ТЕХНОЛОГИЧЕСКОЕ ОБРАЗОВАНИЕ

Корнеев Георгий Александрович,
Шалыто Анатолий Абрамович

АВТОМАТИЗИРОВАННОЕ ПОСТРОЕНИЕ ВИЗУАЛИЗАТОРОВ АЛГОРИТМОВ ДИСКРЕТНОЙ МАТЕМАТИКИ

ВВЕДЕНИЕ

При изучении дискретной математики [1, 2] важную роль играют визуализаторы алгоритмов, позволяющие в наглядной форме динамически отображать детали их работы. Это открывает возможность использования нового подхода к обучению дискретной математике и программированию [3, 4].

Визуализатор – это программа, в процессе работы которой на экране компьютера динамически демонстрируется применение алгоритма к выбранному набору данных. Визуализаторы позволяют изучать работу алгоритмов, как в автоматическом, так и в пошаговом режиме, аналогичном режиму трассировки программ. При этом трассировка может осуществляться малыми и большими шагами, как в прямом, так и в обратном направлении.

Для построения визуализаторов авторами была предложена технология *Vizi* [5] и программный пакет с тем же названием, позволяющие автоматизировать построение визуализатора за счет генерации логики визуализатора на основе конечных автоматов [6]. Для иллюстрации применения этой технологии достаточно построить визуализатор простого алгоритма, что и сделано в настоящей работе. Однако, рассматриваемая технология применима и для построения визуализаторов сложных алгоритмов, примеры которых приведены на сайте http://is.ifmo.ru в разделе «Визуализаторы».

ТЕХНОЛОГИЯ ПОСТРОЕНИЯ ВИЗУАЛИЗАТОРА

Опишем порядок разработки визуализатора с применением технологии *Vizi*.

1. Постановка задачи и анализ литературы.
2. Создание визуализируемой программы:
   – реализация алгоритма,
   – отладка построенной программы.
3. Разработка концепции визуализации:
   – выделение «интересных» шагов алгоритма,
   – разработка концепции визуального представления,
   – разработка набора комментариев,
   – разработка элементов управления.
4. Построение XML-описания визуализируемой программы:
   – выделение модели данных,
   – преобразование программы,
   – запись XML-описания отдельных процедур,
   – запись XML-описания визуализируемой программы,
   – отладка XML-описания визуализируемой программы,
   – интеграция набора комментариев в XML-описание.

5. Реализация визуального представления.

6. Реализация элементов управления.

7. Интеграция и отладка визуализатора:

– интеграция визуального представления в XML-описание,

– генерация кода по XML-описанию визуализатора,

– отладка визуализатора.

На первом этапе производится постановка задачи и анализ литературы. При этом рассматриваются существующие модификации алгоритма и одна из них выбирается для визуализации.

На следующем этапе создается и отлаживается программа, реализующая выбранную модификацию алгоритма.

На третьем этапе разрабатывается концепция визуализации. При этом сначала выделяются те шаги алгоритма, которые представляют наибольший интерес («интересные» шаги). Затем разрабатывается общая концепция визуального представления, в рамках которой разрабатываются слайды для каждого «интересного» шага. Одновременно с разработкой визуального представления для каждого «интересного» шага пишутся комментарии, поясняющие действия, выполняемые алгоритмом. После этого разрабатываются элементы управления визуализатором, в частности, определяется, какие параметры и в каких пределах сможет регулировать пользователь.

На четвертом этапе производится построение XML-описания визуализируемой программы, созданной на втором этапе. Первоначально из программы выделяется модель данных для того, чтобы визуализатор имел доступ к переменным, используемым при визуализации. Так как в работе [7] было предложено ограничить число типов управляющих конструкций в программе операторами присваивания, ветвления, цикла с предусловием, вызова процедур и блочными операторами, то предварительно программу необходимо преобразовать к такому виду. После этого записываются XML-описания отдельных процедур, которые затем объединяются в XML-описание визуализируемой программы. Затем созданное XML-описание отлаживается при помощи средств, предоставляемых пакетом *Vizi*. Этап завершается добавлением комментариев, разработанных на третьем этапе, к XML-описанию визуализируемой программы.

На пятом и шестом этапах производится реализация визуального представления и элементов управления в соответствии с концепцией, разработанной на третьем этапе.

На заключительном этапе интегрируются результаты четвертого и пятого этапов, и производится отладка визуализатора.

Проиллюстрируем применение этой технологии на простом примере – построении визуализатора алгоритма поиска максимума в массиве натуральных чисел.

1. ПОСТАНОВКА ЗАДАЧИ И АНАЛИЗ ЛИТЕРАТУРЫ

Задан массив из N натуральных чисел. Требуется найти в нем максимальное число.

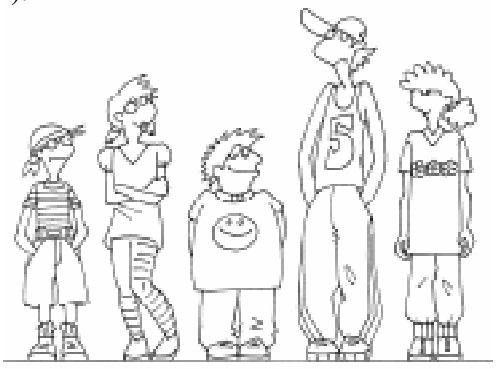
Алгоритм решения этой задачи очень прост и поэтому сразу перейдем к следующему этапу.

2. СОЗДАНИЕ ВИЗУАЛИЗИРУЕМОЙ ПРОГРАММЫ

Задача поиска максимума в массиве решается следующей программой (см. листинг 1).

Листинг 1

```
void main() {
     int max = 0;
     for (int i = 0; i < a.length; i++) {
          if (max < a[i]) {
               max = a[i];
          }
     }
}
```



Здесь **a** – массив, в котором производится поиск максимума, **max** – значение текущего максимума (после **i**-ой итерации – среди первых **i** элементов).

Отметим, что инициализация максимума нулем не приводит к ошибке, так как по условию задачи в массиве содержатся только натуральные числа.

3. РАЗРАБОТКА КОНЦЕПЦИИ ВИЗУАЛИЗАЦИИ

Выделение «интересных» шагов алгоритма. В визуализаторе поиска максимума наиболее интересным являются шаги, осуществляющие ветвление и связанное с ним обновления текущего максимума. Кроме того, должны быть выделены начальное и заключительное состояния.

Специально визуализировать переход к следующему элементу массива вряд ли имеет смысл.

В таблице 1 приведен список «интересных» шагов алгоритма.

Разработка концепции визуального представления. Основными данными в визуализаторе являются элементы массива и значение текущего максимума. Их значения постоянно должны быть представлены на экране. Поэтому для визуального представления будем использовать схему, изображенную на рисунке 1.

Отметим, что индекс текущего элемента в массиве явно не отображается. Вместо этого текущий элемент выделяется цветом.

Для визуализации выделенных «интересных» шагов требуются слайды, представленные в таблице 2.

Разработка набора комментариев. Для каждого из «интересных» состояний разрабатываются комментарии. Отметим, что, так как шаг «*Проверка на обновление максимума*» соответствует оператору ветвления, то для него должны быть разработаны два комментария: для истинного и ложного условий.

Отметим, что комментарии могут содержать параметры, места включения которых задаются следующим образом:

{*номер параметра*}

Значения параметров вычисляются и подставляются в процессе работы визуализатора.

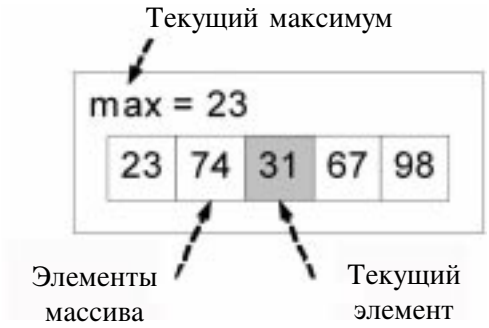


Рисунок 1. Схема визуального представления.

Таблица 1. «Интересные» шаги алгоритма

Шаг	Пояснение
Начальное состояние	Описывается цель алгоритма
Инициализация максимума	Инициализация текущего максимума нулем и соответствующие пояснения
Проверка на обновление максимума	Проверяется необходимость обновить текущий максимум
Обновление текущего максимума	Присваивание текущему максимуму значения текущего элемента
Заключительное состояние	Отображение результатов вычисления

Таблица 2. Слайды для «интересных» шагов

Шаг	Пояснение	Слайд
Начальное состояние	Текущий элемент и максимум не подсвечивается	max=0 23 74 31 67 98
Инициализация максимума	Текущий элемент не подсвечивается	max=0 23 74 31 67 98
Проверка на обновление максимума	Текущий элемент подсвечивается зеленым цветом	max=74 23 74 31 67 98
Обновление текущего максимума	Текущий элемент подсвечивается красным цветом	max=74 23 74 31 67 98
Заключительное состояние	Текущий элемент не подсвечивается	max=98 23 74 31 67 98

Приведем набор комментариев для визуализатора поиска максимума (таблица 3). Выражения, соответствующие параметрам визуализатора, указаны в столбце «Параметры».

Разработка элементов управления. Список элементов управления визуализатора приведен в таблице 4. Отметим, что только два из них не являются стандартными, а все остальные входят в библиотеку *Vizi*.

Элементы управления визуализатором компонуются в области элементов управления, как показано на рисунке 2.

Рисунок 2. Область элементов управления.

4. ПОСТРОЕНИЕ XML-ОПИСАНИЯ ВИЗУАЛИЗИРУЕМОЙ ПРОГРАММЫ

Рассмотрим этапы построения XML-описания.

Выделение модели данных. В визуализируемой программе используются три переменные:

`max` – значение текущего максимума,

`a` – массив, в котором осуществляется поиск,

`i` – индекс текущего элемента массива **`a`**.

Эти переменные должны быть вынесены в модель данных для того, как отмечено выше, визуализатор имел доступ к ним. При этом значения переменных **`max`** и **`a`** будут визуализироваться, а значения переменной **`i`** – нет. Поэтому сделаем переменные **`max`**

Таблица 3. Набор комментариев

Шаг алгоритма	Комментарий	Параметры
Начальное состояние	На экране изображен массив, в котором будет осуществляться поиск максимума	
Инициализация максимума	Инициализируем максимум нулем (так как в массиве только натуральные числа)	
Проверка на обновление максимума (условие истинно)	{0} больше текущего максимума ({1})	`a[i], max`
Проверка на обновление максимума (условие ложно)	{0} не больше текущего максимума ({1})	`a[i], max`
Обновление текущего максимума	Обновляем текущий максимум	
Заключительное состояние	Максимум найден ({0})	`max`

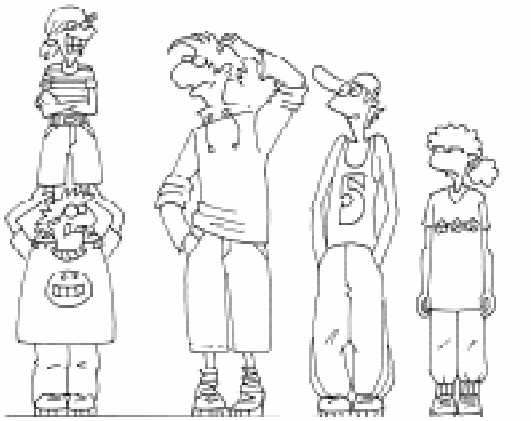
Таблица 4. Элементы управления

Элемент управления	Назначение	Стандартный
<< и >>	Шаг назад и вперед	Да
Рестарт	Начало визуализации	Да
Авто и Стоп	Вход в автоматический режим и выход из него	Да
<< Задержка: 1000 >>	Регулирование задержки между шагами в автоматическом режиме. Задержка измеряется в миллисекундах	Да
?	Вывод информации о визуализаторе	Да
Случайно	Генерация случайного набора данных	Нет
Сохранить/Загрузить	Загрузка/сохранение состояния визуализатора	Да
<< Элементов: 7 >>	Задание количества элементов в массиве	Нет

и **a** глобальными, а переменную **i** – локальной для процедуры поиска максимума. XML-описание переменных имеет следующий вид (см. листинг 2).

Отметим, что для глобальных переменных указаны их начальные значения, которые могут быть использованы при отладке.

В модель данных также вынесем переменную, содержащую ссылку на экземпляр визуализатора. Она будет применяться для формирования визуального представления текущего шага. Опишем ее следующим образом (см. листинг 3).



Листинг 2

```
<variable
     description = "Массив для поиска"
     name        = "a"
     type        = "int[]"
     value       = "new int[]{1, 2, 3, 1, 3, 5, 6}"
/>
<variable
     description = "Текущий максимум"
     name        = "max"
     type        = "int"
     value       = "0"
/>
<variable
     description = "Переменная цикла"
     name        = "i"
     type        = "int"
/>
```

Листинг 3

```
<variable
     description = "Экземпляр апплета"
     name        = "visualizer"
     type        = "FindMaximumVisualizer"
     value       = "null"
/>
```

Листинг 4

```
void main() {
     @max = 0;
     for (@i = 0; @i < @a.length; @i++) {
          if (@max < @a[@i]) {
               @max = @a[@i];
          }
     }
}
```

Для ссылок на переменные, вынесенные в модель данных, применяется нотация вида

@имя переменной

Таким образом, после выделения модели данных программа имеет следующий вид (см. листинг 4).

Преобразование программы. Сначала преобразуем цикл со счетчиком в цикл с предусловием. После преобразования программа выглядит следующим образом (см. листинг 5).

Теперь преобразуем выражение **`@i++`** к виду простого присваивания. В соответствии с семантикой языка *Java* [8], это приводит к оператору

```
@i = @i + 1
```

Для автоматического построения кода, осуществляющего обратную трассировку алгоритма, требуется заменить операторы присваивания на операторы обратимого присваивания [5], имеющие вид @=. В результате, визуализируемая программа будет записана следующим образом (см. листинг 6).

Запись XML-описания отдельных процедур. В XML-описании процедуре соответствует элемент **`auto`**, который содержит описание локальных переменных (элемент **`variable`**) и шаги алгоритма (элементы **`step`** – для присваивания, **`if`** – для ветвления и **`while`** – для цикла с предусловием) [5].

Отметим, что для каждого шага алгоритма должен быть указан его идентификатор (атрибут **`id`**), используемый в дальнейшем при генерации кода и словесное описание (атрибут **`description`**), применяемое при отладке [5].

Листинг 5

```
void main() {
     @max = 0;
     @i = 0;
     while (@i < @a.length)) {
          if (@max < @a[@i]) {
               @max = @a[@i];
          }
          @i++;
     }
}
```

Листинг 6

```
void main() {
     @max @= 0;
     @i @= 0;
     while (@i < @a.length)) {
          if (@max < @a[@i]) {
               @max @= @a[@i];
          }
          @i @= @i + 1;
     }
}
```

Листинг 7

```
<auto id="Main" description="Ищет максимум в массиве">
     <variable description="Переменная цикла» name="i" type="int"/>
     <step id="Initialization" description="Инициализация">
          <action>@max @= 0;</action>
     </step>
     <step id="LoopInit" description="Начало цикла">
          <action>@i @= 0;</action>
     </step>
     <while    id="Loop" description="Цикл" test="@i &lt; @a.length">
          <if id="Cond" description="Условие" test="@max &lt; @a[@i]">
               <then>
                    <step id="newMax" description="Обновление максимума">
                         <action>@max @= @a[@i];</action>
                    </step>
               </then>
          </if>
          <step id="inc" description="Инкремент">
               <action>@i @= @i + 1;</action>
          </step>
     </while>
</auto>
```

Визуализируемая программа содержит одну процедуру **`main`**, XML-описание которой будет иметь вид (см. листинг 7).

Отметим, что **`i`**, как локальная переменная процедуры **`main`**, описана в теле этой процедуры.

Запись XML-описания визуализируемой программы. Для построения полного описания визуализируемой программы добавим к описанию процедуры **`main`** описания переменных модели данных и метода **`toString`** [5], который применяется при отладке (см. листинг 8).

Для получения XML-описания визуализатора в целом, требуется включить описание программы в описание визуализатора (см. листинг 9).

В атрибутах элемента **`visualizer`** описания визуализатора указываются пакет и имена классов визуализатора, а также информация об авторе [5].

Отладка XML-описания визуализируемой программы. Отладка XML-описания визуализатора производится при помощи средств, входящих в пакет *Vizi*.

Листинг 8

```
<algorithm>
     <... описание переменных ...>

     <toString>
          StringBuffer s = new StringBuffer();
          s.append("max = ").append(@max).append("\n");
          s.append("i = ").append(@Main@i).append("\n");
          return s.toString();
     </toString>

     <... XML-описание процедуры main ...>
</algorithm>
```

Листинг 9

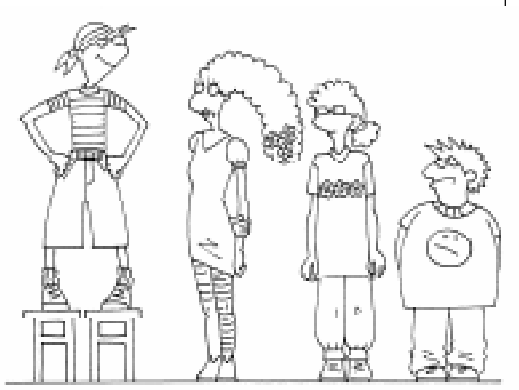
```
<!DOCTYPE visualizer PUBLIC
    "-//IFMO Vizi//Visualizer description"
    "http://ips.ifmo.ru/vizi/dtd/visualizer.dtd"
>
<visualizer
     id="FindMaximum"
     package="ru.ifmo.vizi.find_max"
     main-class="FindMaximumVisualizer"

     name-ru="Поиск максимума"
     author-ru="Георгий Корнеев"
     author-email="kgeorgiy@rain.ifmo.ru"

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

     copyright-ru="Кафедра КТ, СПбГУ ИТМО, 2005"

     preferred-width="400"
     preferred-height="250"
>
     <... XML-описание алгоритма ...>
     <configuration>
     </configuration>
</visualizer>
```



Для этого вызовом *build*-скрипта с параметром debug-source [5] генерируется файл с реализацией алгоритма, заданного XML-описанием (см. листинг 10).

Данный файл позволяет убедиться в том, что XML-описание корректно. Отметим, что сгенерированный код содержит комментарии с информацией, перенесенной из атрибутов description XML-описания.

Проверим также корректность автоматического обращения визуализируемой программы. Для этого вызовом *build*-скрипта с параметром **debug-check** создадим тест и запустим его командой **CheckFindMaximum** из каталога **deploy** [5]. Удостоверимся что, тест прошел успешно:

```
Check 1 steps... OK ()
Check 2 steps... OK ()
...
Check 36 steps... OK ()
Check 37 steps... OK ()
OK
```

Листинг 10

```
/**
  * Ищет максимум в массиве.
  */
private final void Main() {
     // Инициализация
     d.max = 0;
     // Начало цикла
     d.Main_i = 0;
     // Цикл
     while (d.Main_i < d.a.length) {
          // Условие
          if (d.max < d.a[d.Main_i])
{
               // Обновление максимума
               d.max = d.a[d.Main_i];
          }
          // Инкремент
          d.Main_i = d.Main_i + 1;
     }
}
```

Интеграция набора комментариев в XML-описание. Добавим разработанные комментарии к XML-описанию.

Для отображения комментариев в начальном и конечном состояниях к описанию процедуры **main** требуется добавить шаги **start** и **finish** (см. листинг 11).

Здесь и далее полужирным шрифтом выделены внесенные дополнения.

Отметим, что шагам присвоены уровни (атрибут **level**), которые определяют какие шаги будут отображаться пользователю, в различных режимах работы:

-1 – не отображается пользователю,

0 – отображается при выполнении обычных шагов и пропускается при больших шагах (значение по умолчанию),

1 – отображается при выполнении как обычных, так и больших шагов.

5. РЕАЛИЗАЦИЯ ВИЗУАЛЬНОГО ПРЕДСТАВЛЕНИЯ

Разработанное визуальное представление весьма просто (таблица 2) и его отображение может производиться вызовом одного метода

updateArray(<индекс элемента>, <номер цвета>)

При вызове этого метода первым параметром передается индекс выделенного элемента, а вторым – способ его отображения: 0 – без выделения, 1 – зеленый, 2 – красный.

6. РЕАЛИЗАЦИЯ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ

Визуализатор содержит два нестандартных элемента управления: кнопку генерации случайного набора данных и панель задания количество элементов. Первый их

Листинг 11

```
<auto id="Main" description="Ищет максимум в массиве">
     <variable description="Переменная цикла" name="i" type="int"/>
     <start comment-ru="На экране изображен массив, в котором будет
                                осуществляться поиск максимума"/>
     <step id="Initialization" description="Инициализация"
               comment-ru="Инициализируем максимум нулем (так как в
                                массиве только натуральные числа).">
          <action>@max @= 0;</action>
     </step>
     <step id="LoopInit" description="…" level="-1">…</step>
     <while    id="Loop" description="Цикл" test="@i &lt; @a.length"
               level="-1">
          <if id="Cond" description="Условие" test="@max &lt; @a[@i]"
               true-comment-ru="{0} больше текущего максимума ({1})"
               false-comment-ru="{0} не больше текущего максимума ({1})"
               comment-args="new Integer(@a[@i]), new Integer(@max)"
          >
               <then>
                    <step id="newMax" description="Обновление максимума"
                              comment-ru="Обновляем текущий максимум">
                         <action>@max @= @a[@i];</action>
                    </step>
               </then>
          </if>
          <step id="inc" description="…" level="-1">…</step>
               <action>@i @= @i + 1;</action>
          </step>
     </while>
     <finish comment-ru="Максимум найден ({0})"
               comment-args="new Integer(@max)"/>
</auto>
```

них может быть реализован на основе кнопки с подсказкой (HintedButton), а второй на основе панели выбора (AdjustablePanel), входящих в пакет *Vizi*.

Отметим, что в результате воздействия на эти элементы управления изменяются входные данные. Поэтому при их использовании визуализатор должен переходить в начальное состояние.

7. ИНТЕГРАЦИЯ И ОТЛАДКА ВИЗУАЛИЗАТОРА

Интеграция визуального представления в XML-описание. Добавим вызовы, осуществляющие обновление визуального представления, к шагам, имеющим неотрицательный уровень, путем указания их в элементах **`draw`** [5] (см. листинг 12).

Генерация кода по XML-описанию визуализатора. Код визуализатора генерируется вызовом *build*-скрипта с параметром **`all`** [5].

В результате этого по XML-описанию будет сгенерирован код, реализующий, в том числе, и логику визуализатора [7]. Для данного визуализатора будут автоматически построены два взаимодействующих автомата с девятью состояниями в каждом.

На рисунке 3 приведен скриншот построенного визуализатора, запущенного на массиве значений 56 15 59 10 87.

Отладка визуализатора. После построения визуализатора запустим его командой FindMaximum_ru из каталога deploy [5] и проверим его работоспособность. В случае возникновения ошибок они устраняются обычным способом.

ЗАКЛЮЧЕНИЕ

Из рассмотренного примера следует, что построение визуализаторов с применением технологии и пакета *Vizi* автоматизирова-

Рисунок 3. Визуализатор в одном из состояний.

Листинг 12

```
<auto id="Main" description="Ищет максимум в массиве">
     <variable description="Переменная цикла" name="i" type="int"/>
     <start …> <draw>@visualizer.updateArray(0, 0);</draw></start>
     <step …>
          <draw>@visualizer.updateArray(0, 0);</draw>
          <action>@max @= 0;</action>
     </step>
     <step …>…</step>
     <while …>
          <if …>
               <draw>@visualizer.updateArray(@i, 1);</draw>
               <then>
                    <step …>
                         <draw>@visualizer.updateArray(@i, 2);</draw>
                         <action>@max @= @a[@i];</action>
                    </step>
               </then>
          </if>
          <step …>…</step>
     </while>
     <finish …><draw>@visualizer.updateArray(0, 0);</draw></finish>
</auto>
```

но, в отличие от подходов, изложенных в работе [9].

В настоящей работе приведен пример визуализатора простого алгоритма, однако технология *Vizi* позволяет визуализировать и сложные алгоритмы. Например, в работе [10] описывается реализация визуализатора алгоритма нахождения максимального потока в сети методом Диница и Малхотры–Кумара-Махешвари. Реализация этого алгоритма содержит девять процедур, по которым генерируются 18 автоматов, имеющих в сумме более ста состояний.

Визуализатор алгоритма поиска максимума в массиве натуральных чисел и его исходные коды будут опубликованы на сайте http://is.ifmo.ru в разеделе «Статьи».

Литература

1. *Кнут Д.* Искусство программирования. Том 1. Основные алгоритмы. М.: Вильямс, 2000.

2. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы. Построение и анализ. М.: МЦНМО, 1999.

3. *Казаков М. А., Столяр С. Е.* Визуализаторы алгоритмов как элемент технологии преподавания дискретной математики и программирования // Международная научно-методическая конференция «Телематика-2000». СПб.: 2000. С. 189–191.

4. *Корнеев Г. А., Парфенов В. Г., Столяр С. Е., Васильев В. Н.* Визуализаторы алгоритмов как основной инструмент технологии преподавания дискретной математики и программирования // Труды международной научно-методической конференции «Телематика-2001». СПб.: СПбГИТМО (ТУ), 2001. С. 119–120.

5. Vizi home page // http://ctddev.ifmo.ru/vizi

6. *Корнеев Г.А., Шалыто А.А.* Преобразование программ в систему взаимодействующих конечных автоматов. // Труды Второй Всероссийской Научной конференции «Методы и средства обработки информации». М.: МГУ, 2005, с. 385–387.

7. *Корнеев Г. А., Казаков М. 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/

8. *Joy B., Steele G., Gosling J., Bracha G.* Java Language Specification (Second Edition). Addison-Wesley. 2000. http://java.sun.com/docs/books/jls/

9. *Казаков М.А. , Шалыто А.А.* Использование автоматного программирования для реализации визуализаторов // Компьютерные инструменты в образовании. 2004. № 2. С. 19–33. http://is.ifmo.ru/works/art_vis/

10. *Бедный Д.Ю.* Нахождения максимального потока в сети методом Диница и Малхотры–Кумара–Махешвари // Мир ПК-Диск. 2005. № 8. http://is.ifmo.ru/vis/dmkm/

Корнеев Георгий Александрович,
аспирант кафедры «Компьютерные технологии» Санкт-Петербургского государственного университета информационных технологий, механики и оптики (СПбГУ ИТМО), двукратный призер командного чемпионата мира по программированию АСМ,

Шалыто Анатолий Абрамович,
доктор технических наук, профессор, заведующий кафедрой «Технологии программирования» СПбГУ ИТМО.