tree (окончание)
            if (stack.popBoolean()) {
                state = 8; // yes it is
            } else {
                state = 9; // insert
            }
            break;
        }
        case 8: { // yes it is
            state = 6; // is inserting element in tree
            break;
```

```
}
case 9: { // insert
    state = 6; // is inserting element in tree
    break;
}
case 10: { // is inserting element in tree
    state = 4; // is children number 2
    break;
}
case 11: { // is inserting element in tree (окончание)
    if (stack.popBoolean()) {
        state = 12; // yes it is
    } else {
        state = 14; // insert
    }
    break;
}
case 12: { // yes it is
    state = 10; // is inserting element in tree
    break;
}
case 13: { // insert
    state = 10; // is inserting element in tree
    break;
}
case 14: { // insert
    state = 13; // insert
    break;
}
case 15: { // is serching element in first subtree
    state = 1; // is child leaf
    break;
}
case 16: { // is serching element in first subtree (окончание)
    if (stack.popBoolean()) {
        state = 19; // insert auto (автомат)
    } else {
        state = 21; // is children number 2 (окончание)
    }
    break;
}
case 17: { // lt low2
    state = 15; // is serching element in first subtree
    break;
}
case 18: { // init
    state = 17; // lt low2
    break;
}
case 19: { // insert auto (автомат)
    if (child.isAtStart()) {
        child = null;
        state = 18; // init
    }
    break;
}
case 20: { // is children number 2
    state = 15; // is serching element in first subtree
    break;
}
case 21: { // is children number 2 (окончание)
    if (stack.popBoolean()) {
        state = 24; // insert auto (автомат)
    } else {
        state = 26; // is serching element in second subtree (окончание)
    }
    break;
}
case 22: { // gt low2
    state = 20; // is children number 2
    break;
}
case 23: { // init
    state = 22; // gt low2
```

```
    break;
}
case 24: { // insert auto (автомат)
    if (child.isAtStart()) {
        child = null;
        state = 23; // init
    }
    break;
}
case 25: { // is serching element in second subtree
    state = 20; // is children number 2
    break;
}
case 26: { // is serching element in second subtree (окончание)
    if (stack.popBoolean()) {
        state = 29; // insert auto (автомат)
    } else {
        state = 32; // insert auto (автомат)
    }
    break;
}
case 27: { // lt low3
    state = 25; // is serching element in second subtree
    break;
}
case 28: { // init
    state = 27; // lt low3
    break;
}
case 29: { // insert auto (автомат)
    if (child.isAtStart()) {
        child = null;
        state = 28; // init
    }
    break;
}
case 30: { // gt low3
    state = 25; // is serching element in second subtree
    break;
}
case 31: { // init
    state = 30; // gt low3
    break;
}
case 32: { // insert auto (автомат)
    if (child.isAtStart()) {
        child = null;
        state = 31; // init
    }
    break;
}
case 33: { // is 4 children
    state = 16; // is serching element in first subtree (окончание)
    break;
}
case 34: { // is 4 children (окончание)
    if (stack.popBoolean()) {
        state = 39; // is not 4 children (окончание)
    } else {
        state = 41; // retuns settings
    }
    break;
}
case 35: { // to parent
    state = 33; // is 4 children
    break;
}
case 36: { // insert
    state = 35; // to parent
    break;
}
case 37: { // to new node
    state = 36; // insert
    break;
```

```
            }
            case 38: { // is not 4 children
                state = 37; // to new node
                break;
            }
            case 39: { // is not 4 children (окончание)
                if (stack.popBoolean()) {
                    state = 38; // is not 4 children
                } else {
                    state = 38; // is not 4 children
                }
                break;
            }
            case 40: { // to parent
                state = 33; // is 4 children
                break;
            }
            case 41: { // retuns settings
                state = 40; // to parent
                break;
            }
            case END_STATE: { // Начальное состояние
                state = 2; // is child leaf (окончание)
                break;
            }
        }

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

/**
  * Интересно ли текущее состояние.
  */
public boolean isInteresting(int level) {
    // Интересность
    switch (state) {
        case START_STATE: // Начальное состояние
            return true;
        case 1: // is child leaf
            return level <= -1;
        case 2: // is child leaf (окончание)
            return level <= -1;
        case 3: // yes it is
            return level <= 1;
        case 4: // is children number 2
            return level <= -1;
        case 5: // is children number 2 (окончание)
            return level <= -1;
        case 6: // is inserting element in tree
            return level <= -1;
        case 7: // is inserting element in tree (окончание)
            return level <= -1;
        case 8: // yes it is
            return level <= 1;
        case 9: // insert
            return level <= 1;
        case 10: // is inserting element in tree
            return level <= -1;
        case 11: // is inserting element in tree (окончание)
            return level <= -1;
        case 12: // yes it is
            return level <= 1;
        case 13: // insert
            return level <= 1;
        case 14: // insert
            return level <= -1;
        case 15: // is serching element in first subtree
            return level <= -1;
        case 16: // is serching element in first subtree (окончание)
            return level <= -1;
        case 17: // lt low2
            return level <= 1;
        case 18: // init
```

```
            return level <= -1;
        case 19: // insert auto (автомат)
            return (child != null) && !child.isAtEnd();
        case 20: // is children number 2
            return level <= -1;
        case 21: // is children number 2 (окончание)
            return level <= -1;
        case 22: // gt low2
            return level <= 1;
        case 23: // init
            return level <= -1;
        case 24: // insert auto (автомат)
            return (child != null) && !child.isAtEnd();
        case 25: // is serching element in second subtree
            return level <= -1;
        case 26: // is serching element in second subtree (окончание)
            return level <= -1;
        case 27: // lt low3
            return level <= 1;
        case 28: // init
            return level <= -1;
        case 29: // insert auto (автомат)
            return (child != null) && !child.isAtEnd();
        case 30: // gt low3
            return level <= 1;
        case 31: // init
            return level <= -1;
        case 32: // insert auto (автомат)
            return (child != null) && !child.isAtEnd();
        case 33: // is 4 children
            return level <= -1;
        case 34: // is 4 children (окончание)
            return level <= -1;
        case 35: // to parent
            return level <= -1;
        case 36: // insert
            return level <= 1;
        case 37: // to new node
            return level <= -1;
        case 38: // is not 4 children
            return level <= -1;
        case 39: // is not 4 children (окончание)
            return level <= -1;
        case 40: // to parent
            return level <= -1;
        case 41: // retuns settings
            return level <= -1;
        case END_STATE: // Конечное состояние
            return true;
    }

    throw new RuntimeException("isInterest");
}

/**
  * Комментарий к текущему состоянию
  */
public String getComment() {
    String comment = "";
    Object[] args = null;
    // Выбор комментария
    switch (state) {
        case 3: { // yes it is
            comment = Tree23.this.getComment("insertintosubtree.insertyesitisleaf");
            args = new Object[]{new Integer(d.insertingElement)};
            break;
        }
        case 8: { // yes it is
            comment =
    Tree23.this.getComment("insertintosubtree.insertyesitisinsertingelementintree1");
            args = new Object[]{new Integer(d.insertingElement)};
            break;
        }
        case 9: { // insert
```

```
            comment =
   Tree23.this.getComment("insertintosubtree.insertinsertelement1");
            break;
        }
        case 12: { // yes it is
            comment =
   Tree23.this.getComment("insertintosubtree.insertyesitisinsertingelementintree2");
            args = new Object[]{new Integer(d.insertingElement)};
            break;
        }
        case 13: { // insert
            comment =
   Tree23.this.getComment("insertintosubtree.insertinsertelement2");
            break;
        }
        case 17: { // lt low2
            comment = Tree23.this.getComment("insertintosubtree.insertltlow2");
            args = new Object[]{new Integer(d.currentNode.low2)};
            break;
        }
        case 19: { // insert auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 22: { // gt low2
            comment = Tree23.this.getComment("insertintosubtree.insertgtlow2");
            args = new Object[]{new Integer(d.currentNode.low2)};
            break;
        }
        case 24: { // insert auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 27: { // lt low3
            comment = Tree23.this.getComment("insertintosubtree.insertltlow3");
            args = new Object[]{new Integer(d.currentNode.low2),
                   new Integer(d.currentNode.low3)};
            break;
        }
        case 29: { // insert auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 30: { // gt low3
            comment = Tree23.this.getComment("insertintosubtree.insertgtlow3");
            args = new Object[]{new Integer(d.currentNode.low3)};
            break;
        }
        case 32: { // insert auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 36: { // insert
            comment =
   Tree23.this.getComment("insertintosubtree.insertinsertelement3");
            break;
        }
        case 40: { // to parent
            comment = Tree23.this.getComment("insertintosubtree.insertgotoparent2");
            break;
        }
    }

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

/**
  * Выполняет действия по отрисовке состояния
  */
public void drawState() {
```

```
switch (state) {
    case 3: { // yes it is
        d.visualizer.DrawTree();
        d.visualizer.UpdateScreen();
        break;
    }
    case 8: { // yes it is
        d.visualizer.DrawTree();
        d.visualizer.UpdateScreen();
        break;
    }
    case 9: { // insert
        d.visualizer.DrawTree();
        d.visualizer.UpdateScreen();
        break;
    }
    case 12: { // yes it is
        d.visualizer.DrawTree();
        d.visualizer.UpdateScreen();
        break;
    }
    case 13: { // insert
        d.visualizer.DrawTree();
        d.visualizer.UpdateScreen();
        break;
    }
    case 14: { // insert
        break;
    }
    case 17: { // lt low2
        d.visualizer.DrawTree();
        d.visualizer.UpdateScreen();
        break;
    }
    case 18: { // init
        break;
    }
    case 19: { // insert auto (автомат)
        child.drawState();
        break;
    }
    case 22: { // gt low2
        d.visualizer.DrawTree();
        d.visualizer.UpdateScreen();
        break;
    }
    case 23: { // init
        break;
    }
    case 24: { // insert auto (автомат)
        child.drawState();
        break;
    }
    case 27: { // lt low3
        d.visualizer.DrawTree();
        d.visualizer.UpdateScreen();
        break;
    }
    case 28: { // init
        break;
    }
    case 29: { // insert auto (автомат)
        child.drawState();
        break;
    }
    case 30: { // gt low3
        d.visualizer.DrawTree();
        d.visualizer.UpdateScreen();
        break;
    }
    case 31: { // init
        break;
    }
    case 32: { // insert auto (автомат)
```

```
                        child.drawState();
                        break;
                    }
                    case 35: { // to parent
                        break;
                    }
                    case 36: { // insert
                        d.visualizer.DrawTree();
                        d.visualizer.UpdateScreen();
                        break;
                    }
                    case 37: { // to new node
                        break;
                    }
                    case 40: { // to parent
                        break;
                    }
                    case 41: { // retuns settings
                        break;
                    }
                }
            }

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

        /**
          * delete auto.
          */
        private final class delete implements Automata {
            /**
              * Начальное состояние автомата.
              */
            private final int START_STATE = 0;

            /**
              * Конечное состояние автомата.
              */
            private final int END_STATE = 16;

            /**
              * Описания состояний.
              */
            private final String[] descriptions = new String[]{"Начальное состояние", "is tree
empty", "is tree empty (окончание)", "tree is empty", "init", "begins from root", "delete auto
(автомат)", "is 1 children", "is 1 children (окончание)", "insert", "returns setings", "is root",
"is root (окончание)", "to parent", "returns setings", "end", "Конечное состояние"};

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

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

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

```
/**
  * Переход в конечное состояние.
  */
public void toEnd() {
    state = END_STATE;
    child = null;
}

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

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

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

/**
  * Сделать шаг в перед.
  */
public void stepForward(int level) {
    do {
        step++;
        // Переход в следующее состояние
        switch (state) {
            case START_STATE: { // Начальное состояние
                state = 1; // is tree empty
                break;
            }
            case 1: { // is tree empty
                if (d.tree.childrenNumber == 0) {
                    state = 3; // tree is empty
                } else {
                    state = 4; // init
                }
                break;
            }
            case 2: { // is tree empty (окончание)
                state = 7; // is 1 children
                break;
            }
            case 3: { // tree is empty
                stack.pushBoolean(true);
                state = 2; // is tree empty (окончание)
                break;
            }
            case 4: { // init
                state = 5; // begins from root
                break;
            }
            case 5: { // begins from root
                state = 6; // delete auto (автомат)
                break;
            }
            case 6: { // delete auto (автомат)
                if (child.isAtEnd()) {
                    child = null;
                    stack.pushBoolean(false);
                    state = 2; // is tree empty (окончание)
                }
                break;
```

```
    }
    case 7: { // is 1 children
        if (d.tree.rootNode1.childrenNumber == 1) {
            state = 9; // insert
        } else {
            state = 11; // is root
        }
        break;
    }
    case 8: { // is 1 children (окончание)
        state = 15; // end
        break;
    }
    case 9: { // insert
        state = 10; // returns setings
        break;
    }
    case 10: { // returns setings
        stack.pushBoolean(true);
        state = 8; // is 1 children (окончание)
        break;
    }
    case 11: { // is root
        if (d.currentNode != null  && !d.currentNode.isRoot) {
            state = 13; // to parent
        } else {
            stack.pushBoolean(false);
            state = 12; // is root (окончание)
        }
        break;
    }
    case 12: { // is root (окончание)
        state = 14; // returns setings
        break;
    }
    case 13: { // to parent
        stack.pushBoolean(true);
        state = 12; // is root (окончание)
        break;
    }
    case 14: { // returns setings
        stack.pushBoolean(false);
        state = 8; // is 1 children (окончание)
        break;
    }
    case 15: { // end
        state = END_STATE;
        break;
    }
}

// Действие в текущем состоянии
switch (state) {
    case 1: { // is tree empty
        break;
    }
    case 2: { // is tree empty (окончание)
        break;
    }
    case 3: { // tree is empty
        break;
    }
    case 4: { // init
        d.currentNode = d.tree.rootNode1;
        d.currentNode.SetCurrentNodeStyle();
        break;
    }
    case 5: { // begins from root
        break;
    }
    case 6: { // delete auto (автомат)
        if (child == null) {
            child = new deletefromsubtree();
            child.toStart();
```

```
                }
                child.stepForward(level);
                step--;
                break;
            }
            case 7: { // is 1 children
                break;
            }
            case 8: { // is 1 children (окончание)
                break;
            }
            case 9: { // insert
                d.tree.DeleteChild(1);
                d.tree.AddChild(d.tree.rootNode1.child1);
                d.tree.rootNode1.SetCurrentNodeStyle();
                break;
            }
            case 10: { // returns setings
                d.tree.rootNode1.SetOldStyle();
                break;
            }
            case 11: { // is root
                break;
            }
            case 12: { // is root (окончание)
                break;
            }
            case 13: { // to parent
                d.currentNode.SetOldStyle();
                d.currentNode = d.currentNode.parent;
                d.currentNode.SetCurrentNodeStyle();
                d.myStack.pushInteger(d.currentNode.low1);
                d.myStack.pushInteger(d.currentNode.low2);
                d.myStack.pushInteger(d.currentNode.low3);
                d.myStack.pushInteger(d.currentNode.low4);
                if(d.currentNode.childrenNumber == 1)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                }
                if(d.currentNode.childrenNumber == 2)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
                }
                if(d.currentNode.childrenNumber == 3)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
                    d.currentNode.low3 = d.currentNode.child3.low1;
                }
                break;
            }
            case 14: { // returns setings
                d.tree.rootNode1.SetOldStyle();
                break;
            }
            case 15: { // end
                d.deleteEnd = true;
                break;
            }
        }
    } while (!isInteresting(level));
}

/**
  * Сделать шаг в назад.
  */
public void stepBackward(int level) {
    do {
        // Обращение действия в текущем состоянии
        switch (state) {
            case 1: { // is tree empty
                break;
            }
```

```
    case 2: { // is tree empty (окончание)
        break;
    }
    case 3: { // tree is empty
        break;
    }
    case 4: { // init
        d.currentNode.SetOldStyle();
        break;
    }
    case 5: { // begins from root
        break;
    }
    case 6: { // delete auto (автомат)
        if (child == null) {
            child = new deletefromsubtree();
            child.toEnd();
        }
        child.stepBackward(level);
        step++;
        break;
    }
    case 7: { // is 1 children
        break;
    }
    case 8: { // is 1 children (окончание)
        break;
    }
    case 9: { // insert
        d.tree.rootNode1.SetOldStyle();
        d.tree.DeleteChild(1);
        d.tree.NewChildInterior(d.tree.rootNode1);
        break;
    }
    case 10: { // returns setings
        d.tree.rootNode1.SetCurrentNodeStyle();
        break;
    }
    case 11: { // is root
        break;
    }
    case 12: { // is root (окончание)
        break;
    }
    case 13: { // to parent
        d.currentNode.SetOldStyle();
        d.currentNode.low4 = d.myStack.popInteger();
        d.currentNode.low3 = d.myStack.popInteger();
        d.currentNode.low2 = d.myStack.popInteger();
        d.currentNode.low1 = d.myStack.popInteger();
        int returned = d.currentNode.FindChild(d.deletingElement);
        if(returned == 1)
            d.currentNode = d.currentNode.child1;
        if(returned == 2)
            d.currentNode = d.currentNode.child2;
        if(returned == 3)
            d.currentNode = d.currentNode.child3;
        d.currentNode.SetCurrentNodeStyle();
        break;
    }
    case 14: { // returns setings
        d.tree.rootNode1.SetCurrentNodeStyle();
        break;
    }
    case 15: { // end
        d.deleteEnd = false;
        break;
    }
}

// Переход в предыдущее состояние
switch (state) {
    case 1: { // is tree empty
        state = START_STATE;
```

```
    break;
}
case 2: { // is tree empty (окончание)
    if (stack.popBoolean()) {
        state = 3; // tree is empty
    } else {
        state = 6; // delete auto (автомат)
    }
    break;
}
case 3: { // tree is empty
    state = 1; // is tree empty
    break;
}
case 4: { // init
    state = 1; // is tree empty
    break;
}
case 5: { // begins from root
    state = 4; // init
    break;
}
case 6: { // delete auto (автомат)
    if (child.isAtStart()) {
        child = null;
        state = 5; // begins from root
    }
    break;
}
case 7: { // is 1 children
    state = 2; // is tree empty (окончание)
    break;
}
case 8: { // is 1 children (окончание)
    if (stack.popBoolean()) {
        state = 10; // returns setings
    } else {
        state = 14; // returns setings
    }
    break;
}
case 9: { // insert
    state = 7; // is 1 children
    break;
}
case 10: { // returns setings
    state = 9; // insert
    break;
}
case 11: { // is root
    state = 7; // is 1 children
    break;
}
case 12: { // is root (окончание)
    if (stack.popBoolean()) {
        state = 13; // to parent
    } else {
        state = 11; // is root
    }
    break;
}
case 13: { // to parent
    state = 11; // is root
    break;
}
case 14: { // returns setings
    state = 12; // is root (окончание)
    break;
}
case 15: { // end
    state = 8; // is 1 children (окончание)
    break;
}
case END_STATE: { // Начальное состояние
```

```
                state = 15; // end
                break;
            }
        }

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

/**
  * Интересно ли текущее состояние.
  */
public boolean isInteresting(int level) {
    // Интересность
    switch (state) {
        case START_STATE: // Начальное состояние
            return true;
        case 1: // is tree empty
            return level <= -1;
        case 2: // is tree empty (окончание)
            return level <= -1;
        case 3: // tree is empty
            return level <= 1;
        case 4: // init
            return level <= -1;
        case 5: // begins from root
            return level <= 1;
        case 6: // delete auto (автомат)
            return (child != null) && !child.isAtEnd();
        case 7: // is 1 children
            return level <= -1;
        case 8: // is 1 children (окончание)
            return level <= -1;
        case 9: // insert
            return level <= 1;
        case 10: // returns setings
            return level <= -1;
        case 11: // is root
            return level <= -1;
        case 12: // is root (окончание)
            return level <= -1;
        case 13: // to parent
            return level <= 1;
        case 14: // returns setings
            return level <= 1;
        case 15: // end
            return level <= -1;
        case END_STATE: // Конечное состояние
            return true;
    }

    throw new RuntimeException("isInterest");
}

/**
  * Комментарий к текущему состоянию
  */
public String getComment() {
    String comment = "";
    Object[] args = null;
    // Выбор комментария
    switch (state) {
        case 3: { // tree is empty
            comment = Tree23.this.getComment("delete.deletetreeempty");
            break;
        }
        case 5: { // begins from root
            comment = Tree23.this.getComment("delete.deletebeginsfromroot");
            args = new Object[]{new Integer(d.deletingElement)};
            break;
        }
        case 6: { // delete auto (автомат)
            comment = child.getComment();
            args = new Object[0];
```

```
                break;
            }
            case 9: { // insert
                comment = Tree23.this.getComment("delete.deleteinsertelement0");
                break;
            }
            case 13: { // to parent
                comment = Tree23.this.getComment("delete.deletegotoparent2");
                break;
            }
        }

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

    /**
      * Выполняет действия по отрисовке состояния
      */
    public void drawState() {
        switch (state) {
            case 3: { // tree is empty
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
                break;
            }
            case 4: { // init
                break;
            }
            case 5: { // begins from root
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
                break;
            }
            case 6: { // delete auto (автомат)
                child.drawState();
                break;
            }
            case 9: { // insert
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
                break;
            }
            case 10: { // returns setings
                break;
            }
            case 13: { // to parent
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
                break;
            }
            case 14: { // returns setings
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
                break;
            }
            case 15: { // end
                break;
            }
        }
    }

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

/**
```

```
     * delete auto.
     */
    private final class deletefromsubtree implements Automata {
        /**
          * Начальное состояние автомата.
          */
        private final int START_STATE = 0;

        /**
          * Конечное состояние автомата.
          */
        private final int END_STATE = 63;

        /**
          * Описания состояний.
          */
        private final String[] descriptions = new String[]{"Начальное состояние", "is child
leaf", "is child leaf (окончание)", "yes it is", "is children number 2", "is children number 2
(окончание)", "is deleting element in tree", "is deleting element in tree (окончание)", "no it is
not", "insert", "is inserting element in tree", "is inserting element in tree (окончание)", "yes it
is", "insert", "is deleting element in first subtree", "is deleting element in first subtree
(окончание)", "lt low2", "init", "delete auto (автомат)", "is children number 2", "is children
number 2 (окончание)", "gt low2", "init", "delete auto (автомат)", "is serching element in second
subtree", "is serching element in second subtree (окончание)", "lt low3", "init", "delete auto
(автомат)", "gt low3", "init", "delete auto (автомат)", "is 1 children", "is 1 children
(окончание)", "is it 1 child", "is it 1 child (окончание)", "is it 1 child", "is it 1 child
(окончание)", "insert", "insert", "is it 2 child", "is it 2 child (окончание)", "is it 2 child", "is
it 2 child (окончание)", "is it 1 child", "is it 1 child (окончание)", "insert", "insert", "is it 1
child", "is it 1 child (окончание)", "insert", "is it 1 child", "is it 1 child (окончание)",
"insert", "insert", "is it 3 child", "is it 3 child (окончание)", "is it 1 child", "is it 1 child
(окончание)", "insert", "insert", "to parent", "to parent", "Конечное состояние"};

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

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

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

        /**
          * Переход в конечное состояние.
          */
        public void toEnd() {
            state = END_STATE;
            child = null;
        }

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

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

        /**
          * Номер текущего шага.
```

```
  */
public int getStep() {
    return step;
}

/**
  * Сделать шаг в перед.
  */
public void stepForward(int level) {
    do {
        step++;
        // Переход в следующее состояние
        switch (state) {
            case START_STATE: { // Начальное состояние
                state = 1; // is child leaf
                break;
            }
            case 1: { // is child leaf
                if (d.currentNode.child1.isLeaf) {
                    state = 3; // yes it is
                } else {
                    state = 14; // is deleting element in first subtree
                }
                break;
            }
            case 2: { // is child leaf (окончание)
                state = END_STATE;
                break;
            }
            case 3: { // yes it is
                state = 4; // is children number 2
                break;
            }
            case 4: { // is children number 2
                if (d.currentNode.childrenNumber == 2) {
                    state = 6; // is deleting element in tree
                } else {
                    state = 10; // is inserting element in tree
                }
                break;
            }
            case 5: { // is children number 2 (окончание)
                stack.pushBoolean(true);
                state = 2; // is child leaf (окончание)
                break;
            }
            case 6: { // is deleting element in tree
                if (d.currentNode.child1.element != d.deletingElement &&
                           d.currentNode.child2.element != d.deletingElement) {
                    state = 8; // no it is not
                } else {
                    state = 9; // insert
                }
                break;
            }
            case 7: { // is deleting element in tree (окончание)
                stack.pushBoolean(true);
                state = 5; // is children number 2 (окончание)
                break;
            }
            case 8: { // no it is not
                stack.pushBoolean(true);
                state = 7; // is deleting element in tree (окончание)
                break;
            }
            case 9: { // insert
                stack.pushBoolean(false);
                state = 7; // is deleting element in tree (окончание)
                break;
            }
            case 10: { // is inserting element in tree
                if (d.currentNode.child1.element != d.deletingElement &&
                           d.currentNode.child2.element != d.deletingElement &&
                           d.currentNode.child3.element != d.deletingElement) {
```

```
            state = 12; // yes it is
        } else {
            state = 13; // insert
        }
        break;
    }
    case 11: { // is inserting element in tree (окончание)
        stack.pushBoolean(false);
        state = 5; // is children number 2 (окончание)
        break;
    }
    case 12: { // yes it is
        stack.pushBoolean(true);
        state = 11; // is inserting element in tree (окончание)
        break;
    }
    case 13: { // insert
        stack.pushBoolean(false);
        state = 11; // is inserting element in tree (окончание)
        break;
    }
    case 14: { // is deleting element in first subtree
        if (d.currentNode.low2 > d.deletingElement) {
            state = 16; // lt low2
        } else {
            state = 19; // is children number 2
        }
        break;
    }
    case 15: { // is deleting element in first subtree (окончание)
        state = 32; // is 1 children
        break;
    }
    case 16: { // lt low2
        state = 17; // init
        break;
    }
    case 17: { // init
        state = 18; // delete auto (автомат)
        break;
    }
    case 18: { // delete auto (автомат)
        if (child.isAtEnd()) {
            child = null;
            stack.pushBoolean(true);
            state = 15; // is deleting element in first subtree (окончание)
        }
        break;
    }
    case 19: { // is children number 2
        if (d.currentNode.childrenNumber == 2) {
            state = 21; // gt low2
        } else {
            state = 24; // is serching element in second subtree
        }
        break;
    }
    case 20: { // is children number 2 (окончание)
        stack.pushBoolean(false);
        state = 15; // is deleting element in first subtree (окончание)
        break;
    }
    case 21: { // gt low2
        state = 22; // init
        break;
    }
    case 22: { // init
        state = 23; // delete auto (автомат)
        break;
    }
    case 23: { // delete auto (автомат)
        if (child.isAtEnd()) {
            child = null;
            stack.pushBoolean(true);
```

```
            state = 20; // is children number 2 (окончание)
        }
        break;
    }
    case 24: { // is serching element in second subtree
        if (d.currentNode.low3 > d.deletingElement) {
            state = 26; // lt low3
        } else {
            state = 29; // gt low3
        }
        break;
    }
    case 25: { // is serching element in second subtree (окончание)
        stack.pushBoolean(false);
        state = 20; // is children number 2 (окончание)
        break;
    }
    case 26: { // lt low3
        state = 27; // init
        break;
    }
    case 27: { // init
        state = 28; // delete auto (автомат)
        break;
    }
    case 28: { // delete auto (автомат)
        if (child.isAtEnd()) {
            child = null;
            stack.pushBoolean(true);
            state = 25; // is serching element in second subtree (окончание)
        }
        break;
    }
    case 29: { // gt low3
        state = 30; // init
        break;
    }
    case 30: { // init
        state = 31; // delete auto (автомат)
        break;
    }
    case 31: { // delete auto (автомат)
        if (child.isAtEnd()) {
            child = null;
            stack.pushBoolean(false);
            state = 25; // is serching element in second subtree (окончание)
        }
        break;
    }
    case 32: { // is 1 children
        if (d.currentNode.childrenNumber == 1) {
            state = 34; // is it 1 child
        } else {
            state = 62; // to parent
        }
        break;
    }
    case 33: { // is 1 children (окончание)
        stack.pushBoolean(false);
        state = 2; // is child leaf (окончание)
        break;
    }
    case 34: { // is it 1 child
        if (d.currentNode.thisChildNumber == 1) {
            state = 36; // is it 1 child
        } else {
            stack.pushBoolean(false);
            state = 35; // is it 1 child (окончание)
        }
        break;
    }
    case 35: { // is it 1 child (окончание)
        state = 40; // is it 2 child
        break;
```

```
}
case 36: { // is it 1 child
    if (d.currentNode.parent.child2.childrenNumber == 3) {
        state = 38; // insert
    } else {
        state = 39; // insert
    }
    break;
}
case 37: { // is it 1 child (окончание)
    stack.pushBoolean(true);
    state = 35; // is it 1 child (окончание)
    break;
}
case 38: { // insert
    stack.pushBoolean(true);
    state = 37; // is it 1 child (окончание)
    break;
}
case 39: { // insert
    stack.pushBoolean(false);
    state = 37; // is it 1 child (окончание)
    break;
}
case 40: { // is it 2 child
    if (d.currentNode.thisChildNumber == 2) {
        state = 42; // is it 2 child
    } else {
        stack.pushBoolean(false);
        state = 41; // is it 2 child (окончание)
    }
    break;
}
case 41: { // is it 2 child (окончание)
    state = 55; // is it 3 child
    break;
}
case 42: { // is it 2 child
    if (d.currentNode.parent.childrenNumber == 2) {
        state = 44; // is it 1 child
    } else {
        state = 48; // is it 1 child
    }
    break;
}
case 43: { // is it 2 child (окончание)
    stack.pushBoolean(true);
    state = 41; // is it 2 child (окончание)
    break;
}
case 44: { // is it 1 child
    if (d.currentNode.parent.child1.childrenNumber == 3) {
        state = 46; // insert
    } else {
        state = 47; // insert
    }
    break;
}
case 45: { // is it 1 child (окончание)
    stack.pushBoolean(true);
    state = 43; // is it 2 child (окончание)
    break;
}
case 46: { // insert
    stack.pushBoolean(true);
    state = 45; // is it 1 child (окончание)
    break;
}
case 47: { // insert
    stack.pushBoolean(false);
    state = 45; // is it 1 child (окончание)
    break;
}
case 48: { // is it 1 child
```

```
    if (d.currentNode.parent.child1.childrenNumber == 3) {
        state = 50; // insert
    } else {
        state = 51; // is it 1 child
    }
    break;
}
case 49: { // is it 1 child (окончание)
    stack.pushBoolean(false);
    state = 43; // is it 2 child (окончание)
    break;
}
case 50: { // insert
    stack.pushBoolean(true);
    state = 49; // is it 1 child (окончание)
    break;
}
case 51: { // is it 1 child
    if (d.currentNode.parent.child3.childrenNumber == 3) {
        state = 53; // insert
    } else {
        state = 54; // insert
    }
    break;
}
case 52: { // is it 1 child (окончание)
    stack.pushBoolean(false);
    state = 49; // is it 1 child (окончание)
    break;
}
case 53: { // insert
    stack.pushBoolean(true);
    state = 52; // is it 1 child (окончание)
    break;
}
case 54: { // insert
    stack.pushBoolean(false);
    state = 52; // is it 1 child (окончание)
    break;
}
case 55: { // is it 3 child
    if (d.currentNode.thisChildNumber == 3) {
        state = 57; // is it 1 child
    } else {
        stack.pushBoolean(false);
        state = 56; // is it 3 child (окончание)
    }
    break;
}
case 56: { // is it 3 child (окончание)
    state = 61; // to parent
    break;
}
case 57: { // is it 1 child
    if (d.currentNode.parent.child2.childrenNumber == 3) {
        state = 59; // insert
    } else {
        state = 60; // insert
    }
    break;
}
case 58: { // is it 1 child (окончание)
    stack.pushBoolean(true);
    state = 56; // is it 3 child (окончание)
    break;
}
case 59: { // insert
    stack.pushBoolean(true);
    state = 58; // is it 1 child (окончание)
    break;
}
case 60: { // insert
    stack.pushBoolean(false);
    state = 58; // is it 1 child (окончание)
```

```
            break;
        }
        case 61: { // to parent
            stack.pushBoolean(true);
            state = 33; // is 1 children (окончание)
            break;
        }
        case 62: { // to parent
            stack.pushBoolean(false);
            state = 33; // is 1 children (окончание)
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // is child leaf
            break;
        }
        case 2: { // is child leaf (окончание)
            break;
        }
        case 3: { // yes it is
            break;
        }
        case 4: { // is children number 2
            break;
        }
        case 5: { // is children number 2 (окончание)
            break;
        }
        case 6: { // is deleting element in tree
            break;
        }
        case 7: { // is deleting element in tree (окончание)
            break;
        }
        case 8: { // no it is not
            break;
        }
        case 9: { // insert
            int returned = d.currentNode.FindChild(d.deletingElement);
            if(returned == 1)
                d.currentNode.DeleteChild(1);
            if(returned == 2)
                d.currentNode.DeleteChild(2);
            break;
        }
        case 10: { // is inserting element in tree
            break;
        }
        case 11: { // is inserting element in tree (окончание)
            break;
        }
        case 12: { // yes it is
            break;
        }
        case 13: { // insert
            int returned = d.currentNode.FindChild(d.deletingElement);
            if(returned == 1)
                d.currentNode.DeleteChild(1);
            if(returned == 2)
                d.currentNode.DeleteChild(2);
            if(returned == 3)
                d.currentNode.DeleteChild(3);
            d.currentNode.SetCurrentNodeStyle();
            break;
        }
        case 14: { // is deleting element in first subtree
            break;
        }
        case 15: { // is deleting element in first subtree (окончание)
            break;
        }
```

```
case 16: { // lt low2
    break;
}
case 17: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.child1;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 18: { // delete auto (автомат)
    if (child == null) {
        child = new deletefromsubtree();
        child.toStart();
    }
    child.stepForward(level);
    step--;
    break;
}
case 19: { // is children number 2
    break;
}
case 20: { // is children number 2 (окончание)
    break;
}
case 21: { // gt low2
    break;
}
case 22: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.child2;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 23: { // delete auto (автомат)
    if (child == null) {
        child = new deletefromsubtree();
        child.toStart();
    }
    child.stepForward(level);
    step--;
    break;
}
case 24: { // is serching element in second subtree
    break;
}
case 25: { // is serching element in second subtree (окончание)
    break;
}
case 26: { // lt low3
    break;
}
case 27: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.child2;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 28: { // delete auto (автомат)
    if (child == null) {
        child = new deletefromsubtree();
        child.toStart();
    }
    child.stepForward(level);
    step--;
    break;
}
case 29: { // gt low3
    break;
}
case 30: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.child3;
    d.currentNode.SetCurrentNodeStyle();
    break;
```

```
}
case 31: { // delete auto (автомат)
    if (child == null) {
        child = new deletefromsubtree();
        child.toStart();
    }
    child.stepForward(level);
    step--;
    break;
}
case 32: { // is 1 children
    break;
}
case 33: { // is 1 children (окончание)
    break;
}
case 34: { // is it 1 child
    break;
}
case 35: { // is it 1 child (окончание)
    break;
}
case 36: { // is it 1 child
    break;
}
case 37: { // is it 1 child (окончание)
    break;
}
case 38: { // insert
    Node tempNode = d.currentNode.parent.child2.child1;
    d.currentNode.parent.child2.DeleteChild(1);
    d.currentNode.AddChild(tempNode);
    break;
}
case 39: { // insert
    Node tempNode = d.currentNode.child1;
    d.currentNode.parent.DeleteChild(1);
    d.currentNode.parent.child1.AddChild(tempNode);
    d.currentNode = d.currentNode.parent.child1;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 40: { // is it 2 child
    break;
}
case 41: { // is it 2 child (окончание)
    break;
}
case 42: { // is it 2 child
    break;
}
case 43: { // is it 2 child (окончание)
    break;
}
case 44: { // is it 1 child
    break;
}
case 45: { // is it 1 child (окончание)
    break;
}
case 46: { // insert
    Node tempNode = d.currentNode.parent.child1.child3;
    d.currentNode.parent.child1.DeleteChild(3);
    d.currentNode.AddChild(tempNode);
    break;
}
case 47: { // insert
    Node tempNode = d.currentNode.child1;
    d.currentNode.parent.DeleteChild(2);
    d.currentNode.parent.child1.AddChild(tempNode);
    d.currentNode = d.currentNode.parent.child1;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
```

```
case 48: { // is it 1 child
    break;
}
case 49: { // is it 1 child (окончание)
    break;
}
case 50: { // insert
    Node tempNode = d.currentNode.parent.child1.child3;
    d.currentNode.parent.child1.DeleteChild(3);
    d.currentNode.AddChild(tempNode);
    break;
}
case 51: { // is it 1 child
    break;
}
case 52: { // is it 1 child (окончание)
    break;
}
case 53: { // insert
    Node tempNode = d.currentNode.parent.child3.child1;
    d.currentNode.parent.child3.DeleteChild(1);
    d.currentNode.AddChild(tempNode);
    break;
}
case 54: { // insert
    Node tempNode = d.currentNode.child1;
    d.currentNode.parent.DeleteChild(2);
    d.currentNode.parent.child1.AddChild(tempNode);
    d.currentNode = d.currentNode.parent.child1;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 55: { // is it 3 child
    break;
}
case 56: { // is it 3 child (окончание)
    break;
}
case 57: { // is it 1 child
    break;
}
case 58: { // is it 1 child (окончание)
    break;
}
case 59: { // insert
    Node tempNode = d.currentNode.parent.child2.child3;
    d.currentNode.parent.child2.DeleteChild(3);
    d.currentNode.AddChild(tempNode);
    break;
}
case 60: { // insert
    Node tempNode = d.currentNode.child1;
    d.currentNode.parent.DeleteChild(3);
    d.currentNode.parent.child2.AddChild(tempNode);
    d.currentNode = d.currentNode.parent.child2;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 61: { // to parent
    if(d.currentNode.parent.childrenNumber == 1)
    {
        d.currentNode.SetOldStyle();
        d.currentNode = d.currentNode.parent;
        d.currentNode.SetCurrentNodeStyle();
    }
    break;
}
case 62: { // to parent
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    d.myStack.pushInteger(d.currentNode.low1);
    d.myStack.pushInteger(d.currentNode.low2);
    d.myStack.pushInteger(d.currentNode.low3);
```

```
                d.myStack.pushInteger(d.currentNode.low4);
                if(d.currentNode.childrenNumber == 1)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                }
                if(d.currentNode.childrenNumber == 2)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
                }
                if(d.currentNode.childrenNumber == 3)
                {
                    d.currentNode.low1 = d.currentNode.child1.low1;
                    d.currentNode.low2 = d.currentNode.child2.low1;
                    d.currentNode.low3 = d.currentNode.child3.low1;
                }
                break;
            }
        }
    } while (!isInteresting(level));
}

/**
  * Сделать шаг в назад.
  */
public void stepBackward(int level) {
    do {
        // Обращение действия в текущем состоянии
        switch (state) {
            case 1: { // is child leaf
                break;
            }
            case 2: { // is child leaf (окончание)
                break;
            }
            case 3: { // yes it is
                break;
            }
            case 4: { // is children number 2
                break;
            }
            case 5: { // is children number 2 (окончание)
                break;
            }
            case 6: { // is deleting element in tree
                break;
            }
            case 7: { // is deleting element in tree (окончание)
                break;
            }
            case 8: { // no it is not
                break;
            }
            case 9: { // insert
                d.currentNode.NewChildLeaf(d.deletingElement);
                d.visualizer.DrawTree();
                d.visualizer.UpdateScreen();
                break;
            }
            case 10: { // is inserting element in tree
                break;
            }
            case 11: { // is inserting element in tree (окончание)
                break;
            }
            case 12: { // yes it is
                break;
            }
            case 13: { // insert
                d.currentNode.NewChildLeaf(d.deletingElement);
                break;
            }
            case 14: { // is deleting element in first subtree
                break;
```

```
}
case 15: { // is deleting element in first subtree (окончание)
    break;
}
case 16: { // lt low2
    break;
}
case 17: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 18: { // delete auto (автомат)
    if (child == null) {
        child = new deletefromsubtree();
        child.toEnd();
    }
    child.stepBackward(level);
    step++;
    break;
}
case 19: { // is children number 2
    break;
}
case 20: { // is children number 2 (окончание)
    break;
}
case 21: { // gt low2
    break;
}
case 22: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 23: { // delete auto (автомат)
    if (child == null) {
        child = new deletefromsubtree();
        child.toEnd();
    }
    child.stepBackward(level);
    step++;
    break;
}
case 24: { // is serching element in second subtree
    break;
}
case 25: { // is serching element in second subtree (окончание)
    break;
}
case 26: { // lt low3
    break;
}
case 27: { // init
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 28: { // delete auto (автомат)
    if (child == null) {
        child = new deletefromsubtree();
        child.toEnd();
    }
    child.stepBackward(level);
    step++;
    break;
}
case 29: { // gt low3
    break;
}
case 30: { // init
```

```
    d.currentNode.SetOldStyle();
    d.currentNode = d.currentNode.parent;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 31: { // delete auto (автомат)
    if (child == null) {
        child = new deletefromsubtree();
        child.toEnd();
    }
    child.stepBackward(level);
    step++;
    break;
}
case 32: { // is 1 children
    break;
}
case 33: { // is 1 children (окончание)
    break;
}
case 34: { // is it 1 child
    break;
}
case 35: { // is it 1 child (окончание)
    break;
}
case 36: { // is it 1 child
    break;
}
case 37: { // is it 1 child (окончание)
    break;
}
case 38: { // insert
    Node tempNode = d.currentNode.child2;
    d.currentNode.DeleteChild(2);
    d.currentNode.parent.child2.AddChild(tempNode);
    break;
}
case 39: { // insert
    d.currentNode.SetOldStyle();
    Node tempNode = d.currentNode.child1;
    d.currentNode.DeleteChild(1);
    d.currentNode.parent.NewChildInterior(tempNode);
    d.currentNode = d.currentNode.parent.child1;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 40: { // is it 2 child
    break;
}
case 41: { // is it 2 child (окончание)
    break;
}
case 42: { // is it 2 child
    break;
}
case 43: { // is it 2 child (окончание)
    break;
}
case 44: { // is it 1 child
    break;
}
case 45: { // is it 1 child (окончание)
    break;
}
case 46: { // insert
    Node tempNode = d.currentNode.child1;
    d.currentNode.DeleteChild(1);
    d.currentNode.parent.child1.AddChild(tempNode);
    break;
}
case 47: { // insert
    d.currentNode.SetOldStyle();
    Node tempNode = d.currentNode.child3;
```

```
    d.currentNode.parent.child1.DeleteChild(3);
    d.currentNode.parent.NewChildInterior(tempNode);
    d.currentNode = d.currentNode.parent.child2;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 48: { // is it 1 child
    break;
}
case 49: { // is it 1 child (окончание)
    break;
}
case 50: { // insert
    Node tempNode = d.currentNode.child1;
    d.currentNode.DeleteChild(1);
    d.currentNode.parent.child1.AddChild(tempNode);
    break;
}
case 51: { // is it 1 child
    break;
}
case 52: { // is it 1 child (окончание)
    break;
}
case 53: { // insert
    Node tempNode = d.currentNode.child2;
    d.currentNode.DeleteChild(2);
    d.currentNode.parent.child3.AddChild(tempNode);
    break;
}
case 54: { // insert
    d.currentNode.SetOldStyle();
    Node tempNode = d.currentNode.child3;
    d.currentNode.parent.child1.DeleteChild(3);
    d.currentNode.parent.NewChildInterior(tempNode);
    d.currentNode = d.currentNode.parent.child2;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 55: { // is it 3 child
    break;
}
case 56: { // is it 3 child (окончание)
    break;
}
case 57: { // is it 1 child
    break;
}
case 58: { // is it 1 child (окончание)
    break;
}
case 59: { // insert
    Node tempNode = d.currentNode.child1;
    d.currentNode.DeleteChild(1);
    d.currentNode.parent.child2.AddChild(tempNode);
    break;
}
case 60: { // insert
    d.currentNode.SetOldStyle();
    Node tempNode = d.currentNode.child3;
    d.currentNode.parent.child2.DeleteChild(3);
    d.currentNode.parent.NewChildInterior(tempNode);
    d.currentNode = d.currentNode.parent.child3;
    d.currentNode.SetCurrentNodeStyle();
    break;
}
case 61: { // to parent
    if(d.currentNode.childrenNumber == 1)
    {
        d.currentNode.SetOldStyle();
        int returned = d.currentNode.FindChild(d.deletingElement);
        if(returned == 1)
            d.currentNode = d.currentNode.child1;
        if(returned == 2)
```

```
                d.currentNode = d.currentNode.child2;
            if(returned == 3)
                d.currentNode = d.currentNode.child3;
            d.currentNode.SetCurrentNodeStyle();
        }
        break;
    }
    case 62: { // to parent
        d.currentNode.SetOldStyle();
        d.currentNode.low4 = d.myStack.popInteger();
        d.currentNode.low3 = d.myStack.popInteger();
        d.currentNode.low2 = d.myStack.popInteger();
        d.currentNode.low1 = d.myStack.popInteger();
        int returned = d.currentNode.FindChild(d.deletingElement);
        if(returned == 1)
            d.currentNode = d.currentNode.child1;
        if(returned == 2)
            d.currentNode = d.currentNode.child2;
        if(returned == 3)
            d.currentNode = d.currentNode.child3;
        d.currentNode.SetCurrentNodeStyle();
        break;
    }
}

// Переход в предыдущее состояние
switch (state) {
    case 1: { // is child leaf
        state = START_STATE;
        break;
    }
    case 2: { // is child leaf (окончание)
        if (stack.popBoolean()) {
            state = 5; // is children number 2 (окончание)
        } else {
            state = 33; // is 1 children (окончание)
        }
        break;
    }
    case 3: { // yes it is
        state = 1; // is child leaf
        break;
    }
    case 4: { // is children number 2
        state = 3; // yes it is
        break;
    }
    case 5: { // is children number 2 (окончание)
        if (stack.popBoolean()) {
            state = 7; // is deleting element in tree (окончание)
        } else {
            state = 11; // is inserting element in tree (окончание)
        }
        break;
    }
    case 6: { // is deleting element in tree
        state = 4; // is children number 2
        break;
    }
    case 7: { // is deleting element in tree (окончание)
        if (stack.popBoolean()) {
            state = 8; // no it is not
        } else {
            state = 9; // insert
        }
        break;
    }
    case 8: { // no it is not
        state = 6; // is deleting element in tree
        break;
    }
    case 9: { // insert
        state = 6; // is deleting element in tree
        break;
```

```
}
case 10: { // is inserting element in tree
    state = 4; // is children number 2
    break;
}
case 11: { // is inserting element in tree (окончание)
    if (stack.popBoolean()) {
        state = 12; // yes it is
    } else {
        state = 13; // insert
    }
    break;
}
case 12: { // yes it is
    state = 10; // is inserting element in tree
    break;
}
case 13: { // insert
    state = 10; // is inserting element in tree
    break;
}
case 14: { // is deleting element in first subtree
    state = 1; // is child leaf
    break;
}
case 15: { // is deleting element in first subtree (окончание)
    if (stack.popBoolean()) {
        state = 18; // delete auto (автомат)
    } else {
        state = 20; // is children number 2 (окончание)
    }
    break;
}
case 16: { // lt low2
    state = 14; // is deleting element in first subtree
    break;
}
case 17: { // init
    state = 16; // lt low2
    break;
}
case 18: { // delete auto (автомат)
    if (child.isAtStart()) {
        child = null;
        state = 17; // init
    }
    break;
}
case 19: { // is children number 2
    state = 14; // is deleting element in first subtree
    break;
}
case 20: { // is children number 2 (окончание)
    if (stack.popBoolean()) {
        state = 23; // delete auto (автомат)
    } else {
        state = 25; // is serching element in second subtree (окончание)
    }
    break;
}
case 21: { // gt low2
    state = 19; // is children number 2
    break;
}
case 22: { // init
    state = 21; // gt low2
    break;
}
case 23: { // delete auto (автомат)
    if (child.isAtStart()) {
        child = null;
        state = 22; // init
    }
    break;
```

```
}
case 24: { // is serching element in second subtree
    state = 19; // is children number 2
    break;
}
case 25: { // is serching element in second subtree (окончание)
    if (stack.popBoolean()) {
        state = 28; // delete auto (автомат)
    } else {
        state = 31; // delete auto (автомат)
    }
    break;
}
case 26: { // lt low3
    state = 24; // is serching element in second subtree
    break;
}
case 27: { // init
    state = 26; // lt low3
    break;
}
case 28: { // delete auto (автомат)
    if (child.isAtStart()) {
        child = null;
        state = 27; // init
    }
    break;
}
case 29: { // gt low3
    state = 24; // is serching element in second subtree
    break;
}
case 30: { // init
    state = 29; // gt low3
    break;
}
case 31: { // delete auto (автомат)
    if (child.isAtStart()) {
        child = null;
        state = 30; // init
    }
    break;
}
case 32: { // is 1 children
    state = 15; // is deleting element in first subtree (окончание)
    break;
}
case 33: { // is 1 children (окончание)
    if (stack.popBoolean()) {
        state = 61; // to parent
    } else {
        state = 62; // to parent
    }
    break;
}
case 34: { // is it 1 child
    state = 32; // is 1 children
    break;
}
case 35: { // is it 1 child (окончание)
    if (stack.popBoolean()) {
        state = 37; // is it 1 child (окончание)
    } else {
        state = 34; // is it 1 child
    }
    break;
}
case 36: { // is it 1 child
    state = 34; // is it 1 child
    break;
}
case 37: { // is it 1 child (окончание)
    if (stack.popBoolean()) {
        state = 38; // insert
```

```
    } else {
        state = 39; // insert
    }
    break;
}
case 38: { // insert
    state = 36; // is it 1 child
    break;
}
case 39: { // insert
    state = 36; // is it 1 child
    break;
}
case 40: { // is it 2 child
    state = 35; // is it 1 child (окончание)
    break;
}
case 41: { // is it 2 child (окончание)
    if (stack.popBoolean()) {
        state = 43; // is it 2 child (окончание)
    } else {
        state = 40; // is it 2 child
    }
    break;
}
case 42: { // is it 2 child
    state = 40; // is it 2 child
    break;
}
case 43: { // is it 2 child (окончание)
    if (stack.popBoolean()) {
        state = 45; // is it 1 child (окончание)
    } else {
        state = 49; // is it 1 child (окончание)
    }
    break;
}
case 44: { // is it 1 child
    state = 42; // is it 2 child
    break;
}
case 45: { // is it 1 child (окончание)
    if (stack.popBoolean()) {
        state = 46; // insert
    } else {
        state = 47; // insert
    }
    break;
}
case 46: { // insert
    state = 44; // is it 1 child
    break;
}
case 47: { // insert
    state = 44; // is it 1 child
    break;
}
case 48: { // is it 1 child
    state = 42; // is it 2 child
    break;
}
case 49: { // is it 1 child (окончание)
    if (stack.popBoolean()) {
        state = 50; // insert
    } else {
        state = 52; // is it 1 child (окончание)
    }
    break;
}
case 50: { // insert
    state = 48; // is it 1 child
    break;
}
case 51: { // is it 1 child
```

```
                state = 48; // is it 1 child
                break;
            }
            case 52: { // is it 1 child (окончание)
                if (stack.popBoolean()) {
                    state = 53; // insert
                } else {
                    state = 54; // insert
                }
                break;
            }
            case 53: { // insert
                state = 51; // is it 1 child
                break;
            }
            case 54: { // insert
                state = 51; // is it 1 child
                break;
            }
            case 55: { // is it 3 child
                state = 41; // is it 2 child (окончание)
                break;
            }
            case 56: { // is it 3 child (окончание)
                if (stack.popBoolean()) {
                    state = 58; // is it 1 child (окончание)
                } else {
                    state = 55; // is it 3 child
                }
                break;
            }
            case 57: { // is it 1 child
                state = 55; // is it 3 child
                break;
            }
            case 58: { // is it 1 child (окончание)
                if (stack.popBoolean()) {
                    state = 59; // insert
                } else {
                    state = 60; // insert
                }
                break;
            }
            case 59: { // insert
                state = 57; // is it 1 child
                break;
            }
            case 60: { // insert
                state = 57; // is it 1 child
                break;
            }
            case 61: { // to parent
                state = 56; // is it 3 child (окончание)
                break;
            }
            case 62: { // to parent
                state = 32; // is 1 children
                break;
            }
            case END_STATE: { // Начальное состояние
                state = 2; // is child leaf (окончание)
                break;
            }
        }

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

/**
  * Интересно ли текущее состояние.
  */
public boolean isInteresting(int level) {
    // Интересность
```

```
switch (state) {
    case START_STATE: // Начальное состояние
        return true;
    case 1: // is child leaf
        return level <= -1;
    case 2: // is child leaf (окончание)
        return level <= -1;
    case 3: // yes it is
        return level <= 1;
    case 4: // is children number 2
        return level <= -1;
    case 5: // is children number 2 (окончание)
        return level <= -1;
    case 6: // is deleting element in tree
        return level <= -1;
    case 7: // is deleting element in tree (окончание)
        return level <= -1;
    case 8: // no it is not
        return level <= 1;
    case 9: // insert
        return level <= 1;
    case 10: // is inserting element in tree
        return level <= -1;
    case 11: // is inserting element in tree (окончание)
        return level <= -1;
    case 12: // yes it is
        return level <= 1;
    case 13: // insert
        return level <= 1;
    case 14: // is deleting element in first subtree
        return level <= -1;
    case 15: // is deleting element in first subtree (окончание)
        return level <= -1;
    case 16: // lt low2
        return level <= 1;
    case 17: // init
        return level <= -1;
    case 18: // delete auto (автомат)
        return (child != null) && !child.isAtEnd();
    case 19: // is children number 2
        return level <= -1;
    case 20: // is children number 2 (окончание)
        return level <= -1;
    case 21: // gt low2
        return level <= 1;
    case 22: // init
        return level <= -1;
    case 23: // delete auto (автомат)
        return (child != null) && !child.isAtEnd();
    case 24: // is serching element in second subtree
        return level <= -1;
    case 25: // is serching element in second subtree (окончание)
        return level <= -1;
    case 26: // lt low3
        return level <= 1;
    case 27: // init
        return level <= -1;
    case 28: // delete auto (автомат)
        return (child != null) && !child.isAtEnd();
    case 29: // gt low3
        return level <= 1;
    case 30: // init
        return level <= -1;
    case 31: // delete auto (автомат)
        return (child != null) && !child.isAtEnd();
    case 32: // is 1 children
        return level <= -1;
    case 33: // is 1 children (окончание)
        return level <= -1;
    case 34: // is it 1 child
        return level <= -1;
    case 35: // is it 1 child (окончание)
        return level <= -1;
    case 36: // is it 1 child
```

```
            return level <= -1;
        case 37: // is it 1 child (окончание)
            return level <= -1;
        case 38: // insert
            return level <= 1;
        case 39: // insert
            return level <= 1;
        case 40: // is it 2 child
            return level <= -1;
        case 41: // is it 2 child (окончание)
            return level <= -1;
        case 42: // is it 2 child
            return level <= -1;
        case 43: // is it 2 child (окончание)
            return level <= -1;
        case 44: // is it 1 child
            return level <= -1;
        case 45: // is it 1 child (окончание)
            return level <= -1;
        case 46: // insert
            return level <= 1;
        case 47: // insert
            return level <= 1;
        case 48: // is it 1 child
            return level <= -1;
        case 49: // is it 1 child (окончание)
            return level <= -1;
        case 50: // insert
            return level <= 1;
        case 51: // is it 1 child
            return level <= -1;
        case 52: // is it 1 child (окончание)
            return level <= -1;
        case 53: // insert
            return level <= 1;
        case 54: // insert
            return level <= 1;
        case 55: // is it 3 child
            return level <= -1;
        case 56: // is it 3 child (окончание)
            return level <= -1;
        case 57: // is it 1 child
            return level <= -1;
        case 58: // is it 1 child (окончание)
            return level <= -1;
        case 59: // insert
            return level <= 1;
        case 60: // insert
            return level <= 1;
        case 61: // to parent
            return level <= -1;
        case 62: // to parent
            return level <= 1;
        case END_STATE: // Конечное состояние
            return true;
    }

    throw new RuntimeException("isInterest");
}

/**
  * Комментарий к текущему состоянию
  */
public String getComment() {
    String comment = "";
    Object[] args = null;
    // Выбор комментария
    switch (state) {
        case 3: { // yes it is
            comment = Tree23.this.getComment("deletefromsubtree.deleteyesitisleaf");
            args = new Object[]{new Integer(d.deletingElement)};
            break;
        }
        case 8: { // no it is not
```

```
            comment =
Tree23.this.getComment("deletefromsubtree.deletenoitisnotdeletingelementintree1");
            args = new Object[]{new Integer(d.deletingElement)};
            break;
        }
        case 9: { // insert
            comment =
Tree23.this.getComment("deletefromsubtree.deleteinsertelement1");
            break;
        }
        case 12: { // yes it is
            comment =
Tree23.this.getComment("deletefromsubtree.deleteyesitisinsertingelementintree2");
            args = new Object[]{new Integer(d.deletingElement)};
            break;
        }
        case 13: { // insert
            comment =
Tree23.this.getComment("deletefromsubtree.deleteinsertelement2");
            break;
        }
        case 16: { // lt low2
            comment = Tree23.this.getComment("deletefromsubtree.deleteltlow2");
            args = new Object[]{new Integer(d.currentNode.low2)};
            break;
        }
        case 18: { // delete auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 21: { // gt low2
            comment = Tree23.this.getComment("deletefromsubtree.insertgtlow2");
            args = new Object[]{new Integer(d.currentNode.low2)};
            break;
        }
        case 23: { // delete auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 26: { // lt low3
            comment = Tree23.this.getComment("deletefromsubtree.deleteltlow3");
            args = new Object[]{new Integer(d.currentNode.low2),
 new Integer(d.currentNode.low3)};
            break;
        }
        case 28: { // delete auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 29: { // gt low3
            comment = Tree23.this.getComment("deletefromsubtree.deletegtlow3");
            args = new Object[]{new Integer(d.currentNode.low3)};
            break;
        }
        case 31: { // delete auto (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 38: { // insert
            comment =
Tree23.this.getComment("deletefromsubtree.delete3childrenin2child");
            break;
        }
        case 39: { // insert
            comment =
Tree23.this.getComment("deletefromsubtree.delete2childrenin2child");
            break;
        }
        case 46: { // insert
            comment =
```

```
   Tree23.this.getComment("deletefromsubtree.delete3childrenin1child");
            break;
        }
        case 47: { // insert
            comment =
   Tree23.this.getComment("deletefromsubtree.delete2childrenin1child");
            break;
        }
        case 50: { // insert
            comment =
    Tree23.this.getComment("deletefromsubtree.delete3childrenin1child1");
            break;
        }
        case 53: { // insert
            comment =
   Tree23.this.getComment("deletefromsubtree.delete3childrenin3child");
            break;
        }
        case 54: { // insert
            comment =
   Tree23.this.getComment("deletefromsubtree.delete2childrenin1and2child");
            break;
        }
        case 59: { // insert
            comment =
   Tree23.this.getComment("deletefromsubtree.delete3childrenin2child1");
            break;
        }
        case 60: { // insert
            comment =
   Tree23.this.getComment("deletefromsubtree.delete2childrenin2child1");
            break;
        }
        case 62: { // to parent
            comment = Tree23.this.getComment("deletefromsubtree.deletegotoparent1");
            break;
        }
    }

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

/**
  * Выполняет действия по отрисовке состояния
  */
public void drawState() {
    switch (state) {
        case 3: { // yes it is
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 8: { // no it is not
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 9: { // insert
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 12: { // yes it is
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 13: { // insert
            d.visualizer.DrawTree();
            d.visualizer.UpdateScreen();
            break;
        }
        case 16: { // lt low2
            d.visualizer.DrawTree();
```

```
    d.visualizer.UpdateScreen();
    break;
}
case 17: { // init
    d.visualizer.DrawTree();
    d.visualizer.UpdateScreen();
    break;
}
case 18: { // delete auto (автомат)
    child.drawState();
    break;
}
case 21: { // gt low2
    d.visualizer.DrawTree();
    d.visualizer.UpdateScreen();
    break;
}
case 22: { // init
    break;
}
case 23: { // delete auto (автомат)
    child.drawState();
    break;
}
case 26: { // lt low3
    d.visualizer.DrawTree();
    d.visualizer.UpdateScreen();
    break;
}
case 27: { // init
    break;
}
case 28: { // delete auto (автомат)
    child.drawState();
    break;
}
case 29: { // gt low3
    d.visualizer.DrawTree();
    d.visualizer.UpdateScreen();
    break;
}
case 30: { // init
    break;
}
case 31: { // delete auto (автомат)
    child.drawState();
    break;
}
case 38: { // insert
    d.visualizer.DrawTree();
    d.visualizer.UpdateScreen();
    break;
}
case 39: { // insert
    d.visualizer.DrawTree();
    d.visualizer.UpdateScreen();
    break;
}
case 46: { // insert
    d.visualizer.DrawTree();
    d.visualizer.UpdateScreen();
    break;
}
case 47: { // insert
    d.visualizer.DrawTree();
    d.visualizer.UpdateScreen();
    break;
}
case 50: { // insert
    d.visualizer.DrawTree();
    d.visualizer.UpdateScreen();
    break;
}
case 53: { // insert
```

```
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                    break;
                }
                case 54: { // insert
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                    break;
                }
                case 59: { // insert
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                    break;
                }
                case 60: { // insert
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                    break;
                }
                case 61: { // to parent
                    break;
                }
                case 62: { // to parent
                    d.visualizer.DrawTree();
                    d.visualizer.UpdateScreen();
                    break;
                }
            }
        }

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

Приложение 4. Исходные коды интерфейса визуализатора

Tree.java

```
package ru.ifmo.vizi.Tree23;

import ru.ifmo.vizi.base.Configuration;
import ru.ifmo.vizi.base.SmartTokenizer;

import java.awt.*;
import java.io.IOException;


/**
 * 2-3 tree
 */
public class Tree
{
    /**
     * Number of children
     */
    int childrenNumber;

    /**
     * Children
     */
    public Node rootNode1, rootNode2;

    /**
     * Node key
     */
    public int low1, low2;

    /**
     * Configuration
     */
    Configuration config;

    /**
     * Client pane
     */
    Container clientPane;

    /**
     * Sets defaut styles
     */
    void SetDefaultStyles()
    {
        if (childrenNumber == 1)
        {
            rootNode1.SetDefaultStyles();
        }

        if (childrenNumber == 2)
        {
            rootNode1.SetDefaultStyles();
            rootNode2.SetDefaultStyles();
        }
    }

    /**
     * Adds tree
```

```
 *
 * @param clientPane - client pane
 * @param config     - configuration
 */
public void InitAndAddToClientPane(Container clientPane,
                                   Configuration config)
{
    this.config = config;
    this.clientPane = clientPane;

    if (childrenNumber == 1)
    {
        rootNode1.InitAndAddToClientPaneSubtree(clientPane, config);
    }

    if (childrenNumber == 2)
    {
        rootNode1.InitAndAddToClientPaneSubtree(clientPane, config);
        rootNode2.InitAndAddToClientPaneSubtree(clientPane, config);
    }
}

/**
 * Removes component
 */
public void RemoveFromClientPane()
{
    if (childrenNumber == 1)
    {
        rootNode1.RemoveFromClientPaneSubtree();
    }

    if (childrenNumber == 2)
    {
        rootNode1.RemoveFromClientPaneSubtree();
        rootNode2.RemoveFromClientPaneSubtree();
    }
}

/**
 * Seves tree
 *
 * @param toSave - string to save
 * @return tree as string
 */
public StringBuffer Save(StringBuffer toSave)
{
    if (childrenNumber != 0)
    {
        toSave.append("(");
        toSave = rootNode1.Save(toSave);
        toSave.append(")");
    }

    return toSave;
}

/**
 * Seves tree
 *
 * @return tree as string
 */
public String Save()
{
    StringBuffer toSave = new StringBuffer();

    if (childrenNumber != 0)
    {
        toSave.append("(");
        toSave = rootNode1.Save(toSave);
        toSave.append(")");
    }

    return toSave.toString();
```

```
}


/**
 * Loads tree
 *
 * @param toLoad - string to load
 */
public boolean Load(String toLoad, Configuration config)
{
    SmartTokenizer tokenizer =
            new SmartTokenizer(toLoad, config);

    try
    {
        tokenizer.nextToken();

        if (tokenizer.ttype == tokenizer.TT_EOF)
        {
            childrenNumber = 0;
            return true;
        }

        if (tokenizer.ttype == '(')
        {
            childrenNumber = 1;

            rootNode1 = new Node();
            rootNode1.isRoot = true;
            rootNode1.thisChildNumber = 1;

            int returned[] = rootNode1.Load(tokenizer, 0);
            if (returned[0] == 0)
            {
                return false;
            }
            low1 = rootNode1.low1;
        }
        else
        {
            //error
            return false;
        }

        tokenizer.nextToken();

        if (tokenizer.ttype != tokenizer.TT_EOF)
        {
            //error
            return false;
        }
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }

    return true;
}

/**
 * Loads tree
 *
 * @param tokenizer - string to load
 */
public int Load(SmartTokenizer tokenizer)
{
    try
    {
        tokenizer.nextToken();

        if (tokenizer.ttype == tokenizer.TT_EOF)
        {
            childrenNumber = 0;
```

```
            return 1;
        }

        if (tokenizer.ttype == '(')
        {
            childrenNumber = 1;

            rootNode1 = new Node();
            rootNode1.isRoot = true;
            rootNode1.thisChildNumber = 1;

            int returned[] = rootNode1.Load(tokenizer, 0);
            if (returned[3] <= 0)
            {
                return returned[3];
            }
            low1 = rootNode1.low1;
        }
        else
        {
            //error
            return -4;
        }

        tokenizer.nextToken();

        if (tokenizer.ttype != tokenizer.TT_EOF)
        {
            //error
            return -5;
        }
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }

    return 1;
}

/**
 * Draws tree
 */
public void Draw(int winWidth, int winHeight)
{
    int coordinatePaneDx = 0, coordinatePaneDy = 0;
    coordinatePaneDx = winWidth /
                       (config.getInteger("max-element-number") + 1);
    coordinatePaneDy = (int) (winHeight /
                              (Math.log(
                                      config.getInteger("max-element-number")
                                      + 1) / Math.log(2) + 1));
    int deltaX;
    deltaX = (winWidth - (FindLeafsNumber() + 1) * coordinatePaneDx) / 2;

    if (childrenNumber == 1)
    {
        rootNode1.Draw(0,
                winWidth,
                winHeight,
                deltaX,
                coordinatePaneDx,
                coordinatePaneDy);
    }

    if (childrenNumber == 2)
    {
        rootNode2.Draw(rootNode1.Draw(0,
                winWidth,
                winHeight,
                deltaX,
                coordinatePaneDx,
                coordinatePaneDy)[0],
                winWidth,
```

```
                winHeight,
                deltaX,
                coordinatePaneDx,
                coordinatePaneDy);
    }
}

/**
 * Finds leaf by element
 *
 * @param element - element
 * @return
 */
public int FindChild(int element)
{
    if (childrenNumber == 1)
    {
        return 1;
    }

    if (childrenNumber == 2)
    {
        if (element < low2)
        {
            return 1;
        }
        else
        {
            return 2;
        }
    }

    return 0;
}

/**
 * Adds element
 *
 * @param newChild - node
 * @return
 */
public int AddChild(Node newChild)
{
    int toReturn = 0;

    if (childrenNumber == 1)
    {
        if (low1 < newChild.low1)
        {
            rootNode2 = newChild;
            rootNode2.InitAndAddToClientPaneSubtree(clientPane, config);
            childrenNumber++;
            toReturn = 2;
        }
        else
        {
            rootNode2 = rootNode1;
            rootNode1 = newChild;
            rootNode1.InitAndAddToClientPaneSubtree(clientPane, config);
            childrenNumber++;
            toReturn = 1;
        }
        low1 = rootNode1.low1;
        low2 = rootNode2.low1;

        rootNode1.thisChildNumber = 1;
        rootNode2.thisChildNumber = 2;
    }

    if (childrenNumber == 0)
    {
        rootNode1 = newChild;
        rootNode1.InitAndAddToClientPaneSubtree(clientPane, config);
        childrenNumber++;
```

```
        toReturn = 1;

        low1 = rootNode1.low1;
        rootNode1.thisChildNumber = 1;
    }

    return toReturn;
}

/**
 * Deletes element
 *
 * @param childNumber - node
 */
public void DeleteChild(int childNumber)
{
    if (childrenNumber == 1)
    {
        rootNode1.RemoveFromClientPaneSubtree();
        childrenNumber--;
    }

    if (childrenNumber == 2)
    {
        if (childNumber == 2)
        {
            rootNode2.RemoveFromClientPaneSubtree();
            childrenNumber--;
        }
        else
        {
            rootNode1.RemoveFromClientPaneSubtree();
            rootNode1 = rootNode2;
            childrenNumber--;
        }
        low1 = rootNode1.low1;

        rootNode1.thisChildNumber = 1;
    }
}

/**
 * Creates Leaf
 *
 * @param element - new element
 * @return
 */
public int NewChildLeaf(int element)
{
    Node leaf = new Node();

    leaf.isLeaf = true;
    leaf.isRoot = true;
    leaf.childrenNumber = 0;
    leaf.element = element;
    leaf.low1 = element;
    leaf.style = config.getInteger("leaf-default-style");

    return AddChild(leaf);
}

/**
 * Creates new node with 1 child
 *
 * @param child1 - new node
 * @return
 */
public int NewChildInterior(Node child1)
{
    Node interior = new Node();

    interior.child1 = child1;
    interior.child1.parent = interior;
    interior.isLeaf = false;
```

```
    interior.isRoot = true;
    interior.childrenNumber = 1;
    interior.low1 = child1.low1;
    interior.style = config.getInteger("interior-default-style");

    return AddChild(interior);
}

/**
 * Creates new node with 2 children
 *
 * @param child1 - child 1
 * @param child2 - child 2
 * @return
 */
public int NewChildInterior(Node child1, Node child2)
{
    Node interior = new Node();

    interior.child1 = child1;
    interior.child2 = child2;
    interior.child1.parent = interior;
    interior.child2.parent = interior;
    interior.isLeaf = false;
    interior.isRoot = true;
    interior.childrenNumber = 2;
    interior.low1 = child1.low1;
    interior.low2 = child2.low1;
    interior.style = config.getInteger("interior-default-style");

    return AddChild(interior);
}

/**
 * Creates new node with 3 children
 *
 * @param child1 - child 1
 * @param child2 - child 2
 * @param child3 - child 3
 * @return
 */
public int NewChildInterior(Node child1, Node child2, Node child3)
{
    Node interior = new Node();

    interior.child1 = child1;
    interior.child2 = child2;
    interior.child3 = child3;
    interior.child1.parent = interior;
    interior.child2.parent = interior;
    interior.child3.parent = interior;
    interior.isLeaf = false;
    interior.isRoot = true;
    interior.childrenNumber = 3;
    interior.low1 = child1.low1;
    interior.low2 = child2.low1;
    interior.low3 = child3.low1;
    interior.style = config.getInteger("interior-default-style");

    return AddChild(interior);
}

/**
 * Finds leafs number
 *
 * @return number of leafs
 */
public int FindLeafsNumber()
{
    int leafsNumber = 0;

    if (childrenNumber == 1)
    {
        leafsNumber = rootNode1.FindLeafsNumber(0);
```

```
        }
        if (childrenNumber == 2)
        {
            leafsNumber = rootNode1.FindLeafsNumber(0);
            leafsNumber = rootNode2.FindLeafsNumber(leafsNumber);
        }

        return leafsNumber;
    }
}
```

Node.java

```
package ru.ifmo.vizi.Tree23;

import ru.ifmo.vizi.base.widgets.ShapeStyle;
import ru.ifmo.vizi.base.widgets.Rect;
import ru.ifmo.vizi.base.widgets.Ellipse;
import ru.ifmo.vizi.base.Configuration;
import ru.ifmo.vizi.base.SmartTokenizer;

import java.awt.*;
import java.io.IOException;

/**
 * Node of tree
 */
public class Node
{
    /**
     * Shows is node leaf
     */
    public boolean isLeaf;

    /**
     * Shows is node root
     */
    public boolean isRoot;

    /**
     * Number of children
     */
    int childrenNumber;

    /**
     * Leaf element
     */
    public int element;

    /**
     * Child numder of this node
     */
    int thisChildNumber;

    /**
     * Children
     */
    public Node child1, child2, child3, child4;

    /**
     * Parent
     */
    public Node parent;

    /**
     * Node's keys
     */
    public int low1, low2, low3, low4;

    /**
```

```
 * Configuration
 */
Configuration config;

/**
 * Client pane
 */
Container clientPane;

/**
 * Shows is element inited
 */
boolean isInited = false;

/**
 * Information area
 */
private Rect infoRects[];

/**
 * Information area
 */
private Ellipse infoCircles[];


/**
 * Lines
 */
private Line lines[];


/**
 * Temporary varaible for load string check
 */
private int rang;


/**
 * Styles
 */
int style;

/**
 * Sets "current-node-style"
 */
void SetCurrentNodeStyle()
{
    style = config.getInteger("current-node-style");
}

/**
 * Constructor
 */
public Node()
{

}


/**
 * Sets old style
 */
void SetOldStyle()
{
    if (isLeaf)
    {
        style = config.getInteger("leaf-default-style");
    }
    else
    {
        style = config.getInteger("interior-default-style");
    }
}
```

```
/**
 * Sets defaut styles
 */
void SetDefaultStyles()
{
    if (childrenNumber == 1)
    {
        child1.SetDefaultStyles();
    }
    if (childrenNumber == 2)
    {
        child1.SetDefaultStyles();
        child2.SetDefaultStyles();
    }
    if (childrenNumber == 3)
    {
        child1.SetDefaultStyles();
        child2.SetDefaultStyles();
        child3.SetDefaultStyles();
    }
    if (childrenNumber == 4)
    {
        child1.SetDefaultStyles();
        child2.SetDefaultStyles();
        child3.SetDefaultStyles();
        child4.SetDefaultStyles();
    }

    if (isLeaf)
    {
        style = config.getInteger("leaf-default-style");
    }
    else
    {
        style = config.getInteger("interior-default-style");
    }
}

/**
 * Adds component
 *
 * @param clientPane - client pane
 * @param config     - configuration
 */
public void InitAndAddToClientPane(Container clientPane,
                                   Configuration config)
{
    if (!isInited)
    {
        this.config = config;
        this.clientPane = clientPane;

        infoRects = new Rect[4];
        infoRects[0] = new Rect(ShapeStyle.loadStyleSet(config, "styles"));
        infoRects[1] = new Rect(ShapeStyle.loadStyleSet(config, "styles"));
        infoRects[2] = new Rect(ShapeStyle.loadStyleSet(config, "styles"));
        clientPane.add(infoRects[0]);
        clientPane.add(infoRects[1]);
        clientPane.add(infoRects[2]);

        infoCircles = new Ellipse[1];
        infoCircles[0] = new Ellipse(ShapeStyle.loadStyleSet(config,
                "styles"));
        clientPane.add(infoCircles[0]);
        infoCircles[0].setBounds(-10, -10, 3, 3);

        lines = new Line[4];
        lines[0] = new Line(ShapeStyle.loadStyleSet(config, "styles"));
        lines[1] = new Line(ShapeStyle.loadStyleSet(config, "styles"));
        lines[2] = new Line(ShapeStyle.loadStyleSet(config, "styles"));
        lines[3] = new Line(ShapeStyle.loadStyleSet(config, "styles"));
        clientPane.add(lines[0]);
        clientPane.add(lines[1]);
        clientPane.add(lines[2]);
```

```
        clientPane.add(lines[3]);

        isInited = true;
    }
}

/**
 * Removes component
 */
public void RemoveFromClientPane()
{
    if (isInited)
    {
        clientPane.remove(infoRects[0]);
        clientPane.remove(infoRects[1]);
        clientPane.remove(infoRects[2]);

        clientPane.remove(infoCircles[0]);

        clientPane.remove(lines[0]);
        clientPane.remove(lines[1]);
        clientPane.remove(lines[2]);
        clientPane.remove(lines[3]);

        isInited = false;
    }
}

/**
 * Adds subtree
 *
 * @param clientPane - client pane
 * @param config     - configuration
 */
public void InitAndAddToClientPaneSubtree(Container clientPane,
                                          Configuration config)
{
    if (childrenNumber == 1)
    {
        child1.InitAndAddToClientPaneSubtree(clientPane, config);
    }
    if (childrenNumber == 2)
    {
        child1.InitAndAddToClientPaneSubtree(clientPane, config);
        child2.InitAndAddToClientPaneSubtree(clientPane, config);
    }
    if (childrenNumber == 3)
    {
        child1.InitAndAddToClientPaneSubtree(clientPane, config);
        child2.InitAndAddToClientPaneSubtree(clientPane, config);
        child3.InitAndAddToClientPaneSubtree(clientPane, config);
    }
    if (childrenNumber == 4)
    {
        child1.InitAndAddToClientPaneSubtree(clientPane, config);
        child2.InitAndAddToClientPaneSubtree(clientPane, config);
        child3.InitAndAddToClientPaneSubtree(clientPane, config);
        child4.InitAndAddToClientPaneSubtree(clientPane, config);
    }

    InitAndAddToClientPane(clientPane, config);
}

/**
 * Removes subtree
 */
public void RemoveFromClientPaneSubtree()
{
    RemoveFromClientPane();

    if (childrenNumber == 1)
    {
        child1.RemoveFromClientPaneSubtree();
    }
```

```
    if (childrenNumber == 2)
    {
        child1.RemoveFromClientPaneSubtree();
        child2.RemoveFromClientPaneSubtree();
    }
    if (childrenNumber == 3)
    {
        child1.RemoveFromClientPaneSubtree();
        child2.RemoveFromClientPaneSubtree();
        child3.RemoveFromClientPaneSubtree();
    }
    if (childrenNumber == 4)
    {
        child1.RemoveFromClientPaneSubtree();
        child2.RemoveFromClientPaneSubtree();
        child3.RemoveFromClientPaneSubtree();
        child4.RemoveFromClientPaneSubtree();
    }
}

/**
 * Seves tree
 *
 * @param toSave - string to save
 * @return tree as string
 */
public StringBuffer Save(StringBuffer toSave)
{
    if (isLeaf)
    {
        toSave.append(element);
    }
    else
    {
        if (childrenNumber == 1)
        {
            toSave.append("(");
            toSave = child1.Save(toSave);
            toSave.append(")");
        }
        if (childrenNumber == 2)
        {
            toSave.append("(");
            toSave = child1.Save(toSave);
            toSave.append(")");
            toSave.append("(");
            toSave = child2.Save(toSave);
            toSave.append(")");
        }
        if (childrenNumber == 3)
        {
            toSave.append("(");
            toSave = child1.Save(toSave);
            toSave.append(")");
            toSave.append("(");
            toSave = child2.Save(toSave);
            toSave.append(")");
            toSave.append("(");
            toSave = child3.Save(toSave);
            toSave.append(")");
        }
    }

    return toSave;
}

/**
 * Loads tree
 *
 * @param tokenizer - string to load
 * @return low, leafCount and rang
 */
public int[] Load(SmartTokenizer tokenizer, int leafCount)
        throws IOException
```

```
{
    int[] toReturn = new int[4];
    toReturn[3] = 1;
    int[] returned = new int[4];


    tokenizer.nextToken();

    if (tokenizer.ttype == '(' || tokenizer.ttype == ')')
    {
        while (tokenizer.ttype == '(')
        {
            if (!isLeaf)
            {
                if (childrenNumber == 3)
                {
                    toReturn[3] = 0;
                    return toReturn;
                }

                if (childrenNumber == 2)
                {
                    childrenNumber = 3;
                    child3 = new Node();
                    child3.parent = this;
                    child3.thisChildNumber = 3;
                    returned = child3.Load(tokenizer, leafCount);
                    if (returned[3] <= 0)
                    {
                        toReturn[3] = returned[3];
                        return toReturn;
                    }
                    low3 = returned[0];
                    if (low3 <= low2)
                    {
                        toReturn[3] = -1;
                        return toReturn;
                    }
                    if (rang != returned[2])
                    {
                        toReturn[3] = -2;
                        return toReturn;
                    }
                }

                if (childrenNumber == 1)
                {
                    childrenNumber = 2;
                    child2 = new Node();
                    child2.parent = this;
                    child2.thisChildNumber = 2;
                    returned = child2.Load(tokenizer, leafCount);
                    if (returned[3] <= 0)
                    {
                        toReturn[3] = returned[3];
                        return toReturn;
                    }
                    low2 = returned[0];
                    if (low2 <= low1)
                    {
                        toReturn[3] = -1;
                        return toReturn;
                    }
                    if (rang != returned[2])
                    {
                        toReturn[3] = -2;
                        return toReturn;
                    }
                }

                if (childrenNumber == 0)
                {
                    childrenNumber = 1;
                    child1 = new Node();
```

```
                    child1.parent = this;
                    child1.thisChildNumber = 1;
                    returned = child1.Load(tokenizer, leafCount);
                    if (returned[3] <= 0)
                    {
                        toReturn[3] = returned[3];
                        return toReturn;
                    }
                    low1 = returned[0];
                    rang = returned[2];
                }
            }
            else
            {
                toReturn[3] = -2;
                return toReturn;
            }

            tokenizer.nextToken();
        }

        if (tokenizer.ttype == ')')
        {
            if (isLeaf || isRoot || (childrenNumber >= 2))
            {
                if (isLeaf)
                {
                    rang = 0;
                }
                toReturn[0] = low1;
                toReturn[1] = leafCount;
                toReturn[2] = rang;

                return toReturn;
            }
        }


        toReturn[3] = -3;
        return toReturn;
    }

    if (tokenizer.ttype == tokenizer.TT_NUMBER)
    {

        isLeaf = true;

        element = (int) tokenizer.nval;
        low1 = element;

        leafCount++;

        return Load(tokenizer, leafCount);
    }

    //error

    toReturn[3] = -3;

    return toReturn;
}


/**
 * Draws this component
 *
 * @return leafCount, rang and coordinate
 */
public int[] Draw
        (int leafCount,
         int winWidth,
         int winHeight,
         int deltaX,
         int coordinatePaneDx,
```

```
         int coordinatePaneDy)
{
    int toReturn[] = new int[3];

    int cx = 0, cy = 0, cxr = 0, cyr = 0, width = 0, height = 0;
    int rang = 0;

    if (!isLeaf)
    {
        if (childrenNumber == 1)
        {
            for (int i = 1; i < 3; i++)
            {
                clientPane.remove(infoRects[i]);
            }
        }
        else
        {
            for (int i = childrenNumber - 1; i < 3; i++)
            {
                clientPane.remove(infoRects[i]);
            }
        }
    }

    if (!isLeaf)
    {
        clientPane.remove(infoCircles[0]);
    }

    if (isLeaf)
    {
        for (int i = 0; i < 3; i++)
        {
            clientPane.remove(infoRects[i]);
        }
    }

    if (isLeaf)
    {
        leafCount++;

        rang = 0;

        cx = leafCount * coordinatePaneDx + deltaX;
        cy = winHeight - coordinatePaneDy;
        width = config.getInteger("leaf-width");
        height = config.getInteger("leaf-height");

        infoCircles[0].setStyle(style);
        infoCircles[0].setBounds(cx - config.getInteger("leaf-width") / 2,
                cy - config.getInteger("leaf-height") / 2,
                width,
                height);
        infoCircles[0].setMessage(Integer.toString(element));
        infoCircles[0].adjustFontSize();

        toReturn[0] = leafCount;
        toReturn[1] = rang;
        toReturn[2] = cx;

        return toReturn;
    }
    else
    {
        if (childrenNumber == 1)
        {
            int returned1[] = child1.Draw(leafCount, winWidth, winHeight,
                    deltaX, coordinatePaneDx, coordinatePaneDy);

            leafCount = returned1[0];
            rang = returned1[1] + 1;

            cx = returned1[2];
```

```
    cy = winHeight - rang *
                    coordinatePaneDy -
         coordinatePaneDy;
    width = config.getInteger("interior-width");
    height = config.getInteger("interior-height");


    infoRects[0].setStyle(style);
    infoRects[0].setBounds(cx - config.getInteger("interior-width") / 2,
            cy - config.getInteger("interior-height") / 2,
            width,
            height);
    infoRects[0].setMessage("");
    infoRects[0].adjustFontSize();

    cxr = returned1[2];
    cyr = winHeight - returned1[1] *
                     coordinatePaneDy -
          coordinatePaneDy;

    lines[0].setStyle(config.getInteger("line-style"));
    lines[0].setCoordinates(cx, cy, cxr, cyr);
}

if (childrenNumber == 2)
{
    int returned1[] = child1.Draw(leafCount, winWidth, winHeight,
            deltaX, coordinatePaneDx, coordinatePaneDy);
    int returned2[] = child2.Draw(returned1[0], winWidth, winHeight,
            deltaX, coordinatePaneDx, coordinatePaneDy);

    leafCount = returned2[0];
    rang = returned1[1] + 1;

    cx = (returned1[2] + returned2[2]) / 2;
    cy = winHeight - rang *
                    coordinatePaneDy -
         coordinatePaneDy;
    width = config.getInteger("interior-width");
    height = config.getInteger("interior-height");


    infoRects[0].setStyle(style);
    infoRects[0].setBounds
            (cx - config.getInteger("interior-width") / 2,
            cy - config.getInteger("interior-height") / 2,
            width,
            height);
    infoRects[0].setMessage(Integer.toString(low2));
    infoRects[0].adjustFontSize();

    cxr = returned1[2];
    cyr = winHeight - returned1[1] *
                     coordinatePaneDy -
          coordinatePaneDy;

    lines[0].setStyle(config.getInteger("line-style"));
    lines[0].setCoordinates(cx, cy, cxr, cyr);

    cxr = returned2[2];
    cyr = winHeight - returned2[1] *
                     coordinatePaneDy -
          coordinatePaneDy;

    lines[1].setStyle(config.getInteger("line-style"));
    lines[1].setCoordinates(cx, cy, cxr, cyr);
}

if (childrenNumber == 3)
{
    int returned1[] = child1.Draw(leafCount, winWidth, winHeight,
            deltaX, coordinatePaneDx, coordinatePaneDy);
    int returned2[] = child2.Draw(returned1[0], winWidth, winHeight,
            deltaX, coordinatePaneDx, coordinatePaneDy);
```

```
    int returned3[] = child3.Draw(returned2[0], winWidth, winHeight,
            deltaX, coordinatePaneDx, coordinatePaneDy);

    leafCount = returned3[0];
    rang = returned1[1] + 1;

    cx = (returned1[2] + returned2[2] + returned3[2]) / 3;
    cy = winHeight - rang *
                    coordinatePaneDy -
         coordinatePaneDy;
    width = config.getInteger("interior-width");
    height = config.getInteger("interior-height");


    infoRects[0].setStyle(style);
    infoRects[0].setBounds
            (cx - config.getInteger("interior-width") + 1,
            cy - config.getInteger("interior-height") / 2,
            width,
            height);
    infoRects[0].setMessage(Integer.toString(low2));
    infoRects[0].adjustFontSize();

    infoRects[1].setStyle(style);
    infoRects[1].setBounds(cx,
            cy - config.getInteger("interior-height") / 2,
            width,
            height);
    infoRects[1].setMessage(Integer.toString(low3));
    infoRects[1].adjustFontSize();


    cxr = returned1[2];
    cyr = winHeight - returned1[1] *
                      coordinatePaneDy -
          coordinatePaneDy;

    lines[0].setStyle(config.getInteger("line-style"));
    lines[0].setCoordinates(cx, cy, cxr, cyr);

    cxr = returned2[2];
    cyr = winHeight - returned2[1] *
                      coordinatePaneDy -
          coordinatePaneDy;

    lines[1].setStyle(config.getInteger("line-style"));
    lines[1].setCoordinates(cx, cy, cxr, cyr);

    cxr = returned3[2];
    cyr = winHeight - returned3[1] *
                      coordinatePaneDy -
          coordinatePaneDy;

    lines[2].setStyle(config.getInteger("line-style"));
    lines[2].setCoordinates(cx, cy, cxr, cyr);


}

if (childrenNumber == 4)
{
    int returned1[] = child1.Draw(leafCount, winWidth, winHeight
            , deltaX, coordinatePaneDx, coordinatePaneDy);
    int returned2[] = child2.Draw(returned1[0], winWidth, winHeight,
            deltaX, coordinatePaneDx, coordinatePaneDy);
    int returned3[] = child3.Draw(returned2[0], winWidth, winHeight,
            deltaX, coordinatePaneDx, coordinatePaneDy);
    int returned4[] = child4.Draw(returned3[0], winWidth, winHeight,
            deltaX, coordinatePaneDx, coordinatePaneDy);

    leafCount = returned4[0];
    rang = returned1[1] + 1;

    cx = (returned1[2] +
```

```
            returned2[2] +
            returned3[2] +
            returned4[2]) / 4;
        cy = winHeight - rang *
                         coordinatePaneDy -
             coordinatePaneDy;
        width = config.getInteger("interior-width");
        height = config.getInteger("interior-height");


        infoRects[0].setStyle(style);
        infoRects[0].setBounds
                (cx - 3 * config.getInteger("interior-width") / 2 + 1,
                cy - config.getInteger("interior-height") / 2,
                width,
                height);
        infoRects[0].setMessage(Integer.toString(low2));
        infoRects[0].adjustFontSize();

        infoRects[1].setStyle(style);
        infoRects[1].setBounds
                (cx - config.getInteger("interior-width") / 2,
                cy - config.getInteger("interior-height") / 2,
                width,
                height);
        infoRects[1].setMessage(Integer.toString(low3));
        infoRects[1].adjustFontSize();

        infoRects[2].setStyle(style);
        infoRects[2].setBounds
                (cx + config.getInteger("interior-width") / 2 - 1,
                cy - config.getInteger("interior-height") / 2,
                width,
                height);
        infoRects[2].setMessage(Integer.toString(low4));
        infoRects[2].adjustFontSize();


        cxr = returned1[2];
        cyr = winHeight - returned1[1] *
                          coordinatePaneDy -
              coordinatePaneDy;

        lines[0].setStyle(config.getInteger("line-style"));
        lines[0].setCoordinates(cx, cy, cxr, cyr);

        cxr = returned2[2];
        cyr = winHeight - returned2[1] *
                          coordinatePaneDy -
              coordinatePaneDy;

        lines[1].setStyle(config.getInteger("line-style"));
        lines[1].setCoordinates(cx, cy, cxr, cyr);

        cxr = returned3[2];
        cyr = winHeight - returned3[1] *
                          coordinatePaneDy -
              coordinatePaneDy;

        lines[2].setStyle(config.getInteger("line-style"));
        lines[2].setCoordinates(cx, cy, cxr, cyr);

        cxr = returned4[2];
        cyr = winHeight - returned4[1] *
                          coordinatePaneDy -
              coordinatePaneDy;

        lines[3].setStyle(config.getInteger("line-style"));
        lines[3].setCoordinates(cx, cy, cxr, cyr);


    }

    for (int i = 0; i < 4; i++)
```

```
        {
            clientPane.remove(lines[i]);
        }
        if (!isLeaf)
        {
            for (int i = 0; i < childrenNumber; i++)
            {
                clientPane.add(lines[i]);
            }
        }


        toReturn[0] = leafCount;
        toReturn[1] = rang;
        toReturn[2] = cx;

        return toReturn;
    }
}

public int FindChild(int element)
{
    if (childrenNumber == 1)
    {
        return 1;
    }

    if (childrenNumber == 2)
    {
        if (element < low2)
        {
            return 1;
        }
        else
        {
            return 2;
        }
    }

    if (childrenNumber == 3)
    {
        if (element < low2)
        {
            return 1;
        }
        else
        {
            if (element < low3)
            {
                return 2;
            }
            else
            {
                return 3;
            }
        }
    }

    if (childrenNumber == 4)
    {
        if (element < low2)
        {
            return 1;
        }
        else
        {
            if (element < low3)
            {
                return 2;
            }
            else
            {
                if (element < low4)
                {
```

```
                    return 3;
                }
                else
                {
                    return 4;
                }
            }
        }
    }

    return 0;
}

public int AddChild(Node newChild)
{
    int toReturn = 0;

    newChild.parent = this;

    if (childrenNumber == 3)
    {
        if (low3 < newChild.low1)
        {
            child4 = newChild;
            child4.InitAndAddToClientPaneSubtree(clientPane, config);
            childrenNumber++;
            toReturn = 4;
        }
        else
        {
            if (low2 < newChild.low1)
            {
                child4 = child3;
                child3 = newChild;
                child3.InitAndAddToClientPaneSubtree(clientPane, config);
                childrenNumber++;
                toReturn = 3;
            }
            else
            {
                if (low1 < newChild.low1)
                {
                    child4 = child3;
                    child3 = child2;
                    child2 = newChild;
                    child2.InitAndAddToClientPaneSubtree(clientPane, config);
                    childrenNumber++;
                    toReturn = 2;
                }
                else
                {
                    child4 = child3;
                    child3 = child2;
                    child2 = child1;
                    child1 = newChild;
                    child1.InitAndAddToClientPaneSubtree(clientPane, config);
                    childrenNumber++;
                    toReturn = 1;
                }
            }
        }
        low1 = child1.low1;
        low2 = child2.low1;
        low3 = child3.low1;
        low4 = child4.low1;

        child1.thisChildNumber = 1;
        child2.thisChildNumber = 2;
        child3.thisChildNumber = 3;
        child4.thisChildNumber = 4;


    }
```

```
if (childrenNumber == 2)
{
    if (low2 < newChild.low1)
    {
        child3 = newChild;
        child3.InitAndAddToClientPaneSubtree(clientPane, config);
        childrenNumber++;
        toReturn = 3;
    }
    else
    {
        if (low1 < newChild.low1)
        {
            child3 = child2;
            child2 = newChild;
            child2.InitAndAddToClientPaneSubtree(clientPane, config);
            childrenNumber++;
            toReturn = 2;
        }
        else
        {
            child3 = child2;
            child2 = child1;
            child1 = newChild;
            child1.InitAndAddToClientPaneSubtree(clientPane, config);
            childrenNumber++;
            toReturn = 1;
        }
    }
    low1 = child1.low1;
    low2 = child2.low1;
    low3 = child3.low1;

    child1.thisChildNumber = 1;
    child2.thisChildNumber = 2;
    child3.thisChildNumber = 3;
}

if (childrenNumber == 1)
{
    if (low1 < newChild.low1)
    {
        child2 = newChild;
        child2.InitAndAddToClientPaneSubtree(clientPane, config);
        childrenNumber++;
        toReturn = 2;
    }
    else
    {
        child2 = child1;
        child1 = newChild;
        child1.InitAndAddToClientPaneSubtree(clientPane, config);
        childrenNumber++;
        toReturn = 1;
    }
    low1 = child1.low1;
    low2 = child2.low1;

    child1.thisChildNumber = 1;
    child2.thisChildNumber = 2;
}

if (childrenNumber == 0)
{
    child1 = newChild;
    child1.InitAndAddToClientPaneSubtree(clientPane, config);
    childrenNumber++;
    toReturn = 1;

    low1 = child1.low1;

    child1.thisChildNumber = 1;
}
```

```
    RemoveFromClientPane();
    InitAndAddToClientPane(clientPane, config);

    if (toReturn == 0)
    {
    }
    return toReturn;
}

public void DeleteChild(int childNumber)
{
    if (childrenNumber == 2)
    {
        if (childNumber == 2)
        {
            child2.RemoveFromClientPaneSubtree();
            childrenNumber--;
        }
        else
        {
            child1.RemoveFromClientPaneSubtree();
            child1 = child2;
            childrenNumber--;
        }
        low1 = child1.low1;

        child1.thisChildNumber = 1;
    }

    if (childrenNumber == 3)
    {
        if (childNumber == 3)
        {
            child3.RemoveFromClientPaneSubtree();
            childrenNumber--;
        }
        else
        {
            if (childNumber == 2)
            {
                child2.RemoveFromClientPaneSubtree();
                child2 = child3;
                childrenNumber--;
            }
            else
            {
                child1.RemoveFromClientPaneSubtree();
                child1 = child2;
                child2 = child3;
                childrenNumber--;
            }
        }
        low1 = child1.low1;
        low2 = child2.low1;

        child1.thisChildNumber = 1;
        child2.thisChildNumber = 2;
    }

    if (childrenNumber == 4)
    {
        if (childNumber == 4)
        {
            child4.RemoveFromClientPaneSubtree();
            childrenNumber--;
        }
        else
        {
            if (childNumber == 3)
            {
                child3.RemoveFromClientPaneSubtree();
                child3 = child4;
                childrenNumber--;
            }
```

```
            else
            {
                if (childNumber == 2)
                {
                    child2.RemoveFromClientPaneSubtree();
                    child2 = child3;
                    child3 = child4;
                    childrenNumber--;
                }
                else
                {
                    child1.RemoveFromClientPaneSubtree();
                    child1 = child2;
                    child2 = child3;
                    child3 = child4;
                    childrenNumber--;
                }
            }
        }
        low1 = child1.low1;
        low2 = child2.low1;
        low3 = child3.low1;

        child1.thisChildNumber = 1;
        child2.thisChildNumber = 2;
        child3.thisChildNumber = 3;
    }
}

public int NewChildLeaf(int element)
{
    Node leaf = new Node();

    leaf.isLeaf = true;
    leaf.isRoot = false;
    leaf.childrenNumber = 0;
    leaf.element = element;
    leaf.low1 = element;
    leaf.parent = this;
    leaf.style = config.getInteger("leaf-default-style");

    return AddChild(leaf);
}

public int NewChildInterior(Node child1)
{
    Node interior = new Node();

    interior.child1 = child1;
    interior.child1.parent = interior;
    interior.child1.thisChildNumber = 1;
    interior.isLeaf = false;
    interior.isRoot = false;
    interior.childrenNumber = 1;
    interior.low1 = child1.low1;
    interior.parent = this;
    interior.style = config.getInteger("interior-default-style");

    return AddChild(interior);
}

public int NewChildInterior(Node child1, Node child2)
{
    Node interior = new Node();

    interior.child1 = child1;
    interior.child2 = child2;
    interior.child1.parent = interior;
    interior.child2.parent = interior;
    interior.child1.thisChildNumber = 1;
    interior.child2.thisChildNumber = 2;
    interior.isLeaf = false;
    interior.isRoot = false;
    interior.childrenNumber = 2;
```

```
            interior.low1 = child1.low1;
            interior.low2 = child2.low1;
            interior.parent = this;
            interior.style = config.getInteger("interior-default-style");

            return AddChild(interior);
        }

        public int NewChildInterior(Node child1, Node child2, Node child3)
        {
            Node interior = new Node();

            interior.child1 = child1;
            interior.child2 = child2;
            interior.child3 = child3;
            interior.child1.parent = interior;
            interior.child2.parent = interior;
            interior.child3.parent = interior;
            interior.child1.thisChildNumber = 1;
            interior.child2.thisChildNumber = 2;
            interior.child3.thisChildNumber = 3;
            interior.isLeaf = false;
            interior.isRoot = false;
            interior.childrenNumber = 3;
            interior.low1 = child1.low1;
            interior.low2 = child2.low1;
            interior.low3 = child3.low1;
            interior.parent = this;
            interior.style = config.getInteger("interior-default-style");

            return AddChild(interior);
        }

        /**
         * Finds leafs number in subtree
         *
         * @param leafsNumber - number of leafs on this moment of searching
         * @return number of leafs on this moment of searching
         */
        public int FindLeafsNumber(int leafsNumber)
        {
            if (isLeaf)
            {
                leafsNumber++;
            }
            else
            {
                if (childrenNumber == 1)
                {
                    leafsNumber = child1.FindLeafsNumber(leafsNumber);
                }
                if (childrenNumber == 2)
                {
                    leafsNumber = child1.FindLeafsNumber(leafsNumber);
                    leafsNumber = child2.FindLeafsNumber(leafsNumber);
                }
                if (childrenNumber == 3)
                {
                    leafsNumber = child1.FindLeafsNumber(leafsNumber);
                    leafsNumber = child2.FindLeafsNumber(leafsNumber);
                    leafsNumber = child3.FindLeafsNumber(leafsNumber);
                }
                if (childrenNumber == 4)
                {
                    leafsNumber = child1.FindLeafsNumber(leafsNumber);
                    leafsNumber = child2.FindLeafsNumber(leafsNumber);
                    leafsNumber = child3.FindLeafsNumber(leafsNumber);
                    leafsNumber = child4.FindLeafsNumber(leafsNumber);
                }
            }

            return leafsNumber;
        }
    }
```

Tree23Applet.java

```
package ru.ifmo.vizi.Tree23;


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

import java.awt.*;

//import ru.ifmo.vizi.base.widgets.Rect;

import ru.ifmo.vizi.base.ui.*;
import ru.ifmo.vizi.base.Base;

import java.awt.*;
import java.awt.event.*;
//import java.lang.*;

//import java.io.StreamTokenizer;
//import java.io.StringReader;
import java.io.IOException;
import java.io.StreamTokenizer;
import java.io.StringReader;


/**
 * BSM applet
 */
public final class Tree23Applet extends Base implements MouseListener,
        MouseMotionListener,
        KeyListener
{
    /**
     * BSM automata instance
     */
    private final Tree23 auto;

    /**
     * BSM automata data
     */
    private final Tree23.Data data;

    /**
     * Fields for element number input
     */
    private HintedTextField textFieldElement;

    /**
     * Button "Find"
     */
    HintedButton buttonFind;

    /**
     * Button "Remove"
     */
    HintedButton buttonRemove;

    /**
     * Button "Add"
     */
    HintedButton buttonAdd;

    /**
     * Height and width of window
     */
    private int winHeight, winWidth;
```

```
    /**
     * Box for saving and loading
     */
//    SaveLoadBox save_load_dialog;
    SaveLoadDialog saveDialog;

    /**
     * Save
     */
    String saveOfTree;

    private final Frame forefather;
    private AutoControlsPane autoControlsPane;

    /**
     * Default constructor
     */
    public Tree23Applet(VisualizerParameters parameters)
    {
        super(parameters);
        this.forefather = parameters.getForefather();
        auto = new Tree23(locale);
        data = auto.d;
        data.visualizer = this;

        createInterface(auto);

        clientPane.addMouseListener(this);
        clientPane.addMouseMotionListener(this);

        auto.d.tree = new Tree();

        auto.d.tree.Load(config.getParameter("example-of-tree"), config);
        auto.d.tree.InitAndAddToClientPane(clientPane, config);
        auto.d.tree.SetDefaultStyles();

        saveOfTree = auto.d.tree.Save();

        textFieldElement.addKeyListener(this);
    }

    /**
     * This method creates panel with applet's controls
     *
     * @return created pane
     */
    public Component createControlsPane()
    {
        Panel panel = new Panel(new BorderLayout());

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

        Panel addRemoveFindPanel = new Panel();

        addRemoveFindPanel.add(new Label(config.getParameter("label-element"),
                Label.RIGHT));

        textFieldElement = new HintedTextField(config, "text-field-element");
        textFieldElement.setColumns(config.getInteger("text-field-element-rows"));
        addRemoveFindPanel.add(textFieldElement);

        addRemoveFindPanel.add(buttonAdd = new HintedButton(config, "button-add")
        {
            protected void click()
            {
                AddButtonClick();
            }
        });

        addRemoveFindPanel.add(buttonRemove = new HintedButton(config,
                "button-remove")
```

```
{
    protected void click()
    {
        RemoveButtonClick();
    }
});

addRemoveFindPanel.add(buttonFind = new HintedButton(config,
        "button-find")
{
    protected void click()
    {
        FindButtonClick();
    }
});

panel.add(addRemoveFindPanel, BorderLayout.CENTER);

Panel SaveLoadPanel = new Panel();

SaveLoadPanel.add(new HintedButton(config, "save-load")
{
    protected void click()
    {
        setComment(" ");


        Component forefather = this;
        while (forefather.getParent() != null)
        {
            forefather = forefather.getParent();
        }
        Rectangle b = forefather.getBounds();
        if (!(forefather instanceof Frame))
        {
            forefather = new Frame();
        }

        saveDialog = new SaveLoadDialog(config, (Frame) forefather)
        {
            public boolean load(String txt) throws Exception
            {
                try
                {
                    Load(txt);
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
                return false;
            }
        };

        saveDialog.setSize(config.getInteger("save-load-width"),
                config.getInteger("save-load-height"));

        Dimension s = saveDialog.getSize();
        saveDialog.setLocation(b.x + (b.width - s.width) / 2,
                b.y + (b.height - s.height) / 2);

        StringBuffer sb = new StringBuffer();

        if (auto.d.delete || auto.d.insert || auto.d.find)
        {
            sb.append("/* ");
            sb.append(config.getParameter("comment-alg"));
            sb.append(" */\n");
        }
        if (auto.d.find)
        {
            sb.append(config.getParameter("find")).append('\n');
            sb.append("/* ");
            sb.append(config.getParameter("comment-element"));
```

```
                sb.append(" */\n");
                sb.append(auto.d.serchingElement).append('\n');
            }
            if (auto.d.insert)
            {
                sb.append(config.getParameter("insert")).append('\n');
                sb.append("/* ");
                sb.append(config.getParameter("comment-element"));
                sb.append(" */\n");
                sb.append(auto.d.insertingElement).append('\n');
            }
            if (auto.d.delete)
            {
                sb.append(config.getParameter("delete")).append('\n');
                sb.append("/* ");
                sb.append(config.getParameter("comment-element"));
                sb.append(" */\n");
                sb.append(auto.d.deletingElement).append('\n');
            }

            if (auto.d.delete || auto.d.insert || auto.d.find)
            {
                sb.append("/* ");
                sb.append(config.getParameter("comment-step"));
                sb.append(" */\n");
                sb.append(auto.getStep()).append('\n');
            }

            sb.append("/* ");
            sb.append(config.getParameter("comment-tree"));
            sb.append(" */\n");

            sb.append(saveOfTree);

            saveDialog.show(sb.toString());
        }
    });

    if (config.getBoolean("save-load-mode"))
    {
        panel.add(SaveLoadPanel, BorderLayout.SOUTH);
    }

    return panel;
}

void FindButtonClick()
{
    if (auto.d.insertEnd || auto.d.deleteEnd)
    {
        auto.d.insertEnd = false;
        auto.d.deleteEnd = false;
        saveOfTree = auto.d.tree.Save();
    }
    else
    {
        auto.d.tree.RemoveFromClientPane();
        auto.d.tree.Load(saveOfTree, config);
        auto.d.tree.InitAndAddToClientPane(clientPane, config);
        auto.d.tree.SetDefaultStyles();
    }

    setComment(" ");

    AlgToStart();

    StreamTokenizer tokenizer =
            new StreamTokenizer(
                    new StringReader(textFieldElement.getText()));

    Object[] args0 = new Object[]{
        new Integer(config.getInteger("min-element")),
        new Integer(config.getInteger("max-element"))};
```

```
    try
    {
        tokenizer.nextToken();
        if (tokenizer.ttype == tokenizer.TT_NUMBER)
        {
            auto.d.serchingElement = (int) tokenizer.nval;
            if (!(auto.d.serchingElement <=
                  config.getInteger("max-element") &&
                  auto.d.serchingElement >=
                  config.getInteger("min-element")))
            {
                setComment(java.text.MessageFormat.format(
                        config.getParameter("not-number-error"), args0));
                return;
            }
        }
        else
        {
            setComment(java.text.MessageFormat.format(
                    config.getParameter("not-number-error"), args0));
            return;
        }

        tokenizer.nextToken();
        if (tokenizer.ttype != tokenizer.TT_EOF)
        {
            setComment(java.text.MessageFormat.format(
                    config.getParameter("not-number-error"), args0));
            return;
        }
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }

    auto.d.find = true;
    auto.d.delete = false;
    auto.d.insert = false;

    buttonFind.setBackground(
            new Color(config.getInteger("mark-out-button-color-r"),
            config.getInteger("mark-out-button-color-g"),
            config.getInteger("mark-out-button-color-b")));
    buttonAdd.setBackground(
            autoControlsPane.getComponent(0).getBackground());
    buttonRemove.setBackground(
            autoControlsPane.getComponent(0).getBackground());
}

void RemoveButtonClick()
{
    if (auto.d.insertEnd || auto.d.deleteEnd)
    {
        auto.d.insertEnd = false;
        auto.d.deleteEnd = false;
        saveOfTree = auto.d.tree.Save();
    }
    else
    {
        auto.d.tree.RemoveFromClientPane();
        auto.d.tree.Load(saveOfTree, config);
        auto.d.tree.InitAndAddToClientPane(clientPane, config);
        auto.d.tree.SetDefaultStyles();
    }

    setComment(" ");

    AlgToStart();

    StreamTokenizer tokenizer =
            new StreamTokenizer(new StringReader(textFieldElement.getText()));

    Object[] args0 = new Object[]{
```

```
            new Integer(config.getInteger("min-element")),
            new Integer(config.getInteger("max-element"))};

        try
        {
            tokenizer.nextToken();
            if (tokenizer.ttype == tokenizer.TT_NUMBER)
            {
                auto.d.deletingElement = (int) tokenizer.nval;
                if (!(auto.d.deletingElement <=
                      config.getInteger("max-element") &&
                      auto.d.deletingElement >=
                      config.getInteger("min-element")))
                {
                    setComment(java.text.MessageFormat.format(
                            config.getParameter("not-number-error"), args0));
                    return;
                }
            }
            else
            {
                setComment(java.text.MessageFormat.format(
                        config.getParameter("not-number-error"), args0));
                return;
            }

            tokenizer.nextToken();
            if (tokenizer.ttype != tokenizer.TT_EOF)
            {
                setComment(java.text.MessageFormat.format(
                        config.getParameter("not-number-error"), args0));
                return;
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }


        auto.d.find = false;
        auto.d.delete = true;
        auto.d.insert = false;

        buttonRemove.setBackground(
                new Color(config.getInteger("mark-out-button-color-r"),
                config.getInteger("mark-out-button-color-g"),
                config.getInteger("mark-out-button-color-b")));
        buttonAdd.setBackground(autoControlsPane.getComponent(0).getBackground());
        buttonFind.setBackground(autoControlsPane.getComponent(0).getBackground());
    }

    void AddButtonClick()
    {
        if (auto.d.insertEnd || auto.d.deleteEnd)
        {
            auto.d.insertEnd = false;
            auto.d.deleteEnd = false;
            saveOfTree = auto.d.tree.Save();
        }
        else
        {
            auto.d.tree.RemoveFromClientPane();
            auto.d.tree.Load(saveOfTree, config);
            auto.d.tree.InitAndAddToClientPane(clientPane, config);
            auto.d.tree.SetDefaultStyles();
        }

        setComment(" ");

        AlgToStart();

        StreamTokenizer tokenizer =
                new StreamTokenizer(new StringReader(textFieldElement.getText()));
```

```
    Object[] args0 = new Object[]{new Integer(config.getInteger("min-element")),
                                  new Integer(config.getInteger("max-element"))};
    Object[] args1 = new Object[]{
        new Integer(config.getInteger("min-element-number")),
        new Integer(config.getInteger("max-element-number"))};


    try
    {
        tokenizer.nextToken();
        if (tokenizer.ttype == tokenizer.TT_NUMBER)
        {
            auto.d.insertingElement = (int) tokenizer.nval;
            if (!(auto.d.insertingElement <=
                  config.getInteger("max-element") &&
                  auto.d.insertingElement >=
                  config.getInteger("min-element")))
            {
                setComment(java.text.MessageFormat.format(
                        config.getParameter("not-number-error"), args0));
                return;
            }
            if (auto.d.tree.FindLeafsNumber() ==
                config.getInteger("max-element-number"))
            {
                setComment(java.text.MessageFormat.format(
                        config.getParameter("max-point-number-error"), args1));
                return;
            }
        }
        else
        {
            setComment(java.text.MessageFormat.format(
                    config.getParameter("not-number-error"), args0));
            return;
        }

        tokenizer.nextToken();
        if (tokenizer.ttype != tokenizer.TT_EOF)
        {
            setComment(java.text.MessageFormat.format(
                    config.getParameter("not-number-error"), args0));
            return;
        }
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }


    auto.d.find = false;
    auto.d.delete = false;
    auto.d.insert = true;

    buttonAdd.setBackground(
            new Color(config.getInteger("mark-out-button-color-r"),
            config.getInteger("mark-out-button-color-g"),
            config.getInteger("mark-out-button-color-b")));
    buttonFind.setBackground(autoControlsPane.getComponent(0).getBackground());
    buttonRemove.setBackground(autoControlsPane.getComponent(0).getBackground());
}


/**
 * Loading data fron SaveLoadBox
 *
 * @param toLoad string to load
 */
public void Load(final String toLoad) throws IOException
{
    int step = 0, element = 0;
    String alg = "";
```

```
SmartTokenizer tokenizer = new SmartTokenizer(toLoad, config);

tokenizer.nextToken();
if (tokenizer.ttype == tokenizer.TT_WORD && tokenizer.sval != "(")
{
    alg = tokenizer.sval;

    if (!alg.equals(config.getParameter("find")) &&
        !alg.equals(config.getParameter("insert")) &&
        !alg.equals(config.getParameter("delete")))
    {
        Object[] args = new Object[]{new String(config.getParameter("insert")),
                                     new String(config.getParameter("delete")),
                                     new String(config.getParameter("find"))};
        saveDialog.setComment(
                java.text.MessageFormat.format(config.getParameter("load-info-6"),
                args));
        return;
    }

    tokenizer.nextToken();
    if (tokenizer.ttype == tokenizer.TT_NUMBER)
    {
        element = (int) tokenizer.nval;
    }
    else
    {
        saveDialog.setComment(config.getParameter("load-info-7"));
        return;
    }

    tokenizer.nextToken();
    if (tokenizer.ttype == tokenizer.TT_NUMBER)
    {
        step = (int) tokenizer.nval;
    }
    else
    {
        saveDialog.setComment(config.getParameter("load-info-8"));
        return;
    }
}
else
{
    tokenizer = new SmartTokenizer(toLoad, config);
}

auto.d.tree.RemoveFromClientPane();

int result = auto.d.tree.Load(tokenizer);
if (result <= 0)
{
    auto.d.tree.Load(saveOfTree, config);
    auto.d.tree.InitAndAddToClientPane(clientPane, config);
    auto.d.tree.SetDefaultStyles();
    saveOfTree = auto.d.tree.Save();
    auto.d.tree.Draw(winWidth, winHeight);

    if (result == 0)
    {
        saveDialog.setComment(config.getParameter("load-info-0"));
    }
    if (result == -1)
    {
        saveDialog.setComment(config.getParameter("load-info-1"));
    }
    if (result == -2)
    {
        saveDialog.setComment(config.getParameter("load-info-2"));
    }
    if (result == -3)
    {
        saveDialog.setComment(config.getParameter("load-info-3"));
```

```
    }
    if (result == -4)
    {
        saveDialog.setComment(config.getParameter("load-info-4"));
    }
    if (result == -5)
    {
        saveDialog.setComment(config.getParameter("load-info-5"));
    }
    return;
}
Node tempNode = auto.d.tree.rootNode1;
while (!tempNode.isLeaf)
{
    tempNode = tempNode.child1;
}
if (tempNode.element < config.getInteger("min-element"))
{
    auto.d.tree.Load(saveOfTree, config);
    auto.d.tree.InitAndAddToClientPane(clientPane, config);
    auto.d.tree.SetDefaultStyles();
    saveOfTree = auto.d.tree.Save();
    auto.d.tree.Draw(winWidth, winHeight);

    Object[] args = new Object[]{new Integer(config.getInteger("min-element")),
                                 new Integer(config.getInteger("max-element"))};
    saveDialog.setComment(java.text.MessageFormat.format(
            config.getParameter("load-info-10"),
            args));
    return;
}

tempNode = auto.d.tree.rootNode1;
while (!tempNode.isLeaf)
{
    if (tempNode.childrenNumber == 2)
    {
        tempNode = tempNode.child2;
    }
    if (tempNode.childrenNumber == 3)
    {
        tempNode = tempNode.child3;
    }
}
if (tempNode.element > config.getInteger("max-element"))
{
    auto.d.tree.Load(saveOfTree, config);
    auto.d.tree.InitAndAddToClientPane(clientPane, config);
    auto.d.tree.SetDefaultStyles();
    saveOfTree = auto.d.tree.Save();
    auto.d.tree.Draw(winWidth, winHeight);

    Object[] args = new Object[]{new Integer(config.getInteger("min-element")),
                                 new Integer(config.getInteger("max-element"))};
    saveDialog.setComment(java.text.MessageFormat.format(
            config.getParameter("load-info-10"),
            args));
    return;
}

auto.d.tree.InitAndAddToClientPane(clientPane, config);
auto.d.tree.SetDefaultStyles();

if (auto.d.tree.FindLeafsNumber() > config.getInteger("max-element-number"))
{
    auto.d.tree.Load(saveOfTree, config);
    auto.d.tree.InitAndAddToClientPane(clientPane, config);
    auto.d.tree.SetDefaultStyles();
    saveOfTree = auto.d.tree.Save();
    auto.d.tree.Draw(winWidth, winHeight);

    Object[] args = new Object[]{
        new Integer(config.getInteger("min-element-number")),
        new Integer(config.getInteger("max-element-number"))};
```

```
        saveDialog.setComment(java.text.MessageFormat.format(
                config.getParameter("load-info-9"),
                args));
        return;
    }

    auto.d.tree.Draw(winWidth, winHeight);

    saveOfTree = auto.d.tree.Save();

    textFieldElement.setText(new Integer(element).toString());

    if (alg.equals(config.getParameter("find")))
    {
        this.FindButtonClick();
    }
    if (alg.equals(config.getParameter("insert")))
    {
        this.AddButtonClick();
    }
    if (alg.equals(config.getParameter("delete")))
    {
        this.RemoveButtonClick();
    }

    auto.d.tree.RemoveFromClientPane();
    auto.d.tree.InitAndAddToClientPane(clientPane, config);
    auto.d.tree.SetDefaultStyles();
    auto.d.tree.Draw(winWidth, winHeight);

    UpdateScreen();

    saveDialog.dispose();

    while (auto.getStep() != step && step != 0)
    {
        auto.stepForward(0);
        if (auto.isAtEnd())
        {
            break;
        }
    }
}

/**
 * Returns auto to start
 */
protected void AlgToStart()
{
    auto.d.tree.SetDefaultStyles();
    DrawTree();

    auto.toStart();

    UpdateScreen();
}


public void DrawTree()
{
    auto.d.tree.Draw(winWidth, winHeight);
}


/**
 * Updates screen
 */
public void UpdateScreen()
{
    update(true);
}


/**
```

```
 * Resize window
 *
 * @param clientWidth  new width
 * @param clientHeight new height
 */
protected void layoutClientPane(int clientWidth, int clientHeight)
{
    winHeight = clientHeight;
    winWidth = clientWidth;


    auto.d.tree.Draw(winWidth, winHeight);
    UpdateScreen();
}


/**
 * Mouse pressed
 *
 * @param me mouse event
 */
public void mousePressed(MouseEvent me)
{
}

/**
 * Mouse relesed
 *
 * @param me mouse event
 */
public void mouseReleased(MouseEvent me)
{
}

/**
 * Mouse dragged
 *
 * @param me mouse event
 */
public void mouseDragged(MouseEvent me)
{
}

/**
 * Mouse entered
 *
 * @param me mouse event
 */
public void mouseEntered(MouseEvent me)
{
}

/**
 * Mouse exit
 *
 * @param me mouse event
 */
public void mouseExited(MouseEvent me)
{
}

/**
 * Mouse clicked
 *
 * @param me mouse event
 */
public void mouseClicked(MouseEvent me)
{
}

/**
 * Mouse moved
 *
 * @param me mouse event
```

```
     */
    public void mouseMoved(MouseEvent me)
    {
    }

    /**
     * Key typed
     *
     * @param ke key event
     */
    public void keyTyped(KeyEvent ke)
    {
        setComment(" ");

        if (ke.getSource() == textFieldElement)
        {

            buttonAdd.setBackground(
                    autoControlsPane.getComponent(0).getBackground());
            buttonFind.setBackground(
                    autoControlsPane.getComponent(0).getBackground());
            buttonRemove.setBackground(
                    autoControlsPane.getComponent(0).getBackground());

            auto.d.find = false;
            auto.d.delete = false;
            auto.d.insert = false;
        }
    }

    /**
     * Key pressed
     *
     * @param ke key event
     */
    public void keyPressed(KeyEvent ke)
    {
    }

    /**
     * Key released
     *
     * @param ke key event
     */
    public void keyReleased(KeyEvent ke)
    {
    }
}
```