компьютер в учебном процессе

УДК 004.5

Пухов Алексей Фёдорович

ИНТЕРАКТИВНЫЕ ФОРМЫ ПРЕДСТАВЛЕНИЯ БАЗОВЫХ ИДЕЙ ДИСКРЕТНОЙ МАТЕМАТИКИ НА ПРИМЕРЕ ПРОЕКТА «ШКОЛА КИО»

Аннотация

В статье описывается программная платформа для создания манипуляторов по проекту «Школа КИО», который связан с организацией среды для интерактивного знакомства школьников с основными алгоритмами дискретной математики. Рассматриваются основные принципы работы манипуляторов, их алгоритмическое представление и архитектура всей системы в целом. Описаны полученные результаты, пути внедрения и перспективы развития.

Ключевые слова: Интерактивные формы представления знаний, дискретная математика, дистанционное обучение, манипулятор, программная платформа.

ВВЕДЕНИЕ

Проект «Школа КИО» (КИО является аббревиатурой от «Конструируй, исследуй, оптимизируй») связан с разработкой интерактивных форм представления научных знаний с целью популяризации идей и методов дискретной математики в рамках работы дистанционной Школы современного программирования для школьников.

В рамках проекта «Школа КИО» разработаны новые формы интерактивного представления алгоритмов дискретной математики для учебных целей, основанные на развитии идей, сформулированных в работах [1–3; 4]. Эти работы посвящены разработке наборов визуализаторов алгоритмов дискретной математики. Использование визуализаторов при обучении дискретной математике получило широкое распространение [5], что способствовало дальнейшему развитию технологий их разработки и созданию универсальных компонент для визуализации алгоритмов [6–7].

В данной работе представлен другой подход к разработке технологии средств поддержки обучения элементам дискретной математики, основанный на расширении их функциональности и добавлении обратной связи с пользователем. Используемые в проекте модули – манипуляторы – отличаются от визуализаторов тем, что они не только знакомят ученика с алгоритмами, позволяя просматривать их действия по шагам, но также дают возможность выполнить алгоритм самостоятельно и проверить правильность своих действий. Кроме того, особое внимание уделяется графическому оформлению манипуляторов, представляющему основные особенности каждого алгоритма.

В основе создания манипуляторов лежат инструментальный подход [8], а так-

же множественность представления знания [9] в рамках отдельно взятого алгоритма.

На рис.1 представлен интерфейс манипулятора, созданного в рамках проекта «Школа КИО» на основе описанной в статье технологии.

Рассмотрим подробнее основные принципы, которые положены в основу создания и работы манипулятора.

1. Графическая метафора. Специфика проекта заключается в том, чтобы заинтересовать ученика и помочь ему разобраться в представленных алгоритмах, поэтому визуальное оформление каждого манипулятора должно быть уникальным и занимательным. Графическая метафора «обыгрывает» отличительные особенности алгоритма.

2. Каждый манипулятор поддерживает три режима: демонстрация, тренажер, контроль. В режиме демонстрации ученик знакомится с моделью, осуществляя переходы по шагам алгоритма, при этом система сопровождает действия комментариями. В режиме тренажера ученик самостоятельно выполняет шаги алгоритма, а система блокирует ввод неправильных данных или неправильные действия, комментирует сделанные ошибки. Режим контроля отличается от режима тренажера тем, что система не блокирует и не сообщает о неправильных действиях ученика, но фиксирует их. Результат сообщается ученику только после окончания работы с алгоритмом.

3. Возможность переключений между режимами. При этом в определенных случаях исходные данные алгоритма сохраняются.

4. Автоматическая генерация новых исходных данных для алгоритма.

5. Возможность пошагового просмотра и прохождения алгоритма как вперед, так и назад.

В статье рассматриваются архитектурные решения, принятые при проектировании программной платформы, в рамках которой создаются и работают манипуляторы. Внимание уделено формализации понятия шага в рамках системы, механизму перехода между шагами, механизму переключения между режимами.

УЧАСТНИКИ ПРОЦЕССА РАЗРАБОТКИ МАНИПУЛЯТОРОВ

Для разработки манипуляторов удобно использовать программную платформу, которая определяет универсальный программный и пользовательский интерфейс, стандартизирует процесс разработки манипуляторов. Программный интерфейс формализует общие понятия, использующиеся в работе всех манипуляторов, такие как шаги алгоритма, режимы работы манипуляторов, переключения между шагами и режимами, анализ ошибок пользователя. Пользовательский интерфейс содержит элементы управления, общие для всех манипуляторов. Процесс разработки манипуляторов стандартизируется, в

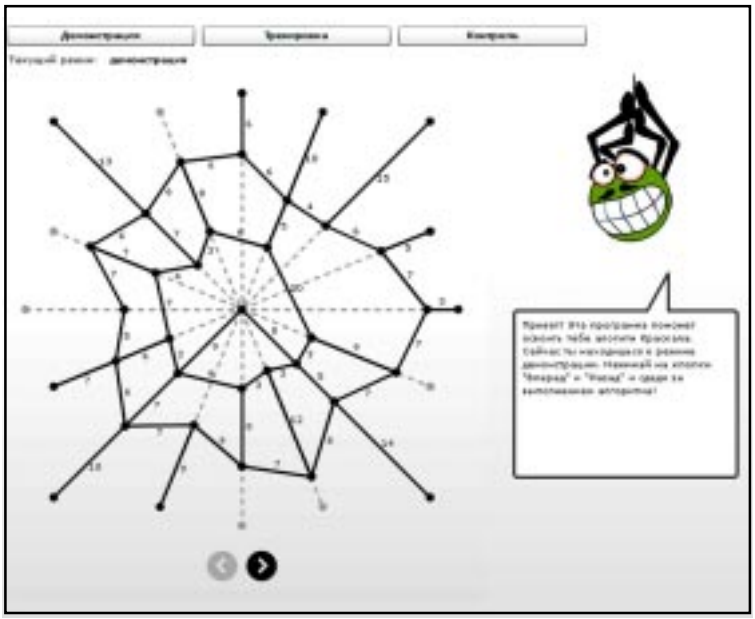

Рис. 1. Манипулятор алгоритма Краскала для нахождения минимального остовного дерева
(автор: *Михайлов Денис*, магистрант СПбГЭТУ «ЛЭТИ»)

частности, способом взаимодействия программистов и дизайнеров.

Таким образом, программисту каждого конкретного манипулятора достаточно сконцентрироваться только на сути выбранного им алгоритма и не реализовывать логику работы манипулятора. Так, например, при правильной реализации алгоритма он автоматически заработает во всех режимах манипулятора. Кроме того, стандартизация процесса программирования позволит привлечь более широкий круг специалистов к участию в проекте.

Определим основных участников технического процесса разработки манипуляторов:

- *Архитектор-программист.* Занимается созданием платформы. Результатом его работы является стандартная библиотека классов, а также шаблон, расширяя который, можно создавать собственные манипуляторы.
- *Программист и автор конкретного манипулятора.* Его работа состоит в том, чтобы реализовать логику работы выбранного алгоритма в рамках предлагаемого платформой программного интерфейса. Как автор манипулятора он должен работать в контакте с дизайнером, в частности, найти подходящую метафору к алгоритму.
- *Дизайнер.* Его работа состоит в том, чтобы помочь программисту подобрать метафору, а также графически оформить манипулятор в соответствии с выбранной метафорой.

ВЫБОР ТЕХНОЛОГИИ РЕАЛИЗАЦИИ

Для разработки платформы создания манипуляторов были рассмотрены технологии Adobe Flash, Java, .NET, «чистый» Web (JavaScript). При выборе технологии важно было дать возможность ученикам запускать манипуляторы внутри браузера, то есть избежать процесса установки дополнительных программ, а также обеспечить программистам удобную работу с графикой. Из всех перечисленных технологий пользователю проще всего запускать приложения на языке JavaScript, потому что JavaScript не требует установки никаких дополнительных приложений кроме браузера. Тем не менее, разработчик на JavaScript имеет ограниченный набор инструментов для работы с графикой и анимацией, необходимых для создания красочных манипуляторов. К тому же, разработка на JavaScript сопряжена с трудностями создания приложений, одинаково работающих в разных типах и версиях браузеров. Второй по доступности ученикам технологией является Flash, он требует установки браузерных плагинов, но при этом они уже установлены на большинстве современных компьютеров. Java-апплеты недостаточно распространены, и существует большая вероятность, что они не запустятся у конечного пользователя. Недостатком .NET является то, что устойчивую работу приложений можно гарантировать только под операционной системой Windows.

Архитектура платформы создания манипуляторов основана на методологии программирования MVC (Model-View-Controller, Модель-Представление-Контроллер). Суть методологии состоит в том, что программа разбивается на три компонента: модель, реализующая удобный доступ к используемым в программе данным, представление, отображающее данные, и контроллер, связывающий модель и представление и управляющий происходящими в системе событиями, в первую очередь, обрабатывая действия пользователя. Использование методологии MVC обеспечивает требуемую стандартизацию процесса разработки манипуляторов, однако в значительной степени увеличивает размер и сложность кода манипуляторов. Этот недостаток компенсируется тем, что, во-первых, мы имеем возможность значительного переиспользования кода. В частности, оказывается, что практически полностью переиспользуется контроллер и частично переиспользуется представление. Во-вторых, именно из-за своего четкого разделения на компоненты методология MVC дает гибкие возможности редактирования графического дизайна манипулятора.

Использующийся в технологии Flash язык программирования ActionScript является полноценным объектно-ориентированным языком программирования и обладает всеми возможностями для реализации проекта «Школа КИО» с помощью методологии MVC. Таким образом, работа программиста и дизайнера может происходить следующим образом: программист реализует конкретный алгоритм, используя в качестве основных объектов оформления графические примитивы (квадратики, кружочки, линии и пр.) и собственные схематичные изображения, а параллельно с процессом разработки дизайнер заменяет нарисованные программистом изображения привлекательными картинками. Однако у технологии Flash есть несколько неудобств. Во-первых, формат Adobe Flash является закрытым. Свободно можно только компилировать код, написанный на ActionScript, и проигрывать полученный результат. Создавать изображения с помощью средств Adobe Flash можно только с помощью платного программного обеспечения.

ТЕХНИЧЕСКАЯ РЕАЛИЗАЦИЯ: АЛГОРИТМИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ ВОЗМОЖНОСТЕЙ МАНИПУЛЯТОРА

РЕАЛИЗАЦИЯ ШАГОВ

Как уже упоминалось, важной особенностью любого алгоритма является шаг, поэтому разбор технической реализации манипуляторов мы начнем именно с него.

Шаг – это состояние алгоритма на определенном этапе исполнения. Перечислим возможные виды шагов:

- *Однократный шаг.* Содержит единичное действие, а точнее, все действия можно выполнить одновременно,
- *Итеративный шаг.* Состоит из однотипных действий, которые могут повторяться несколько раз.

В качестве примера рассмотрим шаги алгоритма Хаффмана для поиска оптимального префиксного кода (напомним, что на входе мы имеем последовательность символов):

1) подсчет статистики для каждого символа,

2) «склеивание» двух наиболее редко встречающихся символов в один составной символ до тех пор, пока не останется ровно два символа,

3) присваивание одному из получившихся символов кода 0, а второму – 1,

4) «расклеивание» символов в порядке, обратном «склеиванию», и приписывание к кодовой последовательности каждого из двух «расклеенных» символов либо 0, либо 1 до тех пор, пока не останутся только единичные символы (то есть исходные символы),

5) запись исходной последовательности символов в закодированной форме согласно кодам, полученным в конце предыдущего шага.

Нетрудно заметить, что шаги 1, 3 и 5 являются однократными, так как, например, несмотря на различные исходные данные, подсчет статистики для всех символов можно произвести за одно действие. Шаги 2 и 4 являются итеративными, так как состоят из однотипных действий, связанных со «склеиванием» и «расклеиванием» символов.

Естественно, что деление шагов алгоритма на однократные и итеративные является условным, так как при различных подходах к реализации одного и того же алгоритма одни и те же шаги могут оказаться как итеративными, так и однократными. Именно поэтому система реализуется таким образом, чтобы было неважно, каким по характеру является каждый шаг алгоритма в выбранной реализации.

В нашей реализации шаг имеет часть, относящуюся к модели, и часть, относящуюся к представлению. Таким образом, в системе выделено два интерфейса, описывающих один шаг. Рассмотрим методы, которые реализуют шаг внутри модели (листинг 1) (здесь и далее код приводится на языке ActionScript).

Поясним эти методы:

`isFirst()`, `isLast()` – методы возвращают значения, является ли шаг первым или последним;

Листинг 1

```
public interface IStep
{
     function isFirst():Boolean;
     function isLast():Boolean;
     function next():IStep;
     function back():IStep;
     function checkInput():String;
     function oracle():void;
     function getDrawFactoryId():String;
}
```

next(), back() – методы возвращают шаг, который должен следовать или предшествовать текущему шагу. В случае если в зависимости от данных алгоритма переход может осуществляться на разные шаги, условия перехода также определяются здесь. Это актуально, например, в ситуации, когда мы хотим перейти на следующий шаг или остаться на текущем – тем самым мы имитируем как однократный, так и итеративный шаг. Таким образом, методы **next()** и **back()** исключительно задают последовательность шагов алгоритма;

checkInput() – метод возвращает результат проверки на данном шаге проделанных действий и введенных данных. Возвращает значения **null**, если проверка прошла успешно, и сообщение об ошибке – в противном случае. В режиме тренажера эти сообщения сразу отображаются пользователю, чтобы он мог исправить ошибку. В режиме контроля сообщения не отображаются, но используются для определения шага, на котором произошла первая ошибка. Если реализовать метод так, что он работает, даже если на предыдущих шагах ошибка уже произошла, то манипулятор имеет возможность оценить общую правильность действий. Тем самым, если пользователь ошибся в самом начале, но все остальные шаги были выполнены формально правильно, манипулятор точно укажет, что ошибка произошла только в самом начале;

oracle() – метод ничего не возвращает. Моделирует действия пользователя с элементами на экране, но выполняет их заведомо правильно. Этот метод используется для режима демонстрации при переходе на следующий шаг.

getDrawFactoryId() – метод возвращает идентификатор данного шага, который используется для поиска подходящего набора методов для его отображения.

Рассмотрим методы, которые реализуют шаг внутри представления (листинг 2).

Листинг 2

```
public interface IStepView
{
     function drawEditor():void;
     function drawRenderer():void;
     function drawHidden():void;
}
```

Поясним эти методы:

drawEditor() – отображает данный шаг в режиме редактирования – то есть в режиме для непосредственной работы пользователя с шагом;

drawRenderer() – отображает данный шаг в упрощенном виде. Служит для сохранения на экране информации введенной пользователем на текущем шаге (возможно, в переработанном или упрощенном виде) при переходе не следующий шаг. То есть на экране фиксируется та информация, которую пользователь получил на текущем шаге, если она может пригодиться на следующем;

drawHidden() – скрывает данный шаг. Используется при переходе к предыдущему шагу.

Чтобы связать данный шаг и его представление используется реестр, регистрирующий соответствие идентификаторов шагов и реализаций интерфейса **IStepView**.

РЕАЛИЗАЦИЯ ПЕРЕХОДОВ МЕЖДУ ШАГАМИ

Описав методы для работы с шагами алгоритма, рассмотрим теперь механизм переходов меж-

ду шагами. Как упоминалось выше, механизмы перехода между шагами алгоритма являются универсальным для всех манипуляторов.

Так как реализации переходов на следующий и предыдущий шаг похожи, рассмотрим здесь только функцию, реализующую переход на следующий шаг (листинг 3).

Первое, что выполняет функция, – это анализ текущего режима.

В случае если это демонстрация, то до перехода в следующий режим, система должна выполнить за пользователя правильные действия, и поэтому вызывается метод **oracle()** у текущего шага (текущий шаг хранится в поле step класса контроллера, в котором реализована функция).

В случае режима тренажера, прежде чем перейти на следующий шаг, система должна проверить правильность выполнения текущего шага (с помощью метода **checkInput()**). В том случае, если была допущена ошибка, необходимо сообщить об этом пользователю и прекратить выполнение функции перехода на следующий шаг – таким образом, будут блокироваться неправильные действия пользователя.

В режиме контроля система также вызовет метод **checkInput()** для проверки правильности выполнения текущего шага, однако не будет ни блокировать действия пользователя, ни сообщать ему об ошибке, а просто сохранит этот результат в массиве ошибок (объект **errorsHistory**). В результате всегда можно определить, на каком шаге пользователь совершил ошибку.

После проверки шага (в режиме тренажера и контроля) или заведомо правильного его выполнения (в режиме демонстрации) будет осуществляться переход на следующий шаг. Для этого у текущего шага выясняется, какой шаг следует за ним (метод **next()**), и по идентификаторам текущего шага и следующего шага (переменные **id** и **newId**) определяется набор методов для графического представления каждого из этих шагов.

Листинг 3

```
function toNextStep() {
     if (regime == _DEMO) {
          step.oracle();
     } else if (regime == _TRAINING) {
          if (step.checkInput() != null) {
<сообщить пользователю результат step.checkInput()>;
return;
}

     } else if (regime == _CHECK) {
          errorsHistory.push(step.checkInput());
     }
     var newStep = step.next();
     var newId = newStep.getDrawFactoryId();
     var id = step.getDrawFactoryId();

     var newFactory = stepViewRegistry.getStepViewFactory(newId);
     var factory = stepViewRegistry.getStepViewFactory(id);

     factory(regime,step).drawRenderer();
     newFactory(regime,newStep).drawEditor();
     step = newStep;

     setEnabledForNavigationButtons();
}
```

Затем текущий шаг рисуется в упрощенном виде (вызов метода **drawRenderer()** представления текущего шага), а следующий шаг – в режиме, при котором пользователь может приступать к работе с ним (вызов метода **drawEditor()** представления следующего шага).

После отрисовки шагов изменится значение поля **step**, в которой запишется новый текущий шаг.

В самом конце вызовется функция **setEnabledForNavigationButtons()**, которая обновит на экране представления для кнопок «Вперед» и «Назад». Эта функция выполняет проверку, является ли шаг, который мы только что отрисовали, последним или первым (анализирует результат работы методов **isFirst()**/**isLast()** нарисованного шага) и отключает соответственно кнопку «Вперед» или «Назад».

РЕАЛИЗАЦИЯ ПЕРЕХОДОВ МЕЖДУ РЕЖИМАМИ

Одной из следующих общих особенностей всех манипуляторов является наличие нескольких режимов, которые необходимо учитывать не только при переходе на следующий и предыдущий шаги, но и при кодировании графических представлений для каждого шага. Ответственность за реализацию адекватного графического отображения шагов алгоритма в зависимости от режима целиком и полностью лежит на программисте, однако правила переходов между этими режимами являются общими для всех манипуляторов. Поэтому ниже рассмотрим функцию, реализующую эти правила (листинг 4).

Функция перехода между режимами имеет один параметр – режим, на который производится переход (переменная **newRegime**). Если попытаться перейти на тот же самый режим, функция завершит исполнение.

Далее, если переход осуществляется с режима контроля, то система должна откатиться к первому шагу, на котором произошла ошибка. Реализация функции связана с анализом массива ошибок (объект **errorsHistory**) и поиском первого непустого значения (значение не **null**). После того как найден шаг, на котором произошла первая ошибка, система моделирует нажатие кнопки «Назад» столько раз, сколько нужно для перехода на требуе-

Листинг 4

```
function setRegime(newRegime) {
    if (newRegime == regime) {
        return;
    }
    if (regime == _CHECK) {
        rollToFirstError();
    }
    if (newRegime == _CHECK) {
        regime = newRegime;
        initController();
    } else {
        var id = step.getDrawFactoryId();
        var factory = stepViewRegistry.getStepViewFactory(id);
        factory(regime, step).drawHidden();
        factory(newRegime, step).drawEditor();
        regime = newRegime;
    }

    setEnabledForNavigationButtons();
}
```

мый шаг. С таким подходом мы можем гарантировать непротиворечивость данных с их внешним представлением в той мере, насколько правильно реализованы методы отображения шагов.

Далее функция проверяет, осуществляется ли переход к режиму контроля, и в этом случае вызывает метод **`initController()`**, который обновляет данные алгоритма (то есть генерирует новое условие), обновляет экран манипулятора, убирая все лишнее, и вызывает метод отображения первого шага алгоритма. Именно этот метод вызывается при первом запуске манипулятора. Основное отличие этих ситуаций в том, что при первом запуске значением поля **`regime`** является **`_DEMO`**, и поэтому модель инициализируется в режиме демонстрации, а в данном случае перед вызовом **`initController()`** полю regime присваивается значение **`_CHECK`**, и поэтому манипулятор инициализируется в режиме контроля.

Если переход осуществляется к любому режиму, отличному от контроля, то система анализирует текущий шаг и по его идентификатору **`(step.getDrawFactoryId())`** запрашивает реализацию интерфейса с набором методов для его графического представления. Далее система нарисует текущий шаг в текущем режиме скрытым и снова нарисует его же для редактирования, но уже в новом режиме. После этого система обновит значение поля regime на новый режим.

В самом конце вызывается функция **`setEnabledForNavigationButtons()`**, уже рассмотренная выше.

Чтобы подытожить данный раздел, следует еще раз отметить, на какие моменты программисту стоит уделить особое внимание при программировании манипуляторов:

1. *Разбиение алгоритма на шаги.* Неправильное разбиение на шаги может усложнить работу программиста по реализации логики переходов между шагами и сделать пользовательский интерфейс неочевидным.

2. *Реализация набора визуальных представлений для отображения шагов.* Особенность здесь заключается в том, что каждое представление должно быть независимым, то есть рисование каждого шага не учитывает того, что, возможно, было нарисовано другими шагами.

ЗАКЛЮЧЕНИЕ

На основе рассмотренной в статье технологии были созданы манипуляторы по алгоритмам дискретного анализа. Опыт их разработки показал эффективность применения описанной в статье технологии производства манипуляторов.

В качестве направлений развития представленной технологии можно отметить внедрение нового режима – режима верификации, основная задача которого будет состоять в том, чтобы дать ученику возможность с помощью инструментария конкретного манипулятора убедиться в истинности своих гипотез. Режим верификации следует рассматривать как новую форму представления доказательства корректности алгоритма.

Работа осуществлена при финансовой поддержке РГНФ в рамках научно-исследовательского проекта РГНФ «Интерактивные формы представления информации в развитии научно-популярного жанра как способа формирования научного мировоззрения у молодёжи (от школьной медиатеки к виртуальному музею занимательной науки)», проект № 08-06-00446а.

Литература

1. *Романовский И.В.* Пакет демонстрационных программ по курсу дискретного анализа «DADemo» // Компьютерные инструменты в образовании. СПб., 2002. № 1. С. 55–61.

2. *Романовский И.В.* Демонстрационные программы: 1. Решето Эратосфена // Компьютерные инструменты в образовании. СПб., 2007. № 1. С. 63–65.

3. *Романовский И.В.* Демонстрационные программы: 2. Суффиксный массив // Компьютерные инструменты в образовании. СПб., 2007. № 2. С. 56–60.

4. *Столяр С.Е., Осипова Т.Г., Крылов И.П., Столяр С.С.* Об инструментальных средствах для курса информатики / Труды II Всероссийской конференции «Компьютеры в образовании». СПб.: 1994.

5. *Казаков М.А., Столяр С.Е.* Визуализаторы алгоритмов как элемент технологии преподавания дискретной математики и программирования / Труды международной научно-методической конференции Телематика-2000 (СПб.: СПбГИТМО (ТУ), 2000). Санкт-Петербург, 2000. С. 189–191.

6. *Корнеев Г.А., Шалыто А.А.* Автоматизированное построение визуализаторов алгоритмов дискретной математики // Компьютерные инструменты в образовании. СПб., 2006. № 5. С. 16–26.

7. *Корнеев Г.А.* Технология разработки визуализаторов алгоритмов: труды II межвузовской конференции молодых ученых (СПб.: СПбГУ ИТМО, 2005). Санкт-Петербург, 2005. С. 18–23.

8. *Пухов А.Ф.* Использование инструментального подхода к популяризации научных знаний // Образовательные технологии и общество, 2010. Том 13. Номер 3. С. 332–346.

9. *Bogdanov M., Pozdnyakov S., Pukhov A.* Multiplicity of the knowledge representation forms as a base of using a computer for the studying of the discrete mathematics // Teaching Mathematics: Retrospective and Perspectives: 9th International conference (Vilnius, 2008). Vilnius Pedagogical University., 2008.

Abstract

The article describes a software framework for the CTE-School project. The objectives of the project consist in creating interactive school tools for distance learning discrete mathematics algorithms. The main principles of all the tools operation as well as their algorithmic representation and the architecture of the whole system are explained. The main actors of the tools development process are listed. The project results, perspectives and the guidelines for its implementation are also described.

Keywords: Interactive forms of knowledge representation, discrete mathematic, computer tool, distance learning, program platform

Пухов Алексей Фёдорович,
аспирант математико-механического факультета СПбГУ,
puhov_alex@mail.ru.