

Обзор техник визуализации алгоритмов на графах

Д. С. Гордеев

Институт систем информатики им. А. П. Ершова СО РАН, Новосибирск

EMAIL: gds@iis.nsk.su

ORCID: 0000-0003-1623-9553

Аннотация

Настоящий обзор посвящен описанию существующих визуальных техник описания поведения алгоритмов на графах. Приведённые примеры рассматривались с позиции наличия возможности задания графов-параметров пользователем, возможности задания алгоритмов-параметров, а также возможности настройки визуальной части изображения.

В существующих системах часто используются автоматически сгенерированные графы в качестве параметров алгоритмов, а сами алгоритмы фиксированы. Большинство систем визуализации представляют собой каталоги, где для каждого фиксированного алгоритма визуализации строятся для различных графов с помощью некоторых библиотек визуализации. При этом, если потребуется ввести новый алгоритм в такую систему, то будет необходимо разработать всю визуализацию заново. Также интересной представляется возможность гладкой анимации, являющейся весьма удобным дополнением к алгоритму визуализации для создания непрерывного изображения на дисплее и для захвата внимания пользователя.

Также рассматривается вопрос используемой техники визуализации алгоритмов. Какие преимущества даёт событийно-ориентированный подход, а какие преимущества даёт подход ориентированный на данные и их изменение. Также интересной характеристикой представляется возможность настраивать параметры рисования графических примитивов для процесса визуализации.

Ключевые слова: графовые алгоритмы, визуализация алгоритмов, анимация алгоритмов.

Работа выполнена при частичной финансовой поддержке Российского фонда фундаментальных исследований (грант РФФИ № 18-07-00024).

1. Введение

Графом называется пара множеств, первое из которых есть непустое конечное множество элементов произвольной природы или множество вершин, а второе является подмножеством декартового произведения множества вершин и называется множеством дуг. Дуги могут быть ориентированными и неориентированными. Графы и графовые модели имеют широкое применение, так как могут быть использованы для наглядного представления сложной информации. Графовыми моделями являются расширения графов, например, графы, снабжённые атрибутами для элементов [1, 2].

Построение визуальных образов применяется для наглядного представления графов и графовых моделей. Визуализация графов сопряжена с графическим изображением элементов графа, отражающим связи между вершинами и удовлетворяющим эстетическим критериям. Примером эстетических критериев могут служить минимальность числа пересечения дуг или отсутствие пересечений дуг и вершин. Существует множество методов визуализации графов, на основе которых построены системы визуализации графовых моделей. Среди последних встречаются как узкоспециализированные [3], так и системы общего назначения [4, 5, 6, 7, 8, 9]. Однако одна лишь

визуализация графовых моделей даёт представление только структуре отображаемой информации. Знание того, как можно оперировать информацией, представляемой графами, повышает понимание самой информации. Алгоритмы на графах отражают операции над информацией, которая может быть представлена с помощью графов. Визуализация графовых алгоритмов даёт представление о методах работы с графовой информацией. Другими словами, визуализация преобразования графовой модели с помощью действия некоторого алгоритма даёт более полное представление об изображаемой информации и, кроме того, описывает алгоритм преобразования графа. Например, если графовая модель представлена с помощью сети Петри в начальном состоянии, то примером визуализации алгоритма может служить демонстрация работы сети. В настоящее время развивается направление визуализации алгоритмов, и в частности, алгоритмов на графах.

Целью данного обзора является исследование существующих методов визуализации алгоритмов, и в частности, алгоритмов на графах. По результатам обзора можно составить представление о том, насколько представляется возможным сейчас создание интерактивной энциклопедии алгоритмов на графах, с помощью которой пользователи смогли бы изучать алгоритмы обработки графов. Такая энциклопедия должна содержать богатый набор известных алгоритмов, с помощью которых можно обрабатывать графы различных классов. Для наиболее эффективного усвоения материала пользователь должен активно участвовать в процессе обучения. Недостаточно только прочитать текстовое представление алгоритма, нужно также иметь возможность проследить процесс обработки графа рассматриваемым алгоритмом, возможность просматривать процесс обработки на разных входных данных, в том числе и заданных самим пользователем, а также возможность добавлять в процессе обучения модифицированные версии алгоритма.

Таким образом, исследование проводится с точки зрения, что система визуализации должна предоставлять возможность визуализации алгоритмов и возможность манипуляций со входными параметрами, то есть с графами. Также должна быть возможность добавлять новые алгоритмы обработки графов, в том числе и модификации существующих алгоритмов. Представляется интересной возможность синхронизации визуального представления алгоритма с текстовальным представлением и демонстрация работы алгоритма в обоих временных направлениях с ускорением или замедлением по времени.

2. Два основных подхода в визуализации алгоритмов

Прежде чем говорить о визуализации алгоритмов на графах, следует уточнить, как следует понимать визуализацию графов. Визуализация графа означает графическое отражение элементов графа и связей между ними с помощью набора графических примитивов, удовлетворяющее некоторому набору эстетических критериев. Например, треугольники для вершин графа, и ломаные линии для дуг. Существует множество методов визуализации графов. Эти методы условно можно разделить на классы, соответствующие классам графов. Например, существует всего одно семейство алгоритмов отображения ориентированных ациклических графов. Это так называемый поровневый метод. Однако для отображения неориентированных графов существует гораздо больше семейств алгоритмов. Это, изображение, когда все вершины располагаются на концентрических окружностях или силовой метод, когда дуги представляются пружинами и для укладки используется физическая модель шаров соединённых пружинами.

Визуализация позволяет облегчить понимание отображаемой информации и, соответственно, упростить её анализ и дальнейшую обработку. Тема визуализации графов изучается давно, и уже

накоплено большое количество результатов в этой области. Однако визуализация алгоритмов на графах является более сложной задачей. Так как алгоритм отражает некоторые преобразования над графом или графовой моделью, то его визуализация представима как последовательная смена состояний графовой модели при применении шагов или инструкций алгоритма. Существует исследование, описывающее основные методы визуализации алгоритмов [10].

Первый естественный подход к визуализации алгоритмов заключается в том, чтобы выделить некоторые события в реализации алгоритма, отвечающие некоторым действиям в заданном алгоритме, и в том, чтобы сопоставить им некоторые графические события с использованием визуализации некоторых графических примитивов. Второй естественный подход заключается в том, чтобы графически интерпретировать состояние вычислений программы, или, другими словами, совокупность всех переменных программы, и дать возможность графическим объектам в визуализации отразить зависимость изображения от переменных программы.

Задача спецификации графических абстракций, иллюстрирующих вычисления при визуализации алгоритмов, является сложной задачей. При попытке её решить возникает два вопроса. Как моделировать графические сцены и анимацию переходов? Как задавать атрибуты, отражающие поведение графических объектов, описывающих заданный алгоритм? Эффективность спецификации в общем случае основывается на гибкости, общности и способности к настраиваемым визуализациям.

Важным свойством визуализации, определяющим связь визуализируемого алгоритма с его реализацией, является то, как события визуализации запускаются вычислениями, которым они соответствуют. Событийно-ориентированный подход означает выделение событий в коде реализации алгоритма, соответствующих действиям алгоритма, имеющим графическую интерпретацию, и в превращении их в графические события

с помощью вставки вызовов соответствующих графических примитивов. Подход, ориентированный на данные, заключается в определении отображения состояния вычислений в графические образы, обычно описывающие атрибуты графических объектов, зависящие от переменных заданной реализации. В этом случае события визуализации запускаются модификациями переменных.

2.1. Событийно-ориентированный подход

Естественный подход к визуализации алгоритмов состоит в аннотации кода вызовами примитивов визуализации. Первый шаг состоит в том, чтобы выделить некоторые действия, исполняемые алгоритмом, который требуется визуализировать. Такие действия можно назвать выделенными событиями. Например, если рассмотреть алгоритм сортировки массива, то можно выделить события обмена местами двух элементов. Массив, вообще говоря, графом не является, но массив можно смоделировать с помощью дерева, в котором у каждой вершины-предка, может быть только одна вершина-потомок, далее унарного дерева. Таким образом, алгоритм сортировки можно рассматривать как алгоритм, обрабатывающий некоторую графовую модель.

Вторым шагом является сопоставление каждому выделенному событию определённой сцены визуализации. Если, например, в алгоритме сортировки изображать значения, которые должны быть отсортированы как последовательность столбцов разной длины, то визуализация перестановки может быть реализована путем перестановки двух столбцов соответствующих числам, которые надо поменять местами. Запуск операций над графическими примитивами визуализации, можно получать за счет аннотирования исходного кода реализации алгоритма в точках, где находятся интересующие события.

Событийно-ориентированный подход является интуитивным, и с его помощью можно получить достаточно богатый

класс визуализаций. К недостаткам этого подхода отнести увеличение размера кода и необходимость детального понимания кода алгоритма, чтобы выделить интересующие события.

Например, для создания визуализации алгоритмов с помощью системы Polka [11], необходимо добавить в исходный код программы вызовы специальных функций, запускающих события визуализации. Также требуется создать сцены, реализующие визуализацию событий алгоритма, и построить соответствие между запускающими функциями и сценами.

2.2. Подход, опирающийся на отображение данных

Системы визуализации алгоритмов, использующие управление данными, опираются на предположение, что, наблюдая изменение переменных программы можно судить о сути исполняемого алгоритма. Такие системы могут быть использованы для графического отображения состояния вычислений. Примером является отладчик интегрированных сред разработки, который показывает, как изменяются переменные в процесс работы алгоритма. Визуализация алгоритма в таких системах состоит в обеспечении графической интерпретации нужных структур данных. Это даёт возможность отображать состояние вычисления в любой момент исполнения. В случае традиционных отладчиков отображение является простым и не может быть изменено пользователем: как правило, обеспечивается непосредственное представление о содержании переменных. Отладчик обновляет отображение данных после каждого изменения, иногда подсвечивая последние измененные переменные, помогая пользователю следить за изменениями.

В общем случае матрица смежности, использованная в коде, может быть представлена визуально как граф с вершинами и дугами, массив чисел может быть визуально представлен как унарное

дерево, данные кучи – как сбалансированное бинарное дерево. Поскольку акцент делается только на структурах данных, то те же графические интерпретации и тот же самый код визуализации могут быть использованы для визуализации алгоритмов, использующих такие же структуры данных. Например, алгоритм, который управляет преобразованием данного массива чисел, может быть визуально представлен так же, как сортируемый массив чисел, в виде последовательности столбцов.

Основные преимущества данного подхода в том, что визуализация является простой, и её понимание не требует специальных навыков: в большинстве случаев, чтобы подготовить базовые визуализации, требуется лишь интерпретация некоторого набора переменных. С другой стороны, при использовании лишь модификации данных может получиться достаточно громоздкая визуализация, что ограничит возможность восприятия и понимания сложного алгоритма. Далее в работе рассматриваются системы, реализующие методы визуализации графовых алгоритмов.

3. Leonardo

Система Leonardo представляет собой интегрированную среду для разработки, выполнения и визуализации C-программ, реализующую подход управления данными [12]. Система Leonardo поддерживает механизм расчета графической визуализации, как реализуется путем добавления в код специальных директив.

Далее будет приведено описание простого примера визуализации алгоритма с использованием выделенных событий и отображения состояния. В качестве примера будет использоваться алгоритм сортировки, где элементы массива визуализируются как столбцы различной высоты. Пример основан на приведённой ниже имплементации алгоритма сортировка массива с помощью языка C.

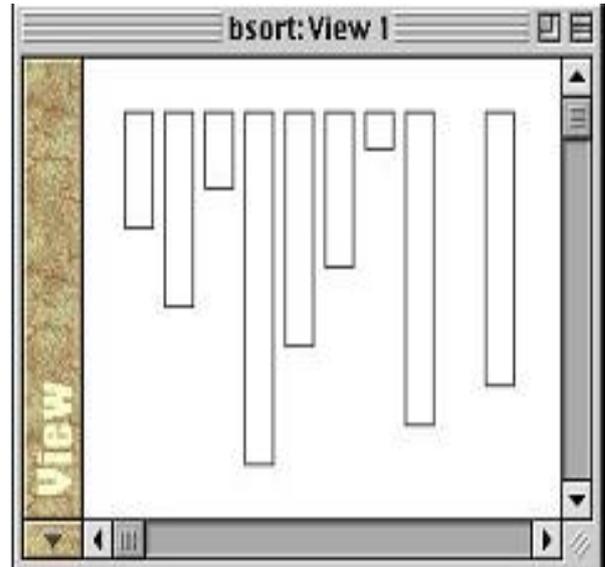
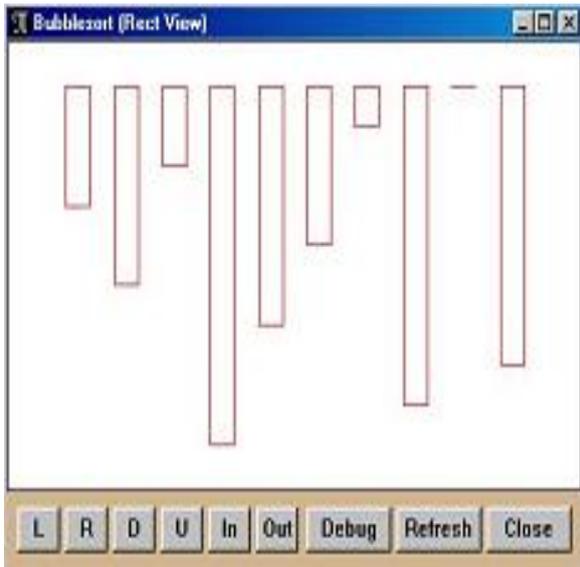


Рисунок 1 – Примеры визуализации алгоритма сортировки системами Polka и Leonardo, высота прямоугольников пропорциональна их значениям. На рисунке показана визуализация не отсортированного массива.

```
void main(int[] v) {
for (j=v.Length; j>0; j--)
for (i=1; i<j; i++)
if (v[i-1]>v[i]) {
int temp=v[i]; v[i]=v[i-1]; v[i-1]=temp;
}
}.
```

Для каждого события в терминах алгоритма следует выбрать событие в терминах событий визуализации. В процессе работы алгоритма элементы массива переставляются местами, и в данном контексте следует выделить события перестановки столбцов, если числа связать с прямоугольниками разной высоты. В системе Polka визуализация задаётся путём добавления в исходный код программы вызовов некоторых функций, запускающих визуализацию выделенных событий. Ниже представлен код алгоритма сортировки, с добавлением вызовов методов, запускающих визуализацию выбранных событий.

```
void main(int[] v) {
bsort.SendAlgoEvt("Input",v.Length,v);
for (j=v.Length; j>0; j--)
for (i=1; i<j; i++)
if (v[i] > v[i+1]) {
int temp = v[i]; v[i] = v[i-1]; v[i-1] =
temp;
```

```
bsort.SendAlgoEvt("Exchange",i,i-1);
}
}.
```

Используя соотношение состояний, можно каждую перестановку элементов массива ассоциировать с некоторыми графическими событиями. В данном примере эти события являются перестановками столбцов. Событие "Input" означает, что алгоритм начал работу, требуется визуализировать начальную конфигурацию массива. В данном случае требуется отобразить набор столбцов с высотами, соответствующими значениям элементов массива. На Рис. 1 представлен пример такой визуализации. Второе событие "Exchange" означает перестановку двух элементов массива. Визуально это означает, что столбцы, соответствующие переставляемым числам, также должны быть переставлены.

С помощью Leonardo и Polka можно создать визуализации более понятные для восприятия. Например, в алгоритме сортировки методом пузырька, можно перемещать прямоугольники одновременно. В системе Polka можно записать инструкции так, чтобы движение графических примитивов начиналось одновременно. В системе Leonardo механизм та-

ких действий выглядит проще за счёт использования директивы `ScreenUpdateOn`. Эта директива позволяет запускать все изменения одновременно. Если `ScreenUpdateOn` отключена, то система визуализации бездействует, и события визуализации не возникают. Чтобы визуализировать каждую перестановку в примере как атомарное действие, можно временно приостановить обновление экрана перед запуском перестановки и включить его снова после.

```
/** Not ScreenUpdateOn; */
int temp=v[i]; v[i]=v[i-1]; v[i-1]=temp;
/** ScreenUpdateOn; */
```

Иногда может потребоваться визуализации действий алгоритма, которые соответствуют отсутствию изменений переменных: к примеру, события сравнения двух элементов в алгоритме сортировки. Такие действия можно отражать с помощью интересующих событий, каждое из которых может быть связано с соответствующим событием алгоритма. При использовании метода отображения состояния такая задача может вызвать трудности, так как каждое событие должно ассоциироваться с некоторой переменной.

Например, чтобы визуализировать событие сравнения двух элементов в системе Polka, достаточно добавить один вызов функции, запускающей визуализацию. При этом событие визуализации "Compare" происходит незадолго до фактического сравнения значений.

```
void main(int[] v) {
bsort.SendAlgoEvt("Input",n,v);
for (j=v.Length; j>0; j--)
for (i=1; i<j; i++) {
bsort.SendAlgoEvt("Compare",i,i-1);
if (v[i-1] > v[i]) {
int temp = v[i]; v[i] = v[i-1]; v[i-1] = temp;
bsort.SendAlgoEvt("Exchange",i,i-1);
}
}
}
```

В системе Leonardo, чтобы визуализировать события сравнения, приходится симулировать метод выделенных событий. А именно, нужно вызывать перед сравнением дополнительную функцию, которая визуально выделяет сравниваемые элементы.

Полезной возможностью при визуализации алгоритма является изображение инвариантных свойств алгоритма. Визуализация инвариантных свойств может помочь обнаружить допущенные ошибки и повысить уровень понимания комбинаторных, алгебраических и численных аспектов решаемой задачи. Примером инвариантного свойства в алгоритме сортировки пузырьком может быть то, что элементы массива с номерами больше J всегда находятся в их конечном положении. Соответственно, столбцы, которые находятся в конечных позициях, можно визуально выделять определённым образом.

```
void main(int[] v) {
bsort.SendAlgoEvt("Input",n,v);
for (j=v.Length; j>0; j--) {
for (i=1; i<j; i++) {
bsort.SendAlgoEvt("Compare",i,i-1);
if (v[i-1] > v[i]) {
int temp = v[i]; v[i] = v[i-1]; v[i-1] = temp;
bsort.SendAlgoEvt("Exchange",i,i-1);
}
}
bsort.SendAlgoEvt("InPlace",j-1);
bsort.SendAlgoEvt("InPlace",0);
}
```

Запуск события "InPlace" помечает графические объекты, соответствующие элементам массива, которые находятся на своей конечной позиции. Для системы Leonardo ситуация аналогична предыдущей. События "In Place" нужно

симулировать с помощью вызова дополнительных функций.

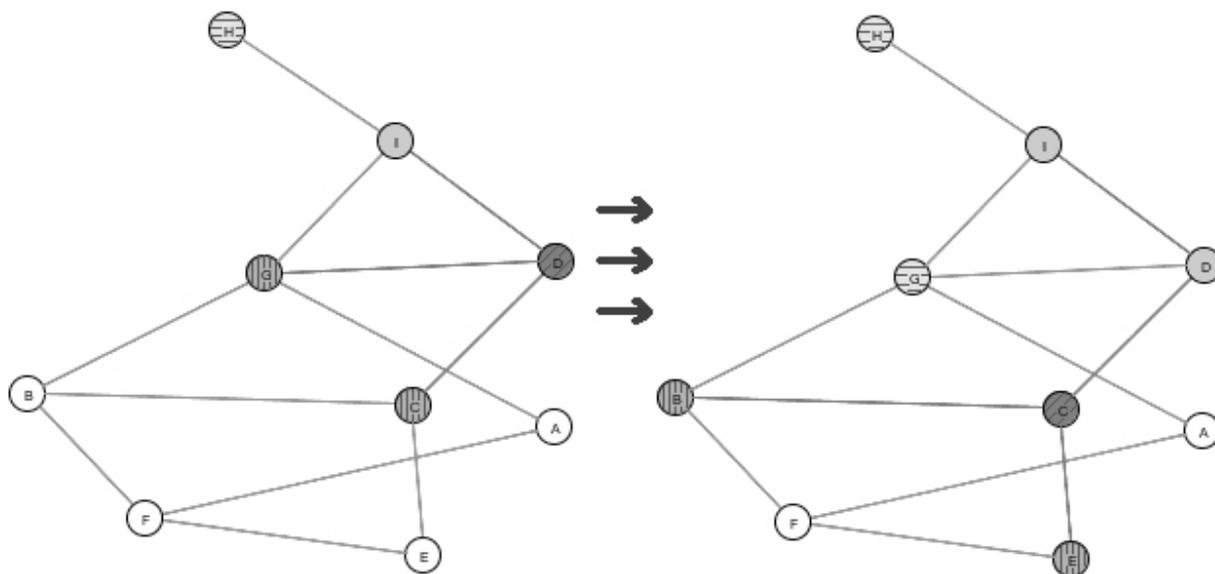


Рисунок 2 – Алгоритм поиска в глубину.

Итак, приведенный пример показывает, каким образом можно описывать визуализацию алгоритма с помощью двух широко применяемых методов визуализации: метода выделенных событий и метода отображения состояний переменных. Метод выделенных событий интуитивен и прост в визуализации. Разрастание кода имплементации алгоритма компенсируется относительной простотой реализации, так как требуется только реализовать визуализацию для каждого отдельно взятого типа событий. При этом реализация возможна в терминах предметной области, то есть передавать не только значения переменных, а более сложные структуры данных, допускающих сложные операции визуализации с ними. В случае же использования метода отображения состояния переменных нужно реализовать несколько функций от многих переменных либо одну функцию от всех переменных программы, что может потребовать более значительных усилий для реализации визуализаций, чем при использовании метода выделенных событий. Однако, если алгоритм оперирует сложными структурами данных, позволяющими сложные операции визуализации, то реализация функций визуализации будет

близка к реализации в случае событийно-ориентированного метода. При этом код алгоритма не будет разрастаться.

4. Data Structure and Algorithm Visualization

Это своего рода каталог, в котором представлен набор визуализаций некоторых алгоритмов. Для каждого алгоритма выделена собственная страница. Алгоритмы добавляются в каталог членами сообщества разработчиков, которые участвуют в проекте. Например, одним из таких модулей является образовательная система, целью которой является облегчить понимание алгоритма поиска с глубиной в графе. Система изготовлена с использованием java-апплетов. Пользователю сначала демонстрируется работа алгоритма на некотором автоматически сгенерированном примере, а далее от пользователя требуется повторить работу алгоритма в ручном режиме на том же примере графа. По окончании упражнения результат работы пользователя сравнивается с эталонным решением. Пользователь может просмотреть в пошаговом режиме, какие шаги были выполнены правильно.

Вершины графа изображаются кругами разного цвета, цветами обозначаются различные состояния вершин.

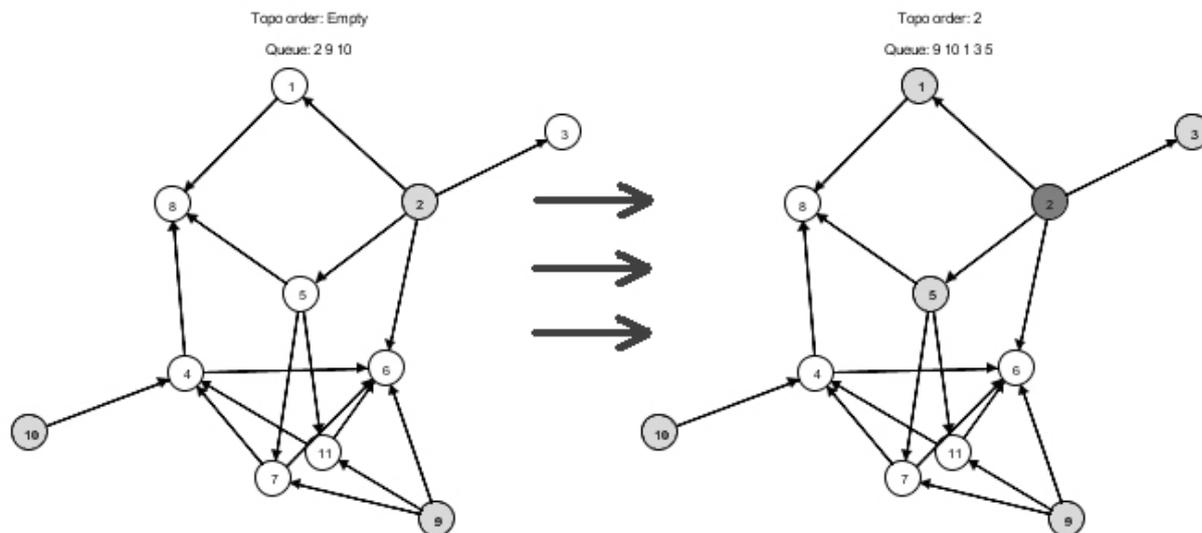


Рисунок 3 – Визуализация алгоритма топологической сортировки.

автоматически, система не позволяет задавать граф в качестве параметра. Данный алгоритм демонстрируется только для неориентированных графов, поэтому все дуги изображаются отрезками прямых без стрелок. Различные состояния дуг также помечаются цветами. Так как состояния дуг обычно связаны с состояниями вершин, инцидентных дугам, то цвет дуг часто совпадают с цветами вершин.

Добавить новые алгоритмы нельзя, как уже было указано, алгоритмы добавляются в виде самостоятельных модулей. На Рис. 2 изображены два состояния графовой модели, обрабатываемой алгоритмом поиска в глубину.

Белым цветом помечены вершины находящиеся, в начальном состоянии. Светло-Серые вершины с горизонтальной штриховкой находятся в очереди обработки. Серые вершины с вертикальной штриховкой являются необработанными соседями текущей обрабатываемой вершины. Серым цветом с наклонной штриховкой помечена текущая обрабатываемая вершина. Серым цветом без штриховки помечены посещённые вершины.

Граф может быть сгенерирован только

5. JHAVE (Java-Hosted Algorithm Visualization Environment)

Система визуализации JHAVE [13] представляет собой самостоятельный виртуальный каталог визуализаций алгоритмов, изготовленный на языке java в виде загружаемого на локальный компьютер java-приложения. Пользователь сначала должен выбрать сервер, на котором хранятся визуализации алгоритмов. На момент написания статьи был доступен всего один сервер. Далее пользователю предоставляется возможность выбрать один алгоритм из фиксированного списка и запустить визуализацию. Каждая демонстрация загружается несколько минут. Набор демонстраций алгоритмов стандартный, и расширить его невозможно. Этот проект входит в состав каталога **Data Structure and Algorithm Visualization**, упомянутого выше.

Окно демонстрации разделено на две области: левая содержит изображение состояния графической модели, правая содержит описание алгоритма в текстовом виде, реализованное с использова-

нием псевдокода и языка java. JHAVE позиционируется как образовательная система, предназначенная для обучения студентов. Целью данной системы является повышение понимания алгоритмов обработки графов.

Демонстрация работает только в пошаговом режиме, причём на каждом шаге система показывает окно с вопросом. Например, вопрос может быть о

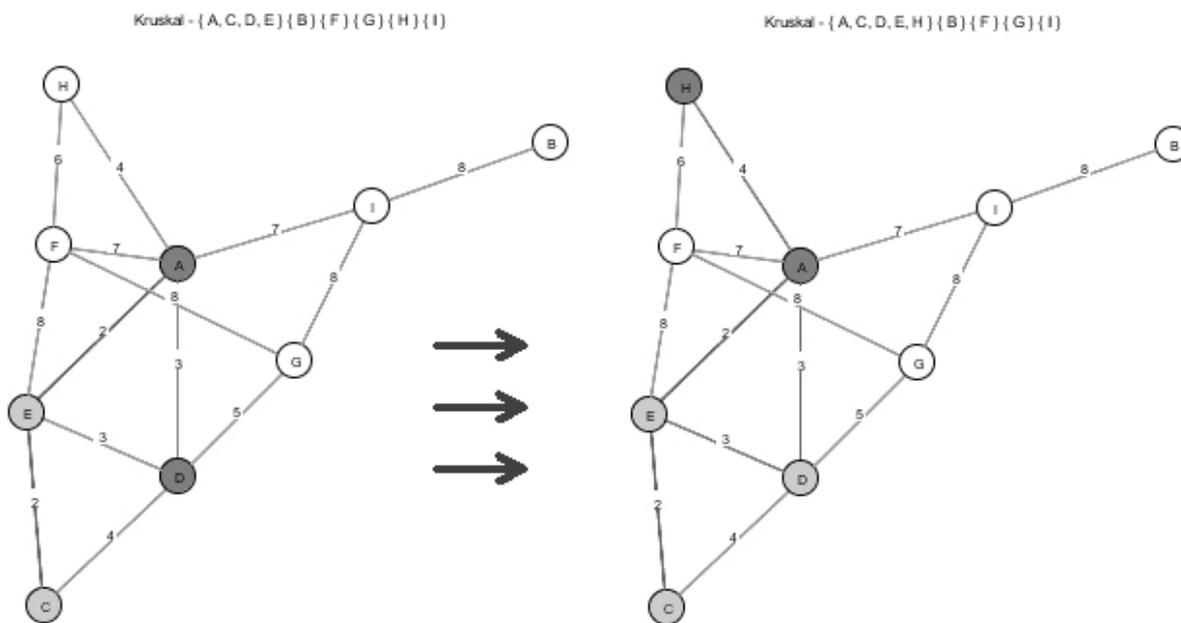


Рисунок 4 –Алгоритм Крускала для нахождения минимального остовного дерева неориентированного связного графа с взвешенными рёбрами.

том, какую вершину алгоритм выберет следующей, в ситуации, когда требуется выбрать такую вершину. Либо система может спросить, каким будет некоторый результат. Например, для алгоритма топологической сортировки система может задать вопрос о конечном порядке задолго до конца работы алгоритма. Гипотетически это может быть полезно, но может помешать пользователю вникнуть в суть алгоритма, если он изучает его впервые. Работа алгоритма демонстрируется с помощью графа и дополнительной структуры. Например, это может быть таблица или стек специальных данных.

Система визуализации не позволяет задать граф в качестве параметра.

На рисунке изображены два состояния графовой модели: первое и следующее за ним. В первом состоянии светло-серым цветом помечены вершины, которые не имеют входящих в них дуг. При этом две дополнительные структуры

отображают промежуточные данные, которые используются в процессе работы алгоритма. Очередь содержит вершины, для которых задан порядок обработки, а другая структура содержит уже упорядоченные вершины. Светло-серые вершины первыми попадают в очередь. На втором шаге видно, что тёмно-серым цветом помечена вершина, которая уже имеет определённый порядок, а светло-серым – все вершины из очереди. Следует заметить, что визуализация не содержит никакого текстового описания алгоритма, что делает задачу понимания алгоритма нетривиальной, особенно в случае, если отсутствует визуализация дополнительных структур данных.

На данном рисунке демонстрируется четвёртый шаг алгоритма, когда для компоненты связности, состоящей из вершин A, E, D, C с дугами, имеющими вес 2, 2, 3, выбирается вершина и инцидентная ей дуга минимального веса, добавление которой не вызовет образования цикла. В данном случае показано, что есть дуга с весом 3, однако она не

подходит, так как ей добавление создаст цикл из вершин А, Е, D. Добавление дуги с весом 4, инцидентной вершинам А и Н удовлетворяет условиям алгоритма. При этом на рисунке видно, что дополнительно отображаются текущие компоненты связности.

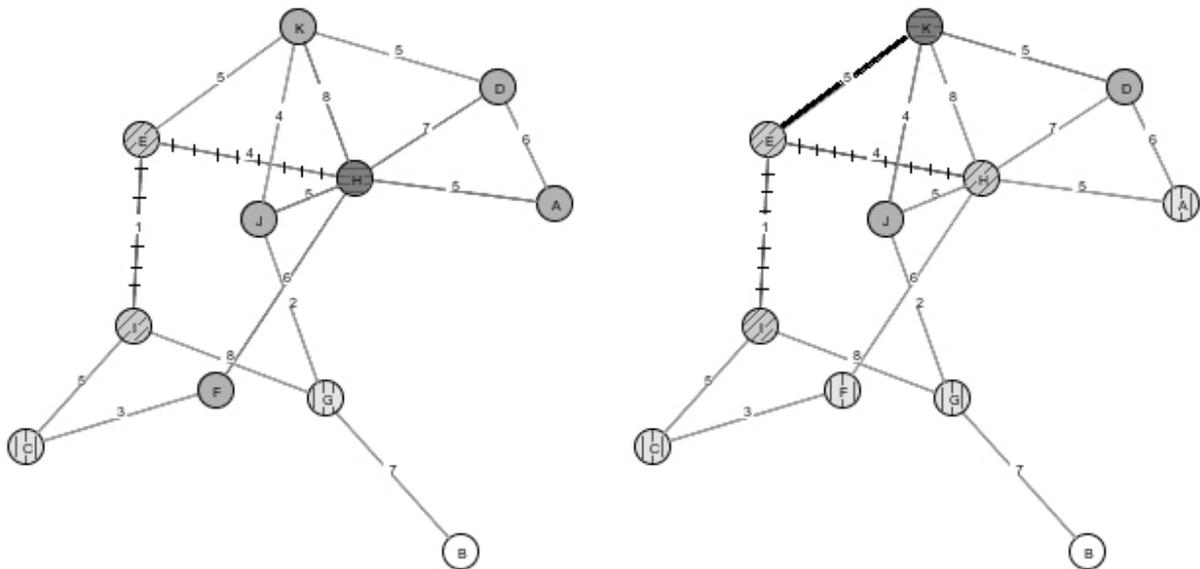


Рисунок 5 – Алгоритм Прима для нахождения минимального остовного дерева неориентированного связного графа с взвешенными рёбрами.

Серым цветом с горизонтальной штриховкой помечается вершина, которая была добавлена в результирующее дерево последней. Дуга, добавленная к дереву последней, выделяется жирной линией. Дуги, уже принадлежащие дереву, выделяются поперечной штриховкой. При переходе от состояния, изображённого в левой части рисунка, к состоянию, изображённое в правой части, была присоединена дуга с весом 5, так как она имеет минимальный вес среди всех дуг, инцидентных вершинам текущего остовного дерева. Серым цветом без штриховки цветом как раз помечены вершины из множества вершин, инцидентных вершинам из текущего остовного дерева

6. TRAKLA2

Система визуализации алгоритмов на графах TRAKLA2 [14] позиционируется как среда для обучения работе со структурами данных и алгоритмами. Процесс обучения представляет собой выполнение упражнений. Сама демонстрация

На данном рисунке демонстрируются два шага – второй и третий – алгоритма, который начал работу с вершины I. Серым цветом с наклонной штриховкой обозначаются вершины, которые принадлежат результирующему дереву.

представляет собой окно с псевдокодом и визуальным представлением графовой модели; в левой части окна располагается текстовое представление алгоритма в виде псевдокода, в правой части присутствует изображение состояния графовой модели. Система реализована на java-платформе. В окне системы выделено место для текстового представления графа-параметра в виде списка смежности. Далее на рисунках представлен пример алгоритма поиска в глубину в ориентированном связном графе. Вершины графа изображены кругами разного цвета, различные цвета также обозначают различные состояние вершин графа. Дуги графа представлены направленными отрезками. Вершины имеют текстовые подписи, соответствующие обозначениям вершин в списке смежности. В рассмотренном примере метки являются будками латинского алфавита. Упражнение представляет собой моделирование работы алгоритма на сгенерированном системой графе. Моделирование заключается в последовательном

изменении цветов вершин. Пользователь помечает определённую вершину некоторым цветом, соответствующим тому или иному состоянию вершины. Ход выполнения упражнения запоминается системой. После окончания выполнения упражнения система сравнивает

предложенный пользователем вариант решения с эталонным решением. На основании этого сравнения пользователю выставляется оценка. В случае неверного решения пользователь может просмотреть историю выполнения и сравнить своё решение с

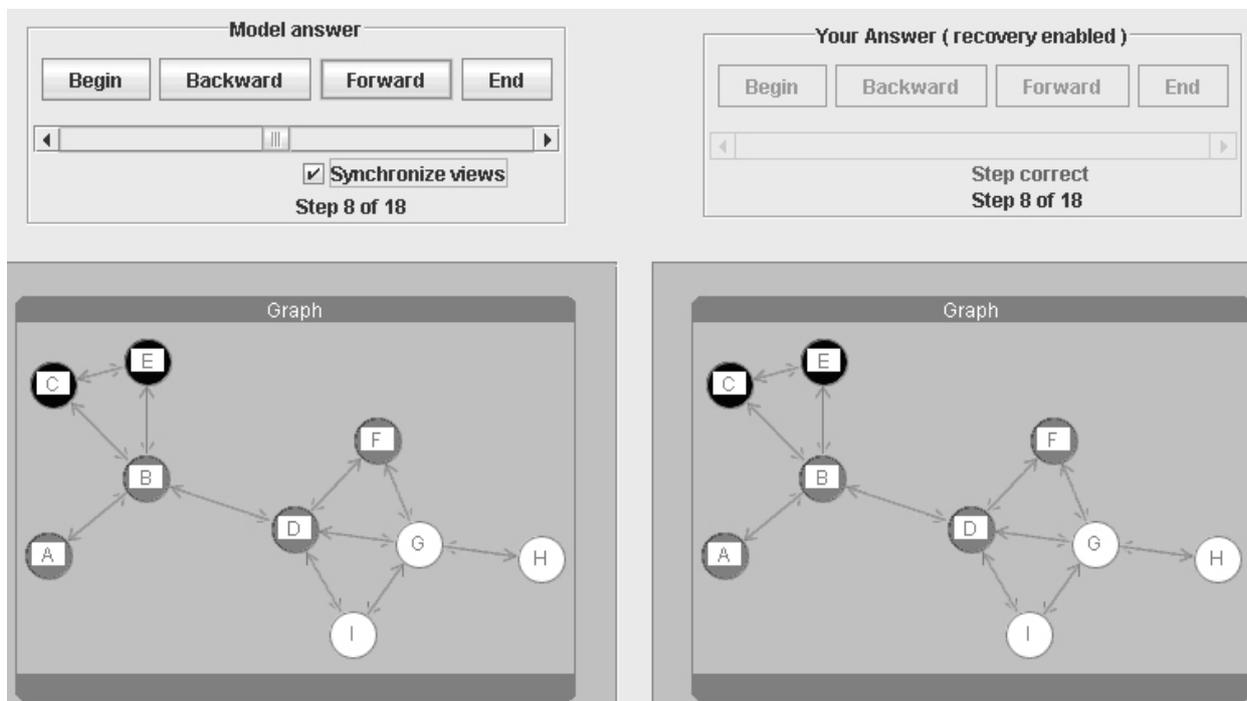


Рисунок 6 – Алгоритм поиска в глубину.

правильным решением. Во время сравнения можно отследить, где предложенное решение отклоняется от правильного решения. К сожалению, строчки текстового представления алгоритма не подсвечиваются при демонстрации эталонного решения. Демонстрация имеет только пошаговый режим.

В данном примере выполнение алгоритма поиска в глубину был начат с вершины A. В списке смежности вершины B дуги (B,A), (B,C), (B,E), (B,E) были заданы в указанном порядке. Поэтому сначала были посещены вершины C и E, а только после этого вершина D. Белым цветом помечаются вершины, которые ещё не участвовали в обходе. Чёрным цветом обозначаются вершины, которые участвовали в обходе и больше будут посещены. Серым цветом, соответственно, помечаются посещенные вершины, обработка которых ещё не закончена. На

этом рисунке изображён результат следующих действий: вершина A помечается серым, вершина B помечается серым, вершина C помечается серым, вершина E помечается серым, вершина E помечается чёрным, так как все её соседи помечены как посещённые, вершина C помечается чёрным, так как все её соседи помечены как посещённые, вершина D помечается серым, вершина F помечается серым. Как видно из рисунка, данная последовательность действий совпадает с эталонным решением.

Данный рисунок является продолжением рисунка 5. Здесь серым цветом помечается следующая белая вершина, инцидентная вершине F.

7. JAWAA

JAWAA [15] – это интерпретируемый язык для построения анимации с помо-

щью специальной библиотеки. Библиотека разработана с использованием языка java. Авторы утверждают, что на этом языке можно быстро генерировать анимации таких структур данных, как очередь, стек, граф или дерево. Рассмотрим пример алгоритма поиска в глубину, сгенерированный с помощью упомянутой библиотеки. Как видно из рисунка 8,

анимация содержит изображение графа, по которому ведётся поиск, и стека, который при этом используется. Поиск начинается в вершины 1. Зелёным цветом помечаются вершины, находящиеся в стеке или, другими словами, вершины, которые были посещены. Синим цветом помечены вершины, которые были посещены, и больше посещаться не будут.

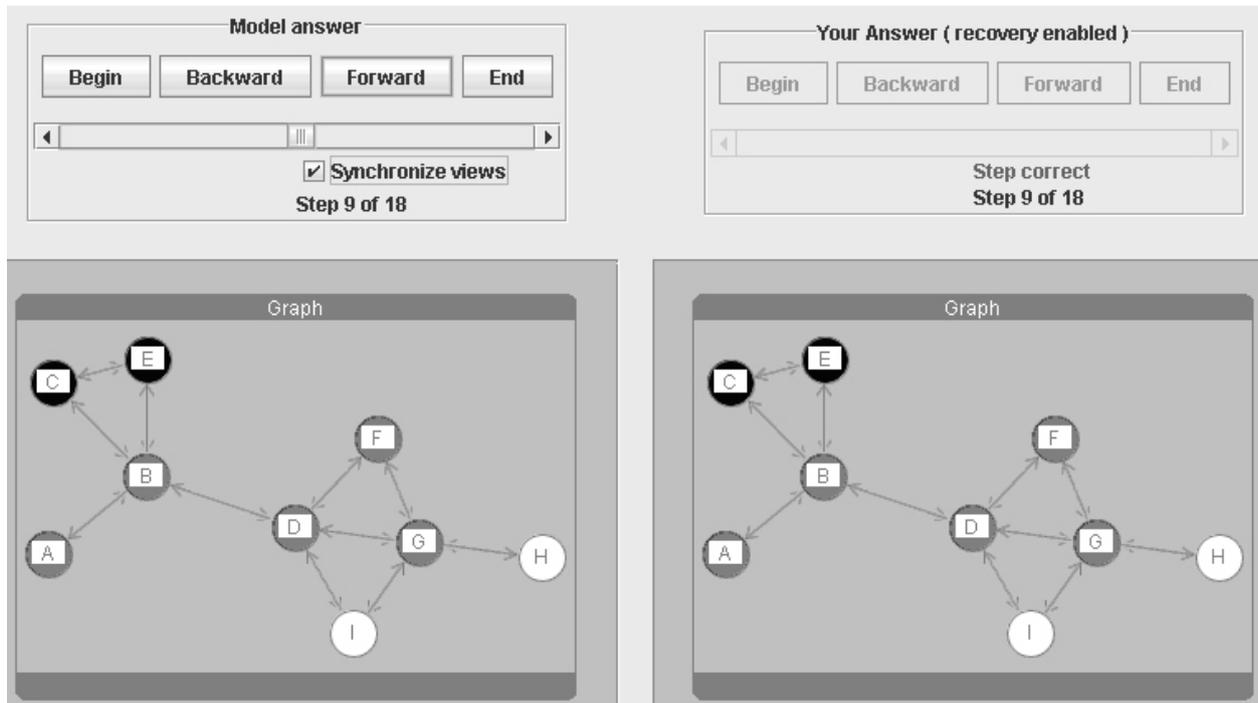


Рисунок 7 – Алгоритм поиска в глубину.

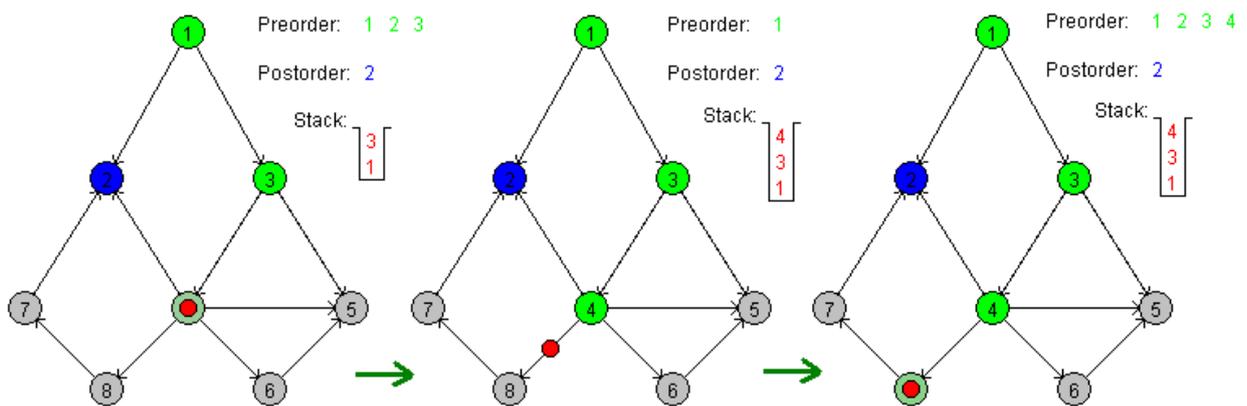


Рисунок 8 – Визуализация алгоритма поиска в глубину с помощью средств JAWAA.

Серым цветом помечены вершины, которые ещё не посещались. Отличительной особенностью данной визуализации является наличие красного маркера, помечающего текущую вершину, либо текущий переход по дуге. Маркер движется

непрерывно по дугам от вершины к вершине.

Также во время анимации дополнительно отображается содержимое стека. Вообще говоря, показывать его необязательно, так как зелёные вершины являются содержимым стека.

8. Animal

Система визуализации Animal позиционируется как инструмент, созданный для визуализации алгоритмов, причём не только алгоритмов на графах [16]. Инструмент разработан с использованием языка java. Каждая анимация может храниться в виде файла в специальном формате; такие файлы имеют расширение

aml. Это означает, что созданные анимации можно сохранять для повторного использования. Для генерации aml-файлов используется специально созданный инструмент, так называемый *генератор анимаций*.

Рабочий генератор найти не удалось, но на рисунке 9 представлен пример анимации алгоритма, сгенерированного из сохранённого ранее aml-файла.

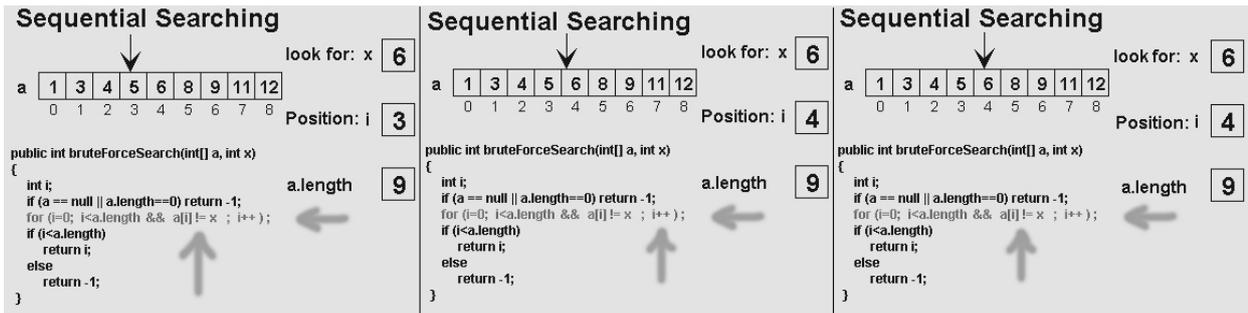


Рисунок 9 – Алгоритм последовательного поиска в массиве.

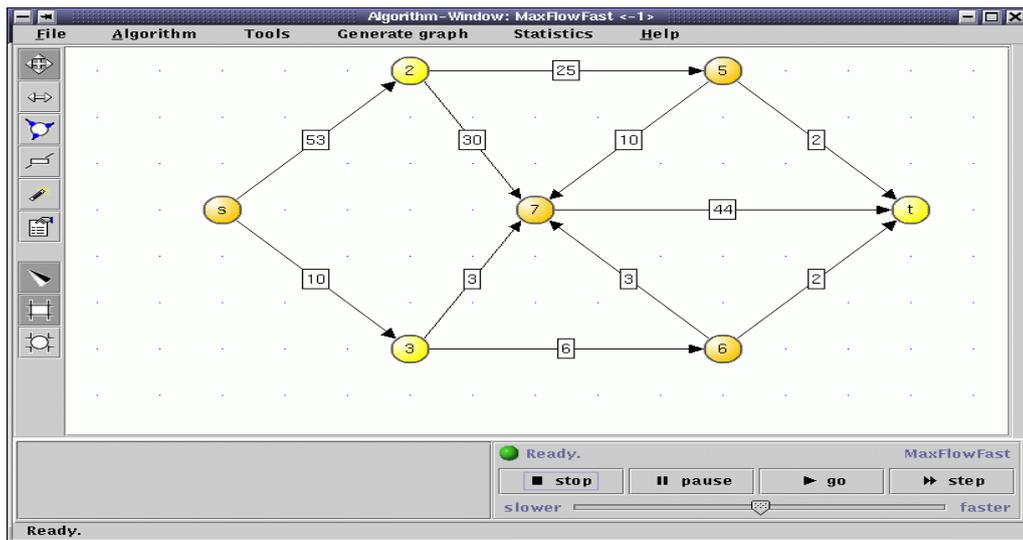


Рисунок 10 – Инструмент визуализации алгоритмов Evega. Окно графического редактора.

На данном рисунке показаны три кадра из анимации, которая демонстрирует переборный алгоритм поиска в массиве. Массив a представлен однострочной таблицей. Как видно из текста алгоритма, используется цикл с индексом. Позиция стрелки, расположенной над ячейками таблицы, соответствует текущему значению индекса. Текущая выполняемая строка алгоритма подсвечивается синим цветом, а текущая выполняемая инструкция дополнительно подсвечивается красным цветом. Более

сложных примеров обнаружить не удалось.

9. Evega (Educational Visualization Environment for Graph Algorithms)

Пользовательский интерфейс инструмента Evega [17] представляет собой окно графического редактора. Он может использоваться для построения графов и анимации алгоритмов. Граф может быть

задан с помощью редактора либо загружен из файла. Во время анимации алгоритма также возможна анимация дополнительных структур данных, таких как стек или очередь. На рисунке 10 показано окно редактора графов. Утверждается, что при реализации алгоритма допускаются различные операции с элементами графа, такие как удаление или добавление вершин, изменение меток

элементов. Инструмент Evega реализован как автономное java-приложение. Утверждается, что инструмент может быть использован для генерации пользовательских анимаций алгоритмов с использованием языка java и библиотек поставляемых в комплекте инструмента Evega. Визуализация алгоритма представляет собой последовательное изменение цвета у вершин и текстовых меток у дуг.

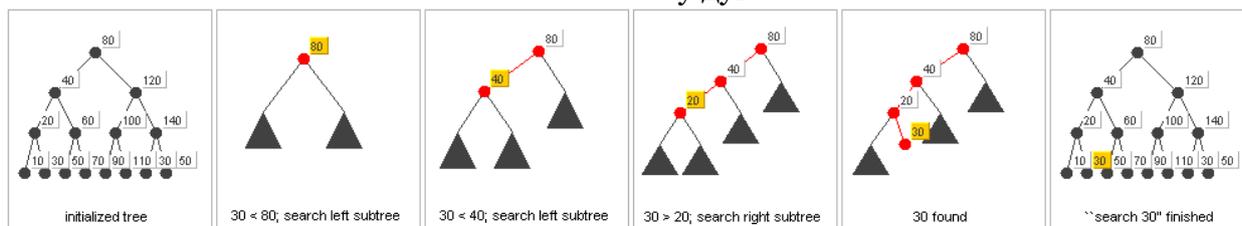


Рисунок 11 – Визуализация алгоритма поиска в бинарных деревьях.

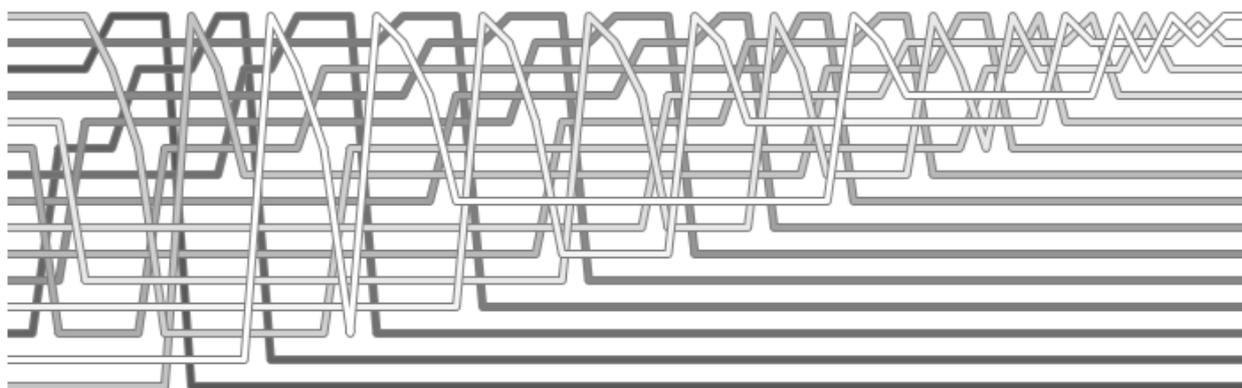


Рисунок 12 – Визуализация пирамидальной сортировки массива с помощью статичного изображения.

В [18] для следующего примера не указано, с помощью какой системы была получена данная визуализация. На рисунке 11 показаны несколько кадров из визуализации алгоритма поиска в бинарном дереве.

Первый кадр содержит изображение бинарного дерева, по которому осуществляется поиск. На следующих кадрах дерево отображается схематично, а именно, красным кругом обозначается текущая тестируемая вершина, а поддеревья, корнями которых являются потомки тестируемой вершины, обозначаются чёрными треугольниками. В процессе работы алгоритма на каждом шаге выбирается некоторое поддерево, далее оно раскрывается таким образом, что его корень становится текущей тестируемой вершиной, и соответственно помечается красным цветом, а поддеревья, имеющие

потомков тестируемой вершины в качестве корней, помечаются чёрными треугольниками. Конечный кадр содержит исходное изображение дерева, с той разницей, что на нём помечена вершина, соответствующая искомому значению.

Следующий пример визуализации алгоритмов представляет собой нетрадиционный подход, в том смысле, что визуализация динамического процесса представляет собой статическое изображение. В данном примере [19] показывается, как можно представить алгоритм сортировки массива в виде статического изображения. В данном примере нужно считать, что белый цвет обозначает минимальный элемент массива, а чёрный – максимальный элемент. Данную визуализацию удобно понимать следующим образом. Индексы элементов массива следует рассматривать как координаты

элементов массива на числовой прямой. Тогда во время работы алгоритма эти координаты будут изменяться, а именно, в момент операции перестановки элементов. Если зафиксировать моменты времени, когда происходят перестановки элементов массива, то для каждого элемента массива можно построить соответствие между этим моментами времени и соответствующими этим моментам индексами. Так для каждого элемента мас-

сива можно получить дискретную зависимость индекса от времени. Если после этого соединить точки, соответствующие одному элементу массива, то для каждого элемента получится кусочно-линейный график зависимости “координаты” от времени. Если теперь совместить все графики на одном рисунке, обозначив графики разными цветами, то получившаяся картина будет выглядеть как рисунок 12.

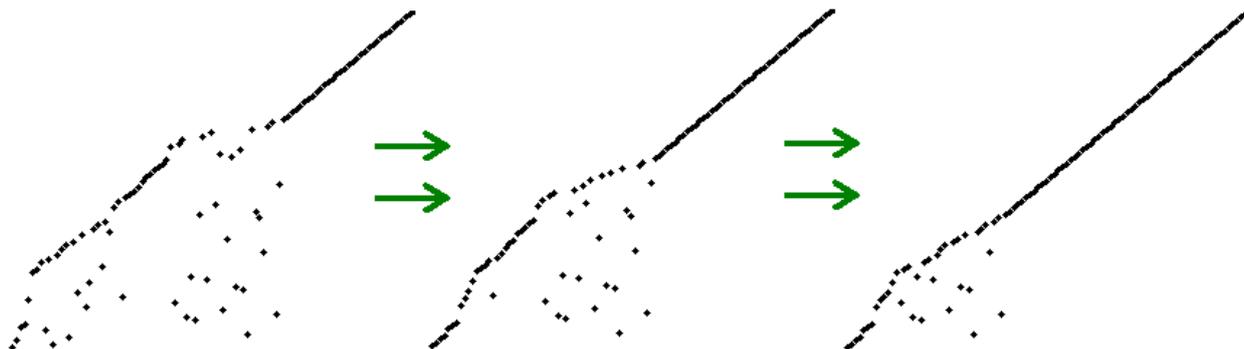


Рисунок 13 – Визуализация сортировки массива методом пузырька.

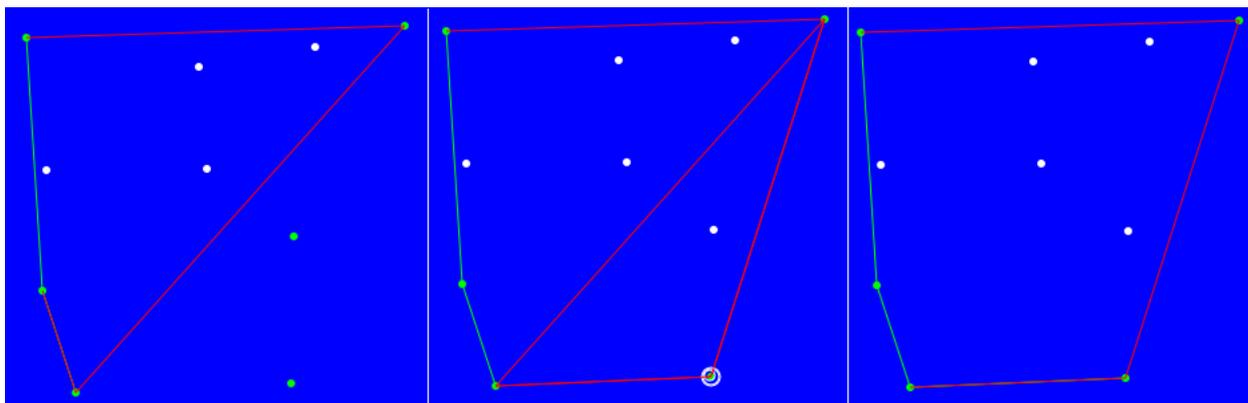


Рисунок 14 – Визуализация алгоритма поиска выпуклой оболочки для множества заданных точек плоскости.

В следующем примере визуализация строится с использованием множества точек на плоскости, при том что одна из координат отражает текущий индекс элемента массива, а другая координата отражает значение элемента массива. Таким образом, отсортированный массив можно изобразить как множество точек на плоскости, близкое к линейной функции.

Однако такие визуализации можно применять к алгоритмам сортировки. Для визуализации алгоритмов, оперирующих более сложными объектами, такие методы не подходят.

Данный пример найден на информационном портале колледжа Franklin & Marshall, посвящённом визуализации алгоритмов в компьютерной геометрии [20]. Визуализация алгоритмов на графах не является специализацией этого портала. Однако на нём есть полезные идеи, такие как, использование видео файлов для изображения работы алгоритмов, а сам процесс работы алгоритма описывается отображением последовательности состояний. Однако работа только демонстрационная, для заранее фиксированных решённых задач. На ри-

сунке 14 изображена визуализация к задаче, которую можно сформулировать следующим образом. Для заданного множества точек на плоскости построить выпуклый многоугольник, содержащий все заданные точки, вершины которого также принадлежат заданному множеству точек.

Сначала алгоритм выбирает самые “нижние”, “верхние”, “левые” и “правые”, и по ним строит первое приближение к выпуклой оболочке, являющееся

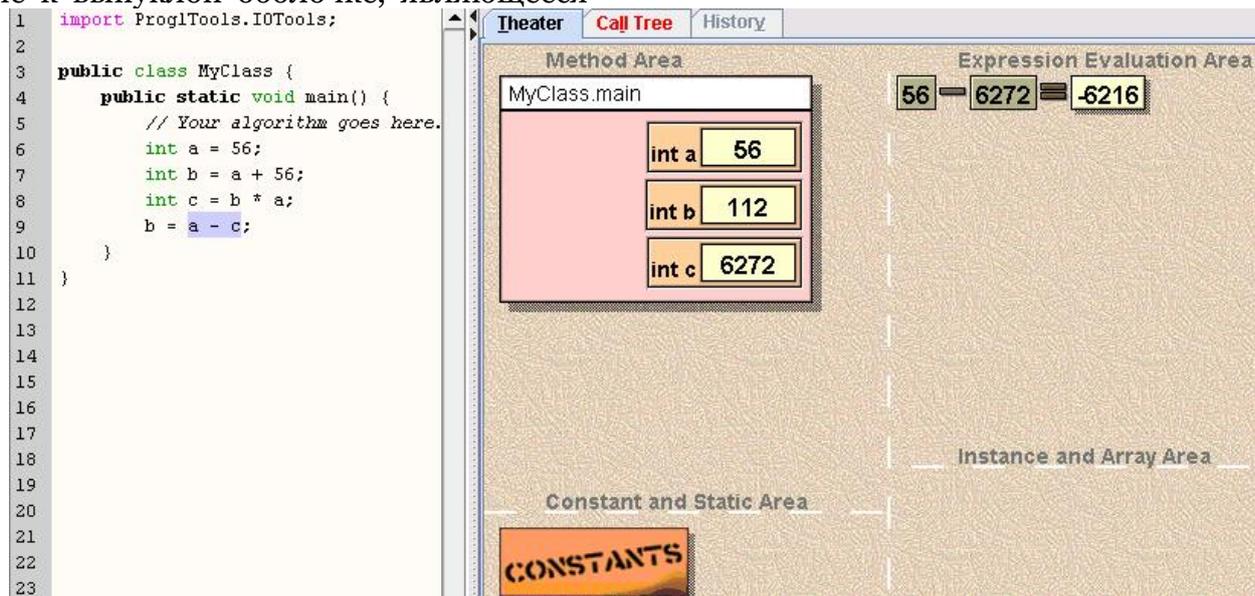


Рисунок 15 – Визуализация инструкции вычисления разности двух операндов.

оболочки. На рисунке 14 изображена итерация этого процесса. На втором фрагменте найдена точка из заданного множества, не лежащая на границе или внутри выпуклой оболочки, а на третьем фрагменте виден результат замены одного ребра на два, соединяющих найденную точку и вершины, уже принадлежащие выпуклой оболочке.

10. Jeliot

Система Jeliot [21] предоставляет возможность визуализации процесса исполнения программ, написанных на языке java. Каждый шаг программы, например, вызов метода или переменной, отображается на экране в процессе выполнения, позволяя пошагово проследить ход исполнения программы. Программу можно ввести в окне программы или загрузить сохранённый пример. java программа может быть тут же визуализирована. Система Jeliot способна обработать

четырёхугольником. Далее алгоритм ищет точки, не попавшие внутрь оболочки. Если такая точка существует, то ближайшее к ней ребро выпуклой оболочки удаляется, и вместо него добавляется два отрезка так, чтобы найденная точка стала вершиной выпуклой оболочки, а новые отрезки соединяли найденную точку и вершины удалённого ребра. Процесс продолжается, пока есть точки, не лежащие внутри выпуклой

большую часть синтаксических конструкций языка java. Особым достижением отмечается то, что приложение способно отображать объектно-ориентированные возможности объектов программы, как например, наследование классов.

В приведённом на Рис. 15 и 16 примере визуализируется выполнение операции разности двух чисел хранящихся в некоторых переменных. Как видно на рисунке, каждая переменная представлена некоторым объектом визуализации. В данном случае переменным соответствуют прямоугольники, которые содержат название переменной, её тип и значение. Вся область визуализации разделена на несколько частей, каждая из которых предназначена для отображения определённых объектов. Во время визуализации процесса объекты визуализации перемещаются из одной части в другую. Например, вычисление разности

двух чисел отображается в области, предназначенной для визуализации вычисляемых выражений. Главная же область содержит объекты, визуализирующие локальные переменные текущей исполняемой функции. На Рис. 15 и 16 демонстрируется исполнение выделенной строки кода, выделенной левой части окна, где представлен текст исполняемой программы. То, что новое вычисленное значение записывается в переменную *b*, демонстрируется с помощью пере-

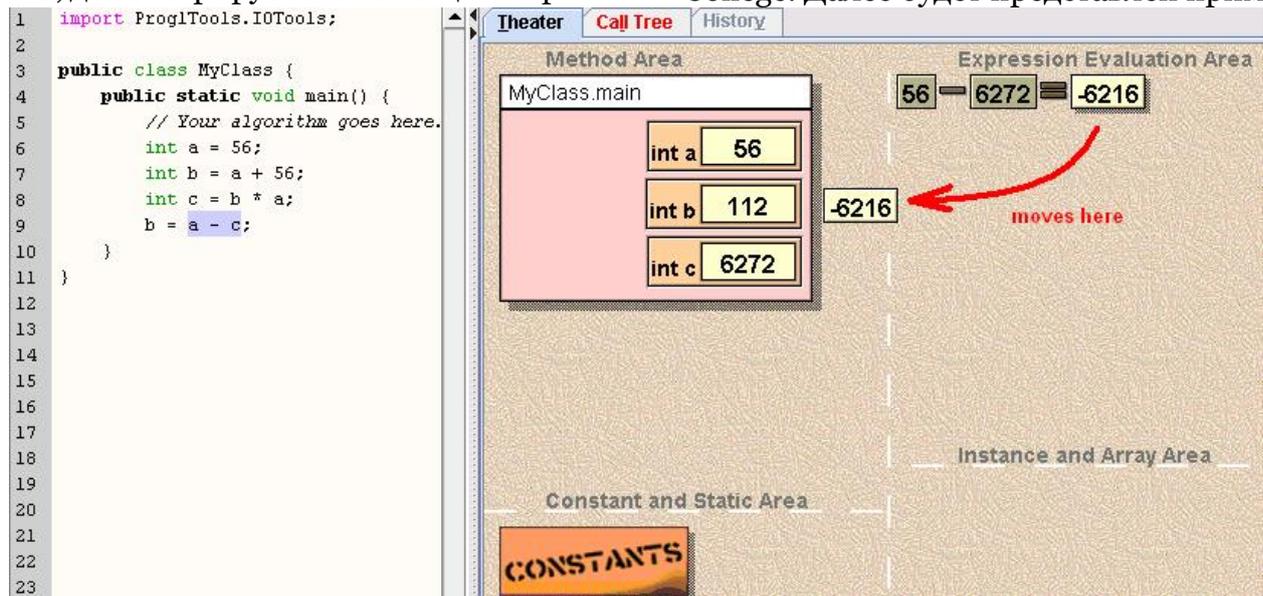


Рисунок 16 – Визуализация инструкции вычисления разности двух операндов.

визуализации, построенный для алгоритма Прима.

Модель системы доказательства корректности алгоритмов состоит из нескольких частей:

1. Абстрактные графовые объекты (графы, подграфы, пути).
2. Элементарные графовые объекты, такие как вершины и дуги.
3. Описания доказываемых теорем и инвариантов, которые проверяются в процессе доказательства.
4. Динамическая помощь, описывающая объекты появляющиеся на экране.
5. Описание диалога доказательства.

Сначала пользователь знакомится с алгоритмом и представлением в псевдокоде. Далее формулируется теорема в терминах инвариантов, которые должны сохраниться при работе алгоритма. Если

мещающегося прямоугольника, содержащего число, которые двигается из области вычисления выражений в область, где хранятся локальные переменные. На Рис. 16 это отражено красной стрелкой.

11. Система визуализации доказательства корректности алгоритмов

Система разрабатывалась в Wellesley College. Далее будет представлен пример

инварианты сохранились, то это означает, что теорема верна и алгоритм работает верно. Пользователь наблюдают пошаговое доказательство, наблюдая неизменность инвариантов на каждом шаге алгоритма. При таком пошаговом просмотре есть возможность сделать несколько шагов назад, чтобы улучшить понимание алгоритма.

Разработчик, который использует данную систему, обязан предоставить следующие сущности:

6. Вступительную часть, в которой обозначается проблема, её значение и дополнительная информация.

1. Анимацию алгоритма, решающего проблему. Данный прототип системы был

реализован с использованием HyperCard.

2. Диалог доказательства в терминах текстового файла в специальном формате.

3. Анимация базовых и вступительных шагов доказательства.

4. Файл справки, содержащий информацию об объектах.

На рисунках 17–25 демонстрируется пример работы системы доказательства корректности алгоритмов на графах. Из рисунков видно, что элементы графа изображаются простыми кругами и прямыми отрезками, а выделение элементов производится с помощью различных цветов и толщины линий. Как нетрудно заметить, вся реализация процесса визуализации алгоритма лежит полностью на разработчике. Для реализации новой демонстрации нужно процесс разработки начинать заново.

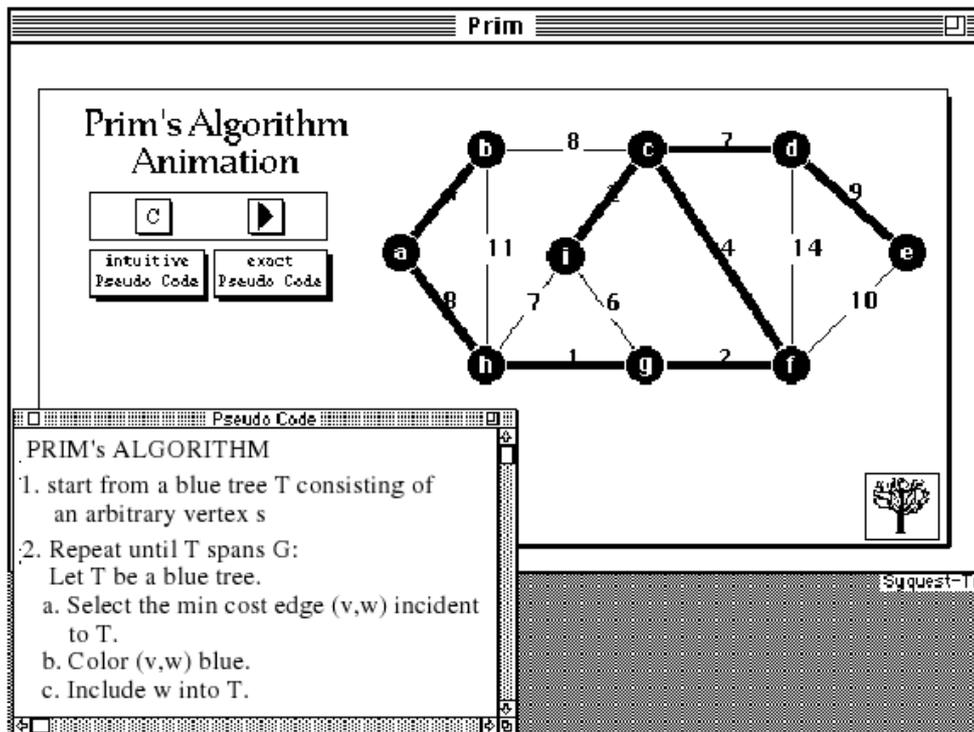


Рисунок 17 – Первый шаг анимации алгоритма Прима в системе доказательства корректности алгоритмов на графах. На первом шаге приводится пример графа, на котором будет проводиться доказательство, и текстовое представление алгоритма.

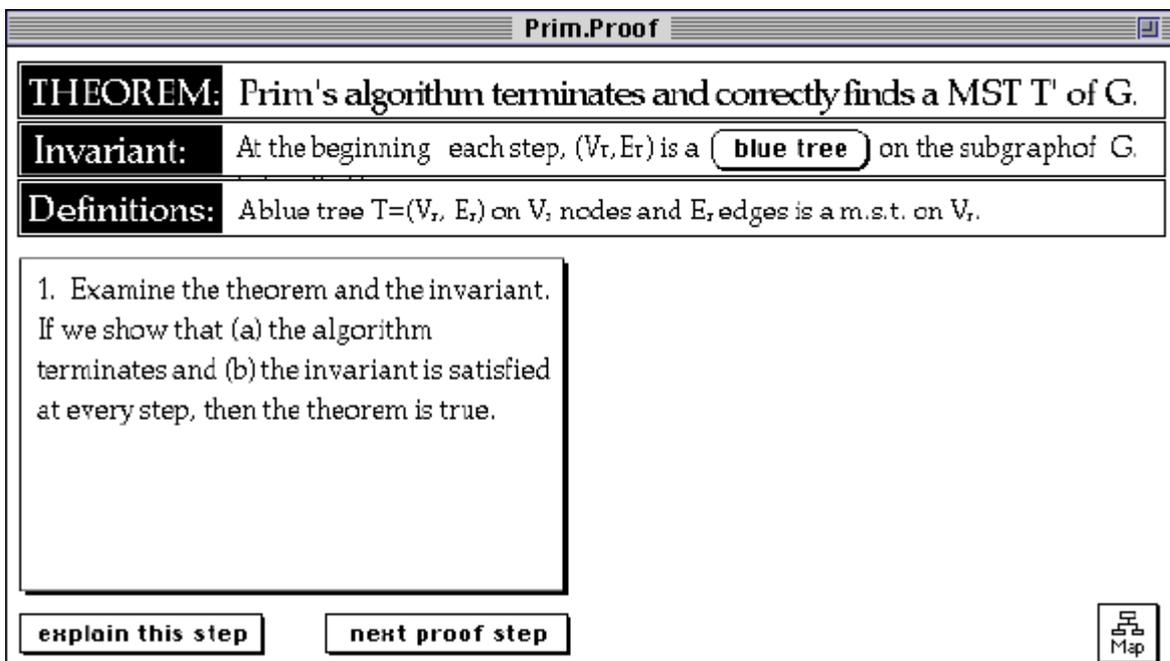


Рисунок 18 – Второй шаг анимации алгоритма в системе доказательства корректности. На втором шаге формулируется теорема в терминах инвариантов, сохранение которых будет показываться в процессе работы алгоритма.

Prim.Proof

THEOREM: Prim's algorithm terminates and correctly finds a MST T' of G .

Invariant: At the beginning each step, (V_T, E_T) is a **blue tree** on the subgraph of G .

Definitions: A blue tree $T=(V_T, E_T)$ on V_T nodes and E_T edges is a m.s.t. on V_T .

3. The invariant holds true for $T=\{s\}$



number of vertices in graph = 1

explain this step

next proof step

Map

Рисунок 19 – Третий шаг анимации алгоритма в системе доказательства корректности. Доказательство производится методом математической индукции по числу шагов, необходимых для построения остовного дерева. Решение задачи для случая, когда граф состоит из одной вершины без дуг. Этот шаг есть база индукции.

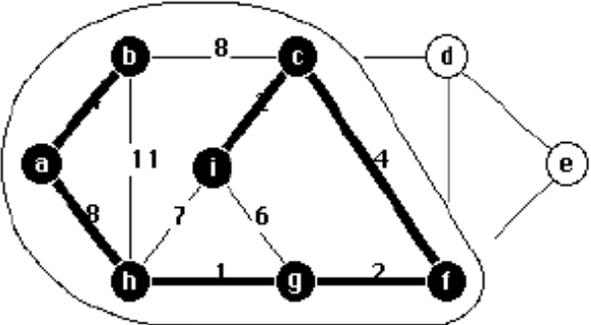
Prim.Proof

THEOREM: Prim's algorithm terminates and correctly finds a MST T' of G .

Invariant: At the beginning each step, (V_T, E_T) is a **blue tree** on the subgraph of G .

Definitions: A blue tree $T=(V_T, E_T)$ on V_T nodes and E_T edges is a m.s.t. on V_T .

4. Assumption hypothesis: assume the invariant holds at the k -th step and let's assume that the invariant is violated for the first time in the $(k+1)$ st step with the incorporation of the **cut-edge** (v,w) and vertex w into the blue tree V_T .



T is a blue tree

next proof step

Map

Рисунок 20 – Четвёртый шаг анимации алгоритма в системе доказательства корректности. Формулировка индукционного перехода метода математической индукции в терминах заданной задачи. В данном случае используется метод доказательства от противного.

Prim.Proof

THEOREM: Prim's algorithm terminates and correctly finds a MST T' of G .

Invariant: At the beginning each step, (V_T, E_T) is a **blue tree** on the subgraph of G .

Definitions: A blue tree $T=(V_T, E_T)$ on V_T nodes and E_T edges is a m.s.t. on V_T .

5. Depending on how w is connected to V_T there are two possible cases:

a. There exists a blue tree T' on $V_T \cup \{w\}$ with exactly one edge (x, w) incident on w . (Note that x is in V_T).

b. Every blue tree on $V_T \cup \{w\}$ has two (or more) edges incident on w .

next proof step **show definition of cut** Map

Рисунок 21 – Пятый шаг анимации алгоритма в системе доказательства корректности. Здесь приведены рассуждения индукционного перехода. Для подграфа меньшего размера построено остовное дерево, и есть два случая того, как оставшиеся вершины могут быть связаны с вершинами подграфа.

Prim.Proof

THEOREM: Prim's algorithm terminates and correctly finds a MST T' of G .

Invariant: There exists a blue tree T' on $V_T \cup \{w\}$ with exactly one edge (x, w) incident on w .

Definition: T' is a m.s.t. on V_T .

6. (explanation of **5a**)
 By the invariant at step k , $\text{cost}(T' - (x, w)) = \text{cost}(T)$, and by the algorithm,
show algorithm
 $\text{cost}(v, w) \leq \text{cost}(x, w)$, since x is in V_T .
show v 'v'
 $\Rightarrow \text{cost}(T' \cup (v, w)) \leq \text{cost}(T')$, a
 \Rightarrow contradiction.

next proof step **Map**

You choose a node as v that fulfills the requirements of the proof.

Рисунок 22 – Шестой шаг анимации алгоритма в системе доказательства корректности. Доказательство противоречивости первого варианта связи оставшихся вершин с вершинами подграфа.

Prim.Proof

THEOREM: Prim's algorithm terminates and correctly finds a MST T' of G .

Invariant: (V_r, E_r) is a **blue tree** on the subgraph of G .

Definition: Every blue tree on $V_r \cup \{w\}$ has two (or more) edges incident on w .

7. (explanation) Let the two edges incident on w be (x, w) and (y, w) , where x and y are in V_r . Now consider the subgraph $H = T \cup (x, w) \cup (y, w)$. H will have exactly one **cycle C**, and C will contain edges (x, w) and (y, w) .

next proof step

Рисунок 23 – Седьмой шаг анимации алгоритма в системе доказательства корректности. Доказательство противоречивости второго варианта связи оставшихся вершин с вершинами подграфа.

Prim.Proof

THEOREM: Prim's algorithm terminates and correctly finds a MST T' of G .

Invariant: At the beginning each step, (V_r, E_r) is a **blue tree** on the subgraph of G .

Definitions: A blue tree $T=(V_r, E_r)$ on V_r nodes and E_r edges is a m.s.t. on V_r .

9. **5b** Without loss of generality, assume that x was admitted to the blue tree before y , and let (u,y) be in C . **show a 'u'** Since both (x, w) and (y, w) are less costly than (u, y) , edges (x, w) and (y, w) would have to have been brought into the blue tree before edge (u, y) was a least costly cut-edge. \Rightarrow contradiction.

next proof step

Рисунок 24 – Доказательство противоречивости второго варианта связи оставшихся вершин с вершинами подграфа (продолжение).

Prim.Proof

THEOREM: Prim's algorithm terminates and correctly finds a MST T' of G .

Invariant: At the beginning each step, (V_r, E_r) is a **blue tree** on the subgraph of G .

Definitions: A blue tree $T=(V_r, E_r)$ on V_r nodes and E_r edges is a m.s.t. on V_r .

10. Since neither case **5a** nor **5b** can happen, $T \cup \{v,w\}$ is a MST.

repeat proof

Рисунок 25 – Заключительный шаг анимации алгоритма в системе доказательства корректности. Так как предположение, сделанное на шаге индукционного перехода, привело к противоречию, то оно ложно.

12. GDR: инструмент визуализации алгоритмов на графах

Утверждается, что благодаря тому, что GDR [22] имеет простой дизайн, несложную графику и естественный программный интерфейс, система может быть использована для реализации широкого класса анимаций. Также утверждается, что для приведённых ниже визуализаций на имплементацию потребовалось не более двух часов. Среди таких визуализаций присутствуют: алгоритм построения покрывающего дерева с взвешенными дугами, поиск в глубину для ориентированных и неориентированных графов, алгоритм выделения двусвязных компонент.

12.1. Поиск в глубину для случая ориентированных графов

Необработанные вершины изображаются белыми кругами, вершины в стеке – серыми, вершины, которые посещены и полностью обработаны – чёрными. Метки вершин показывают два числа –

моменты времени начала обработки вершины и конца обработки вершины (соответственно, для прямого порядка обхода и обратного). Дуги дерева подсвечены и дуги помечены буквой С, если они пересекаются, буквой F, если направлены вперёд, либо буквой B, если направлены назад.

Пользователь может выбирать стартовую вершину для поиска и также может выбирать различные последовательности дуг в списках смежности, чтобы получить различные поисковые деревья для заданного графа. Внешний цикл алгоритма побуждает пользователя выбирать стартовую вершину каждый раз в начале каждой итерации как показано на Рис. 26. Вершины 4 и 5 недоступны (не были достигнуты) для поиска, если стартовая вершина есть 0. Порядок необработанных дуг в списке смежности может быть изменён в любое время, когда включен режим паузы для анимации. Дуги могут быть вставлены в нужный список или удалены из него.

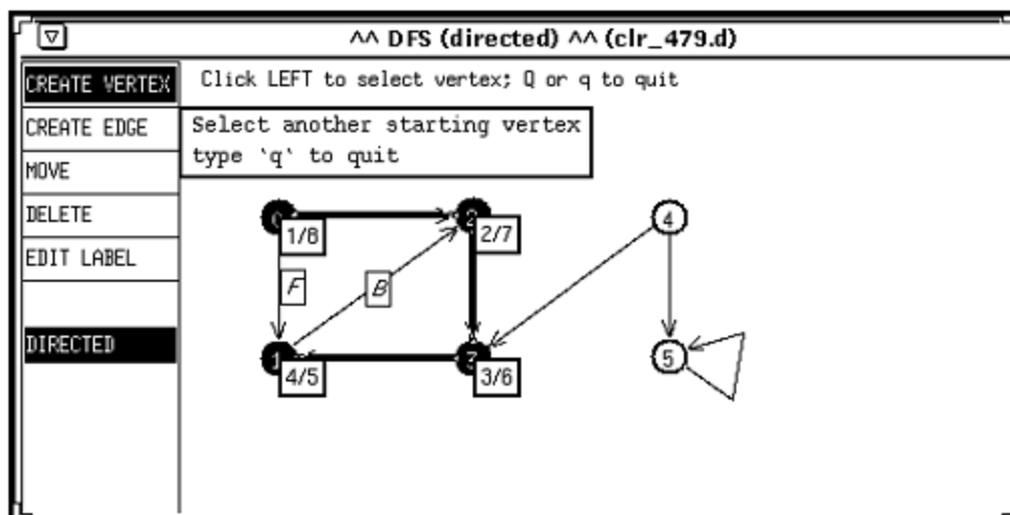


Рисунок 26 –GDR. Анимация поиска в глубину в ориентированном графе.

12.2. Визуализация алгоритма минимизация числа состояний в конечном автомате

Это пример более сложной анимации (сложной только потому, что алгоритм нетривиален). Авторы системы утверждают, что алгоритм можно реализовывать за 12 часов с использованием GDR в качестве средства отладки. Анимации DFA минимизации сначала требует от пользователя или нарисовать DFA, или загрузить DFA из файла. Метки дуг обозначают переходы. В следующей фазе от пользователя требуется установить конечные состояния с помощью операции подсвечивания (или снять выделение с подсвеченных состояний). Алгоритм работает, успешно разделяя множество состояний на классы эквивалентности (основываясь на теореме Майхила–Неруда). На протяжении всей анимации метки вершин показывают классы эквивалентности, которым они принадлежат. В конце алгоритм выбирает состояние-представитель из каждого класса и перерисовывает анимацию только с такими состояниями. На рис. 27 класс o разделён на два класса. Помеченные звёздочками состояния станут частью нового класса 2. Конечный результат показан на Рис. 28.

Редактор графов поддерживает следующие операции. Новый граф можно ввести вручную, добавляя новые вершины и

дуги. Новая вершина создаётся с помощью мыши, можно создать несколько вершин, но есть ограничение на количество, а именно 100 вершин. Вершины будут пронумерованы автоматически, начиная с нуля. Чтобы создать новую дугу, нужно последовательно отметить начальную и конечную вершины. При этом наличие пересечений дуг и вершин никак не контролируется. Дуги можно пометить любыми символами. Также вершины можно перемещать, – это может помочь получить более читаемое изображение. Чтобы избежать ситуаций некорректного изображения нескольких дуг, инцидентных одной и той же паре вершин, система пытается оптимизировать расположение дуг с помощью встроенных эвристик, однако реализованные эвристики не всегда справляются с данной задачей. Построенный граф можно сохранить в файл для дальнейшего использования. После введения графа-параметра можно запустить на нём алгоритм.

13. Higres

Система Higres [4, 5] создавалась в 1996–1999 гг. Данная система предназначена для создания и визуализации иерархических графовых моделей, представляющих собой иерархические графы с заданной семантикой.

Иерархический граф состоит из вершин, дуг и фрагментов. Вершины и дуги

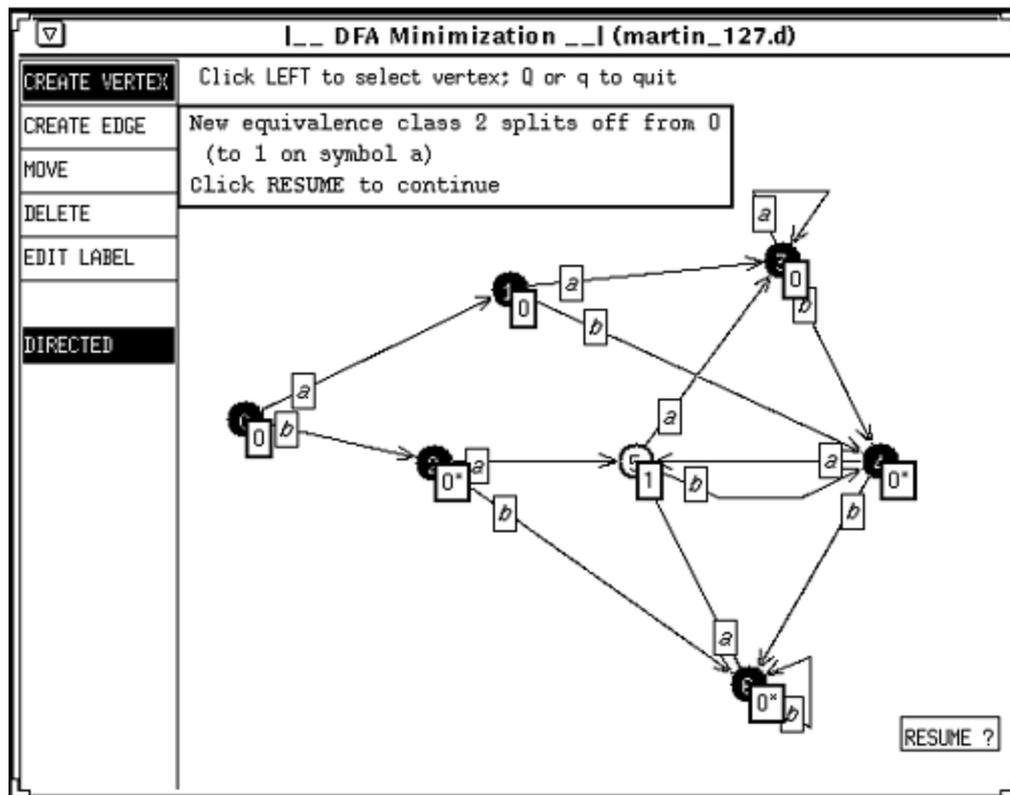


Рисунок 27 – DFA-минимизация в процессе выполнения алгоритма.

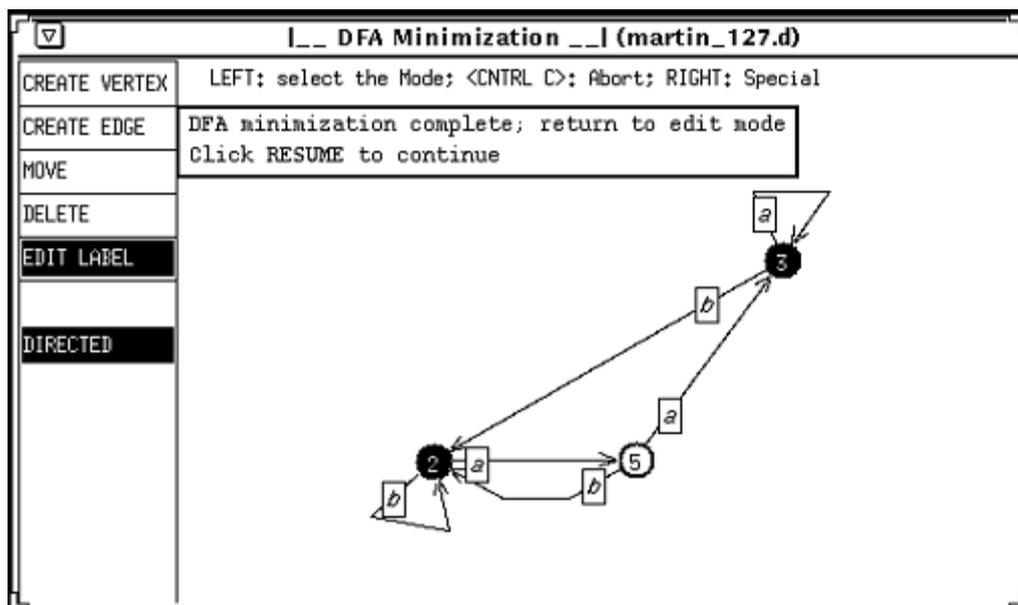


Рисунок 28 – DFA-минимизация завершена.

представляют собой обычный граф, который может быть ориентированным или неориентированным. Также на графе задано дерево фрагментов, где фрагменты есть некоторые подграфы, а дерево задаёт отношение вложенности на множестве фрагментов. Фрагменты могут быть либо вложенными, либо не иметь общих вершин. Набор всех вершин определяет главный фрагмент

иерархического графа. Строгое определение иерархических графов и других понятий, связанных с ними, можно найти в [1, 23]. Пример иерархического графа, созданного в системе Nigres, приведен на Рис. 29.

Возможности визуализации включают:

- различные формы и стили вершин;

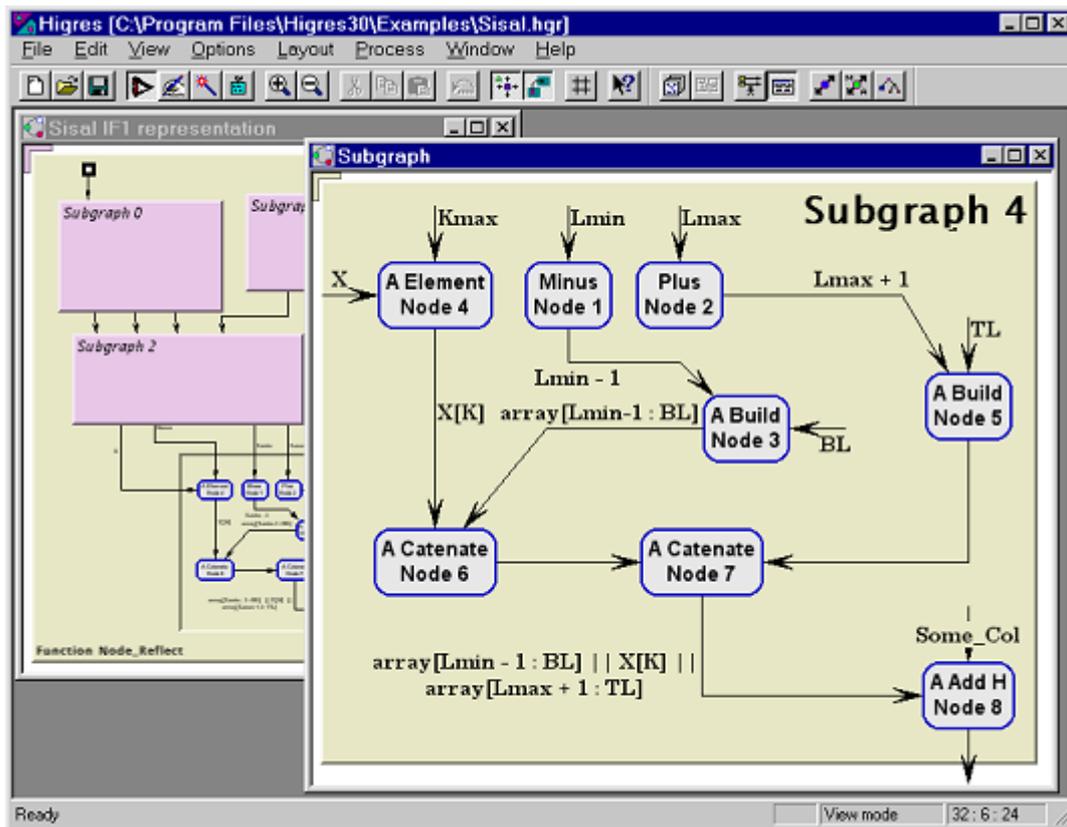


Рисунок 29 – Иерархический граф в системе HIGRES.

- дуги в виде ломаных линий и гладких кривых;
- различные стили линий и стрелок дуг;
- выбор цвета для всех элементов графа;
- возможность задавать произвольный масштаб изображения;
- возможность перемещать текст дуги вдоль ее линии;
- позиционирование внешнего текста вершины в любом месте около ее границы;
- выбор шрифта для всех элементов графа;
- два формата графического вывода;
- широкий набор опций визуализации.

Система HIGRES содержит три встроенных алгоритма размещения графов на плоскости. Первый — известный метод сил, второй — улучшенная версия первого, третий — алгоритм размещения деревьев с помощью уровней. Все алгоритмы реализованы во внешних модулях, прилагаемых к системе. Одним из достоинств системы является удобный

интуитивный пользовательский интерфейс. Главное окно системы содержит меню, несколько панелей инструментов и панель состояния. Внутри главного окна можно открывать произвольное количество окон фрагментов, используя интерфейс MDI.

В системе существует возможность расширения путем добавления новых модулей двух типов: во-первых, содержащих алгоритмы размещения графов на плоскости, и, во-вторых, таких, с помощью которых можно производить семантическую визуальную обработку графовых моделей, то есть выполнять алгоритмы на графах, визуально наблюдая результат каждого шага алгоритма.

Для запуска алгоритма в системе пользователь должен выбрать соответствующий внешний модуль. Результат работы модуля можно наблюдать непосредственно во время его работы. Кроме того, в системе предусмотрена возможность прокрутки в обратном направлении и пошагового выполнения. В отдельном окне показывается протокол исполнения алгоритма и его текстовый вывод.

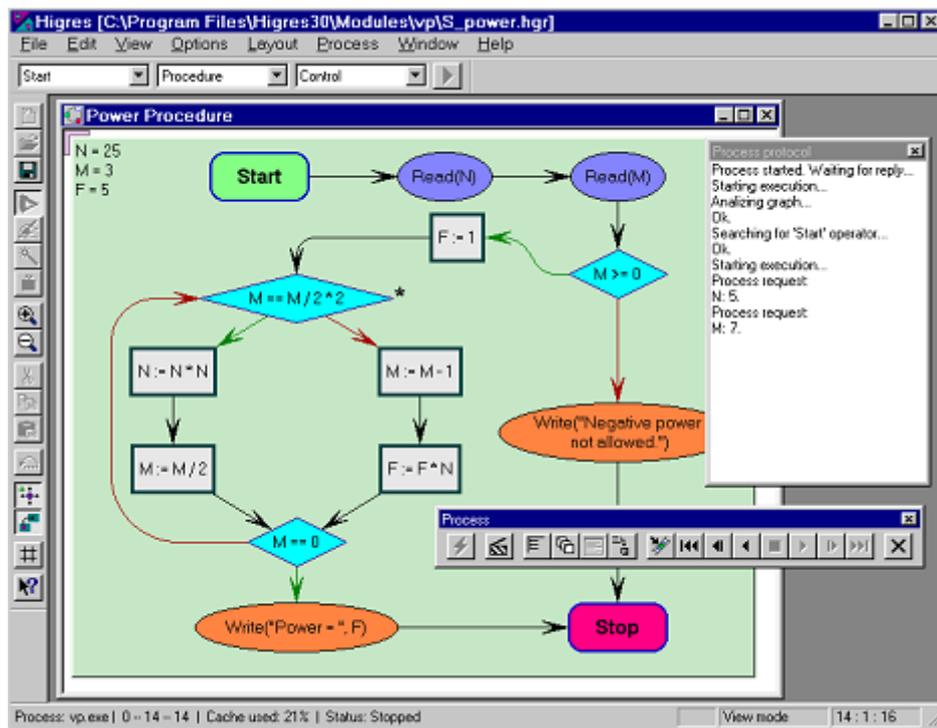


Рисунок 30 – Выполнение алгоритма, реализованного в виде внешнего модуля к системе Higes.

На рис. 30 показан пример исполнения внешнего модуля. В данном случае граф представляет собой схему программы быстрого возведения в степень. К системе прилагается специальная библиотека на языке C++, предназначенная для написания внешних модулей и включающая функции для модификации графа, а также для взаимодействия с системой. Для написания внешних модулей с помощью данной библиотеки необязательно знать детали внутреннего представления графа, что облегчает работу с ней.

14. VisuAlgo

Инструмент позиционируется как инструмент для поддержки студентов в понимании алгоритмов и структур данных [24]. VisuAlgo поддерживает изображение текста алгоритма одновременно с изображением действий, которые оказывает алгоритм на графовую модель. При визуализации используются средства изображения анимаций встроенные в формат SVG [25], что позволяет использовать различные стили для отображения дуг графа, испытывающие те или иные воздействия со стороны инструкций алгоритма. Например, на Рис. 31.

представлен алгоритм обхода графа в глубину, исполняемый на ориентированном графе, вершины которого находятся в прямоугольной сетке. Красным цветом выделены дуги, которые уже были обработаны алгоритмом. Чёрным цветом выделены дуги, ожидающие обработки. Вершины в данном примере отображаются с помощью окружностей с линией чёрного цвета и числовым идентификатором внутри окружности. Чёрным цветом обозначаются вершины и дуги ещё не посещённые в процессе работы алгоритма. Голубым цветом обозначаются вершины уже посещённые в процессе работы алгоритма. Красным цветом обозначены вершины, уже посещённые во время работы алгоритма. Также присутствует эффект, один кадр которого виден на Рис. 31. В процессе работы алгоритма обхода в глубину происходит перенос фокуса активности с вершины на одну из вершин, инцидентных ей. Перенос фокуса активности изображается с помощью плавного изменения толщины и цвета линии, используемой при визуализации дуги, соединяющей эти две вершины. Так на Рис. 31 показан один кадр из процесса, когда чёрная тонкая линия плавно становится оранжевой и толстой.

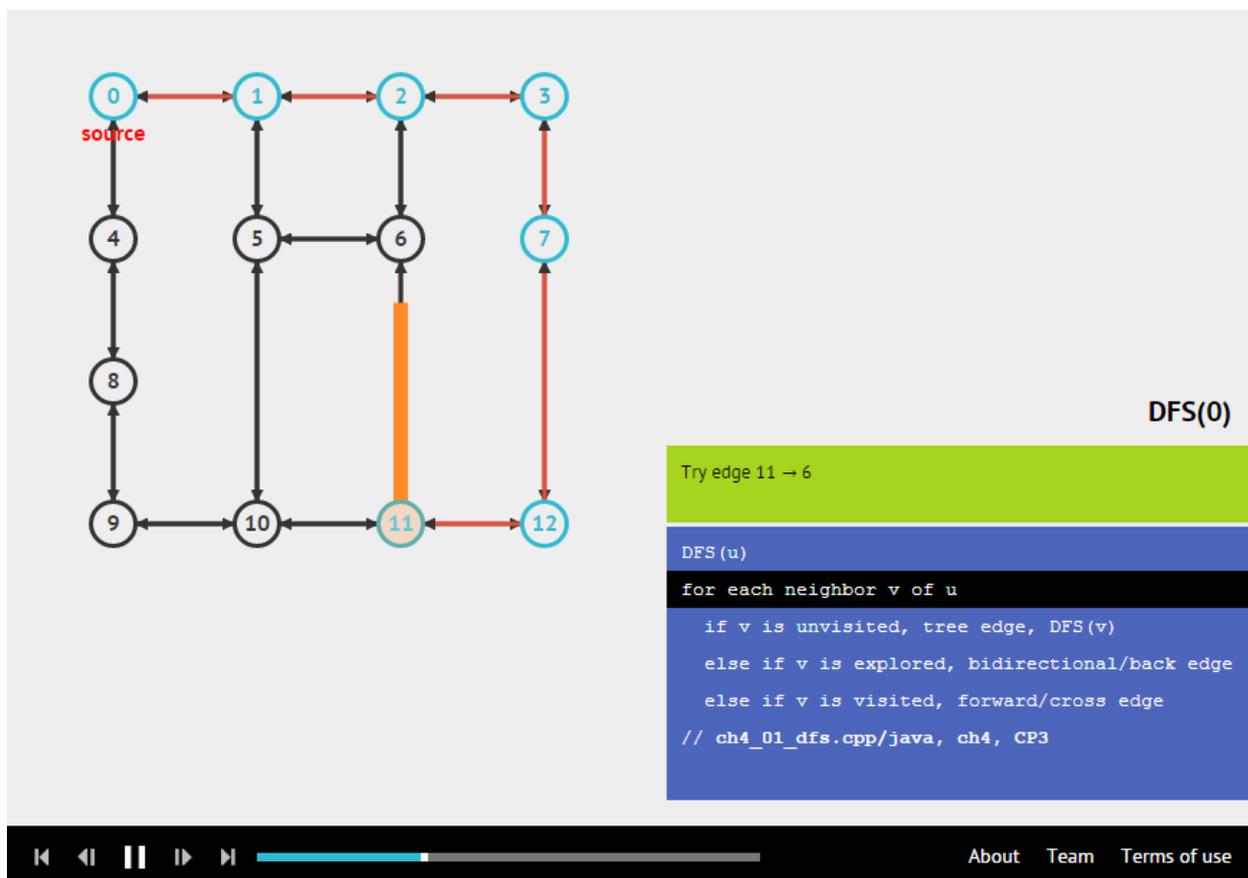


Рисунок 31 – Анимация алгоритма DFS с помощью инструмента VisuAlgo.

После того, как фокус активности перенесён на другую вершину, линия вершины снова становится тонкой, а цвет заменяется на красный. Также на Рис. 31 видно, что при визуализации используется отображение текстового представления алгоритма, чтобы пользователь имел возможность соотнести происходящее с вершинами и дугами графа и текстом алгоритма.

В отображении тексте выделяется активная строка алгоритма, которая приводит к изображаемым эффектам над элементами графа. В данном случае выделение производится с помощью чёрного цвета для фона текущей строки текста. При визуализации имеется возможность прокрутить визуализацию в пошаговом режиме, а также в обратном направлении. Кроме того есть возможность увеличить или уменьшить скорость воспроизведения. Проект разрабатывает при поддержке гранта для поддержки и развития методов обучения. При визуализации вершин и дуг не учитываются точки входа дуг в вершины в том смысле, что нет видимых отдельных

объектов изображения для рисования портов вершин. При визуализации явно используются эффекты, протяжённые во времени. Так как при перематке назад происходит визуализация последнего эффекта до конца. Для визуализации нельзя полностью вручную построить входной граф на момент написания статьи. При визуализации алгоритма обхода в глубину не используется отображение структур данных, стека в данном случае. Как видно из Рис. 31 на изображающем текст алгоритма фрагменте картинки используется рекурсивная версия алгоритма. Однако факт использования рекурсии на изображении не отражается. Если рассмотреть другой пример визуализации с помощью VisuAlgo на Рис. 32, использующий очередь и не использующий рекурсию, то также видно, что дополнительные структуры данных отображаются при визуализации алгоритма. В данном примере со содержимое очереди изображается с помощью цветового выделения вершин, находящихся в очереди. Синим цветом выделены вершины, находящиеся в очереди в данный

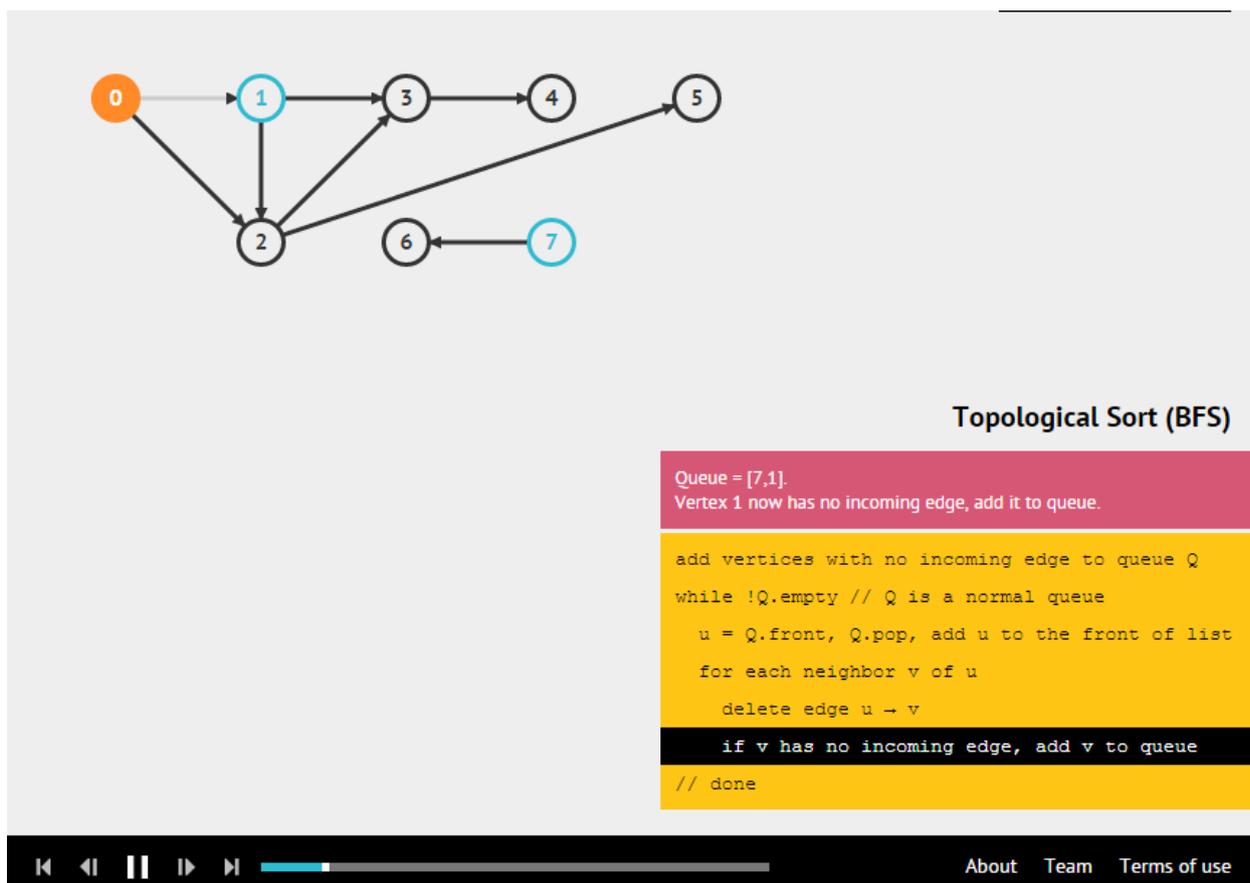


Рисунок 32 – Анимация алгоритма топологической сортировки с помощью инструмента VisuAlgo.

момент. Отображению смены инструкций алгоритма не хватает плавности. Удаляемые вершины помечаются серым цветом.

15. Заключение

В данной обзорной статье были рассмотрены примеры систем визуализации алгоритмов. Приведённые примеры рассматривались с позиции соответствия определённому набору свойств, которым могла бы удовлетворять электронная энциклопедия алгоритмов на графах. К таким свойствам относятся возможность задания графов-параметров пользователем, возможность задания алгоритмов-параметров, а также возможность настройки визуализации алгоритма. Если рассматривать такую энциклопедию алгоритмов как образовательный ресурс, направленный на обучение пользователей программированию на примере алгоритмов на графах, то задание графов-параметров позволяет построить

визуализацию на заданных пользователем данных, а задание алгоритма-параметра позволяет детально разобраться в нюансах самого алгоритма, что является полезным при усвоении нового алгоритма.

И если в существующих системах визуализации алгоритмов часто поддерживается возможность редактирования графов-параметров (хотя чаще используются автоматически сгенерированные графы), то ввод алгоритмов-параметров обычно не реализован никаким образом. Большинство систем визуализации представляют собой каталоги, где для каждого фиксированного алгоритма визуализации строятся для различных графов с помощью некоторых библиотек визуализации. При этом, если потребуется ввести новый алгоритм в такую систему, то будет необходимо разработать всю визуализацию заново.

При визуализации некоторые системы отображают текст алгоритма на

том или ином языке, однако во время работы визуализатора соответствие между строками кода алгоритма и эффектами, оказываемыми ими на графовую модель, не подчеркивается. Это является полезной возможностью. Также полезным свойством можно назвать возможность гладкой анимации, являющейся весьма удобным дополнением к алгоритму визуализации для создания непрерывного изображения на дисплее и для захвата внимания пользователя.

При реализации передачи алгоритма в качестве параметра становится важным вопрос используемого метода запуска визуализации графических эффектов. В случае если алгоритм фиксирован, такие детали можно инкапсулировать внутри модуля, реализующего детали работы для конкретного алгоритма. Однако для случая, когда алгоритм задается параметром в текстовом виде, этот вопрос является важным. Так, например, прямое использование событийно-ориентированного метода запуска графических эффектов может привести к значительному разрастанию текстового представления алгоритма и сделать его сложным для восприятия пользователем. Вместе с тем, событийно-ориентированный подход является более предпочтительным для реализации универсальной системы визуализации алгоритмов, в силу своей абстрактности.

Таким образом, на сегодняшний день для решения задачи визуализации алгоритмов в большинстве случаев создается собственная система визуализации, которая является каталогом реализованных визуализаций для фиксированного набора алгоритмов, часто без возможности добавления нового алгоритма сторонним пользователем в каталог.

Список литературы

1. Касьянов, В. Н., Евстигнеев, В. А. Графы в программировании: обработка, визуализация и применение. – СПб.: БХВ-Петербург, 2003. – 1104 с. – 3000 экз. – ISBN 5-94157-184-4.

2. Касьянов В.Н., Касьянова Е.В. Визуализация информации на основе графовых

моделей // Научная визуализация. - 2014.- Том. 6, N 1. - С. 31 - 50.

3. Система визуализации графов Walrus. – <http://www.caida.org/tools/visualization/walrus/>

4. Lisitsyn I.A., Kasyanov V.N. Higes – Visualization system for clustered graphs and graph algorithms // Proc. of Graph Drawing 99. – Lect. Notes in Comput. Sci. – 1999. – Vol. 1731. – P. 82–89.

5. Система визуализации иерархических Higes. – <http://pcosrv.iis.nsk.su/higes/>

6. Система визуализации графов Graphviz. – <http://www.graphviz.org/>

7. Система визуализации графов aiSee. – <http://www.aisee.com/>

8. Касьянов В.Н., Золотухин Т.А. Visual Graph - система для визуализации сложно структурированной информации большого объема на основе графовых моделей // Научная визуализация. - 2015. - Том. 7, N 4.- С. 44 – 59.

9. Kasyanov V.N., Kasyanova E.V. Graph and cloud-based tools for computer science education // Lecture Notes of Computer Science. - Springer, 2015. - Vol. 9395. - pp. 41-54.

10. 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.

11. Система анимации алгоритмов Polka. – <http://cc.gatech.edu/gvu/softviz/parviz/>.

12. Система анимации алгоритмов Leonardo. – <http://www.dis.uniroma1.it/~demetres/Leonardo/>.

13. Система визуализации алгоритмов JHAVE. – <http://jhave.org/>

14. Система анимации алгоритмов TRAKLA2. – <http://www.cs.hut.fi/Research/TRAKLA2/>

15. Система анимации алгоритмов JAWAA. – <http://www.cs.duke.edu/csed/jawaa2/>

16. Система анимации алгоритмов Animal. – <http://www.algoanim.info/Animal2/>

17. Система анимации алгоритмов EVEGA. – <http://www14.in.tum.de/EVEGA/>

18. Пример визуализации алгоритмов. – <http://www.cs.nyu.edu/algvis/java/Examples.html>

19. Пример визуализации алгоритмов. – <http://corte.si/posts/code/visualisingsorting/>.

20. Пример визуализации алгоритмов. – <http://www.fandm.edu/computer-science/professors-emeriti/anderson/jay-martin-anderson/research-and-development/portal-algorithm-visualization-in-computational-geometry>.

21. Система анимации алгоритмов Jeliot. – <http://cs.joensuu.fi/~jeliot/>.

22. Stallmann M., Cleaveland R., Hebbar P. GDR: A Visualization Tool for Graph Algorithms // In Proc. Computational Support for

Discrete Mathematics, American Mathematical Society, – 1994. – P. 17-28.

23. Kasyanov V.N. Kasyanova E. V. Information visualization based on graph models // Enterprise Information Systems.- 2013.- Vol. 7, N 2.- pp. 187-197.

24. Halim S. VisuAlgo-Visualising Data Structures and Algorithms Throug Animation // OLYMPIADS IN INFORMATICS. - 2015. - С. 243.

25. Scalable Vector Graphics. <https://www.w3.org/TR/SVG11>.