Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

С. Ю. Пименов, Г. А. Корнеев
А. А. Шалыто

Алгоритм Чу Йонджина и Лю Цзенхонга построения кратчайшего корневого дерева в ориентированном графе

Программирование с явным выделением состояний
Проектная документация

Санкт-Петербург
2004

Оглавление

1. Введение **3**

2. Анализ литературы **3**

3. Описание алгоритма **4**

4. Реализация визуализируемого алгоритма **6**

5. Описание модели данных **7**

6. Преобразование программы к виду, удобному для преобразования в автомат **8**

7. Описание интерфейса визуализатора **10**

7.1. Главное окно визуализатора 10
7.2. Редактирование графа 12
7.3. Окно «Сохранить/Загрузить» 13
7.4. Анимация 15

8. Описание конфигурации визуализатора **15**

9. Заключение **18**

Литература **18**

Приложение 1. Исходный код алгоритма на языке Java **19**

Приложение 2. Код алгоритма с использованием модели данных **25**

Приложение 3. Сгенерированные исходные коды автоматов **32**

Приложение 4. Исходные коды интерфейса визуализатора **52**

Приложение 5. Исходный код таймера **65**

Приложение 6. XML-описание визуализатора **67**

CTree.xml (Основные параметры визуализатора) 67
CTree-Algorithm.xml (Описание алгоритма) 68

CTree-Configuration.xml (Конфигурация визуализатора) 79

Аннотация

В проекте разработан визуализатор алгоритма Чу Йонджина и Лю Цзенхонга построения корневого дерева с минимальной суммой весов, содержащихся в нем дуг во взвешенном ориентированном графе. В визуализатор включено несколько примеров графов, которые позволяют понять как основную идею алгоритма, так и некоторые его особенности. Кроме того, имеется возможность редактировать имеющиеся графы и создавать новые, что позволяет проанализировать работу алгоритма в каждом конкретном случае.

1 Введение

Основная идея алгоритма состоит в следующем: сначала он пытается построить искомое дерево непосредственно в исходном графе. В случае, если это удается, алгоритм заканчивает свою работу. Если же на первом шаге построить искомое дерево не удалось, то алгоритм совершает рекурсивный шаг, который в некотором смысле упрощает исходный граф. После этого в упрощенном графе алгоритм ищет минимальное дерево. Затем, используя это дерево, алгоритм строит решение в исходном графе.

2 Анализ литературы

Визуализируемый алгоритм был взят из книги [1]. В этой книге он описан в общих чертах, без указания на конкретную реализацию. Однако, в целом алгоритм изложен понятно. Визуализированный алгоритм немного отличается от описанного в книге. В частности, у И. В. Романовского при построении графа из нулевых дуг, для каждой вершины, кроме корневой, берется лишь одна дуга, входящая в нее, а в корневую вершину будущего дерева не входит ни одна дуга. Далее, если в результате построено дерево из нулевых дуг, то оно и будет решением задачи. В случае если не удалось построить дерево, найдется, по крайней мере, один контур, представляющий собой сильно связанную компоненту в графе из нулевых дуг. Отметим, что при таком построении сильно связанная компонента всегда будет контуром, возможно, из двух вершин.

Реализованный алгоритм, который назовем модифицированным, действует иначе: он сохраняет все дуги нулевого веса. Благодаря этому сильно связанные компоненты могут содержать больше вершин, чем в предыдущем варианте. Следовательно, самих

компонент станет меньше. Это приводит к тому, что алгоритм совершает меньше рекурсивных вызовов и быстрее строит искомое дерево.

При реализации модифицированного алгоритма основную сложность представляет построение сильно связанных компонент. Выбранный в визуализаторе алгоритм их построения описан в разделе «Реализация визуализируемого алгоритма».

3 Описание алгоритма

Задан взвешенный ориентированный граф G и некоторая вершина v. Требуется построить дерево с корнем в вершине v с минимальной суммой весов, содержащихся в нем дуг. Далее такое дерево будем называть минимальным деревом.

Обозначим исходный граф $G_0 = (V_0, E_0)$, в котором $v \in V_0$. Вес дуги, идущей из вершины u в вершину t обозначим $w(ut)$. Сначала убедимся, что любая вершина достижима из корневой вершины v. Если какая-то вершина не может быть достигнута из v, то дерево с корнем в v вообще не может быть построено.

Затем для каждой вершины произведем следующую операцию: «спустим» до нуля веса всех входящих в нее дуг. Определим операцию спуска до нуля на множестве A. Пусть задана функция $f\colon A \to R$. Пусть $m = \inf_{a \in A}\{f(a)\}$. Если множество A конечно, то f достигает на нем минимального значения. Теперь введем новую функцию $g(a) = f(a) - m$. Эта функция неотрицательна на множестве A, а ее минимум (в случае конечного A) равен нулю. После выполнения этой операции в любую вершину, в которую входила хотя бы одна дуга, входит дуга с нулевым весом. Таких дуг может быть и несколько, но, по крайней мере, одна точно найдется. Полученный в результате граф обозначим K.

Теперь надо убедиться, что проведенная операция не изменила решения задачи, то есть минимальное дерево в новом графе K совпадает с минимальным деревом в исходном графе. Введем следующие обозначения:

$$m_u = \min_{v \in V_0}\{w(vu)\}; \tag{1}$$

$$M = \sum_{u \in V_0 \setminus \{v\}} m_u. \tag{2}$$

Теперь рассмотрим произвольное дерево. Для каждой вершины u из множества $V \setminus \{v\}$ в дереве содержится ровно одна дуга, входящая в эту вершину. Следовательно, после изменения весов дуг, суммарный вес всего дерева изменится на M — величину, не зависящую от дерева. Таким образом, минимальное дерево в графе с

новыми весами будет также минимальным деревом в старом графе.

Возникает вопрос: с какой целью произведена операция спуска до нуля? Для того, чтобы это понять, рассмотрим произвольное дерево в новом графе. Если суммарный вес этого дерева равен нулю, то это дерево минимально. В исходном же графе не было такого достаточно простого способа определить, является ли дерево искомым. Отметим, что может так получиться, что не существует дерева из нулевых дуг. В разрешении этой проблемы и заключается основная идея алгоритма.

Рассмотрим граф $H = (V_0, Z_0)$, где Z_0 — дуги нулевого веса, полученные после произведения операции спуска до нуля. Как уже было сказано ранее, если в этом графе найдется дерево с корнем в вершине v, то оно и будет решением. В случае если такого дерева нет, построим новый граф G_1.

Введем несколько определений: сильно связанная компонента графа — это множество вершин A, таких, что для любых двух вершин u и v из множества A в графе существует путь из u в v — из любой вершины сильно связанной компоненты можно добраться в любую другую вершину из той же компоненты. Конденсация графа — это граф, каждая вершина которого представляет собой сильно связанную компоненту исходного графа.

Теперь попытаемся понять, что дают сильно связанные компоненты. Предположим, что сущесвует путь из корневой вершины v в некоторую вершину u из сильно связанной компоненты. Тогда можно добавить пути из u во все вершины компоненты, не изменив общего веса дерева, так как из u можно добраться до любой вершины из компоненты по дугам нулевого веса. После этого требуется построить граф J (конденсацию графа H), определить каким-либо образом веса дуг в J и построить в нем минимальное дерево.

Веса дуг определим следующим образом: пусть ω и ν вершины в J, соответствующие сильно связанным компонентам Ω и N в графе H. Среди всех дуг в графе K, идущих из компоненты Ω в компоненту N, найдем дугу с минимальным весом w, и положим вес дуги, идущей из вершины ω в ν в графе J, равным w. Отметим, что компоненты строятся в графе H, а дуга минимального веса выбирается из K. Построенный граф J содержит меньше вершин, чем исходный граф. Действительно, если в графе J вершин столько же, сколько и в графе H, то в последнем нет ни одной сильно связанной компоненты, состоящей более чем из одной вершины. Следовательно, в графе H нет ни одного контура, а так как в каждую вершину, кроме вершины v, входит, по крайней мере, одна дуга, то в H можно построить дерево с корнем в v, что противоречит сделанному ранее предположению.

Будем считать, что в графе J построено минимальное дерево T. Построим на его основе минимальное дерево в исходном графе. Как уже было отмечено ранее, достаточно построить минимальное дерево в графе K. Сначала возьмем сильно связанную компоненту, содержащую вершину v. Построим в ней дерево с корнем v, состоящее только из нулевых дуг. Теперь рассмотрим две вершины ω и ν в дереве T. Пусть вес дуги в T, идущей из ω в ν, равен w. По построению графа J между соответствующими компонентами Ω и N, в графе K найдется дуга веса w, идущая из некоторой вершины u из Ω в вершину t из N. Дугу ut добавляем к строящемуся дереву и в компоненте N строим дерево с корнем в t. Перебрав все дуги в T, будет построено дерево в K, а, следовательно, и в исходном графе. Задача решена.

4 Реализация визуализируемого алгоритма

В визуализируемом алгоритме основным объектом является граф. Поэтому необходимо решить каким образом его представлять. В визуализаторе принято решение использовать матрицу весов. В этом случае каждая итерация алгоритма требует времени $O(n^2)$, где n — количество вершин в графе, а так как шагов не более n, то время работы алгоритма — $O(n^3)$. В этом случае операция спуска до нуля, рассмотренная в разд. 3, заключается в нахождении в каждом столбце минимального элемента m, и вычитании его из каждого элемента столбца. После проведения этой операции значения m хранятся в массиве размера n.

Проверка связности графа осуществляется с помощью поиска в глубину. Аналогично проверяется возможность построить дерево, состоящее только из нулевых дуг. Для этого процедура поиска в глубину вызывается в графе H (здесь и далее используются обозначения предыдущего пункта), начиная с вершины v.

Следующий шаг — построение конденсации графа. Для этого в визуализаторе используется алгоритм, который с помощью двух поисков в глубину, находит компоненты сильной связности. Далее будет изложен только сам алгоритм без доказательства его корректности. Подробное изложения этого алгоритма и доказательство того, что он находит компоненты сильной связности, можно найти в [2].

При построении сильно связанных компонент для каждой вершины графа хранится номер компоненты, в которую она попадает. Используется также вспомогательный массив из n элементов, значение которого описано далее.

Как отмечалось выше, алгоритм осуществляет два поиска в глубину. На первом из них каждый элемент вспомогательного массива заполняется временем завершения

обработки соответствующей вершины t_n. На втором проходе выполняется построение самих компонент. Для этого исходный граф транспонируется — направление каждой дуги изменяется на противоположное. Затем в транспонированном графе осуществляется поиск в глубину, но при этом вершины перебираются в порядке убывания значений t_n. Деревья, образовавшиеся при втором поиске в глубину, соответствуют компонентам сильной связности.

В визуализаторе минимальное дерево строится всегда с корнем в вершине с номером ноль. Это необходимо учитывать при построении графа J, поскольку нулевая вершина графа H может и не попасть в сильно связанную компоненту с номером ноль. Для этого перед построением J компоненты перенумеровываются. При этом желательно, чтобы при визуализации «конденсации» вершины не слишком далеко смещались со своих исходных мест.

При построении дерева в графе K в каждой компоненте сильной связанности необходимо построить дерево. В визуализаторе это делается следующим образом: сначала для каждой компоненты находится вершина, которая будет корнем поддерева в этой компоненте, а затем, для каждой из таких вершин, вызывается поиск в глубину в графе H. При этом корнем для компоненты с номером ноль всегда будет вершина с таким же номером.

Реализация алгоритма приведена в приложении 1.

5 Описание модели данных

В предыдущем пункте уже упоминалось, что граф является основным объектом в визуализируемом алгоритме. Поэтому именно с него начинается формирование модели данных. Будем хранить количество вершин в графе и матрицу весов. Затем надо иметь матрицу, в которой будет строиться минимальное дерево. Для удобства визуализации добавим еще одну матрицу, где будет храниться граф H. При реализации алгоритма можно было бы обойтись одной матрицей на каждый рекурсивный шаг, но при этом пришлось бы модифицировать алгоритмы поиска в глубину и построения компонент сильной связности. Например, если в этих алгоритмах рассматривать только дуги нулевого веса, то можно было бы обойтись без матрицы, содержащей граф H. К тому же, дерево можно строить в той же матрице, где находился и исходный граф. Данная реализация не претендует на оптимальность ни по скорости, ни по требуемой памяти, так как ее цель — как можно нагляднее показать работу алгоритма, а указанные ранее модификации или сильно усложняют визуализацию, или даже

делают ее невозможной.

Помимо матриц необходимо также хранить и минимумы m_v (разд. 3). Для этого используется массив размера n. Надо также хранить флаги, показывающие достижима ли каждая иершина из v и можно ли построить дерево из дуг нулевого веса.

Удобно использовать массив p, содержащий информацию о последнем поиске в глубину. В нем для каждой вершины будем хранить ее родителя. Этот массив применяется при построении дерева из нулевых дуг в графе H, если это возможно. С помощью него строятся поддеревья в компонентах сильной связности. К тому же он используется для визуализации строящегося дерева.

Для хранения компонент сильной связности также применяется массив размера n, в котором для каждой вершины указан номер компоненты, в которую она попадает. Хранится и количество сильно связанных компонент. Еще один массив — это массив, содержащий корни поддеревьев в каждой компоненте.

Реализация алгоритма с использованием модели данных приведена в приложении 2.

6 Преобразование программы к виду, удобному для преобразования в автомат

Основной задачей при преобразовании алгоритма к виду, удобному для визуализации, является выделение визуализируемых состояний. Именно от их выбора зависит наглядность визуализатора. Самым первым состоянием сделаем инициализацию графа. В принципе, от этого шага можно было бы и отказаться, в том смысле, что не останавливать на нем визуализацию, но он помогает кратко сформулировать задачу перед началом работы визуализатора.

На следующем шаге выясним, каждая ли вершина графа достижима из вершины v. Как отмечалось ранее, если в какую-то вершину u не существует пути из вершины v, то в графе построить минимальное дерево вообще невозможно. На этом шаге исключаются все графы, в которых решения не существует.

Далее перейдем к выполнению алгоритма. Опишем его шаги, используемые при визуализации.

1. На первом шаге выполняется поиск минимумов во всех столбцах матрицы весов. Это и будут минимальные веса дуг, входящих в каждую вершину. Выделение этого шага позволяет увидеть, что веса дуг не просто как-то изменяются при про-

ведении спуска до нуля, а действительно производится вычитание минимальных элементов.

2. Выполним операцию спуска до нуля.
3. После построения графа H, выполняется попытка построения дерева из нулевых дуг. Этот шаг также необходимо визуализировать, так как без него не слишком понятен дальнейший переход либо к окончательному построению минимального дерева, либо к построению компонент сильной связности в графе.
4. Вывод результата попытки построения дерева на предыдущем шаге.
5. Если на предыдущем шаге дерево было построено, то теперь восстанавливаются веса дуг, которые изменились при произведении спуска до нуля. В результате получается минимальное дерево либо в исходном графе, либо в одном из рекурсивных вызовов алгоритма. Это отслеживается с помощью переменной rec в модели данных, и в разных ситуациях выводятся разные комментарии.
6. Если же дерево построить не удалось, то производится поиск компонент сильной связности. Процесс поиска компонент в этом визуализаторе не показывается, так как представляет собой независимый алгоритм. На этом шаге показывается только результат поиска компонент. Различные компоненты раскрашиваются в различные цвета.
7. После построения сильно связанных компонент, требуется построить новый граф. Здесь возможны два варианта визуализации (разд. 7). В любом случае на этом шаге производится «конденсация» компонент сильной связности в вершины нового графа, а также построение новых дуг.
8. После построения нового графа производится рекурсивный шаг алгоритма.
9. После завершения рекурсивного шага вновь возвращаемся к исходному графу. Отображаем дерево T в графе J, в котором вершины, принадлежащие одной сильно связанной компоненте в исходном графе, раскрашены в один цвет.
10. Раскрытие компонент сильной связности. В результате получаем изображение, похожее на то, которое было на шаге построения компонент, с той лишь разницей, что теперь дуги между компонентами взяты из дерева T. На этом же шаге определяются корни поддеревьев в сильно связанных компонентах.
11. Построим поддеревья в сильно связанных компонентах. Начнем с построения поддерева в компоненте, содержащей вершину v. На этом шаге эта компонента подсвечивается цветом, отличным от других.
12. В компоненте, содержащей вершину v, строится поддерево. Эти два шага (этот и предыдущий) пришлось выделить из цикла, описанного далее из-за особо по-

ложения вершины v — она корень строящегося дерева, и в нее не входит ни одна дуга.

13. Выполним перебор компонент в графе H. На каждой итерации производятся два шага:
    - выделение очередной компоненты сильной связности, в которой на следующем шаге будет производиться построение дерева;
    - в выделенной компоненте подсвечиваются те дуги, которые войдут в строящееся минимальное дерево.
14. Этот шаг аналогичен первому шагу предыдущего пункта. Они отличаются тем, что во всех компонентах деревья уже построены, и выделять больше нечего. Таким образом, изображаются все компоненты сильной связности, в каждой из которых выделены дуги, которые войдут в минимальное дерево.
15. Теперь убирем все дуги, не вошедшие в дерево, и восстановим исходные веса дуг. На этом построение дерева в графе завершено. Если это был исходный граф, то завершаем выполнение алгоритма, если же это был рекурсивный шаг, то далее переходим к шагу перед раскрытием компонент. Для различения этих ситуаций, как уже упоминалось, используется переменная rec.

Сгенерированный $Vizi$ код содержит два автомата для движения в «прямом» и «обратном» направлениях по алгоритму. Каждый из них содержит по 28 одноименных состояний. Каждый из них реализуется с помощью двух операторов $switch$, один из которых обеспечивает переход в следующее состояние (предыдущее для автомата, обеспечивающего движение в «обратном» направлении), а другой — действие в текущем состоянии.

Сгенерированные коды автоматов приведены в приложении 3.

7 Описание интерфейса визуализатора

7.1 Главное окно визуализатора

На рис. 1 изображено главное окно визуализатора. Оно позволяет выбирать один из пяти, заложенных в визуализатор примеров. Просматривать шаги визуализатора, и изменять некоторые настройки. Далее подробно описано назначение каждой кнопки.

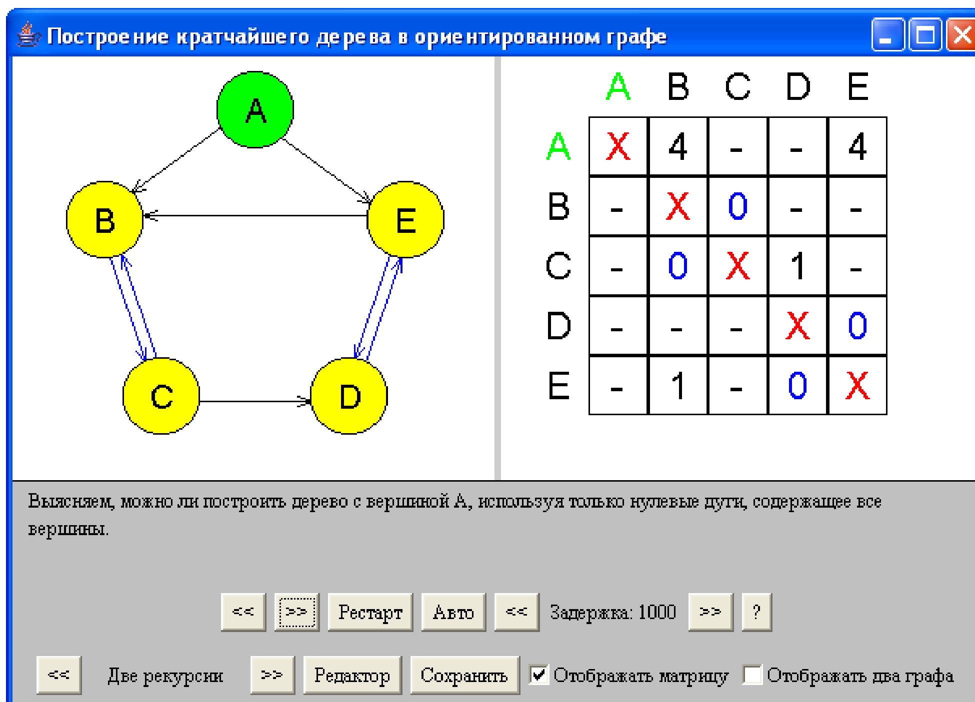

Рис. 1. Главное окно визуализатора

В верхней части окна изображается граф и матрица весов. Между графом и матрицей находится вертикальная разделительная полоса, которую можно сдвинуть либо вправо, либо влево, в зависимости от того, подо что необходимо отвести больше места: под граф или матрицу.

Ниже отображается комментарий к текущему состоянию алгоритма. Еще ниже — два ряда управляющих элементов. В верхнем ряду располагаются стандартные кнопки. В нижнем ряду — кнопки, управляющие специальными настройками для данного визуализатора.

Описание элементов управления верхнего ряда:

- кнопка «<<» обеспечивает переход к предыдущему шагу визуализации;
- кнопка «>>» обеспечивает переход к следующему шагу визуализации;
- кнопка «Рестарт» переводит визуализатор в исходное состояние;
- кнопка «Авто» переводит визуализатор в автоматический режим — шаги осуществляются не по нажатию кнопки «>>», а по прошествии времени, указанного в качестве «Задержки»;
- панель «Задержка» используется для установки времени задержки в миллисекундах. Это время, которое должно пройти перед переходом к следующему шагу в автоматическом режиме;

- кнопка «?» выводит краткую информацию о визуализаторе.

Описание элементов управления нижнего ряда:

- крайняя левая панель используется для выбора текущего примера. Исходно, в визуализатор входит пять примеров, но можно создавать и свои примеры;
- кнопка «Редактор» переводит визуализатор в режим редактирования графа;
- кнопка «Сохранить» открывает окно «Сохранить/Загрузить»;
- с помощью галочки «Отображать матрицу» можно включать и отключать визуализацию матрицы весов;
- галочка «Отображать два графа» управляет режимом построения конденсации графа;

7.2 Редактирование графа

Визуализатор предоставляет удобный интерфейс для редактирования и создания новых графов. Для перехода в режим редактирования надо нажать кнопку «Редактор». Для выхода из режима редактирования — кнопку «Визуализатор» (см. рис. 2).

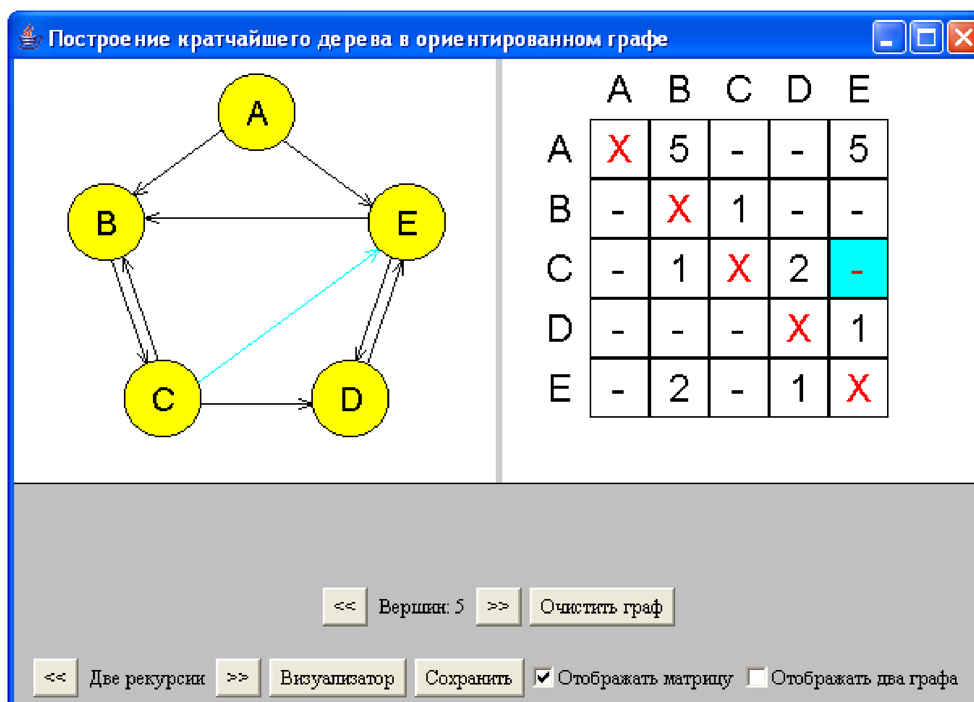

Рис. 2. Режим редактирования графа

Значения всех элементов управления в нижнем ряду совпадают с теми, которые

были указаны в разделе «Главное окно визуализатора». Поэтому опишем только верхний ряд:

- панель слева используется для выбора количества вершин в графе. Их не может быть меньше трех и больше восьми;
- кнопка «Очистить граф» удаляет из текущего графа все дуги, оставляя число вершин неизменным.

Не считая изменения количества вершин в графе, весь процесс редактирования сводится к изменению весов дуг. Опишем, как это делается. Первый способ заключается в том, чтобы «протянуть» дугу от одной вершины к другой. Для этого требуется навести мышь на вершину, откуда должна выходить дуга, нажать левую кнопку мыши и перетащить ее к той вершине, куда дуга должна входить, и после этого отпустить кнопку мыши. При этом редактируемая дуга будет выделена голубым цветом на графе и на матрице весов. Другой способ выбора редактируемой дуги состоит в том, чтобы просто щелкнуть левой кнопкой мыши на соответствующем элементе в матрице весов.

Опишем, как изменит вес дуги. Для этого необходимо набрать новый вес дуги. Он должен быть неотрицательным числом с плавающей точкой (если вес — целое число, то точку писать необязательно). Длина вводимого веса не может превышать трех символов. Если введенный вес не представляет собой число с плавающей точкой, то дуга удаляется из графа. Удаления можно также произвести, нажав на клавиатуре кнопку DEL.

7.3 Окно «Сохранить/Загрузить»

Окно «Сохранить/Загрузить» (рис. 3, 4) позволяет сохранять выбранный пример в файл, загружать пример из файла. Можно также в этом окне редактировать граф. Формат, в котором выводится граф, в окне пояснятся комментариями.

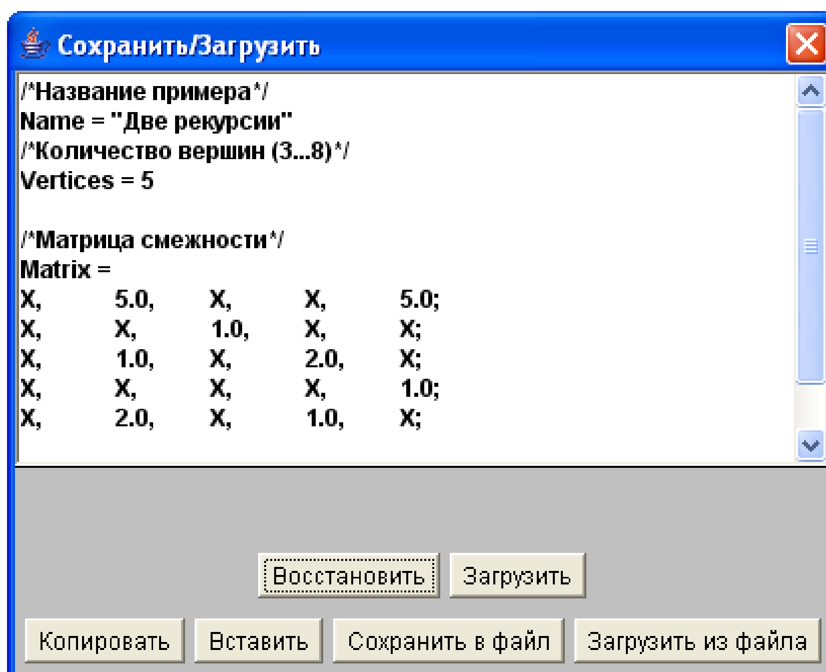

Рис. 3. Окно «Сохранить/Загрузить»

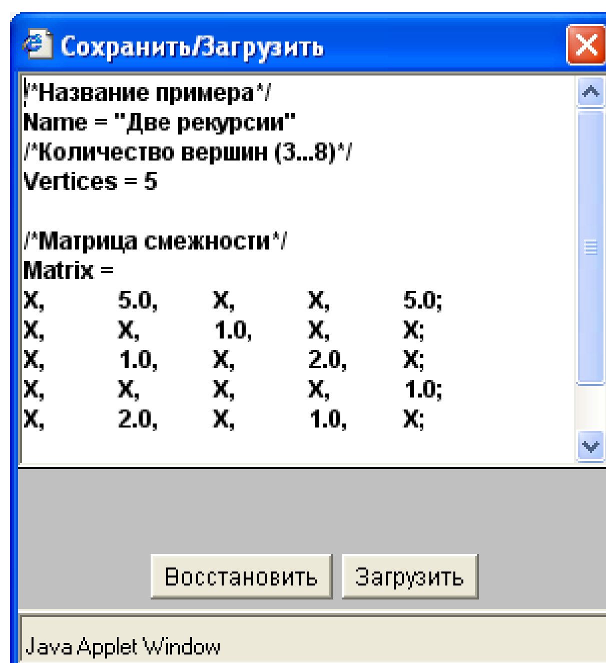

Рис. 4. Окно «Сохранить/Загрузить» (Апплет)

Значения кнопок в этих окнах следующие:

- кнопка «Восстановить» обеспечивает восстанавление исходного текста, описывающего состояние визуализатора;

- кнопка «Загрузить» обеспечивает загрузку примера из окна «Сохранить/Загрузить» и добавляет его к списку доступных примеров. Если в окне указано некорректное описание примера, выводится сообщение об ошибке и указывается ее положение;
- кнопка «Копировать» обеспечивает копирование содержимого окна в буфер обмена;
- кнопка «Вставить» обеспечивает вставку содержимого буфера обмена;
- кнопка «Сохранить в файл» позволяет сохранить содержимое окна в файл;
- кнопка «Загрузить из файла» позволяет загрузить содержимое окно из файла.

Последние две кнопки обеспечивают возможность сохранения примеров (в том числе, новых).

Исходный код интерфейса визуализатора приведен в приложении 4.

7.4 Анимация

Как уже упоминалось, с помощью галочки «Отображать два графа» можно переключать режимы визуализации процессов конденсации графа и раскрытия сильно связанных компонент. В случае, если эта галочка поставлена, при конденсации на экране отображается граф, который был до произведения конденсации, и граф, который получился в результате. Если эта галочка не стоит, то процесс конденсации происходит с анимацией.

Анимация производится с помощью таймера (Приложение 5). Исходное положение каждой вершины соединяется отрезком с ее положением после конденсации. Этот отрезок разбивается на $STEPS$ частей. Задержка таймера ставится равной длительности шага в автоматическом режиме, деленной на $STEPS + 1$. По истечении задержки таймер вызывает метод $tick$, который смещает вершины. После того, как таймер произвел $STEPS$ шагов, все вершины оказались в своих новых положениях и анимация завершается.

8 Описание конфигурации визуализатора

Конфигурацию визуализатора можно разделить на несколько логических блоков. Первый состоит из параметров, описывающих стандартный интерфейс визуализатора. Второй описывает остальной интерфейс, а третий — внутренние параметры ви-

зуализатора. Первый блок здесь не описывается, так как он автоматически создается пакетом $Vizi$. Поэтому перейдет ко второму блоку:

- *comment-height:* описывает высоту панели, отведенной для комментария. Этот параметр является стандартным, но лучше его явно указать, так как в разных визуализаторах комментарии могут быть разной длины;
- *SaveLoadDialog:* группа параметров, описывающая окно «Сохранить/Загрузить»:
  - *CommentPane-lines:* количество строк в панели для комментария;
  - *columns:* количество колонок текстовой области;
  - *rows:* количество строк текстовой области.
- *vertices:* описывает панель для изменения количества вершин в графе;
- *clear-button:* описывает кнопку «Очистить граф», удаляющую все дуги из графа;
- *edit-button:* описывает кнопку «Редактор», переводящую визуализатор в режим редактирования графа;
- *start-button:* описывает кнопку «Визуализатор»;
- *save-button:* описывает кнопку «Сохранить», которая открывает окно «Сохранить/Загрузить»;
- *examples-button-less:* описывает кнопку, загружающую предыдущий пример в списке доступных примеров;
- *examples-button-more:* описывает кнопку, загружающую следующий пример в списке доступных примеров;
- *examples-hint:* описывает всплывающую подсказку для панели выбора примера. Последние три параметра, начинающиеся со слова *example*, описывают панель для выбора вершин. Эта панель устроена аналогично панели для выбора задержки или количества вершин, с той разницей, что в этом случае выбирается не число, а строка. Однообразие в описании конфигурации требует изменения структуры xml-описания конфигурации, что, в свою очередь, нарушит единообразие среди визуализаторов. Несмотря на это, логическую структуру можно выдержать либо внося параметры в одну группу, либо добавляя к ним одинаковый префикс. По сути, эти два способа эквивалентны;
- *show-check-caption:* описание галочки «Отображать матрицу»;
- *show-check-hint:* описание всплывающей подсказки к галочке «Отображать матрицу». Здесь ситуация аналогична той, которая была в пункте *examples-hint*;
- *vizi-mode-caption:* описание галочки «Отображать два графа»;

- *vizi-mode-hint:* описание всплывающей подсказки к галочке «Отображать два графа»;
- *showcheck:* описывает начальное состояние галочки «Отображать матрицу»;
- *modecheck:* описывает начальное состояние галочки «Отображать два графа».

Перейдем к описанию набору стилей *table*. Он содержит описание всех возможных вариантов раскраски элементов в матрице весов:

- *style0:* стандартное состояние элемента — черные буквы на прозрачном фоне;
- *style1:* нулевые элементы матрицы послу проведения операции спуска до нуля;
- *style2:* элементы матрицы, соответствующие дугам, вошедшим в минимальное дерево;
- *style3:* элементы матрицы, соответствующие дугам, соединяющим разные компоненты сильной связности, после их раскрытия;
- *style4:* редактируемый элемент матрицы;
- *style5 — style11:* семь цветов для семи компонент сильной связности (максимально возможное число вершин — восемь, компонент хотя бы на одну меньше);
- *style12:* диагональные элементы матрицы;

Приведем стили для букв, озаглавливающих строки и столбцы матрицы:

- *style13:* стандартный цвет, соответствует *style0*;
- *style14:* вершины, вошедшие в построенное дерево, позволяет сразу увидеть, является ли оно решением — содержит ли все вершины;
- *style15:* не используется;
- *style16 — style22:* семь цветов для семи компонент. Соответствуют стилям *style5 — style11*;
- *style23:* недостижимая вершина;
- *style24:* найденные минимальные элементы в каждом столбце;
- *style25:* текущая компонента сильной связности.

Набор стилей *vertex* описывает цвета вершин в графе. Они соответствуют стилям *style13 — style25* в наборе *table*.

- *style0:* желтая вершина, соответствует стилю *table-style13*;
- *style1:* вершина, вошедшая в построенное дерево, соответствует стилю *table-style14*;
- *style2:* не используется;

- *style3 — style9:* семь цветов для семи компонент (*table-style16 — table-style22*);
- *style10:* недостижимая вершина (*table-style23*);
- *style11:* вершина в текущей компоненте (*table-style25*).

Набор стилей *arrow* описывает цвета дуг в графе. Эти стили соответствуют стилям *style0 — style11* в наборе *table*.

- *style0:* черная дуга.
- *style1:* дуга нулевого веса, после проведения операции спуска до нуля.
- *style2:* дуги, вошедшие в минимальное дерево.
- *style3:* дуги, соединяющие разные компоненты сильной связности, после их раскрытия.
- *style4:* редактируемая дуга.
- *style5 — style11:* семь цветов для семи компонент сильной связности.

XML-описание конфигурации визуализатора приведено в приложении 6.

9 Заключение

Применение технологии $Vizi$ упростило построение вызуализатора по сравнению с традиционным подходом.

Список литературы

[1] Романовский И.В. Дискретный анализ. СПб.: Невский диалект, 2000, с. 160–163.

[2] Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. М.: МЦНМО, 2002, с. 456–461.

[3] Chu Y.J., Liu T.H. On the shortest arborescence of a direct graph //Sci. Sinica. 1965, **14**, pp. 1396–1400.

Приложение 1. Исходный код алгоритма на языке Java

```
import java.io.*;

public class CTree
{
    static private void print(double[][] m)
    {
        System.out.println("--");
        int n = m.length;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                System.out.print(m[i][j] + "\t");
            }
            System.out.print("\n");
        }
    }

    static private void print(int[] m)
    {
        System.out.println("--");
        int n = m.length;
        for (int i = 0; i < n; i++)
        {
            System.out.print(m[i] + "\t");
        }
        System.out.println();
    }

    static private double[][] read(String name) throws IOException
    {
        FileReader fr;
        try
        {
            fr = new FileReader(name);
        }
        catch (FileNotFoundException e)
        {
            return null;
        }
        StreamTokenizer st = new StreamTokenizer(fr);
        int n = 0;
        if (st.nextToken() == StreamTokenizer.TT_NUMBER)
        {
            n = (int) st.nval;
        }
        else return null;
        double[][] m = new double[n][n];
        for (int i = 0; i < n*n; i++)
        {
            if (st.nextToken() == StreamTokenizer.TT_NUMBER)
            {
                m[i / n][i % n] = st.nval;
            }
            else if (st.ttype == '-')
            {
                m[i / n][i % n] = Double.NaN;
            }
            else return null;
```

```
    }

    return m;
}

static public void main(String[] argv)
{
    String fname;
    double[][] m = null;
    if (argv.length > 0)
    {
        try
        {
            m = read(argv[0]);
        }
        catch (IOException e)
        {
            m = null;
        }
    }
    if (m == null)
    {
        System.out.println("No input file is given");
        return;
    }
    int n = m.length;
    System.out.println(n);
    print(m);
    double[][] t = new double[n][n];
    if (alg(m, t)) print(t);
}

static private int dfs(double[][] m, int v,
                       boolean[] visit, int[] fin, int[] scc, int ncc,
                       int[] par, int[] vind, int nvind, int time)
{
    int n = m.length;
    int t = time;
    visit[v] = true;
    if (scc != null) scc[v] = ncc;
    for (int ii = 0; ii < n; ii++)
    {
        int i = ii;
        if ((vind != null) && (nvind < 0)) i = vind[ii];
        if (visit[i]) continue;
        if (!Double.isNaN(m[v][i]))
        {
            if (par != null) par[i] = v;
            int t0 = dfs(m, i, visit, fin, scc, ncc, par, vind,
             nvind, t + 1);
            if (nvind >= 0) nvind += (t0 - t + 1) / 2;
            t = t0;
        }
    }
    t++;
    if (fin != null) fin[v] = t;
    if ((vind != null) && (nvind >= 0)) vind[n - nvind] = v;
    return t;
}

static private int scc(double[][] m, int[] scc)
{
```

```
    int n = m.length;
    double[][] mT = new double[n][n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            mT[i][j] = m[j][i];
        }
    }

    int[] fin = new int[n];
    boolean[] visit = new boolean[n];
    int[] vind = new int[n];
    int nvind = 1;
    int time = 0;
    for (int i = 0; i < n; i++)
    {
        if (!visit[i])
        {
            int t = dfs(m, i, visit, fin, null, 0, null, vind,
            nvind, time);
            if (nvind >= 0) nvind += (t - time + 1) / 2;
            time = t + 1;
        }
    }

    int ncc = 0;
    for (int i = 0; i < n; i++) visit[i] = false;
    for (int i = 0; i < n; i++)
    {
        int v = vind[i];
        if (!visit[v])
        {
            dfs(mT, v, visit, null, scc, ncc, null, vind, -1, 0);
            ncc++;
        }
    }
    return ncc;
}

static private boolean alg(double[][] m, double[][] t)
{
    int n = m.length;
    boolean[] visit = new boolean[n];
    dfs(m, 0, visit, null, null, 0, null, null, -1, 0);
    boolean r = true;
    for (int i = 0; i < n; i++) r &= visit[i];
    if (!r)
    {
        System.out.println("No tree");
        return false;
    }
    algrec(m, t);
    return true;
}

static private void algrec(double[][] m, double[][] t)
{
    int n = m.length;
    double[] mins = new double[n];
    for (int j = 0; j < n; j++)
    {
```

```
        mins[j] = Double.POSITIVE_INFINITY;
        for (int i = 0; i < n; i++)
        {
            if (m[i][j] < mins[j]) mins[j] = m[i][j];
        }
        for (int i = 0; i < n; i++) m[i][j] -= mins[j];
    }

    boolean[] visit = new boolean[n];
    double[][] mZ = new double[n][n];

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (m[i][j] == 0) mZ[i][j] = 0;
            else mZ[i][j] = Double.NaN;
        }
    }

    int[] par = new int[n];
    dfs(mZ, 0, visit, null, null, 0, par, null, -1, 0);
    boolean r = true;
    for (int i = 0; i < n; i++) r &= visit[i];
    if (r)
    {
        restore(m, mins);
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (par[j] == i) t[i][j] = m[i][j];
                else t[i][j] = Double.NaN;
            }
        }
        return;
    }

    int[] scc = new int[n];
    int ncc = scc(mZ, scc);
    renum(scc);

    double[][] gr = new double[ncc][ncc];
    for (int i = 0; i < ncc; i++)
    {
        for (int j = 0; j < ncc; j++)
        {
            gr[i][j] = Double.POSITIVE_INFINITY;
        }
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (scc[i] == scc[j]) gr[scc[i]][scc[j]] = Double.NaN;
            else if (m[i][j] < gr[scc[i]][scc[j]])
            {
                gr[scc[i]][scc[j]] = m[i][j];
            }
        }
    }
```

```java
for (int i = 0; i < ncc; i++)
{
    for (int j = 0; j < ncc; j++)
    {
        if (Double.isInfinite(gr[i][j]))
        {
            gr[i][j] = Double.NaN;
        }
    }
}

double[][] grt = new double[ncc][ncc];

algrec(gr, grt);

for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        if (scc[i] != scc[j])
        {
            mZ[i][j] = Double.NaN;
        }
    }
}
for (int i = 0; i < n; i++) par[i] = 0;
boolean[] grv = new boolean[ncc];
int[] crts = new int[ncc];
crts[0] = 0;
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        if (!grv[scc[j]] && (scc[i] != scc[j]) &&
            (grt[scc[i]][scc[j]] == m[i][j]))
        {
            t[i][j] = m[i][j];
            grv[scc[j]] = true;
            crts[scc[j]] = j;
        }
        else t[i][j] = Double.NaN;
    }
}

for (int i = 0; i < n; i++) visit[i] = false;
for (int i = 0; i < ncc; i++) grv[i] = false;
for (int i = 0; i < ncc; i++)
{
    dfs(mZ, crts[i], visit, null, null, 0, par, null, -1, 0);
}

for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        if ((scc[i] == scc[j]) && (i == par[j]))
        {
            t[i][j] = m[i][j];
        }
    }
}
```

```
        restore(t, mins);
        restore(m, mins);
    }

    static private void restore(double[][] m, double[] mins)
    {
        int n = m.length;
        for (int j = 0; j < n; j++)
        {
            for (int i = 0; i < n; i++)
            {
                m[i][j] += mins[j];
            }
        }
    }

    static private void renum(int[] scc)
    {
        int n = scc.length;
        int v = scc[0];
        for (int i = 0; i < n; i++)
        {
            if (scc[i] == 0) scc[i] = v;
            else if (scc[i] == v) scc[i] = 0;
        }
    }
}
```

Приложение 2. Код алгоритма с использованием модели данных

```
import java.io.*;
import java.util.*;

class Data
{
    public int n;
    public double[][] graph;
    public double[][] zeros;
    public double[][] tree;
    public double[] mins;
    public int[] par;
    public boolean connect;
    public boolean isTree;
    public int[] scc;
    public int ncc;
    public int comp;
    public int[] crts;
}

public class CTree2
{
    static private final Data d = new Data();
    static private final Stack stack = new Stack();

    static private void print(double[][] m)
    {
        System.out.println("--");
        int n = m.length;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                System.out.print(m[i][j] + "\t");
            }
            System.out.print("\n");
        }
    }

    static private void print(int[] m)
    {
        System.out.println("--");
        int n = m.length;
        for (int i = 0; i < n; i++)
        {
            System.out.print(m[i] + "\t");
        }
        System.out.println();
    }

    static private double[][] read(String name) throws IOException
    {
        FileReader fr;
        try
        {
            fr = new FileReader(name);
        }
        catch (FileNotFoundException e)
        {
            return null;
```

```
    }
    StreamTokenizer st = new StreamTokenizer(fr);
    int n = 0;
    if (st.nextToken() == StreamTokenizer.TT_NUMBER)
    {
        n = (int) st.nval;
    }
    else return null;
    double[][] m = new double[n][n];
    for (int i = 0; i < n*n; i++)
    {
        if (st.nextToken() == StreamTokenizer.TT_NUMBER)
        {
            m[i / n][i % n] = st.nval;
        }
        else if (st.ttype == '-')
        {
            m[i / n][i % n] = Double.NaN;
        }
        else return null;
    }

    return m;
}

static public void main(String[] argv)
{
    String fname;
    double[][] m = null;
    if (argv.length > 0)
    {
        try
        {
            m = read(argv[0]);
        }
        catch (IOException e)
        {
            m = null;
        }
    }
    if (m == null)
    {
        System.out.println("No input file is given");
        return;
    }
    int n = m.length;
    System.out.println(n);
    print(m);
    double[][] t = new double[n][n];
    if (alg(m, t)) print(t);
}

static private int dfs(double[][] m, int v,
                       boolean[] visit, int[] fin, int[] scc, int ncc,
                       int[] par, int[] vind, int nvind, int time)
{
    int n = m.length;
    int t = time;
    visit[v] = true;
    if (scc != null) scc[v] = ncc;
    for (int ii = 0; ii < n; ii++)
    {
```

```
            int i = ii;
            if ((vind != null) && (nvind < 0)) i = vind[ii];
            if (visit[i]) continue;
            if (!Double.isNaN(m[v][i]))
            {
                if (par != null) par[i] = v;
                int t0 = dfs(m, i, visit, fin, scc, ncc, par, vind,
                 nvind, t + 1);
                if (nvind >= 0) nvind += (t0 - t + 1) / 2;
                t = t0;
            }
        }
        t++;
        if (fin != null) fin[v] = t;
        if ((vind != null) && (nvind >= 0)) vind[n - nvind] = v;
        return t;
    }

    static private int scc(double[][] m, int[] scc)
    {
        int n = m.length;
        double[][] mT = new double[n][n];
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                mT[i][j] = m[j][i];
            }
        }

        int[] fin = new int[n];
        boolean[] visit = new boolean[n];
        int[] vind = new int[n];
        int nvind = 1;
        int time = 0;
        for (int i = 0; i < n; i++)
        {
            if (!visit[i])
            {
                int t = dfs(m, i, visit, fin, null, 0, null, vind,
                nvind, time);
                if (nvind >= 0) nvind += (t - time + 1) / 2;
                time = t + 1;
            }
        }

        int ncc = 0;
        for (int i = 0; i < n; i++) visit[i] = false;
        for (int i = 0; i < n; i++)
        {
            int v = vind[i];
            if (!visit[v])
            {
                dfs(mT, v, visit, null, scc, ncc, null, vind, -1, 0);
                ncc++;
            }
        }
        return ncc;
    }

    static private boolean alg(double[][] m, double[][] t)
    {
```

```
d.n = m.length;
d.graph = m;
d.zeros = new double[d.n][d.n];
d.tree = t;
d.mins = new double[d.n];
d.par = new int[d.n];
d.scc = new int[d.n];

// - State: Initialization

boolean[] visit = new boolean[d.n];
dfs(d.graph, 0, visit, null, null, 0, null, null, -1, 0);
d.connect = true;
for (int i = 0; i < d.n; i++) d.connect &= visit[i];
if (!d.connect)
{
    // State: error
    System.out.println("No tree");
    return false;
}

for (int j = 0; j < d.n; j++)
{
    d.mins[j] = Double.POSITIVE_INFINITY;
    for (int i = 0; i < d.n; i++)
    {
        if (d.graph[i][j] < d.mins[j]) d.mins[j] = d.graph[i][j];
    }
}

// State: prezero

for (int j = 0; j < d.n; j++)
{
    for (int i = 0; i < d.n; i++) d.graph[i][j] -= d.mins[j];
}

// State: zero

for (int i = 0; i < d.n; i++) visit[i] = false;

for (int i = 0; i < d.n; i++)
{
    for (int j = 0; j < d.n; j++)
    {
        if (d.graph[i][j] == 0) d.zeros[i][j] = 0;
        else d.zeros[i][j] = Double.NaN;
    }
}

dfs(d.zeros, 0, visit, null, null, 0, d.par, null, -1, 0);
d.isTree = true;
for (int i = 0; i < d.n; i++) d.isTree &= visit[i];

// State: tryTree

if (d.isTree)
{
    // State: makeTree
    restore(d.graph, d.mins);
    for (int i = 0; i < d.n; i++)
    {
```

```
        for (int j = 0; j < d.n; j++)
        {
            if (d.par[j] == i) d.tree[i][j] = d.graph[i][j];
            else d.tree[i][j] = Double.NaN;
        }
    }
    return true;
}

d.ncc = scc(d.zeros, d.scc);
renum(d.scc);

// State: SCC

double[][] gr = new double[d.ncc][d.ncc];
for (int i = 0; i < d.ncc; i++)
{
    for (int j = 0; j < d.ncc; j++)
    {
        gr[i][j] = Double.POSITIVE_INFINITY;
    }
}

for (int i = 0; i < d.n; i++)
{
    for (int j = 0; j < d.n; j++)
    {
        if (d.scc[i] == d.scc[j])
        {
        gr[d.scc[i]][d.scc[j]] = Double.NaN;
    }
        else if (d.graph[i][j] < gr[d.scc[i]][d.scc[j]])
        {
            gr[d.scc[i]][d.scc[j]] = d.graph[i][j];
        }
    }
}

for (int i = 0; i < d.ncc; i++)
{
    for (int j = 0; j < d.ncc; j++)
    {
        if (Double.isInfinite(gr[i][j]))
        {
            gr[i][j] = Double.NaN;
        }
    }
}

double[][] grt = new double[d.ncc][d.ncc];

// State: newGraph

stack.push(d.graph);
stack.push(d.zeros);
stack.push(d.tree);
stack.push(d.mins);
stack.push(d.scc);
stack.push(d.crts);
stack.push(d.par);

alg(gr, grt);
```

```
d.par = (int[]) stack.pop();
d.crts = (int[]) stack.pop();
d.scc = (int[]) stack.pop();
d.mins = (double[]) stack.pop();
d.tree = (double[][]) stack.pop();
d.zeros = (double[][]) stack.pop();
d.graph = (double[][]) stack.pop();
d.ncc = d.scc.length;
d.n = d.graph.length;

for (int i = 0; i < d.n; i++)
{
    for (int j = 0; j < d.n; j++)
    {
        if (d.scc[i] != d.scc[j])
        {
            d.zeros[i][j] = Double.NaN;
        }
    }
}
for (int i = 0; i < d.n; i++) d.par[i] = 0;
d.crts = new int[d.ncc];
boolean[] grv = new boolean[d.ncc];
d.crts[0] = 0;
for (int i = 0; i < d.n; i++)
{
    for (int j = 0; j < d.n; j++)
    {
        if (!grv[d.scc[j]] && (d.scc[i] != d.scc[j]) &&
            (grt[d.scc[i]][d.scc[j]] == d.graph[i][j]))
        {
            d.tree[i][j] = d.graph[i][j];
            grv[d.scc[j]] = true;
            d.crts[d.scc[j]] = j;
        }
        else d.tree[i][j] = Double.NaN;
    }
}

// State: backstep

for (int i = 0; i < d.n; i++) visit[i] = false;
for (int i = 0; i < d.ncc; i++) grv[i] = false;
d.comp = 0;
while (d.comp < d.ncc)
{
    // State: whileTrue
    dfs(d.zeros, d.crts[d.comp], visit, null, null, 0, d.par,
    null, -1, 0);
    d.comp++;
}

//State: whileFalse

for (int i = 0; i < d.n; i++)
{
    for (int j = 0; j < d.n; j++)
    {
        if ((d.scc[i] == d.scc[j]) && (i == d.par[j]))
        {
            d.tree[i][j] = d.graph[i][j];
```

```
                }
            }
        }

        // State: buildTree

        restore(d.tree, d.mins);
        restore(d.graph, d.mins);

        return true;
    }

    static private void restore(double[][] m, double[] mins)
    {
        int n = m.length;
        for (int j = 0; j < n; j++)
        {
            for (int i = 0; i < n; i++)
            {
                m[i][j] += mins[j];
            }
        }
    }

    static private void renum(int[] scc)
    {
        int n = scc.length;
        int v = scc[0];
        for (int i = 0; i < n; i++)
        {
            if (scc[i] == 0) scc[i] = v;
            else if (scc[i] == v) scc[i] = 0;
        }
    }
}
```

Приложение 3. Сгенерированные исходные коды автоматов

```
package ru.ifmo.vizi.ctree;

import ru.ifmo.vizi.base.auto.*;
import java.util.Locale;

public final class CTree extends BaseAutomataWithListener {
    /**
      * Модель.
      */
    public final Data d = new Data();

    /**
      * Конструктор для языка
      */
    public CTree(Locale locale) {
        super("ru.ifmo.vizi.ctree.Comments", locale);
        init(new Main(), d);
    }

    /**
      * Данные.
      */
    public final class Data {
        /**
          * Number of recursions.
          */
        public int rec;

        /**
          * Number of nodes.
          */
        public int n;

        /**
          * Matrix of weights.
          */
        public double[][] graph = null;

        /**
          * Matrix of zero edges.
          */
        public double[][] zeros = null;

        /**
          * Built tree.
          */
        public double[][] tree = null;

        /**
          * Array of minimums.
          */
        public double[] mins = null;

        /**
          * Array of parents in dfs tree.
          */
        public int[] par = null;

        /**
          * Is graph connected.
```

```
     */
    public boolean connect;

    /**
     * Is graph a tree.
     */
    public boolean isTree;

    /**
     * Strongly connected components.
     */
    public int[] scc = null;

    /**
     * Number of strongly connected components.
     */
    public int ncc;

    /**
     * Current opening component.
     */
    public int comp;

    /**
     * Hilighted component.
     */
    public int vcomp;

    /**
     * Array of the components' roots.
     */
    public int[] crts;

    /**
     * Stack of graphs.
     */
    public AutoStack grStack = new AutoStack();

    /**
     * Functions for CTree.
     */
    public CTreeFuns funs;

    /**
     * Instance of visualizer.
     */
    public CTreeVisualizer visualizer;

    public String toString() {
        StringBuffer s = new StringBuffer();
        s.append("n = " + n);

        return s.toString();
    }
}

/**
 * Builds a shortest tree of paths.
 */
private final class Main implements Automata {
    /**
     * Начальное состояние автомата.
```

```java
  */
private final int START_STATE = 0;

/**
  * Конечное состояние автомата.
  */
private final int END_STATE = 27;

/**
  * Описания состояний.
  */
private final String[] descriptions = new String[]{
    "Начальное состояние",
    "Initialization",
    "Test if there is path to every vertice from A",
    "Test if there is path to every vertice from A",
    "Test if there is path to every vertice from A (окончание)",
    "Finding minimums",
    "Descend to zero",
    "Try to build tree",
    "If tree is build",
    "If tree is build (окончание)",
    "Build graph of strongly connected components",
    "Creating of new graph",
    "Step before recursion",
    "Builds a shortest tree of paths (автомат)",
    "Pre Backstep",
    "Backstep",
    "Build tree in first component",
    "Tree in the first component is built",
    "Build trees in opened components",
    "Next component",
    "Build trees in one component",
    "No more components",
    "Build tree in the hole graph",
    "Build solution tree",
    "test if it is first graph.",
    "test if it is first graph. (окончание)",
    "Cannot find solution",
    "Конечное состояние"};

/**
  * Текущее состояние автомата.
  */
private int state;

/**
  * Текущий вложенный автомат.
  */
private Automata child;

/**
  * Переход в начальнное состояние.
  */
public void toStart() {
    state = START_STATE;
    child = null;
}

/**
  * Переход в конечное состояние.
  */
```

```
public void toEnd() {
    state = END_STATE;
    child = null;
}

/**
  * Находится ли автомат в начальном состоянии.
  */
public boolean isAtStart() {
    return state == START_STATE;
}

/**
  * Находится ли автомат в конечном состоянии.
  */
public boolean isAtEnd() {
    return state == END_STATE;
}

/**
  * Номер текущего шага.
  */
public int getStep() {
    return step;
}

/**
  * Сделать шаг в перед.
  */
public void stepForward(int level) {
    do {
        step++;
        // Переход в следующее состояние
        switch (state) {
            case START_STATE: { // Начальное состояние
                state = 1; // Initialization
                break;
            }
            case 1: { // Initialization
                state = 2; // Test if there is path to every vertice from A
                break;
            }
            case 2: { // Test if there is path to every vertice from A
                state = 3; // Test if there is path to every vertice from A
                break;
            }
            case 3: { // Test if there is path to every vertice from A
                if (d.connect) {
                    state = 5; // Finding minimums
                } else {
                    state = 26; // Cannot find solution
                }
                break;
            }
            case 4: { // Test if there is path to every vertice from A (окончание)
                state = END_STATE;
                break;
            }
            case 5: { // Finding minimums
                state = 6; // Descend to zero
                break;
            }
```

```
case 6: { // Descend to zero
    state = 7; // Try to build tree
    break;
}
case 7: { // Try to build tree
    state = 8; // If tree is build
    break;
}
case 8: { // If tree is build
    if (!d.isTree) {
        state = 10; // Build graph of strongly connected components
    } else {
        state = 23; // Build solution tree
    }
    break;
}
case 9: { // If tree is build (окончание)
    state = 24; // test if it is first graph.
    break;
}
case 10: { // Build graph of strongly connected components
    state = 11; // Creating of new graph
    break;
}
case 11: { // Creating of new graph
    state = 12; // Step before recursion
    break;
}
case 12: { // Step before recursion
    state = 13; // Builds a shortest tree of paths (автомат)
    break;
}
case 13: { // Builds a shortest tree of paths (автомат)
    if (child.isAtEnd()) {
        child = null;
        state = 14; // Pre Backstep
    }
    break;
}
case 14: { // Pre Backstep
    state = 15; // Backstep
    break;
}
case 15: { // Backstep
    state = 16; // Build tree in first component
    break;
}
case 16: { // Build tree in first component
    state = 17; // Tree in the first component is built
    break;
}
case 17: { // Tree in the first component is built
    stack.pushBoolean(false);
    state = 18; // Build trees in opened components
    break;
}
case 18: { // Build trees in opened components
    if (d.comp < d.ncc) {
        state = 19; // Next component
    } else {
        state = 21; // No more components
    }
```

```
            break;
        }
        case 19: { // Next component
            state = 20; // Build trees in one component
            break;
        }
        case 20: { // Build trees in one component
            stack.pushBoolean(true);
            state = 18; // Build trees in opened components
            break;
        }
        case 21: { // No more components
            state = 22; // Build tree in the hole graph
            break;
        }
        case 22: { // Build tree in the hole graph
            stack.pushBoolean(true);
            state = 9; // If tree is build (окончание)
            break;
        }
        case 23: { // Build solution tree
            stack.pushBoolean(false);
            state = 9; // If tree is build (окончание)
            break;
        }
        case 24: { // test if it is first graph.
            if (d.rec == 0) {
                stack.pushBoolean(true);
                state = 25; // test if it is first graph. (окончание)
            } else {
                stack.pushBoolean(false);
                state = 25; // test if it is first graph. (окончание)
            }
            break;
        }
        case 25: { // test if it is first graph.
                   // (окончание)
            stack.pushBoolean(true);
            state = 4; // Test if there is path to every vertice from A
                       // (окончание)
            break;
        }
        case 26: { // Cannot find solution
            stack.pushBoolean(false);
            state = 4; // Test if there is path to every vertice from A
                       // (окончание)
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // Initialization
            d.funs.init();
            break;
        }
        case 2: { // Test if there is path to every vertice from A
            boolean[] visit = new boolean[d.n];
            d.funs.dfs(d.graph, 0, visit, null, null, 0, d.par, null, -1, 0);
            d.connect = true;
            for (int i = 0; i < d.n; i++) d.connect &= visit[i];
            break;
```

```
}
case 3: { // Test if there is path to every vertice from A
    break;
}
case 4: { // Test if there is path to every vertice from A (окончание)
    break;
}
case 5: { // Finding minimums
    for (int j = 0; j < d.n; j++)
    {
        d.mins[j] = Double.POSITIVE_INFINITY;
        for (int i = 0; i < d.n; i++)
        {
            if (d.graph[i][j] < d.mins[j]) d.mins[j] = d.graph[i][j];
        }
    }
    break;
}
case 6: { // Descend to zero
    for (int j = 0; j < d.n; j++)
    {
        for (int i = 0; i < d.n; i++) d.graph[i][j] -= d.mins[j];
    }
    break;
}
case 7: { // Try to build tree
    boolean[] visit = new boolean[d.n];
    for (int i = 0; i < d.n; i++)
    {
        for (int j = 0; j < d.n; j++)
        {
            if (d.graph[i][j] == 0) d.zeros[i][j] = 0;
            else d.zeros[i][j] = Double.NaN;
        }
    }

    for (int i = 0; i < d.n; i++) d.par[i] = -1;
    d.funs.dfs(d.zeros, 0, visit, null, null, 0, d.par, null, -1, 0);
    d.isTree = true;
    for (int i = 0; i < d.n; i++) d.isTree &= visit[i];
    break;
}
case 8: { // If tree is build
    break;
}
case 9: { // If tree is build (окончание)
    break;
}
case 10: { // Build graph of strongly connected components
    d.funs.pushArrayInt(stack, d.par);
    d.ncc = d.funs.scc(d.zeros, d.scc);
    d.funs.renum(d.scc);
    break;
}
case 11: { // Creating of new graph
    d.funs.pushArrayInt(d.grStack, d.par);
    d.funs.pushArray2(d.grStack, d.graph);
    d.funs.pushArray2(d.grStack, d.zeros);
    d.funs.pushArray2(d.grStack, d.tree);
    d.funs.pushArray1(d.grStack, d.mins);
    d.funs.pushArrayInt(d.grStack, d.scc);
    d.grStack.pushInteger(d.ncc);
```

```
    d.grStack.pushInteger(d.n);

    d.visualizer.copyGraph(true);

    double[][] gr = new double[d.ncc][d.ncc];
    for (int i = 0; i < d.ncc; i++)
    {
        for (int j = 0; j < d.ncc; j++)
        {
            gr[i][j] = Double.POSITIVE_INFINITY;
        }
    }

    for (int i = 0; i < d.n; i++)
    {
        for (int j = 0; j < d.n; j++)
        {
            if (d.scc[i] == d.scc[j]) gr[d.scc[i]][d.scc[j]] = Double.NaN;
            else if (d.graph[i][j] < gr[d.scc[i]][d.scc[j]])
            {
                gr[d.scc[i]][d.scc[j]] = d.graph[i][j];
            }
        }
    }

    for (int i = 0; i < d.ncc; i++)
    {
        for (int j = 0; j < d.ncc; j++)
        {
            if (Double.isInfinite(gr[i][j]))
            {
                gr[i][j] = Double.NaN;
            }
        }
    }

    d.graph = gr;
    d.n = d.ncc;
    d.visualizer.setCollapse(true);
    d.visualizer.reshapePane();
    break;
}
case 12: { // Step before recursion
    d.visualizer.clearGraph();
    d.visualizer.reshapePane();
    d.rec++;
    break;
}
case 13: { // Builds a shortest tree of paths (автомат)
    if (child == null) {
        child = new Main();
        child.toStart();
    }
    child.stepForward(level);
    step-;
    break;
}
case 14: { // Pre Backstep
    d.rec-;
    break;
}
case 15: { // Backstep
```

```
d.funs.pushArrayInt(stack, d.par);
d.funs.pushArray2(stack, d.graph);
d.funs.pushArray2(stack, d.zeros);
d.funs.pushArray2(stack, d.tree);
d.funs.pushArray1(stack, d.mins);
d.funs.pushArrayInt(stack, d.scc);
if (d.crts != null)
{
    d.funs.pushArrayInt(stack, d.crts);
    stack.pushBoolean(true);
}
else stack.pushBoolean(false);
stack.pushInteger(d.ncc);
stack.pushInteger(d.n);
stack.pushInteger(d.comp);
stack.pushInteger(d.vcomp);

d.visualizer.copyGraph(true);

double[][] grt = d.tree;
boolean[] visit = new boolean[d.n];

d.n = d.grStack.popInteger();
d.ncc = d.grStack.popInteger();

d.par = new int[d.n];
d.scc = new int[d.n];
d.crts = null;
d.funs.popArrayInt(d.grStack, d.scc);

d.mins = new double[d.n];
d.funs.popArray1(d.grStack, d.mins);

d.tree = new double[d.n][d.n];
d.funs.popArray2(d.grStack, d.tree);

d.zeros = new double[d.n][d.n];
d.funs.popArray2(d.grStack, d.zeros);

d.graph = new double[d.n][d.n];
d.funs.popArray2(d.grStack, d.graph);

d.par = new int[d.n];
d.funs.popArrayInt(d.grStack, d.par);

d.funs.pushArray2(stack, d.tree);
d.funs.pushArray2(stack, d.zeros);
d.funs.pushArrayInt(stack, d.par);

for (int i = 0; i < d.n; i++)
{
    for (int j = 0; j < d.n; j++)
    {
        if (d.scc[i] != d.scc[j])
        {
            d.zeros[i][j] = Double.NaN;
        }
    }
}
for (int i = 0; i < d.n; i++) d.par[i] = -1;
d.crts = new int[d.ncc];
boolean[] grv = new boolean[d.ncc];
```

```
        //grv[0] = true;
        d.crts[0] = 0;
        for (int i = 0; i < d.n; i++)
        {
            for (int j = 0; j < d.n; j++)
            {
                if (!grv[d.scc[j]] && (d.scc[i] != d.scc[j]) &&
                    (grt[d.scc[i]][d.scc[j]] == d.graph[i][j]))
                {
                    d.tree[i][j] = d.graph[i][j];
                    grv[d.scc[j]] = true;
                    d.crts[d.scc[j]] = j;
                }
                else d.tree[i][j] = d.zeros[i][j];
            }
        }

        d.visualizer.setCollapse(false);
        d.visualizer.reshapePane();
        break;
    }
    case 16: { // Build tree in first component
        d.comp = 0;
        d.vcomp = 0;
        break;
    }
    case 17: { // Tree in the first component is built
        d.funs.pushArray2(stack, d.tree);
        d.visualizer.clearGraph();
        d.visualizer.reshapePane();

        boolean[] visit = new boolean[d.n];
        d.funs.dfs(d.zeros, d.crts[d.comp], visit, null, null, 0,
                   d.par, null, -1, 0);
        d.comp++;
        break;
    }
    case 18: { // Build trees in opened components
        break;
    }
    case 19: { // Next component
        d.vcomp++;
        break;
    }
    case 20: { // Build trees in one component
        d.funs.pushArray2(stack, d.tree);

        boolean[] visit = new boolean[d.n];
        d.funs.dfs(d.zeros, d.crts[d.comp], visit, null, null, 0,
                   d.par, null, -1, 0);
        d.comp++;
        break;
    }
    case 21: { // No more components
        d.vcomp++;
        break;
    }
    case 22: { // Build tree in the hole graph
        d.funs.pushArray2(stack, d.tree);

        for (int i = 0; i < d.n; i++)
        {
```

```
                    for (int j = 0; j < d.n; j++)
                    {
                        if ((d.scc[i] == d.scc[j]) && (i != d.par[j]))
                        {
                            d.tree[i][j] = Double.NaN;
                        }
                    }
                }

                d.funs.restore(d.tree, d.mins);
                break;
            }
            case 23: { // Build solution tree
                d.funs.pushArray2(stack, d.graph);
                d.funs.pushArray2(stack, d.tree);
                d.funs.restore(d.graph, d.mins);
                for (int i = 0; i < d.n; i++)
                {
                    for (int j = 0; j < d.n; j++)
                    {
                        if (d.par[j] == i) d.tree[i][j] = d.graph[i][j];
                        else d.tree[i][j] = Double.NaN;
                    }
                }
                break;
            }
            case 24: { // test if it is first graph.
                break;
            }
            case 25: { // test if it is first graph. (окончание)
                break;
            }
            case 26: { // Cannot find solution
                break;
            }
        }
    } while (!isInteresting(level));
}

/**
  * Сделать шаг в назад.
  */
public void stepBackward(int level) {
    do {
        // Обращение действия в текущем состоянии
        switch (state) {
            case 1: { // Initialization
                break;
            }
            case 2: { // Test if there is path to every vertice from A
                break;
            }
            case 3: { // Test if there is path to every vertice from A
                break;
            }
            case 4: { // Test if there is path to every vertice from A (окончание)
                break;
            }
            case 5: { // Finding minimums
                for (int i=0; i < d.n; i++)
                {
                    d.mins[i] = 0;
```

```
        }
        break;
    }
    case 6: { // Descend to zero
        for (int j = 0; j < d.n; j++)
        {
            for (int i = 0; i < d.n; i++) d.graph[i][j] += d.mins[j];
        }
        break;
    }
    case 7: { // Try to build tree
        d.isTree = false;
        break;
    }
    case 8: { // If tree is build
        break;
    }
    case 9: { // If tree is build (окончание)
        break;
    }
    case 10: { // Build graph of strongly connected components
        d.ncc = 0;
        d.funs.popArrayInt(stack, d.par);
        break;
    }
    case 11: { // Creating of new graph
        d.n = d.grStack.popInteger();
        d.ncc = d.grStack.popInteger();

        d.crts = null;

        d.scc = new int[d.n];
        d.funs.popArrayInt(d.grStack, d.scc);

        d.mins = new double[d.n];
        d.funs.popArray1(d.grStack, d.mins);

        d.tree = new double[d.n][d.n];
        d.funs.popArray2(d.grStack, d.tree);

        d.zeros = new double[d.n][d.n];
        d.funs.popArray2(d.grStack, d.zeros);

        d.graph = new double[d.n][d.n];
        d.funs.popArray2(d.grStack, d.graph);

        d.par = new int[d.n];
        d.funs.popArrayInt(d.grStack, d.par);
        d.visualizer.popGraph();
        d.visualizer.reshapePane();
        break;
    }
    case 12: { // Step before recursion
        d.visualizer.copyGraph(false);
        d.visualizer.reshapePane();
        d.rec–;
        break;
    }
    case 13: { // Builds a shortest tree of paths (автомат)
        if (child == null) {
            child = new Main();
            child.toEnd();
```

```
    }
    child.stepBackward(level);
    step++;
    break;
}
case 14: { // Pre Backstep
    d.rec++;
    break;
}
case 15: { // Backstep
    d.funs.popArrayInt(stack, d.par);
    d.funs.popArray2(stack, d.zeros);
    d.funs.popArray2(stack, d.tree);
    d.funs.pushArrayInt(d.grStack, d.par);
    d.funs.pushArray2(d.grStack, d.graph);
    d.funs.pushArray2(d.grStack, d.zeros);
    d.funs.pushArray2(d.grStack, d.tree);
    d.funs.pushArray1(d.grStack, d.mins);
    d.funs.pushArrayInt(d.grStack, d.scc);
    d.grStack.pushInteger(d.ncc);
    d.grStack.pushInteger(d.n);

    d.vcomp = stack.popInteger();
    d.comp = stack.popInteger();
    d.n = stack.popInteger();
    d.ncc = stack.popInteger();
    boolean b = stack.popBoolean();

    d.scc = new int[d.n];
    if (b)
    {
        d.crts = new int[d.ncc];
        d.funs.popArrayInt(stack, d.crts);
    }
    else d.crts = null;
    d.funs.popArrayInt(stack, d.scc);

    d.mins = new double[d.n];
    d.funs.popArray1(stack, d.mins);

    d.tree = new double[d.n][d.n];
    d.funs.popArray2(stack, d.tree);

    d.zeros = new double[d.n][d.n];
    d.funs.popArray2(stack, d.zeros);

    d.graph = new double[d.n][d.n];
    d.funs.popArray2(stack, d.graph);

    d.par = new int[d.n];
    d.funs.popArrayInt(stack, d.par);
    d.visualizer.popGraph();
    d.visualizer.reshapePane();
    break;
}
case 16: { // Build tree in first component
    break;
}
case 17: { // Tree in the first component is built
    d.visualizer.copyGraph(false);
    d.visualizer.reshapePane();
    d.funs.popArray2(stack, d.tree);
```

```
            break;
        }
        case 18: { // Build trees in opened components
            break;
        }
        case 19: { // Next component
            d.vcomp-;
            break;
        }
        case 20: { // Build trees in one component
            d.funs.popArray2(stack, d.tree);
            d.comp-;
            break;
        }
        case 21: { // No more components
            d.vcomp-;
            break;
        }
        case 22: { // Build tree in the hole graph
            d.funs.popArray2(stack, d.tree);
            break;
        }
        case 23: { // Build solution tree
            d.funs.popArray2(stack, d.tree);
            d.funs.popArray2(stack, d.graph);
            break;
        }
        case 24: { // test if it is first graph.
            break;
        }
        case 25: { // test if it is first graph. (окончание)
            break;
        }
        case 26: { // Cannot find solution
            break;
        }
    }

    // Переход в предыдущее состояние
    switch (state) {
        case 1: { // Initialization
            state = START_STATE;
            break;
        }
        case 2: { // Test if there is path to every vertice from A
            state = 1; // Initialization
            break;
        }
        case 3: { // Test if there is path to every vertice from A
            state = 2; // Test if there is path to every vertice from A
            break;
        }
        case 4: { // Test if there is path to every vertice from A (окончание)
            if (stack.popBoolean()) {
                state = 25; // test if it is first graph. (окончание)
            } else {
                state = 26; // Cannot find solution
            }
            break;
        }
        case 5: { // Finding minimums
            state = 3; // Test if there is path to every vertice from A
```

```
        break;
    }
    case 6: { // Descend to zero
        state = 5; // Finding minimums
        break;
    }
    case 7: { // Try to build tree
        state = 6; // Descend to zero
        break;
    }
    case 8: { // If tree is build
        state = 7; // Try to build tree
        break;
    }
    case 9: { // If tree is build (окончание)
        if (stack.popBoolean()) {
            state = 22; // Build tree in the hole graph
        } else {
            state = 23; // Build solution tree
        }
        break;
    }
    case 10: { // Build graph of strongly connected components
        state = 8; // If tree is build
        break;
    }
    case 11: { // Creating of new graph
        state = 10; // Build graph of strongly connected components
        break;
    }
    case 12: { // Step before recursion
        state = 11; // Creating of new graph
        break;
    }
    case 13: { // Builds a shortest tree of paths (автомат)
        if (child.isAtStart()) {
            child = null;
            state = 12; // Step before recursion
        }
        break;
    }
    case 14: { // Pre Backstep
        state = 13; // Builds a shortest tree of paths (автомат)
        break;
    }
    case 15: { // Backstep
        state = 14; // Pre Backstep
        break;
    }
    case 16: { // Build tree in first component
        state = 15; // Backstep
        break;
    }
    case 17: { // Tree in the first component is built
        state = 16; // Build tree in first component
        break;
    }
    case 18: { // Build trees in opened components
        if (stack.popBoolean()) {
            state = 20; // Build trees in one component
        } else {
            state = 17; // Tree in the first component is built
```

```
                }
                break;
            }
            case 19: { // Next component
                state = 18; // Build trees in opened components
                break;
            }
            case 20: { // Build trees in one component
                state = 19; // Next component
                break;
            }
            case 21: { // No more components
                state = 18; // Build trees in opened components
                break;
            }
            case 22: { // Build tree in the hole graph
                state = 21; // No more components
                break;
            }
            case 23: { // Build solution tree
                state = 8; // If tree is build
                break;
            }
            case 24: { // test if it is first graph.
                state = 9; // If tree is build (окончание)
                break;
            }
            case 25: { // test if it is first graph. (окончание)
                if (stack.popBoolean()) {
                    state = 24; // test if it is first graph.
                } else {
                    state = 24; // test if it is first graph.
                }
                break;
            }
            case 26: { // Cannot find solution
                state = 3; // Test if there is path to every vertice from A
                break;
            }
            case END_STATE: { // Начальное состояние
                state = 4; // Test if there is path to every vertice from A
                           // (окончание)
                break;
            }
        }

        step-;
    } while (!isInteresting(level));
}

/**
  * Интересно ли текущее состояние.
  */
public boolean isInteresting(int level) {
    // Интересность
    switch (state) {
        case START_STATE: // Начальное состояние
            return true;
        case 1: // Initialization
            return level <= 0;
        case 2: // Test if there is path to every vertice from A
            return level <= -1;
```

```
        case 3: // Test if there is path to every vertice from A
            return level <= -1;
        case 4: // Test if there is path to every vertice from A (окончание)
            return level <= -1;
        case 5: // Finding minimums
            return level <= 0;
        case 6: // Descend to zero
            return level <= 0;
        case 7: // Try to build tree
            return level <= 0;
        case 8: // If tree is build
            return level <= 0;
        case 9: // If tree is build (окончание)
            return level <= -1;
        case 10: // Build graph of strongly connected components
            return level <= 0;
        case 11: // Creating of new graph
            return level <= 0;
        case 12: // Step before recursion
            return level <= -1;
        case 13: // Builds a shortest tree of paths (автомат)
            return (child != null) && !child.isAtEnd();
        case 14: // Pre Backstep
            return level <= 0;
        case 15: // Backstep
            return level <= 0;
        case 16: // Build tree in first component
            return level <= 0;
        case 17: // Tree in the first component is built
            return level <= 0;
        case 18: // Build trees in opened components
            return level <= -1;
        case 19: // Next component
            return level <= 0;
        case 20: // Build trees in one component
            return level <= 0;
        case 21: // No more components
            return level <= 0;
        case 22: // Build tree in the hole graph
            return level <= 0;
        case 23: // Build solution tree
            return level <= -1;
        case 24: // test if it is first graph.
            return level <= 0;
        case 25: // test if it is first graph. (окончание)
            return level <= -1;
        case 26: // Cannot find solution
            return level <= 0;
        case END_STATE: // Конечное состояние
            return true;
    }

    throw new RuntimeException("isInterest");
}

/**
  * Комментарий к текущему состоянию
  */
public String getComment() {
    String comment = "";
    Object[] args = null;
    // Выбор комментария
```

```
switch (state) {
    case 1: { // Initialization
        comment = CTree.this.getComment("Main.Initialization");
        break;
    }
    case 5: { // Finding minimums
        comment = CTree.this.getComment("Main.prezero");
        break;
    }
    case 6: { // Descend to zero
        comment = CTree.this.getComment("Main.zero");
        break;
    }
    case 7: { // Try to build tree
        comment = CTree.this.getComment("Main.tryTree");
        break;
    }
    case 8: { // If tree is build
        if (!d.isTree) {
            comment = CTree.this.getComment("Main.testTree.true");
        } else {
            comment = CTree.this.getComment("Main.testTree.false");
        }
        break;
    }
    case 10: { // Build graph of strongly connected components
        comment = CTree.this.getComment("Main.SCC");
        break;
    }
    case 11: { // Creating of new graph
        comment = CTree.this.getComment("Main.newGraph");
        break;
    }
    case 13: { // Builds a shortest tree of paths (автомат)
        comment = child.getComment();
        args = new Object[0];
        break;
    }
    case 14: { // Pre Backstep
        comment = CTree.this.getComment("Main.prebackstep");
        break;
    }
    case 15: { // Backstep
        comment = CTree.this.getComment("Main.backstep");
        break;
    }
    case 16: { // Build tree in first component
        comment = CTree.this.getComment("Main.rootComponent");
        break;
    }
    case 17: { // Tree in the first component is built
        comment = CTree.this.getComment("Main.rootComponentTree");
        break;
    }
    case 19: { // Next component
        comment = CTree.this.getComment("Main.whileTrue");
        break;
    }
    case 20: { // Build trees in one component
        comment = CTree.this.getComment("Main.preBuildtree");
        args = new Object[]{new Character((char) ('A' + d.crts[d.comp - 1]))};
        break;
```

```
        }
        case 21: { // No more components
            comment = CTree.this.getComment("Main.whileFalse");
            break;
        }
        case 22: { // Build tree in the hole graph
            comment = CTree.this.getComment("Main.buildtree");
            break;
        }
        case 24: { // test if it is first graph.
            if (d.rec == 0) {
                comment = CTree.this.getComment("Main.testRec.true");
            } else {
                comment = CTree.this.getComment("Main.testRec.false");
            }
            break;
        }
        case 26: { // Cannot find solution
            comment = CTree.this.getComment("Main.error");
            break;
        }
    }

    return java.text.MessageFormat.format(comment, args);
}

/**
 * Выполняет действия по отрисовке состояния
 */
public void drawState() {
    switch (state) {
        case 1: { // Initialization
            d.visualizer.updatePane(d.visualizer.STANDARD);
            break;
        }
        case 5: { // Finding minimums
            d.visualizer.updatePane(d.visualizer.PREZERO);
            break;
        }
        case 6: { // Descend to zero
            d.visualizer.updatePane(d.visualizer.ZERO);
            break;
        }
        case 7: { // Try to build tree
            d.visualizer.updatePane(d.visualizer.TRY_TREE);
            break;
        }
        case 8: { // If tree is build
            d.visualizer.updatePane(d.visualizer.TRY_TREE);
            break;
        }
        case 10: { // Build graph of strongly connected components
            d.visualizer.updatePane(d.visualizer.BUILD_SCC);
            break;
        }
        case 11: { // Creating of new graph
            d.visualizer.updatePane(d.visualizer.NEW_GRAPH);
            break;
        }
        case 13: { // Builds a shortest tree of paths (автомат)
            child.drawState();
            break;
```

```
            }
            case 14: { // Pre Backstep
                d.visualizer.updatePane(d.visualizer.PRE_BUILD);
                break;
            }
            case 15: { // Backstep
                d.visualizer.updatePane(d.visualizer.OPEN_COMPONENTS);
                break;
            }
            case 16: { // Build tree in first component
                d.visualizer.updatePane(d.visualizer.PRE_TREE);
                break;
            }
            case 17: { // Tree in the first component is built
                    d.visualizer.updatePane(d.visualizer.PRE_TREE);
                break;
            }
            case 19: { // Next component
                d.visualizer.updatePane(d.visualizer.PRE_TREE);
                break;
            }
            case 20: { // Build trees in one component
                d.visualizer.updatePane(d.visualizer.PRE_TREE);
                break;
            }
            case 21: { // No more components
                d.visualizer.updatePane(d.visualizer.PRE_TREE);
                break;
            }
            case 22: { // Build tree in the hole graph
                d.visualizer.updatePane(d.visualizer.BUILD_TREE);
                break;
            }
            case 24: { // test if it is first graph.
                d.visualizer.updatePane(d.visualizer.FINISH);
                break;
            }
            case 26: { // Cannot find solution
                d.visualizer.updatePane(d.visualizer.ERROR);
                break;
            }
        }
    }

    public StringBuffer toString(StringBuffer s) {
        s.append("Main ").append(state).append(" ");
        s.append('(');
        s.append(descriptions[state]);
        s.append(")\n");
        if (child != null && !child.isAtStart() && !child.isAtEnd()) {
            child.toString(s);
        }
        return s;
    }
  }
}
```

Приложение 4. Исходные коды интерфейса визуализатора

```
package ru.ifmo.vizi.ctree;

import ru.ifmo.vizi.ctree.ui.*;
import ru.ifmo.vizi.ctree.widgets.*;
import ru.ifmo.vizi.ctree.timer.*;

import ru.ifmo.vizi.base.ui.*;
import ru.ifmo.vizi.base.*;
import ru.ifmo.vizi.base.widgets.*;

import java.awt.*;
import java.awt.event.*;
import java.util.Stack;

public class CTreeVisualizer extends Base implements MouseListener,
                                                     MouseMotionListener,
                                                     KeyListener
{
    public final static int STANDARD = 0;
    public final static int PREZERO = 11;
    public final static int ZERO = 1;
    public final static int TRY_TREE = 2;
    public final static int BUILD_SCC = 3;
    public final static int NEW_GRAPH = 4;
    public final static int PRE_BUILD = 5;
    public final static int OPEN_COMPONENTS = 6;
    public final static int PRE_TREE = 7;
    public final static int BUILD_TREE = 8;
    public final static int FINISH = 9;
    public final static int ERROR = 10;

    private final Example DEFAULT_EXAMPLE = new Example(3,
        new double[][] {{Double.NaN, 2, 2},
                        {2, Double.NaN, 1},
                        {2, 1, Double.NaN}},
       "Default Example");

    private final CTree auto;
    private final CTree.Data data;
    private final Frame forefather;

    private String minsString = "mins";

    // relative sizes and places of drawing items.
    private double gRadius = 0.75;
    private double vRadius = 0.25;
    private double tSize = 0.9;
    private double split = 0.5;
    private double gsplit = 0.5;
    private boolean gAspectStatus = true;
    private double gAspect = 1.0;
    private boolean tAspectStatus = true;
    private double tAspect = 1.0;
    private double minsplit = 0.0;
    private double maxsplit = 1.0;

    private ShapeStyle[] tableStyle;
    private ShapeStyle[] vertexStyle;
    private ShapeStyle[] arrowStyle;
```

```
private Stack exampleSet;

public int maxVertices = 8;
public int minVertices = 3;

private int nVertices;
private StringPanel examples;
private Rect edit;
private StringBuffer buf;
private int maxChars = 3;
private Panel autoPane;
private Panel controlsPane;
private SpinPanel vertPane;

private boolean editor;
private MultiButton editButton;
private HintedCheckbox showcheck;
private HintedCheckbox modecheck;

private int viziMode;

// variables for splitter.
private int offset;
private boolean moveSplit;
private Graph moveGraph;
private Rect gsplitRect;
private int agsplit;

// components for adding new edge.
private Ellipse arrowBegin;
private Ellipse arrowEnd;
private Arrow newArrow;

// edit frame.
//private EditFrame frame;
private EditDialog frame;

// graphs.
private Graph mainGraph;
private Graph tempGraph;
private final Stack tempGraphs = new Stack();
private CTreeTimer timer;
private DelayPanel delayPanel;
private boolean timerActive;
private boolean collapseSCC;

public CTreeVisualizer(VisualizerParameters parameters)
{
    super(parameters);
    auto = new CTree(locale);
    data = auto.d;
    data.visualizer = this;
    data.funs = new CTreeFuns(auto);
    forefather = parameters.getForefather();

    createInterface(auto);

    clientPane.addMouseListener(this);
    clientPane.addKeyListener(this);
    clientPane.addMouseMotionListener(this);

    tableStyle = ShapeStyle.loadStyleSet(config, "table");
```

```
    vertexStyle = ShapeStyle.loadStyleSet(config, "vertex");
    arrowStyle = ShapeStyle.loadStyleSet(config, "arrow");

    exampleSet = new Stack();
    for (int i = 0; i < config.getInteger("examples", 0); i++)
    {
        Example ex = loadExample("example" + i, DEFAULT_EXAMPLE);
        exampleSet.push(ex);
        examples.addString(ex.getName());
    }

    maxVertices = config.getInteger("vertices-value-max", maxVertices);
    minVertices = config.getInteger("vertices-value-min", minVertices);

    minsString = config.getParameter("minsString", minsString);
    gRadius = config.getDouble("gRadius", gRadius);
    vRadius = config.getDouble("vRadius", vRadius);
    tSize = config.getDouble("tSize", tSize);
    split = config.getDouble("split", split);
    gsplit = config.getDouble("gsplit", gsplit);
    gAspectStatus = config.getBoolean("gAspectStatus", gAspectStatus);
    gAspect = config.getDouble("gAspect", gAspect);
    tAspectStatus = config.getBoolean("tAspectStatus", tAspectStatus);
    tAspect = config.getDouble("tAspect", tAspect);
    minsplit = config.getDouble("min-split", minsplit);
    maxsplit = config.getDouble("max-split", maxsplit);

    gsplitRect = new Rect(new ShapeStyle[]{
                 new ShapeStyle(config, "gsplitRectStyle", tableStyle[0])
                 });
    gsplitRect.setVisible(false);
    gsplitRect.setCursor(new Cursor(Cursor.W_RESIZE_CURSOR));

    mainGraph = new Graph(auto, config, minsString, gRadius, gAspectStatus,
                          gAspect, tAspectStatus, tAspect,
                          vRadius, tSize, split, minsplit, maxsplit,
                          tableStyle, vertexStyle, arrowStyle);

    moveSplit = false;
    boolean show = config.getBoolean("showcheck", true);
    showcheck.setState(show);

    boolean mode = config.getBoolean("modecheck", true);
    modecheck.setState(mode);
    if (mode) viziMode = 0;
    else viziMode = 1;

    timerActive = false;

    setExample((Example) exampleSet.elementAt(0));
    edit = null;
    maxChars = config.getInteger("maxChars", maxChars);

    frame = new EditDialog(config, this, forefather);
    editor = false;
}

public Component createControlsPane()
{
    controlsPane = new Panel(new BorderLayout());

    autoPane = new Panel();
```

```
AutoControlsPane acp = new AutoControlsPane(config, auto, forefather, false);

delayPanel = findDelayPanel(acp);
autoPane.add(acp);
controlsPane.add(autoPane, BorderLayout.NORTH);

Panel specPanel = new Panel();

specPanel.add(examples = new StringPanel(config, "examples", locale)
{
    public void click(double value)
    {
        setExample((Example) exampleSet.elementAt(getCurrentIndex()));
    }
});

specPanel.add(editButton = new MultiButton(config, new String[]{"edit-button",
                                                                 "start-button"})
{
    public int click(int state)
    {
        switch (state)
        {
            case 0:
                setEditor(true);
                return 1;
            case 1:
                setEditor(false);
                return 0;
            default:
                return 0;
        }
    }
});

if (config.getBoolean("has-save", true))
{
    specPanel.add(new HintedButton(config, "save-button")
    {
        public void click()
        {
            int step = 0;
            while (!auto.isAtStart())
            {
                auto.stepBackward(0);
                step++;
            }
            frame.loadExample(new Example(data.n, data.graph,
                                          examples.getCurrentString()), step);
            for (int i = 0; i < step; i++)
            {
                auto.stepForward(0);
            }
            frame.center();
            frame.show();
        }
    });
}
specPanel.add(showcheck = new HintedCheckbox(config, "show-check", locale)
{
    public void click()
    {
```

```
            if (showcheck.getState())
            {
                mainGraph.setTableVisible(true);
                if (tempGraph != null) tempGraph.setTableVisible(true);
            }
            else
            {
                mainGraph.setTableVisible(false);
                if (tempGraph != null) tempGraph.setTableVisible(false);
            }
        }
    });
    specPanel.add(modecheck = new HintedCheckbox(config, "vizi-mode", locale)
    {
        public void click()
        {
            viziMode = (viziMode + 1) % 2;
        }
    });

    controlsPane.add(specPanel, BorderLayout.SOUTH);
    return controlsPane;
}

public DelayPanel findDelayPanel(AutoControlsPane p)
{
    Component[] c = p.getComponents();
    for (int i = 0; i < c.length; i++)
    {
        if (c[i] instanceof DelayPanel) return (DelayPanel) c[i];
    }
    return null;
}

public void clearTimer()
{
    timer.stop();
    timer = null;
}

public void layoutClientPane(int width, int height)
{
    reshapePane();
}

public void copyGraph(boolean isnew)
{
    if (isnew)
    {
        tempGraph = new Graph(mainGraph);
        tempGraphs.push(tempGraph);
    }
    else
    {
        tempGraph = (Graph) tempGraphs.peek();
    }
    if (viziMode == 0) return;       // show two graphs
    timerActive = isnew;
}

public void clearGraph()
{
```

```
        tempGraph = null;
    }

    public void popGraph()
    {
        tempGraphs.pop();
        tempGraph = null;
    }

    public void setCollapse(boolean c)
    {
        collapseSCC = c;
    }

    public void reshapePane()
    {
        if (timer != null)
        {
            timer.stop();
            timer = null;
        }
        int width = clientPane.getSize().width;
        int height = clientPane.getSize().height;
        agsplit = (int) (width * gsplit);
        clientPane.removeAll();
        mainGraph.updateGraph(auto);
        if (tempGraph == null)
        {
            mainGraph.reshapeGraph(0, 0, width, height);
            mainGraph.addTo(clientPane);
            gsplitRect.setVisible(false);
        }
        else if (viziMode == 0)
        {
            mainGraph.reshapeGraph(0, 0, agsplit, height);
            tempGraph.reshapeGraph(0, agsplit, width - agsplit, height);
            mainGraph.addTo(clientPane);
            tempGraph.addTo(clientPane);
            gsplitRect.setVisible(true);
        }
        else if (timerActive)
        {
            mainGraph.reshapeGraph(0, 0, width, height);
            tempGraph.reshapeGraph(0, 0, width, height);
            int delay = delayPanel.getIntValue() / (CTreeTimer.STEPS + 1);
            Graph viewGraph;
            if (collapseSCC) viewGraph = tempGraph;
            else viewGraph = mainGraph;
            timer = new CTreeTimer(this, tempGraph, mainGraph, viewGraph, data, delay);
            timer.start();
            viewGraph.addTo(clientPane);
            gsplitRect.setVisible(false);
        }
        else
        {
            mainGraph.reshapeGraph(0, 0, width, height);
            mainGraph.addTo(clientPane);
            gsplitRect.setVisible(false);
        }

        gsplitRect.setBounds(agsplit - 2, 0, 4, height);
        clientPane.add(gsplitRect);
```

```
    if (newArrow != null)
    {
        clientPane.add(newArrow);
    }
}

public void updatePane(int flag)
{
    switch (flag)
    {
        case NEW_GRAPH:
        case OPEN_COMPONENTS:
            break;
        default:
            if (timer != null)
            {
                timer.stop();
                timer = null;
                mainGraph.reshapeGraph();
            }
    }
    mainGraph.updateGraph(auto);
    mainGraph.updateTable(flag);
    if (tempGraph != null)
    {
        tempGraph.updateTable(flag - 1);
    }
}

public void update()
{
    clientPane.invalidate();
    update(true);
}

public Example loadExample(String prefix, Example def)
{
    String s = config.getParameter(prefix);
    String suffix = "-" + locale.getLanguage();
    Example e = new Example();
    try
    {
        e.readExample(s, config, this, suffix);
    }
    catch (Exception ex)
    {
        System.out.println(ex.getMessage());
    }
    return e;
}

public void addExample(Example e)
{
    exampleSet.push(e);
    examples.addString(e.getName());
    examples.setCurrentIndex(exampleSet.size() - 1);
}

public void deleteExample(int index)
{
    if (exampleSet.size() == 1) return;
    exampleSet.removeElementAt(index);
```

```
        examples.removeString(index);
        if (index < exampleSet.size()) setExample((Example) exampleSet.elementAt(index));
        else setExample((Example) exampleSet.elementAt(index - 1));
    }

    public void setExample(Example e)
    {
        while (!auto.isAtStart()) auto.stepBackward(0);
        data.n = e.getNumber();
        data.graph = e.getMatrix();
        setVertices(data.n);
        int step = e.getStep();
        if (step > 0)
        {
            setEditor(false);
        }
        while (step != 0)
        {
            auto.stepForward(0);
            step-;
        }
    }

    public void setVertices(int n)
    {
        nVertices = n;
        if (vertPane != null) vertPane.setValue(n);
        updateAuto(n);
        mainGraph.updateGraph(auto);
        reshapePane();
        mainGraph.updateTable(STANDARD);
        update();
    }

    public void updateAuto(int n)
    {
        double[][] matrix = new double[n][n];
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if ((i < data.n) && (j < data.n))
                {
                    matrix[i][j] = data.graph[i][j];
                }
                else
                {
                    matrix[i][j] = Double.NaN;
                }
            }
        }
        data.n = n;
        data.graph = matrix;
    }

    public void setEditor(boolean ed)
    {
        editor = ed;
        autoPane.removeAll();
        if (ed)
        {
            while (!auto.isAtStart()) auto.stepBackward(0);
```

```
        autoPane.add(vertPane = new SpinPanel(config, "vertices")
        {
            public void click(double value)
            {
                setVertices((int) value);
            }
        });
        autoPane.add(new HintedButton(config, "clear-button")
        {
            public void click()
            {
                clearEdges();
            }
        });
        setVertices(nVertices);
        editButton.setState(1);
    }
    else
    {
        vertPane = null;
        AutoControlsPane acp = new AutoControlsPane(config, auto, forefather, false);
        delayPanel = findDelayPanel(acp);
        autoPane.add(acp);
        setEdgeLength();
        edit = null;
        newArrow = null;
        arrowBegin = null;
        arrowEnd = null;
        editButton.setState(0);
    }
    controlsPane.doLayout();
    controlsPane.validate();
    mainGraph.updateTable(STANDARD);
    update();
}

public void mouseExited(MouseEvent e)
{
}

public void mouseEntered(MouseEvent e)
{
}

public void mouseClicked(MouseEvent e)
{
    if (editor == false) return;
    Point p = e.getPoint();
    if (edit != null)
    {
        setEdgeLength();
        mainGraph.updateTable(STANDARD);
        edit = null;
    }
    Component c = clientPane.getComponentAt(p);
    if (c instanceof Rect) edit = (Rect) c;
    c = clientPane.getComponentAt(p);
    if (c instanceof Rect) edit = (Rect) c;
    if (edit == null) return;
    int index = mainGraph.getTable().indexOf(edit);
    int i = index / data.n;
    int j = index % data.n;
```

```
    if ((i == j) || (index >= data.n * data.n))
    {
        edit = null;
        return;
    }
    edit.setStyle(4);
    index = i * (data.n - 1) + j;
    if (j > i) index-;
    ((Arrow) mainGraph.getArrows().elementAt(index)).setStyle(4);
    buf = new StringBuffer();
    update();
}

public void mousePressed(MouseEvent e)
{
    if (editor == false) return;
    if (newArrow == null)
    {
        Point p = e.getPoint();
        Component c = clientPane.getComponentAt(p);
        if (c instanceof Ellipse)
        {
            arrowBegin = (Ellipse) c;
        }
        if (arrowBegin != null)
        {
             while (!auto.isAtStart())
            {
                auto.stepBackward(0);
            }
            newArrow = new Arrow(arrowStyle);
            newArrow.setArrow(p.x, p.y, p.x, p.y);
            clientPane.add(newArrow);
        }
    }
    update();
}

public void mouseReleased(MouseEvent e)
{
    if (newArrow == null) return;
    Point p = e.getPoint();
    Component c = clientPane.getComponentAt(p);
    if (c instanceof Ellipse)
    {
        arrowEnd = (Ellipse) c;
    }
    if ((arrowEnd != null) && (arrowBegin != null))
    {
        int i = mainGraph.getVertices().indexOf(arrowBegin);
        int j = mainGraph.getVertices().indexOf(arrowEnd);
        if (i != j)
        {
            if (Double.isNaN(data.graph[i][j])) data.graph[i][j] = 0;
            if (edit != null)
            {
                setEdgeLength();
                mainGraph.updateTable(STANDARD);
            }
            edit = (Rect) mainGraph.getTable().elementAt(i * data.n + j);
            edit.setStyle(4);
            int index = i * (data.n - 1) + j;
```

```
            if (j > i) index-;
            Arrow arr = (Arrow) mainGraph.getArrows().elementAt(index);
            arr.setStyle(4);
            arr.setVisible(true);
            buf = new StringBuffer();
        }
    }
    clientPane.remove(newArrow);
    newArrow = null;
    arrowBegin = null;
    arrowEnd = null;
    update();
}

public void mouseDragged(MouseEvent e)
{
    if (newArrow != null)
    {
        Point p = e.getPoint();
        newArrow.setEnd(p.x, p.y);
        return;
    }
    if (moveSplit)
    {
        if (moveGraph != null)
        {
            moveGraph.setSplit(e.getX() + offset);
        }
        else
        {
            agsplit = e.getX() + offset;
            gsplit = (double) agsplit / clientPane.getSize().width;
            reshapePane();
            update();
        }
    }
}

public void mouseMoved(MouseEvent e)
{
    if (!showcheck.getState()) return;
    if ((e.getX() >= mainGraph.getSplit() - 2) &&
        (e.getX() <= mainGraph.getSplit() + 2))
    {
        offset = mainGraph.getSplit() - e.getX();
        moveSplit = true;
        moveGraph = mainGraph;
        return;
    }
    if (tempGraph != null)
    {
        if ((e.getX() >= tempGraph.getSplit() - 2) &&
            (e.getX() <= tempGraph.getSplit() + 2))
        {
            offset = tempGraph.getSplit() - e.getX();
            moveSplit = true;
            moveGraph = tempGraph;
            return;
        }
    }
    if ((e.getX() >= agsplit - 2) && (e.getX() <= agsplit + 2))
    {
```

```
        offset = agsplit - e.getX();
        moveSplit = true;
        moveGraph = null;
        return;
    }
    moveSplit = false;
}

public void setEdgeLength()
{
    if (edit == null) return;
    int i,j;
    int index = mainGraph.getTable().indexOf(edit);
    i = index / data.n;
    j = index % data.n;
    String s = new String(buf);
    if (s.length() != 0)
    {
        try
        {
            data.graph[i][j] = Double.valueOf(s).doubleValue();
            if (data.graph[i][j] < 0) throw new NumberFormatException();
        }
        catch (NumberFormatException ex)
        {
            data.graph[i][j] = Double.NaN;
        }
    }
    mainGraph.updateGraph(auto);
}

public void clearEdges()
{
    for (int i = 0; i < data.n; i++)
    {
        for (int j = 0; j < data.n; j++)
        {
            data.graph[i][j] = Double.NaN;
        }
    }
    mainGraph.updateGraph(auto);
    mainGraph.updateTable(STANDARD);
    update();
}

public void setEdgeToNaN()
{
    int i,j;
    int index = mainGraph.getTable().indexOf(edit);
    i = index / data.n;
    j = index % data.n;
    data.graph[i][j] = Double.NaN;
    mainGraph.updateGraph(auto);
}

public void keyTyped(KeyEvent e)
{
    if (edit == null) return;
    switch (e.getKeyChar())
    {
        case '\n':
            setEdgeLength();
```

```java
                mainGraph.updateTable(STANDARD);
                edit = null;
                update();
                return;
            case '\b':
                if (buf.length() > 0) buf.setLength(buf.length() - 1);
                break;
            case 27:
                mainGraph.updateTable(STANDARD);
                edit = null;
                break;
            default:
                if (buf.length() < maxChars)
                {
                    buf.append(e.getKeyChar());
                }
        }
        edit.setMessage(new String(buf));
        edit.adjustFontSize();
        update();
    }

    public void keyPressed(KeyEvent e)
    {
        switch (e.getKeyChar())
        {
            case 127:
                if (edit == null) break;
                setEdgeToNaN();
                mainGraph.updateTable(STANDARD);
                edit = null;
                update();
                break;
            default:
        }
    }

    public void keyReleased(KeyEvent e)
    {
    }
}
```

Приложение 5. Исходный код таймера

```
package ru.ifmo.vizi.ctree.timer;

import java.awt.*;
import ru.ifmo.vizi.ctree.*;
import ru.ifmo.vizi.base.timer.*;

public class CTreeTimer extends Timer
{
    public static final int STEPS = 10;

    private CTreeVisualizer vizi;
    private Graph viewGraph;

    private Point[] beginCoords;
    private Point[] endCoords;
    private Point[] currentCoords;
    private int step;

    public CTreeTimer(CTreeVisualizer v, Graph gr, Graph destgr, Graph viewgr, CTree.Data d,
int delay)
    {
        super(delay);
        vizi = v;
        viewGraph = viewgr;

        if (viewgr == gr)
        {
            beginCoords = gr.getCoords();
            currentCoords = gr.getCoords();
        }
        if (viewgr == destgr)
        {
            endCoords = destgr.getCoords();
        }

        Point[] tempCoords;
        if (viewgr == gr)
        {
            tempCoords = destgr.getCoords();
            endCoords = new Point[beginCoords.length];
            for (int i = 0; i < beginCoords.length; i++)
            {
                endCoords[i] = new Point( tempCoords[d.scc[i]] );
            }
        }
        if (viewgr == destgr)
        {
            tempCoords = gr.getCoords();
            beginCoords = new Point[endCoords.length];
            currentCoords = new Point[endCoords.length];
            for (int i = 0; i < endCoords.length; i++)
            {
                beginCoords[i] = new Point( tempCoords[d.scc[i]] );
                currentCoords[i] = new Point( tempCoords[d.scc[i]] );
            }
        }
        step = 0;
        viewGraph.setCoords(currentCoords);
    }
```

```
    public void tick()
    {
        step++;
        for (int i = 0; i < beginCoords.length; i++)
        {
            currentCoords[i].x = beginCoords[i].x +
                    (int) ((double) step * (endCoords[i].x - beginCoords[i].x) / STEPS);
            currentCoords[i].y = beginCoords[i].y +
                    (int) ((double) step * (endCoords[i].y - beginCoords[i].y) / STEPS);
        }
        viewGraph.setCoords(currentCoords);
        if (step == STEPS) vizi.clearTimer();
    }

    public boolean finished()
    {
        return step == STEPS;
    }
}
```

Приложение 6. XML-описание визуализатора

CTree.xml (Основные параметры визуализатора)

```
<?xml version="1.0" encoding="WINDOWS-1251"?>

<!-
    "CTree" visualizer description (example)
    Version: $Id: CTree.xml,v 1.0
->

<!DOCTYPE visualizer PUBLIC
    "-//IFMO Vizi//Visualizer description"
    "c:/work/vizi/scripts/visualizer.dtd"
[
    <!ENTITY algorithm SYSTEM "CTree-Algorithm.xml">
    <!ENTITY configuration SYSTEM "CTree-Configuration.xml">
]>

<visualizer
    id="CTree"
    package="ru.ifmo.vizi.ctree"
    main-class="CTreeVisualizer"

    preferred-width="600"
    preferred-height="400"

    name-ru="Построение кратчайшего дерева в ориентированном графе"
    name-en="Build the shortest tree in oriented graph"

    author-ru="Святослав Пименов"
    author-en="Svyatoslav Pimenov"
    author-email="pimenov@rain.ifmo.ru"

    supervisor-ru="Георгий Корнеев"
    supervisor-en="Georgiy Korneev"
    supervisor-email="kgeorgiy@rain.ifmo.ru"

    copyright-ru="Copyright \u00A9 Кафедра КТ, СПбГУ ИТМО, 2003"
    copyright-en="Copyright \u00A9 Computer Technologies Department, SPb IFMO, 2003"
>
    &algorithm;
    &configuration;
</visualizer>
```

CTree-Algorithm.xml (Описание алгоритма)

```xml
<?xml version="1.0" encoding="WINDOWS-1251"?>

<!-    <!DOCTYPE algorithm SYSTEM "c:/work/vizi/scripts/algorithm.dtd">    ->


<algorithm type="auto-reverse">
    <data>
        <variable description="Number of recursions"> int rec;</variable>
        <variable description="Number of nodes"> int n;</variable>
        <variable description="Matrix of weights"> double[][] graph = null;</variable>
        <variable description="Matrix of zero edges"> double[][] zeros = null;</variable>
        <variable description="Built tree"> double[][] tree = null;</variable>
        <variable description="Array of minimums"> double[] mins = null;</variable>
        <variable description="Array of parents in dfs tree"> int[] par = null;</variable>
        <variable description="Is graph connected"> boolean connect;</variable>
        <variable description="Is graph a tree"> boolean isTree;</variable>
        <variable description="Strongly connected components"> int[] scc = null;</variable>
        <variable description="Number of strongly connected components"> int ncc;</variable>
        <variable description="Current opening component"> int comp;</variable>
        <variable description="Hilighted component"> int vcomp;</variable>
        <variable description="Array of the components' roots"> int[] crts;</variable>
        <variable description="Stack of graphs"> AutoStack grStack = new AutoStack();</variable>
        <variable description="Functions for CTree"> CTreeFuns funs;</variable>
        <variable description="Instance of visualizer"> CTreeVisualizer visualizer;</variable>
        <toString>
            StringBuffer s = new StringBuffer();
            s.append("n = " + n);

            return s.toString();
        </toString>
    </data>

    <auto id="Main" description="Builds a shortest tree of paths">
        <step
            id="Initialization"
            description="Initialization"
            comment-ru="Инициализируем граф. Будем строить кратчайшее дерево путей с корнем A."
            comment-en="Initialize graph. We'll build the shortest tree with root A."
        >
            <draw>
                d.visualizer.updatePane(d.visualizer.STANDARD);
            </draw>
            <direct>
                d.funs.init();
            </direct>
        </step>

        <step
            id="isconnect"
            description="Test if there is path to every vertice from A"
            level="-1"
        >
            <direct>
                boolean[] visit = new boolean[d.n];
                d.funs.dfs(d.graph, 0, visit, null, null, 0, d.par, null, -1, 0);
                d.connect = true;
                for (int i = 0; i &lt; d.n; i++) d.connect &amp;= visit[i];
            </direct>
        </step>

        <if
```

```
id="testconnect"
description="Test if there is path to every vertice from A"
test="d.connect"
level="-1">

<then>
    <step
        id="prezero"
        description="Finding minimums"
        comment-ru="Находим минимум в каждом столбце (то есть минимальный вес дуг,
                    входящих в каждую вершину)"
        comment-en="Find minimums in each colon"
    >
        <draw>
            d.visualizer.updatePane(d.visualizer.PREZERO);
        </draw>
        <direct>
            for (int j = 0; j &lt; d.n; j++)
            {
                d.mins[j] = Double.POSITIVE_INFINITY;
                for (int i = 0; i &lt; d.n; i++)
                {
                    if (d.graph[i][j] &lt; d.mins[j]) d.mins[j] = d.graph[i][j];
                }
            }
        </direct>
        <reverse>
            for (int i=0; i &lt; d.n; i++)
            {
                d.mins[i] = 0;
            }
        </reverse>
    </step>
    <step
        id="zero"
        description="Descend to zero"
        comment-ru="Произвели спуск до нуля во всех столбцах. (т.е. вычли минимум
                    каждого столбца)."
        comment-en="Make descend to zero in every column (subtract minimum
                    of each column)."
    >
        <draw>
            d.visualizer.updatePane(d.visualizer.ZERO);
        </draw>
        <direct>
            for (int j = 0; j &lt; d.n; j++)
            {
                for (int i = 0; i &lt; d.n; i++) d.graph[i][j] -= d.mins[j];
            }
        </direct>
        <reverse>
            for (int j = 0; j &lt; d.n; j++)
            {
                for (int i = 0; i &lt; d.n; i++) d.graph[i][j] += d.mins[j];
            }
        </reverse>
    </step>

    <step
        id="tryTree"
        description="Try to build tree"
        comment-ru="Выясняем, можно ли построить дерево с вершиной A, используя
```

```
                только нулевые дуги, содержащее все вершины."
    comment-en="Try to build tree with root A using edges with zero weight,
                that contains all vertices."
>
    <draw>
        d.visualizer.updatePane(d.visualizer.TRY_TREE);
    </draw>
    <direct>
        boolean[] visit = new boolean[d.n];
        for (int i = 0; i &lt; d.n; i++)
        {
            for (int j = 0; j &lt; d.n; j++)
            {
                if (d.graph[i][j] == 0) d.zeros[i][j] = 0;
                else d.zeros[i][j] = Double.NaN;
            }
        }

        for (int i = 0; i &lt; d.n; i++) d.par[i] = -1;
        d.funs.dfs(d.zeros, 0, visit, null, null, 0, d.par, null, -1, 0);
        d.isTree = true;
        for (int i = 0; i &lt; d.n; i++) d.isTree &amp;= visit[i];
    </direct>
    <reverse>
        d.isTree = false;
    </reverse>
</step>

<if
    id="testTree"
    description="If tree is build"
    test="!d.isTree"
    false-comment-ru="Дерево, содержащее все вершины, удалось построить."
    false-comment-en="Tree, that contains all vertices is built."
    true-comment-ru="Дерево, содержащее все вершины, построь не удалось,
                     поэтому ..."
    true-comment-en="Tree, that contains all vertices cannot be built, so ..."
>
    <draw>
        d.visualizer.updatePane(d.visualizer.TRY_TREE);
    </draw>
    <then>
        <step
            id="SCC"
            description="Build graph of strongly connected components"
            comment-ru="Нашли сильно связанные компоненты графа с нулевыми
                        дугами."
            comment-en="Find strongly connected components of graph with zero
                        length edges."
        >
            <draw>
                d.visualizer.updatePane(d.visualizer.BUILD_SCC);
            </draw>
            <direct>
                d.funs.pushArrayInt(stack, d.par);
                d.ncc = d.funs.scc(d.zeros, d.scc);
                d.funs.renum(d.scc);
            </direct>
            <reverse>
                d.ncc = 0;
                d.funs.popArrayInt(stack, d.par);
            </reverse>
```

```
</step>

<step
    id="newGraph"
    description="Creating of new graph"
    comment-ru="Создали новый граф, вершинами которого будут сильно
               связанные компоненты. Ребру, соединяющие две такие
               компоненты присваиваем минимальный вес из весов ребер,
               соединяющих вершины, находящиеся в разных компонентах."
    comment-en="Create new graph wich vertices are the strongly connected
               components. The length connecting two components is the
               minimum length of the edges, that connect vertices from
               different components."
>
    <draw>
        d.visualizer.updatePane(d.visualizer.NEW_GRAPH);
    </draw>
    <direct>
        d.funs.pushArrayInt(d.grStack, d.par);
        d.funs.pushArray2(d.grStack, d.graph);
        d.funs.pushArray2(d.grStack, d.zeros);
        d.funs.pushArray2(d.grStack, d.tree);
        d.funs.pushArray1(d.grStack, d.mins);
        d.funs.pushArrayInt(d.grStack, d.scc);
        d.grStack.pushInteger(d.ncc);
        d.grStack.pushInteger(d.n);

        d.visualizer.copyGraph(true);

        double[][] gr = new double[d.ncc][d.ncc];
        for (int i = 0; i &lt; d.ncc; i++)
        {
            for (int j = 0; j &lt; d.ncc; j++)
            {
                gr[i][j] = Double.POSITIVE_INFINITY;
            }
        }

        for (int i = 0; i &lt; d.n; i++)
        {
            for (int j = 0; j &lt; d.n; j++)
            {
                if (d.scc[i] == d.scc[j])
                    gr[d.scc[i]][d.scc[j]] = Double.NaN;
                else if (d.graph[i][j] &lt; gr[d.scc[i]][d.scc[j]])
                {
                    gr[d.scc[i]][d.scc[j]] = d.graph[i][j];
                }
            }
        }

        for (int i = 0; i &lt; d.ncc; i++)
        {
            for (int j = 0; j &lt; d.ncc; j++)
            {
                if (Double.isInfinite(gr[i][j]))
                {
                    gr[i][j] = Double.NaN;
                }
            }
        }
```

```
        d.graph = gr;
        d.n = d.ncc;
        d.visualizer.setCollapse(true);
        d.visualizer.reshapePane();
    </direct>

    <reverse>
        d.n = d.grStack.popInteger();
        d.ncc = d.grStack.popInteger();

        d.crts = null;

        d.scc = new int[d.n];
        d.funs.popArrayInt(d.grStack, d.scc);

        d.mins = new double[d.n];
        d.funs.popArray1(d.grStack, d.mins);

        d.tree = new double[d.n][d.n];
        d.funs.popArray2(d.grStack, d.tree);

        d.zeros = new double[d.n][d.n];
        d.funs.popArray2(d.grStack, d.zeros);

        d.graph = new double[d.n][d.n];
        d.funs.popArray2(d.grStack, d.graph);

        d.par = new int[d.n];
        d.funs.popArrayInt(d.grStack, d.par);
        d.visualizer.popGraph();
        d.visualizer.reshapePane();
    </reverse>
</step>

<step
    id="beforeRecursion"
    description="Step before recursion"
    level="-1"
>
    <direct>
        d.visualizer.clearGraph();
        d.visualizer.reshapePane();
        d.rec++;
    </direct>
    <reverse>
        d.visualizer.copyGraph(false);
        d.visualizer.reshapePane();
        d.rec-;
    </reverse>
</step>

<call-auto id="Main" level="0"/>

<step
    id="prebackstep"
    description="Pre Backstep"
    comment-ru="Раскроем компоненты сильной связности предыдущего
                графа."
    comment-en="Open strongly connected components of previous graph."
>
    <draw>
        d.visualizer.updatePane(d.visualizer.PRE_BUILD);
```

```
    </draw>
    <direct>
        d.rec-;
    </direct>
    <reverse>
        d.rec++;
    </reverse>
</step>

<step
    id="backstep"
    description="Backstep"
    comment-ru="Раскрыли компоненты сильной связности предыдущего
                графа."
    comment-en="Open strongly connected components of previous graph."
>
    <draw>
        d.visualizer.updatePane(d.visualizer.OPEN_COMPONENTS);
    </draw>
    <direct>
        d.funs.pushArrayInt(stack, d.par);
        d.funs.pushArray2(stack, d.graph);
        d.funs.pushArray2(stack, d.zeros);
        d.funs.pushArray2(stack, d.tree);
        d.funs.pushArray1(stack, d.mins);
        d.funs.pushArrayInt(stack, d.scc);
        if (d.crts != null)
        {
            d.funs.pushArrayInt(stack, d.crts);
            stack.pushBoolean(true);
        }
        else stack.pushBoolean(false);
        stack.pushInteger(d.ncc);
        stack.pushInteger(d.n);
        stack.pushInteger(d.comp);
        stack.pushInteger(d.vcomp);

        d.visualizer.copyGraph(true);

        double[][] grt = d.tree;
        boolean[] visit = new boolean[d.n];

        d.n = d.grStack.popInteger();
        d.ncc = d.grStack.popInteger();

        d.par = new int[d.n];
        d.scc = new int[d.n];
        d.crts = null;
        d.funs.popArrayInt(d.grStack, d.scc);

        d.mins = new double[d.n];
        d.funs.popArray1(d.grStack, d.mins);

        d.tree = new double[d.n][d.n];
        d.funs.popArray2(d.grStack, d.tree);

        d.zeros = new double[d.n][d.n];
        d.funs.popArray2(d.grStack, d.zeros);

        d.graph = new double[d.n][d.n];
        d.funs.popArray2(d.grStack, d.graph);
```

```
    d.par = new int[d.n];
    d.funs.popArrayInt(d.grStack, d.par);

    d.funs.pushArray2(stack, d.tree);
    d.funs.pushArray2(stack, d.zeros);
    d.funs.pushArrayInt(stack, d.par);

    for (int i = 0; i &lt; d.n; i++)
    {
        for (int j = 0; j &lt; d.n; j++)
        {
            if (d.scc[i] != d.scc[j])
            {
                d.zeros[i][j] = Double.NaN;
            }
        }
    }
    for (int i = 0; i &lt; d.n; i++) d.par[i] = -1;
    d.crts = new int[d.ncc];
    boolean[] grv = new boolean[d.ncc];
    //grv[0] = true;
    d.crts[0] = 0;
    for (int i = 0; i &lt; d.n; i++)
    {
        for (int j = 0; j &lt; d.n; j++)
        {
            if (!grv[d.scc[j]] &amp;&amp;
                (d.scc[i] != d.scc[j]) &amp;&amp;
                (grt[d.scc[i]][d.scc[j]] == d.graph[i][j]))
            {
                d.tree[i][j] = d.graph[i][j];
                grv[d.scc[j]] = true;
                d.crts[d.scc[j]] = j;
            }
            else d.tree[i][j] = d.zeros[i][j];
        }
    }

    d.visualizer.setCollapse(false);
    d.visualizer.reshapePane();
</direct>
<reverse>
    d.funs.popArrayInt(stack, d.par);
    d.funs.popArray2(stack, d.zeros);
    d.funs.popArray2(stack, d.tree);
    d.funs.pushArrayInt(d.grStack, d.par);
    d.funs.pushArray2(d.grStack, d.graph);
    d.funs.pushArray2(d.grStack, d.zeros);
    d.funs.pushArray2(d.grStack, d.tree);
    d.funs.pushArray1(d.grStack, d.mins);
    d.funs.pushArrayInt(d.grStack, d.scc);
    d.grStack.pushInteger(d.ncc);
    d.grStack.pushInteger(d.n);

    d.vcomp = stack.popInteger();
    d.comp = stack.popInteger();
    d.n = stack.popInteger();
    d.ncc = stack.popInteger();
    boolean b = stack.popBoolean();

    d.scc = new int[d.n];
    if (b)
```

```
        {
            d.crts = new int[d.ncc];
            d.funs.popArrayInt(stack, d.crts);
        }
        else d.crts = null;
        d.funs.popArrayInt(stack, d.scc);

        d.mins = new double[d.n];
        d.funs.popArray1(stack, d.mins);

        d.tree = new double[d.n][d.n];
        d.funs.popArray2(stack, d.tree);

        d.zeros = new double[d.n][d.n];
        d.funs.popArray2(stack, d.zeros);

        d.graph = new double[d.n][d.n];
        d.funs.popArray2(stack, d.graph);

        d.par = new int[d.n];
        d.funs.popArrayInt(stack, d.par);
        d.visualizer.popGraph();
        d.visualizer.reshapePane();
    </reverse>
</step>

<step
    id="rootComponent"
    description="Build tree in first component"
    comment-ru="Построим дерево в сильно связанной компоненте,
                содержащей вершину A."
    comment-en="Build tree in the strongly connected component,
                that contains vertex A."
    >
    <draw>
        d.visualizer.updatePane(d.visualizer.PRE_TREE);
    </draw>
    <direct>
        d.comp = 0;
        d.vcomp = 0;
    </direct>
    <reverse>
    </reverse>
</step>
<step
    id="rootComponentTree"
    description="Tree in the first component is built"
    comment-ru="Дерево с корнем в вершине A в этой компоненте
                построили."
    comment-en="Tree in the component with root A is built."
>
    <draw>
        d.visualizer.updatePane(d.visualizer.PRE_TREE);
    </draw>
    <direct>
        d.funs.pushArray2(stack, d.tree);
        d.visualizer.clearGraph();
        d.visualizer.reshapePane();

        boolean[] visit = new boolean[d.n];
        d.funs.dfs(d.zeros, d.crts[d.comp], visit, null, null, 0,
                   d.par, null, -1, 0);
```

```
        d.comp++;
    </direct>
    <reverse>
        d.visualizer.copyGraph(false);
        d.visualizer.reshapePane();
        d.funs.popArray2(stack, d.tree);
    </reverse>
</step>

<while
    id="openComponents"
    description="Build trees in opened components"
    test="d.comp &lt; d.ncc"
    level="-1"
    >

    <step
        id="whileTrue"
        description="Next component"
        comment-ru="Построим дерево в очередной компоненте сильной
                    связности"
        comment-en="Build tree in the next strongly connected component"
        >
        <draw>
            d.visualizer.updatePane(d.visualizer.PRE_TREE);
        </draw>
        <direct>
            d.vcomp++;
        </direct>
        <reverse>
            d.vcomp-;
        </reverse>
    </step>

    <step
        id="preBuildtree"
        description="Build trees in one component"
        comment-ru="В этой компоненте построили дерево с корнем
                    в вершине {0}, в которую входит ребро из другой
                    компоненты."
        comment-en="In this component build tree with root {0},
                    that have parent in another component."
        comment-args="new Character((char) ('A' + d.crts[d.comp - 1]))"
        >
        <draw>
            d.visualizer.updatePane(d.visualizer.PRE_TREE);
        </draw>
        <direct>
            d.funs.pushArray2(stack, d.tree);

            boolean[] visit = new boolean[d.n];
            d.funs.dfs(d.zeros, d.crts[d.comp], visit, null, null, 0,
                       d.par, null, -1, 0);
            d.comp++;
        </direct>
        <reverse>
            d.funs.popArray2(stack, d.tree);
            d.comp-;
        </reverse>
    </step>

</while>
```

```
    <step
        id="whileFalse"
        description="No more components"
        comment-ru="Во всех компонентах деревья построены"
        comment-en="Trees are built in all components"
        >
        <draw>
            d.visualizer.updatePane(d.visualizer.PRE_TREE);
        </draw>
        <direct>
            d.vcomp++;
        </direct>
        <reverse>
            d.vcomp-;
        </reverse>
    </step>

    <step
        id="buildtree"
        description="Build tree in the hole graph"
        comment-ru="В полученном графе построили дерево с корнем
                    в вершине A."
        comment-en="In this graph build tree with root A."
        >
        <draw>
            d.visualizer.updatePane(d.visualizer.BUILD_TREE);
        </draw>
        <direct>
            d.funs.pushArray2(stack, d.tree);

            for (int i = 0; i &lt; d.n; i++)
            {
                for (int j = 0; j &lt; d.n; j++)
                {
                    if ((d.scc[i] == d.scc[j]) &amp;&amp; (i != d.par[j]))
                    {
                        d.tree[i][j] = Double.NaN;
                    }
                }
            }

            d.funs.restore(d.tree, d.mins);
        </direct>
        <reverse>
            d.funs.popArray2(stack, d.tree);
        </reverse>
    </step>
</then>

<else>
    <step
        id="makeTree"
        description="Build solution tree"
        level="-1"
    >
        <direct>
            d.funs.pushArray2(stack, d.graph);
            d.funs.pushArray2(stack, d.tree);
            d.funs.restore(d.graph, d.mins);
            for (int i = 0; i &lt; d.n; i++)
            {
```

```
                                for (int j = 0; j &lt; d.n; j++)
                                {
                                    if (d.par[j] == i) d.tree[i][j] = d.graph[i][j];
                                    else d.tree[i][j] = Double.NaN;
                                }
                            }
                        </direct>
                        <reverse>
                            d.funs.popArray2(stack, d.tree);
                            d.funs.popArray2(stack, d.graph);
                        </reverse>

                    </step>
                </else>
            </if>

            <if
                id="testRec"
                description="test if it is first graph."
                test="d.rec == 0"
                true-comment-ru="Кратчайшее дерево путей построено. Это и есть искомое
                                 решение."
                true-comment-en="The shortest tree of paths is built. This tree is a
                                 solution."
                false-comment-ru="Кратчайшее дерево путей в этом графе построено."
                false-comment-en="The shortest tree of paths in this graph is built."
            >
                <draw>
                    d.visualizer.updatePane(d.visualizer.FINISH);
                </draw>
                <then/>
            </if>
        </then>
        <else>

            <step
                id="error"
                description="Cannot find solution"
                comment-ru="Дерево не может быть построено, так как не все вершины могут
                            быть достигнуты из вершины A."
                comment-en="Tree cannot be built, because not every vertex can be reached
                            from vertex A."
            >
                <draw>
                    d.visualizer.updatePane(d.visualizer.ERROR);
                </draw>
                <direct/>
            </step>
        </else>
    </if>
  </auto>
</algorithm>
```

CTree-Configuration.xml (Конфигурация визуализатора)

```
<?xml version="1.0" encoding="WINDOWS-1251"?>

<!-
    "CTree" visualizer configuration (example).
    Version: $Id: CTree-Configuration.xml,v 1.0
->

<!-<!DOCTYPE configuration SYSTEM "c:\work\CTreeVizi\meta\scripts\configuration.dtd">->

<configuration>
    <property
        description = "Comment pane height"
        param       = "comment-height"
        value       = "60"
    />
    <group
        description = "Save/Load dialog configuration"
        param       = "SaveLoadDialog"
    >
        <property
            description = "Height of the comment pane"
            param       = "CommentPane-lines"
            value       = "2"
        />
        <property
            description = "Width of the text area"
            param       = "columns"
            value       = "40"
        />
        <property
            description = "Height of the text area"
            param       = "rows"
            value       = "12"
        />
    </group>
    <spin-panel
        description = "Number of vertices in the graph"
        param       = "vertices"
        caption-ru  = "Вершин: {0,number,####}"
        caption-en  = "Vertices: {0,number,####}"
        hint-ru     = "Количество вершин в графе"
        hint-en     = "Number of vertices in the graph"
        value       = "3"
        min-value   = "3"
        max-value   = "8"
        step        = "1"
    >
        <button
            param       = "button-less"
            caption-ru  = "&lt;&lt;"
            caption-en  = "&lt;&lt;"
            hint-ru     = "Уменьшить количество вершин"
            hint-en     = "Decrease number of vertices"
        />
        <button
            param       = "button-more"
            caption-ru  = "»"
            caption-en  = "»"
            hint-ru     = "Увеличить количество вершин"
            hint-en     = "Increase number of vertices"
        />
```

```
</spin-panel>
<button
    description = "Removes all edges in the graph"
    param       = "clear-button"
    caption-en  = "Clear graph"
    caption-ru  = "Очистить граф"
    hint-en     = "Remove all edges from the graph"
    hint-ru     = "Удалить все вершины из графа"
/>
<button
    description = "Starts the graph editor"
    param       = "edit-button"
    caption-en  = "Editor"
    caption-ru  = "Редактор"
    hint-en     = "Allows to edit graph"
    hint-ru     = "Редактировать граф"
/>
<button
    description = "Starts the visualizer"
    param       = "start-button"
    caption-en  = "Visualizer"
    caption-ru  = "Визуализатор"
    hint-en     = "Begin visualizing of the algorithm"
    hint-ru     = "Запустить визуализатор"
/>
<button
    description = "Opens save-load dialog"
    param       = "save-button"
    caption-en  = "Save"
    caption-ru  = "Сохранить"
    hint-en     = "Open window to edit graph"
    hint-ru     = "Открыть окно для редактирования графа"
/>
<button
    description = "Previous example"
    param       = "examples-button-less"
    caption-en  = "&lt;&lt;"
    caption-ru  = "&lt;&lt;"
    hint-en     = "Previous example"
    hint-ru     = "Предыдущий пример"
/>
<button
    description = "Next example"
    param       = "examples-button-more"
    caption-en  = "»"
    caption-ru  = "»"
    hint-en     = "Next example"
    hint-ru     = "Следующий пример"
/>

<message
    description = "Hint for Examples string panel"
    param       = "examples-hint"
    message-en  = "Current example"
    message-ru  = "Текущий пример"
/>
<message
    description = "Shows and hides table"
    param       = "show-check-caption"
    message-en  = "Show table"
    message-ru  = "Отображать матрицу"
/>
```

```
<message
    description = "Hint for show-check"
    param       = "show-check-hint"
    message-en  = "Show and hide table"
    message-ru  = "Показать или убрать матрицу"
/>
<message
    description = "Switch between visualization modes"
    param       = "vizi-mode-caption"
    message-en  = "Show two graph"
    message-ru  = "Отображать два графа"
/>
<message
    description = "Hint for vizi-mode-check"
    param       = "vizi-mode-hint"
    message-en  = "Set the mode of visualization"
    message-ru  = "Устанавливает режим визуализации"
/>

<property
    description = "Initial state of show-check"
    param       = "showcheck"
    value       = "true"
/>
<property
    description = "Initial state of vizi-mode-check"
    param       = "modecheck"
    value       = "false"
/>

<styleset
    description = "Table style set"
    param       = "table"
>
    <style
        description     = "text color - 000000"
        border-status   = "true"
        border-color    = "000000"
        fill-status     = "false"
        text-color      = "000000"
        aspect-status   = "false"
        text-align      = "0.5"
        padding         = "0.2"
    >
        <font
            face        = "SansSerif"
            size        = "20"
            style       = "plain"
        />
    </style>
    <style
        description     = "text color - 0000ff"
        text-color      = "0000ff"
    />
    <style
        description      = "text color - 000000, fill color - 00ff00"
        fill-status     = "true"
        fill-color      = "00ff00"
    />
    <style
        description      = "text color - 000000, fill color - 777777"
        fill-status     = "true"
```

```
    fill-color      = "777777"
/>
<style
    description      = "text color - ff0000, fill color - 00ffff"
    fill-status     = "true"
    fill-color      = "00ffff"
    text-color      = "ff0000"
/>
<style
    description      = "text color - 000000, fill color - 0000cc"
    fill-status     = "true"
    fill-color      = "0000cc"
/>
<style
    description      = "text color - 000000, fill color - cc7700"
    fill-status     = "true"
    fill-color      = "cc7700"
/>
<style
    description      = "text color - 000000, fill color - 7700cc"
    fill-status     = "true"
    fill-color      = "7700cc"
/>
<style
    description      = "text color - 000000, fill color - 0077cc"
    fill-status     = "true"
    fill-color      = "0077cc"
/>
<style
    description      = "text color - 000000, fill color - ff0077"
    fill-status     = "true"
    fill-color      = "ff0077"
/>
<style
    description      = "text color - 000000, fill color - 00ff77"
    fill-status     = "true"
    fill-color      = "00ff77"
/>
<style
    description      = "text color - 000000, fill color - ff77cc"
    fill-status     = "true"
    fill-color      = "ff77cc"
/>
<style
    description      = "text color - ff0000"
    text-color      = "ff0000"
/>
<style
    description      = "text color - 000000, no border"
    border-status   = "false"
/>
<style
    description      = "text color - 00ff00, no border"
    border-status   = "false"
    text-color      = "00ff00"
/>
<style
    description      = "text color - cc00ff, no border"
    border-status   = "false"
    text-color      = "cc00ff"
/>
<style
```

```
        description      = "text color - 0000ff, no border"
        border-status   = "false"
        text-color      = "0000ff"
    />
    <style
        description      = "text color - cc7700, no border"
        border-status   = "false"
        text-color      = "cc7700"
    />
    <style
        description      = "text color - 7700cc, no border"
        border-status   = "false"
        text-color      = "7700cc"
    />
    <style
        description      = "text color - 0077cc, no border"
        border-status   = "false"
        text-color      = "0077cc"
    />
    <style
        description      = "text color - ff0077, no border"
        border-status   = "false"
        text-color      = "ff0077"
    />
    <style
        description      = "text color - 00ff77, no border"
        border-status   = "false"
        text-color      = "00ff77"
    />
    <style
        description      = "text color - ff77cc, no border"
        border-status   = "false"
        text-color      = "ff77cc"
    />
    <style
        description      = "text color - ff0000, no border"
        border-status   = "false"
        text-color      = "ff0000"
    />
    <style
        description      = "text color - 0000ff, no border"
        border-status   = "false"
        text-color      = "0000ff"
    />
    <style
        description      = "text color - ff00ff, no border"
        border-status   = "false"
        text-color      = "ff00ff"
    />
</styleset>

<styleset
    description = "Vertex style set"
    param       = "vertex"
>
    <style
        description     = "fill color - ffff00"
        border-status   = "true"
        border-color    = "000000"
        fill-status     = "true"
        fill-color      = "ffff00"
        text-color      = "000000"
```

```
        aspect-status    = "false"
        text-align       = "0.5"
        padding          = "0.2"
    >
        <font
            face         = "SansSerif"
            size         = "20"
            style        = "plain"
        />
    </style>
    <style
        description      = "fill color - 00ff00"
        fill-color       = "00ff00"
    />
    <style
        description      = "fill color - cc00ff"
        fill-color       = "cc00ff"
    />
    <style
        description      = "fill color - 0000ff"
        fill-color       = "0000ff"
    />
    <style
        description      = "fill color - cc7700"
        fill-color       = "cc7700"
    />
    <style
        description      = "fill color - 7700cc"
        fill-color       = "7700cc"
    />
    <style
        description      = "fill color - 0077cc"
        fill-color       = "0077cc"
    />
    <style
        description      = "fill color - ff0077"
        fill-color       = "ff0077"
    />
    <style
        description      = "fill color - 00ffcc"
        fill-color       = "00ffcc"
    />
    <style
        description      = "fill color - ff77cc"
        fill-color       = "ff77cc"
    />
    <style
        description      = "fill color - ff0000"
        fill-color       = "ff0000"
    />
    <style
        description      = "fill color - ff00ff"
        fill-color       = "ff00ff"
    />
</styleset>

<styleset
    description = "Arrow style set"
    param       = "arrow"
>
    <style
        description      = "color - 000000"
```

```
        border-status   = "true"
        border-color    = "000000"
        text-color      = "000000"
        aspect-status   = "false"
        fill-status     = "false"
        text-align      = "0.5"
        padding         = "0.2"
    >
        <font
            face        = "SansSerif"
            size        = "10"
            style       = "plain"
        />
    </style>
    <style
        description     = "color - 0000ff"
        border-color    = "0000ff"
    />
    <style
        description     = "color - 00ff00"
        border-color    = "00ff00"
    />
    <style
        description     = "color - 777777"
        border-color    = "777777"
    />
    <style
        description     = "color - 00ffff"
        border-color    = "00ffff"
    />
    <style
        description     = "color - 0000cc"
        border-color    = "0000cc"
    />
    <style
        description     = "color - cc7700"
        border-color    = "cc7700"
    />
    <style
        description     = "color - 7700cc"
        border-color    = "7700cc"
    />
    <style
        description     = "color - 0077cc"
        border-color    = "0077cc"
    />
    <style
        description     = "color - ff00cc"
        border-color    = "ff00cc"
    />
    <style
        description     = "color - 00ffcc"
        border-color    = "00ffcc"
    />
    <style
        description     = "color - ff77cc"
        border-color    = "ff77cc"
    />
</styleset>

<style
    description     = "Style of splitting rectangle that splits graph and table"
```

```
    param           = "splitRectStyle"
    border-color    = "cccccc"
    fill-status     = "true"
    fill-color      = "cccccc"
/>
<style
    description     = "Style of graph splitting rectangle that splits two graphs"
    param           = "gsplitRectStyle"
    border-color    = "000000"
    fill-status     = "true"
    fill-color      = "000000"
/>

<property
    description = "String of maximum length in minimums row"
    param       = "minsString"
    value       = "888"
/>

<property
    description = "Graph radius relative to width and height of given space"
    param       = "gRadius"
    value       = "0.75"
/>
<property
    description = "Vertex radius relative to graph radius"
    param       = "vRadius"
    value       = "0.25"
/>
<property

    description = "Table size relative to given space"
    param       = "tSize"
    value       = "1.0"
/>
<property
    description = "Position of splitter between graph and table relative to width of
                   given space"
    param       = "split"
    value       = "0.5"
/>
<property
    description = "Position of splitter between two graphs relative to applet width"
    param       = "gsplit"
    value       = "0.5"
/>
<property
    description = "True if graph width and height have constant relation"
    param       = "gAspectStatus"
    value       = "true"
/>
<property
    description = "Relation of graph height to graph width"
    param       = "gAspect"
    value       = "1"
/>
<property
    description = "True if table width and height have constant relation"
    param       = "tAspectStatus"
    value       = "true"
/>
<property
```

```
    description = "Relation of graph height to graph width"
    param       = "tAspect"
    value       = "1"
/>
<property
    description = "Minimal value of splitter position"
    param       = "min-split"
    value       = "0.1"
/>
<property
    description = "Maximal value of splitter position"
    param       = "max-split"
    value       = "0.9"
/>

<group
    description = "Parameters for save-load dialog"
    param       = "edit-frame"
>
    <property
        description = "Name of example"
        param       = "name"
        value       = "Name"
    />
    <property
        description = "Number of vertices"
        param       = "vertices"
        value       = "Vertices"
    />
    <property
        description = "Adjacency matrix"
        param       = "matrix"
        value       = "Matrix"
    />
    <property
        description = "Visualizer step"
        param       = "current"
        value       = "Step"
    />

    <property
        description = "True if comments should be shown in save-load dialog"
        param       = "show-comment"
        value       = "true"
    />
    <message
        description = "Comment for save-load dialog"
        param       = "name-comment"
        message-en  = "/*Name of the example*/"
        message-ru  = "/*Название примера*/"
    />
    <message
        description = "Comment for save-load dialog"
        param       = "vertices-comment"
        message-en  = "/*Number of vertices ({0}...{1})*/"
        message-ru  = "/*Количество вершин ({0}...{1})*/"
    />
    <message
        description = "Comment for save-load dialog"
        param       = "matrix-comment"
        message-en  = "/*The adjacency matrix*/"
        message-ru  = "/*Матрица смежности*/"
```

```
    />
    <message
        description = "Comment for save-load dialog"
        param       = "current-comment"
        message-en  = "/*Current step of the visualizer*/"
        message-ru  = "/*Текущий шаг визуализатора*/"
    />
    <message
        description = "Message displayed if number of vertices is not given before matrix"
        param       = "n-not-given"
        message-en  = "Number of vertices must be given before matrix"
        message-ru  = "Количество вершин должно быть задано перед матрицей"
    />
    <message
        description = "Message displayed if an invalid parameter was encountered"
        param       = "inv-param"
        message-en  = "Line {0}. Invalid parameter {1}. 'Name', 'Vetices', 'Matrix' or
                       'Step' is expected"
        message-ru  = "Строка {0}. Неверный параметр {1}. Допустимы 'Name', 'Vetices',
                       'Matrix' или 'Step'"
    />
    <message
        description = "Message displayed if matrix is not given correctly"
        param       = "no-eq"
        message-en  = "Line {0}. {1} must be followed by &quot;=&quot;"
        message-ru  = "Строка {0}. После {1} должно идти &quot;=&quot;"
    />
    <message
        description = "Message displayed if matrix is not given correctly"
        param       = "no-num"
        message-en  = "Line {0}. Floating-point value or &quot;X&quot; expected
                       instead of &apos;{1}&apos;"
        message-ru  = "Строка {0}: Матрица должна состоять из чисел с плавающей
                       точкой или &quot;X&quot;"
    />
    <message
        description = "Message displayed if matrix is not given correctly"
        param       = "bounds"
        message-en  = "Line {0}. Number between {1} and {2} or &quot;X&quot; expected
                       instead of {3}"
        message-ru  = "Строка {0}. Ожидалось число в диапазоне от {1} до {2} или
                       &quot;X&quot; вместо {3}"
    />
    <message
        description = "Message displayed if matrix is not given correctly"
        param       = "diag"
        message-en  = "Line {0}. Diagonal element ({1}, {1}) in matrix must be
                       &quot;X&quot;, found {2}"
        message-ru  = "Строка {0}: Диагональный элемент ({1}, {1}) в матрице должен
                       быть &quot;X&quot;, обнаружено {2}"
    />
</group>
<property
    description = "Maximum number in matrix"
    param       = "graph-max-val"
    value       = "999"
/>

<property
    description = "Number of examples"
    param       = "examples"
    value       = "5"
```

```
    />
    <property
        description  = "Example: &quot;Test example&quot;"
        param       = "example0"
        value       = "Nameen=&quot;Test example&quot; Nameru= &quot;Тестовый пример&quot;
                       Vertices=3 Matrix=X, 1, 1; 1, X, 2; 1, 2, X;"
    />
    <property
        description  = "Example: &quot;Four vertices&quot;"
        param       = "example1"
        value       = "Nameen=&quot;Four vertices&quot; Nameru= &quot;Четыре вершины&quot;
                       Vertices=4 Matrix=X,2,X,X; 2,X,1,X; X,1,X,1; X,1,1,X;"
    />
    <property
        description  = "Example: &quot;Two recursions&quot;"
        param       = "example2"
        value       = "Nameen=&quot;Two recursions&quot; Nameru= &quot;Две рекурсии&quot;
                       Vertices=5 Matrix=X,5,X,X,5; X,X,1,X,X; X,1,X,2,X; X,X,X,X,1; X,2,X,1,X;"
    />
    <property
        description  = "Example: &quot;Seven vertices1&quot;"
        param       = "example3"
        value       = "Nameen=&quot;Seven vertices1&quot; Nameru= &quot;Семь вершин1&quot;
                       Vertices=7 Matrix=X,5,X,X,5,X,X; X,X,1,X,X,X,X; X,1,X,2,X,X,X;
                       X,X,X,X,1,X,10; X,2,X,1,X,9,X; X,X,X,X,X,X,2; X,X,X,X,X,2,X;"
    />
    <property
        description  = "Example: &quot;Seven vertices2&quot;"
        param       = "example4"
        value       = "Nameen=&quot;Seven vertices2&quot; Nameru= &quot;Семь вершин2&quot;
                       Vertices=7 Matrix=X,5,X,X,5,X,X; X,X,1,X,X,X,X; X,1,X,2,X,X,X;
                       X,X,X,X,1,X,5; X,2,X,1,X,9,X; X,X,X,X,X,X,1; X,X,X,X,X,2,X;"
    />
</configuration>
```