Санкт-Петербургский государственный университет
информационных технологий механики и оптики

Кафедра «Компьютерные технологии»

В.А. Добровольский, А.В. Степук

Визуализатор алгоритма сортировки подсчетом

Проектная документация

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

Санкт-Петербург
2004

Оглавление

Введение 3

1. Постановка задачи 3

2. Решение задачи 3

3. Выбор визуализируемых переменных 4

4. Анализ алгоритма для его визуализации 4

5. Алгоритм решения задачи 4

6. Реализация алгоритма на языке *Java* 4

7. Построение схемы алгоритма по программе . . . 5

8. Преобразование схемы алгоритма в граф переходов автомата Мили 7

9. Формирование набора невизуализируемых переходов . . 8

10. Выбор интерфейса визуализатора 9

11. Сопоставление иллюстраций и комментариев с состояниями автомата 10

12. Архитектура программы визуализатора 13

13. Программная реализация визуализатора 13

Заключение 15

Литература 15

Приложение. Исходный текст визуализатора 15

Введение

В работах, приведенных на сайте [1], показана целесообразность применения автоматного программирования для построения визуализаторов.

Визуализатор отображает пошаговое выполнение алгоритма, передвигаясь от состояния к состоянию. Такая логика соответствует работе автомата.

Целью данной работы является построение визуализатора алгоритма сортировки подсчетом на основе автоматного программирования [2].

1. Постановка задачи

Отсортировать за линейное время последовательность, в которой каждый элемент – целое положительное число в известном диапазоне (не превосходящее заранее известного k).

В реализации визуализатора k равно десяти.

2. Решение задачи

Приведем словесную формулировку решения этой задачи [3].

Идея алгоритма состоит в том, чтобы для каждого элемента x предварительно подсчитать, сколько элементов входной последовательности меньше x. После этого x записывается напрямую в выходной массив в соответствии с этим числом. Например, если семнадцать элементов входного массива меньше x, то в выходном массиве x должен быть записан на место с номером восемнадцать.

Далее в тексте и в самом визуализаторе используются следующие обозначения:

$A[1..n]$ – входная последовательность;

$C[1..k]$ – вспомогательный массив из k элементов;

$B[1..n]$ – выходная отсортированная последовательность.

Работа алгоритма заключается в следующем.

В элемент $C[i]$ заносится количество элементов массива A, равных i. Затем находятся частичные суммы последовательности $C[1]$, $C[2]$,…, $C[k]$, каждая из которых является количеством элементов, не превосходящих i.

После этого каждый элемент $A[i]$ из входного массива помещается в выходной массив B на позицию, равную значению элемента $C[A[i]]$.

3. Выбор визуализируемых переменных

Перечислим переменные, которые содержат значения, необходимые для визуализации:

$A[]$ – значения входного массива;

$B[]$ – значения выходного массива;

$C[]$ – значения промежуточного массива;

j – текущая позиция в массиве A;

i – текущая позиция в массиве C;

m – текущая позиция в массиве B.

4. Анализ алгоритма для его визуализации

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

- показать массив *A*, заполненный начальными значениями;
- отразить ход решения задачи – процесс заполнения массива *C*, основанный на значениях элементов массива *A*;
- показать подсчет частичных сумм в массиве *C*;
- отразить процесс последнего этапа решения задачи – заполнения массива *B*.

5. Алгоритм решения задачи

Приведем алгоритм решения задачи на псевдокоде [3]:

```
1     for i ß 1 to k
2           do C [ i ] ß 0
3     for j ß 1 to length[A]
4           do C[A[i]] = C[A[i]] + 1
5     for i ß 2 to K
6           do C[i] = C[I] +C[I-1]
7     for m ß length[a] downto 1
8           do B[C[A[m]]] ßA[m]
9              C[A[m]]ßC[A[m]]-1
```

6. Реализация алгоритма на языке *Java*

Перепишем программу на псевдокоде с помощью языка *Java.*

```
/**
 * @param a входной массив
 * @param K – максимальное значение для элементов A
 * @return отсортированный массив,
 */
private static Integer[] solve(int K, int[] a) {
    // Переменные циклов
    int s, i, j, m;
    // Количество предметов
    int N = M.length;
    // Вспомогательный массив c[]
    int[] c = new int[K];
    // Выходной массив b[]
   int[] b = new int[N];
   // Заполнение нулями массива c[]
   for(int s = 0; s < K; s++)
         c[s] = 0;
  // Заполнение массива c[]
  for (j = 0; j < N; j++)
         c[a[j] - 1] = c[a[j] – 1] + 1;
  // Подсчет частичных сумм c[]
  for(i = 1; i < K; i++)
         c[i] = c[i] + c[i-1];
  // Формирование выходного массива b[]
  for(m = N - 1; m >= 0; m--)
  {
         b[c[a[m]-1] = a[m];
```

```
        c[a[m] – 1] = c[a[m] – 1] – 1;
}
```

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).

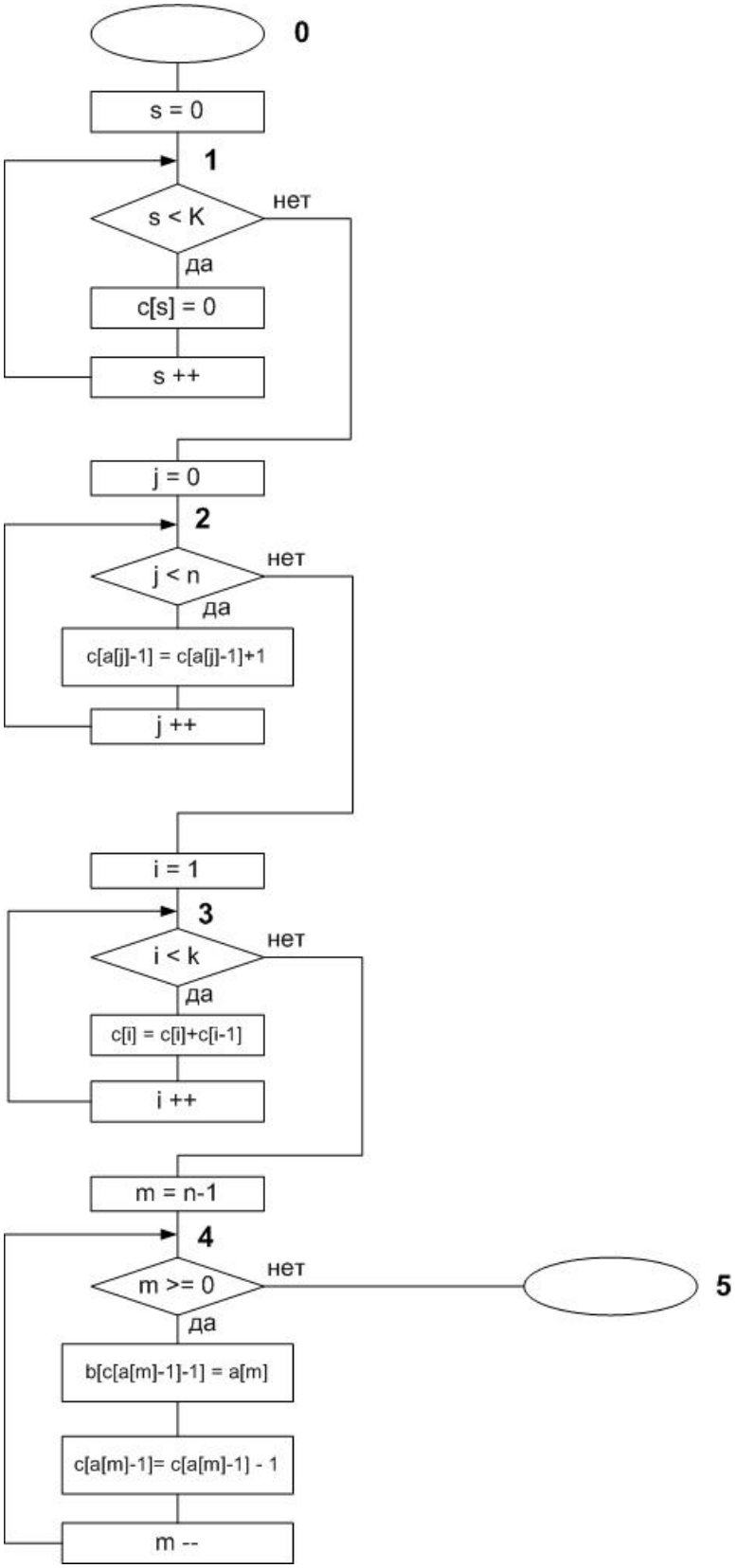

Рис. 1. Схема алгоритма

8. Преобразование схемы алгоритма в граф переходов автомата Мили

В работе [4] 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 [4], 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.

Исходя из изложенного, для схемы алгоритма на рис. 1 выделим шесть состояний автомата с номерами 0 – 5. Таким образом, автомат визуализации будет иметь шесть состояний, а его граф переходов, который строится за счет определения путей между выделенными точками на схеме алгоритма – шесть вершин (рис. 2).

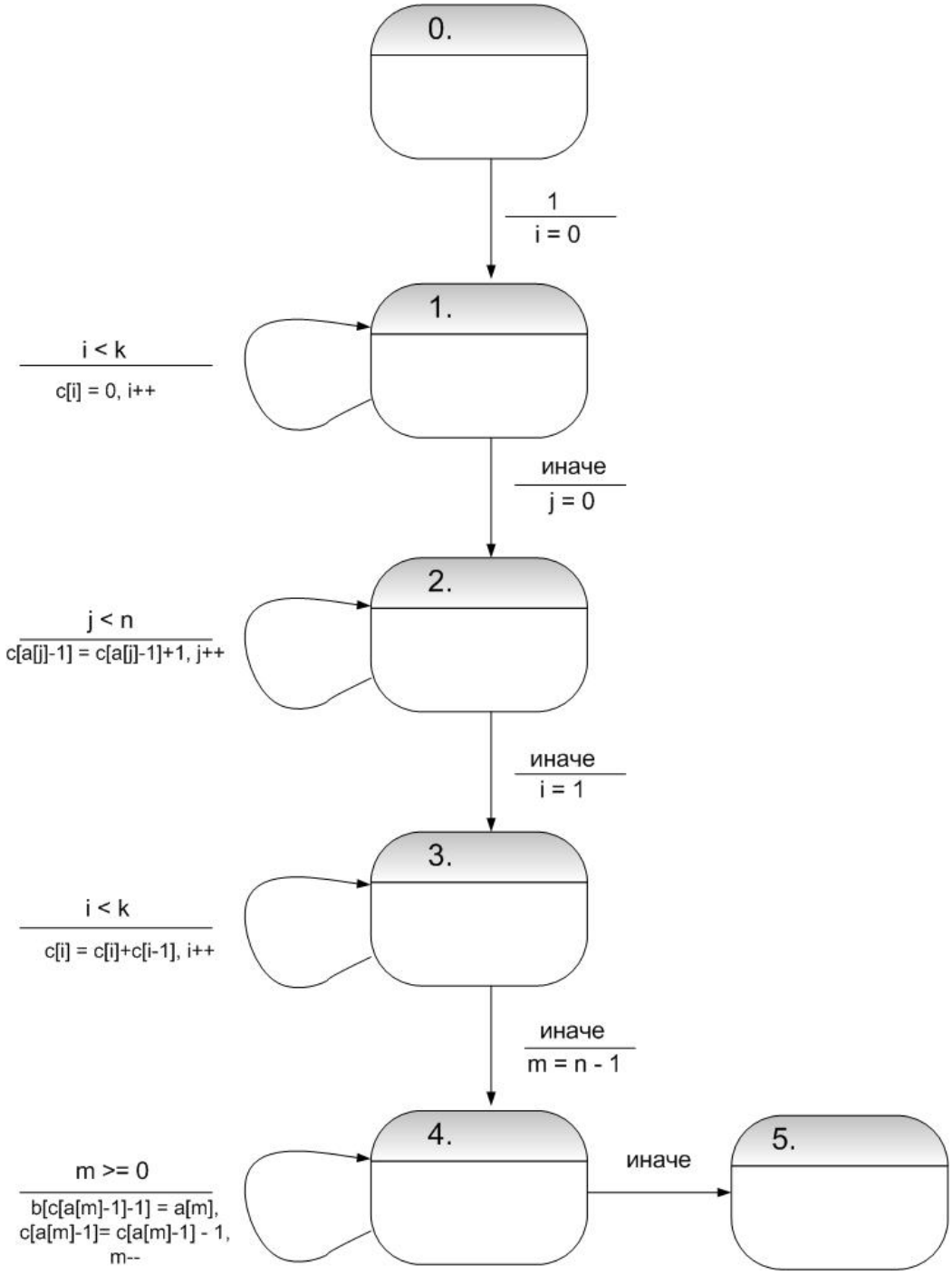

Рис. 2. Граф переходов автомата, реализующего сортировку подсчетом

9. Формирование набора невизуализируемых переходов

Из анализа графа переходов (рис. 2) следует, все переходы будут визуализируемыми.

10. Выбор интерфейса визуализатора

В верхней части визуализатора представляется следующая информация:

- значения массивов *A*[], *B*[], *C*[];
- текущие индексы активных элементов массива.

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

- «>>» – сделать шаг алгоритма;
- «Рестарт» – начать алгоритм заново;
- «Случайно» – начать алгоритм заново с новым произвольным набором данных;
- «Авто» – перейти в автоматический режим;
- «Стоп» – оставить работы в автоматическом режиме и перейти в пошаговый режим;
- «<», «>» – изменение паузы между автоматическими шагами алгоритма.

Помимо кнопок в нижней части расположена надпись «Пауза», отображающая величину задержки в автоматическом режиме.

Среднюю часть визуализатора занимает область комментариев, в которой на каждом шаге отображается описание выполняемого алгоритмом действия.

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

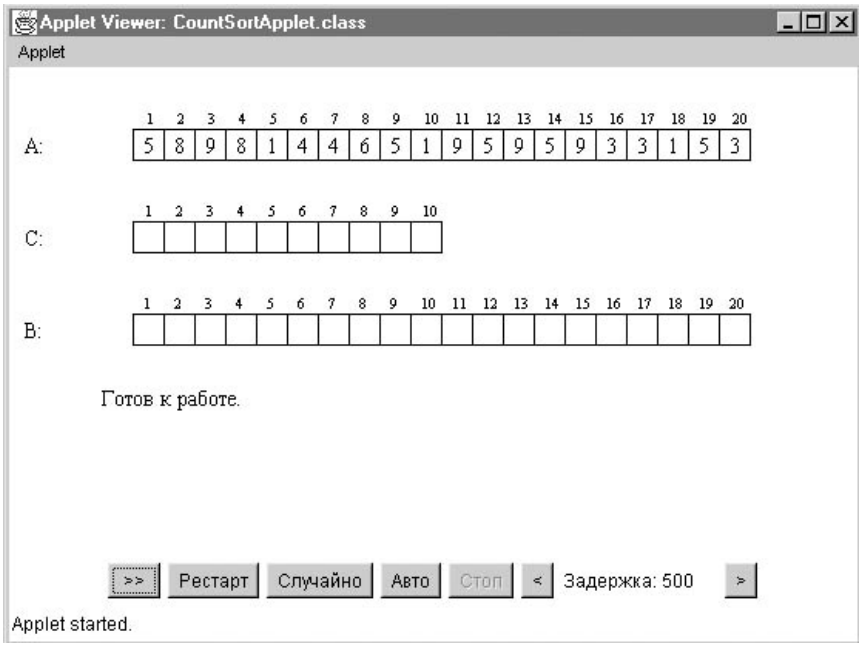

Рис. 3. Состояние 0

11. Сопоставление иллюстраций и комментариев с состояниями автомата

Так как при построении визуализатора используется автомат Мили, то будем в рассматриваемом состоянии отображать действия, которые выполняются при переходе из него. При этом зеленым, желтым или красным цветом выделяются активные ячейки.

Перечислим состояния автомата визуализации.

Состояние 0. «Готов к работе» (рис.3).

Состояние 1. «Заполнить нулями массив *C*» (рис.4).

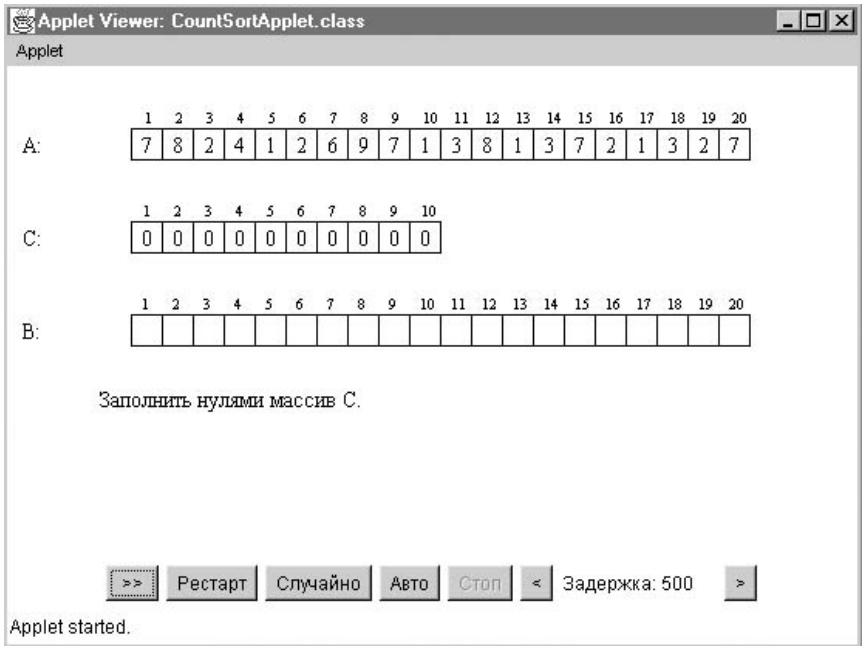

Рис. 4. Состояние 1

Состояние 2. «В массиве *C* увеличить на единицу *A*[*j*]-й элемент» (рис. 5).

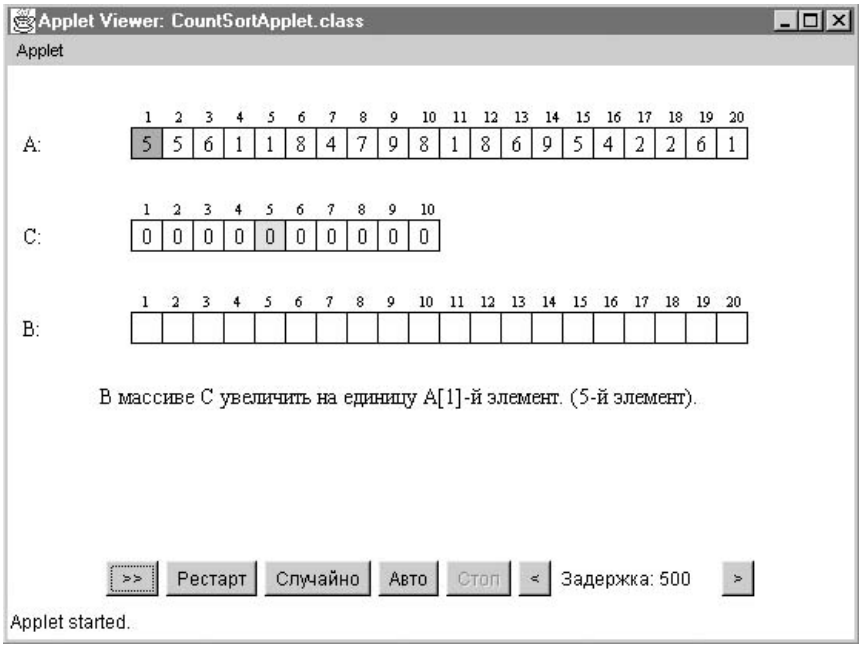

Рис. 5. Состояние 2

Состояние 3. «В *m*-й элемент массива *C* записывается сумма *m*-го и *m*+1-го элементов» (рис.6).

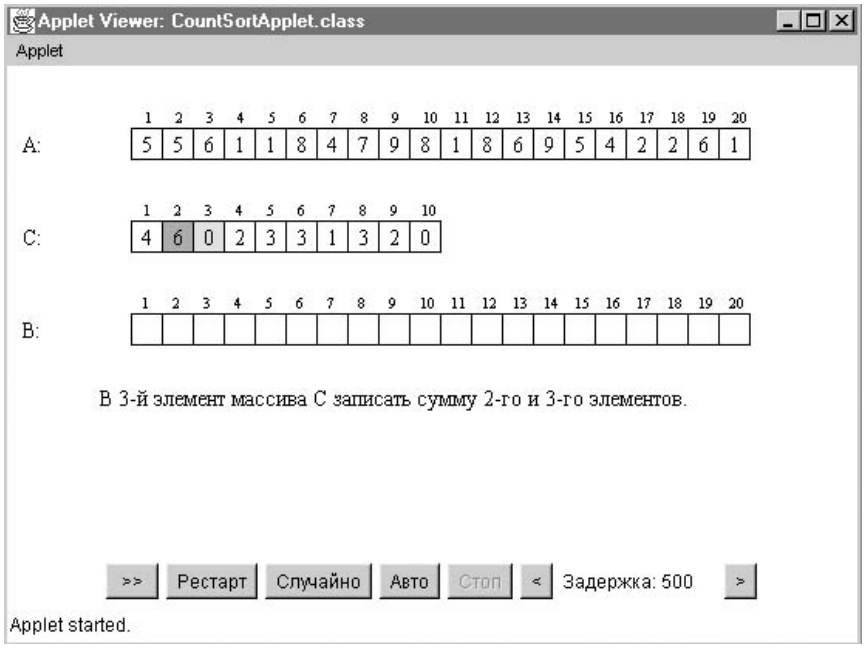

Рис. 6. Состояние 3

Состояние 4. «Найти значение в $A[m]$-го элемента массива C. Занести $A[m]$ на позицию с этим значением в выходной массив B» (рис.7).

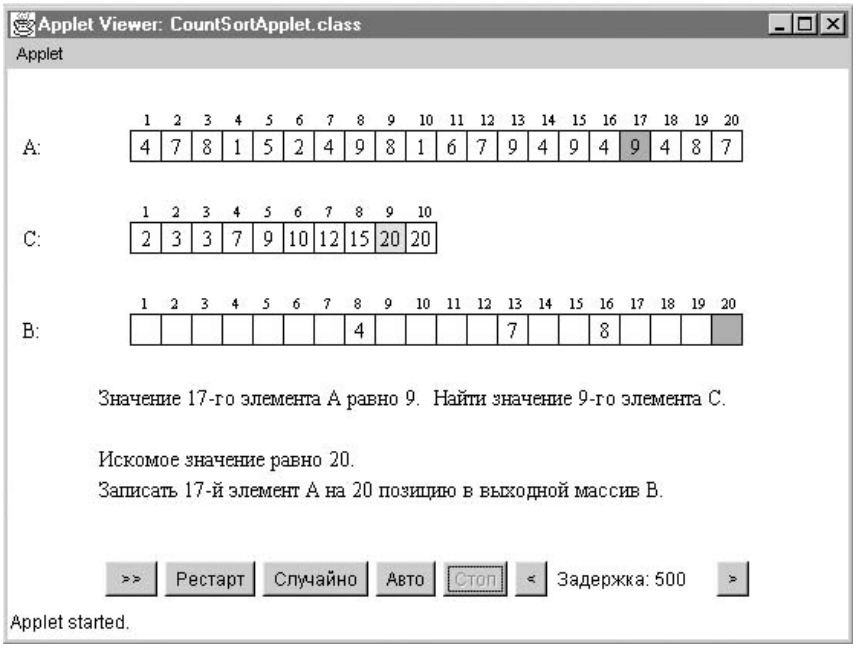

Рис. 7. Состояние 4

Состояние 5. «Массив отсортирован. Работа алгоритма завершена» (рис.8).

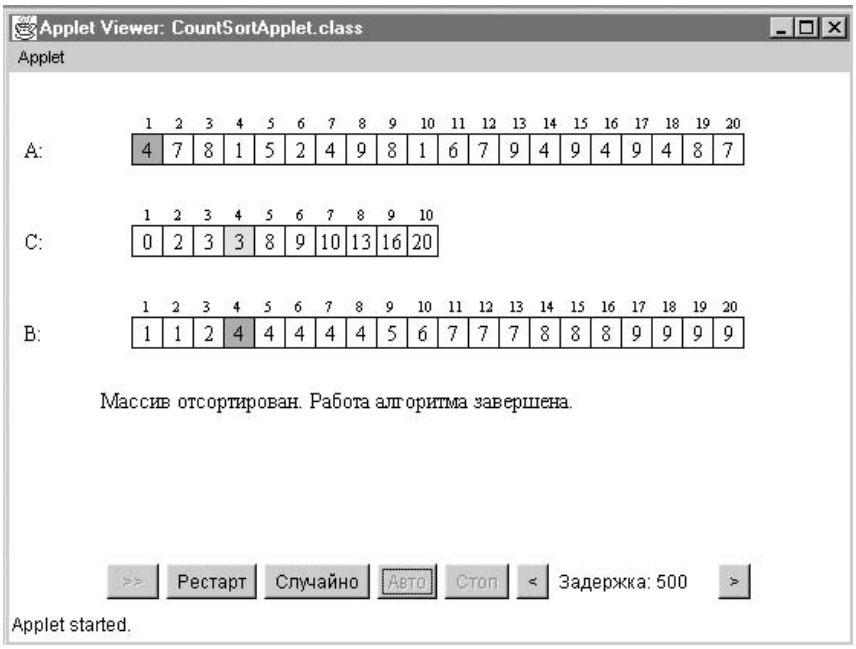

Рис. 8. Состояние 5

12. Архитектура программы визуализатора

Программа состоит из четырех классов:

- `CSControls` – отвечает за действия пользователей;
- `CSCanvas` – реализует работу автомата и отрисовку графики;
- `CountSortApplet` – объединяет части программы в запускаемый апплет;
- `AutoThread` – нить, обеспечивающая выполнение шагов в автоматическом режиме.

Диаграмма классов приведена на рис. 9.

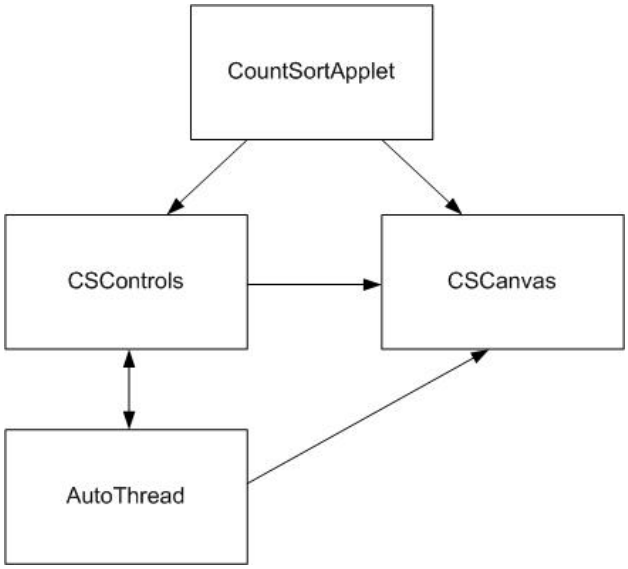

Рис. 9. Диаграмма классов визуализатора

13. Программная реализация визуализатора

В работе [2] 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 программа, приведенная ниже:

```
switch (state)
{
  case 0:                    // Начальное состояние
    state = 1; i0 = 0;
  for (int step = 0; step < N; step++)
      b[step] = -1;

  for (int step = 0; step < K; step++)
      c[step] = 0;
  break;

  case 1:                    // Заполнение нулями массива C
         for (i0 = 0; i0 < K; i0++)
           c[i0] = 0;
         state = 2;
```

```
j = 0;
       break;

case 2:                    // Поэлементное заполнение массива C
        if (j < N) {
          c[a[j]-1] = c[a[j]-1] + 1;
          j++;
        }
        else {
          state = 3;
          i = 1;
        }
        break;

      case 3:                    // Преобразование массива C
        if (i < K) {
       c[i] = c[i] + c[i-1];
       i++;
        }
        else {
       state = 4;
             m = N - 1;
        }
        break;

      case 4:                    // Формирование массива B
        if (m  >= 0){
       b[c[a[m]-1]-1] = a[m];
       c[a[m]-1] = c[a[m]-1] - 1;
       l--;
 }
        else {
       state = 5;
        }
        break;

      case 5:                    // Конечное состояние
        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` (см. приложение). 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.

Полный исходный текст программы визуализатора приведен на сайте http://is.ifmo.ru/ в разделе «Визуализаторы».

Заключение

Использование автоматного подхода для создания визуализаторов была развита в работах на сайте [1].

Данная работа подтверждает успешность этого направления.

Литература

1. *Сайт кафедры «Технологии программирования»*. http://is.ifmo.ru

2. *Казаков М.А., Шалыто А.А.* Использование автоматного программирования для реализации визуализаторов //Компьютерные инструменты в образовании. 2004. № 2.

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

4. *Шалыто А.А., Туккель Н.И.* Преобразование итеративных алгоритмов в автоматные //Программирование. 2002. № 5. С.12-26. http://is.ifmo.ru, раздел «Статьи».

Приложение. Исходный текст визуализатора

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import java.text.MessageFormat;

// Класс апплета
public class CountSortApplet extends Applet{
  CSControls controls;
  CSCanvas canvas;

  public void init(){
  setLayout(new BorderLayout());
  canvas = new CSCanvas();
  add("Center", canvas);
  add("South", controls = new CSControls(canvas));
  canvas.init();
}

public void destroy() {
  remove(controls);
  remove(canvas);
}

public void start() {
  controls.setEnabled(true);
  controls.start();
}

public void stop() {
  controls.setEnabled(false);
  controls.stop();
}
```

```
public void processEvent(AWTEvent e) {
  if (e.getID() == Event.WINDOW_DESTROY) {
      System.exit(0);
  }
}

public static void main(String args[]) {
  Frame f = new Frame("CountSort");
  CountSortApplet csApplet = new CountSortApplet();

  csApplet.init();
  csApplet.start();

  f.add("Center", csApplet);
  f.setSize(300, 200);
  f.show();
}

public String getAppletInfo() {
  return "Count Sort visualisation Applet.";
}
}

// Класс, реализующий отрисовку
class CSCanvas extends Canvas {

    int N = 20;
    int K = 10;

    int i0, i, j, m;

    int a[], b[], c[];
    int vis_stage;

    int astep, bstep, cstep;
    int state = 0;
    String comment = "";
    String comment2 = "";
    String comment3 = "";

    public void init(){
      a = new int [N];
      b = new int [N];
      c = new int [K];

      for(int step = 0; step < N; step++)
        a[step] = (int)((K-1) * Math.random())+1;

      for(int step = 0; step < N; step++)
         b[step] = -1;
      for(int step = 0; step < K; step++)
         c[step] = 0;

      vis_stage = 0;


      astep = -1;
      bstep = -1;
      cstep = -1;

      comment = "Готов к работе.";
      comment2 = "";
      comment3 = "";
    }

    public void restart(){
      state = 0;
```

```
  this.init();
  this.repaint();
}

public void randomStart(){
  a = new int[N];
  for(int step = 0; step < N; step++)
    a[step] = (int)((K-1) * Math.random())+1;

  this.restart();
}


public void paint(Graphics g) {
  int xi;
  int boxh, boxw, x0, y0;
  String str = "";

  boxw = 20; boxh = 20;

  Color activeColor = new Color(255, 140, 140);
  Color activeColor2 = new Color(120, 255, 120);
  Color mainColor = new Color(0, 0, 0);
  g.setColor(mainColor);
  Font font = new Font("Times New Roman", Font.PLAIN, 14);
  g.setFont(font);

  // Массив A:
  x0 = 80;  y0 = 40;
  for(xi = 0; xi < N; xi++)
  {
    if (xi == astep)
    {
      g.setColor(activeColor);
      g.fillRect(x0, y0, boxw, boxh);
      g.setColor(mainColor);
    }

    g.drawRect(x0, y0, boxw, boxh);
    str = Integer.toString(a[xi]);
    g.drawString(str, x0 + boxw/2 - str.length()*font.getSize()/4,
                 y0 + boxh/2 + font.getSize()/2 - 1);
    x0 += boxw;
  }

  g.drawString("A:", 10, y0 + boxh/2 + font.getSize()/2 - 1);

  // Массив C:
  x0 = 80; y0 = 100;
  for(xi = 0; xi < K; xi++)
  {

      if (xi == cstep)
      {
        g.setColor(activeColor2);
        g.fillRect(x0, y0, boxw, boxh);
        g.setColor(mainColor);
      }
    if (vis_stage == 1)
      if (xi == (cstep-1))
      {
        g.setColor(activeColor);
        g.fillRect(x0, y0, boxw, boxh);
        g.setColor(mainColor);
      }


    g.drawRect(x0, y0, boxw, boxh);
```

```
    if (state > 0)
    {
      str = Integer.toString(c[xi]);
      g.drawString(str, x0 + boxw/2 - str.length()*font.getSize()/4,
                 y0 + boxh/2 + font.getSize()/2 - 1);
    }
    x0 += boxw;
  }
  g.drawString("C:", 10, y0 + boxh/2 + font.getSize()/2 - 1);


  // Массив B:
  x0 = 80; y0 = 160;
  for(xi = 0; xi < N; xi++)
  {
    if (xi == bstep)
    {
      g.setColor(activeColor);
      g.fillRect(x0, y0, boxw, boxh);
      g.setColor(mainColor);
    }

    g.drawRect(x0, y0, boxw, boxh);
    if (b[xi] > 0)
    {
      str = Integer.toString(b[xi]);
      g.drawString(str, x0 + boxw/2 - str.length()*font.getSize()/4,
                   y0 + boxh/2 + font.getSize()/2 - 1);
    }

    x0 += boxw;
  }
  g.drawString("B:", 10, y0 + boxh/2 + font.getSize()/2 - 1);


  Font fontLittle = new Font("Times New Roman", Font.PLAIN, 9);
  g.setFont(fontLittle);

  String sNum = "";
  x0 = 80; y0 = 40;
  for(xi = 0; xi < N; xi++){
    sNum = str.valueOf(xi + 1);
    g.drawString(sNum, x0 - 1 + boxw*xi + boxw/2, y0-3);
  }
  x0 = 80; y0 = 100;
  for(xi = 0; xi < K; xi++){
    sNum = str.valueOf(xi + 1);
    g.drawString(sNum, x0 - 2 + boxw*xi + boxw/2, y0-3);
  }
  x0 = 80; y0 = 160;
  for(xi = 0; xi < N; xi++){
    sNum = str.valueOf(xi + 1);
    g.drawString(sNum, x0 - 3 + boxw*xi + boxw/2, y0-3);
  }
  g.setFont(font);


  g.drawString(comment, 60, 220);
  g.drawString(comment2, 60, 260);
  g.drawString(comment3, 60, 280);
}

public void MakeStep() {

// Автомата
switch (state)
{
  case 0:
```

```
        state = 1; i0 = 0;
      for (int step = 0; step < N; step++)
          b[step] = -1;

      for (int step = 0; step < K; step++)
          c[step] = 0;
      break;

      case 1:
       for (i0 = 0; i0 < K; i0++)
         c[i0] = 0;
       comment = "Заполнить нулями массив C.";
       state = 2; j = 0;
       break;

      case 2:
        if (j < N)
        {
          c[a[j]-1] = c[a[j]-1] + 1;
          j++;

        }
        else
        {
          state = 3;
          i = 1;
        }

        break;

      case 3:
        if (i < K)  {c[i] = c[i] + c[i-1];     i++; }
        else        {state = 4;   m = N - 1; }
        break;

      case 4:
        if (m  >= 0)       {b[c[a[m]-1]-1] = a[m];  c[a[m]-1] = c[a[m]-1] -
1;  m--; }
        else        {state = 5;}
        break;

      case 5:
        break;
      }


    // Визуализация в состояниях автомата
   switch (state)
    {
      case 0:
        break;
      case 1:
       comment = "Заполнить нулями массив C.";
       break;

       case 2:
        if (j < N)
       {
          astep = j;
          cstep = a[j]-1;
          bstep = -1;
          String ss1 = Integer.toString(j+1);
          String ss2 = Integer.toString(cstep+1);

          comment = "В массиве C увеличить на единицу A[" + ss1+ "]-й
          элемент. (" + ss2 + "-й элемент).";
        }
```

```
        break;

      case 3:
        vis_stage = 1;

        if (i < K)
        {
          astep = -1;
          cstep = i;
          bstep = -1;
          if (i == 1)
            comment = "Во " + Integer.toString(cstep+1) + "-й элемент
массива C записать сумму " + Integer.toString(cstep) + "-го и " +
Integer.toString(cstep+1) + "-го элементов.";
          else
            comment = "В " + Integer.toString(cstep+1) + "-й элемент массива
C записать сумму " + Integer.toString(cstep) + "-го и " +
Integer.toString(cstep+1) + "-го элементов.";
        }
        else
          comment = "Массив C сформирован.";

        break;

      case 4:
        vis_stage = 2;

        if (m >= 0)
        {
          astep = m;
          cstep = a[m]-1;
          bstep = c[a[m]-1]-1;

          String ss1 = "Значение "+ Integer.toString(astep+1) + "-го
элемента A равно " + Integer.toString(a[m]) + ".  ";
          String ss2 = "Найти значение " + Integer.toString(a[m]) + "-го
элемента C.";
          comment = ss1 + ss2;
          comment2 = "Искомое значение равно " + Integer.toString(bstep+1) +
".";
          comment3 = "Записать " + Integer.toString(astep+1) +"-й элемент A
на " + Integer.toString(bstep+1) + " позицию в выходной массив B.";
        }
        else
        {
           comment = "Массив отсортирован. Работа алгоритма завершена.";
           comment2 = "";
           comment3 = "";
        }

        break;
       case 5:
         break;

      }


    repaint();


    } // Makestep

}

// Класс элементов управления
class CSControls extends Panel
                  implements ActionListener {
```

```
    CSCanvas canvas;

    private boolean auto = false;
    private Thread autoThread = new AutoThread();
    private static final int INITIAL_DELAY = 4;
    private static final int[] DELAYS = new int[]{100, 200, 250, 400, 500,
1000, 2000, 2500};
    private int delay = INITIAL_DELAY;

    Button buttonNext, buttonRestart, buttonAuto, buttonStop, buttonDelayUp,
buttonDelayDown;
    Button buttonRandomize;
    Label labelDelay;

    public CSControls(CSCanvas canvas) {
        this.canvas = canvas;

        buttonNext = new Button(" >> ");
        buttonRestart = new Button("Рестарт");
        buttonAuto = new Button("Авто");
        buttonStop = new Button("Стоп");
        buttonDelayDown = new Button("<");
        buttonDelayUp = new Button(">");

        labelDelay = new Label("Задержка: 500 ");
        buttonRandomize = new Button("Случайно");

        buttonNext.addActionListener(this);
        buttonRestart.addActionListener(this);
        buttonStop.addActionListener(this);
        buttonAuto.addActionListener(this);
        buttonDelayDown.addActionListener(this);
        buttonDelayUp.addActionListener(this);
        buttonRandomize.addActionListener(this);

        buttonNext.setSize(50,20);

        buttonStop.setEnabled(false);

        add(buttonNext);
        add(buttonRestart);
        add(buttonRandomize);
        add(buttonAuto);
        add(buttonStop);
        add(buttonDelayDown);
        add(labelDelay);
        add(buttonDelayUp);
    }

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

    public void stop(){
      auto = false;
      autoThread.interrupt();
    }


    public void actionPerformed(ActionEvent ev) {
        Object source = ev.getSource();
        String stLabel;
        if (source == buttonNext){
          canvas.MakeStep();
        }

        if (source == buttonAuto){
          auto = true;
```

```
      buttonStop.setEnabled(true);
      buttonAuto.setEnabled(false);
      buttonNext.setEnabled(false);
    }

    if (source == buttonStop){
      auto = false;
      buttonStop.setEnabled(false);
      buttonAuto.setEnabled(true);
      buttonNext.setEnabled(true);
    }

    if (source == buttonDelayDown){
      if (delay == 7)
        buttonDelayUp.setEnabled(true);

      if (delay > 0)
        delay--;
      else
        buttonDelayDown.setEnabled(false);

      String str = "";
      stLabel = "Задержка: " + str.valueOf(DELAYS[delay]);
      labelDelay.setText(stLabel);
    }

    if (source == buttonDelayUp){
      if (delay == 0)
        buttonDelayDown.setEnabled(true);

      if (delay < 7)
        delay++;
      else
        buttonDelayUp.setEnabled(false);

      String str = "";
      stLabel = "Задержка: " + str.valueOf(DELAYS[delay]);
      labelDelay.setText(stLabel);
    }

    if (source == buttonRestart) {
      if (canvas.state == 5){
        buttonAuto.setEnabled(true);
        buttonNext.setEnabled(true);
      }
      canvas.restart();
    }

    if (source == buttonRandomize) {
       if (canvas.state == 5){
         buttonAuto.setEnabled(true);
         buttonNext.setEnabled(true);
       }
       canvas.randomStart();
    }

}


 // Нить, в случае автоматического режима через определенные
 // интервалы инициирующая шаг визуализатора

 private class AutoThread extends Thread {
   public AutoThread() {
     super("Auto thread");
     setDaemon(true);
   }
```

```
      public void run() {
        while (true) {

          if (auto) {
                canvas.MakeStep();
                if (canvas.state == 5) {
                  auto = false;
                  buttonStop.setEnabled(false);
                  buttonNext.setEnabled(false);
                }
          }
            try {
              sleep(DELAYS[delay]);
            } catch (InterruptedException e) {}
        }
      }
    }
}
```