Статья опубликована в журнале «Открытое образование», 2005, № 1, с. 53-58.

Методы построения логики визуализаторов алгоритмов

***М.А. Казаков**, аспирант каф. компьютерных технологий*
Тел.: (812) 233-42-98. E-mail: kazakov@rain.ifmo.ru
***А.А. Шалыто**, д.т.н., проф., зав. каф. технологий программирования*
Тел.: (812) 247-95-45. E-mail: shalyto@mail.ifmo.ru
Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

В статье рассматриваются три метода построения логики визуализаторов алгоритмов: эвристический, непосредственное построение автомата по словесному описанию алгоритма, формальное построение автомата по схеме алгоритма. Для их сравнения все методы рассмотрены применительно к алгоритму пузырьковой сортировки.

The following article describes three methods to algorithm visualization logic implementation: heuristic, straight automata implantation, formal conversion from algorithm to automata. All methods are used for Bubble sorting algorithm visualization in order to compare them.

Введение

Одним из методических приемов [1 – 4] при обучении программированию и дискретной математике является разработка визуализаторов классических вычислительных алгоритмов [5, 6].

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-аплетов. 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. В результате учащийся освобождается от необходимости выполнять шаги алгоритма, так как их автоматически выполняет визуализатор.

Визуализатор выполняет обычно следующие функции.

1. Пошаговое выполнение алгоритма.
2. Просмотр действия алгоритма при разных наборах входных данных, в том числе и введенных пользователем.
3. Просмотр действия алгоритма в динамике.
4. Перезапуск 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.

Анализ литературы [7, 8], а также реализации визуализаторов [9 – 12] показал, что технологии разработки визуализаторов рассматриваемого класса обычно являются эвристическими.

Опыт построения визуализаторов [3, 4] свидетельствует о том, что при эвристическом подходе каждый алгоритм требует индивидуальной реализации. Другими словами, технология разработки визуализаторов рассматриваемого типа отсутствовала, и основные успехи в этой области были педагогическими [3, 4, 13].

При этом основным недостатком является то, что обычно при построении визуализатора используются только такие понятия, как «входные и выходные переменные», а понятие «состояние» в явном виде не используется. При традиционном подходе состояния задаются неявно, как значения внутренних переменных, которые часто называются «флагами». Однако, по мнению авторов, совершенно естественным кажется, что каждый шаг визуализации необходимо проводить в соответствующем явно выделенном состоянии или на переходе. Поэтому при построении программ визуализации целесообразно использовать технологию автоматного программирования, базирующуюся на конечных автоматах [14].

Простейший подход к использованию автоматов для построения визуализаторов, который, видимо, применим только для очень простых алгоритмов, состоит в непосредственном построении графа переходов по словесному описанию алгоритма.

В настоящей работе показано, что формальное построение графа переходов по программе превращает процесс разработки 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.

В настоящей работе на примере простого алгоритма «пузырьковая сортировка» выполнено сравнение трех методов построения логики визуализаторов. При этом рассматриваются следующие методы:

- эвристический (без использования автоматов для описания поведения визуализатора);
- непосредственное построение автомата по словесному описанию алгоритма;
- формальное построение автомата по схеме алгоритма.

В работе даны рекомендации по их использованию.

1. Алгоритм «пузырьковая сортировка»

Задан массив из n элементов, который требуется отсортировать по возрастанию. Слева направо поочередно сравниваются два соседних элемента (первый и второй), и если первый элемент больше второго, то они меняются местами. После этого берется следующая пара элементов (второй и третий) и она упорядочивается, как указано выше. Аналогичные действия выполняются для всех оставшихся пар. После первого прохода на последней позиции массива (n-ой позиции) будет стоять максимальный элемент (“Всплыл первый пузырек”). Поскольку максимальный элемент уже размещен, то во втором проходе выполняется сортировка ($n – 1$)-го

элемента. После этого сортируются $n - 2$ элементов и т.д. Таким образом, за $n - 1$ проход массив будет отсортирован.

2. Интерфейс визуализатора

На рис. 1 приведен интерфейс визуализатора, который используется для всех трех рассматриваемых в работе методов.

Рис. 1. Интерфейс визуализатора

3. Эвристический метод

Метод широко используется на практике и состоит в эвристическом построении по словесному алгоритму реализующей его программы с дальнейшим, также эвристическим ее преобразованием в визуализатор. При этом описание логики визуализаторов выполняется без применения конечных автоматов [4].

Программа, реализующая заданный алгоритм традиционным путем, приведена в листинге 1.

Листинг 1. Программа, реализующая алгоритм «пузырьковая сортировка» традиционным путем

```
public class Bubble {

    private static int a[] = {4, 3, 7, 1, 9, -2};
    private static int n = a.length;
    private static int i = 0;
    private static int j = 0;

    public static void main(String[] args) {
        for (int i = 1; i < n; i++) {
            for (int j = 1; j < n - i + 1; j++) {
                if (a[j-1] > a [j]) {
                    out();                          // Пошаговый вывод
                    swap(j,j-1);
                }
            }
        }
        out();
    }

    // Обмен и вывод

    private static void swap (int i, int j) {
        int t = a[i];
        a[i] = a[j];
        a[j] = t;
    }

    private static void out () {
        for (int i = 0; i < n; i++) {
```

```
            System.out.print("" + a[i] +" ");
        }
        System.out.println();
    }
}
```

Правильность работы этой программы подтверждается протоколом:

```
4 3 7 1 9 -2
3 4 7 1 9 -2
3 4 1 7 9 -2
3 4 1 7 -2 9

3 1 4 7 -2 9
3 1 4 -2 7 9

1 3 4 -2 7 9
1 3 -2 4 7 9

1 -2 3 4 7 9

-2 1 3 4 7 9
```

Отметим, что такой же протокол будут иметь программы, построенные на основе других методов.

В заключение раздела отметим, что для построения визуализатора в целом эту программу необходимо эвристически связать с пользовательским интерфейсом.

4. Автоматный метод

Суть этого метода состоит в том, что по словесной формулировке алгоритма строится непосредственно граф переходов автомата, задающего логику визуализатора. Привязка этой программы к интерфейсу осуществляется значительно проще, чем в предыдущем случае, так как в модели, описывающей логику визуализатора, присутствуют состояния, к которым «весьма легко» привязать визуализируемую информацию.

На рис.2 приведен граф переходов автомата, построенного непосредственно по словесной формулировке алгоритма.

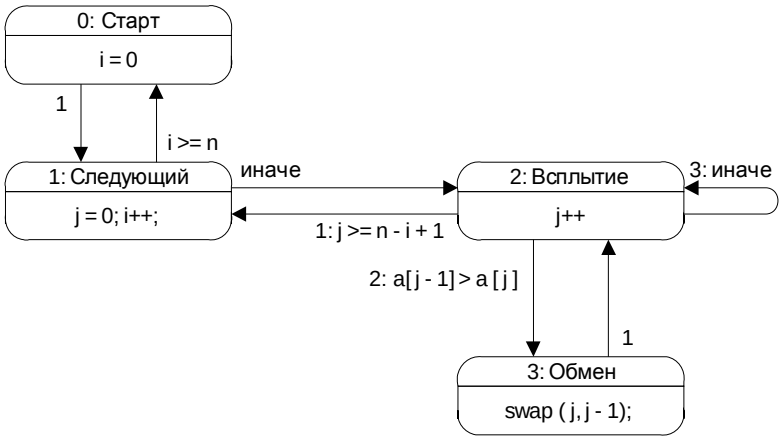

Рис. 2. Граф переходов, построенный непосредственно по словесному описанию алгоритма

Программа, изоморфная этому графу переходов, приведена в листинге 2.

Листинг 2. Программа, реализующая алгоритм «пузырьковая сортировка», построенная по графу переходов

```
public class BubbleA {

    public static final int STATE_INIT   = 0;
    public static final int STATE_NEXT   = 1;
    public static final int STATE_EMERGE = 2;
    public static final int STATE_SWAP   = 3;

    private static int state = STATE_INIT;
    private static int a[] = {4, 3, 7, 1, 9, -2};
    private static int n = a.length;
    private static int i = 0;
    private static int j = 0;

    public static void main(String[] args) {
        do {
            nextStep();
        } while (state != STATE_INIT);
        out();
    }

    private static void nextStep() {
        switch(state) {
            case STATE_INIT:
                i = 0;
                state = STATE_NEXT;
                break;

            case STATE_NEXT:
                j = 0;
                i++;

                if (i >= n) {
                    state = STATE_INIT;
                } else {
                    state = STATE_EMERGE;
                }
                break;

            case STATE_EMERGE:
                j++;

                if (j >= n - i + 1) {
                    state = STATE_NEXT;
                } else if (a[j-1] > a [j]) {
                    state = STATE_SWAP;
                }
                break;

            case STATE_SWAP:
                out();                          // Пошаговый вывод
                swap(j,j-1);
                state = STATE_EMERGE;
                break;
        }
    }

    // Обмен и вывод
    ...
}
```

Построение визуализатора по этой программе выполняется на основе подхода, как изложено в работе [15].

5. Формальное построение автомата по схеме алгоритма

Этот метод состоит в следующем:

- по словесному описанию алгоритма эвристически строится реализующая его программа;
- по программе эвристически строится схема алгоритма;
- по схеме алгоритма формально строится граф переходов конечного автомата;
- граф переходов формально и изоморфно реализуется соответствующей программой;
- в построенную программу, содержащую «состояния» вводятся компоненты, обеспечивающие визуализацию алгоритма.

Программа для рассматриваемого алгоритма приведена в листинге 1. На рис. 3 приведена схема алгоритма, построенная по этой программе.

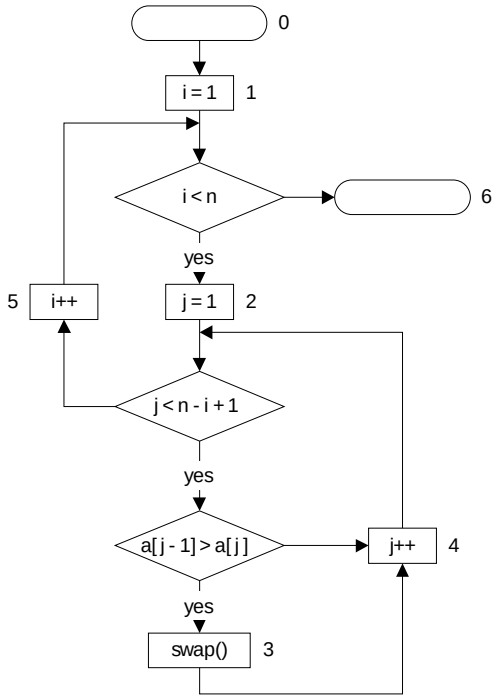

Рис. 3. Схема алгоритма «пузырьковая сортировка»

В работе [16] описан метод преобразования схемы алгоритма в граф переходов (в данном случае автомата Мура). На рис. 4 приведен этот граф, построенный на основе указанного метода.

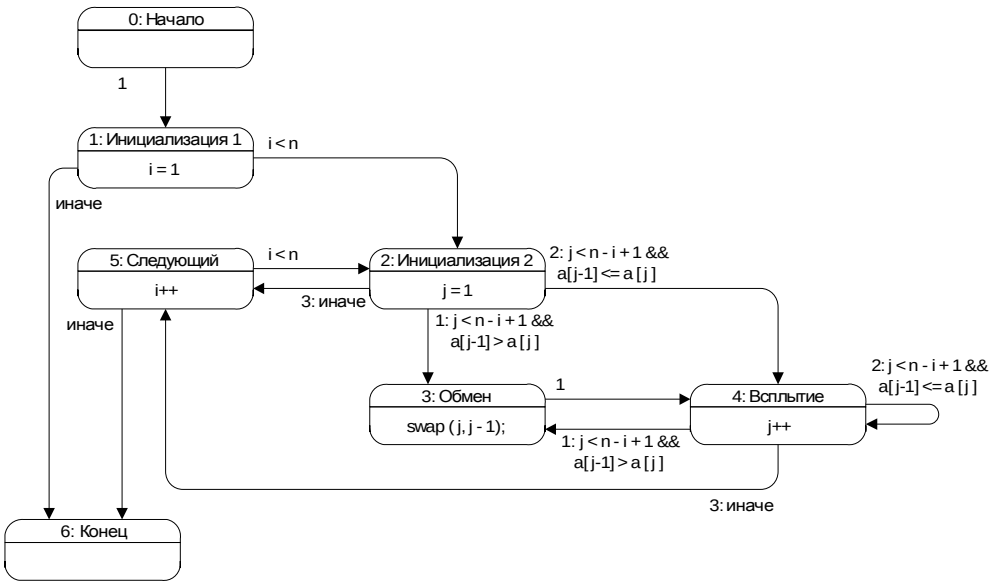

Рис. 4. Автомат Мура, построенный по схеме алгоритма «пузырьковая сортировка»

Программа, изоморфная этому графу переходов, приведена в листинге 3.

Листинг 3. Программа, изоморфная графу переходов, построенному формально по схеме алгоритма

```
public class BubbleAFormal {

    public static final int STATE_START  = 0;
    public static final int STATE_INIT1  = 1;
    public static final int STATE_INIT2  = 2;
    public static final int STATE_SWAP   = 3;
    public static final int STATE_EMERGE = 4;
    public static final int STATE_NEXT   = 5;

    private static int state = STATE_START;
    private static int a[] = {4, 3, 7, 1, 9, -2};
    private static int n = a.length;
    private static int i = 0;
    private static int j = 0;

    public static void main(String[] args) {
        do {
            nextStep();
        } while (state != STATE_START);
        out();
    }

    private static void nextStep() {
        switch(state) {
            case STATE_START:
                state = STATE_INIT1;
                break;

            case STATE_INIT1:
                i = 1;
                if (i < n) {
                    state = STATE_INIT2;
                } else {
                    state = STATE_START;
                }
                break;
```

```
            case STATE_INIT2:
                j = 1;
                if (j < n - i + 1 && a[j-1] > a [j]) {
                    state = STATE_SWAP;
                } else if (j < n - i + 1 && a[j-1] <= a [j]) {
                    state = STATE_EMERGE;
                } else {
                    state = STATE_NEXT;
                }
                break;

            case STATE_SWAP:
                out();                          // Пошаговый вывод
                swap(j,j-1);
                state = STATE_EMERGE;
                break;

            case STATE_EMERGE:
                j++;

                if (j < n - i + 1 && a[j-1] > a [j]) {
                    state = STATE_SWAP;
                } else if (j < n - i + 1 && a[j-1] <= a [j]) {
                    state = STATE_EMERGE;
                } else {
                    state = STATE_NEXT;
                }
                break;

            case STATE_NEXT:
                i++;

                if (i < n) {
                    state = STATE_INIT2;
                } else {
                    state = STATE_START;
                }
                break;

        }
    }

    private static void swap (int i, int j) {
        int t = a[i];
        a[i] = a[j];
        a[j] = t;
    }

    private static void out () {
        for (int i = 0; i < n; i++) {
            System.out.print("" + a[i] +" ");
        }
        System.out.println();
    }
}
```

Заключение

На основе выполненной работы можно сделать следующие выводы:

- несмотря на большую распространенность «эвристический подход к построению визуализаторов алгоритмов» методом не является;
- для классических вычислительных алгоритмов явное выделение состояний может быть выполнено сравнительно редко. Однако когда это удается (как в рассмотренном примере) применение автоматного метода для построения визуализаторов весьма эффективно, так как состояния выделяются естественным образом, что приводит к сравнительно небольшой сложности графа переходов;

- метод построения визуализаторов на основе формального построения графа переходов по схеме алгоритма совмещает достоинства предыдущих двух методов: по алгоритму эвристически строится программа, что характерно для традиционного программирования, а граф переходов получается по этой программе формально. Универсальность формального метода построения логики визуализаторов приводит к усложнению графа переходов и программы, построенной на его основе.

Дальнейшее совершенствование методов построения визуализаторов связано, как с автоматизацией третьего метода, так и с его развитием в части построения автоматов для визуализации алгоритма не только при его прохождении в прямом направлении, но и в обратном [17].

Аплет и исходный код программы визуализатора приведен на сайте http://is.ifmo.ru в разделе «Визуализаторы».

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)
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. Кормен Т., Лайзерсон Ч., Ривест Р. Алгоритмы. Построение и анализ. М.: МЦМНО, 2000. 960 с.
6. Кнут Д. Искусство программирования. Т. 1: Основные алгоритмы. М.: Вильямс, 2001. 712 с.
7. Gitta D. Tutorial on Visualization. http://www.siggraph.org/education/materials/ HyperVis/domik/folien.html
8. Owen G. S. Visualization Concepts: Overview. http://www.siggraph.org/education/ materials/HyperVis/concepts/overview.htm
9. Complete Collection of Algorithm Animations. http://www.cs.hope.edu/~alganim/ccaa/
10. Interactive Data Structure Visualizations. http://www.student.seas.gwu.edu/~idsv/idsv.html
11. Jawaa examples. http://www.cs.duke.edu/csed/jawaa2/examples.html
12. Sorting Algorithms. http://www.cs.rit.edu/~atk/Java/Sorting/sorting.html
13. Rling G., Naps T. L. A Testbed for Pedagogical Requirements in Algorithm Visualizations. // 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education (ITiCSE'02). http://nibbler.tk.informatik.tu-darmstadt.de/publications/2002/TestbedPedReqinAV.pdf
14. Шалыто А.А. Технология автоматного программирования // Мир ПК. 2003. № 10, с.74-78. http://is.ifmo.ru, раздел «Статьи».
15. 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, раздел «Статьи».
16. Шалыто А.А., Туккель Н.И. Преобразование итеративных алгоритмов в автоматные // Программирование. 2002. № 5, с.12-26. http://is.ifmo.ru, раздел «Статьи».
17. Казаков М.А., Корнеев Г.А., Шалыто А.А. Разработка логики визуализаторов алгоритмов на основе конечных автоматов // Телекоммуникации и информатизация образования. 2003. № 6, с. 27-58. http://is.ifmo.ru, раздел «Статьи».