Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

Ю.Д. Бедный

Построение визуализатора нахождения максимального потока в сети методом Диница и Малхотры – Кумара – Махешвари на базе технологии *Vizi*

Программирование с явным выделением состояний
Проектная документация

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

Санкт-Петербург
2005

Оглавление

Введение

На кафедре «Компьютерные технологии» СПбГУ ИТМО для разработки и реализации визуализаторов алгоритмов на основе конечных автоматов была предложена технология *Vizi* [1, 2].

Визуализатор — программа, в процессе работы которой на экране компьютера динамически демонстрируется применение алгоритма к выбранному набору данных.

В данной работе на базе технологии *Vizi* строится визуализатор алгоритма поиска максимального потока в сети методом Диница и Малхотры – Кумара – Махешвари.

1. Анализ литературы

Теоретическую базу для понимания алгоритмов нахождения максимального потока в сети представляет материал в работе [3]. В этой книге приведен также ряд основных определений (граф, сеть, поток, максимальный поток, разрез, дополняющий путь и т.д.) Кроме того, изложен ряд теорем (с доказательствами), представляющих базу обобщенного алгоритма отыскания максимального потока в сети, рассмотрен простой (но малоэффективный) алгоритм Форда – Фалкерсона, приведена временная оценка сложности данного алгоритма. Далее рассмотрено его улучшение (алгоритм Адмондса – Карпа), имеющий полиномиальную временную сложность.

Однако описания алгоритма, являющегося основой для рассматриваемого визуализатора, в работе [3] не приводится. Его можно найти в работе [4], на материал которой целиком опирается данная работа. В этой же книге дано полное теоретическое обоснование метода Диница и Малхотры – Кумара – Махешвари, приведен ряд лемм и теорем, произведена оценка сложности.

Информацию о сравнительной сложности алгоритмов нахождения максимального потока в сети можно найти в работе [5]. В этом же источнике можно ознакомиться с хронологией создания данных алгоритмов.

2. Описание и анализ алгоритма

Рассмотрим один из наиболее эффективных алгоритмов построения максимального потока в сети, предложенный в 1978 году В.М. Малхотрой, П.М Кумаром и С.Н. Махешвари (V.M. Malhotra, P.M. Kumar, S.N. Maheshvari). Данный алгоритм является модификацией метода блокирующего потока, предложенного E.A. Диницем.

Этот алгоритм поддерживает некоторый текущий поток в сети. В начале поток полагается нулевым. Алгоритм состоит из фаз. На каждой фазе на базе остаточной сети строится вспомогательная сеть, такая, что все кратчайшие дополняющие пути из истока в сток являются путями в построенной сети. Сеть состоит из $d + 1$ слоя, где d — расстояние от истока до стока в остаточной сети в начале данной фазы. Отметим, что вспомогательная сеть является ациклической.

Затем во вспомогательной сети строится блокирующий поток. Блокирующим потоком в сети называется такой поток, что любая цепь из истока в сток в этой сети содержит дугу, полностью насыщенную этим потоком. Иначе говоря, поток нельзя увеличить, лишь изменив его на некоторых дугах на больший.

После этого построенный вспомогательный поток суммируется с текущим потоком, остаточная сеть изменяется, и алгоритм переходит к следующей фазе. После каждой фазы длина кратчайшего пути в остаточной сети от источника к стоку увеличивается. Следовательно, общее количество фаз не превышает n — количества вершин в сети. Алгоритм Малхотры, Кумара и Махешвари позволяет построить блокирующий поток в ациклической сети за $O(n^2)$.

Рассмотрим алгоритм подробнее. Для каждой вершины вычислим входящий и исходящий потенциалы вершин — сумму пропускных способностей дуг сети Диница, входящих и исходящих из вершины соответственно. Входящий потенциал истока и исходящий потенциал стока положим равными бесконечности. Определим потенциал или пропускную способность вершины в сети как минимум из ее входящего и исходящего потенциалов. Таким образом, потенциал вершины определяет максимально возможное количество потока, который может через нее проходить. Ясно, что через вершины с нулевым потенциалом поток проходить не может. Следовательно, их можно удалить из вспомогательной сети. Удалим эти вершины и дуги, им инцидентные, обновив должным образом потенциалы вершин, смежных с удаленными. Если в результате появятся новые

вершины с нулевым потенциалом, удалим рекурсивно и их. В результате во вспомогательной сети останутся только вершины с ненулевым потенциалом.

После этого приступим к построению блокирующего потока. Пусть вершина v принадлежит k-ому слою. «Протолкнем» p единиц потока из вершины с минимальным потенциалом в смежные с ней вершины по исходящим дугам с ненулевой остаточной пропускной способностью. Попутно будем переносить проталкиваемый поток в исходную сеть, а также корректировать потенциалы вершин, отправляющих и принимающих избыток потока. В результате весь (в виду минимальности потенциала вершины v) проталкиваемый поток соберется в вершинах (k+1)-го слоя.

Повторим процесс отправки потока из вершин (k+1)-го слоя, содержащих избыток потока, в смежные им вершины (k+2)-го слоя. Это повторяется до тех пор, пока весь поток не соберется в последнем слое. Заметим, что в этом слое содержится только сток, ибо все остальные вершины, ранее ему принадлежащие, были удалены из сети Диница, как вершины, имеющие нулевой потенциал. Следовательно, весь поток величины p, отправленный из вершины с минимальным потенциалом полностью соберется в стоке. На втором этапе вновь, начиная с вершины v, осуществляется «подвод» потока уже по входящим дугам. В результате на первом шаге недостаток потока перенаправляется узлам (k-1)-го слоя, затем (k-2)-го и т.д., пока весь поток величины p, отправленный из вершины с минимальным потенциалом, не соберется в истоке. Таким образом, поток и во вспомогательной и в основной сети увеличится на величину p.

В процессе увеличения потока могли появиться вершины с нулевым потенциалом. Более того, одна такая вершина есть наверняка — это вершина v, ибо ее потенциал используется полностью. Удалим из сети Диница вершины, имеющие нулевой потенциал. Если после данного шага множество вершин сети Диница все еще не пусто, то вновь перейдем к выбору вершины с минимальным значением потенциала, иначе вернемся к построению вспомогательной сети, относительно нового (увеличенного на величину p) потока в основной сети.

3. Описание модели данных

Модель данных – класс, содержащий все необходимые для реализации алгоритма структуры данных. В табл. 1 приведены основные используемые структуры данных.

Таблица 1

Экземпляр апплета	`DMKMVisualizer vis`
Исходная сеть	`int g[][]`
Количество вершин в сети	`int n`
Исток	`int s`
Сток	`int t`
Вспомогательная сеть (сеть Диница)	`int dg[][]`
Поток основной сети	`int f[][]`
Блокирующий поток	`int df[][]`
Величина блокирующего потока	`int pseudoResultFlow`
Потенциалы вершин	`int p[]`
Входной потенциал	`int pin[]`
Выходной потенциал	`int pout[]`
Удаленные вершины	`boolean del[]`
Вершина с минимальным потенциалом	`int minv`
Минимальный потенциал	`int minp`
Можно ли построить сеть Диница	`boolean isNetwork`
Существуют ли не удаленные узлы	`boolean isDel`
Дуги, поток на которых был обновлен за проталкивание	`boolean newArc[][]`
Дуги, поток на которых был обновлен за фазу	`boolean newFlowArc[][]`
Величина максимального потока	`int resultFlow`

Исходная сеть (в которой будет осуществляться поиск максимального потока) и вспомогательная сеть (сеть Диница, в которой ищется блокирующий поток), а также потоки в них, задаются наиболее простым и удобным для реализации визуализатора способом – матрицей смежности, представленной в виде двумерного массива. Следует отметить, что столь простой способ задания не пригоден для написания данного алгоритма, так как не позволяет реализовать его за требуемое время – $O(n^3)$. Однако использование такой структуры в визуализаторе допустимо, так как ограничения по

времени работы алгоритма для визуализатора соблюдаться не должны, а матрица смежности гораздо более удобна в использовании.

4. Описание алгоритма в формате *XML / Java*

В приложении 1 приведен код программы в *xml*-формате, в котором визуализируемый алгоритм представлен на языке *Java*. Программа содержит следующие базовые конструкции:

- последовательности операторов;
- условные операторы (*if-then-else*);
- вызовы функций (*call-auto*);
- циклы с предусловием (*while*)

В таком виде программа может быть преобразована в систему конечных автоматов. Технология *Vizi* генерирует на ее основе программу на языке програvмирования *Java*, работающую под управлением конечных автоматов, код которой приведен в приложении 2.

5. Анализ автоматически сгенерированного кода, реализующего автоматы

На основе технологии *Vizi* из созданного описания алгоритма в формате *XML / Java* автоматически сгенерирован *Java*-код, который соответствует автоматам, реализующим алгоритм нахождения максимального потока в сети методом Диница и Малхотра – Кумары – Мехешвари.

При этом сгенерировано девять автоматов. Каждый автомат соответствует выделенной в алгоритме функции. В табл. 2 указано число состояний для каждого автомата.

Таблица 2

Функция	*Количество состояний*
`Backward`	16
`DeleteZero`	8
`Forward`	18
`GetMinv`	4
`GetPotentials`	3
`Main`	11
`MakeFlow`	15
`MakeNetwork`	22
`UpdateFlow`	3

Большое суммарное число состояний автоматов объясняется сложностью логики реализованного алгоритма и достаточно подробной его визуализацией. Наибольшее число состояний имеют автоматы `MakeNetwork` (построение сети Диница), `Forward` (проталкивание потока из узла в сток), `Backward` (проталкивание потока из этого узла в исток) и `MakeFlow` (построение блокирующего потока в сети Диница). Эти автоматы реализуют практически всю логику алгоритма, а, следовательно, и визуализатора.

Отметим, что технология визи предоставляет возможность автоматической генерации автоматов для прямого и обратного хода визуализации. В данной работе реализация обратного хода выполнена вручную, без использования автоматов.

6. Описание визуализатора, как *Java*-апплета

Созданный на базе технологии *Vizi* визуализатор нахождения максимального потока в сети методом Диница и Малхотры – Кумара – Махешвари (как и любой другой визуализатор, созданный при использовании этой технологии) представляет собой *Java*-апплет. Таким образом, он может быть частью произвольного *HTML*-документа, и, следовательно, просмотрен с помощью любого *Web*-браузера, поддерживающего технологию *Java*-апплетов – имеющего встроенную виртуальную машину *Java*.

Данный визуализатор будет гарантированно работать в следующих средах:

- *AppletViever* vv. 1.1.4, 1.1.8, 1.3, 1.4
- *Internet Explorer* v. 5.0+
- *Netscape Navigator* v. 4.0+
- *Opera* v. 5.0+
- *Mozilla* v. 1.0+

Наиболее «близкой» к визуализатору является виртуальная машина *Java* компании *Sun*. Установить ее можно со следующего адреса: http://java.sun.com

7. Описание интерфейса визуализатора

На рис. 1 представлен типичный вид окна апплета «Нахождение максимального потока в сети методом Диница и Малхотры – Кумара – Махешвари». Основные элементы пользовательского интерфейса пронумерованы, а соответствующие им числа выписаны справа. Ниже дано краткое описание каждого из элементов.

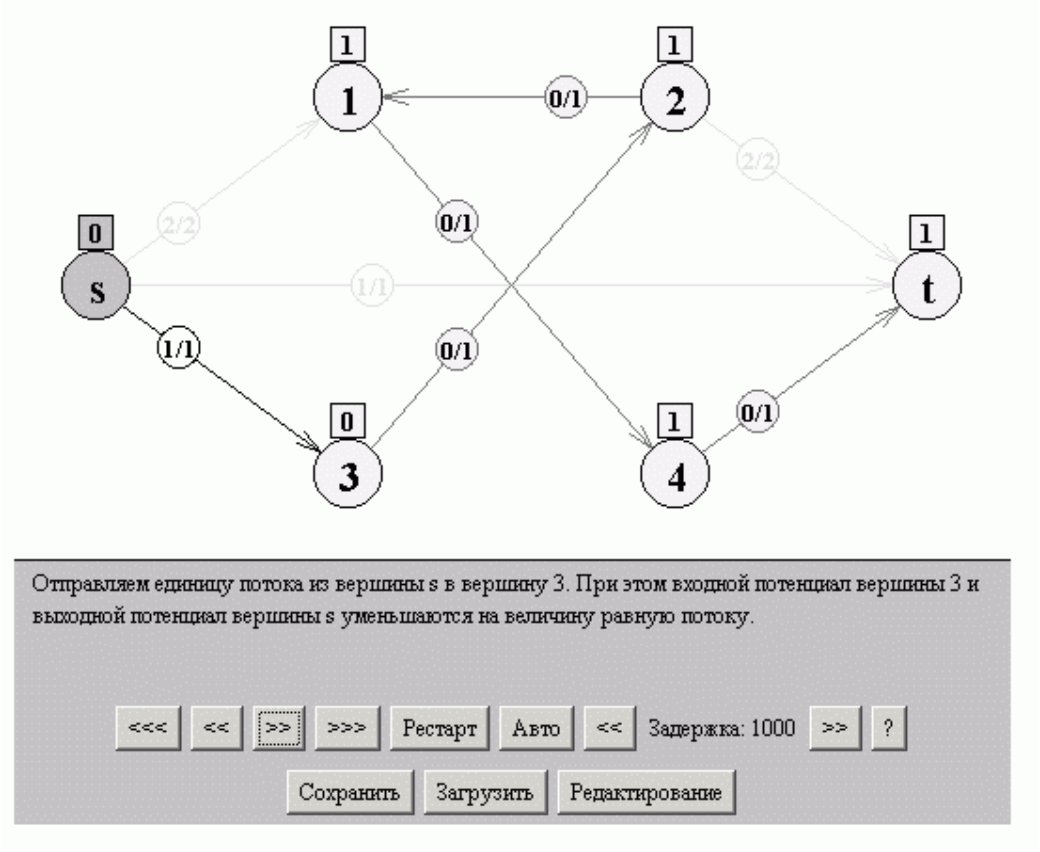

Рис.1. Внешний вид визуализатора

Эта область предназначена для отображения и редактирования сети. В этом режиме пользователь может изменять количество вершин в сети, проводить и удалять дуги, а так же изменять их пропускные способности. В режиме визуализации вершины и рёбра графа выделяются различным цветом (описание значения каждого цвета дано далее).

В этой части окна на каждом шаге визуализации отображаются комментарии к алгоритму.

При помощи кнопок “<<” и “>>” пользователь может перейти к очередному подробному шагу алгоритма или возвратиться к предыдущему. При помощи кнопок “<<<” и “>>>“ осуществляется навигация более крупными шагами. Кнопка «Рестарт» позволяет вернуть алгоритм в начало. Кнопка «Авто» предоставляет возможность запустить визуализацию в автоматическом режиме. Пауза между шагами выставляется элементом

управления «Задержка». Используя кнопку «?», пользователь может получить информацию о визуализаторе.

С помощью кнопки «Сохранить» у пользователя есть возможность сохранить текущее состояние визуализации, которое позже может быть восстановлено кнопкой «Загрузить». При нажатии на нее загружается новая сеть, а визуализация автоматически переводится в начальное состояние. Смена режима редактирование / визуализация осуществляется при помощи кнопки «Редактирование» («Визуализация»). При этом панель управления, предназначенная для визуализации алгоритма, заменяется панелью управления для редактирования сети.

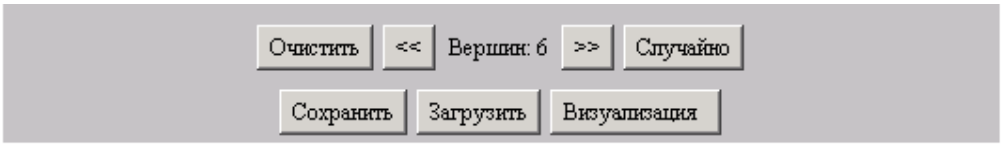

Рис. 2. Внешний вид нижней панели визуализатора

Для удаления из сети всех дуг необходимо использовать кнопку «Очистить». Кнопки “<<” и “>>” предназначены для уменьшения и увеличения вершин в сети соответственно. С помощью кнопки «Случайно» осуществляется генерация случайной сети. В табл.3 приведены обозначения, используемые в визуализаторе.

Таблица 3

Элемент	Описание
	Невыделенная вершина сети Диница
	Вершина сети Диница с минимальным потенциалом
	Вершина основной сети
	Невыделенная дуга сети Диница
	Активная дуга сети Диница
	Дуга основной сети
	Потенциал невыделенной вершины сети Диница
	Минимальный потенциал вершины сети Диница

8. Конфигурация визуализатора

Параметры визуализатора изменяются в файле конфигурации `DMKM-Configuration.xml` (приложение 2). Их описание приведено в табл. 4.

Таблица 4

Параметр	Описание
	Меню редактора
-plus-message	Надпись увеличения пропускной способности дуги.
-minus-message	Надпись уменьшения пропускной способности дуги.
-delete-message	Надпись удаления дуги.
-example	Примеры для загрузки
-num	Количество примеров
-net1-name	Описание первого примера
-net1	Первый пример
-net2-name	Описание второго примера
-net2	Второй пример
-net3-name	Описание третьего примера
-net3	Третий пример
-SaveLoadDialog	Настройки диалога сохранить / загрузить
-CommentPane-lines	Высота области комментария в линиях
-columns	Ширина области комментария
-rows	Высота области комментария
-edit-arc	
-style0	Стиль изменяемой дуги
-style1	Стиль изменяемой дуги
-vertex	
-style0	Стиль вершины
-style1	Стиль вершины
-style2	Стиль вершины
-arc	
-style0	Стиль дуги
-style1	Стиль дуги
-style2	Стиль дуги
-circle	
-style0	Стиль пропускной способности
-style1	Стиль пропускной способности
-style2	Стиль пропускной способности
-rect	
-style0	Стиль потенциала

-style1	Стиль потенциала
-menu-plus	
-style0	Стиль +
-menu-minus	
-style0	Стиль -
-menu-delete	
-style0	Стиль x
-nPanel	Панель для выбора количества вершин в сети
-button-less	Кнопка для увеличения количества вершин
-button-more	Кнопка для уменьшения количества вершин
Дополнительные параметры	
-save-load-button	'Save/Load' button
-netDialog-saveButton	Кнопка для сохранения сети
-netDialog-loadButton	Кнопка для загрузки сети
-netDialog-cancelButton	Кнопка для отмены
-netDialog-save	Сохранить сеть
-netDialog-load	Загрузить сеть
-netDialog-recordNameCaption	Имя записи
-netDialog-listLabel	Список сохраненных сетей:
-source-caption	Надпись на истоке
-sink-caption	Надпись на стоке
-multibutton-edit	Кнопка переключения в режим редактирования
-multibutton-vis	Кнопка переключения в режим визуализации
-save-button	Кнопка сохранения состояния визуализации
-load-button	Кнопка загрузки состояния визуализации
-clear-button	Кнопка для очистки сети
-random-button	Кнопка для генерации случайной сети
-comment-lines	Количество строк в комментариях
-maxArcCapacity	Максимальная пропускная способность дуги
-arc-width-percent	Ширина дуги
-arc-length-percent	Длина дуги
-vertex-diameter-percent	Диаметр вершины
-circle-diameter-percent	Диаметр пропускной способности дуги
-TwoArcsExceptionMsg	Предупреждение: две дуги между вершинами
-ArcsComment	Комментарий параметра "ArcsComment"
-NumberOfVerticesComment	Комментарий параметра "NumberOfVertices"
-NumberOfArcsComment	Комментарий параметра "NumberOfArcs"
-StepComment	Комментарий параметра "Step"

Заключение

Данный визуализатор создан с использованием технология *Vizi,* имеющий ряд положительных особенностей.

Так, например, при ее использовании в значительной степени унифицируется весь процесс создания визуализатора, что избавляет автора от поиска подхода (который может оказаться далеко не лучшим и весьма трудоемким) к реализации поставленной задачи (визуализации алгоритма).

Большинство элементов визуализатора предоставлено разработчику уже в конечном виде, что существенно облегчает его написание. К примеру, автору не стоит заботиться о создании и проработке элементов интерфейса.

Еще одним достоинством унификации интерфейса является то, что все визуализаторы, созданные с использованием технологии *Vizi* обладают стандартным интерфейсом, что облегчает их изучение, желающим ознакомится с визуализируемыми алгоритмами.

К основным недостаткам технологии *Vizi* , на мой взгляд, можно отнести:

- сравнительная трудность ее освоения – разработчику приходится знакомиться с непростой технологией (на что уходит довольно много времени), которая имеет ряд нетривиальных особенностей (скажем *XML*-описание алгоритма), в то время как индивидуальный подход к задаче визуализации гораздо в большей степени понятен разработчику (так как он и есть автор этого подхода);
- неоправданные усилия на визуализацию тривиальных алгоритмов. Размер конечного кода и время, затраченное на его написание в сумме с необходимым для понимания технологии, превышают аналогичные при “простой” реализации.
- непростая отладка кода и поиск ошибок при создании визуализатора объясняются тем, что автор редактирует *XML*-описание алгоритма, а не конечный код (который генерируется автоматически).

Источники

1. *Казаков М.А., Корнеев Г.А., Шалыто А.А*. Разработка логики визуализаторов алгоритмов на основе конечных автоматов //Телекоммуникации и информатизация образования. 2003, № 6, с.27–58. http://is.ifmo.ru/works/vis/

2. *Vizi Home Page*. http://ctddev.ifmo.ru/vizi/

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

4. Асанов М.О. Дискретная оптимизация. Урал Наука, 1998, с.107-138.

5. *Касьянов В.Н., Евстигнеев В.А*. Графы в программировании: обработка, визуализация и применение. БХВ – Петербург, 2003, с.51–57.

Приложение 1. *XML*-описание визуализатора

DMKM.xml (основные параметры)

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


<!DOCTYPE visualizer PUBLIC
    "-//IFMO Vizi//Visualizer description"
    "http://ips.ifmo.ru/vizi/dtd/visualizer.dtd"
[
    <!ENTITY algorithm SYSTEM "DMKM-Algorithm.xml">
    <!ENTITY configuration SYSTEM "DMKM-Configuration.xml">
]>

<visualizer
  id="DMKM"
  package="ru.ifmo.vizi.dmkm"
  main-class="DMKMVisualizer"

  preferred-width="530"
  preferred-height="430"

  name-ru = "Нахождение максимального потока в сети\nметодом Диница и Малхотры-Кумара-Махешвари"
  name-en = "Finding maximal flow in network using\nalgorithm by Malhotra, Kumar and Maheshwari"

  copyright-ru = "Copyright \u00A9 Кафедра КТ, СПб ГИТМО (ТУ), 2003"
  copyright-en = "Copyright \u00A9 Computer Technologies Department, SPb IFMO, 2003"
  author-ru = "Юрий Бедный"
  author-en = "Yuri Bedny"
  author-email = "bedny@rain.ifmo.ru"
  supervisor-ru = "Андрей Станкевич"
  supervisor-en = "Andrew Stankevich"
  supervisor-email = "stankev@rain.ifmo.ru"
>
  &algorithm;
  &configuration;
</visualizer>
```

DMKM-Algorithm.xml (описание алгоритма)

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



   <algorithm>
     <data>
       <variable description="Экземпляр апплета">DMKMVisualizer vis;</variable>
       <variable description="Исходная сеть">int g[][];</variable>
       <variable description="Вспомогательная сеть (сеть Диница)">int dg[][];</variable>
       <variable description="Поток">int f[][];</variable>
       <variable description="Псевдо-поток">int df[][];</variable>
       <variable description="Потенциалы">int p[];</variable>
       <variable description="Входной потенциал">int pin[];</variable>
       <variable description="Выходной потенциал">int pout[];</variable>
       <variable description="Запоминаем удаленные вершины">boolean del[];</variable>
       <variable description="Количество вершин в сети">int n;</variable>
       <variable description="Исток">int s;</variable>
       <variable description="Сток">int t;</variable>
       <variable description="Вершина с минимальным потенциалом">int minv;</variable>

       <variable description="Можно ли построить сеть Диница">boolean isNetwork;</variable>
       <variable description="Есть ли неудаленные узлы">boolean isDel;</variable>
       <variable description="Дуги, поток на котторых юыл обновлен во время
                 forward(backward)">boolean newArc[][];</variable>
       <variable description="Дуги, поток на которых был обновлен за фазу"> boolean
                  newFlowArc[][];</variable>
       <variable description="Впервые ли после построения сети Диница вызывается get_minv">
                  boolean isFirstGetMinv;</variable>
       <variable description="maxint">final int maxint = Integer.MAX_VALUE;</variable>
       <variable description="Величина максимального потока">int resultFlow;</variable>
       <variable description="Минимальный потенциал">int minp;</variable>
```

```
  <variable description="Величнина псевдо-максимального потока">int
            pseudoResultFlow;</variable>

  <variable description="Вспомогательные переменные">int k, c1, c2, q1[], q2[], dst[], i, j,
            pr, pw, dpr, dpw, delta, fl[], q[], dq[];</variable>
  <variable description="Вспомогательные переменные">char sj, si;</variable>
  <variable description="Вспомогательные переменные">boolean isFirstDelete,
           diniz[];</variable>
  <toString>
    return "";
  </toString>
</data>


<auto id="MakeNetwork" description="Строим сеть Диница">
  <step description="Инициализация переменных"
    id="init"
    level="1"
    comment-ru="Начнем очередную фазу с построения вспомогательной сети (сети Диница),
         структура которой отображает все кратчайшие дополняющие s-t цепи в остаточной
         сети."
    comment-en="Start next phase with constructing auxiliary net (Diniz net).
    Structure of this net represents all shortest augmenting s-t chains in residual net."
  >
    <draw>
      d.vis.showDiniz(false);
    </draw>
    <direct>
      int i, j;
      stack.pushInteger(d.c1);
      stack.pushInteger(d.c2);
      for (i = 0; i &lt; d.n; i++)
        stack.pushInteger(d.q1[i]);
      for (i = 0; i &lt; d.n; i++)
        stack.pushInteger(d.q2[i]);
      for (i = 0; i &lt; d.n; i++)
        stack.pushInteger(d.dst[i]);
      for (i = 0; i &lt; d.n; i++)
        for (j = 0; j &lt; d.n; j++)
          stack.pushInteger(d.dg[i][j]);
      for (i = 0; i &lt; d.n; i++)
        for (j = 0; j &lt; d.n; j++)
          stack.pushInteger(d.df[i][j]);
      for (i = 0; i &lt; d.n; i++)
        stack.pushBoolean(d.del[i]);
      d.c1 = 0;
      d.c2 = 1;
      d.q2[0] = d.s;
      for (i = 0; i &lt; d.n; i++) d.dst[i] = d.maxint;
      d.dst[d.s] = 0;
      for (i = 0; i &lt; d.n; i++)
        for (j = 0; j &lt; d.n; j++)
          d.dg[i][j] = d.df[i][j] = 0;
      for (i = 0; i &lt; d.n; i++)
        d.del[i] = true;
    </direct>
    <reverse>
      int i, j;
      for (i = d.n - 1; i &gt;= 0; i--)
        d.del[i] = stack.popBoolean();
      for (i = d.n - 1; i &gt;= 0; i--)
        for (j = d.n - 1; j &gt;= 0; j--)
          d.df[i][j] = stack.popInteger();
      for (i = d.n - 1; i &gt;= 0; i--)
        for (j = d.n - 1; j &gt;= 0; j--)
          d.dg[i][j] = stack.popInteger();
      for (i = d.n - 1; i &gt;= 0; i--)
        d.dst[i] = stack.popInteger();
      for (i = d.n - 1; i &gt;= 0; i--)
        d.q2[i] = stack.popInteger();
      for (i = d.n - 1; i &gt;= 0; i--)
        d.q1[i] = stack.popInteger();
      d.c2 = stack.popInteger();
      d.c1 = stack.popInteger();
    </reverse>
  </step>
  <step description=""
    id="Layer"
    comment-ru="Отнесем исток в нулевой слой. Затем, в (k+1)-ый слой будем включать вершины,
         достижимые из k-го по дугам с ненулевой остаточной пропускной способностью. Также
```

```
        для всех k включим в сеть дуги, которые соединяют вершины k-го слоя с вершинами
        k+1-го."
  comment-en="Register the source at the zero layer. Than we will include in (k + 1) layer
        vertices which are reachable from k layer by arcs with positive residual capacity.
        Also for each k we will include arcs which connect vertices of k and (k+1) layers
        into net."
>
  <draw>
    d.vis.showDiniz(false);
  </draw>
  <direct>
    d.del[0] = false;
  </direct>
  <reverse>
    d.del[0] = true;
  </reverse>
</step>
<while description="Выполняем цикл пока либо не достигнем стока, либо во втором фронте не
        окажется врешин."
  level="-1"
  test="(d.dst[d.t] == d.maxint) &amp;amp;&amp;amp; (d.c2 &gt; 0)"
>
  <step description="Инициализация" level="-1">
    <direct>
      int i;
      for (i = 0; i &lt; d.n; i++)
        stack.pushInteger(d.q1[i]);
      stack.pushInteger(d.c1);
      stack.pushInteger(d.c2);
      stack.pushInteger(d.k);

      for (i = 0; i &lt; d.n; i++) d.q1[i] = d.q2[i];
      d.c1 = d.c2;
      d.c2 = 0;
      d.k = 0;
    </direct>
    <reverse>
      int i;
      d.k = stack.popInteger();
      d.c2 = stack.popInteger();
      d.c1 = stack.popInteger();
      for (i = d.n - 1; i &gt;= 0; i--)
        d.q1[i] = stack.popInteger();
    </reverse>
  </step>
  <while description="Проход по фронту" test="d.k &lt; d.c1" level="-1">
    <step description="Проход по фронту." level="-1">
      <direct>
        stack.pushInteger(d.i);
        stack.pushInteger(d.j);
        d.i = d.q1[d.k];
        d.j = 0;
      </direct>
      <reverse>
        d.j = stack.popInteger();
        d.i = stack.popInteger();
      </reverse>
    </step>
    <while description="" test="d.j &lt; d.n" level="-1">
      <step description="" level="-1">
        <direct>
          int i, j;
          stack.pushInteger(d.c2);
          for (i = 0; i &lt; d.n; i++)
            for (j = 0; j &lt; d.n; j++)
              stack.pushBoolean(d.newFlowArc[i][j]);
          for (i = 0; i &lt; d.n; i++)
            for (j = 0; j &lt; d.n; j++)
              stack.pushInteger(d.dg[i][j]);
          for (i = 0; i &lt; d.n; i++)
            for (j = 0; j &lt; d.n; j++)
              stack.pushInteger(d.df[i][j]);
          for (i = 0; i &lt; d.n; i++)
            stack.pushInteger(d.dst[i]);
          for (i = 0; i &lt; d.n; i++)
            stack.pushInteger(d.q2[i]);
          for (i = 0; i &lt; d.n; i++)
            for (j = 0; j &lt; d.n; j++)
              d.newFlowArc[i][j] = false;
```

```
    if ( (d.dst[d.i] &amp;lt; d.dst[d.j]) &amp;amp;&amp;amp; (d.g[d.i][d.j] &amp;gt;
         d.f[d.i][d.j]) ){
      if (d.dst[d.j] == d.maxint) d.q2[d.c2++] = d.j;
      d.dst[d.j] = d.dst[d.i] + 1;
      d.dg[d.i][d.j] = d.g[d.i][d.j] - d.f[d.i][d.j];
    }
    if ( (d.dst[d.i] &amp;lt; d.dst[d.j]) &amp;amp;&amp;amp; (d.f[d.j][d.i] &amp;gt; 0) ){
      if (d.dst[d.j] == d.maxint) d.q2[d.c2++] = d.j;
      d.dst[d.j] = d.dst[d.i] + 1;
      d.dg[d.i][d.j] = d.f[d.j][d.i];
    }
    if (d.dg[d.i][d.j] > 0) d.newFlowArc[d.i][d.j] = true;
  </direct>
  <reverse>
    int i, j;
    for (i = d.n - 1; i &gt;= 0; i--)
    d.q2[i] = stack.popInteger();
    for (i = d.n - 1; i &gt;= 0; i--)
    d.dst[i] = stack.popInteger();
    for (i = d.n - 1; i &gt;= 0; i--)
      for (j = d.n - 1; j &gt;= 0; j--)
        d.df[i][j] = stack.popInteger();
    for (i = d.n - 1; i &gt;= 0; i--)
      for (j = d.n - 1; j &gt;= 0; j--)
        d.dg[i][j] = stack.popInteger();
    for (i = d.n - 1; i &gt;= 0; i--)
      for (j = d.n - 1; j &gt;= 0; j--)
        d.newFlowArc[i][j] = stack.popBoolean();
    d.c2 = stack.popInteger();
  </reverse>
</step>
<if description="Проеряем добавилось ли новая дуга"
  test="d.dg[d.i][d.j] &gt; 0"
  level="-1"
>
  <then>
    <step description="Случай, когда дуга добавлена"
      id="ArcExist"
      comment-ru="Добавляем в сеть Диница дугу из вершины {0}, принадлежащей {2,
         choice, 0#нулевому| 1#первому| 2#второму| 3#третьему| 4#четвертому|
         5#пятому| 6#шестому| 7#седьмому| 8#восьмому| 9#девятому|10#десятому} слою,
         в вершину {1}, принадлежащую {3, choice, 0#нулевому| 1#первому| 2#второму|
         3#третьему| 4#четвертому| 5#пятому| 6#шестому| 7#седьмому| 8#восьмому|
         9#девятому} слою."
      comment-en="Add an arrow between {0} and {1} vertices in to the Diniz
         network."
      comment-args="new Character(d.si), new Character(d.sj), new
         Integer(d.dst[d.i]), new Integer(d.dst[d.j]), new Integer(d.n - 1)"
    >
      <draw>
        d.vis.showDiniz(false);
      </draw>
      <direct>
        stack.pushCharacter(d.si);
        stack.pushCharacter(d.sj);
        stack.pushBoolean(d.del[d.i]);
        stack.pushBoolean(d.del[d.j]);
        if (d.i == 0) d.si = 's';
        else if (d.i == d.n - 1) d.si = 't';
        else d.si = (char)(d.i + 48);
        if (d.j == 0) d.sj = 's';
        else if (d.j == d.n - 1) d.sj = 't';
        else d.sj = (char)(d.j + 48);

        d.del[d.i] = false;
        d.del[d.j] = false;
      </direct>
      <reverse>
        d.del[d.j] = stack.popBoolean();
        d.del[d.i] = stack.popBoolean();

        d.sj = stack.popCharacter();
        d.si = stack.popCharacter();
      </reverse>
    </step>
  </then>
</if>
<step description="" level="-1">
  <direct>
    d.j++;
```

```
        </direct>
        <reverse>
          d.j--;
        </reverse>
      </step>
    </while>
    <step description="" level="-1">
      <direct>
        d.k++;
      </direct>
      <reverse>
        d.k--;
      </reverse>
    </step>
  </while>
</while>

<step description="Проверяем построили ли мы сеть Диница" level="-1">
  <direct>
    stack.pushBoolean(d.isNetwork);

    d.isNetwork = (d.dst[d.t] != d.maxint);
  </direct>
  <reverse>
    d.isNetwork = stack.popBoolean();
  </reverse>
</step>

<if description="IfDinizNetworkExist" test="d.isNetwork" level="-1">
  <then>
    <step
      description = "LastLayerBuilt"
      id = "LastLayerBuilt"
      level = "0"
      comment-ru = "Слой, которому принадлежит сток достроен."
      comment-en = "The layer which contains the sink has been built."
    >
      <draw>
        d.vis.showDiniz(false);
      </draw>
      <direct>

        int i, j;
        for (i = 0; i &lt; d.n; i++)
          for (j = 0; j &lt; d.n; j++)
            stack.pushBoolean(d.newFlowArc[i][j]);

        for (i = 0; i &lt; d.n; i++)
          for (j = 0; j &lt; d.n; j++)
            d.newFlowArc[i][j] = false;

      </direct>
      <reverse>
        int i, j;
        for (i = d.n - 1; i &gt;= 0; i--)
          for (j = d.n - 1; j &gt;= 0; j--)
            d.newFlowArc[i][j] = stack.popBoolean();

      </reverse>
    </step>
    <step description="Строим сеть Диница"
      id="MakeNetworkBigStep"
      level="1"
      comment-ru="Построение сети Диница завершено. Расстояние до стока в ней равно {0}
       (сток относится {0, choice, 0#к нулевому|1#к первому|2#ко второму|3#к третьему|4#к
       четвертому|5#к пятому| 6#к шестому| 7#к седьмому| 8#к восьмому| 9#к девятому| 10#к
       десятому} слою)."
      comment-en="Diniz network has been built. The distance from source to sink in this
       net equals to {0} (the sink belongs to {0, choice, 0#zero| 1#first| 2#second|
       3#third| 4#fourth|    5#fifth| 6#sixth| 7#seventh| 8#eighth| 9#ninth} layer)."
      comment-args="new Integer(d.dst[d.t])"
    >
      <draw>
       d.vis.showDiniz(false);
      </draw>
      <direct>
      </direct>
      <reverse>
      </reverse>
    </step>
```

```
      <step id="StartPseudomax" description="Определяем псевдомаксимальный поток" level="1"
        comment-ru="Перейдем к построению псевдомаксимального потока в сети Диница.
          Псевдомаксимальным или блокирующим потоком в некоторой сети, называется такой
          поток f, что любая s-t цепь в этой сети содержит дугу, полность насыщенную потоком
          f. Иначе говоря, этот поток нельзя увеличить, лишь изменив поток по некоторым
          дугам на больший."
        comment-en="Turn to building pseudomaximum flow in Diniz net. Pseudomaximum flow is
          a flow that every s-t chain consists the arc, which is saturated with flow f. In
          other words this flow can't be incremented only by incrementing flow on some
          arcs."

      >
        <draw>
          d.vis.showDiniz(false);
        </draw>
        <direct>
        </direct>
        <reverse>
        </reverse>
      </step>
    </then>
  </if>
  <step description="Запоминаем сеть Диница" level="-1">
    <direct>
      int i;
      for (i = 0; i &lt; d.n; i++)
        stack.pushBoolean(d.diniz[i]);
      for (i = 0; i &lt; d.n; i++)
        d.diniz[i] = d.del[i];
    </direct>
    <reverse>
      for (int i = d.n - 1; i &gt;= 0; i--)
        d.diniz[i] = stack.popBoolean();
    </reverse>
  </step>
</auto>

<auto id="GetPotentials" description="Вычисляем потенциалы в узлах вспомогательной сети">
  <step description="Вычисляем потенциалы в узлах вспомогательной сети."
    id="GetPotentials"
    level="1"
    comment-ru="Вычисляем потенциалы вершин. Для этого сначала вычисляем входные и выходные
        потенциалы вершин - сумму пропускных способностей дуг сети Диница, входящих и
        выходящих из вершины, соответственно. Входной потенциал истока и выходной
        потенциал стока полагаем равными бесконечности. Потенциал вершины равен минимуму
        из ее входного и выходного потенциалов."
    comment-en="Calculating vertices potentials. At first we will calculate incoming and
        outcoming potentials - the sum of incoming and outcoming arcs capacities. The
        source outcoming and the sink incoming poteintials let to be infinity. Potential
        of vertex is minimum of it's incoming and outcoming potentials.">
    <draw>
      d.vis.showPotentials();
    </draw>
    <direct>
      int i;
      for (i = 0; i &lt; d.n; i++) stack.pushInteger(d.p[i]);
      for (i = 0; i &lt; d.n; i++) stack.pushInteger(d.pin[i]);
      for (i = 0; i &lt; d.n; i++) stack.pushInteger(d.pout[i]);
      for (i = 0; i &lt; d.n; i++) stack.pushInteger(d.dq[i]);
      stack.pushInteger(d.dpw);
      int j;
      for (i = 0; i &lt; d.n; i++) d.p[i] = d.pin[i] = d.pout[i] = 0;
      for (i = 0; i &lt; d.n; i++)
        for (j = 0; j &lt; d.n; j++)
          if (d.dg[i][j] > 0){
            d.pin[j] += d.dg[i][j];
            d.pout[i] += d.dg[i][j];
          }
      d.pin[d.s] = d.pout[d.t] = d.maxint;
      for (i = 0; i &lt; d.n; i++) {
        if (d.pin[i] &lt; d.pout[i]) d.p[i] = d.pin[i]; else d.p[i] = d.pout[i];
        if (d.p[i] == 0 &amp;amp;&amp;amp; !d.del[i]) d.dq[d.dpw++] = i;
      }
    </direct>
    <reverse>
      int i;
      d.dpw = stack.popInteger();
      for (i = d.n - 1; i &gt;= 0; i--) d.dq[i] = stack.popInteger();
      for (i = d.n - 1; i &gt;= 0; i--) d.pout[i] = stack.popInteger();
      for (i = d.n - 1; i &gt;= 0; i--) d.pin[i] = stack.popInteger();
```

```
      for (i = d.n - 1; i &gt;= 0; i--) d.p[i] = stack.popInteger();
    </reverse>
  </step>
</auto>
<auto id="DeleteZero" description="Удаляем узлы с нулевым потенциалом">
  <step description="Удаляем узлы с нулевыми потенциалами." level="-1">
    <direct>
      stack.pushInteger(d.dpr);
      d.dpr = 0;
    </direct>
    <reverse>
      d.dpr = stack.popInteger();
    </reverse>
  </step>
  <while description="" test="d.dpr &lt; d.dpw" level="-1">
    <step description="" level="0"
      id="DeleteStep"
      comment-ru="Удаляем вершину {0}."
      comment-en="Deleting the vertex number {0}."
      comment-args="new Character(d.si)"
    >
      <draw>
        d.vis.showDelete(!d.isFirstDelete);
      </draw>
      <direct>
        stack.pushCharacter(d.si);
        stack.pushInteger(d.i);
        stack.pushInteger(d.j);
        for (int i = 0; i &lt; d.n; i++)
          stack.pushBoolean(d.del[i]);

        d.i = d.dq[d.dpr++];

        if (d.i == 0) d.si = 's';
        else if (d.i == d.n - 1) d.si = 't';
        else d.si = (char)(d.i + 48);

        d.del[d.i] = true;
        d.j = 0;
      </direct>
      <reverse>
        d.dpr--;
        for (int i = d.n - 1; i &gt;= 0; i--)
          d.del[i] = stack.popBoolean();
        d.j = stack.popInteger();
        d.i = stack.popInteger();
        d.si = stack.popCharacter();
      </reverse>
    </step>
    <step description="" level="-1">
      <direct>
        int i;
        stack.pushInteger(d.dpw);
        for (i = 0; i &lt; d.n; i++) stack.pushInteger(d.dq[i]);
        for (i = 0; i &lt; d.n; i++) stack.pushInteger(d.p[i]);
        for (i = 0; i &lt; d.n; i++) stack.pushInteger(d.pin[i]);
        for (i = 0; i &lt; d.n; i++) stack.pushInteger(d.pout[i]);

        int min;
        for (int j = 0; j &lt; d.n; j++) if (d.p[j] != 0){
          if (d.dg[d.i][j] &gt; 0){
            d.pout[d.i] -= (d.dg[d.i][j] - d.df[d.i][j]);
            d.pin[j] -= (d.dg[d.i][j] - d.df[d.i][j]);
          }
          if (d.dg[j][d.i] &gt; 0){
            d.pout[j] -= (d.dg[j][d.i] - d.df[j][d.i]);
            d.pin[d.i] -= (d.dg[j][d.i] - d.df[j][d.i]);
          }
          if (d.pin[d.i] &lt; d.pout[d.i]) min = d.pin[d.i]; else min = d.pout[d.i];
          if (d.p[d.i] &gt; min) d.p[d.i] = min;
          if (d.pin[j] &lt; d.pout[j]) min = d.pin[j]; else min = d.pout[j];
          if (d.p[j] &gt; min) d.p[j] = min;
          if (d.p[j] == 0) d.dq[d.dpw++] = j;
        }
      </direct>
      <reverse>
        int i;
        for (i = d.n - 1; i &gt;= 0; i--) d.pout[i] = stack.popInteger();
        for (i = d.n - 1; i &gt;= 0; i--) d.pin[i] = stack.popInteger();
        for (i = d.n - 1; i &gt;= 0; i--) d.p[i] = stack.popInteger();
```

```
        for (i = d.n - 1; i &gt;= 0; i--) d.dq[i] = stack.popInteger();
        d.dpw = stack.popInteger();
      </reverse>
    </step>
    <step description=""
      level="0"
      id="UpdatePotentials"
      comment-ru="Корректируем потенциалы вершин, смежных с удаленной вершиной."
      comment-en="Correcting the potentials of vertex which are adjoining with deleted
          vertex."
      comment-args="new Integer(d.i)"
    >
      <draw>
        d.vis.showDelete(!d.isFirstDelete);
      </draw>
      <direct>
      </direct>
      <reverse>
      </reverse>
    </step>
  </while>
  <step description="" level="-1">
    <direct>
      stack.pushBoolean(d.isDel);
      stack.pushInteger(d.dpw);
      d.isDel = true;
      for (int i = 0; i &lt; d.n; i++)
      if (!d.del[i]) d.isDel = false;
      d.dpw = 0;
    </direct>
    <reverse>
      d.dpw = stack.popInteger();
      d.isDel = stack.popBoolean();
    </reverse>
  </step>
</auto>
<auto id="GetMinv" description="Находим узел с минимальным потенциалом">
  <step description="Находим узел с минимальным потенциалом."
    id="GetMinv"
    level="1"
    comment-ru="Находим вершину с минимальным потенциалом. Это вершина {0}, ее потенциал
        равен {1}. Ввиду минимальность потенциала можно гарантировать увеличения потока на
        величину равную этому потенциалу."
    comment-en= "Searching the vertex with minimum potential. This is a vertex number {0}.
        Its potential equals to {1}. The can guarantee flow to be increased by this
        value."
    comment-args="new Character(d.sj), new Integer(d.p[d.minv])"
  >
    <draw>
      d.vis.showMinv(!d.isFirstGetMinv);
    </draw>
    <direct>
      stack.pushCharacter(d.sj);
      stack.pushInteger(d.minv);
      stack.pushInteger(d.minp);

      if (d.minv == 0) d.sj = 's';
      else if (d.minv == d.n - 1) d.sj = 't';
      else d.sj = (char)(d.minv + 48);

      d.minv = d.s;
      for (int i = 0; i &lt; d.n; i++)
        if ( (d.p[i] != 0) &amp;amp;&amp;amp; (d.p[d.minv] &gt; d.p[i]) ) d.minv = i;
          d.minp = d.p[d.minv];

    </direct>
    <reverse>
      d.minp = stack.popInteger();
      d.minv = stack.popInteger();
      d.sj = stack.popCharacter();
    </reverse>
  </step>
  <step description="Отмечаем что мы уже искали вершину с минимальным потенциалом" level="-
      1">
    <direct>
      stack.pushBoolean(d.isFirstGetMinv);
      d.isFirstGetMinv = false;
    </direct>
    <reverse>
      d.isFirstGetMinv = stack.popBoolean();
```

```
    </reverse>
  </step>
</auto>

<auto id="Forward" description="Проталкиваем поток из этого узла в сток">
  <step description="Инициализация" level="1"
    id="forward"
    comment-ru="Будем проталкивать поток величины {0} из вершины {1} в сток, последовательно
         просматривая вершины и жадно перемещая избыток потока в следующий слой, попутно
         корректируя потенциалы вершин, через которые этот поток проходит."
    comment-en="Then we will push flow value of {0} from vertex {1} to sink, greedily moving
         flow's excess to the next layer simultaneously correcting potentials of vertices,
         which are passed by this flow."
    comment-args="new Integer(d.minp), new Character(d.sj)"
  >
    <draw>
      d.vis.showForward();
    </draw>
    <direct>
      stack.pushCharacter(d.sj);
      int i, j;
      for (i = 0; i &lt; d.n; i++)
        for (j = 0; j &lt; d.n; j++)
          stack.pushBoolean(d.newArc[i][j]);
      for (i = 0; i &lt; d.n; i++)
        stack.pushInteger(d.fl[i]);

      stack.pushInteger(d.pr);
      stack.pushInteger(d.pw);
      stack.pushInteger(d.q[0]);

      if (d.minv == 0) d.sj = 's';
      else if (d.minv == d.n - 1) d.sj= 't';
      else d.sj = (char)(d.minv + 48);

      for (i = 0; i &lt; d.n; i++)
        d.fl[i] = 0;
      for (i = 0; i &lt; d.n; i++)
        for (j = 0; j &lt; d.n; j++)
          d.newArc[i][j] = false;
      d.fl[d.minv] = d.p[d.minv];
      d.pr = 0; d.pw = 1; d.q[0] = d.minv;
    </direct>
    <reverse>
      int i, j;
      d.q[0] = stack.popInteger();
      d.pw = stack.popInteger();
      d.pr = stack.popInteger();
      for (i = d.n - 1; i &gt;= 0; i--)
        d.fl[i] = stack.popInteger();
      for (i = d.n - 1; i &gt;= 0; i--)
        for (j = d.n - 1; j &gt;= 0; j--)
          d.newArc[i][j] = stack.popBoolean();
      d.sj = stack.popCharacter();
    </reverse>
  </step>
  <while description="" test="d.pr &lt; d.pw" level="-1">
    <step description=""
      id="HowMuchFlow"
      level="-1"
    >
      <direct>
        stack.pushInteger(d.i);
        d.i = d.q[d.pr++];
      </direct>
      <reverse>
        d.pr--;
        d.i = stack.popInteger();
      </reverse>
    </step>
    <step description="" level="-1">
      <direct>
        stack.pushInteger(d.j);
        d.j = 0;
      </direct>
      <reverse>
        d.j = stack.popInteger();
      </reverse>
    </step>
    <while description="" test="d.j &lt; d.n" level="-1">
```

```
<if description="" test="d.p[d.j] &gt; 0" level="-1">
  <then>
    <step description="" level="-1">
      <direct>
        stack.pushInteger(d.delta);
        if (d.fl[d.i] &lt; d.dg[d.i][d.j] - d.df[d.i][d.j]) d.delta = d.fl[d.i]; else
           d.delta = d.dg[d.i][d.j] - d.df[d.i][d.j];
      </direct>
      <reverse>
        d.delta = stack.popInteger();
      </reverse>
    </step>
    <if description="" test="d.delta &gt; 0" level="-1">
      <then>
        <step description=""
          id="throw"
          level="0"
          comment-ru="Отправляем {0, choice, 1#единицу|2&lt;единицы|5#единиц} потока
                  из вершины {1} в вершину {2}. При этом входной потенциал вершины
                  {2} и выходной потенциал вершины {1} уменьшаются на величину равную
                  потоку."
          comment-en="Pushing flow value {0} from vertex {1} to vertex {2}."
          comment-args="new Integer(d.delta), new Character(d.si), new
                  Character(d.sj)"
        >
          <draw>
            d.vis.showForward();
          </draw>
          <direct>
            stack.pushCharacter(d.si);
            stack.pushCharacter(d.sj);
            stack.pushBoolean(d.newArc[d.i][d.j]);
            stack.pushInteger(d.q[d.pw]);
            stack.pushInteger(d.p[d.j]);
            stack.pushInteger(d.p[d.i]);
            stack.pushInteger(d.f[d.j][d.i]);
            stack.pushInteger(d.f[d.i][d.j]);
            stack.pushInteger(d.dpw);
            for (int i = 0; i &lt; d.n; i++)
              stack.pushInteger(d.dq[i]);

            if (d.i == 0) d.si = 's';
            else if (d.i == d.n - 1) d.si = 't';
            else d.si = (char)(d.i + 48);
            if (d.j == 0) d.sj = 's';
            else if (d.j == d.n - 1) d.sj = 't';
            else d.sj = (char)(d.j + 48);

            d.pin[d.j] -= d.delta;
            d.pout[d.i] -= d.delta;
            if (d.p[d.i] &gt; d.pout[d.i]) {
              if (d.pout[d.i] == 0) d.dq[d.dpw++] = d.i;
              d.p[d.i] = d.pout[d.i];
            }
            if (d.p[d.j] &gt; d.pin[d.j]) {
              if (d.pin[d.j] == 0) d.dq[d.dpw++] = d.j;
              d.p[d.j] = d.pin[d.j];
            }
            d.df[d.i][d.j] += d.delta;
            d.newArc[d.i][d.j] = true;
            d.fl[d.i] -= d.delta;
            d.fl[d.j] += d.delta;
            d.q[d.pw++] = d.j;

            if (d.g[d.i][d.j] &gt; 0) d.f[d.i][d.j] += d.delta;
            else if (d.g[d.j][d.i] &gt; 0) d.f[d.j][d.i] -= d.delta;

          </direct>
          <reverse>
            for (int i = d.n - 1; i &gt;= 0; i--)
              d.dq[i] = stack.popInteger();
            d.dpw = stack.popInteger();
            d.f[d.i][d.j] = stack.popInteger();
            d.f[d.j][d.i] = stack.popInteger();
            d.p[d.i] = stack.popInteger();
            d.p[d.j] = stack.popInteger();
            d.pin[d.j] += d.delta;
            d.pout[d.i] += d.delta;
            d.df[d.i][d.j] -= d.delta;
            d.fl[d.i] += d.delta;
```

```
                    d.fl[d.j] -= d.delta;
                    d.pw--;
                    d.q[d.pw] = stack.popInteger();
                    d.newArc[d.i][d.j] = stack.popBoolean();
                    d.sj = stack.popCharacter();
                    d.si = stack.popCharacter();
                  </reverse>
                </step>
              </then>
            </if>
          </then>
        </if>
        <step description="" level="-1">
          <direct>
            d.j++;
          </direct>
          <reverse>
            d.j--;
          </reverse>
        </step>
      </while>
    </while>
    <step description="" level="-1">
      <direct>
        int i, j;
        for (i = 0; i &lt; d.n; i++)
          for (j = 0; j &lt; d.n; j++)
            stack.pushBoolean(d.newFlowArc[i][j]);

        for (i = 0; i &lt; d.n; i++)
          for (j = 0; j &lt; d.n; j++)
            d.newFlowArc[i][j] = d.newFlowArc[i][j] | d.newArc[i][j];
      </direct>
      <reverse>
        int i, j;
        for (i = d.n - 1; i &gt;= 0; i--)
          for (j = d.n - 1; j &gt;= 0; j--)
            d.newFlowArc[i][j] = stack.popBoolean();
      </reverse>
    </step>
    <step description="Проталкиваем поток из этого узла в сток"
      comment-ru="Проталкивание потока величины {0} из вершины с минимальным потенциалом в
           сток завершено."
      comment-en="Pushing the flow value {0} from vertex with minimum potential to sink has
           been finished."
      comment-args="new Integer(d.minp)"
      id="EndForward"
      level="1"
    >
      <draw>
        d.vis.showForward();
      </draw>
      <direct>
      </direct>
      <reverse>
      </reverse>
    </step>
  </auto>

  <auto id="Backward" description="Проталкиваем поток из этого узла в исток">
    <step description="Инициализация" level="1"
      id="BackwardInit"
      comment-ru="Будем подводить поток величины {0} из истока в вершину {1}, последовательно
           просматривая вершины и жадно подводя недостаток потока из предыдущего слоя,
           попутно корректируя потенциалы вершин, через которые этот поток проходит."
      comment-en="Than we will lead flow value {0} from source to vertex number {1}, greedily
           leading lack of flow from previous layer, simultaneously correcting potentials of
           vertices, which are passed by this flow."
      comment-args="new Integer(d.minp), new Character(d.si)"
    >
      <draw>
        d.vis.showBackward();
      </draw>
      <direct>
        stack.pushCharacter(d.si);
        int i, j;
        for (i = 0; i &lt; d.n; i++)
          for (j = 0; j &lt; d.n; j++)
            stack.pushBoolean(d.newArc[i][j]);
        for (i = 0; i &lt; d.n; i++)
```

```
      stack.pushInteger(d.fl[i]);
    stack.pushInteger(d.pr);
    stack.pushInteger(d.pw);
    stack.pushInteger(d.q[0]);

    if (d.minv == 0) d.si = 's';
    else if (d.minv == d.n - 1) d.si = 't';
    else d.si = (char)(d.minv + 48);

    for (i = 0; i &lt; d.n; i++)
      d.fl[i] = 0;
    for (i = 0; i &lt; d.n; i++)
      for (j = 0; j &lt; d.n; j++)
        d.newArc[i][j] = false;
    d.fl[d.minv] = d.minp;
    d.pr = 0; d.pw = 1; d.q[0] = d.minv;
  </direct>
  <reverse>
    int i, j;
    d.q[0] = stack.popInteger();
    d.pw = stack.popInteger();
    d.pr = stack.popInteger();
    for (i = d.n - 1; i &gt;= 0; i--)
      d.fl[i] = stack.popInteger();
    for (i = d.n - 1; i &gt;= 0; i--)
      for (j = d.n - 1; j &gt;= 0; j--)
        d.newArc[i][j] = stack.popBoolean();
    d.si = stack.popCharacter();
  </reverse>
</step>
<while description="" test="d.pr &lt; d.pw" level="-1">
  <step description=""
    id="HowMuchFlow"
    level="-1"
  >
    <direct>
      stack.pushInteger(d.i);
      d.i = d.q[d.pr++];
    </direct>
    <reverse>
      d.pr--;
      d.i = stack.popInteger();
    </reverse>
  </step>
  <step description="" level="-1">
    <direct>
      stack.pushInteger(d.j);
      d.j = 0;
    </direct>
    <reverse>
      d.j = stack.popInteger();
    </reverse>
  </step>
  <while description="" test="d.j &lt; d.n" level="-1">
    <if description="" test="d.p[d.j] &gt; 0" level="-1">
      <then>
        <step description="" level="-1">
          <direct>
            stack.pushInteger(d.delta);
            if (d.fl[d.i] &lt; d.dg[d.j][d.i] - d.df[d.j][d.i]) d.delta = d.fl[d.i]; else
                    d.delta = d.dg[d.j][d.i] - d.df[d.j][d.i];
          </direct>
          <reverse>
            d.delta = stack.popInteger();
          </reverse>
        </step>
        <if description="" test="d.delta &gt; 0" level="-1">
          <then>
            <step description=""
              id="throw"
              level="0"
              comment-ru="Подводим {0} единиц потока из вершины {2} в вершину {1}. При
               этом входной потенциал вершины {1} и выходной потенциал вершины {2}
               уменьшаются на величину равную потоку."
              comment-en="Leading flow value {0} from vertex {2} to vertex {1}. Also
               incoming potential potential of vertex {1} and outcoming potential of
               vertex {2} decreases by balue of flow."
             comment-args="new Integer(d.delta), new Character(d.si), new Character(d.sj)"
            >
              <draw>
```

```
              d.vis.showForward();
            </draw>
            <direct>
              stack.pushCharacter(d.si);
              stack.pushCharacter(d.sj);
              stack.pushBoolean(d.newArc[d.j][d.i]);
              stack.pushInteger(d.q[d.pw]);
              stack.pushInteger(d.p[d.i]);
              stack.pushInteger(d.p[d.j]);
              stack.pushInteger(d.df[d.i][d.j]);
              stack.pushInteger(d.df[d.j][d.i]);
              stack.pushInteger(d.dpw);
              for (int i = 0; i &lt; d.n; i++)
                stack.pushInteger(d.dq[i]);

              if (d.i == 0) d.si = 's';
              else if (d.i == d.n - 1) d.si = 't';
              else d.si = (char)(d.i + 48);
              if (d.j == 0) d.sj = 's';
              else if (d.j == d.n - 1) d.sj = 't';
              else d.sj = (char)(d.j + 48);

              d.pin[d.i] -= d.delta;
              d.pout[d.j] -= d.delta;
              if (d.p[d.i] &gt; d.pin[d.i]) {
                if (d.pin[d.i] == 0) d.dq[d.dpw++] = d.i;
                d.p[d.i] = d.pin[d.i];
              }
              if (d.p[d.j] &gt; d.pout[d.j]) {
                if (d.pout[d.j] == 0) d.dq[d.dpw++] = d.j;
                d.p[d.j] = d.pout[d.j];
              }
              d.df[d.j][d.i] += d.delta;
              d.newArc[d.j][d.i] = true;
              d.fl[d.i] -= d.delta;
              d.fl[d.j] += d.delta;
              d.q[d.pw++] = d.j;

              if (d.g[d.j][d.i] &gt; 0) d.f[d.j][d.i] += d.delta;
              else if (d.g[d.i][d.j] &gt; 0) d.f[d.i][d.j] -= d.delta;

            </direct>
            <reverse>
              for (int i = d.n - 1; i &gt;= 0; i--)
              d.dq[i] = stack.popInteger();
              d.dpw = stack.popInteger();
              d.f[d.j][d.i] = stack.popInteger();
              d.f[d.i][d.j] = stack.popInteger();
              d.p[d.j] = stack.popInteger();
              d.p[d.i] = stack.popInteger();
              d.pin[d.i] += d.delta;
              d.pout[d.j] += d.delta;
              d.df[d.j][d.i] -= d.delta;
              d.fl[d.i] += d.delta;
              d.fl[d.j] -= d.delta;
              d.pw--;
              d.q[d.pw] = stack.popInteger();
              d.newArc[d.j][d.i] = stack.popBoolean();
              d.sj = stack.popCharacter();
              d.si = stack.popCharacter();
            </reverse>
          </step>
        </then>
      </if>
    </then>
  </if>
  <step description="" level="-1">
    <direct>
      d.j++;
    </direct>
    <reverse>
      d.j--;
    </reverse>
  </step>
 </while>
</while>
<step description="" level="-1">
 <direct>
  int i, j;
  for (i = 0; i &lt; d.n; i++)
```

```
      for (j = 0; j &lt; d.n; j++)
        stack.pushBoolean(d.newFlowArc[i][j]);

      for (i = 0; i &lt; d.n; i++)
        for (j = 0; j &lt; d.n; j++)
          d.newFlowArc[i][j] = d.newFlowArc[i][j] | d.newArc[i][j];
    </direct>
    <reverse>
      int i, j;
      for (i = d.n - 1; i &gt;= 0; i--)
        for (j = d.n - 1; j &gt;= 0; j--)
          d.newFlowArc[i][j] = stack.popBoolean();
    </reverse>
  </step>
  <step description="Проталкиваем поток из этого узла в сток"
    comment-ru="Подвод потока величины {0} к вершине с минимальным потенциалом из истока
        завершен."
    comment-en="Leading flow value of {0} to vertex with minimum potential from source has
        finished."
    comment-args="new Integer(d.minp)"
    id="forward"
    level="1"
  >
    <draw>
      d.vis.showBackward();
    </draw>
    <direct>
    </direct>
    <reverse>
    </reverse>
  </step>
</auto>

<auto id="MakeFlow" description="Строим псевдомаксимальный поток во вспомогательной сети">
  <step description="Изначально нет удаленных узлов" level="-1">
    <direct>
      int i, j;
      stack.pushBoolean(d.isDel);
      stack.pushBoolean(d.isFirstGetMinv);
      for (i = 0; i &lt; d.n; i++)
        for (j = 0; j &lt; d.n; j++)
          stack.pushBoolean(d.newFlowArc[i][j]);

      for (i = 0; i &lt; d.n; i++)
        for (j = 0; j &lt; d.n; j++)
          d.newFlowArc[i][j] = false;
      d.isDel = false;
      d.isFirstGetMinv = true;
    </direct>
    <reverse>
      int i, j;
      for (i = d.n - 1; i &gt;= 0; i--)
        for (j = d.n - 1; j &gt;= 0; j--)
          d.newFlowArc[i][j] = stack.popBoolean();
      d.isFirstGetMinv = stack.popBoolean();
      d.isDel = stack.popBoolean();
    </reverse>
  </step>
  <step description="Удаляем узлы с нулевым потенциалом. После построения сети Диница."
    id="FirstDeleteZero"
    level="1"
    comment-ru="Удалим вершины с нулевым потенциалом. Понятно, что через такие вершины и
        инциндентные им дуги, поток в рассматриваемой вспомогательной сети проходить не
        будет."
    comment-en="Than we will remove the vertices with zero potential value. These vertices
        can't passed by flow in Diniz net."
  >
    <draw>
      d.vis.showPotentials();
    </draw>
    <direct>
      stack.pushBoolean(d.isFirstDelete);
      d.isFirstDelete = true;
    </direct>
    <reverse>
      d.isFirstDelete = stack.popBoolean();
    </reverse>
  </step>
  <call-auto id="DeleteZero"/>
  <step description=""
```

```
  level="1"
  id="FinishFirstDeleteZero"
  comment-ru="Все вершины с нулевым потенциалом удалены."
  comment-en="All vertices with zero potential has been removed."
>
  <draw>
    d.vis.showDelete(false);
  </draw>
  <direct>
  </direct>
  <reverse>
  </reverse>
</step>
<while description="Строим псевдомаксимальный поток в сети Диница, пока не все узлы
       удалены" test="!d.isDel" level="-1"
>
  <call-auto id="GetMinv"/>
  <call-auto id="Forward"/>
  <call-auto id="Backward"/>
  <step description="delete_zero_after_flow"
    id="DeleteZeroAfterFlow"
    level="1"
    comment-ru="Удалим вершины из сети Диница, потенциалы которых стали равны нулю из-за
       полного использования их входного и(или) выходного потенциалов при построении
       потока."
    comment-en="Than we will remove the vertices which potential becomes zero from Diniz
       net, because their incoming or outcoming potential has been used."
  >
    <draw>
      d.vis.showDelete(true);
    </draw>
    <direct>
      stack.pushBoolean(d.isFirstDelete);
      d.isFirstDelete = false;
    </direct>
    <reverse>
      d.isFirstDelete = stack.popBoolean();
    </reverse>
  </step>
  <call-auto id="DeleteZero"/>
  <step description=""
    level="1"
    id="FinishDeleteZero"
    comment-ru="Все вершины с нулевым потенциалом удалены."
    comment-en="All vertices with zero potential has been removed."
  >
    <draw>
      d.vis.showDelete(true);
    </draw>
    <direct>
    </direct>
    <reverse>
    </reverse>
  </step>
</while>
<step description=""
  id="EndPhase"
  comment-ru="Фаза закончена, так как сеть Диница больше не содержит вершин."
  comment-en="The phase has been done, because there are no vertices in Diniz net."
>
  <draw>
    d.vis.showDelete(true);
  </draw>
  <direct>
  </direct>
  <reverse>
  </reverse>
</step>
<step description="Псевдомаксимальный поток в сети Диница построен."
  id="EndDinizStep"
  level="1"
  comment-ru="Псевдомаксимальный поток в сети Диница построен. Его величина равна {0}."

  comment-en="Pseudomaximum flow in Diniz net has been built. It is value equals to {0}"
  comment-args="new Integer(d.pseudoResultFlow)"
>
  <draw>
    d.vis.showDiniz(true);
  </draw>
```

```
    <direct>
      int i;
      for (i = 0; i &lt; d.n; i++)
        stack.pushBoolean(d.del[i]);
      stack.pushInteger(d.pseudoResultFlow);

      for (i = 0; i &lt; d.n; i++)
        d.del[i] = d.diniz[i];
      d.pseudoResultFlow = 0;
      for (i = 0; i &lt; d.n; i++)
      d.pseudoResultFlow += d.df[0][i];
    </direct>
    <reverse>
      d.pseudoResultFlow = stack.popInteger();
    for (int i = d.n - 1; i &gt;= 0; i--)
      d.del[i] = stack.popBoolean();
    </reverse>
  </step>
</auto>

<auto id="UpdateFlow" description="Убновляем поток сети, дополняя его псевдо-потоком из
        вспомогательной сети">
  <step description="Убновляем поток сети, дополняя его псевдо-потоком из вспомогательной
        сети"
    id="UpdateFlow"
    level="1"
    comment-ru="Поток в основной сети после проделанной фазы."
    comment-en="There is a flow in the main net after the current phase done."
  >
    <draw>
      d.vis.showNetwork(true);
    </draw>
    <direct>
      int i, j;
      for (i = 0; i &lt; d.n; i++)
        for (j = 0; j &lt; d.n; j++)
          stack.pushInteger(d.f[i][j]);

      for (i = 0; i &lt; d.n; i++)
        for (j = 0; j &lt; d.n; j++){
          // Прямая дуга
        }
    </direct>
    <reverse>
      int i, j;
      for (i = d.n - 1; i &gt;= 0; i--)
        for (j = d.n - 1; j &gt;= 0; j--)
          d.f[i][j] = stack.popInteger();
    </reverse>
  </step>
</auto>
<auto id="Main" description="Главный автомат" >
  <start
    comment-ru="Исходная сеть. Задача алгоритма - построить в данной сети максимальный
        поток."
    comment-en="There is an initial net. The task of the algorithm is to construct the
        maximum flow in this net."
  >
    <draw>
      d.vis.showNetwork(false);
    </draw>
  </start>
  <step description="Отображаем исходную сеть."
    id="ShowInitialNetwork"
    level="-1"
  >
    <draw>
      d.vis.showNetwork(false);
    </draw>
    <direct>
      int i, j;
      int n = d.n;
      d.t = d.n - 1;
      for (i = 0; i &lt; n; i++)
        for (j = 0; j &lt; n; j++) {
          d.f[i][j] = 0;
          d.dg[i][j] = 0;
          d.df[i][j] = 0;
          d.newArc[i][j] = false;
          d.newFlowArc[i][j] = false;
```

```
            }
            for (i = 0; i &lt; n; i++) {
              d.dq = new int[n];
              d.p[i] = 0;
              d.pin[i] = 0;
              d.pout[i] = 0;
              d.del[i] = false;
              d.dst[i] = 0;
              d.q1[i] = 0;
              d.q2[i] = 0;
              d.q[i] = 0;
              d.diniz[i] = false;
              d.fl[i] = 0;
              d.dq[i] = 0;
            }
            d.pr = 0;
            d.pw = 0;
            d.dpw = 0;
            d.dpr = 0;
          </direct>
          <reverse>
          </reverse>
        </step>
        <call-auto id="MakeNetwork"/>
        <while description="Главный цикл, выполняем пока можно посторить сеть Диница"
             test="d.isNetwork" level="-1">
        <call-auto id="GetPotentials"/>
        <call-auto id="MakeFlow"/>
        <call-auto id="UpdateFlow"/>
        <call-auto id="MakeNetwork"/>
      </while>
      <step description="В сети Диница нет дополняющего пути."
        id="NoPath"
        level="1"
        comment-ru="В сеть Диница не входит сток. А значит в остаточной сети нет ни одной
             дополняющей s-t цепи. "
        comment-en="Diniz net does not contain sink. So there are no augmentation s-t chains in
             (дополняющая) net."
      >
        <draw>
          d.vis.showDiniz(false);
        </draw>
        <direct>
        </direct>
        <reverse>
        </reverse>
      </step>
      <step description="Сеть и построенный в ней поток."
        id="ShowResultNetwork"
        level="-1"
      >
        <draw>
          d.vis.showNetwork(true);
        </draw>
        <direct>
          stack.pushInteger(d.resultFlow);
          d.resultFlow = 0;
          for (int i = 0; i &lt; d.n; i++)
            d.resultFlow += d.f[0][i];
        </direct>
        <reverse>
          d.resultFlow = stack.popInteger();
        </reverse>
      </step>
      <finish
        comment-ru="Работа алгоритма закончена. Максимальный поток в сети построен. Его величина
             равна {0}."
        comment-en="The algorithm has finished its work. Maximum flow in net has been
             constructing. Maximum flow value equals to {0}."
        comment-args="new Integer(d.resultFlow)"
      >
        <draw>
          d.vis.showNetwork(true);
        </draw>
      </finish>
    </auto>
</algorithm>
```

DMKM-Configuration.xml (описание конфигурации)

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

<!--
    <!DOCTYPE algorithm SYSTEM "scripts/configuration.dtd">
-->


<configuration>

    <button
        param       = "save-load-button"
        description = "'Save/Load' button"
        caption-ru  = "Сохранить/Загрузить"
        caption-en  = "Save/Load"
        hint-ru     = "Сохранить или загрузить граф"
        hint-en     = "Save current graph or load new one"
    />

    <button
        description = "Кнопка для сохранения сети"
        param       = "netDialog-saveButton"
        caption-ru  = "Сохранить"
        caption-en  = "Save"
        hint-ru     = "Сохранить сеть в память"
        hint-en     = "Save current net"
    />

    <button
        description = "Кнопка для загрузки сети"
        param       = "netDialog-loadButton"
        caption-ru  = "Загрузить"
        caption-en  = "Load"
        hint-ru     = "Заргузить сеть из памяти"
        hint-en     = "Load net"
    />

    <button
        description = "Кнопка для отмены"
        param       = "netDialog-cancelButton"
        caption-ru  = "Отмена"
        caption-en  = "Cancel"
        hint-ru     = "Отмена"
        hint-en     = "Cancel"
    />

    <message
        description = "Сохранить сеть"
        param       = "netDialog-save"
        message-ru  = "Сохранить сеть"
        message-en  = "Save net"
    />

    <message
        description = "Загрузить сеть"
        param       = "netDialog-load"
        message-ru  = "Загрузить сеть"
        message-en  = "Load net"
    />

    <message
        description = "Имя записи"
```

```
    param       = "netDialog-recordNameCaption"
    message-ru  = "Имя записи:"
    message-en  = "Record name:"
/>

<message
    description = "Список сохраненных сетей:"
    param       = "netDialog-listLabel"
    message-ru  = "Список сохраненных сетей:"
    message-en  = "List of saved nets:"
/>

<message
    description = "Надпись на истоке"
    param       = "source-caption"
    message-ru  = "s"
    message-en  = "s"
/>

<message
    description = "Надпись на стоке"
    param       = "sink-caption"
    message-ru  = "t"
    message-en  = "t"
/>

<button
    description = "Кнопка переключения в режим редактирования"
    param       = "multibutton-edit"
    caption-ru  = "Редактирование"
    caption-en  = "Edit"
    hint-ru     = "Перейти в режим редактирования сети"
    hint-en     = "Enter mode for editing net"
/>

<button
    description = "Кнопка переключения в режим визуализации"
    param       = "multibutton-vis"
    caption-ru  = "Визуализация"
    caption-en  = "Run"
    hint-ru     = "Перейти в режим визуализации алгоритма"
    hint-en     = "Start visualization of algorithm"
/>

<button
    description = "Кнопка сохранения состояния визуализации"
    param       = "save-button"
    caption-ru  = "Сохранить"
    caption-en  = "Save"
    hint-ru     = "Сохранить сеть в память"
    hint-en     = "Save the current net"
/>

<button
    description = "Кнопка загрузки состояния визуализации"
    param       = "load-button"
    caption-ru  = "Загрузить"
    caption-en  = "Load"
    hint-ru     = "Загрузить сеть из памяти"
    hint-en     = "Load net"
/>

<button
    description = "Кнопка для очистки сети"
```

```
    param       = "clear-button"
    caption-ru  = "Очистить"
    caption-en  = "Clear"
    hint-ru     = "Очистить сеть (удалить из нее все дуги)"
    hint-en     = "Clear the net (remove all arcs from net)"
/>

<button
    description = "Кнопка для генерации случайной сети"
    param       = "random-button"
    caption-ru  = "Случайно"
    caption-en  = "Random"
    hint-ru     = "Построить случайную сеть"
    hint-en     = "Generate random net"
/>

<spin-panel
    description = "Панель для выбора количества вершин в сети"
    param       = "nPanel"
    caption-ru  = "Вершин:{0, number, ####}"
    caption-en  = "Vertices:{0, number, ####}"
    hint-ru     = "Количесиво вершин в сети"
    hint-en     = "Number of vertices in net"
    value       = "6"
    min-value   = "2"
    max-value   = "10"
    step        = "1"
>
    <button
        description = "Кнопка для увеличения количества вершин"
        param       = "button-less"
        caption-ru  = "&lt;&lt;"
        caption-en  = "&lt;&lt;"
        hint-ru     = "Уменьшить количество вершин в сети"
        hint-en     = "Decrease number of vertices in net"
    />
    <button
        description = "Кнопка для уменьшения количества вершин"
        param       = "button-more"
        caption-ru  = ">>"
        caption-en  = ">>"
        hint-ru     = "Увеличить количество вершин в сети"
        hint-en     = "Increase number of vertices in net"
    />
</spin-panel>


<group
    description = "Меню редактора"
    param       = "menu"
>
    <property
        description = "Надпись увеличения пропускной способности дуги."
        param = "plus-message"
        value = "+"
    />
    <property
        description = "Надпись уменьшения пропускной способности дуги."
        param = "minus-message"
        value = "-"
    />
    <property
        description = "Надпись удаления дуги."
        param = "delete-message"
```

```
            value = "x"
        />
    </group>

    <styleset
        param = "edit-arc"
    >
        <style
            description = "Стиль изменяемой дуги"
            text-color     = "000000"
            text-align     = "0.5"
            aspect-status  = "false"
            padding        = "0.0"
            border-color   = "000000"
            border-status  = "false"
            fill-color     = "808080"
            fill-status    = "false"
        >
            <font
                face  = "Serif"
                size  = "15"
                style = "bold"
            />
        </style>
        <style
            description = "Стиль изменяемой дуги"
            border-color = "e0e0e0"
            fill-color   = "e0e0e0"
        >
        </style>
    </styleset>

    <group
        description = "Примеры для загрузки"
        param       = "example"
    >
        <property
            description = "Количество примеров"
            param       = "num"
            value       = "3"
        />
        <message
            description = "Описание первого примера"
            param      = "net1-name"
            message-ru = "Простейшая сеть (2 вершины, 1 дуга, 1 фаза)"
            message-en = "The simplest net (2 vertices, 1 arc, 1 phase)"
        />
        <property
            description = "Первый пример"
            param = "net1"
            value = "2   0 1 1"
        />
        <message
            description = "Описание второго примера"
            param      = "net2-name"
            message-ru = "Весьма содержательный пример (6 вершин, 8 дуг, 3
фазы)"
            message-en = "An interesting example (6 vertices, 8 arcs, 3
phases)"
        />
        <property
            description = "Второй пример"
            param = "net2"
```

```
            value = "6   0 1 2   0 3 2   0 5 1   1 2 1   1 4 2   2 5 2   3 2
2   4 5 2"
        />
        <message
            description = "Описание третьего примера"
            param       = "net3-name"
            message-ru  = "Особенность псевдомаксимального потока (6 вершин,
7 дуг, 2 фазы)"
            message-en  = "Pseudomax flow isn't max (6 vertices, 7 arcs, 2
phases)"
        />
        <property
            description = "Третий пример"
            param = "net3"
            value = "6   0 1 1   0 3 1   1 2 1   1 4 1   2 5 1   3 2 1   4 5
1"
        />
    </group>

    <property
        description = "Количество строк в комментариях"
        param       = "comment-lines"
        value       = "4"
    />

    <property
        description = "Максимальная пропускная способность дуги"
        param = "maxArcCapacity"
        value = "9"
    />

    <property
        description = "Ширина дуги"
        param = "arc-width-percent"
        value = "1.65"
    />

    <property
        description = "Длина дуги"
        param = "arc-length-percent"
        value = "5"
    />

    <property
        description = "Диаметр вершины"
        param = "vertex-diameter-percent"
        value = "13"
    />

    <property
        description = "Диаметр пропускной способности дуги"
        param = "circle-diameter-percent"
        value = "8"
    />

    <styleset
        param = "vertex"
    >
        <style
            description="Стиль вершины"
            aspect-status = "false"
            padding       = "0.0"
            text-align    = "0.5"
            text-color    = "000000"
```

```
        border-color  = "000000"
        border-status = "true"
        fill-color    = "f0f0f0"
        fill-status   = "true"
    >
        <font
            face  = "Serif"
            size  = "18"
            style = "bold"
        />
    </style>
    <style
        description="Стиль вершины"
        text-color   = "000000"
        border-color = "000000"
        fill-color   = "c0c0c0"
    />
    <style
        description="Стиль вершины"
        text-color   = "e0e0e0"
        border-color = "e0e0e0"
        fill-color   = "ffffff"
    >
    </style>
</styleset>

<styleset
    param = "arc"
>
    <style
        description="Стиль дуги"
        aspect-status = "false"
        padding       = "0.0"
        text-align    = "0.5"
        text-color    = "000000"
        border-color  = "707070"
        border-status = "false"
        fill-color    = "f0f0f0"
        fill-status   = "false"
    >
        <font
            face  = "Serif"
            size  = "15"
            style = "bold"
        />
    </style>
    <style
        description="Стиль дуги"
        border-color = "000000"
    >
    </style>
    <style
        description="Стиль дуги"
        border-color = "e0e0e0"
    >
    </style>
</styleset>

<styleset
    param = "circle"
>
    <style
        description="Стиль пропускной способности"
        aspect-status = "false"
```

```
        padding       = "0.0"
        text-align    = "0.5"
        text-color    = "000000"
        border-color  = "707070"
        border-status = "true"
        fill-color    = "f0f0f0"
        fill-status   = "true"
    >
        <font
            face  = "Serif"
            size  = "15"
            style = "bold"
        />
    </style>
    <style
        description="Стиль пропускной способности"
        border-color = "000000"
        fill-color   = "ffffff"
    >
    </style>
    <style
        description="Стиль пропускной способности"
        text-color   = "e0e0e0"
        border-color = "e0e0e0"
        fill-color   = "ffffff"
    >
    </style>
</styleset>

<styleset

    param  = "rect"
>
    <style
        description="Стиль потенциала"
        aspect-status = "false"
        padding       = "0.0"
        text-align    = "0.5"
        text-color    = "000000"
        border-color  = "000000"
        border-status = "true"
        fill-color    = "f0f0f0"
        fill-status   = "true"
    >
        <font
            face  = "Serif"
            size  = "13"
            style = "bold"
        />
    </style>
    <style
        description="Стиль потенциала"
        fill-color = "c0c0c0"
    >
    </style>
</styleset>

<styleset
    param = "menu-plus"
>
    <style
        description="Стиль +"
        aspect-status = "false"
        padding       = "0.0"
```

```
        text-align    = "0.5"
        text-color    = "000000"
        border-color  = "000000"
        border-status = "true"
        fill-color    = "f0f0f0"
        fill-status   = "true"
    >
        <font
            face  = "Monospaced"
            style = "bold"
            size  = "15"
        />
    </style>
</styleset>

<styleset

    param = "menu-minus"
>
    <style
        description="Стиль -"
        aspect-status  = "false"
        padding        = "0.0"
        text-align     = "0.5"
        text-color     = "000000"
        border-status  = "true"
        fill-status    = "true"
        border-color   = "000000"
        fill-color     = "f0f0f0"
    >
        <font
            face  = "Monospaced"
            style = "bold"
            size  = "15"
        />
    </style>
</styleset>

<styleset
    param = "menu-delete"
>
    <style
        description="Стиль x"
        aspect-status = "false"
        padding       = "0.0"
        text-align    = "0.5"
        text-color    = "000000"
        border-status = "true"
        fill-status   = "true"
        border-color  = "000000"
        fill-color    = "f0f0f0"
    >
        <font
            face  = "Monospaced"
            style = "bold"
            size  = "15"
        />
    </style>
</styleset>

<group
    description = "Настройки диалога сохранить / загрузить"
    param       = "SaveLoadDialog"
>
```

```
        <property
            description = "Высота области комментария в линиях"
            param       = "CommentPane-lines"
            value       = "2"
        />
        <property
            description = "Ширина области комментария"
            param       = "columns"
            value       = "40"
        />
        <property
            description = "Высота области комментария"
            param       = "rows"
            value       = "7"
        />
    </group>

    <message
        description = 'Предупреждение: две дуги между вершинами'
        param       = "TwoArcsExceptionMsg"
        message-ru  = "Строка {0}. Между любыми двумя вершиными сети не
должно быть более одной дуги"
        message-en  = "Line {0}. There can't be more than one arc between any
two vertices"
    />
    <message
        description = 'Комментарий параметра "ArcsComment"'
        param       = "ArcsComment"
        message-ru  = "Описание дуг сети (i j пропускная_способность)"
        message-en  = "Description of arcs in net (i j capacity)"
    />
    <message
        description = 'Комментарий параметра "NumberOfVertices"'
        param       = "NumberOfVerticesComment"
        message-ru  = "Количество вершин в сети (2 ... 10)"
        message-en  = "Number of vertices in net (2 ... 10)"
    />
    <message
        description = 'Комментарий параметра "NumberOfArcs"'
        param       = "NumberOfArcsComment"
        message-ru  = "Количество дуг в сети"
        message-en  = "Number of arcs in net"
    />
    <message
        description = 'Комментарий параметра "Step"'
        param       = "StepComment"
        message-ru  = "Номер шага"
        message-en  = "Current step"
    />

</configuration>
```

Приложение 2. Автоматически сгенерированный код, реализующий автоматы

DMKM.java

```
package ru.ifmo.vizi.dmkm;


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

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

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

    /**
      * Данные.
      */
    public final class Data {
        /**
          * Экземпляр апплета.
          */
        public DMKMVisualizer vis;

        /**
          * Исходная сеть.
          */
        public int g[][];

        /**
          * Вспомогательная сеть (сеть Диница).
          */
        public int dg[][];


        /**
          * Поток.
          */
        public int f[][];

        /**
          * Псевдо-поток.
          */
        public int df[][];

        /**
          * Потенциалы.
          */
        public int p[];

        /**
          * Входной потенциал.
          */
        public int pin[];

        /**
          * Выходной потенциал.
          */
        public int pout[];

        /**
          * Запоминаем удаленные вершины.
          */
        public boolean del[];

        /**
          * Количество вершин в сети.
          */
        public int n;

        /**
          * Исток.
          */
        public int s;

        /**
          * Сток.
          */
        public int t;
```

```
    /**
      * Вершина с минимальным потенциалом.
      */
    public int minv;

    /**
      * Можно ли построить сеть Диница.
      */
    public boolean isNetwork;

    /**
      * Есть ли неудаленные узлы.
      */
    public boolean isDel;

    /**
      * Дуги, поток на котторых ыыл обновлен во время forward(backward).
      */
    public boolean newArc[][];

    /**
      * Дуги, поток на которых был обновлен за фазу.
      */
    public boolean newFlowArc[][];

    /**
      * Впервые ли после построения сети Диница вызывается get_minv.
      */
    public boolean isFirstGetMinv;

    /**
      * maxint.
      */
    public final int maxint = Integer.MAX_VALUE;

    /**
      * Величина максимального потока.
      */
    public int resultFlow;

    /**
      * Минимальный потенциал.
      */
    public int minp;

    /**
      * Величнина псевдо-максимального потока.
      */
    public int pseudoResultFlow;

    /**
      * Вспомогательные переменные.
      */
    public int k, c1, c2, q1[], q2[], dst[], i, j, pr, pw, dpr, dpw, delta, fl[], q[],
           dq[];

    /**
      * Вспомогательные переменные.
      */
    public char sj, si;

    /**
      * Вспомогательные переменные.
      */
    public boolean isFirstDelete, diniz[];

    public String toString() {
        return "";
    }
}

/**
  * Строим сеть Диница.
  */
private final class MakeNetwork extends BaseAutomata implements Automata {
    /**
      * Начальное состояние автомата.
      */

    private final int START_STATE = 0;
```

```
/**
  * Конечное состояние автомата.
  */
private final int END_STATE = 21;

/**
  * Конструктор.
  */
public MakeNetwork() {
    super(
        "MakeNetwork",
        0, // Номер начального состояния
        21, // Номер конечного состояния
        new String[]{
            "Начальное состояние",
            "Инициализация переменных",
            "",
            "Выполняем цикл пока либо не достигнем стока, либо во втором фронте не
              окажется врешин.",
            "Инициализация",
            "Проход по фронту",
            "Проход по фронту.",
            "",
            "",
            "Проеряем добавилось ли новая дуга",
            "Проеряем добавилось ли новая дуга (окончание)",
            "Случай, когда дуга добавлена",
            "",
            "",
            "Проверяем построили ли мы сеть Диница",
            "IfDinizNetworkExist",
            "IfDinizNetworkExist (окончание)",
            "LastLayerBuilt",
            "Строим сеть Диница",
            "Определяем псевдомаксимальный поток",
            "Запоминаем сеть Диница",
            "Конечное состояние"
        }, new int[]{
            Integer.MAX_VALUE, // Начальное состояние,
            1, // Инициализация переменных
            0, //
            -1, // Выполняем цикл пока либо не достигнем стока, либо во втором
                      фронте не окажется врешин.
            -1, // Инициализация
            -1, // Проход по фронту
            -1, // Проход по фронту.
            -1, //
            -1, //
            -1, // Проеряем добавилось ли новая дуга
            -1, // Проеряем добавилось ли новая дуга (окончание)
            0, // Случай, когда дуга добавлена
            -1, //
            -1, //
            -1, // Проверяем построили ли мы сеть Диница
            -1, // IfDinizNetworkExist
            -1, // IfDinizNetworkExist (окончание)
            0, // LastLayerBuilt
            1, // Строим сеть Диница
            1, // Определяем псевдомаксимальный поток
            -1, // Запоминаем сеть Диница
            Integer.MAX_VALUE, // Конечное состояние
        }
    );
}

/**
  * Сделать один шаг автомата в перед.
  */
protected void doStepForward(int level) {
    // Переход в следующее состояние
    switch (state) {
        case START_STATE: { // Начальное состояние
            state = 1; // Инициализация переменных
            break;
        }
        case 1: { // Инициализация переменных
            state = 2;
            break;
        }
```

```
case 2: {
    stack.pushBoolean(false);
    state = 3; // Выполняем цикл пока либо не достигнем стока, либо во
      втором фронте не окажется врешин.
    break;
}
case 3: { // Выполняем цикл пока либо не достигнем стока, либо во втором
      фронте не окажется врешин.
    if ((d.dst[d.t] == d.maxint) && (d.c2 > 0)) {
        state = 4; // Инициализация
    } else {
        state = 14; // Проверяем построили ли мы сеть Диница
    }

    break;
}
case 4: { // Инициализация
    stack.pushBoolean(false);
    state = 5; // Проход по фронту
    break;
}
case 5: { // Проход по фронту
    if (d.k < d.c1) {
        state = 6; // Проход по фронту.
    } else {
        stack.pushBoolean(true);
        state = 3; // Выполняем цикл пока либо не достигнем стока, либо во
      втором фронте не окажется врешин.
    }
    break;
}
case 6: { // Проход по фронту.
    stack.pushBoolean(false);
    state = 7;
    break;
}
case 7: {
    if (d.j < d.n) {
        state = 8;
    } else {
        state = 13;
    }
    break;
}
case 8: {
    state = 9; // Проеряем добавилось ли новая дуга
    break;
}
case 9: { // Проеряем добавилось ли новая дуга
    if (d.dg[d.i][d.j] > 0) {
        state = 11; // Случай, когда дуга добавлена
    } else {
        stack.pushBoolean(false);
        state = 10; // Проеряем добавилось ли новая дуга (окончание)
    }
    break;
}
case 10: { // Проеряем добавилось ли новая дуга (окончание)
    state = 12;
    break;
}
case 11: { // Случай, когда дуга добавлена
    stack.pushBoolean(true);
    state = 10; // Проеряем добавилось ли новая дуга (окончание)
    break;
}
case 12: {
    stack.pushBoolean(true);
    state = 7;
    break;
}
case 13: {
    stack.pushBoolean(true);
    state = 5; // Проход по фронту
    break;
}
case 14: { // Проверяем построили ли мы сеть Диница
    state = 15; // IfDinizNetworkExist
    break;
}
```

```
    case 15: { // IfDinizNetworkExist
        if (d.isNetwork) {
            state = 17; // LastLayerBuilt
        } else {
            stack.pushBoolean(false);
            state = 16; // IfDinizNetworkExist (окончание)
        }
        break;
    }
    case 16: { // IfDinizNetworkExist (окончание)
        state = 20; // Запоминаем сеть Диница
        break;
    }
    case 17: { // LastLayerBuilt
        state = 18; // Строим сеть Диница
        break;
    }
    case 18: { // Строим сеть Диница
        state = 19; // Определяем псевдомаксимальный поток
        break;
    }
    case 19: { // Определяем псевдомаксимальный поток
        stack.pushBoolean(true);
        state = 16; // IfDinizNetworkExist (окончание)
        break;
    }
    case 20: { // Запоминаем сеть Диница
        state = END_STATE;
        break;
    }
}

// Действие в текущем состоянии
switch (state) {
    case 1: { // Инициализация переменных
        int i, j;
        stack.pushInteger(d.c1);
        stack.pushInteger(d.c2);
        for (i = 0; i < d.n; i++)
            stack.pushInteger(d.q1[i]);
        for (i = 0; i < d.n; i++)
            stack.pushInteger(d.q2[i]);
        for (i = 0; i < d.n; i++)
            stack.pushInteger(d.dst[i]);
        for (i = 0; i < d.n; i++)
            for (j = 0; j < d.n; j++)
                stack.pushInteger(d.dg[i][j]);
        for (i = 0; i < d.n; i++)
            for (j = 0; j < d.n; j++)
                stack.pushInteger(d.df[i][j]);
        for (i = 0; i < d.n; i++)
            stack.pushBoolean(d.del[i]);

        d.c1 = 0;
        d.c2 = 1;
        d.q2[0] = d.s;
        for (i = 0; i < d.n; i++) d.dst[i] = d.maxint;
        d.dst[d.s] = 0;
        for (i = 0; i < d.n; i++)
            for (j = 0; j < d.n; j++)
                d.dg[i][j] = d.df[i][j] = 0;
        for (i = 0; i < d.n; i++)
            d.del[i] = true;
        break;
    }
    case 2: {
        d.del[0] = false;
        break;
    }
    case 3: { // Выполняем цикл пока либо не достигнем стока, либо во втором
          фронте не окажется врешин.
        break;
    }
    case 4: { // Инициализация
        int i;
        for (i = 0; i < d.n; i++)
            stack.pushInteger(d.q1[i]);
        stack.pushInteger(d.c1);
        stack.pushInteger(d.c2);
        stack.pushInteger(d.k);
```

```
    for (i = 0; i < d.n; i++) d.q1[i] = d.q2[i];
    d.c1 = d.c2;
    d.c2 = 0;
    d.k = 0;
    break;
}
case 5: { // Проход по фронту
    break;
}
case 6: { // Проход по фронту.
    stack.pushInteger(d.i);
    stack.pushInteger(d.j);

    d.i = d.q1[d.k];
    d.j = 0;
    break;
}
case 7: {
    break;
}
case 8: {
    int i, j;
    stack.pushInteger(d.c2);
    for (i = 0; i < d.n; i++)
        for (j = 0; j < d.n; j++)
            stack.pushBoolean(d.newFlowArc[i][j]);
    for (i = 0; i < d.n; i++)
        for (j = 0; j < d.n; j++)
            stack.pushInteger(d.dg[i][j]);
    for (i = 0; i < d.n; i++)
        for (j = 0; j < d.n; j++)
            stack.pushInteger(d.df[i][j]);
    for (i = 0; i < d.n; i++)
        stack.pushInteger(d.dst[i]);
    for (i = 0; i < d.n; i++)
        stack.pushInteger(d.q2[i]);

    for (i = 0; i < d.n; i++)
        for (j = 0; j < d.n; j++)
            d.newFlowArc[i][j] = false;
    if ( (d.dst[d.i] < d.dst[d.j]) && (d.g[d.i][d.j] > d.f[d.i][d.j]) ){
        if (d.dst[d.j] == d.maxint) d.q2[d.c2++] = d.j;
        d.dst[d.j] = d.dst[d.i] + 1;
        d.dg[d.i][d.j] = d.g[d.i][d.j] - d.f[d.i][d.j];
    }
    if ( (d.dst[d.i] < d.dst[d.j]) && (d.f[d.j][d.i] > 0) ){
        if (d.dst[d.j] == d.maxint) d.q2[d.c2++] = d.j;
        d.dst[d.j] = d.dst[d.i] + 1;
        d.dg[d.i][d.j] = d.f[d.j][d.i];
    }
    if (d.dg[d.i][d.j] > 0) d.newFlowArc[d.i][d.j] = true;
    break;
}
case 9: { // Проеряем добавилось ли новая дуга
    break;
}
case 10: { // Проеряем добавилось ли новая дуга (окончание)
    break;
}

case 11: { // Случай, когда дуга добавлена
    stack.pushCharacter(d.si);
    stack.pushCharacter(d.sj);
    stack.pushBoolean(d.del[d.i]);
    stack.pushBoolean(d.del[d.j]);

    if (d.i == 0) d.si = 's';
    else if (d.i == d.n - 1) d.si = 't';
    else d.si = (char)(d.i + 48);
    if (d.j == 0) d.sj = 's';
    else if (d.j == d.n - 1) d.sj = 't';
    else d.sj = (char)(d.j + 48);

    d.del[d.i] = false;
    d.del[d.j] = false;
    break;
}
case 12: {
    d.j++;
```

```
            break;
        }
        case 13: {
            d.k++;
            break;
        }
        case 14: { // Проверяем построили ли мы сеть Диница
            stack.pushBoolean(d.isNetwork);

            d.isNetwork = (d.dst[d.t] != d.maxint);
            break;
        }
        case 15: { // IfDinizNetworkExist
            break;
        }
        case 16: { // IfDinizNetworkExist (окончание)
            break;
        }
        case 17: { // LastLayerBuilt
            int i, j;
            for (i = 0; i < d.n; i++)
                for (j = 0; j < d.n; j++)
                    stack.pushBoolean(d.newFlowArc[i][j]);

            for (i = 0; i < d.n; i++)
                for (j = 0; j < d.n; j++)
                    d.newFlowArc[i][j] = false;
            break;
        }
        case 18: { // Строим сеть Диница
            break;
        }
        case 19: { // Определяем псевдомаксимальный поток
            break;
        }
        case 20: { // Запоминаем сеть Диница
            int i;
            for (i = 0; i < d.n; i++)
                stack.pushBoolean(d.diniz[i]);
            for (i = 0; i < d.n; i++)
                d.diniz[i] = d.del[i];
            break;
        }
    }
}

/**
  * Сделать один шаг автомата назад.
  */
protected void doStepBackward(int level) {
    // Обращение действия в текущем состоянии
    switch (state) {
        case 1: { // Инициализация переменных
            int i, j;
            for (i = d.n - 1; i >= 0; i--)
                d.del[i] = stack.popBoolean();
            for (i = d.n - 1; i >= 0; i--)
                for (j = d.n - 1; j >= 0; j--)
                    d.df[i][j] = stack.popInteger();
            for (i = d.n - 1; i >= 0; i--)

                for (j = d.n - 1; j >= 0; j--)
                    d.dg[i][j] = stack.popInteger();
            for (i = d.n - 1; i >= 0; i--)
                d.dst[i] = stack.popInteger();
            for (i = d.n - 1; i >= 0; i--)
                d.q2[i] = stack.popInteger();
            for (i = d.n - 1; i >= 0; i--)
                d.q1[i] = stack.popInteger();
            d.c2 = stack.popInteger();
            d.c1 = stack.popInteger();
            break;
        }
        case 2: {
            d.del[0] = true;
            break;
        }
        case 3: { // Выполняем цикл пока либо не достигнем стока, либо во втором
              фронте не окажется врешин.
            break;
```

```
}
case 4: { // Инициализация
    int i;
    d.k = stack.popInteger();
    d.c2 = stack.popInteger();
    d.c1 = stack.popInteger();
    for (i = d.n - 1; i >= 0; i--)
        d.q1[i] = stack.popInteger();
    break;
}
case 5: { // Проход по фронту
    break;
}
case 6: { // Проход по фронту.
    d.j = stack.popInteger();
    d.i = stack.popInteger();
    break;
}
case 7: {
    break;
}
case 8: {
    int i, j;
    for (i = d.n - 1; i >= 0; i--)
        d.q2[i] = stack.popInteger();
    for (i = d.n - 1; i >= 0; i--)
        d.dst[i] = stack.popInteger();
    for (i = d.n - 1; i >= 0; i--)
        for (j = d.n - 1; j >= 0; j--)
            d.df[i][j] = stack.popInteger();
    for (i = d.n - 1; i >= 0; i--)
        for (j = d.n - 1; j >= 0; j--)
            d.dg[i][j] = stack.popInteger();
    for (i = d.n - 1; i >= 0; i--)
        for (j = d.n - 1; j >= 0; j--)
            d.newFlowArc[i][j] = stack.popBoolean();
    d.c2 = stack.popInteger();
    break;
}
case 9: { // Проеряем добавилось ли новая дуга
    break;
}
case 10: { // Проеряем добавилось ли новая дуга (окончание)
    break;
}
case 11: { // Случай, когда дуга добавлена
    d.del[d.j] = stack.popBoolean();
    d.del[d.i] = stack.popBoolean();

    d.sj = stack.popCharacter();
    d.si = stack.popCharacter();
    break;
}
case 12: {
    d.j--;
    break;
}
case 13: {
    d.k--;
    break;
}
case 14: { // Проверяем построили ли мы сеть Диница
    d.isNetwork = stack.popBoolean();
    break;
}
case 15: { // IfDinizNetworkExist
    break;
}
case 16: { // IfDinizNetworkExist (окончание)
    break;
}
case 17: { // LastLayerBuilt
    int i, j;
    for (i = d.n - 1; i >= 0; i--)
        for (j = d.n - 1; j >= 0; j--)
            d.newFlowArc[i][j] = stack.popBoolean();
    break;
}
case 18: { // Строим сеть Диница
    break;
```

```
    }
    case 19: { // Определяем псевдомаксимальный поток
        break;
    }
    case 20: { // Запоминаем сеть Диница
        for (int i = d.n - 1; i >= 0; i--)
            d.diniz[i] = stack.popBoolean();
        break;
    }
}

// Переход в предыдущее состояние
switch (state) {
    case 1: { // Инициализация переменных
        state = START_STATE;
        break;
    }
    case 2: {
        state = 1; // Инициализация переменных
        break;
    }
    case 3: { // Выполняем цикл пока либо не достигнем стока, либо во втором
          фронте не окажется врешин.
        if (stack.popBoolean()) {
            state = 5; // Проход по фронту
        } else {
            state = 2;
        }
        break;
    }
    case 4: { // Инициализация
        state = 3; // Выполняем цикл пока либо не достигнем стока, либо во
          втором фронте не окажется врешин.
        break;
    }
    case 5: { // Проход по фронту
        if (stack.popBoolean()) {
            state = 13;
        } else {
            state = 4; // Инициализация
        }
        break;
    }
    case 6: { // Проход по фронту.
        state = 5; // Проход по фронту
        break;
    }
    case 7: {
        if (stack.popBoolean()) {
            state = 12;
        } else {
            state = 6; // Проход по фронту.
        }
        break;
    }
    case 8: {
        state = 7;
        break;
    }
    case 9: { // Проеряем добавилось ли новая дуга
        state = 8;
        break;
    }
    case 10: { // Проеряем добавилось ли новая дуга (окончание)
        if (stack.popBoolean()) {
            state = 11; // Случай, когда дуга добавлена
        } else {
            state = 9; // Проеряем добавилось ли новая дуга
        }
        break;
    }
    case 11: { // Случай, когда дуга добавлена
        state = 9; // Проеряем добавилось ли новая дуга
        break;
    }
    case 12: {
        state = 10; // Проеряем добавилось ли новая дуга (окончание)

        break;
    }
```

```
        case 13: {
            state = 7;
            break;
        }
        case 14: { // Проверяем построили ли мы сеть Диница
            state = 3; // Выполняем цикл пока либо не достигнем стока, либо во
              втором фронте не окажется врешин.
            break;
        }
        case 15: { // IfDinizNetworkExist
            state = 14; // Проверяем построили ли мы сеть Диница
            break;
        }
        case 16: { // IfDinizNetworkExist (окончание)
            if (stack.popBoolean()) {
                state = 19; // Определяем псевдомаксимальный поток
            } else {
                state = 15; // IfDinizNetworkExist
            }
            break;
        }
        case 17: { // LastLayerBuilt
            state = 15; // IfDinizNetworkExist
            break;
        }
        case 18: { // Строим сеть Диница
            state = 17; // LastLayerBuilt
            break;
        }
        case 19: { // Определяем псевдомаксимальный поток
            state = 18; // Строим сеть Диница
            break;
        }
        case 20: { // Запоминаем сеть Диница
            state = 16; // IfDinizNetworkExist (окончание)
            break;
        }
        case END_STATE: { // Начальное состояние
            state = 20; // Запоминаем сеть Диница
            break;
        }
    }
}

/**
  * Комментарий к текущему состоянию
  */
public String getComment() {
    String comment = "";
    Object[] args = null;
    // Выбор комментария
    switch (state) {
        case 1: { // Инициализация переменных
            comment = DMKM.this.getComment("MakeNetwork.init");
            break;
        }
        case 2: {
            comment = DMKM.this.getComment("MakeNetwork.Layer");
            break;
        }
        case 11: { // Случай, когда дуга добавлена
            comment = DMKM.this.getComment("MakeNetwork.ArcExist");
            args = new Object[]{new Character(d.si), new Character(d.sj), new
       Integer(d.dst[d.i]), new Integer(d.dst[d.j]), new Integer(d.n - 1)};
            break;
        }
        case 17: { // LastLayerBuilt
            comment = DMKM.this.getComment("MakeNetwork.LastLayerBuilt");
            break;
        }
        case 18: { // Строим сеть Диница
            comment = DMKM.this.getComment("MakeNetwork.MakeNetworkBigStep");
            args = new Object[]{new Integer(d.dst[d.t])};
            break;
        }
        case 19: { // Определяем псевдомаксимальный поток
            comment = DMKM.this.getComment("MakeNetwork.StartPseudomax");
            break;
        }
    }
```

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

        /**
          * Выполняет действия по отрисовке состояния
          */
        public void drawState() {
            switch (state) {
                case 1: { // Инициализация переменных
                    d.vis.showDiniz(false);
                    break;
                }
                case 2: {
                    d.vis.showDiniz(false);
                    break;
                }
                case 11: { // Случай, когда дуга добавлена
                    d.vis.showDiniz(false);
                    break;
                }
                case 17: { // LastLayerBuilt
                    d.vis.showDiniz(false);
                    break;
                }
                case 18: { // Строим сеть Диница
                    d.vis.showDiniz(false);
                    break;
                }
                case 19: { // Определяем псевдомаксимальный поток
                    d.vis.showDiniz(false);
                    break;
                }
            }
        }
    }

    /**
      * Вычисляем потенциалы в узлах вспомогательной сети.
      */
    private final class GetPotentials extends BaseAutomata implements Automata {
        /**
          * Начальное состояние автомата.
          */
        private final int START_STATE = 0;

        /**
          * Конечное состояние автомата.
          */
        private final int END_STATE = 2;

        /**
          * Конструктор.
          */
        public GetPotentials() {
            super(
                "GetPotentials",
                0, // Номер начального состояния
                2, // Номер конечного состояния
                new String[]{
                    "Начальное состояние",
                    "Вычисляем потенциалы в узлах вспомогательной сети.",
                    "Конечное состояние"
                }, new int[]{
                    Integer.MAX_VALUE, // Начальное состояние,
                    1, // Вычисляем потенциалы в узлах вспомогательной сети.
                    Integer.MAX_VALUE, // Конечное состояние
                }
            );
        }

        /**
          * Сделать один шаг автомата в перед.
          */
        protected void doStepForward(int level) {
            // Переход в следующее состояние
            switch (state) {
                case START_STATE: { // Начальное состояние
                    state = 1; // Вычисляем потенциалы в узлах вспомогательной сети.
```

```
            break;
        }
        case 1: { // Вычисляем потенциалы в узлах вспомогательной сети.
            state = END_STATE;
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // Вычисляем потенциалы в узлах вспомогательной сети.
            int i;
            for (i = 0; i < d.n; i++) stack.pushInteger(d.p[i]);
            for (i = 0; i < d.n; i++) stack.pushInteger(d.pin[i]);
            for (i = 0; i < d.n; i++) stack.pushInteger(d.pout[i]);
            for (i = 0; i < d.n; i++) stack.pushInteger(d.dq[i]);
            stack.pushInteger(d.dpw);

            int j;
            for (i = 0; i < d.n; i++) d.p[i] = d.pin[i] = d.pout[i] = 0;
            for (i = 0; i < d.n; i++)
                for (j = 0; j < d.n; j++)
                    if (d.dg[i][j] > 0){
                        d.pin[j] += d.dg[i][j];
                        d.pout[i] += d.dg[i][j];
                    }
            d.pin[d.s] = d.pout[d.t] = d.maxint;
            for (i = 0; i < d.n; i++) {
                if (d.pin[i] < d.pout[i]) d.p[i] = d.pin[i]; else d.p[i] =
              d.pout[i];
                if (d.p[i] == 0 && !d.del[i]) d.dq[d.dpw++] = i;
            }
            break;
        }
    }
}

/**
  * Сделать один шаг автомата назад.
  */
protected void doStepBackward(int level) {
    // Обращение действия в текущем состоянии
    switch (state) {
        case 1: { // Вычисляем потенциалы в узлах вспомогательной сети.
            int i;
            d.dpw = stack.popInteger();
            for (i = d.n - 1; i >= 0; i--) d.dq[i] = stack.popInteger();
            for (i = d.n - 1; i >= 0; i--) d.pout[i] = stack.popInteger();
            for (i = d.n - 1; i >= 0; i--) d.pin[i] = stack.popInteger();
            for (i = d.n - 1; i >= 0; i--) d.p[i] = stack.popInteger();
            break;
        }
    }

    // Переход в предыдущее состояние
    switch (state) {
        case 1: { // Вычисляем потенциалы в узлах вспомогательной сети.
            state = START_STATE;
            break;
        }
        case END_STATE: { // Начальное состояние
            state = 1; // Вычисляем потенциалы в узлах вспомогательной сети.
            break;
        }
    }
}

/**
  * Комментарий к текущему состоянию
  */
public String getComment() {
    String comment = "";
    Object[] args = null;
    // Выбор комментария
    switch (state) {
        case 1: { // Вычисляем потенциалы в узлах вспомогательной сети.
            comment = DMKM.this.getComment("GetPotentials.GetPotentials");
            break;
        }
    }
```

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

    /**
      * Выполняет действия по отрисовке состояния
      */
    public void drawState() {
        switch (state) {
            case 1: { // Вычисляем потенциалы в узлах вспомогательной сети.
                d.vis.showPotentials();
                break;
            }
        }
    }
}

/**
  * Удаляем узлы с нулевым потенциалом.
  */
private final class DeleteZero extends BaseAutomata implements Automata {
    /**
      * Начальное состояние автомата.
      */
    private final int START_STATE = 0;

    /**
      * Конечное состояние автомата.
      */
    private final int END_STATE = 7;

    /**
      * Конструктор.
      */
    public DeleteZero() {
        super(
            "DeleteZero",
            0, // Номер начального состояния
            7, // Номер конечного состояния
            new String[]{
                "Начальное состояние",
                "Удаляем узлы с нулевыми потенциалами.",
                "",
                "",
                "",
                "",
                "",
                "Конечное состояние"
            }, new int[]{
                Integer.MAX_VALUE, // Начальное состояние,
                -1, // Удаляем узлы с нулевыми потенциалами.
                -1, //
                0, //
                -1, //
                0, //
                -1, //
                Integer.MAX_VALUE, // Конечное состояние
            }
        );
    }

    /**
      * Сделать один шаг автомата в перед.
      */
    protected void doStepForward(int level) {
        // Переход в следующее состояние
        switch (state) {
            case START_STATE: { // Начальное состояние
                state = 1; // Удаляем узлы с нулевыми потенциалами.
                break;
            }
            case 1: { // Удаляем узлы с нулевыми потенциалами.
                stack.pushBoolean(false);
                state = 2;
                break;
            }
            case 2: {
                if (d.dpr < d.dpw) {
                    state = 3;
                } else {
```

```
            state = 6;
        }
        break;
    }
    case 3: {
        state = 4;
        break;
    }
    case 4: {
        state = 5;
        break;
    }
    case 5: {
        stack.pushBoolean(true);
        state = 2;
        break;
    }
    case 6: {
        state = END_STATE;
        break;
    }
}

// Действие в текущем состоянии
switch (state) {
    case 1: { // Удаляем узлы с нулевыми потенциалами.
        stack.pushInteger(d.dpr);

        d.dpr = 0;
        break;
    }
    case 2: {
        break;
    }
    case 3: {
        stack.pushCharacter(d.si);
        stack.pushInteger(d.i);
        stack.pushInteger(d.j);
        for (int i = 0; i < d.n; i++)
            stack.pushBoolean(d.del[i]);

        d.i = d.dq[d.dpr++];

        if (d.i == 0) d.si = 's';
        else if (d.i == d.n - 1) d.si = 't';
        else d.si = (char)(d.i + 48);

        d.del[d.i] = true;
        d.j = 0;
        break;
    }
    case 4: {
        int i;
        stack.pushInteger(d.dpw);
        for (i = 0; i < d.n; i++) stack.pushInteger(d.dq[i]);
        for (i = 0; i < d.n; i++) stack.pushInteger(d.p[i]);
        for (i = 0; i < d.n; i++) stack.pushInteger(d.pin[i]);
        for (i = 0; i < d.n; i++) stack.pushInteger(d.pout[i]);

        int min;
        for (int j = 0; j < d.n; j++) if (d.p[j] != 0){
            if (d.dg[d.i][j] > 0){
                d.pout[d.i] -= (d.dg[d.i][j] - d.df[d.i][j]);
                d.pin[j] -= (d.dg[d.i][j] - d.df[d.i][j]);
            }
            if (d.dg[j][d.i] > 0){
                d.pout[j] -= (d.dg[j][d.i] - d.df[j][d.i]);
                d.pin[d.i] -= (d.dg[j][d.i] - d.df[j][d.i]);
            }
            if (d.pin[d.i] < d.pout[d.i]) min = d.pin[d.i]; else min =
                d.pout[d.i];
            if (d.p[d.i] > min) d.p[d.i] = min;
            if (d.pin[j] < d.pout[j]) min = d.pin[j]; else min = d.pout[j];
            if (d.p[j] > min) d.p[j] = min;
            if (d.p[j] == 0) d.dq[d.dpw++] = j;
        }
        break;
    }
    case 5: {
        break;
```

```
        }
        case 6: {
            stack.pushBoolean(d.isDel);
            stack.pushInteger(d.dpw);
            d.isDel = true;
            for (int i = 0; i < d.n; i++)
                if (!d.del[i]) d.isDel = false;
            d.dpw = 0;
            break;
        }
    }
}

/**
  * Сделать один шаг автомата назад.
  */
protected void doStepBackward(int level) {
    // Обращение действия в текущем состоянии
    switch (state) {
        case 1: { // Удаляем узлы с нулевыми потенциалами.
            d.dpr = stack.popInteger();
            break;
        }
        case 2: {
            break;
        }
        case 3: {
            d.dpr--;
                 for (int i = d.n - 1; i >= 0; i--)
                     d.del[i] = stack.popBoolean();
                 d.j = stack.popInteger();
                 d.i = stack.popInteger();
                 d.si = stack.popCharacter();
            break;
        }
        case 4: {
            int i;
            for (i = d.n - 1; i >= 0; i--) d.pout[i] = stack.popInteger();
            for (i = d.n - 1; i >= 0; i--) d.pin[i] = stack.popInteger();
            for (i = d.n - 1; i >= 0; i--) d.p[i] = stack.popInteger();
            for (i = d.n - 1; i >= 0; i--) d.dq[i] = stack.popInteger();
            d.dpw = stack.popInteger();
            break;
        }
        case 5: {
            break;
        }
        case 6: {
            d.dpw = stack.popInteger();
            d.isDel = stack.popBoolean();
            break;
        }
    }

    // Переход в предыдущее состояние
    switch (state) {
        case 1: { // Удаляем узлы с нулевыми потенциалами.
            state = START_STATE;
            break;
        }
        case 2: {
            if (stack.popBoolean()) {
                state = 5;
            } else {
                state = 1; // Удаляем узлы с нулевыми потенциалами.
            }
            break;
        }
        case 3: {
            state = 2;
            break;
        }
        case 4: {
            state = 3;
            break;
        }
        case 5: {
            state = 4;
            break;
        }
```

```
            case 6: {
                state = 2;
                break;
            }
            case END_STATE: { // Начальное состояние
                state = 6;
                break;
            }
        }
    }

    /**
      * Комментарий к текущему состоянию
      */
    public String getComment() {
        String comment = "";
        Object[] args = null;
        // Выбор комментария
        switch (state) {
            case 3: {
                comment = DMKM.this.getComment("DeleteZero.DeleteStep");
                args = new Object[]{new Character(d.si)};
                break;
            }
            case 5: {
                comment = DMKM.this.getComment("DeleteZero.UpdatePotentials");
                args = new Object[]{new Integer(d.i)};
                break;
            }
        }

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

    /**
      * Выполняет действия по отрисовке состояния
      */
    public void drawState() {
        switch (state) {
            case 3: {
                d.vis.showDelete(!d.isFirstDelete);
                break;
            }
            case 5: {
                d.vis.showDelete(!d.isFirstDelete);
                break;
            }
        }
    }
}

/**
  * Находим узел с минимальным потенциалом.
  */
private final class GetMinv extends BaseAutomata implements Automata {
    /**
      * Начальное состояние автомата.
      */
    private final int START_STATE = 0;

    /**
      * Конечное состояние автомата.
      */
    private final int END_STATE = 3;

    /**
      * Конструктор.
      */
    public GetMinv() {
        super(
            "GetMinv",
            0, // Номер начального состояния
            3, // Номер конечного состояния
            new String[]{
                "Начальное состояние",
                "Находим узел с минимальным потенциалом.",
                "Отмечаем что мы уже искали вершину с минимальным потенциалом",
                "Конечное состояние"
            }, new int[]{
                Integer.MAX_VALUE, // Начальное состояние,
```

```
            1, // Находим узел с минимальным потенциалом.
            -1, // Отмечаем что мы уже искали вершину с минимальным потенциалом
            Integer.MAX_VALUE, // Конечное состояние
        }
    );
}

/**
  * Сделать один шаг автомата в перед.
  */
protected void doStepForward(int level) {
    // Переход в следующее состояние
    switch (state) {
        case START_STATE: { // Начальное состояние
            state = 1; // Находим узел с минимальным потенциалом.
            break;
        }
        case 1: { // Находим узел с минимальным потенциалом.
            state = 2; // Отмечаем что мы уже искали вершину с минимальным
              потенциалом
            break;
        }
        case 2: { // Отмечаем что мы уже искали вершину с минимальным потенциалом
            state = END_STATE;
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // Находим узел с минимальным потенциалом.
            stack.pushCharacter(d.sj);
            stack.pushInteger(d.minv);
            stack.pushInteger(d.minp);

            if (d.minv == 0) d.sj = 's';
            else if (d.minv == d.n - 1) d.sj = 't';
            else d.sj = (char)(d.minv + 48);

            d.minv = d.s;
            for (int i = 0; i < d.n; i++)
                if ( (d.p[i] != 0) && (d.p[d.minv] > d.p[i]) ) d.minv = i;
            d.minp = d.p[d.minv];
            break;
        }
        case 2: { // Отмечаем что мы уже искали вершину с минимальным потенциалом
            stack.pushBoolean(d.isFirstGetMinv);
            d.isFirstGetMinv = false;
            break;
        }
    }
}

/**
  * Сделать один шаг автомата назад.
  */
protected void doStepBackward(int level) {
    // Обращение действия в текущем состоянии
    switch (state) {
        case 1: { // Находим узел с минимальным потенциалом.
            d.minp = stack.popInteger();
            d.minv = stack.popInteger();

            d.sj = stack.popCharacter();
            break;
        }
        case 2: { // Отмечаем что мы уже искали вершину с минимальным потенциалом
            d.isFirstGetMinv = stack.popBoolean();
            break;
        }
    }

    // Переход в предыдущее состояние
    switch (state) {
        case 1: { // Находим узел с минимальным потенциалом.
            state = START_STATE;
            break;
        }
        case 2: { // Отмечаем что мы уже искали вершину с минимальным потенциалом
            state = 1; // Находим узел с минимальным потенциалом.
```

```
                break;
            }
            case END_STATE: { // Начальное состояние
                state = 2; // Отмечаем что мы уже искали вершину с минимальным
                           потенциалом
                break;
            }
        }
    }

    /**
      * Комментарий к текущему состоянию
      */
    public String getComment() {
        String comment = "";
        Object[] args = null;
        // Выбор комментария
        switch (state) {
            case 1: { // Находим узел с минимальным потенциалом.
                comment = DMKM.this.getComment("GetMinv.GetMinv");
                args = new Object[]{new Character(d.sj), new Integer(d.p[d.minv])};
                break;
            }
        }

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

    /**
      * Выполняет действия по отрисовке состояния
      */
    public void drawState() {
        switch (state) {
            case 1: { // Находим узел с минимальным потенциалом.
                d.vis.showMinv(!d.isFirstGetMinv);
                break;
            }
        }
    }
}

/**
  * Проталкиваем поток из этого узла в сток.
  */
private final class Forward extends BaseAutomata implements Automata {
    /**
      * Начальное состояние автомата.
      */
    private final int START_STATE = 0;

    /**
      * Конечное состояние автомата.
      */
    private final int END_STATE = 15;

    /**
      * Конструктор.
      */
    public Forward() {
        super(
            "Forward",
            0, // Номер начального состояния
            15, // Номер конечного состояния
            new String[]{
                "Начальное состояние",
                "Инициализация",
                "",
                "",
                "",
                "",
                "",
                " (окончание)",
                "",
                "",
                " (окончание)",
                "",
                "",
                "",
                "Проталкиваем поток из этого узла в сток",
                "Конечное состояние"
```

```
        }, new int[]{
            Integer.MAX_VALUE, // Начальное состояние,
            1, // Инициализация
            -1, //
            -1, //
            -1, //
            -1, //
            -1, //
            -1, //  (окончание)
            -1, //
            -1, //
            -1, //  (окончание)
            0, //
            -1, //
            -1, //
            1, // Проталкиваем поток из этого узла в сток
            Integer.MAX_VALUE, // Конечное состояние
        }
    );
}

/**
  * Сделать один шаг автомата в перед.
  */
protected void doStepForward(int level) {
    // Переход в следующее состояние
    switch (state) {
        case START_STATE: { // Начальное состояние
            state = 1; // Инициализация
            break;
        }
        case 1: { // Инициализация
            stack.pushBoolean(false);
            state = 2;
            break;
        }
        case 2: {
            if (d.pr < d.pw) {
                state = 3;
            } else {
                state = 13;
            }
            break;
        }
        case 3: {
            state = 4;
            break;
        }
        case 4: {
            stack.pushBoolean(false);
            state = 5;
            break;
        }
        case 5: {
            if (d.j < d.n) {
                state = 6;
            } else {
                stack.pushBoolean(true);
                state = 2;
            }
            break;
        }
        case 6: {
            if (d.p[d.j] > 0) {
                state = 8;
            } else {
                stack.pushBoolean(false);
                state = 7; //  (окончание)
            }
            break;
        }
        case 7: { //  (окончание)
            state = 12;
            break;
        }
        case 8: {
            state = 9;
            break;
        }
        case 9: {
```

```
            if (d.delta > 0) {
                state = 11;
            } else {
                stack.pushBoolean(false);
                state = 10; //  (окончание)
            }
            break;
        }
        case 10: { //  (окончание)
            stack.pushBoolean(true);
            state = 7; //  (окончание)
            break;
        }
        case 11: {
            stack.pushBoolean(true);
            state = 10; //  (окончание)
            break;
        }
        case 12: {
            stack.pushBoolean(true);
            state = 5;
            break;
        }
        case 13: {
            state = 14; // Проталкиваем поток из этого узла в сток
            break;
        }
        case 14: { // Проталкиваем поток из этого узла в сток
            state = END_STATE;
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // Инициализация
            stack.pushCharacter(d.sj);
            int i, j;
            for (i = 0; i < d.n; i++)
                for (j = 0; j < d.n; j++)
                    stack.pushBoolean(d.newArc[i][j]);
            for (i = 0; i < d.n; i++)
                stack.pushInteger(d.fl[i]);
            stack.pushInteger(d.pr);
            stack.pushInteger(d.pw);
            stack.pushInteger(d.q[0]);

            if (d.minv == 0) d.sj = 's';
            else if (d.minv == d.n - 1) d.sj= 't';
            else d.sj = (char)(d.minv + 48);

            for (i = 0; i < d.n; i++)
                d.fl[i] = 0;
            for (i = 0; i < d.n; i++)
                for (j = 0; j < d.n; j++)
                    d.newArc[i][j] = false;
            d.fl[d.minv] = d.p[d.minv];
            d.pr = 0; d.pw = 1; d.q[0] = d.minv;
            break;
        }
        case 2: {
            break;
        }
        case 3: {
            stack.pushInteger(d.i);
            d.i = d.q[d.pr++];
            break;
        }
        case 4: {
            stack.pushInteger(d.j);
            d.j = 0;
            break;
        }
        case 5: {
            break;
        }
        case 6: {
            break;
        }
        case 7: { //  (окончание)
```

```
            break;
        }
        case 8: {
            stack.pushInteger(d.delta);
            if (d.fl[d.i] < d.dg[d.i][d.j] - d.df[d.i][d.j]) d.delta = d.fl[d.i];
                    else d.delta = d.dg[d.i][d.j] - d.df[d.i][d.j];
            break;
        }
        case 9: {
            break;
        }
        case 10: { //  (окончание)
            break;
        }
        case 11: {
            stack.pushCharacter(d.si);
            stack.pushCharacter(d.sj);
            stack.pushBoolean(d.newArc[d.i][d.j]);
            stack.pushInteger(d.q[d.pw]);
            stack.pushInteger(d.p[d.j]);
            stack.pushInteger(d.p[d.i]);
            stack.pushInteger(d.f[d.j][d.i]);
            stack.pushInteger(d.f[d.i][d.j]);
            stack.pushInteger(d.dpw);
            for (int i = 0; i < d.n; i++)
                stack.pushInteger(d.dq[i]);

            if (d.i == 0) d.si = 's';
            else if (d.i == d.n - 1) d.si = 't';
            else d.si = (char)(d.i + 48);
            if (d.j == 0) d.sj = 's';
            else if (d.j == d.n - 1) d.sj = 't';
            else d.sj = (char)(d.j + 48);

            d.pin[d.j] -= d.delta;
            d.pout[d.i] -= d.delta;
            if (d.p[d.i] > d.pout[d.i]) {
                if (d.pout[d.i] == 0) d.dq[d.dpw++] = d.i;
                d.p[d.i] = d.pout[d.i];
            }
            if (d.p[d.j] > d.pin[d.j]) {
                if (d.pin[d.j] == 0) d.dq[d.dpw++] = d.j;
                d.p[d.j] = d.pin[d.j];
            }
            d.df[d.i][d.j] += d.delta;
            d.newArc[d.i][d.j] = true;
            d.fl[d.i] -= d.delta;
            d.fl[d.j] += d.delta;
            d.q[d.pw++] = d.j;

            if (d.g[d.i][d.j] > 0) d.f[d.i][d.j] += d.delta;
            else if (d.g[d.j][d.i] > 0) d.f[d.j][d.i] -= d.delta;
            break;
        }
        case 12: {
            d.j++;
            break;
        }
        case 13: {
            int i, j;
            for (i = 0; i < d.n; i++)
                for (j = 0; j < d.n; j++)
                    stack.pushBoolean(d.newFlowArc[i][j]);

            for (i = 0; i < d.n; i++)
                for (j = 0; j < d.n; j++)
                    d.newFlowArc[i][j] = d.newFlowArc[i][j] | d.newArc[i][j];
            break;
        }
        case 14: { // Проталкиваем поток из этого узла в сток
            break;
        }
    }
}

/**
  * Сделать один шаг автомата назад.
  */
protected void doStepBackward(int level) {
    // Обращение действия в текущем состоянии
```

```
switch (state) {
    case 1: { // Инициализация
        int i, j;
        d.q[0] = stack.popInteger();
        d.pw = stack.popInteger();
        d.pr = stack.popInteger();
        for (i = d.n - 1; i >= 0; i--)
            d.fl[i] = stack.popInteger();
        for (i = d.n - 1; i >= 0; i--)
            for (j = d.n - 1; j >= 0; j--)
                d.newArc[i][j] = stack.popBoolean();
        d.sj = stack.popCharacter();
        break;
    }
    case 2: {
        break;
    }
    case 3: {
        d.pr--;
        d.i = stack.popInteger();
        break;
    }
    case 4: {
        d.j = stack.popInteger();
        break;
    }
    case 5: {
        break;
    }
    case 6: {
        break;
    }
    case 7: { //  (окончание)
        break;
    }
    case 8: {
        d.delta = stack.popInteger();
        break;
    }
    case 9: {
        break;
    }
    case 10: { //  (окончание)
        break;
    }
    case 11: {
        for (int i = d.n - 1; i >= 0; i--)
            d.dq[i] = stack.popInteger();
        d.dpw = stack.popInteger();
        d.f[d.i][d.j] = stack.popInteger();
        d.f[d.j][d.i] = stack.popInteger();
        d.p[d.i] = stack.popInteger();
        d.p[d.j] = stack.popInteger();
        d.pin[d.j] += d.delta;
        d.pout[d.i] += d.delta;
        d.df[d.i][d.j] -= d.delta;
        d.fl[d.i] += d.delta;
        d.fl[d.j] -= d.delta;
        d.pw--;
        d.q[d.pw] = stack.popInteger();
        d.newArc[d.i][d.j] = stack.popBoolean();
        d.sj = stack.popCharacter();
        d.si = stack.popCharacter();
        break;
    }
    case 12: {
        d.j--;
        break;
    }
    case 13: {
        int i, j;
        for (i = d.n - 1; i >= 0; i--)
            for (j = d.n - 1; j >= 0; j--)
                d.newFlowArc[i][j] = stack.popBoolean();
        break;
    }
    case 14: { // Проталкиваем поток из этого узла в сток
        break;
    }
}
```

```
// Переход в предыдущее состояние
switch (state) {
    case 1: { // Инициализация
        state = START_STATE;
        break;
    }
    case 2: {
        if (stack.popBoolean()) {
            state = 5;
        } else {
            state = 1; // Инициализация
        }
        break;
    }
    case 3: {
        state = 2;
        break;
    }
    case 4: {
        state = 3;
        break;
    }
    case 5: {
        if (stack.popBoolean()) {
            state = 12;
        } else {
            state = 4;
        }
        break;
    }
    case 6: {
        state = 5;
        break;
    }
    case 7: { //  (окончание)
        if (stack.popBoolean()) {
            state = 10; //  (окончание)
        } else {
            state = 6;
        }
        break;
    }
    case 8: {
        state = 6;
        break;
    }
    case 9: {
        state = 8;
        break;
    }
    case 10: { //  (окончание)
        if (stack.popBoolean()) {
            state = 11;
        } else {
            state = 9;
        }
        break;
    }
    case 11: {
        state = 9;
        break;
    }
    case 12: {
        state = 7; //  (окончание)
        break;
    }
    case 13: {
        state = 2;
        break;
    }
    case 14: { // Проталкиваем поток из этого узла в сток
        state = 13;
        break;
    }
    case END_STATE: { // Начальное состояние
        state = 14; // Проталкиваем поток из этого узла в сток
        break;
    }
}
```

```
    }

    /**
      * Комментарий к текущему состоянию
      */
    public String getComment() {
        String comment = "";
        Object[] args = null;
        // Выбор комментария
        switch (state) {
            case 1: { // Инициализация
                comment = DMKM.this.getComment("Forward.forward");
                args = new Object[]{new Integer(d.minp), new Character(d.sj)};
                break;
            }
            case 11: {
                comment = DMKM.this.getComment("Forward.throw");
                args = new Object[]{new Integer(d.delta), new Character(d.si), new
                  Character(d.sj)};
                break;
            }
            case 14: { // Проталкиваем поток из этого узла в сток
                comment = DMKM.this.getComment("Forward.EndForward");
                args = new Object[]{new Integer(d.minp)};
                break;
            }
        }

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

    /**
      * Выполняет действия по отрисовке состояния
      */
    public void drawState() {
        switch (state) {
            case 1: { // Инициализация
                d.vis.showForward();
                break;
            }
            case 11: {
                d.vis.showForward();
                break;
            }
            case 14: { // Проталкиваем поток из этого узла в сток
                d.vis.showForward();
                break;
            }
        }
    }
}

/**
  * Проталкиваем поток из этого узла в исток.
  */
private final class Backward extends BaseAutomata implements Automata {
    /**
      * Начальное состояние автомата.
      */
    private final int START_STATE = 0;

    /**
      * Конечное состояние автомата.
      */
    private final int END_STATE = 15;

    /**
      * Конструктор.
      */
    public Backward() {
        super(
            "Backward",
            0, // Номер начального состояния
            15, // Номер конечного состояния
            new String[]{
                "Начальное состояние",
                "Инициализация",
                "",
                "",
                "",
```

```
            "",
            "",
            " (окончание)",
            "",
            "",
            " (окончание)",
            "",
            "",
            "",
            "Проталкиваем поток из этого узла в сток",
            "Конечное состояние"
        }, new int[]{
            Integer.MAX_VALUE, // Начальное состояние,
            1, // Инициализация
            -1, //
            -1, //
            -1, //
            -1, //
            -1, //
            -1, //  (окончание)
            -1, //
            -1, //
            -1, //  (окончание)
            0, //
            -1, //
            -1, //
            1, // Проталкиваем поток из этого узла в сток
            Integer.MAX_VALUE, // Конечное состояние
        }
    );
}

/**
  * Сделать один шаг автомата в перед.
  */
protected void doStepForward(int level) {
    // Переход в следующее состояние
    switch (state) {
        case START_STATE: { // Начальное состояние
            state = 1; // Инициализация
            break;
        }
        case 1: { // Инициализация
            stack.pushBoolean(false);
            state = 2;
            break;
        }
        case 2: {
            if (d.pr < d.pw) {
                state = 3;
            } else {
                state = 13;
            }
            break;
        }
        case 3: {
            state = 4;
            break;
        }
        case 4: {
            stack.pushBoolean(false);
            state = 5;
            break;
        }
        case 5: {
            if (d.j < d.n) {
                state = 6;
            } else {
                stack.pushBoolean(true);
                state = 2;
            }
            break;
        }
        case 6: {
            if (d.p[d.j] > 0) {
                state = 8;
            } else {
                stack.pushBoolean(false);
                state = 7; //  (окончание)
            }
```

```
            break;
        }
        case 7: { //  (окончание)
            state = 12;
            break;
        }
        case 8: {
            state = 9;
            break;
        }
        case 9: {
            if (d.delta > 0) {
                state = 11;
            } else {
                stack.pushBoolean(false);
                state = 10; //  (окончание)

            }
            break;
        }
        case 10: { //  (окончание)
            stack.pushBoolean(true);
            state = 7; //  (окончание)
            break;
        }

        case 11: {
            stack.pushBoolean(true);
            state = 10; //  (окончание)
            break;
        }
        case 12: {
            stack.pushBoolean(true);
            state = 5;
            break;
        }
        case 13: {
            state = 14; // Проталкиваем поток из этого узла в сток
            break;
        }
        case 14: { // Проталкиваем поток из этого узла в сток
            state = END_STATE;
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // Инициализация
            stack.pushCharacter(d.si);
            int i, j;
            for (i = 0; i < d.n; i++)
                for (j = 0; j < d.n; j++)
                    stack.pushBoolean(d.newArc[i][j]);
            for (i = 0; i < d.n; i++)
                stack.pushInteger(d.fl[i]);
            stack.pushInteger(d.pr);
            stack.pushInteger(d.pw);
            stack.pushInteger(d.q[0]);

            if (d.minv == 0) d.si = 's';
            else if (d.minv == d.n - 1) d.si = 't';
            else d.si = (char)(d.minv + 48);

            for (i = 0; i < d.n; i++)
                d.fl[i] = 0;
            for (i = 0; i < d.n; i++)
                for (j = 0; j < d.n; j++)
                    d.newArc[i][j] = false;
            d.fl[d.minv] = d.minp;
            d.pr = 0; d.pw = 1; d.q[0] = d.minv;
            break;
        }
        case 2: {
            break;
        }
        case 3: {
            stack.pushInteger(d.i);
            d.i = d.q[d.pr++];
            break;
```

```
}
case 4: {
    stack.pushInteger(d.j);
    d.j = 0;
    break;
}
case 5: {
    break;
}
case 6: {
    break;
}
case 7: { //  (окончание)
    break;
}
case 8: {
    stack.pushInteger(d.delta);
    if (d.fl[d.i] < d.dg[d.j][d.i] - d.df[d.j][d.i]) d.delta = d.fl[d.i];
              else d.delta = d.dg[d.j][d.i] - d.df[d.j][d.i];
    break;
}
case 9: {
    break;
}
case 10: { //  (окончание)
    break;
}
case 11: {
    stack.pushCharacter(d.si);
    stack.pushCharacter(d.sj);
    stack.pushBoolean(d.newArc[d.j][d.i]);
    stack.pushInteger(d.q[d.pw]);
    stack.pushInteger(d.p[d.i]);
    stack.pushInteger(d.p[d.j]);
    stack.pushInteger(d.df[d.i][d.j]);
    stack.pushInteger(d.df[d.j][d.i]);
    stack.pushInteger(d.dpw);
    for (int i = 0; i < d.n; i++)
        stack.pushInteger(d.dq[i]);

    if (d.i == 0) d.si = 's';
    else if (d.i == d.n - 1) d.si = 't';
    else d.si = (char)(d.i + 48);
    if (d.j == 0) d.sj = 's';
    else if (d.j == d.n - 1) d.sj = 't';
    else d.sj = (char)(d.j + 48);

    d.pin[d.i] -= d.delta;
    d.pout[d.j] -= d.delta;
    if (d.p[d.i] > d.pin[d.i]) {
        if (d.pin[d.i] == 0) d.dq[d.dpw++] = d.i;
        d.p[d.i] = d.pin[d.i];
    }
    if (d.p[d.j] > d.pout[d.j]) {
        if (d.pout[d.j] == 0) d.dq[d.dpw++] = d.j;
        d.p[d.j] = d.pout[d.j];
    }
    d.df[d.j][d.i] += d.delta;
    d.newArc[d.j][d.i] = true;
    d.fl[d.i] -= d.delta;
    d.fl[d.j] += d.delta;
    d.q[d.pw++] = d.j;

    if (d.g[d.j][d.i] > 0) d.f[d.j][d.i] += d.delta;
    else if (d.g[d.i][d.j] > 0) d.f[d.i][d.j] -= d.delta;
    break;
}
case 12: {
    d.j++;
    break;
}
case 13: {
    int i, j;
    for (i = 0; i < d.n; i++)
        for (j = 0; j < d.n; j++)
            stack.pushBoolean(d.newFlowArc[i][j]);

    for (i = 0; i < d.n; i++)
        for (j = 0; j < d.n; j++)
            d.newFlowArc[i][j] = d.newFlowArc[i][j] | d.newArc[i][j];
```

```
            break;
        }
        case 14: { // Проталкиваем поток из этого узла в сток
            break;
        }
    }
}

/**
  * Сделать один шаг автомата назад.
  */
protected void doStepBackward(int level) {
    // Обращение действия в текущем состоянии
    switch (state) {
        case 1: { // Инициализация
            int i, j;
            d.q[0] = stack.popInteger();
            d.pw = stack.popInteger();
            d.pr = stack.popInteger();
            for (i = d.n - 1; i >= 0; i--)
                d.fl[i] = stack.popInteger();
            for (i = d.n - 1; i >= 0; i--)
                for (j = d.n - 1; j >= 0; j--)
                    d.newArc[i][j] = stack.popBoolean();
            d.si = stack.popCharacter();
            break;
        }
        case 2: {
            break;
        }
        case 3: {
            d.pr--;
            d.i = stack.popInteger();
            break;
        }
        case 4: {
            d.j = stack.popInteger();
            break;
        }
        case 5: {
            break;
        }
        case 6: {
            break;
        }
        case 7: { //  (окончание)
            break;
        }
        case 8: {
            d.delta = stack.popInteger();
            break;
        }
        case 9: {
            break;
        }
        case 10: { //  (окончание)
            break;
        }
        case 11: {
            for (int i = d.n - 1; i >= 0; i--)
                d.dq[i] = stack.popInteger();
            d.dpw = stack.popInteger();
            d.f[d.j][d.i] = stack.popInteger();
            d.f[d.i][d.j] = stack.popInteger();
            d.p[d.j] = stack.popInteger();
            d.p[d.i] = stack.popInteger();
            d.pin[d.i] += d.delta;
            d.pout[d.j] += d.delta;
            d.df[d.j][d.i] -= d.delta;
            d.fl[d.i] += d.delta;
            d.fl[d.j] -= d.delta;
            d.pw--;
            d.q[d.pw] = stack.popInteger();
            d.newArc[d.j][d.i] = stack.popBoolean();
            d.sj = stack.popCharacter();
            d.si = stack.popCharacter();
            break;
        }
        case 12: {
            d.j--;
```

```
            break;
        }
        case 13: {
            int i, j;
            for (i = d.n - 1; i >= 0; i--)
                for (j = d.n - 1; j >= 0; j--)
                    d.newFlowArc[i][j] = stack.popBoolean();
            break;
        }
        case 14: { // Проталкиваем поток из этого узла в сток
            break;
        }
    }

    // Переход в предыдущее состояние
    switch (state) {

        case 1: { // Инициализация
            state = START_STATE;
            break;
        }
        case 2: {
            if (stack.popBoolean()) {
                state = 5;
            } else {
                state = 1; // Инициализация
            }
            break;
        }
        case 3: {
            state = 2;
            break;
        }
        case 4: {
            state = 3;
            break;
        }
        case 5: {
            if (stack.popBoolean()) {
                state = 12;
            } else {
                state = 4;
            }
            break;
        }
        case 6: {
            state = 5;
            break;
        }
        case 7: { //  (окончание)
            if (stack.popBoolean()) {
                state = 10; //  (окончание)
            } else {
                state = 6;
            }
            break;
        }
        case 8: {
            state = 6;
            break;
        }
        case 9: {
            state = 8;
            break;
        }
        case 10: { //  (окончание)
            if (stack.popBoolean()) {
                state = 11;
            } else {
                state = 9;
            }
            break;
        }
        case 11: {
            state = 9;
            break;
        }
        case 12: {
            state = 7; //  (окончание)
            break;
```

```
            }
            case 13: {
                state = 2;
                break;
            }
            case 14: { // Проталкиваем поток из этого узла в сток
                state = 13;
                break;
            }
            case END_STATE: { // Начальное состояние
                state = 14; // Проталкиваем поток из этого узла в сток
                break;
            }
        }
    }

    /**
      * Комментарий к текущему состоянию
      */
    public String getComment() {
        String comment = "";
        Object[] args = null;
        // Выбор комментария
        switch (state) {
            case 1: { // Инициализация
                comment = DMKM.this.getComment("Backward.BackwardInit");
                args = new Object[]{new Integer(d.minp), new Character(d.si)};
                break;
            }
            case 11: {
                comment = DMKM.this.getComment("Backward.throw");
                args = new Object[]{new Integer(d.delta), new Character(d.si), new
                  Character(d.sj)};
                break;
            }
            case 14: { // Проталкиваем поток из этого узла в сток
                comment = DMKM.this.getComment("Backward.forward");
                args = new Object[]{new Integer(d.minp)};
                break;
            }
        }

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

    /**
      * Выполняет действия по отрисовке состояния
      */
    public void drawState() {
        switch (state) {
            case 1: { // Инициализация
                d.vis.showBackward();
                break;
            }
            case 11: {
                d.vis.showForward();
                break;
            }
            case 14: { // Проталкиваем поток из этого узла в сток
                d.vis.showBackward();
                break;
            }
        }
    }
}

/**
  * Строим псевдомаксимальный поток во вспомогательной сети.
  */
private final class MakeFlow extends BaseAutomata implements Automata {
    /**
      * Начальное состояние автомата.
      */
    private final int START_STATE = 0;

    /**
      * Конечное состояние автомата.
      */
    private final int END_STATE = 14;
```

```
/**
  * Конструктор.
  */
public MakeFlow() {
    super(
        "MakeFlow",
        0, // Номер начального состояния
        14, // Номер конечного состояния
        new String[]{
            "Начальное состояние",
            "Изначально нет удаленных узлов",
            "Удаляем узлы с нулевым потенциалом. После построения сети Диница.",
            "Удаляем узлы с нулевым потенциалом (автомат)",
            "",
            "Строим псевдомаксимальный поток в сети Диница, пока не все узлы
                    удалены",
            "Находим узел с минимальным потенциалом (автомат)",
            "Проталкиваем поток из этого узла в сток (автомат)",
            "Проталкиваем поток из этого узла в исток (автомат)",
            "delete_zero_after_flow",
            "Удаляем узлы с нулевым потенциалом (автомат)",
            "",
            "",
            "Псевдомаксимальный поток в сети Диница построен.",
            "Конечное состояние"
        }, new int[]{
            Integer.MAX_VALUE, // Начальное состояние,
            -1, // Изначально нет удаленных узлов
            1, // Удаляем узлы с нулевым потенциалом. После построения сети
              Диница.
            0, // Удаляем узлы с нулевым потенциалом (автомат)
            1, //
            -1, // Строим псевдомаксимальный поток в сети Диница, пока не все узлы
              удалены
            0, // Находим узел с минимальным потенциалом (автомат)
            0, // Проталкиваем поток из этого узла в сток (автомат)
            0, // Проталкиваем поток из этого узла в исток (автомат)
            1, // delete_zero_after_flow
            0, // Удаляем узлы с нулевым потенциалом (автомат)
            1, //
            0, //
            1, // Псевдомаксимальный поток в сети Диница построен.
            Integer.MAX_VALUE, // Конечное состояние
        }
    );
}

/**
  * Сделать один шаг автомата в перед.
  */
protected void doStepForward(int level) {
    // Переход в следующее состояние
    switch (state) {
        case START_STATE: { // Начальное состояние
            state = 1; // Изначально нет удаленных узлов
            break;
        }
        case 1: { // Изначально нет удаленных узлов
            state = 2; // Удаляем узлы с нулевым потенциалом. После построения
              сети Диница.
            break;
        }
        case 2: { // Удаляем узлы с нулевым потенциалом. После построения сети
              Диница.
            state = 3; // Удаляем узлы с нулевым потенциалом (автомат)
            break;
        }
        case 3: { // Удаляем узлы с нулевым потенциалом (автомат)
            if (child.isAtEnd()) {
                child = null;
                state = 4;
            }
            break;
        }
        case 4: {
            stack.pushBoolean(false);
            state = 5; // Строим псевдомаксимальный поток в сети Диница, пока не
              все узлы удалены
            break;
        }
```

```
    case 5: { // Строим псевдомаксимальный поток в сети Диница, пока не все
          узлы удалены
        if (!d.isDel) {
            state = 6; // Находим узел с минимальным потенциалом (автомат)
        } else {
            state = 12;
        }
        break;
    }
    case 6: { // Находим узел с минимальным потенциалом (автомат)
        if (child.isAtEnd()) {
            child = null;
            state = 7; // Проталкиваем поток из этого узла в сток (автомат)
        }

        break;
    }
    case 7: { // Проталкиваем поток из этого узла в сток (автомат)
        if (child.isAtEnd()) {
            child = null;
            state = 8; // Проталкиваем поток из этого узла в исток (автомат)
        }
        break;
    }
    case 8: { // Проталкиваем поток из этого узла в исток (автомат)
        if (child.isAtEnd()) {
            child = null;
            state = 9; // delete_zero_after_flow
        }
        break;
    }
    case 9: { // delete_zero_after_flow
        state = 10; // Удаляем узлы с нулевым потенциалом (автомат)
        break;
    }
    case 10: { // Удаляем узлы с нулевым потенциалом (автомат)
        if (child.isAtEnd()) {
            child = null;
            state = 11;
        }
        break;
    }
    case 11: {
        stack.pushBoolean(true);
        state = 5; // Строим псевдомаксимальный поток в сети Диница, пока не
          все узлы удалены
        break;
    }
    case 12: {
        state = 13; // Псевдомаксимальный поток в сети Диница построен.
        break;
    }
    case 13: { // Псевдомаксимальный поток в сети Диница построен.
        state = END_STATE;
        break;
    }
}

// Действие в текущем состоянии
switch (state) {
    case 1: { // Изначально нет удаленных узлов
        int i, j;
        stack.pushBoolean(d.isDel);
        stack.pushBoolean(d.isFirstGetMinv);
        for (i = 0; i < d.n; i++)
            for (j = 0; j < d.n; j++)
                stack.pushBoolean(d.newFlowArc[i][j]);

        for (i = 0; i < d.n; i++)
            for (j = 0; j < d.n; j++)
                d.newFlowArc[i][j] = false;
        d.isDel = false;
        d.isFirstGetMinv = true;
        break;
    }
    case 2: { // Удаляем узлы с нулевым потенциалом. После построения сети
          Диница.
        stack.pushBoolean(d.isFirstDelete);
        d.isFirstDelete = true;
        break;
```

```
            }
            case 3: { // Удаляем узлы с нулевым потенциалом (автомат)
                if (child == null) {
                    child = new DeleteZero();
                    child.toStart();
                }
                child.stepForward(level);
                step--;
                break;
            }
            case 4: {
                break;
            }
            case 5: { // Строим псевдомаксимальный поток в сети Диница, пока не все
                  узлы удалены
                break;
            }
            case 6: { // Находим узел с минимальным потенциалом (автомат)
                if (child == null) {
                    child = new GetMinv();
                    child.toStart();
                }
                child.stepForward(level);
                step--;

                break;
            }
            case 7: { // Проталкиваем поток из этого узла в сток (автомат)
                if (child == null) {
                    child = new Forward();
                    child.toStart();
                }
                child.stepForward(level);
                step--;
                break;
            }
            case 8: { // Проталкиваем поток из этого узла в исток (автомат)
                if (child == null) {
                    child = new Backward();
                    child.toStart();
                }
                child.stepForward(level);
                step--;
                break;
            }
            case 9: { // delete_zero_after_flow
                stack.pushBoolean(d.isFirstDelete);
                d.isFirstDelete = false;
                break;
            }
            case 10: { // Удаляем узлы с нулевым потенциалом (автомат)
                if (child == null) {
                    child = new DeleteZero();
                    child.toStart();
                }
                child.stepForward(level);
                step--;
                break;
            }
            case 11: {
                break;
            }
            case 12: {
                break;
            }
            case 13: { // Псевдомаксимальный поток в сети Диница построен.
                int i;
                for (i = 0; i < d.n; i++)
                    stack.pushBoolean(d.del[i]);
                stack.pushInteger(d.pseudoResultFlow);

                for (i = 0; i < d.n; i++)
                    d.del[i] = d.diniz[i];
                d.pseudoResultFlow = 0;
                for (i = 0; i < d.n; i++)
                    d.pseudoResultFlow += d.df[0][i];
                break;
            }
        }
    }
```

```
/**
  * Сделать один шаг автомата назад.
  */
protected void doStepBackward(int level) {
    // Обращение действия в текущем состоянии
    switch (state) {
        case 1: { // Изначально нет удаленных узлов
            int i, j;
            for (i = d.n - 1; i >= 0; i--)
                for (j = d.n - 1; j >= 0; j--)
                    d.newFlowArc[i][j] = stack.popBoolean();
            d.isFirstGetMinv = stack.popBoolean();
            d.isDel = stack.popBoolean();
            break;
        }
        case 2: { // Удаляем узлы с нулевым потенциалом. После построения сети
              Диница.
            d.isFirstDelete = stack.popBoolean();
            break;
        }
        case 3: { // Удаляем узлы с нулевым потенциалом (автомат)
            if (child == null) {
                child = new DeleteZero();
                child.toEnd();
            }
            child.stepBackward(level);
            step++;
            break;
        }
        case 4: {

            break;
        }
        case 5: { // Строим псевдомаксимальный поток в сети Диница, пока не все
              узлы удалены
            break;
        }
        case 6: { // Находим узел с минимальным потенциалом (автомат)
            if (child == null) {
                child = new GetMinv();
                child.toEnd();
            }
            child.stepBackward(level);
            step++;
            break;
        }
        case 7: { // Проталкиваем поток из этого узла в сток (автомат)
            if (child == null) {
                child = new Forward();
                child.toEnd();

            }
            child.stepBackward(level);
            step++;
            break;
        }
        case 8: { // Проталкиваем поток из этого узла в исток (автомат)
            if (child == null) {
                child = new Backward();
                child.toEnd();
            }
            child.stepBackward(level);
            step++;
            break;
        }

        case 9: { // delete_zero_after_flow
            d.isFirstDelete = stack.popBoolean();
            break;
        }
        case 10: { // Удаляем узлы с нулевым потенциалом (автомат)
            if (child == null) {
                child = new DeleteZero();
                child.toEnd();
            }
            child.stepBackward(level);
            step++;
            break;
        }
```

```
    case 11: {
        break;
    }
    case 12: {
        break;
    }
    case 13: { // Псевдомаксимальный поток в сети Диница построен.
        d.pseudoResultFlow = stack.popInteger();
        for (int i = d.n - 1; i >= 0; i--)
            d.del[i] = stack.popBoolean();
        break;
    }
}

// Переход в предыдущее состояние
switch (state) {
    case 1: { // Изначально нет удаленных узлов
        state = START_STATE;
        break;
    }
    case 2: { // Удаляем узлы с нулевым потенциалом. После построения сети
          Диница.
        state = 1; // Изначально нет удаленных узлов
        break;
    }
    case 3: { // Удаляем узлы с нулевым потенциалом (автомат)
        if (child.isAtStart()) {
            child = null;
            state = 2; // Удаляем узлы с нулевым потенциалом. После построения
                  сети Диница.
        }
        break;
    }
    case 4: {
        state = 3; // Удаляем узлы с нулевым потенциалом (автомат)
        break;
    }
    case 5: { // Строим псевдомаксимальный поток в сети Диница, пока не все
          узлы удалены
        if (stack.popBoolean()) {
            state = 11;
        } else {
            state = 4;
        }
        break;
    }
    case 6: { // Находим узел с минимальным потенциалом (автомат)
        if (child.isAtStart()) {
            child = null;
            state = 5; // Строим псевдомаксимальный поток в сети Диница, пока
                  не все узлы удалены
        }
        break;
    }
    case 7: { // Проталкиваем поток из этого узла в сток (автомат)
        if (child.isAtStart()) {
            child = null;
            state = 6; // Находим узел с минимальным потенциалом (автомат)
        }
        break;
    }
    case 8: { // Проталкиваем поток из этого узла в исток (автомат)
        if (child.isAtStart()) {
            child = null;
            state = 7; // Проталкиваем поток из этого узла в сток (автомат)
        }
        break;
    }
    case 9: { // delete_zero_after_flow
        state = 8; // Проталкиваем поток из этого узла в исток (автомат)
        break;
    }
    case 10: { // Удаляем узлы с нулевым потенциалом (автомат)
        if (child.isAtStart()) {
            child = null;
            state = 9; // delete_zero_after_flow
        }
        break;
    }
    case 11: {
```

```
            state = 10; // Удаляем узлы с нулевым потенциалом (автомат)
            break;
        }
        case 12: {
            state = 5; // Строим псевдомаксимальный поток в сети Диница, пока не
              все узлы удалены
            break;
        }
        case 13: { // Псевдомаксимальный поток в сети Диница построен.
            state = 12;
            break;
        }
        case END_STATE: { // Начальное состояние
            state = 13; // Псевдомаксимальный поток в сети Диница построен.
            break;
        }
    }
}

/**
  * Комментарий к текущему состоянию
  */
public String getComment() {
    String comment = "";
    Object[] args = null;
    // Выбор комментария
    switch (state) {
        case 2: { // Удаляем узлы с нулевым потенциалом. После построения сети
              Диница.
            comment = DMKM.this.getComment("MakeFlow.FirstDeleteZero");
            break;
        }
        case 3: { // Удаляем узлы с нулевым потенциалом (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 4: {
            comment = DMKM.this.getComment("MakeFlow.FinishFirstDeleteZero");
            break;
        }
        case 6: { // Находим узел с минимальным потенциалом (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 7: { // Проталкиваем поток из этого узла в сток (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 8: { // Проталкиваем поток из этого узла в исток (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 9: { // delete_zero_after_flow
            comment = DMKM.this.getComment("MakeFlow.DeleteZeroAfterFlow");
            break;
        }
        case 10: { // Удаляем узлы с нулевым потенциалом (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 11: {
            comment = DMKM.this.getComment("MakeFlow.FinishDeleteZero");
            break;
        }
        case 12: {
            comment = DMKM.this.getComment("MakeFlow.EndPhase");
            break;
        }
        case 13: { // Псевдомаксимальный поток в сети Диница построен.
            comment = DMKM.this.getComment("MakeFlow.EndDinizStep");
            args = new Object[]{new Integer(d.pseudoResultFlow)};
            break;
        }
    }
```

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

        /**
          * Выполняет действия по отрисовке состояния
          */
        public void drawState() {
            switch (state) {
                case 2: { // Удаляем узлы с нулевым потенциалом. После построения сети
                      Диница.
                    d.vis.showPotentials();
                    break;
                }
                case 3: { // Удаляем узлы с нулевым потенциалом (автомат)
                    child.drawState();
                    break;
                }
                case 4: {
                    d.vis.showDelete(false);
                    break;
                }
                case 6: { // Находим узел с минимальным потенциалом (автомат)
                    child.drawState();
                    break;
                }
                case 7: { // Проталкиваем поток из этого узла в сток (автомат)
                    child.drawState();
                    break;
                }
                case 8: { // Проталкиваем поток из этого узла в исток (автомат)
                    child.drawState();
                    break;
                }
                case 9: { // delete_zero_after_flow
                    d.vis.showDelete(true);
                    break;
                }
                case 10: { // Удаляем узлы с нулевым потенциалом (автомат)
                    child.drawState();
                    break;
                }
                case 11: {
                    d.vis.showDelete(true);
                    break;
                }
                case 12: {
                    d.vis.showDelete(true);
                    break;
                }
                case 13: { // Псевдомаксимальный поток в сети Диница построен.
                    d.vis.showDiniz(true);
                    break;
                }
            }
        }
    }

    /**
      * Убновляем поток сети, дополняя его псевдо-потоком из вспомогательной сети.
      */
    private final class UpdateFlow extends BaseAutomata implements Automata {
        /**
          * Начальное состояние автомата.
          */
        private final int START_STATE = 0;

        /**
          * Конечное состояние автомата.
          */
        private final int END_STATE = 2;

        /**
          * Конструктор.
          */
        public UpdateFlow() {
            super(
                "UpdateFlow",
                0, // Номер начального состояния
                2, // Номер конечного состояния
                new String[]{
```

```
            "Начальное состояние",
            "Убновляем поток сети, дополняя его псевдо-потоком из вспомогательной
              сети",
            "Конечное состояние"
        }, new int[]{
            Integer.MAX_VALUE, // Начальное состояние,
            1, // Убновляем поток сети, дополняя его псевдо-потоком из
              вспомогательной сети
            Integer.MAX_VALUE, // Конечное состояние
        }
    );
}

/**
  * Сделать один шаг автомата в перед.
  */
protected void doStepForward(int level) {

    // Переход в следующее состояние
    switch (state) {
        case START_STATE: { // Начальное состояние
            state = 1; // Убновляем поток сети, дополняя его псевдо-потоком из
              вспомогательной сети
            break;
        }
        case 1: { // Убновляем поток сети, дополняя его псевдо-потоком из
              вспомогательной сети
            state = END_STATE;
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // Убновляем поток сети, дополняя его псевдо-потоком из
              вспомогательной сети
            int i, j;
            for (i = 0; i < d.n; i++)
                for (j = 0; j < d.n; j++)
                    stack.pushInteger(d.f[i][j]);

            for (i = 0; i < d.n; i++)
                for (j = 0; j < d.n; j++){
                    // Прямая дуга
                }
            break;
        }
    }
}

/**
  * Сделать один шаг автомата назад.
  */
protected void doStepBackward(int level) {
    // Обращение действия в текущем состоянии
    switch (state) {
        case 1: { // Убновляем поток сети, дополняя его псевдо-потоком из
              вспомогательной сети
            int i, j;
            for (i = d.n - 1; i >= 0; i--)
                for (j = d.n - 1; j >= 0; j--)
                    d.f[i][j] = stack.popInteger();
            break;
        }
    }

    // Переход в предыдущее состояние
    switch (state) {
        case 1: { // Убновляем поток сети, дополняя его псевдо-потоком из
              вспомогательной сети
            state = START_STATE;
            break;
        }
        case END_STATE: { // Начальное состояние
            state = 1; // Убновляем поток сети, дополняя его псевдо-потоком из
              вспомогательной сети
            break;
        }
    }
}
```

```
    /**
      * Комментарий к текущему состоянию
      */
    public String getComment() {
        String comment = "";
        Object[] args = null;
        // Выбор комментария
        switch (state) {
            case 1: { // Убновляем поток сети, дополняя его псевдо-потоком из
                  вспомогательной сети
                comment = DMKM.this.getComment("UpdateFlow.UpdateFlow");
                break;
            }
        }

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

    /**
      * Выполняет действия по отрисовке состояния
      */
    public void drawState() {
        switch (state) {
            case 1: { // Убновляем поток сети, дополняя его псевдо-потоком из
                  вспомогательной сети
                d.vis.showNetwork(true);
                break;
            }
        }
    }
}

/**
  * Главный автомат.
  */
private final class Main extends BaseAutomata implements Automata {
    /**
      * Начальное состояние автомата.
      */
    private final int START_STATE = 0;

    /**
      * Конечное состояние автомата.
      */
    private final int END_STATE = 10;

    /**
      * Конструктор.
      */
    public Main() {
        super(
            "Main",
            0, // Номер начального состояния
            10, // Номер конечного состояния
            new String[]{
                "Начальное состояние",
                "Отображаем исходную сеть.",
                "Строим сеть Диница (автомат)",
                "Главный цикл, выполняем пока можно посторить сеть Диница",
                "Вычисляем потенциалы в узлах вспомогательной сети (автомат)",
                "Строим псевдомаксимальный поток во вспомогательной сети (автомат)",
                "Убновляем поток сети, дополняя его псевдо-потоком из вспомогательной
                  сети (автомат)",
                "Строим сеть Диница (автомат)",
                "В сети Диница нет дополняющего пути.",
                "Сеть и построенный в ней поток.",
                "Конечное состояние"
            }, new int[]{
                Integer.MAX_VALUE, // Начальное состояние,
                -1, // Отображаем исходную сеть.
                0, // Строим сеть Диница (автомат)
                -1, // Главный цикл, выполняем пока можно посторить сеть Диница
                0, // Вычисляем потенциалы в узлах вспомогательной сети (автомат)
                0, // Строим псевдомаксимальный поток во вспомогательной сети
                  (автомат)
                0, // Убновляем поток сети, дополняя его псевдо-потоком из
                      вспомогательной сети (автомат)
                0, // Строим сеть Диница (автомат)
                1, // В сети Диница нет дополняющего пути.
```

```
            -1, // Сеть и построенный в ней поток.
            Integer.MAX_VALUE, // Конечное состояние
        }

    );
}

/**
  * Сделать один шаг автомата в перед.
  */
protected void doStepForward(int level) {
    // Переход в следующее состояние
    switch (state) {
        case START_STATE: { // Начальное состояние
            state = 1; // Отображаем исходную сеть.
            break;
        }
        case 1: { // Отображаем исходную сеть.
            state = 2; // Строим сеть Диница (автомат)
            break;
        }
        case 2: { // Строим сеть Диница (автомат)
            if (child.isAtEnd()) {
                child = null;
                stack.pushBoolean(false);
                state = 3; // Главный цикл, выполняем пока можно посторить сеть
                                Диница
            }
            break;
        }
        case 3: { // Главный цикл, выполняем пока можно посторить сеть Диница
            if (d.isNetwork) {
                state = 4; // Вычисляем потенциалы в узлах вспомогательной сети
                            (автомат)
            } else {
                state = 8; // В сети Диница нет дополняющего пути.
            }
            break;
        }
        case 4: { // Вычисляем потенциалы в узлах вспомогательной сети (автомат)
            if (child.isAtEnd()) {
                child = null;
                state = 5; // Строим псевдомаксимальный поток во вспомогательной
                            сети (автомат)
            }
            break;
        }
        case 5: { // Строим псевдомаксимальный поток во вспомогательной сети
                    (автомат)
            if (child.isAtEnd()) {
                child = null;
                state = 6; // Убновляем поток сети, дополняя его псевдо-потоком из
                            вспомогательной сети (автомат)
            }
            break;
        }
        case 6: { // Убновляем поток сети, дополняя его псевдо-потоком из
                    вспомогательной сети (автомат)
            if (child.isAtEnd()) {
                child = null;
                state = 7; // Строим сеть Диница (автомат)
            }
            break;
        }
        case 7: { // Строим сеть Диница (автомат)
            if (child.isAtEnd()) {
                child = null;
                stack.pushBoolean(true);
                state = 3; // Главный цикл, выполняем пока можно посторить сеть
                    Диница
            }
            break;
        }
        case 8: { // В сети Диница нет дополняющего пути.
            state = 9; // Сеть и построенный в ней поток.
            break;
        }
        case 9: { // Сеть и построенный в ней поток.
            state = END_STATE;
            break;
```

```
    }
}

// Действие в текущем состоянии
switch (state) {
    case 1: { // Отображаем исходную сеть.
        int i, j;
        int n = d.n;
        d.t = d.n - 1;

        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++) {
                d.f[i][j] = 0;
                d.dg[i][j] = 0;
                d.df[i][j] = 0;
                d.newArc[i][j] = false;
                d.newFlowArc[i][j] = false;
           }
        for (i = 0; i < n; i++) {
            d.dq = new int[n];
            d.p[i] = 0;
            d.pin[i] = 0;
            d.pout[i] = 0;
            d.del[i] = false;
            d.dst[i] = 0;
            d.q1[i] = 0;
            d.q2[i] = 0;
            d.q[i] = 0;

            d.diniz[i] = false;
            d.fl[i] = 0;
            d.dq[i] = 0;
        }
        d.pr = 0;
        d.pw = 0;
        d.dpw = 0;
        d.dpr = 0;
        break;
    }
    case 2: { // Строим сеть Диница (автомат)
        if (child == null) {
            child = new MakeNetwork();
            child.toStart();
        }
        child.stepForward(level);
        step--;
        break;
    }
    case 3: { // Главный цикл, выполняем пока можно посторить сеть Диница
        break;
    }
    case 4: { // Вычисляем потенциалы в узлах вспомогательной сети (автомат)
        if (child == null) {
            child = new GetPotentials();
            child.toStart();
        }
        child.stepForward(level);
        step--;
        break;
    }
    case 5: { // Строим псевдомаксимальный поток во вспомогательной сети
          (автомат)
        if (child == null) {
            child = new MakeFlow();
            child.toStart();
        }
        child.stepForward(level);
        step--;
        break;
    }
    case 6: { // Убновляем поток сети, дополняя его псевдо-потоком из
          вспомогательной сети (автомат)
        if (child == null) {
            child = new UpdateFlow();
            child.toStart();
        }
        child.stepForward(level);
        step--;
        break;
    }
```

```
        case 7: { // Строим сеть Диница (автомат)
            if (child == null) {
                child = new MakeNetwork();
                child.toStart();
            }
            child.stepForward(level);
            step--;
            break;
        }
        case 8: { // В сети Диница нет дополняющего пути.
            break;
        }
        case 9: { // Сеть и построенный в ней поток.
            stack.pushInteger(d.resultFlow);
            d.resultFlow = 0;
            for (int i = 0; i < d.n; i++)
                d.resultFlow += d.f[0][i];
            break;
        }
    }
}

/**
  * Сделать один шаг автомата назад.
  */
protected void doStepBackward(int level) {
    // Обращение действия в текущем состоянии
    switch (state) {
        case 1: { // Отображаем исходную сеть.
            break;
        }
        case 2: { // Строим сеть Диница (автомат)
            if (child == null) {
                child = new MakeNetwork();
                child.toEnd();
            }
            child.stepBackward(level);
            step++;
            break;
        }
        case 3: { // Главный цикл, выполняем пока можно посторить сеть Диница
            break;
        }
        case 4: { // Вычисляем потенциалы в узлах вспомогательной сети (автомат)
            if (child == null) {
                child = new GetPotentials();
                child.toEnd();
            }
            child.stepBackward(level);
            step++;
            break;
        }
        case 5: { // Строим псевдомаксимальный поток во вспомогательной сети
              (автомат)
            if (child == null) {
                child = new MakeFlow();
                child.toEnd();
            }
            child.stepBackward(level);
            step++;
            break;
        }
        case 6: { // Убновляем поток сети, дополняя его псевдо-потоком из
              вспомогательной сети (автомат)
            if (child == null) {
                child = new UpdateFlow();
                child.toEnd();
            }
            child.stepBackward(level);
            step++;
            break;
        }
        case 7: { // Строим сеть Диница (автомат)
            if (child == null) {
                child = new MakeNetwork();
                child.toEnd();
            }
            child.stepBackward(level);
            step++;
            break;
```

```
            }
            case 8: { // В сети Диница нет дополняющего пути.
                break;
            }
            case 9: { // Сеть и построенный в ней поток.
                d.resultFlow = stack.popInteger();
                break;
            }
        }

        // Переход в предыдущее состояние
        switch (state) {
            case 1: { // Отображаем исходную сеть.
                state = START_STATE;
                break;
            }
            case 2: { // Строим сеть Диница (автомат)
                if (child.isAtStart()) {
                    child = null;
                    state = 1; // Отображаем исходную сеть.
                }
                break;
            }
            case 3: { // Главный цикл, выполняем пока можно посторить сеть Диница
                if (stack.popBoolean()) {
                    state = 7; // Строим сеть Диница (автомат)
                } else {
                    state = 2; // Строим сеть Диница (автомат)
                }
                break;
            }
            case 4: { // Вычисляем потенциалы в узлах вспомогательной сети (автомат)
                if (child.isAtStart()) {
                    child = null;
                    state = 3; // Главный цикл, выполняем пока можно посторить сеть
                                                                                    Диница
                }
                break;
            }
            case 5: { // Строим псевдомаксимальный поток во вспомогательной сети
                        (автомат)
                if (child.isAtStart()) {
                    child = null;
                    state = 4; // Вычисляем потенциалы в узлах вспомогательной сети
                  (автомат)
                }
                break;
            }
            case 6: { // Убновляем поток сети, дополняя его псевдо-потоком из
                        вспомогательной сети (автомат)
                if (child.isAtStart()) {
                    child = null;
                    state = 5; // Строим псевдомаксимальный поток во вспомогательной
                                        сети (автомат)
                }
                break;
            }
            case 7: { // Строим сеть Диница (автомат)
                if (child.isAtStart()) {
                    child = null;
                    state = 6; // Убновляем поток сети, дополняя его псевдо-потоком из
                                        вспомогательной сети (автомат)
                }
                break;
            }
            case 8: { // В сети Диница нет дополняющего пути.
                state = 3; // Главный цикл, выполняем пока можно посторить сеть Диница
                break;
            }
            case 9: { // Сеть и построенный в ней поток.
                state = 8; // В сети Диница нет дополняющего пути.
                break;
            }
            case END_STATE: { // Начальное состояние
                state = 9; // Сеть и построенный в ней поток.
                break;
            }
        }
    }
```

```
/**
  * Комментарий к текущему состоянию
  */
public String getComment() {
    String comment = "";
    Object[] args = null;
    // Выбор комментария
    switch (state) {
        case START_STATE: { // Начальное состояние
            comment = DMKM.this.getComment("Main.START_STATE");
            break;
        }
        case 2: { // Строим сеть Диница (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 4: { // Вычисляем потенциалы в узлах вспомогательной сети (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 5: { // Строим псевдомаксимальный поток во вспомогательной сети
              (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 6: { // Убновляем поток сети, дополняя его псевдо-потоком из
                    вспомогательной сети (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 7: { // Строим сеть Диница (автомат)
            comment = child.getComment();
            args = new Object[0];
            break;
        }
        case 8: { // В сети Диница нет дополняющего пути.
            comment = DMKM.this.getComment("Main.NoPath");
            break;
        }
        case END_STATE: { // Конечное состояние
            comment = DMKM.this.getComment("Main.END_STATE");
            args = new Object[]{new Integer(d.resultFlow)};
            break;
        }
    }

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

/**
  * Выполняет действия по отрисовке состояния
  */
public void drawState() {
    switch (state) {
        case START_STATE: { // Начальное состояние
            d.vis.showNetwork(false);
            break;
        }
        case 1: { // Отображаем исходную сеть.
            d.vis.showNetwork(false);
            break;
        }
        case 2: { // Строим сеть Диница (автомат)
            child.drawState();
            break;
        }
        case 4: { // Вычисляем потенциалы в узлах вспомогательной сети (автомат)
            child.drawState();
            break;
        }
        case 5: { // Строим псевдомаксимальный поток во вспомогательной сети
                    (автомат)
            child.drawState();
            break;
        }
```

```
                case 6: { // Убновляем поток сети, дополняя его псевдо-потоком из
                                    вспомогательной сети (автомат)
                    child.drawState();
                    break;
                }
                case 7: { // Строим сеть Диница (автомат)
                    child.drawState();
                    break;
                }
                case 8: { // В сети Диница нет дополняющего пути.
                    d.vis.showDiniz(false);
                    break;
                }
                case 9: { // Сеть и построенный в ней поток.
                    d.vis.showNetwork(true);
                    break;
                }
                case END_STATE: { // Конечное состояние
                    d.vis.showNetwork(true);
                    break;
                }
            }
        }
    }
}
```

Приложение 3. Исходные коды визуализатора

DMKMVisualizer.java

```
package ru.ifmo.vizi.dmkm;


import java.util.StringTokenizer;
import java.io.*;
import ru.ifmo.vizi.base.SmartTokenizer;
import ru.ifmo.vizi.base.Base;
import ru.ifmo.vizi.base.I18n;
import ru.ifmo.vizi.base.VisualizerParameters;
```

```
import ru.ifmo.vizi.base.ui.SaveLoadDialog;
import ru.ifmo.vizi.base.ui.AutoControlsPane;
import ru.ifmo.vizi.base.ui.HintedButton;
import ru.ifmo.vizi.base.ui.MultiButton;
import ru.ifmo.vizi.base.ui.SpinPanel;
import ru.ifmo.vizi.dmkm.net.Net;
import ru.ifmo.vizi.dmkm.net.RandomNetGenerator;


import java.awt.*;
import java.awt.*;

/**
 * Applet.
 *
 * @author Yuri Bedny
 * @version $Id$
 */

public final class DMKMVisualizer extends Base {

    private Frame forefather;

    private final DMKM auto;

    private final DMKM.Data data;

    private Net net;

    private NetEditor editor;

    private AutoControlsPane controlsPane;

    private Panel panel;

    private Panel editPanel;

    private MultiButton modeButton;

    private SaveLoadDialog saveLoadDialog;
       private HintedButton saveLoadButton;

    private NetRecord lNets[];

    private SpinPanel nPanel;

    private Panel bottomPanel;


    /**
     * Rewinds algorithm to the specified step.
     *
     * @param step spet of the algorith to rewind to.
     */

    private void rewind(int n, int g[][], int step) {
       clearNet();
        data.n = n;
        data.t = n - 1;
        net.setNumOfVertices(n);
        allocateMem();
        setNet(g);
        auto.toStart();
        while (!auto.isAtEnd() && auto.getStep() < step) {
            auto.stepForward(0);
        }
        setVisMode();
    }


    /**
     * This method creates panel with vis's controls.
     * @return panel with vis's controls.
     */
    public Component createControlsPane() {

        /**
         * Creates a panel.
         */
        panel = new Panel();
```

```
/**
 * Sets the layout manager.
 */
panel.setLayout(new BorderLayout());

/**
 * Creates the panel to manage visualizer.
 */
controlsPane = new AutoControlsPane(config, auto, forefather, true);

/**
 * Creates the panel for editing net.
 */
editPanel = new Panel();


if (config.getBoolean("button-ShowSaveLoad")) {
    saveLoadButton = new HintedButton(config, "button-SaveLoad") {
        protected void click() {
            saveLoadDialog.center();
            StringBuffer buffer = new StringBuffer();

                            /* Counts the number of arcs in the current net */
                            int m = 0;
                            for (int i = 0; i < data.n; i++) {
                                    for (int j = 0; j < data.n; j++) {
                                            if (data.g[i][j] > 0) {
                                                    m++;
                                            }
                                    }
                            }

            buffer.append("/* ").append(
                    config.getParameter("NumberOfVerticesComment")
            ).append(" */\n");

            buffer.append("NumberOfVertices = ").append(data.n).append("\n\n");

            buffer.append("/* ").append(
                I18n.message(
                    config.getParameter("NumberOfArcsComment"),
                    new Integer(data.n)
                )
            ).append(" */\n");

            buffer.append("NumberOfArcs = ").append(m).append("\n\n");

            buffer.append("/* ").append(
                    config.getParameter("ArcsComment")
            ).append(" */\n");

            buffer.append("Arcs = \n");
            for (int i = 0; i < data.n; i++) {
              for (int j = 0; j < data.n; j++) {
                     if (data.g[i][j] > 0) {
                             buffer.append(i + " " + j + " " +
                                     data.g[i][j]).append("\n");
                     }
                }
            }

            buffer.append("\n/* ").append(
                config.getParameter("StepComment")
            ).append(" */\n");
            buffer.append("Step = ").append(auto.getStep());

            saveLoadDialog.show(buffer.toString());
        }
    };
}

saveLoadDialog = new SaveLoadDialog(config, forefather) {
    public boolean load(String text) throws Exception {
        SmartTokenizer tokenizer = new SmartTokenizer(text, config);

        tokenizer.expect("NumberOfVertices");
        tokenizer.expect("=");
        int n = tokenizer.nextInt(2, 10);
        int[][] g = new int[n][n];
```

```
        tokenizer.expect("NumberOfArcs");
        tokenizer.expect("=");
        int m = tokenizer.nextInt(0, n * (n - 1) / 2);

        tokenizer.expect("Arcs");
        tokenizer.expect("=");
        int i, j, c;
        for (int k = 0; k < m; k++) {
                            i = tokenizer.nextInt(0, n - 1);
                            j = tokenizer.nextInt(0, n - 1);
                            c = tokenizer.nextInt(1,
                                   config.getInteger("maxArcCapacity"));
                            if (g[j][i] > 0 || g[i][j] > 0) {
                                   throw new Exception( I18n.message(
                     config.getParameter( "TwoArcsExceptionMsg"), new
                     Integer(tokenizer.lineno() ) ) );
                            }
              g[i][j] = c;
        }

        tokenizer.expect("Step");
        tokenizer.expect("=");
        int step = tokenizer.nextInt();

        tokenizer.expectEOF();

        rewind(n, g, step);

        return true;
    }
};


/**
 * Creates the button to change visualizing mode to editing mode and vice versa.
 */
String states[] = new String[2];
states[0] = "multibutton-edit";
states[1] = "multibutton-vis";
modeButton = new MultiButton(config, states) {
    protected int click(int state) {
        if (state == 0) {
            auto.toStart();
            setEditMode();
            showNetwork(false);
            return 1;
        } else {
            setVisMode();
            showNetwork(false);
            return 0;
        }
    }
};

/**
 * Creates the button for randomizing the net.
 */
HintedButton randomButton = new HintedButton(config, "random-button") {
    protected void click() {
         auto.toStart();
        randomizeNet();
        showNetwork(false);
    }
};

/**
 * Create the panel to show number of the vertices.
 */
nPanel = new SpinPanel(config, "nPanel") {
    protected void click(double value) {

        /**
         * Sets an automata to the start state.
         */
        while (!auto.isAtStart()) {
            auto.stepBackward(1);
        }

        /**
```

```
             * Gets number of the vertices and sets number to the sink according to
                                                                    this information.
             */
            int oldn = data.n;
            data.n = getIntValue();
            data.t = data.n - 1;
            resizeNet(oldn);

            /**
             * Sets number of vertices in net.
             */
            net.setNumOfVertices(data.n);

            /**
             * Shows a net.
             */
            showNetwork(false);
            net.repaint();
        }
    };

    HintedButton clearButton = new HintedButton(config, "clear-button") {
        protected void click() {
            /**
             * Sets an automata to start state.
             */
            auto.toStart();

            /**
             * Clear the net.
             */
            clearNet();

            /**
             * Shows an initial net without flow.
             */
            showNetwork(false);
        }
    };

    bottomPanel = new Panel();
    bottomPanel.add(saveLoadButton);
    bottomPanel.add(modeButton);
    editPanel.add(clearButton);
    editPanel.add(nPanel);
    editPanel.add(randomButton);

    /**
     * Ands this panel to main panel.
     */
    panel.add(controlsPane, BorderLayout.NORTH);
    panel.add(editPanel, BorderLayout.CENTER);
    panel.add(bottomPanel, BorderLayout.SOUTH);
    return panel;


}

/**
 * Changes the size of the net.
 * @param oldn old size.
 */
private void resizeNet(int oldn) {
    int oldg[][] =  new int[oldn][oldn];
    for (int i = 0; i < oldn; i++)
        for (int j = 0; j < oldn; j++)
            oldg[i][j] = data.g[i][j];
    allocateMem();
    if (oldn < data.n) {
        for (int i = 0; i < oldn; i++) {
            for (int j = 0; j < oldn; j++) {
                if (i != oldn - 1 && j != oldn - 1) {
                    data.g[i][j] = oldg[i][j];
                } else if (i == oldn - 1) {
                    data.g[data.n - 1][j] = oldg[i][j];
                } else {
                    data.g[i][data.n - 1] = oldg[i][j];
                }
            }
        }
```

```
    } else {
        for (int i = 0; i < oldn; i++) {
            for (int j = 0; j < oldn; j++) {
                if (i == oldn - 2 || j == oldn - 2 || oldg[i][j] == 0) continue;
                if (i != oldn - 1 && j != oldn - 1) {
                    data.g[i][j] = oldg[i][j];
                } else if (i == oldn - 1) {
                    data.g[data.n - 1][j] = oldg[oldn - 1][j];
                } else if (j == oldn - 1) {
                    data.g[i][data.n - 1] = oldg[i][oldn - 1];
                }
            }
        }
    }
}

/**
 * Save the net.
 */
public String saveNet() {
    NetRecord netRecord = new NetRecord(data.n, data.g);
    return netRecord.writeNet();
}

/**
 * Load the net.
 */
public void loadNet(String s) {
           NetRecord netRecord = new NetRecord(s, true);

           if (netRecord.n == 0) {
                  return;
           }
    clearNet();
    data.n = netRecord.n;
    data.t = data.n - 1;
    net.setNumOfVertices(data.n);
    allocateMem();
    setNet(netRecord.g);
    auto.toStart();
    if (netRecord.step > 0) {
        while (netRecord.step != auto.getStep()) {
            auto.stepForward(0);
        }
        setVisMode();
    }
}


// Устанавливаем матрицу смежности сети (data.g)
private void setNet(int g[][]) {
    for (int i = 0; i < data.n; i++)
        for (int j = 0; j < data.n; j++)
            data.g[i][j] = g[i][j];
}

// Переход в режим редактирования сети
private void setEditMode() {
    modeButton.setState(1);
    controlsPane.setVisible(false);
    editPanel.setVisible(true);
    if (!panel.isValid()) {
        panel.validate();
    }
    modeButton.requestFocus();
    activateNetEditor();
}

// Переход в режим визуализации
private void setVisMode() {
    modeButton.setState(0);
    editPanel.setVisible(false);
    controlsPane.setVisible(true);
    if (!panel.isValid()) {
        panel.validate();
    }
    modeButton.requestFocus();
    deactivateNetEditor();
 }
```

```
private void clearNet() {
    int i, j;
    int n = data.n;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++) {
            data.g[i][j] = 0;
            data.f[i][j] = 0;
            data.dg[i][j] = 0;
            data.df[i][j] = 0;
            data.newArc[i][j] = false;
            data.newFlowArc[i][j] = false;
       }
    for (i = 0; i < n; i++) {
    data.dq = new int[n];
        data.p[i] = 0;

        data.pin[i] = 0;
        data.pout[i] = 0;
        data.del[i] = false;
        data.dst[i] = 0;
        data.q1[i] = 0;
        data.q2[i] = 0;
    data.q[i] = 0;
        data.diniz[i] = false;
        data.fl[i] = 0;
    data.dq[i] = 0;
    }
    data.pr = 0;
    data.pw = 0;
    data.dpw = 0;
    data.dpr = 0;
}

private void randomizeNet() {
    clearNet();
    RandomNetGenerator.setConfig(config);
    RandomNetGenerator.setMethod(RandomNetGenerator.ALL_VERTICES_METHOD);
    RandomNetGenerator.random(data.n, data.g);
}

/**
 * Invoked when state changed.
 */

// Выделение памяти под переменные апплета
private void allocateMem() {
    int n = data.n;
    data.g = new int[n][n];
    data.f = new int[n][n];
    data.dg = new int[n][n];
    data.df = new int[n][n];
    data.p = new int[n];
    data.pin = new int[n];
    data.pout = new int[n];
    data.del = new boolean[n];
    data.diniz = new boolean[n];
    data.newArc = new boolean[n][n];
    data.newFlowArc = new boolean[n][n];
    data.fl = new int[n];
    data.q = new int[n];
    data.dq = new int[n];
    data.dst = new int[n];
    data.q1 = new int[n];
    data.q2 = new int[n];
    data.s = 0;
    data.pr = 0;
    data.pw = 0;
    data.dpw = 0;
    data.dpr = 0;

    // Все ссылки на data, теперь приходится переделывать (таковый есть у net и у
                                                                    editor-a)
    net.updateAppletData(data.g,  data.f);
    editor.updateAppletData(data.g);
}

/**
 * Creates an vis.
 */
public DMKMVisualizer(VisualizerParameters visualizerParameters) {
```

```
        super(visualizerParameters);

        forefather = visualizerParameters.getForefather();
        auto = new DMKM(locale);
        data = auto.d;
        data.vis = this;

        /**
         * Creates an editor for the net.
         */
        editor = new NetEditor(config, this);

        /**
         * Create the net.
         */
        net = new Net(config, editor);

        createInterface(auto);


        lNets = new NetRecord[0];

        /**
         *  Loads some examples.
         */
        loadExampleNets();

        /**
         * Loads the first net from examples.
         */
        onLoad(1);

        /**
         * Sets the net to editor.
         */
        editor.setNet(net);

        /**
         * Sets number of vertices in net.
         */
        net.setNumOfVertices(data.n);

        /**
         * Adds net to the client pane.
         */
        clientPane.add(net);

        /**
         * Sets visualization mode.
         */
        setVisMode();

        showNetwork(false);

    }

    public void onLoad(int i) {
        if (i == -1) return;
        clearNet();
        NetRecord lNet = lNets[i];
        nPanel.setValue(lNet.n);
        data.n = lNet.n;
        data.t = data.n - 1;
        net.setNumOfVertices(data.n);
        allocateMem();
        setNet(lNet.g);
        auto.toStart();
        if (lNet.step > 0) {
            while (lNet.step != auto.getStep()) {
                auto.stepForward(0);
            }
            setVisMode();
        }
    }

    /**
     * Sets an editor to active mode.
     */
    private void activateNetEditor() {
        editor.setActive(data.g, net.getVertexLayouter());
```

```
    net.addMouseListener(editor);
    net.addMouseMotionListener(editor);
}

/**
 * Sets an editor to passive mode.
 */
private void deactivateNetEditor() {
    editor.setPassive();
    net.removeMouseListener(editor);
    net.removeMouseMotionListener(editor);
}

/**
 * Invoked when net changed via NetEditor.
 */
public void onNetChanged() {
    showNetwork(false);
    net.repaint();
}

/**
 * Invoked when arc dragged in edit mode.
 */
public void onArcDragged() {
    net.repaint();
}

/**
 * Invoked when menu PopUped in edit mode.
 */
public void onMenuPopUp() {
    showNetwork(false);
    net.repaint();
}

/**
 * Load some examples.

 */
private void loadExampleNets() {
    int num = config.getInteger("example-num");
    String strNet, netName;
    for (int i = 0; i < num; i++) {
        strNet = config.getParameter("example-net" + (i + 1));
        netName =  config.getParameter("example-net" + (i + 1) + "-name");;
        if (!strNet.equals("")) {

            int len = lNets.length;
            NetRecord temp[] = new NetRecord[len];
            for (int j = 0; j < len; j++)
                temp[j] = lNets[j];
            lNets = new NetRecord[len + 1];
            for (int j = 0; j < len; j++)
                lNets[j] = temp[j];
            lNets[len] = new NetRecord(strNet);
        }
    }
}

/**
 *  Invoked when client pane reshaped.
 */
protected void layoutClientPane(int width, int height) {
    net.setSize(width, height);
    net.paint();
}

/**
 * Paints network in current step.
 * @param isFlow if flow need to show.
 */
public void showNetwork(boolean isFlow) {
    net.setNet(data.g);
    net.setFlow(data.f, isFlow);
    net.setPotentials(false);
    net.paint();
}

/**
```

```
     * Paints diniz network in current step.
     * @param isFlow if flow need to show.
     */
    public void showDiniz(boolean isFlow) {
        net.setNet(data.dg, data.del);
        net.setFlow(data.df, isFlow, data.newFlowArc);
        net.setPotentials(false);
        net.paint();
    }

    /**
     * Paints network with vertices potentials in current step.
     */
    public void showPotentials() {
        net.setNet(data.dg, data.del);
        net.setFlow(data.f, false);
        net.setPotentials(data.p);
        net.paint();
    }

    /**
     * Paints network after removing vertices with zero potential.
     * @param isFlow

     */
    public void showDelete(boolean isFlow) {
        net.setNet(data.dg, data.del);
        net.setFlow(data.df, isFlow);
        net.setPotentials(data.p);
        net.paint();
    }

    /**
     * Paints network which contains selected vertex with minimum potential.
     * @param isFlow
     */
    public void showMinv(boolean isFlow) {
        net.setNet(data.dg, data.del);
        net.setFlow(data.df, isFlow);
        net.setPotentials(data.p, data.minv);
        net.paint();
    }

    /**
     * Paints network after pushing flow forward.
     */
    public void showForward() {
        net.setNet(data.dg, data.del);
        net.setFlow(data.df, true, data.newArc);
        net.setPotentials(data.p, data.minv);
        net.paint();
    }

    /**
     * Paints network after pushing flow backward.
     */
    public void showBackward() {
        net.setNet(data.dg, data.del);
        net.setFlow(data.df, true, data.newArc);
        net.setPotentials(data.p, data.minv);
        net.paint();
    }

}
```