Д.С. Гордеев

ВИЗУАЛИЗАЦИЯ АЛГОРИТМОВ НА ГРАФАХ: ИНТЕРПРЕТАЦИЯ АЛГОРИТМА В КАЧЕСТВЕ ПРОГРАММЫ[1]

ВВЕДЕНИЕ

Аппарат теории графов с успехом используется для моделирования различных задач, возникающих в компьютерных науках и в практических приложениях. Это, например, синтаксические деревья в программировании, раскраска графов в задачах конструирования электрических цепей, поиск кратчайшего пути в задачах создания компьютерных игр и т.д. [1]. Рисование графов является полезным способом изображения этих моделей. Визуализация графов также используется во многих приложениях для проектирования и анализа коммуникационных сетей, связанных документов, а также статических и динамических структур программ. Однако такие системы связанных объектов редко бывают статичными. Другими словами, в таких системах протекают некоторые процессы, которые могут изменять связи или иным способом перестраивать структуру модели. В случае если такие процессы можно формализовать и представить в виде некоторого алгоритма, возникает необходимость составить графическое представление процесса. Существуют методы, позволяющие представлять такие процессы как в динамической, так и в статической форме. К статическим формам можно отнести различные диаграммы и блок схемы. Например, на рис. 1 приведён пример визуализации алгоритма сортировки с помощью статичного изображения [2].

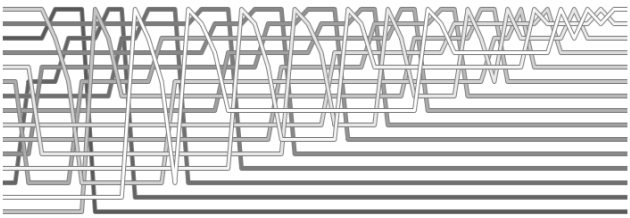

Рис. 1. Визуализация пирамидальной сортировки массива с помощью статичного изображения

[1] Работа выполнена при частичной финансовой поддержке Российского фонда фундаментальных исследований (грант РФФИ № 12-07-00091).

К динамическим формам можно отнести различные визуализации алгоритмов. На рис. 2 представлен пример динамической системы визуализации алгоритмов [3, 4].

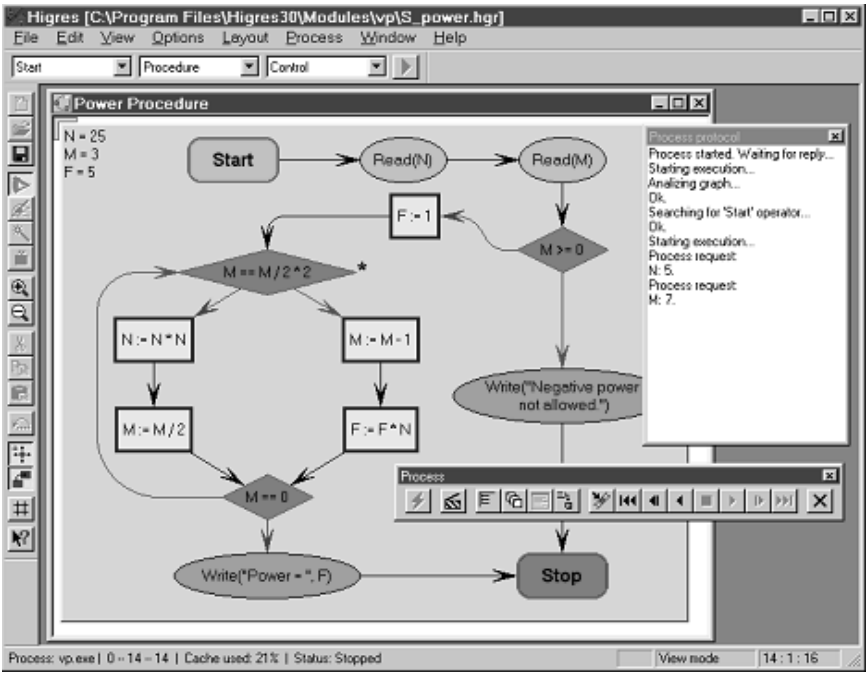

Рис. 2. Выполнение алгоритма, реализованного в виде внешнего модуля к системе Higres

Существование подобных методов позволяет изучать алгоритмы на графах и, в частности, процессы в связанных системах. Большинство работ по визуализации алгоритмов сконцентрировано на построении примеров визуализаций. Основной отличительной особенностью таких работ можно назвать узкую специализацию, в том смысле, что для каждого нового алгоритма требуется создавать новых визуализатор.

МЕТОДЫ ВИЗУАЛИЗАЦИИ АЛГОРИТМОВ

В данной работе описывается метод построения визуализаций, основанный на визуальных эффектах, генерируемых по входному алгоритму. Визуализация графа – это представление графа в графическом виде. Обычно элементам графа сопоставляются некоторые графические примитивы, что даёт возможность построить его изображение. Например, вершины отображаются окружностями, а дуги прямыми, ломаными или гладкими кривыми. Среди применяющихся методов визуализации алгоритмов можно выделить два метода: метод интересующих событий и метод, основанный на данных [5]. Первый метод основан на выделении событий, которые происходят во время исполнения алгоритма. Это, например, сравнение величин атрибутов некоторых элементов графаили удаление дуги. Суть этого метода в том, чтобы для каждого такого события реализовать визуальный эффект. Второй метод основан на изменении данных. Во время работы изменяется состояние памяти, например, значения переменных. Далее эти изменения некоторым образом визуализируются. В простейшем случае для этого используется отображение значений переменных в таблице. Такой метод применяется в отладчиках интегрированных систем разработки программного обеспечения.

У существующих визуализаторов алгоритмов есть ряд недостатков. Один из главных недостатков заключается в том, что для визуализации нового алгоритма строится новый визуализатор. Это означает, что если есть необходимость построить визуализацию алгоритма, сколь угодно мало отличающегося от исходного алгоритма, потребуется заново изготовить визуализатор. Также визуализаторы часто не отображают соответствие между инструкциями алгоритма и генерируемыми визуальными эффектами. Часто визуализаторы не позволяют перенастроить визуальные эффекты, соответствующие событиям. К недостаткам также можно отнести излишнюю перегруженность исходного текста алгоритма декларативными инструкциями. На рис. 3 представлен кадр, полученный при визуализации алгоритма в системе Leonardo [6, 7]. Как видно, в системе Leonardo используются директивы в специальном формате, /** Not VisualUpdate **/. Такие декларативные конструкции используются для группировки визуальных событий или же для непосредственного запуска визуальных эффектов.

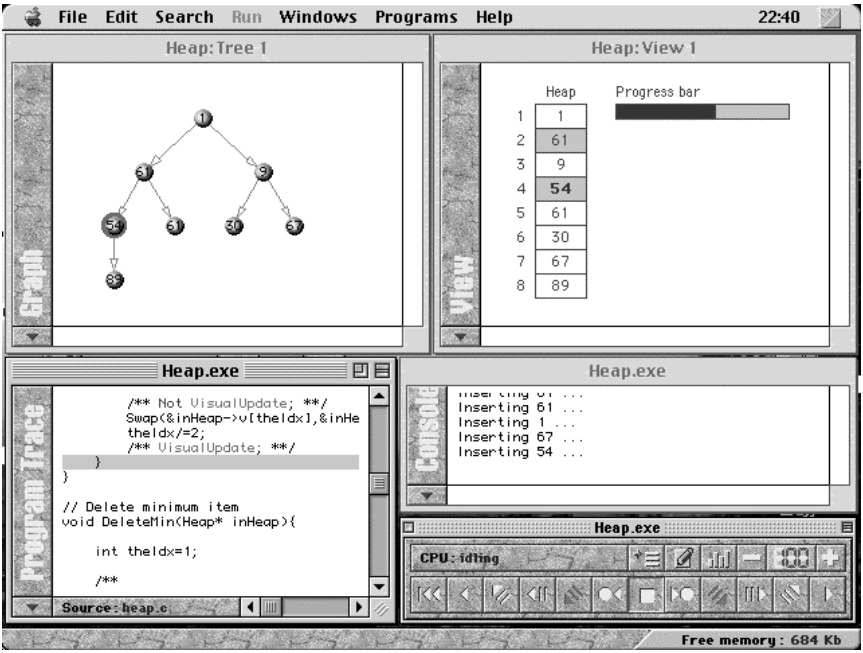

Рис. 3. Пример визуализации операций над binary heaps в системе Leonardo, A C Programming Environment for Reversible Execution and Software Visualization

МОДЕЛЬ ВИЗУАЛИЗАЦИИ

Для решения этих задач предложена модель визуализации графовых алгоритмов, основанная на динамическом подходе. Суть данного подхода заключается в том, что заданный алгоритм формулируется на некотором языке программирования, допускающем использование графовых конструкций, а также с возможностью исполнить программу, полученную из текста алгоритма после некоторого набора преобразований [8]. Во время исполнения полученной программы генерируется информация, которую можно использовать для визуализации оригинального алгоритма. Примером графовой конструкции может быть операция добавления дуги или изменения атрибута вершины графа. В приведённом ниже примере представ-

лен алгоритм обхода в ширину для произвольного графа. В данном случае используются конструкции Get(vertexid, attributename) и Set(vertexid, attributename, attributevalue) для чтения и изменения значения атрибута вершины, соответственно. Для построения визуализации алгоритма обхода в ширину вершинам входного графа добавляется атрибут *state*, по значению которого можно определить, была ли посещена конкретная вершина в процессе обхода графа или нет.

```
VertexQueue.Enqueue(Graph.Vertices[0]);
while (VertexQueue.Count > 0)
{
    Vertex v = VertexQueue.Dequeue();
    Set(v.ID, "state", "visited");
    foreach(Edge e in v.InEdges)
    {
        Vertex t = e.PortFrom.Owner;
        string c = Get(t, "state");
        if(c != "visited")
       {
            Set(t, "state", "visited");
            VertexQueue.Enqueue(t);
        }
    }
    foreach(Edge e in v.OutEdges)
    {
        Vertex t = e.PortTo.Owner;
        string c = Get(t, "state");
        if(c != "visited")
        {
            Set(t, "state", "visited");
            VertexQueue.Enqueue(t);
        }
    }
}
VertexQueue.Clear();
```

Каждая инструкция алгоритма генерирует один или несколько образов текущего состояния графовой модели. Графическая модель является иерархическим графом с пометками. Иерархический граф H – это пара элементов, первый из которых граф G, а второй T – дерево фрагментов. Каждый фрагмент является подграфом графа G. Для каждых двух фрагментов U и

V выполняется только одно из следующих утверждений: U подграф V, V подграф U, или U совпадает с V. Подробнее иерархические графы описаны в [1].

Для решения задачи подсветки текущей исполняемой инструкции в кадре, сгенерированном во время исполнения данной инструкции, используется следующий подход. Текст алгоритма преобразуется так, чтобы добавить в каждую значимую строку текста номер этой строки. После такого преобразования текста алгоритма обхода в ширину из примера, приведённого выше, будет выглядеть так:

```
VertexQueue.Enqueue(Graph.Vertices[0]);
while (WhileCondition(1, VertexQueue.Count > 0))
{
    Vertex v = VertexQueue.Dequeue();
    Set(4, v.ID, "state", "visited");
    foreach(Edge e in ForeachCollection(5, v.InEdges))
    {
        Vertex t = e.PortFrom.Owner;
        string c = Get(7, t, "state");
        if(IfCondition(8, c != "visited"))
       {
            Set(10, t, "state", "visited");
            VertexQueue.Enqueue(t);
        }
    }
    foreach(Edge e in ForeachCollection(13, v.OutEdges))
    {
        Vertex t = e.PortTo.Owner;
        string c = Get(16, t, "state");
        if(IfCondition(17, c != "visited"))
        {
            Set(19, t, "state", "visited");
            VertexQueue.Enqueue(t);
        }
    }
}
VertexQueue.Clear();
```

Пример, приведённый выше, демонстрирует использование инструкций, изменяющих атрибуты вершин графа. Это типичная ситуация для алгоритмов, реализующих различные обходы графов. Например, алгоритм кодирования по Прюферу конструирует последовательность чисел, соответствующих вершинам заданного дерева. В процессе кодирования вершины графа одна за другой удаляются. Чтобы выполнить такое действие, нужно

использовать инструкцию RemoveVertex(...). Использование данной инструкции приводит к генерации визуального эффекта исчезновения соответствующей вершины. Это пример записи в текстовой форме алгоритма кодирования по Прюферю в качестве параметра системы визуализации:

```
int i=0;
List<Vertex> Leafs = new List<Vertex>();
int n = Graph.Vertices.Count;
while(i++ <= n-2)
{
    Leafs.Clear();
    foreach(Vertex v in Graph.Vertices)
if(v.OutEdges.Count == 0) Leafs.Add(v);
    Vertex codeItem =
Leafs[0].InEdges[0].PortFrom.Owner;
    Output.Add(codeItem);
    RemoveVertex(Leafs[0]);
}
```

Во время исполнения преобразованной программы каждая инструкция алгоритма генерирует информацию, описывающую номер исполняемой строки, идентификатор обрабатываемого элемента графа, название изменяемого атрибута элемента графа, предыдущее значение атрибута, новое значение атрибута, а также временную метку. Такая информация о каждой исполненной инструкции позволяет получить журнал исполняемых операций, содержащий подробную информацию о состоянии графовой модели во время исполнения алгоритма. Далее полученный журнал операций и граф, задаваемый в качестве параметра, можно использовать для генерации визуализации алгоритма. Каждой записи журнала соответствует некоторый графический эффект. Простейший пример такого графического эффекта – это цветовое выделение текущей исполняемой строки алгоритма.

СИСТЕМА ВИЗУАЛИЗАЦИИ

С использованием предложенной модели реализован прототип системы визуализации алгоритмов. Система визуализации графовых алгоритмов содержит несколько компонентов. Это модуль исполнения алгоритма, графовый редактор и, собственно, визуализатор графовых алгоритмов. Так как компоненты могут быть реализованы на разных платформах, не теряя общности, можно считать, что информация между компонентами передаётся в

текстовом виде. Поэтому можно выделить подсистему конвертации данных в текстовую форму и обратно. Назначение данного модуля – это преобразование структуры графа в текстовое представление и обратно. Назначением модуля исполнения является запуск программы, полученной из текста алгоритма-параметра, и порождение журнала исполненный операций. Таким образом, следует заметить, что исполнение алгоритма-параметра отделено от визуализации. Это позволяет исполнить алгоритм однажды и затем строить и уточнять визуализацию без повторных запусков. Это может быть полезно при визуализации вычислительно-ёмких алгоритмов, когда перезапуск программы алгоритма требует большого времени. Чтобы обеспечить корректное функционирование модуля исполнения, требуется выполнение нескольких условий. Во-первых, любой существующий компилятор или интерпретатор можно использовать для создания модуля исполнения. Соответственно, текст алгоритма-параметра должен быть сформулирован на входном языке выбранного компилятора или интерпретатора. Этот пункт не делает существенных ограничения на язык описания входных алгоритмов, так как многие языки программирования допускают использование графовых конструкций. Во-вторых, запрещается использование специальных инструкций более одного раза в одной строке. Нарушение этого условия приводит к генерации нескольких визуальных эффектов при том, что выделяться цветом будет только одна строка из текста алгоритма. Такой подход позволяет рассматривать текст входного алгоритм как готовую к исполнению программу. Также это позволяет передавать входной граф в построенную программу, чтобы генерировать журнал выполненных операций. Однако существуют некоторые алгоритмические ограничения. В данном подходе разумно визуализировать только эффективные алгоритмы. Журнал операций, генерируемый во время исполнения неэффективного алгоритма, может строиться долгое время. Но для таких случаев можно сделать предположение, что входные графы будут маленькими. Такое предположение позволяет строить визуализации за приемлемое время. Модуль исполнения алгоритма получает текст, сформулированный на подходящем языке программирования, и исполняет его, если не возникло никаких ошибок. Модуль исполнения строит журнал исполненных операций, построенный во время работы алгоритма над заданным графом. Журнал операций содержит информацию обо всех изменениях атрибутов графовых элементов, а также о добавленных или удалённых элементах графа. Далее эта информация используется для генерации визуализации.

Вторым основным компонентом является визуализатор. Он получает на вход текст алгоритма, журнал операций, граф и дополнительных настройки.

Каждая запись журнала операций содержит информацию об исполняемой инструкции алгоритма. Это может быть номер строки, изменившееся значение атрибута элемента графа, добавление или удаление элементов графа. Добавление той или иной информации в журнал обеспечивается исполнением специальных инструкций, которые добавляются на стадии подготовки текста алгоритма к исполнению. Для этого достаточно использовать контекстную замену. Цель таких преобразований – устранить необходимость декларативных конструкций в тексте алгоритма, которые не имеют отношения собственно к сути этого алгоритма. Таким образом, можно вводить текст алгоритма, и действия заданных инструкций будут визуализированы без вмешательства пользователя. В требованиях к системе отмечено, что текст должен содержать только алгоритм без дополнительных конструкций. Добавление в запись журнала информации о текущей исполняемой строке осуществляется простой контекстной заменой. Так как текст алгоритма есть упорядоченный пронумерованный набор строк, то в запись журнала можно добавлять индекс текущей исполняемой строки. Использование в дальнейшем, при визуализации, этого индекса позволяет визуальной выделить текущую исполняемую строку. Запись журнала также может содержать информацию об изменении значения атрибутов вершин, дуг или портов. Порты представляют собой точки входа дуг в вершины и точки выхода дуг из вершин. При визуализации бывает полезно, когда для таких точек выделены специальные сущности. Строго математически можно моделировать порты через помеченные вершины. Добавление таких объектов не ограничивает общности рассматриваемых графов. Атрибут вершины, дуги или порта может иметь любое строковое название и строковое значение. В записи журнала может храниться предыдущее значение данного атрибута. Эта информация полезна и для построения визуализации, так как позволяет направить визуальный эффект от предыдущего элемента графа к текущему элементу. Итак, данный подход к визуализации алгоритмов основан на сопоставлении информации из записей журнала операций некоторого множества визуальных эффектов. Неочевидно, как сопоставлять информацию из записи журнала с элементами множества визуальных эффектов. В данном случае потребуется вмешательство пользователя с целью задания явного соответствия между множеством атрибутов в тексте алгоритма и желаемыми визуальными эффектами. Например, если в тексте алгоритма есть операция изменения координат элемента графа с использованием атрибута “*позиция*”, то разумно сопоставить этому атрибуту визуальный эффект, который приводит к смещению элемента графа.

На рис. 4 представлен пример визуализации алгоритма поиска в глубину на графе, являющимся бинарным деревом. Рисунок представляет собой один из кадров, снятый в процесс визуализации алгоритма. В левой части рисунка отображён текст алгоритма, сформулированный в терминах графа. Наличие атрибута цвета у вершины графа означает, что вершина уже была посещена в процессе обхода. В правой части представлен иерархический помеченный граф с атрибутами. Наличие или отсутствие атрибута приводит к реализации соответствующих визуальных эффектов. В данном примере посещённые вершины получают установленный атрибут цвета. Такому значению атрибута соответствует увеличение толщины линии, применяемой для рисования окружности. Вершины с тонким контуром ещё не были посещены.

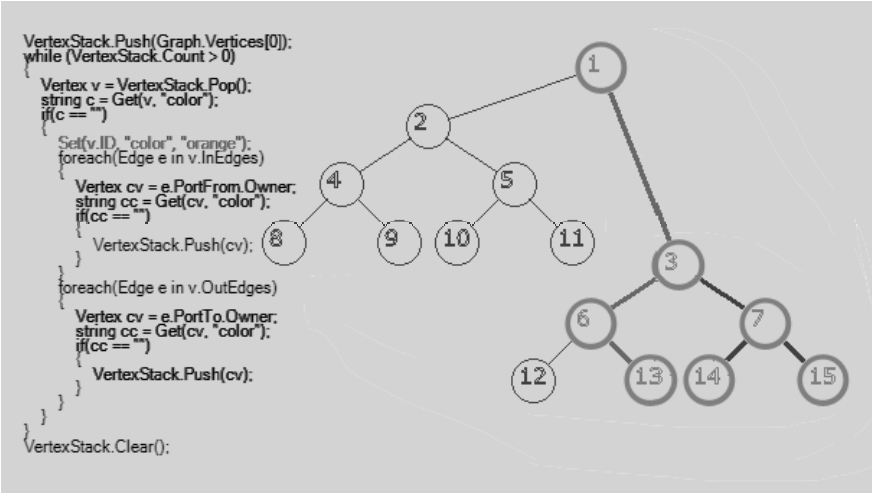

Рис. 4. Пример визуализации алгоритма поиска в глубину, один из промежуточных кадров

При визуализации алгоритмов на графах есть возможность применять методы, повышающие наглядность. Если для некоторой записи журнала определён контекст визуализации эффекта, то можно использовать этот контекст для повышения наглядности работы алгоритма. Например, для визуализации алгоритма обхода в глубину можно применять плавную визуализацию вдоль дуги, соединяющей предыдущую и текущую вершины. В

таком случае контекстом визуализации для текущей вершины будет предыдущая вершина. В случае если предыдущая вершина не является смежной с текущей вершиной, для повышения наглядности визуализации можно применять плавную визуализацию вдоль дуг, принадлежащих кратчайшему пути, соединяющему текущую и предыдущую вершины. На рис. 4 представлен также пример использования визуализации с использованием контекста. В данном примере в процессе обхода при продвижении от предыдущей вершины к текущей вершине изменяется толщина линий, применяемых для визуализации инцидентных дуг. При этом утолщённая светлая линия применяется для рисования дуг, принадлежащих пути от текущей вершины до корня дерева. Утолщённая тёмная линия применяется для рисования дуг, инцидентных уже посещённым вершинам.

Также для повышения наглядности визуализации можно применять отображение дополнительных структур данных. Например, для отображения содержимого стека для алгоритма обхода в глубину можно использовать визуализацию связного графа, у каждой вершины которого степень равна двум или единице. Во время работы алгоритма количество элементов в стеке изменяется, и соответствующие вершины добавляются или удаляются из графа визуализации стека.

ЗАКЛЮЧЕНИЕ

В данной статье описана модель визуализации алгоритмов на графах, обеспечивающая построение визуализаций алгоритмов за счёт использования алгоритма в качестве параметра, а также за счёт гибкой системы визуальных эффектов. Также описывается метод для повышения наглядности визуализации алгоритмов на графах, позволяющий использовать дополнительную информацию, генерируемую во время исполнения алгоритма-параметра. Разработана система визуализации алгоритмов, обеспечивающая апробацию на практике предлагаемого метода визуализации алгоритмов на графах. Система визуализации позволяет задать в качестве параметров граф, алгоритм и настройки визуализации. Результатом работы системы является последовательность изображений, соответствующих промежуточным состояниям графовой модели во время работы алгоритма.

СПИСОК ЛИТЕРАТУРЫ

1. Касьянов В. Н., Евстигнеев В. А. Графы в программировании: обработка, визуализация и применение. – СПб.. БХВ-Петербург, 2003. – 1104 с
2. http://corte.si/posts/code/visualisingsorting/
3. Lisitsyn I.A., Kasyanov V.N. Higres — visualization system for clustered graphs and graph algorithms // Proc. of Graph Drawing 99. — Berlin a.o.: Springer Verlag, 1999. – P. 82–89. — (Lect. Notes in Comput. Sci.; Vol. 1731).
4. http://pcosrv.iis.nsk.su/higres/
5. Demetrescu C., Finocchi I., Stasko J. T., Specifying Algorithm Visualizations: Interesting Events or State Mapping? // In Proc. of Dagstuhl Seminar on Software Visualization – Lect. Notes in Comput. Sci. – 2001. – P. 16–30.
6. C. Demetrescu C.Finocchi I. A general-purpose logic-based visualization framework, Proceedings of the 7th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media (WSCG'99). Plzen, Czech Republic, February 1999. P. 55–62,
7. http://www.dis.uniroma1.it/~demetres/Leonardo/
8. Гордеев Д.С. Модель интерактивной визуализации графовых алгоритмов. // Труды НПО 2011 / Рабочий семинар «Наукоёмкое программное обеспечение». – Новосибирск: Ин-т систем информатики имени А. П. Ершова СО РАН, 2011. – С. 58-62.