

Статья опубликована в журнале «Компьютерные инструменты в образовании». 2004. № 2, с. 19 – 33.

**Казakov Матвей Алексеевич**  
**Шалыто Анатолий Абрамович**

## ИСПОЛЬЗОВАНИЕ АВТОМАТНОГО ПРОГРАММИРОВАНИЯ ДЛЯ РЕАЛИЗАЦИИ ВИЗУАЛИЗАТОРОВ

### ВВЕДЕНИЕ

Один из авторов данной работы давно занимается автоматным программированием, а другой – технологиями дистанционного обучения программированию в рамках проекта «Интернет-школа программирования» [1,2] на кафедре «Компьютерные технологии» Санкт-Петербургского государственного университета информационных технологий, механики и оптики (СПбГУ ИТМО). При этом важным направлением при таком обучении программированию является технология разработки визуализаторов [3,4]. Авторы объединило желание создать технологию построения логики визуализаторов, которая до сих пор отсутствовала [5 – 8].

Это произошло следующим образом. В октябре 2000 года на лекциях одним из авторов было рассказано, как по программе строить автомат, и было предложено будущему соавтору выполнить это преобразование на примере алгоритма Дейкстры, предназначенного для нахождения кратчайшего пути в графе. Через некоторое время вместо сравнительно простой и изящной программы, написанной традиционным путем, формально была построена автоматная программа, содержащая оператор *switch* с девятью метками *case*, который являлся телом цикла *do-while*. На вопрос, заданный автором идеи преобразования, о целесообразности такого программирования, был получен следующий ответ: «Программировать так действительно нецелесообразно, а визуализаторы, похоже, следует делать именно так».

*Визуализатор* – это программа, иллюстрирующая выполнение алгоритма при определенных входных данных. Примерами визуализаторов могут служить, например, программа, занимающаяся построением графика функции, либо программа, визуально моделирующая какой-либо физический процесс. Применительно к дискретной математике и программированию, визуализаторы обычно моделируют некоторые алгоритмы, давая возможность обучающемуся при помощи интуитивно понятного интерфейса проходить алгоритм шаг за шагом от начала до конца, а при необходимости, и обратно.

Перечислим отличительные характеристики визуализаторов.

1. Простота использования, определяемая понятностью интерфейса. Поэтому для работы с визуализатором обычно не требуется специальная подготовка.
2. Четкость и простота представления визуализируемого процесса.
3. Компактность визуализаторов, ввиду их реализации в виде Java-апплетов. Это при необходимости упрощает передачу визуализаторов в сети Интернет, что особенно важно при дистанционном обучении.

Визуализаторы в рассматриваемой области решают следующие задачи, возникающие в процессе обучения.

1. Графическое и словесное разъяснение действий алгоритма на конкретных наборах входных данных. При этом понимание алгоритма не обязательно, поскольку именно визуализатор должен объяснить действие алгоритма.

2. Предоставление пользователю инструмента, реализующего данный алгоритм. При этом учащийся освобождается от необходимости выполнять шаги алгоритма, так как их автоматически выполняет визуализатор.

Визуализатор выполняет обычно следующие функции.

1. Пошаговое выполнение алгоритма.
2. Возможность просмотра действия алгоритма при разных наборах входных данных, в том числе и введенных пользователем.
3. Возможность просмотра действия алгоритма в динамике.
4. Возможность перезапуска алгоритма на текущем наборе входных данных.

Динамическая визуализация наглядно демонстрирует такую характеристику алгоритма, как трудоемкость (особенно при пошаговой демонстрации). Для некоторых алгоритмов (например, машины Тьюринга) динамический вариант демонстрации вообще представляется более естественным, чем любой набор статических иллюстраций.

До недавнего времени технология разработки визуализаторов сводилась к эвристике. При этом каждый алгоритм требовал нового подхода и осмысления реализации визуализатора (<http://rain.ifmo.ru/cat/view.php/vis>). Другими словами, технология разработки визуализаторов отсутствовала, и основные успехи в этой области были педагогическими [3, 4, 9].

Основным недостатком при построении визуализаторов является то, что обычно используются только такие понятия, как «входные и выходные переменные», а понятие «состояние» в явном виде не используется. Однако, по мнению авторов, совершенно естественным кажется, что каждый шаг визуализации необходимо проводить в соответствующем состоянии или на переходе. Поэтому при построении программ визуализации целесообразно использовать технологию автоматного программирования, базирующуюся на конечных автоматах [10].

В настоящей работе показано, что применение программирования с явным выделением состояний превращает процесс разработки логики визуализаторов в технологию.

Основным требованием к визуализатору является возможность пошаговой реализации алгоритма. Поэтому предлагаемая технология должна преобразовать исходный алгоритм в набор шагов для отображения на экране.

Визуализатор «по-крупному» предлагается строить следующим образом:

- по программе строится автомат логики визуализатора (контроллер – controller);
- выбираются визуализируемые переменные (модель – model);
- проектируется формирователь изображений и комментариев, который преобразует номер состояния и соответствующие значения визуализируемых переменных в «картинку» и поясняющий текст (представление – view).

Такая конструкция визуализатора соответствует одному из основных паттернов проектирования объектно-ориентированных программ, который обозначается аббревиатурой MVC (Model-View-Controller) [11].

Излагаемая ниже технология основана на методе построения конечного автомата по тексту программы [12]. При этом по указанному тексту строится схема алгоритма, которая на основе соответствующего кодирования преобразуется в граф перехода конечного автомата.

Исходя из изложенного, сформулируем технологию построения визуализаторов.

1. **Постановка задачи.**
2. **Решение задачи** (в словесно-математической форме).
3. **Выбор визуализируемых переменных.**
4. **Анализ алгоритма для визуализации.** Анализируется решение с целью определения того, что и как отображать на экране.
5. **Алгоритм решения задачи.**
6. **Реализация алгоритма на выбранном языке программирования.** На этом шаге производится реализация алгоритма, его отладка и проверка работоспособности.
7. **Построение схемы алгоритма по программе.**
8. **Преобразование схемы алгоритма в граф переходов автомата Мили** [12, 13].

9. **Формирование набора невизуализируемых переходов.**
10. **Выбор интерфейса визуализатора.**
11. **Сопоставление иллюстраций и комментариев с состояниями автомата.**
12. **Архитектура программы визуализатора.**
13. **Программная реализация визуализатора.**

Известно, что обычно создание программы разбивается на пять стадий: разработка требований, анализ, проектирование, реализация и тестирование. В данном случае к разработке требований относятся первые три пункта технологии. Анализ – пункты четыре и пять технологии. К проектированию относятся пункты с шестого по одиннадцатый. В отличие от традиционного для объектно-ориентированного программирования подхода, выбор архитектуры программы относится не к стадии проектирования, а к реализации, так как все предыдущие пункты технологии не зависят от реализации. Естественно, что к стадии реализации относится и последний пункт технологии.

Как будет показано ниже, при использовании предлагаемой технологии тестирование резко упрощается, а время написания визуализатора сокращается.

Продемонстрируем предложенную технологию на примере построения визуализатора алгоритма, использующего динамическое программирование [14], которое является одним из самых сложных разделов алгоритмики.

## 1. ПОСТАНОВКА ЗАДАЧИ

Рассмотрим словесную постановку задачи, которая в литературе носит название «дискретная целочисленная задача о рюкзаке» [14, 15]. Приведем одну из формулировок этой задачи.

*Имеется набор из  $K$  неделимых предметов, для каждого из которых известна масса  $M_i$  (в кг), являющаяся натуральным числом. Задача состоит в том, чтобы определить, существует ли хотя бы один набор предметов, размещаемый в рюкзаке, суммарная масса которых равна  $N$ . Если такой набор существует, то требуется определить его состав.*

## 2. РЕШЕНИЕ ЗАДАЧИ

Приведем словесную формулировку решения этой задачи.

### 2.1. Построение функции, представляющей оптимальное решение.

*Построим вектор  $M(1, K)$ , каждый элемент  $M_i$  которого соответствует массе  $i$ -го предмета.*

*Введем функцию  $T(i, j)$ , значение которой равно единице, если среди предметов с первого по  $i$ -ый существует хотя бы один набор предметов, сумма масс которых равна  $j$ , и нулю – в противном случае.*

### 2.2. Рекуррентные соотношения, связывающие оптимальные значения для подзадач.

*Определим начальные значения функции  $T$ :*

- $T(0, j) = 0$  при  $j \geq 1$  { без предметов нельзя набрать массу  $j \geq 1$ };
- $T(i, 0) = 1$  при  $i \geq 0$  { всегда можно набрать нулевую массу }.

*Определим возможные значения функции  $T$  при ненулевых значениях аргументов.*

*Существуют две возможности при выборе предметов: включить предмет с номером  $i$  в набор или не включать. Это в математической форме может быть записано следующим образом:*

- если предмет с номером  $i$  не включается в набор, то решение задачи с  $i$  предметами сводится к решению подзадачи с  $(i - 1)$ -им предметом. При этом  $T(i, j) = T(i - 1, j)$ ;

- если предмет с номером  $i$  включается в набор, то это уменьшает суммарную массу для  $i-1$  первых предметов на величину  $M_i$ . При этом  $T(i, j) = T(i-1, j - M_i)$ . Эта ситуация возможна только тогда, когда  $M_i \leq j$ .

Для получения решения необходимо выбрать большее из значений рассматриваемой функции. Поэтому рекуррентные соотношения при  $i \geq 1$  и  $j \geq 1$  имеют вид:

- $T(i, j) = T(i-1, j)$  при  $j < M_i$ ;
- $T(i, j) = \max(T(i-1, j), T(i-1, j - M_i))$  при  $j \geq M_i$ .

### 2.3. Задание исходных данных.

Пусть  $K = 5$  – заданы пять предметов.

Эти предметы имеют следующие массы:  $M_1=4$ ;  $M_2=5$ ;  $M_3=3$ ;  $M_4=7$ ;  $M_5=6$ .

Суммарная масса предметов, которые должны быть размещены в рюкзаке, равна 16 ( $N=16$ ).

### 2.4. Вычисление значений функции $T(i, j)$ (прямой ход алгоритма).

Используя приведенные выше рекуррентные соотношения, построим таблицу значений рассматриваемой функции (табл. 1).

Таблица 1

		$j$																
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$i$	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	2	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0
	3	1	0	0	1	1	1	0	1	1	1	0	0	1	0	0	0	0
	4	1	0	0	1	1	1	0	1	1	1	1	1	1	0	1	1	1
	5	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Из таблицы следует, что  $T(5,16)=1$ , и поэтому существует хотя бы один набор предметов, удовлетворяющий постановке задачи.

### 2.5. Определение набора предметов (обратный ход алгоритма).

Рассмотрим элементы  $T(5,16)$  и  $T(4,16)$  таблицы. Так как их значения равны, то массу в 16 кг можно набрать с помощью первых четырех предметов – пятый предмет в набор можно не включать.

Теперь рассмотрим элементы  $T(4,16)$  и  $T(3,16)$ . Их значения не равны, и поэтому набор из трех предметов не обеспечивает решение задачи. Следовательно, четвертый предмет должен быть включен в набор. Поэтому «остаток» массы, размещаемой в рюкзаке, который должен быть набран из оставшихся предметов, равен:  $16 - M_4 = 16 - 7 = 9$ .

Для элементов  $T(3,9)$  и  $T(2,9)$  значения равны – третий предмет в набор не включается.

Для элементов  $T(2,9)$  и  $T(1,9)$  значения не равны – второй предмет должен быть включен в набор. При этом остаток равен:  $9 - M_2 = 9 - 5 = 4$ .

И, наконец, рассмотрим элементы  $T(1,4)$  и  $T(0,4)$ . Их значения не равны – первый предмет должен быть включен в набор, а «остаток» массы для других предметов равен 0.

Таким образом, одно из решений задачи найдено – в набор включены первый, второй и четвертый предметы.

## 3. ВЫБОР ВИЗУАЛИЗИРУЕМЫХ ПЕРЕМЕННЫХ

Перечислим переменные, которые содержат значения, необходимые для визуализации:

- $i$  – номер предмета;
- $j$  – сумма весов предметов;

- $T[][]$  – значения функции  $T$ ;
- `positions` – номера предметов.

#### 4. АНАЛИЗ АЛГОРИТМА ДЛЯ ЕГО ВИЗУАЛИЗАЦИИ

Из рассмотрения алгоритма следует, что его визуализация должна выполняться следующим образом:

- показать таблицу, заполненную начальными значениями;
- отразить прямой ход решения задачи – процесс построения таблицы значений  $T(i, j)$ ;
- отразить обратный ход решения задачи – процесс поиска набора предметов за счет «обратного» движения по указанной таблице.

#### 5. АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ

Приведем алгоритм решения задачи на псевдокоде [14]:

```

1  for  $j \leftarrow 1$  to  $N$ 
2      do  $T[0, j] \leftarrow 0$ 
3  for  $i \leftarrow 0$  to  $K$ 
4      do  $T[i, 0] \leftarrow 1$ 
5  for  $i \leftarrow 1$  to  $K$ 
6      do for  $j \leftarrow 1$  to  $N$ 
7          do if  $j < M[i]$ 
8              then  $T[i, j] \leftarrow T[i-1, j]$ 
9              else  $T[i, j] \leftarrow \text{Max}(T[i-1, j], T[i-1, j-M[i]])$ 
10  $sum \leftarrow N$ 
11 if  $T[K, sum] = 1$ 
12     then
13         for  $i \leftarrow K$  downto 1
14             do if  $T[i, sum] \neq T[i-1, sum]$ 
15                 then
16                     добавить в искомый набор
17                      $sum \leftarrow sum - M[i]$ 
18         return искомый набор
19     else return «набор не существует»

```

#### 6. РЕАЛИЗАЦИЯ АЛГОРИТМА НА ЯЗЫКЕ JAVA

Перепишем программу на псевдокоде с помощью языка Java.

```

/**
 * @param N суммарный вес предметов в рюкзаке
 * @param M массив весов предметов
 * @return список номеров предметов, сумма весов которых равна N,
 * либо null если это невозможно
 */
private static Integer[] solve(int N, int[] M) {
    // Переменные циклов
    int i, j;
    // Количество предметов
    int K = M.length;
    // Массив T

```

```

int[][] T = new int[K + 1][N + 1];
// Результирующие номера
ArrayList positions = new ArrayList();
// Результат работы true, если суммарный вес можно получить
boolean result = false;

// Определение начальных значений функции T
for (j = 1; j <= N; j++) {
    T[0][j] = 0;
}
for (i = 0; i <= K; i++) {
    T[i][0] = 1;
}

// Построение таблицы значений функции T
for (i = 1; i <= K; i++) {
    for (j = 1; j <= N; j++) {
        if (j >= M[i - 1]) {
            T[i][j] = Math.max(T[i - 1][j], T[i - 1][j - M[i - 1]]);
        }
        else {
            T[i][j] = T[i - 1][j];
        }
    }
}

// Определение набора предметов
int sum = N;
if (T[K][sum] == 1) {
    // Решение существует
    for (i = K; i >= 1; i--) {
        if (T[i][sum] != T[i - 1][sum]) {
            positions.add(new Integer(i));
            sum -= M[i - 1];
        }
    }
    // Решение найдено
    result = true;
}

// Если решение найдено, то оно возвращается, иначе возвращается null
return (Integer[])(result ? positions.toArray(new Integer[0]) : null);
}

```

В этой программе операции ввода/вывода не приведены, так как их будет выполнять визуализатор.

## 7. ПОСТРОЕНИЕ СХЕМЫ АЛГОРИТМА ПО ПРОГРАММЕ

Построим по тексту программы схему алгоритма (рис. 1).

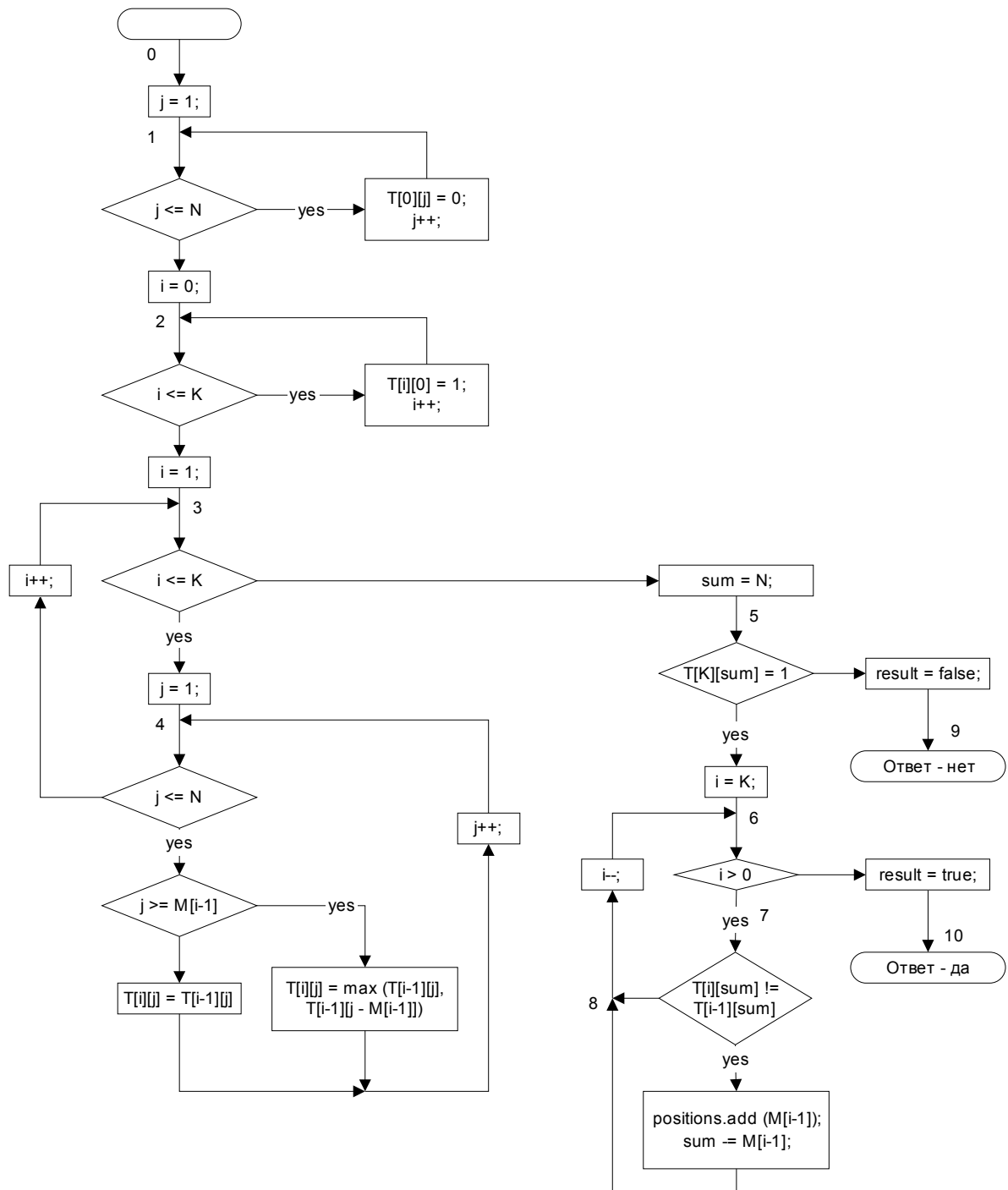


Рис. 1. Схема алгоритма решения задачи о рюкзаке

## 8. ПРЕОБРАЗОВАНИЕ СХЕМЫ АЛГОРИТМА В ГРАФ ПЕРЕХОДОВ АВТОМАТА МИЛИ

В работе [12] показано, что при программной реализации по схеме алгоритма может быть построен как автомат Мура, так и автомат Мили. В автоматах первого типа действия выполняются в вершинах, а в автоматах второго типа – на переходах. Визуализаторы могут быть построены как с использованием автоматов Мура, так и автоматов Мили. Для уменьшения числа состояний в настоящей работе применяются автоматы Мили.

В соответствии с методом, изложенным в работе [12], определим состояния, которые будет иметь автомат Мили, соответствующий этой схеме алгоритма. Для этого присвоим

номера точкам, следующим за последовательно расположенными операторными вершинами (последовательность может состоять и из одной вершины). Номера присваиваются также начальной и конечным вершинам.

Количество состояний, определенных описанным методом, бывает для целей визуализации недостаточным, так как в ряде случаев необходимо показывать также условия и результаты некоторых действий.

Исходя из изложенного, для схемы алгоритма на рис. 1 выделим девять состояний автомата с номерами 0 – 6, 9, 10. Еще одно состояние с номером 7 введено для визуализации обратного хода алгоритма. И, наконец, состояние с номером 8 используется для визуализации результата выбора на обратном ходе. Таким образом, автомат визуализации будет иметь одиннадцать состояний, а его граф переходов, который строится за счет определения путей между выделенными точками на схеме алгоритма – одиннадцать вершин (рис. 2).

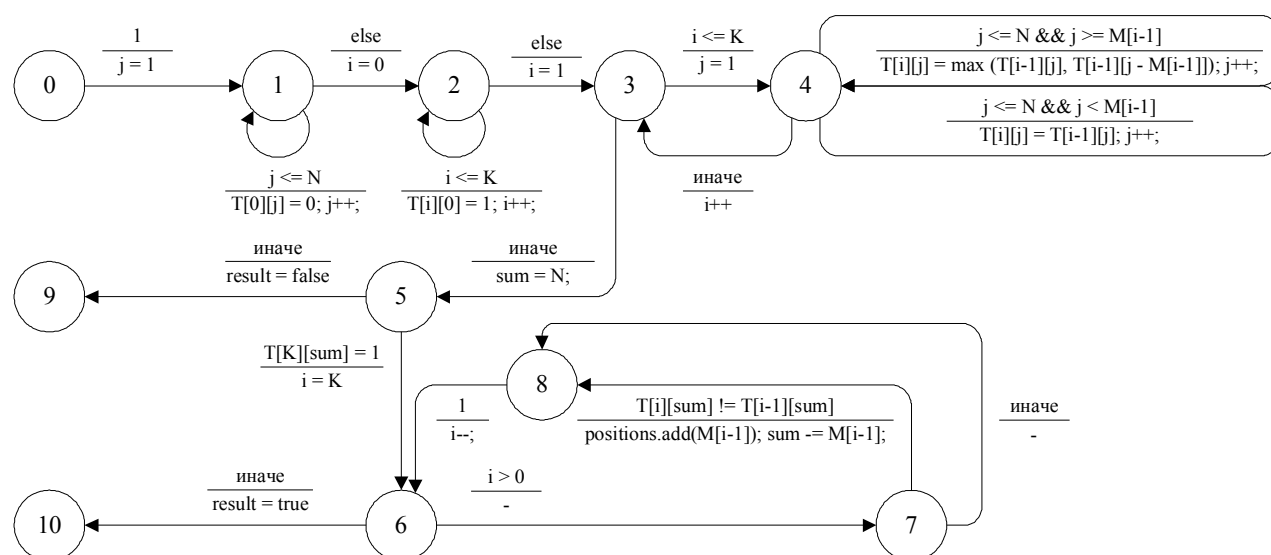


Рис. 2. Граф переходов для визуализации логики алгоритма

Как следует из графа переходов, на петлях в состояниях 1 и 2 осуществляется заполнение таблицы значений функции  $T$  начальными данными в первой строке и в первом столбце. На петлях в состоянии 4 реализуется алгоритм заполнения таблицы значений функции  $T$ . Переходы из состояния 7 в состояние 8 отображают первую половину шага алгоритма обратного хода, причем более сложный переход соответствует включению предмета с номером  $i$  в результирующий набор, а второй переход – не включению предмета в этот набор. Переход из состояния 8 в состояние 6 соответствует второй половине шага алгоритма обратного хода.

## 9. ФОРМИРОВАНИЕ НАБОРА НЕВИЗУАЛИЗИРУЕМЫХ ПЕРЕХОДОВ

Из анализа графа переходов (рис. 2) следует, что «неинтересными» (не изменяющими визуализацию) для пользователя являются следующие переходы:  $1 \rightarrow 2$ ,  $3 \rightarrow 4$ ,  $3 \rightarrow 5$ ,  $4 \rightarrow 3$ ,  $6 \rightarrow 7$ ,  $6 \rightarrow 10$ . Устранение таких переходов осуществляется за счет того, что один шаг работы визуализатора состоит из нескольких переходов автомата, реализующего его логику.

Для исключения «неинтересных» переходов шаг визуализатора реализуется следующим образом:

- 1     **while** следующий переход **in** ( $0 \rightarrow 1$ ,  $1 \rightarrow 2$ ,  $3 \rightarrow 4$ ,  $3 \rightarrow 5$ ,  $4 \rightarrow 3$ ,  $6 \rightarrow 7$ ,  $6 \rightarrow 10$ )
- 2     **do** сделать переход автомата



## 10. ВЫБОР ИНТЕРФЕЙСА ВИЗУАЛИЗАТОРА

В верхней части визуализатора (в дальнейшем называемой *иллюстрацией*) представляется следующая информация:

- таблица значений функции  $T(i, j)$ , где  $i = 0, \dots, 5$ ;  $j = 0, \dots, 16$ ;
- набор весов предметов (крайний левый столбец с цифрами 4, 5, 3, 7, 6).

В нижней части визуализатора расположена панель управления, которая содержит следующие кнопки:

- «>>» – сделать шаг алгоритма;
- «Заново» – начать алгоритм заново;
- «Авто» – перейти в автоматический режим;
- «<<», «>>» – изменение паузы между автоматическими шагами алгоритма.

Среднюю часть визуализатора занимает область комментариев, в которой на каждом шаге отображается номер состояния и описание действия, выполняемого в этом состоянии.

Визуализатор в исходном состоянии, которое соответствует начальному состоянию автомата, представлен на рис. 3.

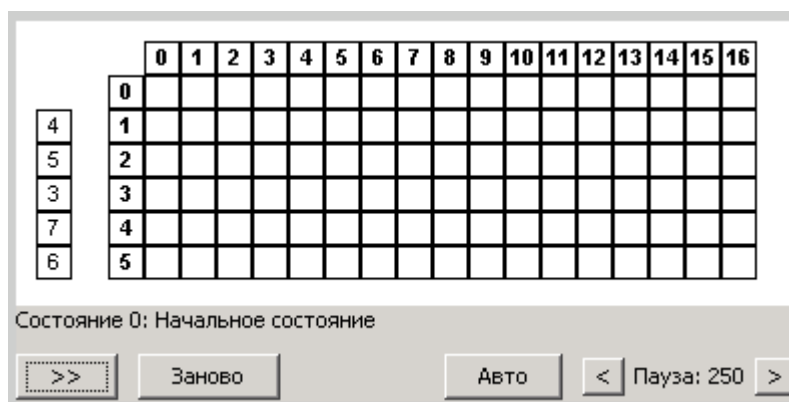


Рис. 3. Начальное состояние визуализатора

В следующем разделе приведены иллюстрации, отображающие внешний вид визуализатора в различных состояниях, которые должны формироваться в процессе его работы. Они перенесены в статью с указанием соответствующих состояний, как скриншоты работающего визуализатора.

## 11. СОПОСТАВЛЕНИЕ ИЛЛЮСТРАЦИЙ И КОММЕНТАРИЕВ С СОСТОЯНИЯМИ АВТОМАТА

Ввиду того, что при построении визуализатора используется автомат Мили, то будем в рассматриваемом состоянии отображать **действия, которые будут выполнены при переходе** из него. При этом серым цветом подсвечивается текущая ячейка таблицы, а черным – рабочие ячейки. При этом на прямом ходе ищется значение в текущей ячейке, исходя из значений в рабочих ячейках. На обратном ходе – значение в текущей ячейке сравнивается со значениями в рабочих ячейках для определения вхождения элемента в набор.

Состояние 0. Описано в предыдущем разделе.

Состояние 1. Заполнение нулями верхней строки таблицы  $T$  (рис. 4). Обратим внимание, что одному управляющему состоянию визуализатора обычно соответствует некоторое число вычислительных состояний (шагов визуализации) [16]. Поэтому рис. 4 разбит на два, первый из них (рис. 4а) соответствует первому шагу в этом состоянии, а второй (рис. 4б) – последнему.

Состояние 2. Заполнение единицами левого столбца таблицы  $T$  (рис. 5).



Рис. 4а. Состояние 1. Начало



Рис. 4б. Состояние 1. Конец



Рис. 5. Состояние 2

Состояние 3 является «невизуализируемым», так как оно служебное.

Состояние 4 может отображаться в двух различных вариантах, потому что граф на рис. 2 имеет в этом состоянии две петли, помеченных разными условиями и соответствующими действиями. На рис. 6 приведена иллюстрация, соответствующая нижней петле, в которой выполняется копирование значения из расположенной выше ячейки таблицы (помечена темным цветом) в ячейку, расположенную ниже (помечена серым цветом), поскольку рассматриваемый в данный момент предмет не может быть использован в наборе. Это имеет место в случае, когда сумма весов предметов для текущего столбца меньше веса предмета, соответствующего текущей строке. На рис. 7 приведена иллюстрация для верхней петли в рассматриваемом

состоянии, на которой показано, что имеет место альтернатива в выборе значения, поскольку вес  $T(i, j)$  может быть получен двумя разными способами.

При использовании первого способа текущий предмет включается в набор, а для второго способа рассматриваемый предмет в набор не включается. Наличие единицы в одной из двух черных (рис. 7) ячеек соответствует одному из вариантов. Поскольку на рис. 7 выполняется соотношение  $T(1, 4) = 1$ , то набор с весом 4 может быть получен с применением одного предмета. Поэтому набор с весом 9 может быть получен из двух предметов за счет добавления второго предмета с весом 5. Следовательно,  $T(2, 9) = 1$ .



Рис. 6. Состояние 4 (нижняя петля)



Рис. 7. Состояние 4 (верхняя петля)

На этом прямой ход алгоритма завершен.

Состояние 5 отображает проверку того, что искомый суммарный вес набора предметов может быть получен, поскольку значение в правой нижней ячейке равно единице (рис. 8). Это состояние является промежуточным между прямым и обратным ходом алгоритма, на котором собственно и выполняется поиск результирующего набора.



Рис. 8. Состояние 5

Рассмотрим состояния автомата, соответствующие обратному ходу алгоритма.

Состояния 6 – 8 автомата, образующие замкнутый контур на рис. 2, порождают искомый набор предметов. Опишем, как эти состояния иллюстрируются.

Состояние 6 не изображается.

Состояние 7 отображает альтернативу первой половины шага при обратном ходе алгоритма (рис. 9) из серой ячейки в одну из черных. Вариант перехода в ячейку в том же столбце означает, что текущий предмет не включается в результирующий набор. Переход во вторую черную ячейку означает включение предмета с номером  $i$  в набор. При этом отметим, что собственно переход всегда осуществляется в ячейку, помеченную единицей. В случае если обе ячейки содержат по единице, то выбирается первый из описанных вариантов.

Состояние 8 отображает результат выбора на предыдущем шаге (рис. 10). Из левой части рисунка следует, что предмет с весом 7 включен в результирующий набор.



Рис. 9. Состояние 7



Рис. 10. Состояние 8

Для рассматриваемых исходных данных состояние 9 отсутствует.  
Состояние 10 отображает искомый результат – набор предметов, реализующих суммарный вес, равный 16 (рис. 11).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0
3	1	0	0	1	1	1	0	1	1	1	0	0	1	0	0	0	0
7	1	0	0	1	1	1	0	1	1	1	1	1	1	0	1	1	1
6	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Состояние 10: Конец. Искомый набор найден: [7, 5, 4]

>>    Заново    Авто    < Пауза: 250 >

Рис. 11. Состояние 10

## 12. АРХИТЕКТУРА ПРОГРАММЫ ВИЗУАЛИЗАТОРА

Во введении было признано целесообразным реализовать визуализатор на основе паттерна «Модель – Вид – Контроллер».

В данном случае «Модель» – визуализируемые переменные, которые являются глобальными (Globals), «Вид» – формирователь иллюстраций и комментариев (Drawer), а «Контроллер» – автомат (Automaton). Объединение указанных составляющих осуществляется с помощью четвертой компоненты – Applet, которая осуществляет также реализацию панели управления.

Диаграмма классов, реализующая этот паттерн для рассматриваемого примера, приведена на рис. 12.

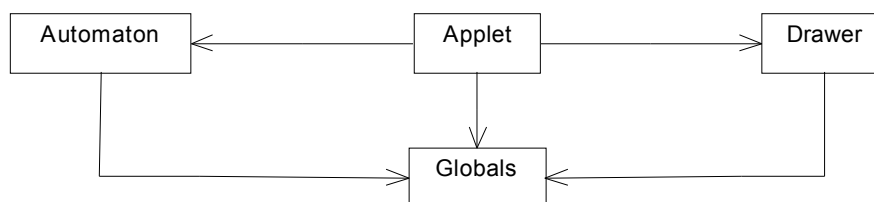


Рис.12. Диаграмма классов визуализатора

В качестве глобальных переменных используются переменные, описанные в разд. 3. Спроектированный автомат описан в разд. 8, а иллюстрации и комментарии, «привязанные к состояниям автомата» – в разд. 10, 11.

## 13. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ВИЗУАЛИЗАТОРА

В работе [13] было предложено переходить формально и изоморфно от графа переходов автомата к его программной реализации на основе оператора *switch*. Для рассматриваемого примера результатом преобразования графа переходов (рис. 2) является текст программы, приведенный ниже:

```
private void makeAutomationStep() {
    switch (state) {
```

```

case 0: // Начальное состояние
    j = 1;
    state = 1;
    break;

case 1: // Заполнение 0-й строки
    if (j <= N) {
        T[0][j] = 0;
        j++;
    } else {
        i = 0;
        state = 2;
    }
    break;

case 2: // Заполнение 0-го столбца
    if (i <= K) {
        T[i][0] = 1;
        i++;
    } else {
        i = 1;
        state = 3;
    }
    break;

case 3: // Переход к следующей строке
    if (i <= K) {
        j = 1;
        state = 4;
    } else {
        sum = N;
        state = 5;
    }
    break;

case 4: // Заполнение очередной ячейки
    if (j <= N && j >= M[i - 1]) {
        T[i][j] = Math.max(T[i - 1][j], T[i - 1][j - M[i - 1]]);
        j++;
    } else if (j <= N && j < M[i - 1]) {
        T[i][j] = T[i - 1][j];
        j++;
    } else {
        i++;
        state = 3;
    }
    break;

case 5: // Конец заполнения таблицы
    // Решение найдено
    if (T[K][sum] == 1) {
        i = K;
        state = 6; // Делать обратный ход
    } else {
        result = false;
        state = 9; // Обратный ход не требуется
    }
    break;

case 6: // Обратный ход
    if (i > 0) {
        state = 7;
    } else {
        result = true;
        state = 10;
    }
    break;

case 7: // Поиск очередного предмета
    if (T[i][sum] != T[i - 1][sum]) {
        positions.add(new Integer(i - 1));
        sum -= M[i - 1];
    }

```

```

        state = 8;
    } else {
        state = 8;
    }
    break;

case 8:
    i--;
    state = 6;
    break;
}
}

```

Такая реализация автомата резко упрощает построение визуализатора, так как к каждому состоянию автомата (метке *case*) могут быть «привязаны» соответствующие иллюстрации и комментарии. В силу того, что реализация визуализатора выполняется с помощью указанного выше паттерна, «привязка» в данном случае осуществляется с помощью второго оператора *switch*. Таким образом, в визуализаторе используется два оператора *switch*, первый из которых реализует автомат, а второй применяется в формирователе иллюстраций и комментариев.

Отметим, что формальный подход к построению логики визуализатора привел к тому, что отладка логики отсутствовала, а небольших изменений потребовала неформализуемая часть программы, связанная с построением иллюстраций и комментариев.

Аплет и исходный текст программы визуализатора приведены на сайте <http://is.ifmo.ru/> в разделах «Статьи» и «Визуализаторы». Исходный текст программы приведен также и в настоящей статье (приложения 1 – 7).

## ЗАКЛЮЧЕНИЕ

Изложенный метод демонстрирует эффективность стиля программирования от состояний [17], который совместно с объектно-ориентированным программированием позволяет говорить об «объектно-ориентированном программировании с явным выделением состояний» [10].

Отметим, что для рассмотренного примера переход от классически написанной программы к автоматной увеличил размер кода незначительно – всего на 15 – 20 %.

При этом построение визуализатора для рассмотренного примера на основе предлагаемого подхода заняло около четырех часов, в то время как при традиционном подходе путь от классической программы до визуализатора занимает обычно несколько дней из-за «неформального движения по этому пути». Кроме того, автоматный визуализатор практически не требовал отладки.

Для других классов алгоритмов увеличение объема кода автоматной реализации по сравнению с классической может достигать нескольких раз. Однако и в этом случае время написания визуализатора на основе предлагаемого подхода остается крайне незначительным при простой отладке.

Описанный подход может быть усовершенствован. Во-первых, вместо автоматов Мили могут применяться автоматы Мура. При этом в общем случае число состояний увеличивается, однако визуализация действий в вершинах является более естественной, чем на переходах. Во-вторых, этапы технологии, связанные с проектированием и реализацией, могут быть автоматизированы.

Таким образом, анализ зарубежных и отечественных Интернет-ресурсов в области построения визуализаторов, а также опыт их разработки и приема у студентов в рамках учебного процесса в 1999/2000 гг. на кафедре «Компьютерные технологии» СПбГУ ИТМО, позволяет утверждать, что предложенный подход является «прорывом» в области построения визуализаторов.

Работа выполнена в рамках научно–исследовательской работы «Разработка основных положений создания программного обеспечения систем управления на основе автоматного подхода», финансируемой Министерством образования Российской Федерации.

## ЛИТЕРАТУРА

1. Интернет-школа программирования. <http://ips.ifmo.ru/>
2. Казаков М.А., Мельничук О.П., Парфенов В.Г. Интернет школа программирования в СПбГИТМО (ТУ). Реализация и внедрение // Всероссийская научно-методическая конференция «Телематика'2002». СПб.: 2002. – С.308-309. ([http://tm.ifmo.ru/tm2002/db/doc/get\\_thes.php?id=170](http://tm.ifmo.ru/tm2002/db/doc/get_thes.php?id=170))
3. Столяр С.Е., Осипова Т.Г., Крылов И.П., Столяр С.С. Об инструментальных средствах для курса информатики // II Всероссийская конференция «Компьютеры в образовании». СПб.: 1994.
4. 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.
5. Complete Collection of Algorithm Animations. <http://www.cs.hope.edu/~algnim/ccaa/>
6. Interactive Data Structure Visualizations <http://www.student.seas.gwu.edu/~idsv/idsv.html>
7. Javaa examples <http://www.cs.duke.edu/csed/jawaa2/examples.html>
8. Sorting Algorithms <http://www.cs.rit.edu/~atk/Java/Sorting/sorting.html>
9. A Testbed for Pedagogical Requirements in Algorithm Visualizations <http://nibbler.tk.informatik.tu-darmstadt.de/publications/2002/TestbedPedReqinAV.pdf>
10. Шалыто А.А. Технология автоматного программирования // Мир ПК. 2003 № 10. С.74-78. <http://is.ifmo.ru>, раздел «Статьи».
11. Гамма Э., Хэлм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2001. – 325 с.
12. Шалыто А.А., Туккель Н.И. Преобразование итеративных алгоритмов в автоматные // Программирование. 2002. № 5. – С.12-26. <http://is.ifmo.ru>, раздел «Статьи».
13. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. – 628 с.
14. Кормен Т., Лайзерсон Ч., Ривест Р. Алгоритмы. Построение и анализ. М.: МЦМНО, 2000. – 960 с.
15. Седжвик Р. Фундаментальные алгоритмы на С++. Анализ. Структуры данных. Сортировка. Поиск. М.: DiaSoft, 2001. – 687 с.
16. Шалыто А.А., Туккель Н.И. От тьюрингова программирования к автоматному // Мир ПК. 2002. №2. – С.144-149. <http://is.ifmo.ru>, раздел «Статьи».
17. Непейвода Н.Н., Скопин И.Н. Основания программирования. Ижевск: Научно-издат. центр «Регулярная хаотическая динамика», 2003. – 896 с.

### Об авторах

Казаков Матвей Алексеевич, аспирант кафедры «Компьютерные технологии» СПбГУ ИТМО, капитан команды СПбГУ ИТМО, занявшей третье место на командном чемпионате мира по программированию ACM 1998/1999 гг.

Шалыто Анатолий Абрамович, доктор технических наук, профессор кафедры «Компьютерные технологии» СПбГУ ИТМО, победитель конкурса исследовательских проектов в области автоматизации проектирования интегральных схем, который проводился компанией *Intel* в СНГ в 2003 г.



## Приложение 1. Программа, реализующая решение «дискретной задачи о рюкзаке»

```
/*
 * Date: Jan 10, 2004
 * Copyright (c) 2004 Matvey Kazakov.
 */
package ru.ifmo.ips.visualizers.backpack;

import java.util.ArrayList;

/**
 * Решение задачи.
 *
 * @author Matvey Kazakov
 */
public class Solution {
    /**
     * @param N суммарный вес предметов в рюкзаке
     * @param M массив весов предметов
     * @return список номеров предметов, сумма весов которых равна N,
     *         либо null если это невозможно.
     */
    private static Integer[] solve(int N, int[] M) {
        // Переменные циклов
        int i, j;
        // Количество предметов
        int K = M.length;
        // Массив T
        int[][] T = new int[K + 1][N + 1];
        // Результирующие номера
        ArrayList positions = new ArrayList();
        // Результат работы true, если суммарный вес можно получить
        boolean result = false;

        // Определение начальных значений функции T
        for (j = 1; j <= N; j++) {
            T[0][j] = 0;
        }
        for (i = 0; i <= K; i++) {
            T[i][0] = 1;
        }

        // Построение таблицы значений функции T
        for (i = 1; i <= K; i++) {
            for (j = 1; j <= N; j++) {
                if (j >= M[i - 1]) {
                    T[i][j] = Math.max(T[i - 1][j], T[i - 1][j - M[i - 1]]);
                } else {
                    T[i][j] = T[i - 1][j];
                }
            }
        }

        // Определение набора предметов
        int sum = N;
        if (T[K][sum] == 1) {
            // Решение существует
            for (i = K; i >= 1; i--) {
                if (T[i][sum] != T[i - 1][sum]) {
                    positions.add(new Integer(i));
                    sum -= M[i - 1];
                }
            }
            // Решение найдено
            result = true;
        }

        // Если решение найдено, то оно возвращается, иначе возвращается null
        return (Integer[])(result ? positions.toArray(new Integer[0]) : null);
    }
}
```

```

public static void main(String[] args) {
    int[] M = new int[]{4, 5, 3, 7, 6};
    int N = 16;
    Integer[] result = solve(N, M);
    System.out.print("[");
    for (int i = 0; i < result.length; i++) {
        Integer integer = result[i];
        if (i > 0) {
            System.out.print(", ");
        }
        System.out.print(M[integer.intValue() - 1]);
    }
    System.out.println("]");
}
}

```

## Приложение 2. Глобальные переменные

```

/*
 * Date: Jan 10, 2004
 * Copyright (c) 2004 Matvey Kazakov.
 */
package ru.ifmo.ips.visualizers.backpack;

import java.util.Collection;
import java.util.ArrayList;

/**
 * Глобальные переменные
 *
 * @author Matvey Kazakov
 */
public class Globals {

    // Состояние автомата
    public int state, i, j, K, N, sum;
    public int[][] T;
    public int[] M;
    public boolean result;
    public Collection positions;

    public Globals(int[] M, int N) {
        K = M.length;
        this.M = M;
        this.N = N;
        positions = new ArrayList();
        T = new int[K + 1][N + 1];
        init();
    }

    public void init() {
        state = 0;
        positions.clear();
        for (int ii = 0; ii <= K; ii++)
            for (int jj = 0; jj <= N; jj++) {
                T[ii][jj] = -1;
            }
        i = -1;
        j = -1;
        sum = 0;
        result = false;
    }
}

```

### Приложение 3. Управляющий автомат

```
/*
 * Date: Jan 10, 2004
 * Copyright (c) 2004 Matvey Kazakov.
 */
package ru.ifmo.ips.visualizers.backpack;

/**
 * Реализует логику визуализатора.
 *
 * @author Matvey Kazakov
 */
public class Automaton {

    private Globals g;

    public Automaton(Globals g) {
        this.g = g;
    }

    private void makeAutomationStep() {
        switch (g.state) {
            case 0: // Начальное состояние
                g.j = 1;
                g.i = 0;
                g.state = 1;
                break;

            case 1: // Заполнение 0-й строки
                if (g.j <= g.N) {
                    g.T[0][g.j] = 0;
                    g.j++;
                } else {
                    g.j = 0;
                    g.state = 2;
                }
                break;

            case 2: // Заполнение 0-го столбца
                if (g.i <= g.K) {
                    g.T[g.i][0] = 1;
                    g.i++;
                } else {
                    g.i = 1;
                    g.state = 3;
                }
                break;

            case 3: // Переход к следующей строке
                if (g.i <= g.K) {
                    g.j = 1;
                    g.state = 4;
                } else {
                    g.sum = g.N;
                    g.state = 5;
                }
                break;

            case 4: // Заполнение очередной ячейки
                if (g.j <= g.N && g.j >= g.M[g.i - 1]) {
                    g.T[g.i][g.j] = Math.max(g.T[g.i - 1][g.j],
                                                g.T[g.i - 1][g.j - g.M[g.i - 1]]);
                    g.j++;
                } else if (g.j <= g.N && g.j < g.M[g.i - 1]) {
                    g.T[g.i][g.j] = g.T[g.i - 1][g.j];
                    g.j++;
                } else {
                    g.i++;
                    g.state = 3;
                }
            }
        }
    }
}
```

```

    }
    break;

case 5: // Конец заполнения таблицы
    if (g.T[g.K][g.sum] == 1) { // Решение найдено
        g.i = g.K;
        g.state = 6; // Делать обратный ход
    } else {
        g.result = false;
        g.state = 9; // Обратный ход не требуется
    }
    break;

case 6: // Обратный ход
    if (g.i > 0) {
        g.state = 7;
    } else {
        g.result = true;
        g.state = 10;
    }
    break;

case 7: // Поиск очередного предмета
    if (g.T[g.i][g.sum] != g.T[g.i - 1][g.sum]) {
        g.positions.add(new Integer(g.i - 1));
        g.sum -= g.M[g.i - 1];
        g.state = 8;
    } else {
        g.state = 8;
    }
    break;

case 8:
    g.i--;
    g.state = 6;
    break;
}
}

/**
 * Пропуск невизуализируемых переходов.
 */
public void makeStep() {
    makeAutomationStep();
    while (!isFinished() && (
        g.state == 1 && g.j == g.N + 1
        || g.state == 2 && g.i == g.K + 1
        || g.state == 3
        || g.state == 6
        || g.state == 4 && g.j == g.N + 1
    )) {
        makeAutomationStep();
    }
}

/**
 * Проверяет, что автомат уже остановился.
 */
public boolean isFinished() {
    return g.state == 9 || g.state == 10;
}
}

```

## Приложение 4. Построение иллюстраций и комментариев

```
/*
 * Date: Jan 10, 2004
 * Copyright (c) 2004 Matvey Kazakov.
 */

package ru.ifmo.ips.visualizers.backpack;

import javax.swing.*;
import java.awt.*;
import java.util.Iterator;
import java.text.MessageFormat;

/**
 * Класс отвечает за формирование сообщений и иллюстраций
 *
 * @author Matvey Kazakov
 */
public class Drawer extends JPanel {

    // ===== Константы для рисования =====
    private static final Color COLOR_BORDER = Color.black;
    private static final Color COLOR_CURRENT = Color.lightGray;
    private static final Color COLOR_SELECT = Color.gray;
    private static final Color COLOR_BACKGROUND = Color.white;

    private static final Font plainFont = new Font("Arial", Font.PLAIN, 11);
    private static final Font headerFont = plainFont.deriveFont(Font.BOLD);

    // ===== Текстовые сообщения =====
    private static final String MESSAGE_STATE_0 = "Состояние 0: Начальное состояние";
    private static final String MESSAGE_STATE_1
        = "Состояние 1: Вес {0} не может быть получен из нуля предметов";
    private static final String MESSAGE_STATE_2
        = "Состояние 2: Вес 0 может быть получен из первых {0} предметов";
    private static final String MESSAGE_STATE_4
        = "Состояние 4 (нижняя петля): Заполнение позиции с номером {0}, {1}: перенос значения";
    private static final String MESSAGE_STATE_4_ALT
        = "Состояние 4 (верхняя петля): Заполнение позиции с номером {0}, {1} : альтернатива";
    private static final String MESSAGE_STATE_5_OK
        = "Состояние 5: Искомый вес может быть получен";
    private static final String MESSAGE_STATE_5_FAIL
        = "Состояние 5: Искомый вес не может быть получен";
    private static final String MESSAGE_STATE_7_ALT
        = "Состояние 7: Выбор обратного хода: альтернатива";
    private static final String MESSAGE_STATE_7
        = "Состояние 7: Выбор обратного хода";
    private static final String MESSAGE_STATE_8 = "Состояние 8: Следующая ячейка найдена";
    private static final String MESSAGE_STATE_9
        = "Состояние 9: Конец. Искомый набор не существует";
    private static final String MESSAGE_STATE_10
        = "Состояние 10: Конец. Искомый набор найден: {0}";

    private Globals globals;

    private JLabel messageLabel;
    private JLabel[][] T;
    private JLabel[] M;

    // Служебные переменные. Хранят ссылки на предыдущие подсвеченные ячейки
    private JLabel previous = null;
    private JLabel prev_sel1 = null;
    private JLabel prev_sel2 = null;

    public Drawer(Globals globals) {
        this.globals = globals;
        setLayout(new BorderLayout());
        JPanel picturePanel = new JPanel();
        JPanel messagePanel = new JPanel(new BorderLayout());
        add(picturePanel, BorderLayout.CENTER);
    }
}
```

```

add(messagePanel, BorderLayout.SOUTH);
messageLabel = new JLabel();
messagePanel.add(messageLabel, BorderLayout.CENTER);
// Иллюстрация представляет из себя массив меток.
// Для подсветки ячеек используется цвет фона.
T = new JLabel[globals.T.length][globals.T.length > 0 ? globals.T[0].length : 0];
M = new JLabel[globals.K];
picturePanel.setLayout(new GridLayout(globals.K + 2, globals.N + 4));
picturePanel.setBackground(COLOR_BACKGROUND);
picturePanel.setBorder(BorderFactory.createLineBorder(Color.white, 10));
for (int i = 0; i < globals.K + 2; i++) {
    if (i > 0) {
        JLabel[] row = T[i - 1];
        for (int j = 0; j < globals.N + 4; j++) {
            if (j > 2) {
                JLabel cell = new JLabel();
                row[j - 3] = cell;
                picturePanel.add(createCell(cell));
            } else if (j == 2) {
                picturePanel.add(createHeaderCell(i - 1));
            } else if (j == 0 && i > 1) {
                M[i - 2] = new JLabel("" + globals.M[i - 2]);
                picturePanel.add(createCell(M[i - 2]));
            } else {
                picturePanel.add(createSpacer());
            }
        }
    } else {
        for (int j = 0; j < globals.N + 4; j++) {
            if (j < 3) {
                picturePanel.add(createSpacer());
            } else {
                picturePanel.add(createHeaderCell(j - 3));
            }
        }
    }
}

/**
 * Формирует панель с черной границей
 */
private JPanel createCell(JLabel label) {
    JPanel panel = new JPanel(new BorderLayout());
    panel.setBorder(BorderFactory.createLineBorder(COLOR_BORDER, 1));
    panel.add(label, BorderLayout.CENTER);
    panel.setOpaque(false);
    label.setOpaque(false);
    label.setFont(plainFont);
    label.setHorizontalAlignment(SwingConstants.CENTER);
    return panel;
}

/**
 * Формирует ячейку в заголовке.
 */
private JPanel createHeaderCell(int i) {
    JLabel label = new JLabel("" + i);
    JPanel panel = createCell(label);
    label.setFont(headerFont);
    return panel;
}

/**
 * Формирует пустую панель
 */
private JPanel createSpacer() {
    JPanel panel = new JPanel();
    panel.setOpaque(false);
    return panel;
}

```

```

/**
 * Осуществляет перерисовку
 */
public void redraw() {
    // Перерисовываем общую картину
    for (int i = 0; i < globals.K + 1; i++) {
        for (int j = 0; j < globals.N + 1; j++) {
            T[i][j].setText(globals.T[i][j] != -1 ? "" + globals.T[i][j] : "");
        }
    }
    for (int i = 0; i < globals.K; i++) {
        if (globals.positions.contains(new Integer(i))) {
            selectCell(M[i], COLOR_CURRENT);
        } else {
            deselectCell(M[i]);
        }
    }
    // Делаем предыдущие подсвеченные ячейки обычными
    deselectCell(previous);
    deselectCell(prev_sel1);
    deselectCell(prev_sel2);
    String newMessage = null;
    // В зависимости от состояния автомата формируем подсвеченные ячейки и сообщение
    switch (globals.state) {
        case 0:
            newMessage = MESSAGE_STATE_0;
            break;
        case 1:
            newMessage = MessageFormat.format(MESSAGE_STATE_1,
                new Object[]{new Integer(globals.j)});
            selectCell(previous = T[globals.i][globals.j], COLOR_CURRENT);
            break;
        case 2:
            newMessage = MessageFormat.format(MESSAGE_STATE_2,
                new Object[]{new Integer(globals.i)});
            selectCell(previous = T[globals.i][globals.j], COLOR_CURRENT);
            break;
        case 4:
            if (globals.j >= globals.M[globals.i - 1]) {
                newMessage = MessageFormat.format(MESSAGE_STATE_4_ALT, new Object[]{
                    new Integer(globals.i), new Integer(globals.j)
                });
            } else {
                newMessage = MessageFormat.format(MESSAGE_STATE_4, new Object[]{
                    new Integer(globals.i), new Integer(globals.j)
                });
            }
            selectCell(previous = T[globals.i][globals.j], COLOR_CURRENT);
            selectCell(prev_sel1 = T[globals.i - 1][globals.j], COLOR_SELECT);
            if (globals.j > globals.M[globals.i - 1]) {
                selectCell(prev_sel2 = T[globals.i - 1][globals.j
                    - globals.M[globals.i - 1]], COLOR_SELECT);
            }
            break;
        case 5:
            if (globals.T[globals.K][globals.N] == 1) {
                newMessage = MESSAGE_STATE_5_OK;
            } else {
                newMessage = MESSAGE_STATE_5_FAIL;
            }
            selectCell(previous = T[globals.i - 1][globals.sum], COLOR_CURRENT);
            break;
        case 7:
            if (globals.j >= globals.M[globals.i - 1]) {
                newMessage = MESSAGE_STATE_7_ALT;
            } else {
                newMessage = MESSAGE_STATE_7;
            }
            selectCell(previous = T[globals.i][globals.sum], COLOR_CURRENT);
            selectCell(prev_sel1 = T[globals.i - 1][globals.sum], COLOR_SELECT);
    }
}

```

```

        if (globals.sum >= globals.M[globals.i - 1]) {
            selectCell(prev_sel2 = T[globals.i - 1][globals.sum
                - globals.M[globals.i - 1]], COLOR_SELECT);
        }
        break;
    case 8:
        newMessage = MESSAGE_STATE_8;
        selectCell(previous = T[globals.i - 1][globals.sum], COLOR_CURRENT);
        break;
    case 9:
        newMessage = MESSAGE_STATE_9;
        break;
    case 10:
        String itemsString = "[";
        boolean first = true;
        for (Iterator iterator = globals.positions.iterator();
            iterator.hasNext();) {
            Integer pos = (Integer) iterator.next();
            if (first) {
                first = false;
            } else {
                itemsString += ", ";
            }
            itemsString += globals.M[pos.intValue()];
        }
        itemsString += "]";
        newMessage = MessageFormat.format(MESSAGE_STATE_10,
            new Object[]{itemsString});

        break;
    }
    messageLabel.setText(newMessage);
}

/**
 * Выделяем ячейку определенным цветом
 */
private void selectCell(JLabel cell, Color color) {
    cell.setOpaque(true);
    cell.setBackground(color);
    cell.revalidate();
    cell.repaint();
}

private void deselectCell(JLabel cell) {
    if (cell != null) {
        cell.setOpaque(false);
        cell.revalidate();
        cell.repaint();
    }
}
}
}

```



## Приложение 5. Главная панель визуализатора

```
/*
 * Date: Jan 10, 2004
 * Copyright (c) 2004 Matvey Kazakov.
 */
package ru.ifmo.ips.visualizers.backpack;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.text.MessageFormat;

/**
 * Главный класс визуализатора
 *
 * @author Matvey Kazakov
 */
public class Visio extends JPanel implements ActionListener {

    // Надписи на кнопках
    private static final String LABEL_AUTO = "Авто";
    private static final String LABEL_STOP = "Стоп";
    private static final String LABEL_STEP = ">>";
    private static final String LABEL_RESTART = "Заново";
    private static final String LABEL_DELAYUP = ">";
    private static final String LABEL_DELAYDOWN = "<";
    private static final String LABEL_DELAY = "Пауза: {0}";

    // Кнопки
    private JButton buttonStep = new JButton(LABEL_STEP);
    private JButton buttonRestart = new JButton(LABEL_RESTART);
    private JButton buttonAuto = new JButton(LABEL_AUTO);
    private JButton buttonDelayDown = new JButton(LABEL_DELAYDOWN);
    private JButton buttonDelayUp = new JButton(LABEL_DELAYUP);
    private JLabel labelDelay = new JLabel();

    // Паузы
    private static final int INITIAL_DELAY = 4;
    private static final int[] DELAYS = new int[]{10, 20, 50, 100, 250, 500, 1000};

    // Глобальные переменные
    private Globals globals = new Globals(new int[]{4, 5, 3, 7, 6}, 16);
    // Автомат
    private Automaton automaton = new Automaton(globals);
    // Формирователь иллюстраций и комментариев
    private Drawer drawer = new Drawer(globals);

    private int delay = INITIAL_DELAY;
    private boolean auto = false;

    // Нить, поддерживающая автоматическое выполнение шагов
    private Thread autoThread = new AutoThread();

    public static void main(String[] args) {
        JFrame frame = new JFrame("Задача о рюкзаке");
        Visio visio = new Visio();
        visio.start();
        frame.setContentPane(visio);
        frame.pack();
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.show();
    }
}
```

```

/**
 * Формирование панели управления
 */
public Visio() {
    setBorder(BorderFactory.createLineBorder(getBackground(), 5));
    setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
    JPanel buttonPanel = new JPanel();
    buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.X_AXIS));
    add(drawer);
    buttonStep.addActionListener(this);
    buttonRestart.addActionListener(this);
    buttonAuto.addActionListener(this);
    buttonDelayDown.addActionListener(this);
    buttonDelayUp.addActionListener(this);

    buttonPanel.add(buttonStep);
    buttonPanel.add(Box.createHorizontalStrut(10));
    buttonPanel.add(buttonRestart);
    buttonPanel.add(Box.createHorizontalStrut(10));
    buttonPanel.add(Box.createHorizontalGlue());

    buttonPanel.add(buttonAuto);
    buttonPanel.add(Box.createHorizontalStrut(10));
    buttonDelayDown.setMargin(new Insets(0, 3, 0, 3));
    buttonDelayUp.setMargin(new Insets(0, 3, 0, 3));
    buttonPanel.add(buttonDelayDown);
    buttonPanel.add(Box.createHorizontalStrut(5));
    buttonPanel.add(labelDelay);
    buttonPanel.add(Box.createHorizontalStrut(5));
    buttonPanel.add(buttonDelayUp);

    updateLabelDelay();

    add(Box.createVerticalStrut(10));
    add(buttonPanel);
    drawer.redraw();
}

/**
 * Стартует поток автоматического перехода
 */
public void start() {
    autoThread.start();
}

/**
 * Останавливает поток автоматического перехода
 */
public void stop() {
    autoThread.interrupt();
}

/**
 * Реакция на нажатие кнопок
 */
public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    if (source == buttonStep) {
        automaton.makeStep();
        drawer.redraw();
        updateButtonStep();
        updateAutoButton();
    } else if (source == buttonRestart) {
        globals.init();
        auto = false;
        updateAutoButton();
        drawer.redraw();
        updateButtonStep();
    } else if (source == buttonAuto) {
        auto = !auto;
        updateAutoButton();
    }
}

```

```

    } else if (source == buttonDelayDown) {
        if (delay > 0) {
            delay--;
            updateLabelDelay();
            autoThread.interrupt();
        }
        updateDelayButtonsState();
    } else if (source == buttonDelayUp) {
        if (delay < DELAYS.length) {
            delay++;
            updateLabelDelay();
            autoThread.interrupt();
        }
        updateDelayButtonsState();
    }
}

// ===== ФУНКЦИИ ОБНОВЛЕНИЯ СОСТОЯНИЯ КНОПОК =====

private void updateButtonStep() {
    buttonStep.setEnabled(!automaton.isFinished());
}

private void updateDelayButtonsState() {
    buttonDelayDown.setEnabled(delay > 0);
    buttonDelayUp.setEnabled(delay < DELAYS.length);
}

private void updateLabelDelay() {
    labelDelay.setText(MessageFormat.format(LABEL_DELAY,
        new Object[]{"" + DELAYS[delay]}));
}

private void updateAutoButton() {
    if (automaton.isFinished()) {
        auto = false;
    }
    buttonAuto.setText(auto ? LABEL_STOP : LABEL_AUTO);
}

/**
 * Нить, в случае автоматического режима через определенные интервалы иницилирующая
 * шаг визуализатора.
 */
private class AutoThread extends Thread {
    public AutoThread() {
        super("Auto thread");
        setDaemon(true);
    }

    public void run() {
        while (true) {
            if (auto) {
                actionPerformed(new ActionEvent(buttonStep, 0, ""));
            }
            try {
                sleep(DELAYS[delay]);
            } catch (InterruptedException e) {
            }
        }
    }
}
}

```

## Приложение 6. Апплет

```
/**
 * Date: 02.07.2004
 * Copyright (c) Matvey Kazakov 2004.
 */

package ru.ifmo.ips.visualizers.backpack;

import com.sun.java.swing.plaf.windows.WindowsLookAndFeel;

import javax.swing.*;
import java.applet.Applet;
import java.awt.*;

/**
 * Класс реализует апплет содержащий визуализатор.
 *
 * @author Matvey Kazakov
 */
public class VisioApplet extends Applet {
    private Visio visio;

    public void init() {
        visio = new Visio();
        setLayout(new BorderLayout());
        // добавляем апплет
        add(visio, BorderLayout.CENTER);
        // устанавливаем Windows Look & Feel
        try {
            UIManager.setLookAndFeel(new WindowsLookAndFeel());
            SwingUtilities.updateComponentTreeUI(this);
        } catch (UnsupportedLookAndFeelException e) {
        }
    }

    public void start() {
        visio.start();
    }

    public void stop() {
        visio.stop();
    }
}
```

## Приложение 7. Сценарий для Apache Ant (build.xml)

```
<project name="Visio" basedir="." default="build">

  <target name="build" description="Main build">
    <mkdir dir="deploy"/>
    <mkdir dir="classes"/>
    <javac destdir="classes" srcdir="src"/>
    <jar jarfile="deploy/visio.jar">
      <manifest id="MANIFEST.MF">
        <attribute name="Specification-Title" value="Backpack Visualizer"/>
        <attribute name="Specification-Version" value="1.0"/>
        <attribute name="Specification-Vendor" value="Matvey Kazakov"/>
        <attribute name="Implementation-Title" value="common"/>
        <attribute name="Implementation-Version" value="1.0"/>
        <attribute name="Implementation-Vendor" value="Matvey Kazakov"/>
        <attribute name="Main-Class"
          value="ru.ifmo.ips.visualizers.backpack.Visio"/>
      </manifest>
      <fileset dir="classes"/>
    </jar>
    <zip zipfile="deploy/src.zip">
      <fileset dir="src"/>
      <fileset file="build.xml"/>
      <fileset file="visio.html"/>
      <fileset file="run.bat"/>
    </zip>
    <copy file="run.bat" tofile="deploy/run.bat"/>
    <copy file="visio.html" tofile="deploy/visio.html"/>
  </target>

  <target name="clean" description="Cleans all">
    <delete dir="classes"/>
    <delete dir="deploy"/>
  </target>

</project>
```