Российская академия наук
Сибирское отделение Институт систем информатики им. А.П. Ершова

ИНФОРМАТИКА В НАУКЕ И ОБРАЗОВАНИИ

Под редакцией
проф. Виктора Николаевича Касьянова

Novosibirsk 2012

УДК 519.68;
681.3.06
ББК З 22.183.49+ З 22.174.2

Информатика в науке и образовании: Сб. науч. тр. / Под ред. В.Н. Касьянова. – Новосибирск: Ин-т систем информатики им. А.П. Ершова СО РАН, 2012. – 178 с. – (Сер.: Конструирование и оптимизация программ; Вып. 21).

Сборник является двадцать первым в серии, издаваемой Институтом систем информатики СО РАН по проблемам конструирования и оптимизации программ. Посвящен актуальным проблемам применения системной информатики в науке и образовании.

Представляет интерес для системных программистов, студентов и аспирантов, специализирующихся в области системного и теоретического программирования, и для всех тех, кто интересуется проблемами современной информатики и программирования.

Siberian Branch of the Russian Academy of Sciences
A.P. Ershov Institute of Informatics Systems

INFORMATICS IN RESEARCH AND EDUCATION

Edited by
prof. V.N. Kasyanov

Novosibirsk 2012

Informatics in research and education / Ed. Victor Kasyanov. – Novosibirsk: A.P. Ershov Institute of Informatics Systems, 2012. – 178 p. – (Ser.: Program construction and optimization; Iss. 21).

This volume is the twenty first in the series on the problems of program construction and optimization, published by the A.P. Ershov Institute of Informatics Systems. The volume is devoted to actual problems of application of system informatics to research and education.

It may be of interest to system programmers, students and postrgraduates specializing in the area of system and theoretical programming, as well as all those interested in modern informatics and programming.

ПРЕДИСЛОВИЕ РЕДАКТОРА

Сборник является двадцатым первым в серии «Конструирование и оптимизация программ», издаваемой Институтом систем информатики имени А.П. Ершова СО РАН (ИСИ СО РАН), и посвящен актуальным проблемам применения системной информатики в науке и образовании.

Продолжая уже сложившиеся традиции, данный выпуск, как и предыдущие, базируется на результатах исследований, ведущихся в лаборатории по конструированию и оптимизации программ ИСИ СО РАН совместно с кафедрой программирования Новосибирского государственного университета при финансовой поддержке Российского фонда фундаментальных исследований, Российского гуманитарного научного фонда, Министерства образования и науки Российской Федерации, а также компаний «Микрософт» и «Интел». Объединяет статьи, составившие сборник, также то, что почти все они подготовлены по результатам трехгодичного проекта «Методы и технологии конструирования и оптимизации программных систем для суперкомпьютеров и компьютерных сетей», выполненного авторами статей в рамках приоритетного направления СО РАН «Архитектура, системные решения, программное обеспечение и информационная безопасность информационно-вычислительных комплексов и сетей новых поколений. Системное программирование».

Открывают сборник две статьи Касьянова В.Н. с описанием текущего состояния работ по международному проекту, направленному на создание стандартизованного языка GraphML для описания графов на основе XML. В первой статье рассматриваются базисные средства языка GraphML, а во второй – дополнительные его возможности.

Статья Касьянова В.Н. и Касьяновой Е.В. посвящена таким средствам поддержки применения теоретико-графовых методов в информатике и программировании, создаваемым в ИСИ СО РАН, как WikiGRAPP, WEGA, HIGRES и Visual Graph.

В первой статье Идрисова Р. И. рассматривается облачный сервис для научных вычислений и образования на базе языка Sisal. Вторая его статья посвящена рассмотрению параллелизма в языке JavaScript.

В своей первой статье Несговорова Г.П. исследует пути развития и перспективы современного направления науки, получившего название «биоинформатика», а во второй – роль и место информационных технологий в гуманитарных исследованиях и гуманитарном образовании.

Статья Стасенко А.П. описывает технологию сокращения объемов тестирования на основе сопоставления изменений в исходном коде с данными о тестовом покрытии.

В статье Шманиной Т.В. представлена информационная система для поддержки процесса проведения исследований на основе литературных источников.

Статья Золотухина Т.А. посвящена вопросам визуализации графов большого размера при помощи программного средства Visual Graph.

В статье Гордеева Д.С. предложены методы и средства визуализации алгоритмов на графах, обеспечивающие задание алгоритма в качестве параметра.

Завершает сборник статья Малининой Ю.В, в которой рассматривается подход к автоматическому выявления тематической карты документа на основе лексических цепей.

Проф. В.Н. Касьянов

В.Н. Касьянов

ЯЗЫК ПРЕДСТАВЛЕНИЯ ГРАФОВ GRAPHML: БАЗОВЫЕ СРЕДСТВА[1]

ВВЕДЕНИЕ

Современное программирование нельзя представить себе без теоретико-графовых методов и алгоритмов [3]. Широкая применимость графов связана с тем, что они являются очень естественным средством объяснения сложных ситуаций на интуитивном уровне. Эти преимущества представления сложных структур и процессов графами становятся еще более ощутимыми при наличии хороших средств их визуализации [4, 11].

Ясно, что инструменты визуализации информации на основе графовых моделей, подобно всем другим инструментам, имеющим дело со структурированными данными, нуждаются в сохранении и передаче графов и ассоциированных с ними данных. Среди многочисленных форматов файлов для представлений графов применяются основанные на ASCII-кодах таблицы (матрицы) или списки, такие как табулированные файлы, к которым относятся *.dl файлы UCINET [5] и *.net файлы Паёка (Pajek) [10]. Есть среди них и основанные на XML форматы для представления графов, такие как GXL [17] и DyNetML [14]. Еще один используемый формат представления графов – это язык GML (Graph Modelling Language) [12], работа над которым началась в 1995 на 4-м симпозиуме по рисованию графов GD-95 в г. Пассау и завершилась в 1996 на 5-м симпозиуме по рисованию графов GD-96 в г. Беркли. Язык GML до сих остается основным файловым форматом для системы Graphlet, а также поддерживается рядом других систем обработки графов.

Однако долгое время среди используемых форматов представления графов не находилось ни одного, который был бы достаточно широко принятым в качестве стандартного; по существу, инструменты работы с графами поддерживали (да и сейчас многие из них поддерживают) лишь некоторую часть из существующих клиентских форматов, обычно состоящую из

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

видов графовых представлений, ограниченных по выразимости и специфике конкретной областью применения.

Поэтому неслучайно в 2000 году наблюдательный комитет симпозиума по рисованию графов (Graph Drawing Steering Committee) организовал рабочее совещание по форматам обменами графовыми данными, состоявшееся в г. Вильямсбурге в рамках 8-го симпозиума по рисованию графов (GD-2000) [9]. Как следствие была сформирована неформальная рабочая группа по выработке основанного на языке XML формата обмена графами GraphML, который, в частности, был бы пригоден для обмена данными между инструментами рисования графов и другими приложениями и в конечном счете лёг бы в основу стандарта описания графов.

Рабочая группа по созданию языка GraphML объединяет десятки специалистов из разных организаций и стран, и её работу наравне с другими координируют Ulrik Brandes (Университет г. Констанц, Германия), Markus Eiglsperger (Тюбингенский университет, Германия), Michael Kaufmann (Тюбингенский университет, Германия), Jürgen Lerner (Университет г. Констанц, Германия) и Christian Pich (Университет г. Констанц, Германия).

Первый отчёт по языку вышел в 2001 году [6]. С тех пор язык был расширен в части поддержки основных типов атрибутов и в части включения информации для использования синтаксическими анализаторами [7, 8]. Ведется работа по включению абстрактной информации для описания топологии графа и шаблонов, с помощью которых эту информацию можно преобразовать в различные графические форматы. Программное обеспечение для поддержки работы с GraphML также находится в стадии разработки.

Благодаря XML синтаксису GraphML может использоваться в комбинации с другими форматами, основанными на XML. С другой стороны, свой собственный механизм расширения позволяет прикреплять <data> метки со сложным содержимым элементов GraphML, возможно, требуемым для исполнения с другими моделями XML содержимого. Примером использования таких меток со сложным содержимым является так называемый механизм SVG (Scalable Vector Graphics) [15], описывающий появление вершин и дуг в изображении. С другой стороны, GraphML может интегрироваться в другие приложения, например, в SOAP сообщения [16].

В данной статье рассматриваются базовые средства языка GraphML, достаточные для представления графовых моделей в большинстве приложений. В ней описывается, как графы и простые графовые данные представляются в формате GraphML. Другим возможностям языка, связанным с его расширением за счет введения дополнительных понятий для графовой

топологии, таких как вложенные графы, гиперграфы и порты, посвящена отдельная статья [2].

1. ЦЕЛИ РАЗРАБОТКИ И ИСПОЛЬЗУЕМАЯ ГРАФОВАЯ МОДЕЛЬ

Разработчики языка убеждены, что современный формат обмена графами не может быть монолитным, поскольку сервисы рисования графов используются в качестве компонентов более больших систем, и возникают сетевые сервисы. Всегда может потребоваться обмен графовыми данными между такими сервисами или этапами сервиса, а также между сервисами рисования графов и системами, специфическими для областей приложений.

Типичные пользовательские сценарии, которые предусматривались авторами для разработанного формата, собраны вокруг систем, спроектированных для произвольных приложений, имеющих дело с графами и другими данными, ассоциированными с ними. Такие системы могут содержать или вызывать сервисы рисования графов, которые добавляют или изменяют раскладку или графическую информацию. Такие сервисы могут вычислять только частичную информацию или промежуточные представления, поскольку они воплощают лишь часть многофазового подхода к укладке, такого как метрики топологических форм (the topology-shape-metrics) или схемы Сугиямы (Sugiyama frameworks) [10, 13].

Основную цель разработчики языка формулируют следующим образом. Формат обмена графами должен быть в состоянии представлять произвольные графы с произвольными дополнительными данными, включая укладку и графическую информацию. Дополнительная информация должна сохраняться в формате, подходящем для заданного конкретного приложения, но не должна усложнять представление данных из других приложений или мешать ему.

GraphML проектировался с ориентацией на эту цель, а также с учётом следующих более прагматических целей.

- Простота (Simplicity). Формат должен был прост для разбора и интерпретации как людьми, так и машинами. В качестве общего принципа формулируется отсутствие неоднозначностей и, таким образом, существование единственной хорошо определенной интерпретации для каждого валидного (valid) GraphML-документа.
- Общность (Generality). Не должно существовать ограничений по от-

ношению к графовой модели, т.е. гиперграфы[2], иерархические графы и т.д. должны быть выразимы с помощью одного и того же базисного формата.

- Расширяемость (Extensibility). Должна существовать возможность расширять формат хорошо определённым способом для представления дополнительных данных, требуемых произвольными приложениями или более сложным использованием (например, посылая алгоритм раскладки вместе с графом).
- Робастность (Robustness). Система, не способная обработать весь диапазон графовых моделей или дополнительной информации, должна быть в состоянии легко распознавать и извлекать то подмножество, которое она может обработать.

Под графовой моделью, описываемой с помощью базовых средств языка, понимается помеченный смешанный мультиграф

$$G = (V, E, L),$$

состоящий из множества вершин V, множества рёбер E (как неориентированных, так и ориентированных), соединяющих пары необязательно различных вершин, и множества разметок L, каждая из которых является частичной функцией, сопоставляющей элементам графа $\{G\} \cup V \cup E$ элементы некоторого заданного множества пометок.

Таким образом, рассматриваемая графовая модель включает графы, которые могут содержать ребра, дуги, петли, кратные дуги и кратные рёбра. Множества пометок могут кодировать, например, различные семантические свойства объектов, представленных в виде данной графовой модели, или различные геометрические свойства элементов графа в заданном его изображении на плоскости.

2. ЗАГОЛОВОК

В качестве примера представления рассмотрим на рис. 1 фрагмент GraphML-документа, который представляет простой непомеченный граф, изображённый на рис. 2.

[2] Здесь и ниже мы без определения используем стандартные понятия из теории графов (см., например, [1]).

```
<graphml>
        <graph edgedefault="directed">
        <node id="v1"/>
        <node id="v2"/>
        <node id="v3"/>
        <node id="v4"/>
        <edge source="v1" target="v2"/>
        <edge source="v1" target="v3"/>
        <edge source="v2" target="v4"/>
        <edge source="v2" target="v4" directed="false"/>
        </graph>
</graphml>
```

Рис.1. Фрагмент GraphML-документа для представления простого графа, изображённого на рис. 2

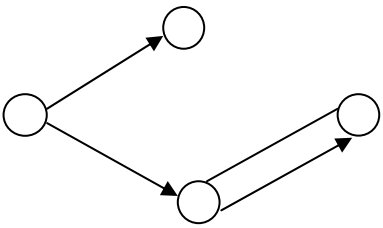


Рис. 2. Пример простого графа

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns=http://graphml.graphdrawing.org/xmlns
        xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
        http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd">
        <!--Content: List of graphs and data-->
</graphml>
```

Рис. 3. Минимальный валидный GraphML-документ

Следует отметить, что указанный фрагмент GraphML-документа как XML-документ не валиден, поскольку каждый валидный XML-документ должен декларировать в своём заголовке, является ли он DTD (document type definition) или XML-схемой. По-существу, заданное множество определений DTD и XML-схем определяет подмножество всех тех XML-документов, которые и формируют некоторый конкретный язык. Но язык GraphML был определен с помощью лишь некоторой схемы. И хотя DTD-определение предназначено для поддержки парсеров, которые не могут обработать схемные определения, единственной нормативной спецификацией языка GraphML является GraphML-схема, размещенная по адресу

http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd

Документ, представленный на рис. 3, является минимальным GraphML-документом, который валиден по данной схеме. Понятно, что данный документ определяет пустое множество графов. Части документа, начинающиеся с символов <!-- и завершающиеся символами -->, являются комментариями.

Первая строка документа – это инструкция обработки, которая определяет, что документ является подмножеством стандарта XML 1.0, и что документ выполнен в кодировке UTF-8, являющейся стандартом для XML-документов. Конечно, для GraphML-документов могут быть выбраны и другие кодировки.

Вторая строка содержит корневой элемент GraphML-документа – graphml, который, как и все остальные элементы языка GraphML, принадлежит пространству имен

http://graphml.graphdrawing.org/xmlns.

По этой причине с помощью XML-атрибута

xmlns="http://graphml.graphdrawing.org/xmlns"

именно это пространство имен в нашем документе и определено в качестве пространства имен данного документа, заданного по умолчанию. Следующие два XML-атрибута определяют XML-схему, которая используется для валидации данного документа. В нашем примере используется стандартная схема GraphML-документа, расположенная на сервере graphdrawing.org. Первый атрибут,

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance",

определяет xsi в качестве префикса пространства имен XML-схемы. Второй атрибут,

xsi: schemaLocation="http://graphml.graphdrawing.org/xmlns
http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd",

определяет местонахождение XML-схемы для элементов пространства имен GraphML. Он предоставляет информацию о том, что все элементы в пространстве имен GraphML являются валидными по отношению к файлу graphml.xsd, расположенному по указанному адресу. Конечно, валидация не должна обязательно выполняться с использованием данного файла. Локальные копии graphml.xsd также могут специфицироваться в качестве местонахождений XML-схем. Заметим, что обычно значение атрибута schemaLocation является списком пар, в которых первый элемент обозначает некоторое пространство имен, а второй указывает на файл, в котором элементы этого пространства определены.

Ссылка на XML-схему не обязательна, но она обеспечивает механизм для синтаксической проверки документа и поэтому строго рекомендуется. Минимальный GraphML-документ без ссылки на схему приведен на рис. 4. Заметим, что этот файл не является валидным документом в соответствии с XML-спецификацией.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml
xmlns="http://graphml.graphdrawing.org/xmlns" >
        <!--Content: List of graphs and data-->
</graphml>
```

Рис. 4. Минимальный GraphML-документ без ссылки на схему

3. ТОПОЛОГИЯ

Опишем как топология графа (его элементы) представляется на языке GraphML.

Вначале еще раз рассмотрим документ, приведенный на рис. 1. Отдельный граф представлен на языке GraphML посредством элемента <graph>. Элемент <graphml> может содержать произвольное число элементов <graph>.

Каждый граф в GraphML может являться смешанным, другими словами, он может содержать одновременно как ориентированные, так и неориенти-

рованные ребра. Если при объявлении ребра его ориентированность не определена, то применяется ориентированность, заданная для ребер графа по умолчанию. Ориентированность ребер графа, присваиваемая по умолчанию, задается с помощью XML-атрибута edgedefault элемента <graph>. Данный XML-атрибут может принимать одно из двух значений: directed (ориентированный) и undirected (неориентированный). Значение по умолчанию должно быть задано обязательно. Дополнительно с помощью атрибута id, графу может быть присвоен некоторый идентификатор. Этот идентификатор нужен тогда, когда на данный граф требуется организовать ссылку.

Вершины графа представляются в виде списка элементов <node>. Каждая вершина имеет уникальный в пределах данного документа идентификатор, который задается с помощью атрибута id.

Множество ребер представляется в виде списка элементов <edge>. Каждое ребро имеет две инцидентные вершины, задаваемые с помощью XML-атрибутов source и target. Значения атрибутов source и target должны содержать идентификаторы вершин, определенных в том же документе что и ребро. Ребра с одной инцидентной вершиной, так называемые петли, определяются с помощью одинаковых значений, заданных в атрибутах source и target. Дополнительный XML-атрибут directed определяет ориентированность ребра, заданную в явном виде. Значение true задает ориентированное ребро, а false – неориентированное. Если ориентированность в явном виде не задана, то применяется ориентированность, заданная по умолчанию при объявлении графа. Дополнительно с помощью XML-атрибута id может быть задан идентификатор ребра. XML-атрибут id задается, когда необходимо организовать ссылку на данное ребро.

Вершины и ребра упорядочиваются произвольным образом, и язык не требует перечислять все вершины до перечисления всех рёбер. Ясно, что память, требуемая для сохранения на языке GraphML графа с n вершинами и m рёбрами, составляет $O(n + m)$.

4. АТРИБУТЫ

В предыдущем разделе мы обсудили порядок описания топологии графа на языке GraphML. Хотя имеется целый ряд приложений, для которых информации о топологии графов может быть достаточно, большинство приложений работает с графовыми моделями, обладающими дополнительной

информацией. Поэтому в языке GraphML предусмотрены средства для включения различной информации в описание графа.

С помощью механизма расширения, который называется *GraphML-атрибуты,* для элементов графа может быть задана дополнительная информация простого типа. Простой тип подразумевает, что данные ограничены скалярными величинами, например, числами и строками. Расширение GraphML-атрибуты уже включено в файл

http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd

и таким образом заголовок графа со скалярными атрибутами может иметь вид, рассмотренный в разд. 2.

GraphML-атрибуты не следует путать с XML-атрибутами, которые имеют совсем другой смысл. GraphML-атрибуты добавляют информацию к графам, множествам графов или частям графов, а XML-атрибуты добавляют информацию к XML-элементам.

В большинстве случаев дополнительная информация может и должна прикрепляться к элементам GraphML с помощью механизма GraphML-атрибутов, описанного здесь. Это гарантирует читаемость описаний графов другими GraphML-парсерами. Если же необходим более сложный формат данных в качестве атрибутов, можно воспользоваться механизмом расширения языка GraphML произвольными данными в четко определённых местах. Осуществление таких расширений мы описали в другой статье [2].

GraphML-атрибуты рассматриваются как частичные функции, приписывающие элементам графа значения атрибутов, которые, как правило, имеют один и тот же тип. Например, веса ребер могут рассматриваться как функция из множества ребер E в множество вещественных чисел R:

$$\text{weight}: E \to R .$$

Другой пример – это формы изображения вершин, которые можно представить в виде функции из множества вершин V в множество слов над заданным алфавитом Σ:

$$\text{shape}: V \to \Sigma^* .$$

Для добавления указанных функций к элементам графа следует использовать key/data-механизм языка GraphML. Элемент <key>, размещаемый в начале документа, декларирует новую функцию разметки; более точно элемент <key> специфицирует для функции её идентификатор, имя, области

определения и значения. Сами же значения функции определяются с помощью <data> элементов.

Декларация всех функций разметки в самом начале документа позволяет парсерам построить подходящие структуры данных в начале процесса разбора. Также парсеры могут распознавать ситуации, когда необходимые данные опущены. На рис. 5 приведен пример применения key/data-механизма. Здесь функция веса описывается строкой

<key id="d1" for="edge" attr.name="weight" attr.type="double"/>.

Элемент <key> имеет XML-атрибут с именем for, который специфицирует область определения функции. Для атрибута for в качестве значений можно указать graph, node, edge, graphml, а также имена других типов элементов графа, рассмотренные в разд. 2. Данный XML-атрибут может также получить значение all, что означает, что данные пометки могут помечать любые элементы графа. Атрибут for вместе с уникальным атрибутом id являются обязательными для элементов <key>. Расширение GraphML предоставляет еще два атрибута для <key>: это атрибут attr.name, который определяет имя функции и используется парсером для распознавания соответствующих данных, а также атрибут attr.type, который задает область значения функции и может принимать в качестве значений имена типов[3] boolean, int, long, float, double, или string.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
<key id="d0" for="node" attr.name="shape" attr.type="string">
<default>circle </default>
</key>
<key id="d1" for="edge" attr.name="weight" attr.type="double"/>
<graph id="G" edgedefault="undirected">
<node id="n0">
<data key="d0">square</data>
</node>
<node id="n1"/>
<node id="n2">
<data key="d0">oval</data>
</node>
<node id="n3">
```

[3] Эти типы определены в соответствии с аналогичными типами в языке Java.

```
<data key="d0">square</data>
</node>
<node id="n4"/>
<node id="n5">
<data key="d0">oval</data>
</node>
<node id="n6"/>
<edge id="e0" source="n0" target="n2">
<data key="d1">1.0</data>
</edge>
<edge id="e1" source="n0" target="n1">
<data key="d1">1.0</data>
</edge>
<edge id="e2" source="n1" target="n3">
<data key="d1">2.0</data>
</edge>
<edge id="e3" source="n3" target="n2"/>
<edge id="e4" source="n2" target="n4"/>
<edge id="e5" source="n3" target="n5"/>
<edge id="e6" source="n5" target="n4">
<data key="d1">1.1</data>
</edge>
</graph>
</graphml>
```

Рис. 5. Представление атрибутированного графа, в котором ребра имеют веса, а вершины форму

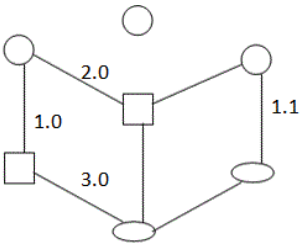


Рис. 6. Изображение атрибутированного графа, представленного в виде GraphML-документа на рис. 5

Обычный парсер, обрабатывая веса ребер, как правило, после обработки описания функции веса инициализирует внутреннюю структуру данных, которая сохраняет двойное вещественное с каждым ребром. В отличие от него другой парсер, который не знает о функции пометок ребер весами или не нуждается в этой функции, будет просто игнорировать ассоциированные <data> элементы.

Значения функции данных на некотором элементе графа (или, что то же самое, значение GraphML-атрибута у данного элемента графа) задается с помощью элемента <data>, вложенного в описание данного элемента. Например, фрагмент документа

```
<edge id="e0" source="n0" target="n2">
<data key="d1">1.0</data>
</edge>
```

определяет значение 1.0 в качестве веса для заданной дуги <edge>. Элемент data имеет XML-атрибут <key>, который ссылается на идентификатор GraphML-атрибута. Значение GraphML-атрибута задается текстовым содержимым элемента <data>. Это значение должно иметь тип, объявленный в соответствующем элементе <key>.

Для GraphML-атрибутов можно определить значение по умолчанию. Содержимое элемента default определяет текстовое значение по умолчанию. Например, фрагмент документа

```
<key id="d0" for="node" attr.name="shape" attr.type="string">
<default>circle </default>
</key>
```

определяет значение circle в качестве значения по умолчанию для атрибута форма у вершин графа.

Могут быть такие GraphML-атрибуты, которые определены, но не объявлены с помощью элемента <data>. Если значение по умолчанию определено для данного GraphML-атрибута, то тогда это значение применяется к соответствующему (входящему в домен GraphML-атрибута) элементу графа. В вышеприведенном примере значение не определено для вершины с идентификатором n1 и GraphML-атрибута с именем shape. Однако для данного GraphML-атрибута определено значение по умолчанию circle, которое будет присвоено данной вершине. Если же значение по умолчанию не задано, как это имеет место для GraphML-атрибута weight в вышеприведенном

примере, то значение GraphML-атрибута для такого элемента графа считается неопределенным. В вышеприведенном примере не определено значение GraphML-атрибута, задающего вес, у ребра с идентификатором e3.

5. ИНФОРМАЦИЯ ДЛЯ ПАРСЕРА

Помимо возможности задания атрибутов к базовым средствам языка относят еще одно расширение формата описания графа, называемое GraphML-Parseinfo. GraphML-Parseinfo делает возможным писать простые парсеры, основанные на дополнительной информации в GraphML-файлах. Это расширение уже включено в файл:

http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd

Таким образом, заголовок файла с описанием графа, использующим это расширение, может иметь тот же вид, что и рассмотренный ранее (в разд. 2).

Указанное расширение направлено на оптимизацию синтаксического разбора документа с помощью парсера за счет использования специальных метаданных, которые могут быть добавлены к некоторым GraphML-элементам с помощью XML-атрибутов. Имеется два вида метаданных: информация о количестве элементов и информация о способе кодирования конкретных данных в документе. Например, для парсера, который сохраняет вершины и инцидентные ребра в форме массивов, может оказаться весьма полезной информация о количествах вершин и ребер в графе, а также степенях его вершин. Все XML-атрибуты, задающие указанные метаданные, имеют префикс parse.

Для первого вида метаданных, связанного с информацией о количеством элементов, определены следующие XML-атрибуты для элемента <graph>: XML-атрибут parse.nodes задает количество вершин в графе, XML-атрибут parse.edges определяет количество ребер в графе, XML-атрибут parse.maxindegree определяет максимальное количество ребер, входящих в одну вершину графа, а XML-атрибут parse.maxoutdegree определяет максимальное количество ребер, исходящих из одной вершины графа. Кроме того, для элемента <node> введены новые XML-атрибуты parse.indegree и parse.outdegree, котрые определяют для данной вершины количества входящих и исходящих ребер.

Для метаданных, связанных со способом кодирования, определены XML-атрибуты parse.nodeids, parse.edgeids и parse.order для элемента

<graph> со следующей семантикой. Если XML-атрибут parse.nodeids имеет значение canonical, все вершины получают идентификатор вида nX, где X обозначает количество элементов <node>, предшествующих данному элементу. Другое возможное значение данного XML-атрибута равно free. Аналогичным образом действует XML-атрибут parse.edgeids, который задает вид идентификатора для узлов. Отличие состоит только в том, что этот идентификатор имеет вид eX. XML-атрибут parse.order определяет порядок, в котором узлы и ребра располагаются в документе. При значении, равном nodesfirst, все элементы <node> располагаются раньше элементов <edge>. При значении, равном adjacencylist, объявление вершины предшествует объявлению смежных ей вершин. Для значения free порядок следования вершин и ребер никак не ограничивается.

Пример на рис. 7 иллюстрирует использование указанного вида информации для парсера.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
<graph id="G" edgedefault="directed"
                parse.nodes="11"
                parse.edges="12"
                parse.maxindegree="2"
                parse.maxoutdegree="3"
                parse.nodeids="canonical"
                parse.edgeids="free"
                parse.order="nodesfirst">
        <node    id="n0"   parse.indegree="0"     parse.outdegree="1"/>
        <node    id="n1"   parse.indegree="0"     parse.outdegree="1"/>
        <node    id="n2"   parse.indegree="2"     parse.outdegree="1"/>
        <node    id="n3"   parse.indegree="1"     parse.outdegree="2"/>
        <node    id="n4"   parse.indegree="1"     parse.outdegree="1"/>
        <node    id="n5"   parse.indegree="2"     parse.outdegree="1"/>
        <node    id="n6"   parse.indegree="1"     parse.outdegree="2"/>
        <node    id="n7"   parse.indegree="2"     parse.outdegree="0"/>
        <node    id="n8"   parse.indegree="1"     parse.outdegree="3"/>
        <node    id="n9"   parse.indegree="1"     parse.outdegree="0"/>
        <node    id="n10"  parse.indegree="1"     parse.outdegree="0"/>
        <edge     id="edge0001"     source="n0"    target="n2"/>
        <edge     id="edge0002"     source="n1"    target="n2"/>
        <edge     id="edge0003"     source="n2"    target="n3"/>
```

```
        <edge    id="edge0004"    source="n3"    target="n5"/>
        <edge    id="edge0005"    source="n3"    target="n4"/>
        <edge    id="edge0006"    source="n4"    target="n6"/>
        <edge    id="edge0007"    source="n6"    target="n5"/>
        <edge    id="edge0008"    source="n5"    target="n7"/>
        <edge    id="edge0009"    source="n6"    target="n8"/>
        <edge    id="edge0010"    source="n8"    target="n7"/>
        <edge    id="edge0011"    source="n8"    target="n9"/>
        <edge    id="edge0012"    source="n8"    target="n10"/>
</graph>
</graphml>
```

Рис. 7. Использование GraphML-Parseinfo метаданных

СПИСОК ЛИТЕРАТУРЫ

1. Евстигнеев В. А., Касьянов В. Н. Толковый словарь по теории графов в информатике и программировании. - Новосибирск: Наука, 1999.
2. Касьянов В.Н. Язык представления графов GraphML: дополнительные возможности // Информатика в науке и образовании. – Новосибирск, 2012. – С. 23–46.
3. Касьянов В. Н., Евстигнеев В. А. Графы в программировании: обработка, визуализация и применение. ─ СПб.: БХВ-Петербург, 2003.
4. Касьянов В. Н., Касьянова Е. В. Визуализация графов и графовых моделей. — Новосибирск: Сибирское Научное Издательство, 2010.
5. Borgatti S.P., Everett M.G., and Freeman L.C. UCINET 6.0 / Analytic Technologies, 1999.
6. Brandes U. GraphML progress report: structural layer proposal / Brandes U., Eiglsperger M. et al. / Lecture Notes in Comput. Sci. ─2002. ─Vol. 2265. ─ P. 501–512. ─ (Proc. 9th Int. Symp. Graph Drawing GD'2001).
7. Brandes U., Eiglsperger M., Lerner J. GraphML Primer. http://graphml.graphdrawing.org/primer/graphml-primer.html#EXT
8. Brandes U. Graph markup language (GraphML)/ Brandes U., Eiglsperger M. et al. http://www.cs.brown.edu/~rt/gdhandbook/chapters/graphml.pdf
9. Brandes U.. Graph data format workshop report / Brandes U., Marshall M.S., North S.C. / Lecture Notes in Comput. Sci. – 2001. -Vol. 1984. – P. 410 – 418. - (Proc. 8th Int. Symp. Graph Drawing GD'2000).
10. De Nooy W., Mrvar A., and Batagelj V. Exploratory social network analysis with Pajek. - Cambridge University Press, 2005.
11. Di Battista G. et al. Graph Drawing: Algorithms for the Visualization of Graphs/ Di Battista G., Eades P., Tamassia R., Tollis I.G. - Prentice Hall, 1999.

12. GML. The Graph Modeling Language File Format. http://www.infosun.fmi.uni-passau.de/Graphlet/GML/.
13. Sugiyama K. Methods for visual understanding of hierarchical system structures / Sugiyama K., Tagawa S., Toda M / IEEE Transactions on Systems, Man and Cybernetics. ─1981. ─Vol. 11, N 2. ─P. 109–125.
14. Tsvetovat M., Reminga J., and Carley K. Dynetml: interchange format for rich social network data / NAACSOS Conference. - Pittsburgh, PA, 2003.
15. W3C. Scalable Vector Graphics. http://www.w3.org/TR/SVG/.
16. W3C. SOAP. http://www.w3.org/TR/soap12-part0/.
17. Winter A. Exchanging Graphs with GXL / Lecture Notes in Comput. Sci. ─2002. ─ Vol. 2265. ─P. 485–500. ─ (Proc. 9th Int. Symp. Graph Drawing GD'2001).

В.Н. Касьянов

ЯЗЫК ПРЕДСТАВЛЕНИЯ ГРАФОВ GRAPHML: ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ[1]

ВВЕДЕНИЕ

Современное программирование нельзя представить себе без теоретико-графовых методов и алгоритмов [3]. Широкая применимость графов связана с тем, что они являются очень естественным средством объяснения сложных ситуаций на интуитивном уровне. Эти преимущества представления сложных структур и процессов графами становятся еще более ощутимыми при наличии хороших средств их визуализации [4, 11].

Ясно, что инструменты визуализации информации на основе графовых моделей, подобно всем другим инструментам, имеющим дело со структурированными данными, нуждаются в сохранении и передаче графов и ассоциированных с ними данных.

Поэтому неслучайно в 2000 году наблюдательный комитет симпозиума по рисованию графов (Graph Drawing Steering Committee) организовал рабочее совещание по форматам обмена графовыми данными, состоявшееся в г. Вильямсбурге в рамках 8-го симпозиума по рисованию графов (GD-2000) [9]. Как следствие была сформирована неформальная рабочая группа по выработке основанного на языке XML формата обмена графами GraphML, который, в частности, был бы пригоден для обмена данными между инструментами рисования графов и другими приложениями и, в конечном счете, лег бы в основу стандарта описания графов.

Первый отчет по языку вышел в 2001 году [6]. С тех пор язык был расширен в части поддержки основных типов атрибутов и в части включения информации для использования парсерами (синтаксическими анализаторами) [7, 8]. Ведется работа по включению абстрактной информации для описания топологии графа и шаблонов, с помощью которых эту информацию можно преобразовать в различные графические форматы. Программное

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

обеспечение для поддержки работы с GraphML также находится в стадии разработки.

Благодаря XML-синтаксису GraphML может использоваться в комбинации с другими форматами, основанными на XML. С другой стороны, свой собственный механизм расширения позволяет прикреплять <data> метки со сложным содержимым (возможно, требуемый для исполнения с другими моделями XML содержимого) элементов GraphML. Примерами таких меток со сложным содержимым является так называемый механизм SVG (Scalable Vector Graphics) [15], описывающий появление вершин и дуг в изображении. С другой стороны, GraphML может интегрироваться в другие приложения, например, в SOAP сообщения [16].

Простейший способ прочитать или записать GraphML файлы состоит в использовании программного обеспечения, обрабатывающего графы, которое поддерживает данный формат. GraphML является стандартным входным/выходным форматом для системы visone [5] и для графового редактора yEd компании yWorks [24]. Помимо них имеется целый ряд программных инструментов и библиотек, которые либо импортируют, либо экспортируют (либо одновременно и то, и другое) GraphML, включая Pajek [13], ORA [12] и JUNG [16]. Если нужно реализовать привычный GraphML-ридер, можно просто использовать один из многих доступных XML-парсеров и адаптировать его вручную под свои цели.

Базовые средства, образующие ядро языка GraphML, позволяют достаточно адекватно представлять графовые объекты в большинстве приложений. Базовая графовая модель языка охватывает графы[2], которые могут содержать ребра, дуги, петли, кратные дуги, кратные ребра и пометки (атрибуты) вершин и ребер. Множества пометок могут кодировать, например, различные семантические свойства объектов, представленных в виде данной графовой модели, или различные геометрические свойства элементов графа в заданном его изображении на плоскости.

Данная статья продолжает статью [2], в которой представлены базисные средства GraphML, и посвящена дополнительным возможностям языка.

Статья начинается с рассмотрения средств, связанных с расширением базовой графовой модели языка за счет введения таких дополнительных понятий для графовой топологии, как вложенные графы, гиперграфы и порты. Далее описываются два основных способа расширения языка GraphML: за счет добавления новых атрибутов к GraphML-элементам и

[2] Здесь и ниже мы без определения используем стандартные понятия из теории графов (см., например, [1]).

путем расширения содержимого элементов <data>. Статья завершается рассмотрением использования XSLT-механизма для преобразования GraphML-документов.

1. РАСШИРЕНИЯ БАЗОВОЙ ГРАФОВОЙ МОДЕЛИ

Для некоторых конкретных приложений базовая графовая модель (см., например, [2]) может оказаться слишком ограниченной и не позволять адекватно моделировать данные прикладной программы. Поэтому авторы предусмотрели в языке расширения данной базовой графовой модели такими конструкциями, как вложенные графы, гиперребра и порты; эти конструкции рассмотрены ниже. Заметим, что все эти дополнительные GraphML-элементы наряду с базовыми могут быть заданы в качестве областей определения функций данных, т.е. в качестве значений для атрибутов <key> (см. [2]).

1.1. Вложенные графы

Помимо атрибутированных смешанных графов, язык GraphML поддерживает и вложенные графы, т.е., графы в которых вершины иерархически упорядочены. Иерархия выражается через структуру GraphML-документа. Вершина в GraphML-документе может иметь элемент <graph>, который содержит вершины, иерархически вложенные в данную вершину.

На рис. 1 и 2 приводится пример вложенного графа и соответствующий ему GraphML-документ. Отметим, что в изображении графа иерархия выражена с помощью включения одного изображения в другое, т.е. вершина *u* находится в иерархии ниже вершины *v*, если графическое представление вершины *u* целиком расположено внутри графического представления вершины *v*.

Ребра, соединяющие две вершины, находящиеся во вложенных графах, должны быть объявлены в графе, который является в иерархии предком обоих вершин.

Обратите внимание, что в рассмотренном выше примере именно так и сделано. Объявление ребра между вершиной n3 и вершиной n2 в графе G1 было бы неправильно, а объявление их в графе G0 – правильно.

В случае неоднозначности, когда в иерархии существует нескольких общих предков, рекомендуется, но не требуется размещать объявление ребра в наименьшем общем предке конечных вершин данного ребра.

Язык GraphML включает элемент, названный <locator>, который делает возможным определять содержимое части данного документа в другом файле. Более точно, элементы <graph> и <node> могут содержать элемент <locator>, чей атрибут xlink:href указывает на файл, в котором определено содержимое данного элемента <graph> (соответственно, данного элемента <node>). В частности, если элемент <graph> или <node> содержит <locator>, то этот граф <graph>, соответственно, данная вершина <node>, не содержит других элементов. Например, следующий фрагмент документа, который является модифицированной версией документа на рис. 2,

```
<graph id="G0" edgedefault="undirected">
  <node id="n1">
    <graph id="G1" edgedefault="undirected">
      <locator xlink:href="content_of_G1.graphml"/>
    </graph>
  </node>
...
</graph>
```

сообщает парсеру, что содержимое графа с id="G1" определено в содержимом файла G1.graphml. Аналогичным образом содержимое элементов <node> может отсылать к другому файлу с помощью элементов <locator>.

Предполагается, что приложения, которые не поддерживают вложенность графов, могут просто игнорировать все те вершины, которые не принадлежат графу верхнего уровня, а также игнорировать все те ребра, у которых хотя бы одна конечная вершина не принадлежит графу верхнего уровня.

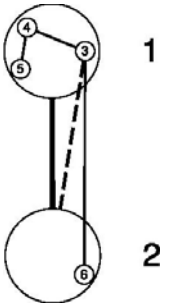


Рис. 1. Вложенный граф

```
<graphml>
  <graph id="G0" edgedefault="undirected">
    <node id="n1">
      <graph id="G1" edgedefault="undirected">
        <node id="n3"/>
        <node id="n4"/>
        <node id="n5"/>
        <edge source="n3" target="n4"/>
        <edge source="n4" target="n5"/>
      </graph>
    </node>
    <node id="n2">
      <graph id="G2" edgedefault="undirected">
        <node id="n6"/>
      </graph>
    </node>
    <edge source="n1" target="n2"/>
    <edge source="n3" target="n2"/>
    <edge source="n3" target="n6"/>
  </graph>
</graphml>
```

Рис. 2. GraphML-документ, описывающий вложенный граф с рис. 1

1.2. Гиперребра

Понятие гиперребра является обобщением понятия обычного ребра в том смысле, что это такое ребро, которое связывает не обязательно только две конечные вершины, но и выражает связь между произвольным числом конечных вершин. Гиперребра объявляются в GraphML-документах с помощью элемента hyperedge. Каждой конечной вершине гиперребра соответствует свой элемент endpoint, входящей в данное гиперребро. Элемент endpoint должен иметь XML-атрибут node, который содержит идентификатор вершины в документе.

Пример гиперграфа и его представления изображены на рис. 3 и 4. Данный гиперграф содержит два гиперребра и два обычных ребра. Гиперребра здесь изображены в виде соединяющихся кривых, а ребра в виде прямых линий.

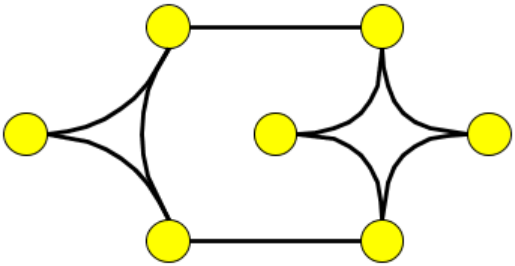


Рис. 3. Пример гиперграфа

Заметим, что ребра можно специфицировать либо элементами <edge>, либо элементами <hyperedge>, содержащими ровно по два элемента <endpoint>. Понятно, что второй способ в большей степени ориентирован на те приложения, которые могут обрабатывать гиперграфы. Элементы <endpoint> имеют факультативный атрибут, называемый типом, который может принимать значения in, out и undir и имеет значение undir по умолчанию.

Предполагается, что приложения, которые не могут обрабатывать гиперребра, будут их просто игнорировать.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
  <graph id="G" edgedefault="undirected">
    <node id="n0"/>
    <node id="n1"/>
    <node id="n2"/>
    <node id="n3"/>
    <node id="n4"/>
    <node id="n5"/>
    <node id="n6"/>
    <hyperedge>
      <endpoint node="n0"/>
      <endpoint node="n1"/>
      <endpoint node="n2"/>
    </hyperedge>
    <hyperedge>
      <endpoint node="n3"/>
```

```
      <endpoint node="n4"/>
      <endpoint node="n5"/>
      <endpoint node="n6"/>
    </hyperedge>
    <hyperedge>
      <endpoint node="n1"/>
      <endpoint node="n3"/>
    </hyperedge>
    <edge source="n0" target="n4"/>
  </graph>
</graphml>
```

Рис. 4. Представление гиперграфа с рис. 3

1.3. Порты

Вершины могут специфицировать различные логические точки для подключения ребер и гиперребер. Такие точки подключения называются *портами*. В качестве аналогии, можно рассматривать граф как материнскую плату, когда вершины представляют интегрированные схемы, а ребра – провода. Тогда контакты интегрированных схем соответствуют портам вершин.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
  <graph id="G" edgedefault="directed">
    <node id="n0">
      <port name="North"/>
      <port name="South"/>
      <port name="East"/>
      <port name="West"/>
    </node>
    <node id="n1">
      <port name="North"/>
      <port name="South"/>
      <port name="East"/>
      <port name="West"/>
    </node>
    <node id="n2">
      <port name="NorthWest"/>
      <port name="SouthEast"/>
```

```
    </node>
    <node id="n3">
      <port name="NorthEast"/>
      <port name="SouthWest"/>
    </node>
    <edge source="n0" target="n3"
        sourceport="North" targetport="NorthEast"/>
    <hyperedge>
      <endpoint node="n0" port="North"/>
      <endpoint node="n1" port="East"/>
      <endpoint node="n2" port="SouthEast"/>
    </hyperedge>
  </graph>
</graphml>
```

Рис. 5. Представление графа с портами

Порты вершины объявляются с помощью элементов <port>, являющимися детьми по отношению к соответствующему элементу <node>. Порты могут быть вложенными, т.е., они могут содержать внутри себя другие элементы <port>. Каждый элемент <port> должен иметь XML-атрибут name, который является идентификатором данного порта. Элемент <edge> имеет необязательные XML-атрибуты sourceport и targetport, которые задают для ребра начальный и конечный порты, соответственно. Аналогично элемент <endpoint> имеет необязательный XML-атрибут port. Пример GraphML-документа с портами показан на рис. 5.

Предполагается, что те приложения, которые не могут обрабатывать порты, будут их просто игнорировать.

2. РАСШИРЕНИЕ GRAPHML

Язык GraphML спроектирован как легко расширяемый. Базовые средства языка GraphML (см., например, [2]) позволяют описать топологию графа и простые атрибуты его элементов. Для представления более сложных прикладных данных язык GraphML должен быть расширен.

Ниже мы рассмотрим два основных способа расширения языка GraphML: с помощью добавления новых атрибутов к GraphML-элементам (см. разд. 2.1) и путем расширения содержимого элементов <data> за счет разрешения им содержать элементы из других XML-языков (см. разд. 2.2).

Расширения GraphML должны быть заданы в XML-схеме. Схема, в которой определены расширения, может быть порождена из схемы GraphML-документа с помощью стандартного механизма, похожего на механизм, который используется в XHTML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace=http://graphml.graphdrawing.org/xmlns
  xmlns=http://graphml.graphdrawing.org/xmlns
  xmlns:xlink=http://www.w3.org/1999/xlink
  xmlns:xs=http://www.w3.org/2001/XMLSchema
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

<xs:import namespace=http://www.w3.org/1999/xlink
          schemaLocation="xlink.xsd"/>

<xs:redefine
  schemaLocation="http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd">
  <xs:attributeGroup name="node.extra.attrib">
    <xs:attributeGroup ref="node.extra.attrib"/>
    <xs:attribute ref="xlink:href" use="optional"/>
  </xs:attributeGroup>
</xs:redefine>

</xs:schema>
```

Рис. 6. Файл graphml+xlink.xsd: определение XML-схемы, которое расширяет GraphML язык путем добавления атрибута xlink:href элементу <node>

2.1. Добавление XML-атрибутов

В большинстве случаев, дополнительная информация может (и должна) быть связана с GraphML-элементами с помощью GraphML-атрибутов (см. [2]), что гарантирует совместимость с другими GraphML-парсерами. Однако в ряде случаев более удобно использовать XML-атрибуты. Предположим, у нас имеется парсер, который умеет обрабатывать XLink-атрибут href и корректно интерпретировать его как URL. Предположим, что нужно сохранить в виде GraphML-документа граф, вершины которого представляют

собой WWW-страницы. Вершина могла бы сослаться на ассоциированную страницу путем сохранения в элементе <node> в качестве значения атрибута xlink:href URL-ссылки на соответствующую страницу:

<node id="n0" xlink:href="http://graphml.graphdrawing.org"/>.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns=http://graphml.graphdrawing.org/xmlns
xmlns:xlink=http://www.w3.org/1999/xlink
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
graphml+xlink.xsd">
<graph edgedefault="directed">
<node id="n0" xlink:href="http://graphml.graphdrawing.org"/>
<node id="n1" />
<edge source="n0" target="n1"/>
</graph>
</graphml>
```

Рис. 7. Документ, который можно верифицировать с XSD, показанным на рис. 6. Заметим, атрибут schemaLocation у элемента <graphml> ссылается на файл graphml+xlink.xsd.

Строка

http://graphml.graphdrawing.org

могла бы быть также сохранена в элементе <data>, содержащемся в вершине n0. Однако если сохранять эту строку в качестве значения атрибута xlink:href, то его семантика (т.е. то, что это — URL) становится более очевидной.

Элемент <node>, как он представлен выше, мог стать неверифицируемым в ядре GraphML, поскольку в нем не определен атрибут xlink:href у элемента <node>. Для добавления XML-атрибутов к GraphML-элементам требуется расширить язык GraphML. Это расширение может быть осуществлено с помощью XML-схемы.

Документ на рис. 6 является определением XML-схемы, которая расширяет язык GraphML путем добавления атрибута xlink:href вершине <node>.

Она имеет элемент <schema> в качестве своего корневого элемента (всякое определение XML-схемы обладает этим свойством). Элемент <schema> имеет несколько атрибутов. Строчка

targetNamespace=http://graphml.graphdrawing.org/xmlns

специфицирует, что язык, определенный этим документом, является языком GraphML. Следующие три строчки специфицируют пространство имен по умолчанию (идентифицированное адресом URL для GraphML) и префиксы пространств имен для XLink и XML-схемы. Атрибуты elementFormDefault и attributeFormDefault несущественны в данном примере.

Инструкция

```
<xs:importnamespace=http://www.w3.org/1999/xlink
                    schemaLocation="xlink.xsd"/>
```

предоставляет доступ к пространству имен XLink (предполагается, что определение схемы XLink расположено в файле xlink.xsd).

Само описанное расширение представлено в элементе <redefine>. Атрибут

schemaLocation=http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd

у элемента <redefine> специфицирует файл (часть файла), который переопределяется. Фрагмент документа

```
<xs:attributeGroup name="node.extra.attrib">
  <xs:attributeGroup ref="node.extra.attrib"/>
  <xs:attribute ref="xlink:href" use="optional"/>
</xs:attributeGroup>
```

расширяет атрибутную группу, названную node.extra.attrib, которая (по спецификации ядра GraphML) является пустым множеством, но включена в список атрибутов элемента <node>. После переопределения эта атрибутная группа в дополнение к старому содержимому получает один новый атрибут, а именно xlink:href. Этот дополнительный атрибут объявляется в качестве факультативного для элемента <node>.

Аналогично существованию атрибутной группы node.extra.attrib для элемента <node> имеются соответствующие атрибутные группы для всех GraphML-элементов. Эти атрибутные группы пусты в определении ядра GraphML, но могут быть расширены аналогичным образом.

Схема graphml+xlink.xsd может использоваться для валидации документа, показанного на рис. 7.

Авторы языка объясняют свое решение всегда добавлять старое содержимое к вновь определенной атрибутной группе стремлением к тому, чтобы сделать возможным существование более одного определения схемы для расширения одной и той же атрибутной группы.

Запоминание дополнительной информации непосредственно в атрибутах GraphML-элементов, как было проиллюстрировано выше, может показаться более предпочтительным, чем ее сохранение в элементах <data>, как это предусматривают базовые средства языка [2]. По крайней мере, можно заметить, что при таком подходе требуется меньше символов. Однако такое специфицированное пользователем расширение имеет свою цену: поскольку эти нестандартные атрибуты не определятся элементами <key>, GraphML-парсеры будут не в состоянии их обрабатывать.

2.2. Добавление структурного содержимого

В некоторых случаях было бы удобно использовать другие XML-языки для представления данных в GraphML-документах. Например, пользователь может пожелать сохранить изображения вершин, записанные в SVG, как это происходит в следующем фрагменте документа.

```
...
xmlns:svg=http://www.w3.org/2000/svg
...
<node id="n0" >
  <data key="k0">
    <svg:svg width="4cm" height="8cm" version="1.1">
      <svg:ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm" />
    </svg:svg>
  </data>
</node>
...
```

Понятно, что атрибуты <svg> и <ellipse> можно также сохранять с помощью функций данных, как это предусматривают базовые средства языка

[2]. Однако представление, приведенное выше, намного более удобно, поскольку приложения, содержащие такое представление, могут использовать существующие парсеры или вьюеры для SVG-изображений.

Язык GraphML можно расширить для верификации документов такого типа. Произвольные элементы могут добавляться к содержимому <data>, но только к элементам <data>, а ядро GraphML не должно быть изменено. Это решение принято для того, чтобы гарантировать способность парсеров всегда понимать структурную часть GraphML-документов и игнорировать возможно неизвестное содержимое элементов <data>.

Рис. 8 содержит определение XML-схемы, которое добавляет SVG-элементы к содержимому <data>.

Схема на рис. 8 подобна примеру на рис. 6. Во-первых, сделаны декларации пространств имен. Затем импортировано пространство имен SVG. Как и прежде, расширение осуществлено в элементе <redefine>. Внутри этого элемента сложный тип data-extension.type расширен SVG-элементом <svg>. Тип data-extension.type является базисным типом для элементов <data> и <default>. Этот тип имеет пустое содержимое в определении ядра GraphML, но может быть расширен произвольными XML-элементами.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="http://graphml.graphdrawing.org/xmlns"
  xmlns="http://graphml.graphdrawing.org/xmlns"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
>

<xs:import namespace="http://www.w3.org/2000/svg"
           schemaLocation="svg.xsd"/>

<xs:redefine
  schemaLocation="http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd">
  <xs:complexType name="data-extension.type">
    <xs:complexContent>
      <xs:extension base="data-extension.type">
        <xs:sequence>
          <xs:element ref="svg:svg"/>
```

```
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:redefine>

</xs:schema>
```

Рис. 8. Файл graphml+svg.xsd: определение XML-схемы, которое добавляет SVG-элемент <svg:svg> к содержимому <data>

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
         xmlns:svg="http://www.w3.org/2000/svg"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
                             graphml+svg.xsd">
  <key id="k0" for="node">
    <default>
      <svg:svg width="5cm" height="4cm" version="1.1">
        <svg:desc>Default graphical representation for nodes
        </svg:desc>
        <svg:rect x="0.5cm" y="0.5cm" width="2cm" height="1cm"/>
      </svg:svg>
    </default>
  </key>
  <key id="k1" for="edge">
    <desc>Graphical representation for edges
    </desc>
  </key>
  <graph edgedefault="directed">
    <node id="n0">
      <data key="k0">
        <svg:svg width="4cm" height="8cm" version="1.1">
          <svg:ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm" />
        </svg:svg>
      </data>
    </node>
    <node id="n1" />
```

```
    <edge source="n0" target="n1">
      <data key="k1">
        <svg:svg width="12cm" height="4cm" viewBox="0 0 1200 400">
          <svg:line x1="100" y1="300" x2="300" y2="100"
          stroke-width="5" />
        </svg:svg>
      </data>
    </edge>
  </graph>
</graphml>
```

Рис. 9. Документ, который можно верифицировать с помощью определения XSD, приведенного на рис. 8. Следует заметить, атибут schemaLocation у <graphml> ссылается на graphml+svg.xsd

Таким образом, документы, верифицируемые по схеме с рис. 8, могут иметь элементы <data>, содержащие <svg>. Пример показан на рис. 9. Вершина с id n1 предполагает в качестве графического изображения по умолчанию изображение с ключом k0. Приведенный выше пример демонстрирует полезность пространств имен XML. Здесь имеется два различных элемента <desc>: один в пространстве имен GraphML, а другой в пространстве имен SVG. С помощью различных пространств имен разрешаются конфликты, которые могут возникать из-за элементов, имеющих одинаковые имена в разных XML-языках.

Следует заметить, что имеется не только возможность использовать другие XML-языки (подобно SVG) в GraphML-документах. Сам язык GraphML также можно использовать для представления графовых данных в других расширяемых XML-языках, таких как SVG и XHTML. Данная возможность модульного комбинирования XML-языков гарантирует переиспользуемость парсеров и другого программного обеспечения. Например, SVG-вьюер может вызывать программные системы рисования графов для построения раскладок графов, которые будут сохраняться на языке GraphML внутри SVG-файла.

3. ПРЕОБРАЗОВАНИЕ GRAPHML

Весьма просто обеспечить доступ к изображениям графов на языке GraphML добавлением входных и выходных фильтров для существующего программного обеспечения. Однако разработчики решили, что механизм Extensible Stylesheet Language Transformations (XSLT) [22] предлагает более естественный способ использования XML-данных, в частности, когда результирующий формат некоторого вычисления снова основывается на XML. Отображения, которые переводит входные GraphML-документы в выходные, определяются в виде стилевых XSLT-страниц и могут использоваться отдельно, в качестве компонентов более крупных систем или в вебсервисах [10].

В основном преобразования определяются в стилевых страницах (style sheets), иногда называемых страницами трансформаций (transformation sheets); в них специфицировано, как входные XML-документы преобразуются в выходные XML-документы в процессе некоторого рекурсивного процесса поиска по образцу. В основе XML-документов лежит модель данных Document Object Model (DOM), некоторое дерево DOM-вершин, изображающее элементы, атрибуты, тексты и т.д., целиком сохраняемые в памяти. Рис. 10 иллюстрирует последовательность шагов XML-преобразования. Вначале XML-данные конвертируются в древовидное представление, которое затем используется для построения результирующего дерева, как это специфицировано в стилевой странице. Наконец, результирующее дерево представляется в виде XML-документа.

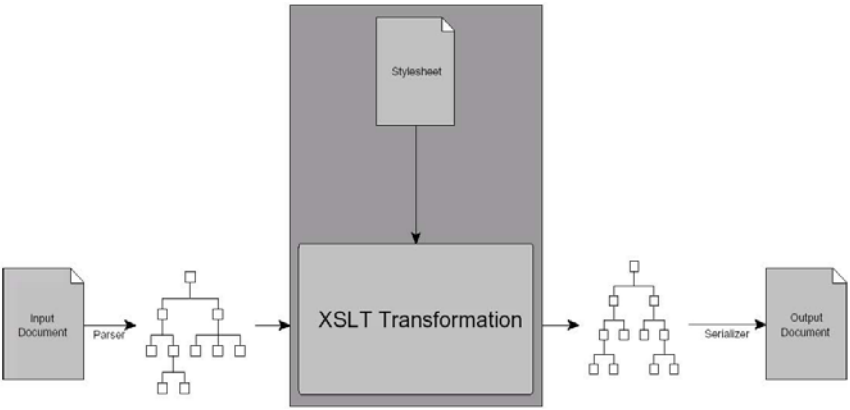


Рис. 10. Последовательность шагов некоторого XSLT-преобразования

По DOM-деревьям можно перемещаться с помощью XPath языка, являющегося подъязыком XSLT. В нем есть средства для представления путей по дереву документа от некоторой конкретной контекстной вершины (подобно дереву директории файловой системы) и нахождения множеств адресов тех его вершин, которые удовлетворяют заданным условиям. Например, если контекстной вершиной является элемент <graph>, все идентификаторы вершин могут адресоваться с помощью child::node/attribute::id, или node/@id в сокращенном виде. Могут использоваться предикаты для более точной спецификации того, какие части DOM-дерева выбираются; например, XPath-выражение edge[@source='n0']/data выбирает только тех <data>-детей ребер <edge>, которые исходят из вершины <node> с заданным идентификатором.

Процесс преобразования грубо можно описать следующим образом. Всякая стилевая страница состоит из некоторого списка шаблонов, каждый из которых имеет некоторый ассоциированный образец и некоторое тело шаблона, описывающее те действия, которые должны быть выполнены, и содержимое, которое должно записано на выходе. Начиная с корня, процессор выполняет поиск в глубину (в порядке записи документа) по DOM-дереву. Для каждой DOM-вершины, куда он попадает, процессор проверяет, есть ли шаблоны с подходящими образцами. Если подходящие шаблоны есть, то выбирается один из них, выполняются действия, содержащиеся в теле данного шаблона (потенциально с дальнейшим поиском по образцу для поддеревьев), и на этом процесс дальнейшего поиска в глубину для DOM-поддеревьев с корнем в данной DOM-вершине прекращается. Если же подходящих шаблонов нет, то процесс поиска в глубину по DOM-дереву продолжается для каждого из сыновей данной DOM-вершины. См. рис. 11 с примером трансформационной XSLT-страницы.

3.1. Средства и типы преобразований

Выразимость и полезность XSLT-преобразований лежит вне их исходной цели добавления некоторого стиля для входа. Ниже приводится обзор важных базисных концепций XSLT и описывается, как эти концепции частично применимы для формулирования продвинутых GraphML-преобразований, которые учитывают помимо структуры DOM-дерева лежащую в его основе комбинаторную структуру графа.

Поскольку язык GraphML проектируется в качестве общего формата, не ограниченного некоторой областью приложений, варианты использования XSLT весьма разнообразны. Однако авторы считают, что все разнообразие

трансформаций можно разделить на три основные категории (типа) в зависимости от реальных целей преобразований. При этом не исключено, что одна и та же трансформация может принадлежать более чем одной из рассмотренных категорий.

Внутренние преобразования. Хотя одной из целей проектирования GraphML является требование хорошо определенной интерпретации всех GraphML-файлов, имеются хорошо известные неоднозначности, связанные с возможностью различных GraphML-представлений для одного и того же графа, например, из-за того, что его вершины <node> и ребра <edge> могут появляться в произвольном порядке. Однако приложения могут требовать, чтобы их GraphML-вход удовлетворял определенным предусловиям, таким, как появление всех вершин <node> перед любым ребром <edge> для того, чтобы строить граф сразу на лету во время его чтения из входного потока.

Среди внутренних преобразований авторы выделяют следующие

- пре- и постпроцессирование GraphML-файла для достижения определенных условий, таких как переупорядочивание выделенных элементов или генерация для них уникальных идентификаторов;
- вставка значений по умолчанию там, где нет явных входов, например, задание ориентации ребер или значений по умолчанию для тэгов <data>,
- разрешение XLink-ссылок в распределенных графах;
- фильтрация ненужных тэгов <data>, которые не релевантны последующей обработке и могут быть опущены с целью сокращения стоимости коммуникации или памяти;
- конвертация между классами графов, например, элиминация гиперребер, подстановка вложенных графов или удаление кратных ребер.

```
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes" encoding="iso-8859-1"/>

  <xsl:template match="data|desc|key|default"/> <!--пустой шаблон-->

  <xsl:template match="/graphml">
    <graphml>
      <xsl:copy-of select="key|desc|@*"/>
```

```
      <xsl:apply-templates match="graph"/> <!—обработка graph(s) -->
    </graphml>
  </xsl:template>

  <xsl:template match="graph"> <!—шаблон переоопределения -->
    <graph>
      <xsl:copy-of select="key|desc|@*"/>
      <xsl:copy-of select="node"/> <!--вершины первыми -->
      <xsl:copy-of select="edge"/> <!--затем ребра -->
    </graph>
  </xsl:template>
</xsl:stylesheet>
```

Рис. 11. Пример страницы XSLT-преобразования, удаляющего элементы <data>, <desc>, <key> и <default> из документа и переупорядочивающего вершины и ребра таким образом, чтобы все элементы <node> появились до любого <edge> элемента

Конвертация форматов. Хотя в последнее время GraphML и подобные ему форматы, такие как GXL [23] и GML [15], все шире используются в различных областях, есть еще много приложений и сервисов, которые пока не в состоянии их обрабатывать. Чтобы быть совместимыми, форматы должны конвертироваться один в другой с как можно более полным сохранением информации.

При такой конвертации важно учитывать возможные структурные ошибки в терминах как графовых моделей, так и концепций, которые могут выражаться вовлеченными форматами и их поддержкой дополнительных данных. Конечно, чем ближе концептуальная связь между исходным и целевым форматом, тем , как правило, проще стилевые страницы.

Хотя могут требоваться различные типы таких конвертаций, с точки зрения авторов следует выделить в качестве наиболее важных следующие два случая использования такого типа преобразований.

- Конвертация в другой графовый формат. Ожидается, что GraphML будет использоваться во многих приложениях для архивирования атрибутированных графовых данных и в Web-сервисах для передачи аспектов графа. Хотя легко просто выдавать GraphML, стилевые страницы могут использоваться для конвертации в другие графовые форматы [8] и, таким образом, могут использоваться в транслирующих сервисах, подобных GraphEx [11].

- Экспорт в другие графовые форматы. Конечно, основанные на графах инструменты вообще и инструменты рисования графов в частности должны будут экспортировать в определенные графовые форматы для целей визуализации.

Указанные преобразования необязательно должны применяться файловому документу, они могут также выполняться в памяти приложениями, у которых возникает потребность экспорта в некоторый целевой формат. Следует заметить, что хотя XSLT-преобразования обычно используются для отображения между XML-документами, они могут также использоваться для генерации выходов, которые не являются XML.

Алгоритмические преобразования. Алгоритмические стилевые страницы появляются в преобразованиях, создающих фрагменты выходного документа, которые не соответствуют напрямую фрагментам входного документа, т.е. когда в исходном документе имеется структура, которая не выражается в разметке. Эта ситуация типична для GraphML-данных. Например, нельзя определить, содержит или нет заданный граф <graph> циклы, рассматривая язык разметки; некоторые алгоритмы должны применяться к представленному графу.

Чтобы понять потенциал алгоритмических стилевых страниц, разработчики языка реализовали ряд базовых графовых алгоритмов с использованием XSLT и рекурсивных шаблонов и пришли к выводу, что указанный механизм достаточно мощен для формулирования даже более продвинутых графовых алгоритмов. Например, можно использовать стилевую страницу для вычисления расстояний от единственной входной вершины до всех остальных вершин или реализовывать некоторый алгоритм раскладки, а затем прикрепить результаты к элементам <node>s в виде <data> меток.

3.2. Языковое связывание

Считается, что чистая XSLT-функциональность достаточно выразительна для решения даже более продвинутых связанных с GraphML проблем, чем рассмотренные выше. Однако она имеет ряд общих недостатков, среди которых авторы выделяют следующие.

- С ростом сложности проблем стилевые листы имеют тенденцию становиться все более многословными.
- Алгоритмы необходимо переформулировать в терминах рекурсивных шаблонов, и нет способа использовать уже существующие реализации.
- Вычисления могут плохо выполняться, особенно для больших входов.

Это часто возникает из-за излишнего обхода DOM-дерева и заглядывания вперед, связанных с алгоритмом подстановки шаблонов, встроенным в XSLT-процессор.

- Нет прямого доступа к системным службам, таким как функции данных или связи базовых данных.

Следовательно, большинство XSLT-процессоров разрешают интеграцию функций расширения, реализованных в XSLT или на некоторых других языках программирования. Обычно они по крайней мере поддерживают свой естественный язык. Например, система Saxon [17] может обращаться и использовать внешние Java-классы, поскольку сама она целиком написана на языке Java. В этом случае функции расширения являются методами Java-классов, доступных на пути класса, когда происходит исполнение преобразования и вызов внутри XPath-выражений. Обычно они являются статическими методами, что согласуется с идеей проектирования XSLT в декларативном стиле и без побочных эффектов. Однако XSLT разрешает создавать объекты и вызывать их методы путем связывания созданных объектов с XPath-переменными.

Архитектура, показанная на рис. 12, состоит из следующих трех слоев.

- Стилевая страница, которая создает экземпляр обертки и взаимодействует с ним.
- Класс обертки (реальное XSLT-расширение), которое конвертирует GraphML-разметку в обернутый графовый объект и обеспечивает результаты вычисления.
- Java-классы для графовых структур данных и алгоритмов.

Таким образом, обертка действует как промежуточное звено между графовым объектом и стилевой страницей. Обертка создает экземпляр графового объекта, соответствующего GraphML-описанию, и, например, применяет к нему алгоритм рисования графа. После выполнения он предоставляет результирующие координаты и другие данные по раскладке для того, чтобы стилевая страница вставила в XML (возможно, в GraphML-документ) результат преобразования или продолжила вычисления.

Подход, представленный здесь, является только одним из многих способов отображения файла с внешним описанием графа во внутреннее графовое представление. Отдельное приложение могло бы интегрировать GraphML-парсер, создать свое графовое представление в памяти, отдельное от XSLT, выполнить преобразование и представить результат в виде GraphML-выхода. Однако преимущество использования XSLT состоит в

том, что XSLT генерирует выход естественным способом, и что процесс генерации выхода поддается простому управлению.

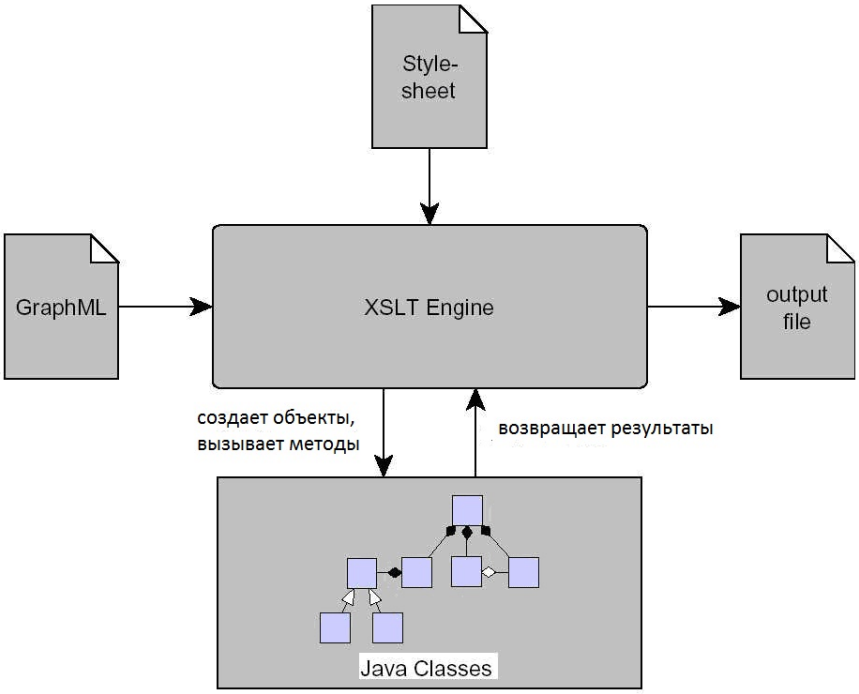


Рис. 12. Использование функций расширения в XSLT

XSL-преобразования образуют простой подход к обработке графов, представленных на языке GraphML. Они доказали свою полезность в различных таких областях применения, где целевой формат некоторого преобразования опять является GraphML-форматом или другим подобным форматом, и когда выходная структура не очень сильно отличается от входной.

Они являются даже достаточно мощными для спецификации преобразований, которые выходят за прямое отображение XML-элементов в другие XML-элементы или другие простые текстовые единицы. Однако такие продвинутые преобразования могут приводить к многословным стилевым страницам, которые сложны для сопровождения и в большинстве представ-

ляются неэффективными. Функции расширения проявили себя в качестве естественного способа борьбы с этими трудностями.

Поэтому механизм XSLT должен использоваться главным образом для выполнения структурных частей преобразования, таких как создание новых элементов или атрибутов, тогда как специализированные расширения более подходят для сложных вычислений, которые при использовании XSLT трудны для выражения или неэффективны для выполнения.

СПИСОК ЛИТЕРАТУРЫ

1. Евстигнеев В. А., Касьянов В. Н. Толковый словарь по теории графов в информатике и программировании. – Новосибирск: Наука, 1999.
2. Касьянов В.Н. Язык представления графов GraphML: базовые средства// Информатика в науке и образовании. – Новосибирск, 2012. – С. 7–22.
3. Касьянов В. Н., Евстигнеев В. А. Графы в программировании: обработка, визуализация и применение. – СПб.: БХВ-Петербург, 2003.
4. Касьянов В. Н., Касьянова Е. В. Визуализация графов и графовых моделей. – Новосибирск: Сибирское Научное Издательство, 2010.
5. Baur M., Benkert M., et all. Visone -software for visual social network analysis // Lect. Notes Comput. Sci. – 2002. – Vol. 2265. – P. 463-464. - (Proc. 9th Int. Symp. Graph Drawing GD'2001).
6. Borgatti S.P., Everett M.G., Freeman L.C. UCINET 6.0 // Analytic Technologies, 1999.
7. Brandes U., Eiglsperger M., Herman I., Himsolt M., Marshall M.S. GraphML progress report: structural layer proposal // Proc. 9th Int. Symp. Graph Drawing GD'2001. – Lect. Notes Comput. Sci. – 2002. – Vol. 2265. – P. 501–512.
8. Brandes U., Lerner J., and Pich C. GXL to GraphML and vice versa with XSLT // Electronic Notes in Theoretical Computer Science. – 2004 – Vol. 127, N 1. – P. 113–125.
9. Brandes U., Marshall M.S., and North S.C. Graph data format workshop report // Proc. 8th Int. Symp. Graph Drawing GD'2000. – Lect. Notes Comput. Sci. – 2001. – Vol. 1984. – P. 410–418.
10. Brandes U., Pich C. Graphml transformation // Proc. 11th Int. Symp. Graph Drawing GD'2004. – Lect. Notes Comput. Sci. – 2004. – Vol. 3383. – P. 89–99.
11. Bridgeman S. GraphEx: an improved graph translation service // Proc. 11th Int. Symp. Graph Drawing GD'2004. – Lect. Notes Comput. Sci. – 2004. – Vol. 3383. — P. 307–313.
12. Carley K., Reminga J. ORA: Organization risk analyzer. – Carnegie Mellon University, 2004. – (Tech. Rep. / CMU-ISRI-04-106).
13. De Nooy W., Mrvar A., Batagelj V. Exploratory social network analysis with Pajek. – Cambridge University Press, 2005.

14. Di Battista G., Eades P., Tamassia R., Tollis I.G. Graph Drawing: Algorithms for the Visualization of Graphs. – Prentice Hall, 1999.
15. GML. The Graph Modeling Language File Format. – http://www.infosun.fmi.uni—passau.de/Graphlet/GML/.
16. O'Madadhain J., Fisher D., Smyth P., White S., Boey Y.B. Analysis and visualization of network data using JUNG // Journal of Statistical Software, 2005. – P. 1–35.
17. Saxon Open Source Project. – http://saxon.sourceforge.net/.
18. Sugiyama K., Tagawa S., Toda M. Methods for visual understanding of hierarchical system structures // IEEE Transactions on Systems, Man and Cybernetics. – 1981. – Vol. 11, N 2. — P. 109–125.
19. Tsvetovat M., Reminga J., Carley K. Dynetml: interchange format for rich social network data // NAACSOS Conference. — Pittsburgh, PA, 2003.
20. W3C. Scalable Vector Graphics. – http://www.w3.org/TR/SVG/.
21. W3C. SOAP. – http://www.w3.org/TR/soap12—part0/.
22. W3C. XSL Transformations. – http://www.w3.org/TR/xslt/.
23. Winter A. Exchanging Graphs with GXL // Proc. 9th Int. Symp. Graph Drawing GD'2001. – Lect. Notes Comput. Sci. – 2002. – Vol. 2265. – P. 485–500. – yWork. http://www.yworks.com.

В.Н. Касьянов, Е.В. Касьянова

СРЕДСТВА ПОДДЕРЖКИ ПРИМЕНЕНИЯ ГРАФОВ В ИНФОРМАТИКЕ И ПРОГРАММИРОВАНИИ[1]

ВВЕДЕНИЕ

Современное состояние программирования нельзя представить себе без графов и графовых алгоритмов. Хорошо известно, что многие задачи повышения качества трансляции с точки зрения улучшения рабочих характеристик транслятора и повышения качества получаемых машинных программ формулируются и решаются как задачи на графах и графовых моделях. К ним относятся, в первую очередь, задачи, связанные с представлением программ в виде схем программ и синтаксических деревьев. Кроме того, необходимо указать на такие области применения граф-моделей, как эффективное использование ресурсов вычислительной системы (оптимизация использования регистров, уменьшение обменов между оперативной и внешней памятью и т.д.), организация больших массивов информации (деревья и графы данных для повышения эффективности информационного поиска вообще), увеличение степени параллелизма программы, повышение эффективности работы многопроцессорных и многомашинных систем (распределение загрузки процессоров, обмен сообщениями между процессами, синхронизация, конфигурация сетей связи между процессорами и т.д.). Решение этих и подобных задач привело к появлению множества граф-моделей, связанных как с программами и структурами данных, так и с вычислительными системами, в том числе параллельными (см., например, [6]).

Широкая применимость графов связана с тем, что они являются естественным средством объяснения сложных ситуаций на интуитивном уровне. Эти преимущества представления сложных структур и процессов графами становятся более ощутимыми при наличии хороших средств их визуализации. Поэтому неслучайно в последнее время в мире растет интерес к методам и системам рисования и визуальной обработки графов и графовых мо-

[1] Работа выполнена при частичной финансовой поддержке Российского фонда фундаментальных исследований (гранты РФФИ № 09-07-0012, № 12-07-0091)

делей [7]. Многие программные системы, особенно те, которые используют информационные модели, включают элементы визуальной обработки графовых объектов. Среди них – системы и окружения программирования, инструменты CASE-технологии, системы автоматизации проектирования и многое другое.

Статья посвящена средствам поддержки применения графов в информатике и программировании, которые разрабатываются в Институте систем информатики им. А.П. Ершова СО РАН при частичной финансовой поддержке Российского фонда фундаментальных исследований (гранты РФФИ № 09-07-0012, № 12-07-0091). В ней рассматриваются работы по созданию электронного словаря по графам в информатике WikiGRAPP и электронной энциклопедии графовых алгоритмов WEGA, а также систем визуализации графов и графовых алгоритмов HIGRES и Visual Graph.

ВИКИ-СЛОВАРЬ WIKIGRAPP

Проблема терминологии, без сомнения, является одной из основных проблем в применении теоретико-графовых методов в программировании и информатике. Терминология в прикладной теории графов далеко не устоялась, при написании статей требуется терминологическая привязка к одной из существующих на русском языке монографий, что становится все более трудным делом из-за сокращения числа издающихся книг, в том числе переводных, и резкого сокращения их тиража. Созданный нами расширяемый электронный словарь по графам в информатике WikiGRAPP (см. ниже) призван если не решить, то значительно облегчить эту проблему.

В 1999 году в издательстве «Наука» вышел в свет толковый словарь по теории графов и её применении [1], который охватывал основные связанные с графами термины из монографий, вышедших на русском языке. Это был первый словарь по графам в информатике, и он вызвал большой интерес среди читателей. Электронная версия словаря получила название GRAPP (GRaphs and their APPlications). Новое исправленное и дополненное издание словаря [2], работа над которым была завершена авторами в 2009 году, представляет собой расширение словаря 1999 года и включает в себя более 1000 новых терминов из статей, рефераты которых публиковались в РЖ «Математика» в разделе «Теория графов», а также из томов ежегодных конференций «Graph-Theoretic Concepts in Computer Science» и книг серии «Graph Theory Notes of New York».

Авторы словарей отдавали себе отчет в том, что теоретико-графовый лексикон в информатике постоянно развивается, и поэтому одновременно с завершением работ по подготовке второго словаря к изданию приступили к созданию на базе изданных словарей новой, расширяемой версии электронного словаря. Новый словарь [10], получивший название WikiGRAPP (см. рис. 1), интерактивен и поддерживает коллективную сетевую работу по его пополнению и развитию. Для его создания мы использовали систему MediaWiki [9] – написанное на препроцессоре гипертекста (PHP) свободно распространяемое программное обеспечение, предназначенное для создания гипертекстовых wiki-систем – таких веб-сайтов, структуру и содержимое которых пользователи могут сообща изменять с помощью инструментов, предоставляемых самими сайтами. И хотя MediaWiki изначально создавалась в качестве движка всем известной Википедии, она имеет гораздо более широкое применение в многочисленных и весьма разнообразных сайтах, включая сайт о самой системе MediaWiki [9].

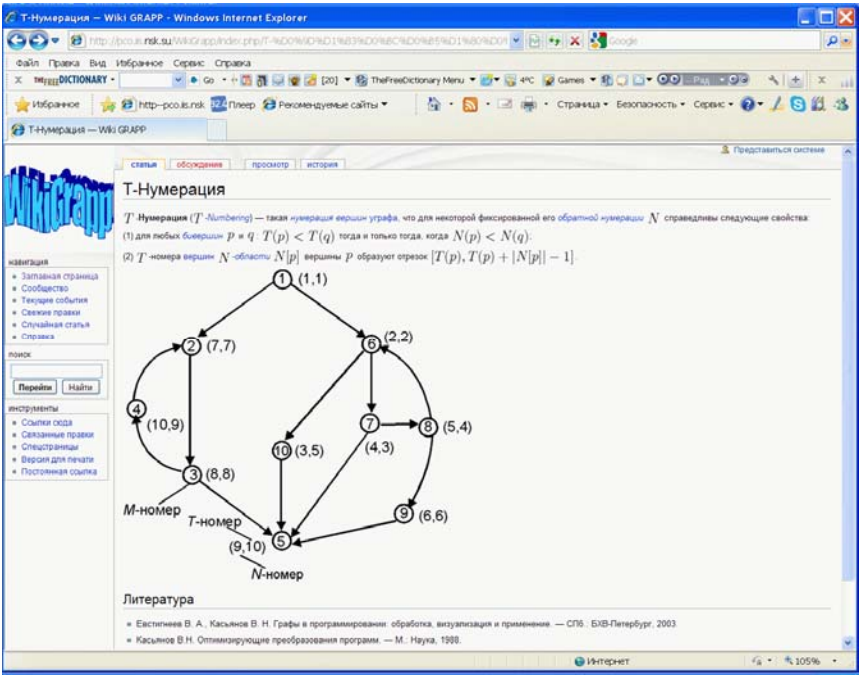


Рис. 1. Электронный словарь по теории графов WikiGRAPP

К настоящему времени практически завершена работа по наполнению словаря WikiGRAPP до уровня, покрывающего печатные издания, и начата работа по исправлению обнаруженных опечаток и пополнению словаря.

ВИКИ-ЭНЦИКЛОПЕДИЯ WEGA

Теория графов из академической дисциплины все более превращается в средство, владение которым становится решающим для успешного применения компьютеров во многих прикладных областях. Несмотря на наличие обширной специальной литературы по решению задач на графах, широкое применение в практике программирования полученных математических результатов затруднено в силу отсутствия систематического их описания, ориентированного на программистов. Поэтому значительный класс практических задач, по существу сводящихся к простому выбору подходящего способа решения и к построению конкретных формулировок абстрактных алгоритмов, для многих программистов все еще остается полем для интеллектуальной деятельности по «переоткрытию» методов.

Выполнен цикл работ по изучению и систематизации алгоритмов обработки, визуализации и применения графовых моделей в программировании. Впервые издана книга [6], которая содержит систематическое и полное изложение фундаментальных основ современных компьютерных технологий, связанных с применением теории графов. Даны основные модели, методы и алгоритмы прикладной теории графов. Подробно описаны такие основные области приложения теории графов в программировании, как хранение и поиск информации, трансляция и оптимизация программ, анализ, преобразование и распараллеливание программ, параллельная и распределенная обработка информации.

Ведётся работа по созданию на базе этой книги вики-системы WEGA, являющейся расширяемой интерактивной электронной энциклопедией теоретико-графовых алгоритмов решения задач информатики и программирования (см. рис. 2).

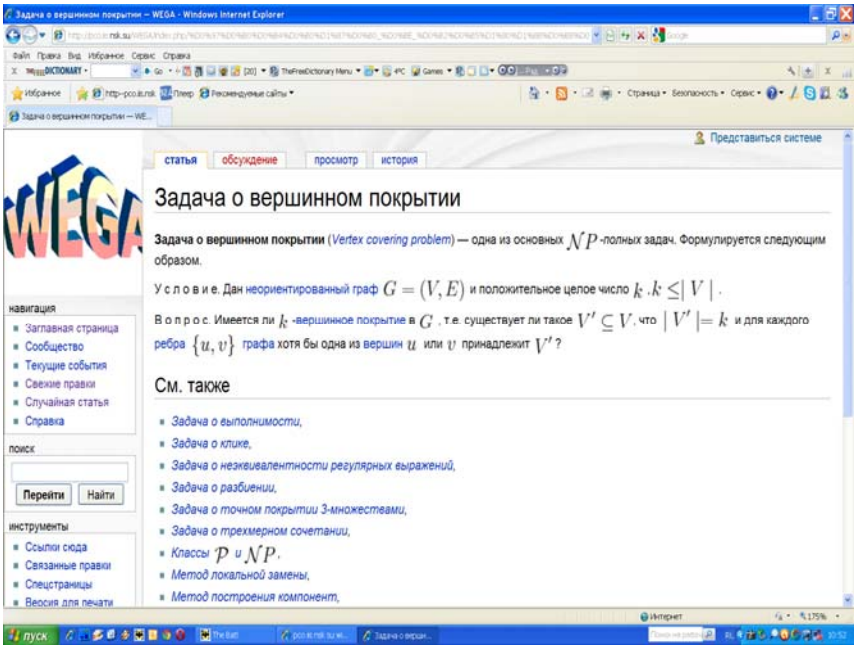


Рис. 2. Электронная энциклопедия WEGA

В отличие от Д. Кнута, при создании данной энциклопедии мы, как и в упомянутой книге, ориентируемся на абстрактную модель современных ЭВМ (равнодоступная адресная машина — РАМ) и высокоуровневое описание алгоритмов в терминах специального языка высокого уровня — ВУ-язык. Этот язык является псевдоязыком (лексиконом) программирования и содержит в качестве базовых традиционные конструкции математики и языков программирования. Наряду с обычными для современных языков типами простых и составных данных он допускает такие более сложные структуры данных, как, например, деревья, графы и т.д. Для каждой базовой конструкции ВУ-языка фиксируется класс её допустимых реализаций на РАМ. Предполагается, что ВУ-язык позволяет наряду с базовыми использовать любые необходимые конструкции, если очевидны или заранее зафиксированы оценки их сложности, а также те реализации этих конструкций на РАМ, которые допускают такие оценки. Такой подход позволяет формулировать алгоритмы в естественной форме, допускающей прямой

анализ их корректности и сложности, а также простой перенос алгоритмов на реальные языки программирования и ЭВМ с сохранением полученных оценок сложности. Подобный стиль описания алгоритмов является также базой для доказательного стиля программирования: он позволяет понять алгоритм на содержательном уровне, оценить пригодность его для решения конкретной задачи и осуществить модификацию алгоритма, не снижая степень математической достоверности окончательного варианта программы.

Еще одной важной особенностью создаваемой энциклопедии является возможность анимационного исполнения представленных в ней графовых алгоритмов. На наш взгляд, анимация является удобным средством демонстрации работы любых алгоритмов, в том числе алгоритмов на графах. Она помогает человеку понять на конкретных примерах смысл и последовательность работы алгоритма, причём гораздо нагляднее, чем любые текстовые описания, пояснения и отдельные рисунки. Поэтому предполагается, что создаваемая энциклопедия будет интерактивной и будет поддерживать полноценную динамическую визуализацию (анимацию) работы представленных в ней графовых алгоритмов.

СИСТЕМЫ ВИЗУАЛИЗАЦИИ ГРАФОВ

Визуализация информации — это процесс преобразования больших массивов и сложных видов абстрактной информации в интуитивно понятную визуальную форму. Универсальным средством такого представления структурированной информации являются графы. Графы применяются для представления любой информации, которую можно промоделировать в виде объектов и связей между объектами. Поэтому визуализация графовых моделей является ключевым компонентом во многих приложениях в науке и технике, а методы визуализации графов представляют собой теоретическую основу методов визуализации абстрактной информации.

Выполнен цикл исследований в области визуализации графов и графовых моделей, в котором наряду с более традиционными вопросами качества и эффективности при автоматическом размещении графов на плоскости большое внимание было уделено вопросам визуализации больших графов, интерактивности и навигации, характерным для большинства современных приложений, использующих визуализацию структурированной информации [7].

Предложена и исследована модель иерархических графов, ориентированная на моделирование сложноорганизованных систем и охватывающая классы составных и кластерных графов, традиционно используемых для представления информационных моделей, обладающих иерархической структурой [3].

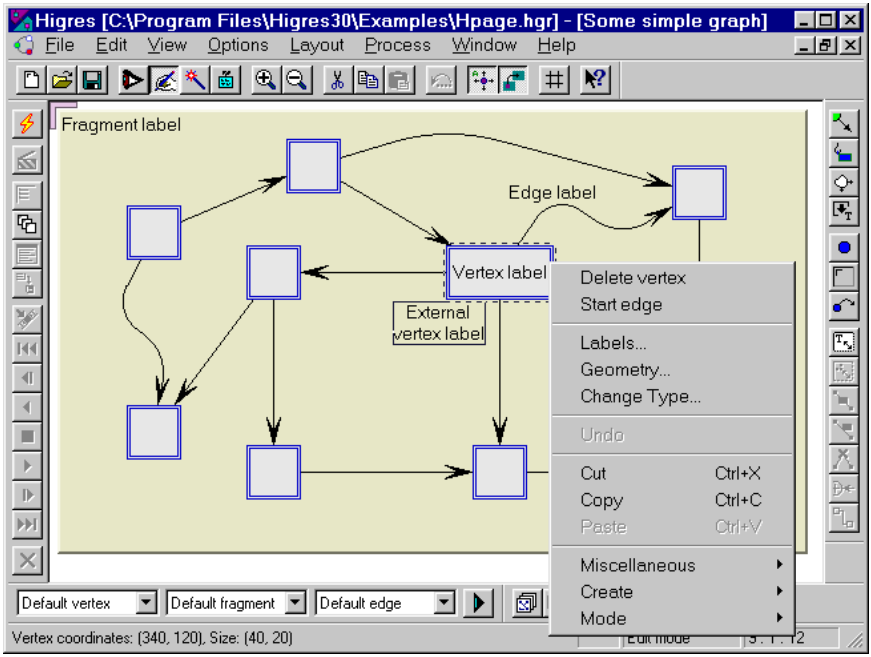


Рис. 3. Графовый редактор HIGRES

Разработаны эффективные методы и алгоритмы построения наглядных изображений иерархических графовых моделей на плоскости и их редактирования. Создан графовый редактор HIGRES (см. рис. 3) [8], поддерживающий многооконную работу с иерархическими графами. Важным его отличием от других графовых редакторов является способность HIGRES сохранять во внутреннем представлении и визуализировать не только сам граф, но и его семантику, представленную в виде системы типов атрибутированных вершин, фрагментов и дуг графа и библиотеки алгоритмов обработки — так называемых внешних модулей. При этом пользователь может

корректировать и доопределять семантику графовой модели с помощью введения новых типов и внешних модулей, а также управлять методами её визуализации. Такой подход обеспечивает, с одной стороны, универсальность системы HIGRES, с другой — возможность её специализации.

Несомненным достоинством системы HIGRES является также то, что она является не только редактором иерархических графовых моделей, но и платформой для исполнения и анимации алгоритмов работы с иерархическими графами.

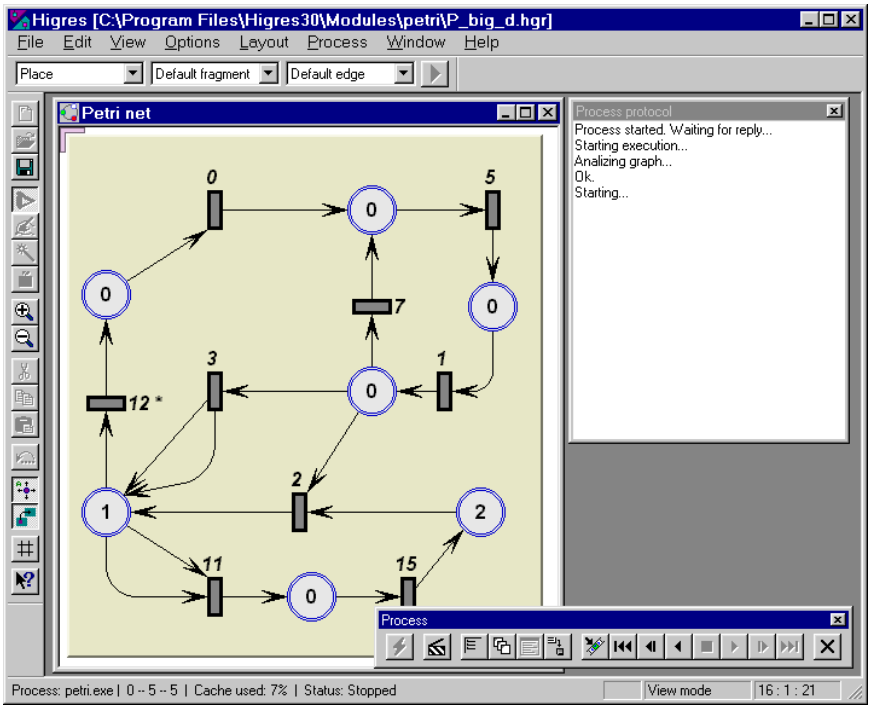


Рис. 4. Анимация функционирования сети Петри в системе HIGRES

Пользователь с помощью системы может выбрать нужный ему алгоритм из расширяемой библиотеки внешних модулей (в частности алгоритм автоматического размещения графа на плоскости) и запустить его. Система передает текущий граф обрабатывающему модулю и открывает специаль-

ное окно, предоставляющее пользователю интерфейс для управления работой модуля. Пользователь может регулировать параметры обработки, прерывать ее на любом шаге, просматривать промежуточные результаты в любую сторону в форме анимации либо в покадровом режиме (см. рис. 4).

Анимация алгоритмов, поддерживаемая системой HIGRES, может быть использована для их тестирования и отладки, для образовательных целей, а также для изучения итеративных процессов, возникающих, например, в некоторых методах рисования графов.

Создана также экспериментальная версия универсальной расширяемой системы Visual Graph (см. рис. 5) для визуализации атрибутированных иерархических графовых моделей большого размера, представленных на языке GraphML [4, 5, 11]. Система Visual Graph поддерживает интерактивное управление визуализацией графовых моделей и удобную навигацию по визуализируемым графовым моделям.

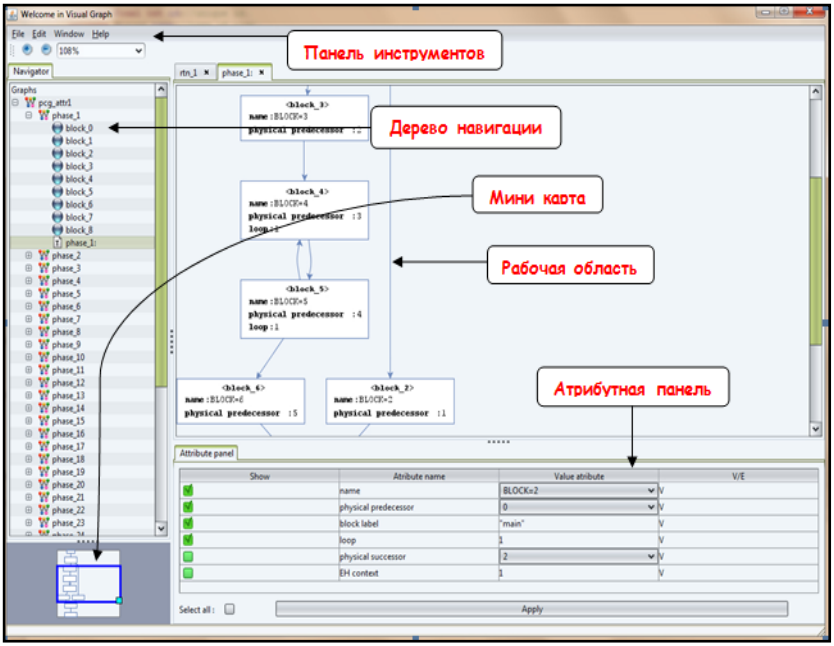


Рис. 5. Система Visual Graph

Благодарности

Авторы благодарны всем, кто принимает участие в выполнении проектов, рассмотренных в данном докладе, в первую очередь проф. В.А. Евстигнееву, И.А. Лисицыну, С.Н. Касьяновой, Э.В. Харитонову и И.Л. Мирзуитовой.

Работа выполнена при частичной финансовой поддержке Российского фонда фундаментальных исследований (гранты РФФИ № 09-07-0012, № 12-07-0091).

СПИСОК ЛИТЕРАТУРЫ

1. Евстигнеев В. А., Касьянов В. Н. Толковый словарь по теории графов в информатике и программировании. – Новосибирск: Наука, 1999. – 288 с.
2. Евстигнеев В. А., Касьянов В. Н. Словарь по графам в информатике. – Новосибирск: Сибирское Научное Издательство, 2009. – 300 с.
3. Касьянов В. Н. Иерархические графы и графовые модели: вопросы визуальной обработки // Проблемы систем информатики и программирования. – Новосибирск: ИСИ СО РАН, 1999. – С. 7–32.
4. Касьянов В. Н. Язык представления графов GraphML: базовые средства // Информатика в науке и образовании. – Новосибирск, 2012. – С. 7–22.
5. Касьянов В. Н. Язык представления графов GraphML: дополнительные возможности // Информатика в науке и образовании. – Новосибирск, 2012. – С. 23–46
6. Касьянов В. Н., Евстигнеев В. А. Графы в программировании: обработка, визуализация и применение. – СПб.: БХВ-Петербург, 2003. – 1104 с.
7. Касьянов В. Н., Касьянова Е. В. Визуализация графов и графовых моделей. — Новосибирск: Сибирское Научное Издательство, 2010. – 123 с.
8. Система HIGRES. – http://pco.iis.nsk.su/higres/
9. Система MediaWiki. – http://www.mediawiki.org/wiki/MediaWiki/ru/
10. Система WikiGRAPP. – http://pco.iis.nsk.su/WikiGrapp/
11. Brandes U., Eiglsperger M., Lerner J. GraphML Primer. – http://graphml.graphdrawing.org/primer/graphml-primer.html#EXT

Р. И. Идрисов

ОБЛАЧНЫЙ СЕРВИС ДЛЯ НАУЧНЫХ ВЫЧИСЛЕНИЙ И ОБРАЗОВАНИЯ [1]

ВВЕДЕНИЕ

На сегодняшний день всё большую популярность набирают облачные сервисы. Конечно, зачастую под активно употребляемым словом «облачный» скрываются обычные вещи, которые просто были названы по-новому. Согласно последней редакции российской Википедии[2], на момент написания статьи облачный сервис – это просто некоторый доступный ресурс в сети, который может быть использован без знания его внутренней структуры.

Статистика поисковых запросов в некотором смысле отражает интересы общества. Если проследить статистику Google[3] по двум терминам – «parallel» и «cloud», можно увидеть, что популярность «cloud» с 2004 года выросла вдвое и продолжает расти, а популярность «parallel», наоборот, неуклонно падает и с 2004 года сократилась вдвое. Рост популярности этого термина обусловлен тем, что облачность представляет пользовательский интерфейс для использования какого-то ресурса. Сам термин «облачный» по одной из версий образовался из того, что Интернет изображался в виде облака: скрытая от пользователя структура, находящаяся в сети и является этим самым облаком. И хотя концепция появилась уже давно, но, как и с функциональными языками программирования, которые были описаны задолго до их популярности, массовое применение облачности стало возможным с развитием соответствующих технологий.

Продолжают набирать популярность облачные хостинги Amazon, cloud9, которые предоставляют вычислительные ресурсы. В частности, они позволяют выполнять код на V8 (JavaScript от Mozilla). Этот язык во многом совместим с браузерным JavaScript, что позволяет создавать переносимый код. Одна и та же программа может быть исполнена как на персональной рабочей станции, так и на потенциально мощном облачном вычислите-

[1] Работа поддержана грантом РФФИ № 12-07-31060 мол_а.

[2] http://ru.wikipedia.org

[3] http://www.google.com/insights/search/

ле. При этом переносимость в данном случае совсем не такая, как в случае Java. Программное обеспечение не требует дополнительной установки библиотек поддержки времени исполнения, а выполняется внутри браузера. Кроме того, это исполнение предполагается безопасным для пользователя. Растущее количество сервисов в сети Интернет, предоставляющих различные услуги, прямо или косвенно связанные с вычислениями, говорит о популярности такого подхода. Возможность в любой момент зайти на соответствующую страницу в сети и выполнить интересующий код, не имея компилятора, очень привлекательна. Подобные сервисы варьируют от совершенно аскетичных, рассчитанных на исключительно короткие программы[4], до предоставляющих среду разработки с группировкой по проектам, подсветкой синтаксиса[5] и т.д.

Несмотря на большое количество таких ресурсов, определённая ниша всё же остаётся незаполненной. В частности, нет таких ресурсов, которые предоставляли бы возможность отладить и запустить научную задачу с массивным параллелизмом.

Следует отметить, что такой подход не является чем-то абсолютно новым, из-за большого количества разных параллельных систем разработчики стараются использовать переносимые варианты, которые могут быть исполнены в различных условиях, в тех случаях, когда речь не идёт о специальном программном обеспечении для конкретного вычислителя.

В этой статье мы опишем систему поддержки параллельного программирования для облачных вычислений, разрабатываемую в ИСИ СО РАН. Дальнейшее изложение организованно следующим образом: в первой главе обсуждается входной язык для системы; во второй главе мы рассмотрим возможности системы с точки зрения пользователя и то, как образовательные задачи могут сочетаться с научными. В заключении будет приведён текущий статус разрабатываемой системы.

ИСПОЛЬЗУЕМЫЙ ЯЗЫК

Разрабатываемая система ставит перед собой две цели: научную и образовательную. С нашей точки зрения, для научной цели более важна масштабируемость, а для образовательной – доступность. Кроме того, было бы неправильно ориентироваться только на один язык программирования, поскольку не существует единого мнения о наилучшем учебном языке. Для

[4] http://codepad.org
[5] http://c9.io – Cloud 9

масштабируемости требуется универсальность описания параллелизма: это значит, что программа не должна быть адаптирована для структуры конкретной вычислительной системы. Согласно работам А. П. Ершова это достигается, если язык программирования приближается к языку описания задач, а не к языку описания алгоритмов. В Институте систем информатики СО РАН мы продолжаем разработку потокового языка программирования Sisal [1] [2], эта работа ставит перед собой именно такие цели. Рассмотрим некоторые возможности Sisal в сравнении с другими языками для параллельного программирования.

Однократное присваивание

Как и многие функциональные языки программирования, Sisal использует однократное присваивание. Этот подход в программировании требует, чтобы каждое значение описывалось в программе только один раз и не изменялось (не присваивалось повторно). Некоторые могут сказать, что любая императивная программа может быть приведена к форме с однократным присваиванием, и что использование подобных ограничений не имеет смысла. Рассмотрим следующий пример на языке Си:

```
int g=0;
void foo(void) { g=1; }
```

В этом случае для приведения программы к форме с однократным присваиванием требуется обозначить новую копию глобальной переменной *g* в момент определения, но этого нельзя сделать внутри функции, а это было бы удобно, поскольку функция может быть не вызвана. Конечно, если записать программу полностью, то такая ситуация будет разрешимой, но для этого потребуется привлечение дополнительных методик анализа и проверок. Идея заключается в том, что программирование в рамках семантики однократного присваивания это нечто вроде исключения оператора goto, которое не только упрощает анализ, но и положительным образом сказывается на описании алгоритма.

Массивы и потоки

Использование этих типов данных не является обычным для функциональных языков программирования, где, как практически всё по возможности выражается в терминах списков и рекурсии. Но массивы существенно упрощают некоторые задачи вычислительного программирования, в частности, с массивами существенно понятней становится описание матрицы. При этом массивы сами по себе не являются императивным или последова-

тельным элементом. Покомпонентное перемножение векторов на языке Sisal можно определить следующим образом:

```
for i in 1, N repeat
 R := A[i] * B[k]
      returns array of R
```

Многословный синтаксис

Это свойство делает программы более читаемыми (проще для человеческого восприятия), и, как результат, долгосрочная поддержка таких программ упрощается. Большинство функциональных языков программирования «страдают» от слишком короткой записи, что, с одной стороны, позволяет описывать алгоритмы более коротко и выразительно, с другой – усложняет понимание этих алгоритмов. Как следствие, гораздо меньше людей в принципе способны на такое программирование. В качестве примера приведём известную программу быстрой сортировки на языке Haskell:

```
qsort :: Ord a => [a] -> [a]
qsort []     = []
qsort (x:xs) =
qsort [y | y <- xs, y < x] ++ [x] ++ qsort [y | y <- xs, y >= x]
```

и для сравнения такая же процедура на языке Sisal:

```
function qsort (Data : array[real] returns array[real] )
if array_size( Data ) > 2 then
let
L, Middle, R := for E in Data
 returns array of E when E < Data[ 1 ]
array of E when E = Data[ 1 ]
array of E when E > Data[ 1 ]
 end for
in
qsort( L ) || Middle || qsort( R )
end let
else
Data
end if
end function
```

Обработка ошибок

На сегодняшний день достаточно популярен механизм try-catch, но он не является естественным для параллельных программ, поскольку требует досрочного завершения определённого участка кода при возникновении ошибки, что приводит к откату части параллельных вычислений, выпол-

няемых в данный момент, либо к потере детерминизма программы. Рассмотрим пример на языке Java:

```
try {
 for (int i=0;i<N;i++) {
     a[i]=a[i]/((i+1)%K);
 }
} catch (Exception e) {
 // display partial results stored in "a"
}
```

В этом случае итерации цикла являются независимыми, что даёт возможность их одновременного исполнения при наличии системы автоматического распараллеливания. Но семантика языка остаётся последовательной, и в момент возникновения ошибки требуется остановить все потоки и создать состояние, соответствующее моменту возникновения первой из ошибок вычислений. При этом первая обработанная ошибка вычислений может не быть первой по семантике языка, поскольку некоторый поток с номером i может получить набор итераций, в которых самая первая ошибочная, а поток i-1 набор, где ошибочная итерация находится ближе к концу. Для сохранения детерминизма и изначальной семантики языка требуется большое количество дополнительной работы системы контроля времени исполнения.

Язык Sisal реализует модель всюду завершаемых частичных вычислений, что означает наличие специального «ошибочного» значения, просачиваемого по графу потока данных. Например, при вычислении матрицы часть элементов в результате может иметь ошибочное значение, что означает ошибку. Таким образом, семантика языка является изначально удобной для параллельных вычислений, и сложного контроля со стороны системы времени исполнения (Run Time System) не требуется.

Поддержка других языков программирования

Включение дополнительного языка не требует больших усилий при наличии реализованного интерпретатора на JavaScript. Так как система ориентирована на функциональные языки программирования, пошаговая отладка не предполагается. Требуется только возможность вычислять выражения без полного описания программы, что упрощает требования к компилятору.

СТРУКТУРА СИСТЕМЫ

С точки зрения надежности распределённые системы являются достаточно привлекательными, поскольку выход из строя или недоступность какой-то из частей далеко не всегда означает полную недоступность системы. Кроме того, не требуется специальной настройки сетевых дисков или постоянного переноса файлов при перемещении пользователя. Хранение предоставляется сервису, причём вместе с версионностью и возможностью отменить изменения, сделанные когда-то. В образовательных системах это открывает дополнительную возможность для преподавателя проследить процесс решения задачи в зависимости от времени. Студенту в этом случае не потребуется создавать версии – система будет автоматически сохранять данные через заданные интервалы времени, что также повышает вероятность быстрого восстановления кода после неожиданного отключения клиента облачного сервиса. Обратим внимание на несколько ключевых особенностей системы:

Иерархия

Распределение ресурсов часто упоминается среди особенностей облачных систем. Как для учебных, так и для научных задач часто требуется ограничивать возможности по использованию ресурсов с целью обеспечения их доступности для остальных. Здесь естественным образом можно использовать иерархию пользователей, которая бы отражала распределение вычислительных ресурсов и давала возможность контроля нижестоящих членов.

Версионность

Современные системы хранения практически немыслимы без версионности, но здесь для образовательных нужд можно дополнительно использовать темпоральную версионность, в которой акцент сделан не на номере версии, а именно на времени, в которое эта версия была реализована. При этом сохранение происходит неявным образом и у преподавателя появляется дополнительная возможность контроля.

Отладка

Кроме отладки при помощи значений, предполагается возможность дополнительной отладки на графах в будущем, но сам по себе интерпретатор

предоставляет возможность отладки только путём вычисления значений. Дополнительные возможности отладки предполагается реализовывать при помощи расширений системы.

ТЕКУЩИЙ СТАТУС И ПЛАНЫ

Структура разрабатываемой системы отражена на рисунке.

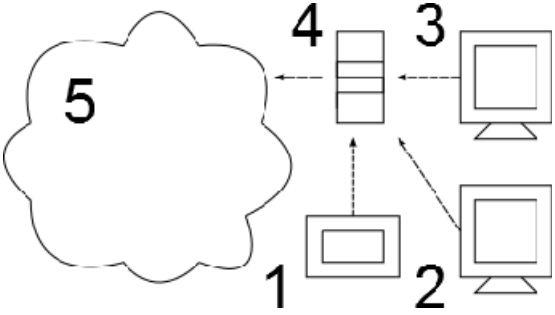


Рис.1. Структура разрабатываемой системы

Цифрами 1, 2 и 3 обозначены устройства, подключающиеся к облачному сервису. На этих устройствах интерпретатор JavaScript, находящийся внутри браузера, используется для реализации среды разработки и отладки программ на задачах меньшей размерности. При отключении от системы (потере интернет-соединения) работоспособность интерпретатора сохраняется.

Цифрой 4 обозначен сервер, предоставляющий доступ к облаку. Этот сервер доступен по фиксированному адресу и может не выполнять каких-либо вычислительных задач. 5 – облачная структура, выполняющая основные вычисления и хранящая пользовательские данные. Под данными предполагаются как программы, так и входная/выходная информация.

На данный момент выбран готовый редактор с подсветкой синтаксиса и открытым кодом. Реализован front-end интерпретатора Sisal на JavaScript, синтаксический анализатор построен при помощи PEG.js. Находится в разработке руководство пользователя языка Sisal. Использование системы

также предполагается в рамках курса по современным методам и языкам программирования, преподаваемого мною в Новосибирском Государственном Университете. Кроме языка Sisal для образовательных целей предполагается использовать Haskell и OCaml.

СПИСОК ЛИТЕРАТУРЫ

1. McGraw J. R. et al. Sisal: Streams and iterations in a single assignment language, Language Reference Manual Version 1.2 // Lawrence Livermore Nat. Lab. Manual M-146 (Rev.1). — Livermore, CA, 1985
2. Касьянов В. Н. Функциональный язык SISAL 3.0 / В.Н. Касьянов, Ю.В. Бирюкова, В.А. Евстигнеев // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 54-67.

Р. И. Идрисов

ПАРАЛЛЕЛИЗМ В JAVASCRIPT[1]

ВВЕДЕНИЕ

О преимуществах параллельного исполнения на сегодняшний день уже можно подробно не рассказывать, поскольку эта тема стала достаточно очевидной. Тенденции таковы, что даже мобильные телефоны оснащаются многоядерными процессорами. Согласно ресурсу Steam[2], который собирает статистику по конфигурации компьютеров собственных клиентов, пропорция однопроцессорных, двухпроцессорных и четырёхпроцессорных персональных ПК изменилась с 14.1%, 56.9% и 27.24% (август 2010) до 5.1%, 48.21% и 43.54% (январь 2012). Есть и другая тенденция, заключающаяся в том, что большое количество приложений переходят в браузеры (программы просмотра html-документов); это обеспечивает некоторую универсальность вместе с неизбежными ограничениями.

В связи с этим возникает вопрос о возможности эффективного использования современных персональных компьютеров при помощи браузера и JavaScript в частности. Существует несколько способов измерить производительность JavaScript, можно воспользоваться тестом от Mozilla[3], который называется v8. Этот тест включает в себя много частей, которые покрывают потенциальные способы использования JavaScript для большого количества вычислений, но если попытаться проверить степень параллелизма этих вычислений, выяснится, что все они исполняются только на одном процессоре/ядре. Тем не менее JavaScript содержит модель конкурентного исполнения. Попробуем выполнить следующее многопоточное приложение:

```
function startThread(num) {
      var l=0, dateObj=new Date(),
          startTime=dateObj.getTime();
      if (num < 1) return;
      console.log("Thread " + num + " started");
      setTimeout('startThread('+(num-1)+');', 100);
      while (dateObj.getTime()-startTime<3000) {
              dateObj=new Date();
              l+=Math.random();
      }
      console.log("Thread " + num + " ended (l=" + l + ")");
}
startThread(10);
```

.

[1] Работа поддержана грантом РФФИ № 12-07-31060 мол_а.

[2] http://store.steampowered.com/hwsurvey

[3] http://dromaeo.com/?dromaeo|sunspider|v8

Это приложение создаёт 10 вычислительных потоков с интервалом в 0.1 секунды, каждый из которых работает в течение трёх секунд. Результаты удивляют, поскольку потоки выполняются полностью последовательно для всех популярных на данный момент браузеров[4]. В случае с IE потребуется заменить console.log на document.write или alert, но исполнение остаётся последовательным.

Таким образом, несмотря на теоретическую возможность, текущие реализации языка не поддерживают исполнение нескольких вычислительных процессов одновременно. В случаях, когда это требуется, можно разбивать задачи на мелкие части и управлять запуском мелких частей [1]. В этом случае вычислительные ресурсы будут использованы ещё менее эффективно из-за накладных расходов и простоя, но поведение других элементов страницы и программы-браузера будет более адекватным (в случае исполнения интенсивного кода другие процессы на странице блокируются, в некоторых случаях и на других страницах тоже).

Тем не менее можно ожидать, что рано или поздно такая возможность будет реализована. Рассмотрим способы конкурентного выполнения вычислительных потоков в рамках JavaScript.

CONCURRENT.THREAD

Эта библиотека, распространяемая по свободной лицензии, позволяет эмулировать многонитевое исполнение программы, разбивая программы на небольшие фрагменты. Конечно, это не может не сказаться на производительности.

Во-первых, расскажем о самой организации вычислений: для запуска параллельного процесса используется функция

Concurrent.Thread.create(function_var, function params…),

что само по себе достаточно удобно для тех, кто знаком, например, с нитями .NET. Если переписать определённый в начале данной статьи пример с использованием этой библиотеки, получим следующий код:

```
function startThread(num) {
     var l=0, dateObj=new Date(),
         startTime=dateObj.getTime();
     if (num < 1) return;
     console.log("Thread " + num + " started");
     Concurrent.Thread.create(startThread, num-1);
```

[4] Firefox 10.0.2, Safari 5.1, Google chrome 17, Opera 11, Internet Explorer 9

```
        while (dateObj.getTime()-startTime<3000) {
                dateObj=new Date();
                l+=Math.random();
        }
        console.log("Thread " + num + " ended (l=" + l
        + ")");
    }
    startThread(10);
    .
```

Кроме того, можно воспользоваться другим способом и обозначить фрагмент JavaScript-кода как многонитевой. Предыдущий пример привести к такому виду не получится, поскольку в нём требуется явное указание запуска различных потоков. Оформим синхронный HTTP-запрос как отдельный вычислительный поток [3]:

```
<script type="text/x-script.multithreaded-js">
    var   req   =   Concurrent.Thread.Http.get(url,   ["Accept",
"*"]);
    if (req.status == 200) {
        alert(req.responseText);
    } else {
        alert(req.statusText);
    }
 </script>
```

Выше мы упомянули понижение производительности при использовании этого подхода. Проведём несколько тестов для того, чтобы выяснить, насколько оно существенно. В качестве вычислительной задачи возьмём генерацию случайных чисел. Не будем приводить полный текст программы, поскольку это займёт достаточно места, используемая задача – генерация 10^7 псевдослучайных чисел. Такой цикл занимает примерно 100 мс в случае последовательного выполнения (в процессе которого блокируется работа браузера). В случае параллельного исполнения при помощи рассматриваемой библиотеки характерные времена исполнения отличаются на 2 порядка:

Можно отметить, что время отличается достаточно сильно, что делает невозможным использование этой библиотеки в вычислительных задачах, потому что потери времени масштаба двух порядков совершенно не оправданы. Тем не менее, у этого способа есть положительный момент: независимые вычисления не мешают друг другу. Это значит, что корреляции с количеством потоков в данном тесте не было обнаружено. Тестирование производилось на Google Chrome 17.

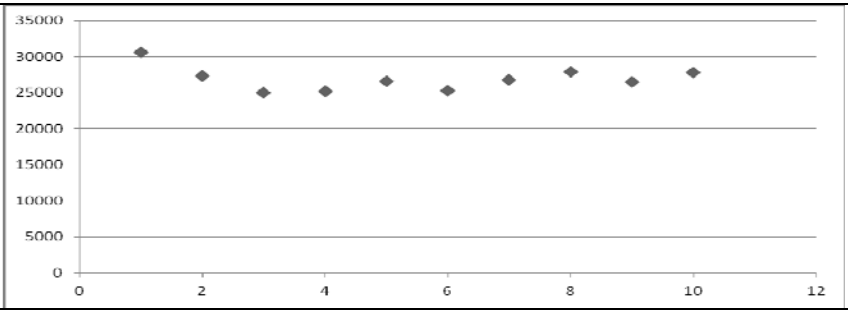


Рис. 1. Зависимость времени в миллисекундах от количества работающих потоков при использовании библиотеки Concurrent.Thread

MULTITHREADING

Другой подход [2], предложенный лабораторией компьютерных сетей École Polytechnique ,включает в себя отдельный компилятор и средства для отладки параллельных приложений на JavaScript. Подход изначально направлен не на получение высокопроизводительного кода, а на другую модель описания параллельных процессов. Эта модель также реализована в языке Synchronous C++ или sC++ [4]. Синхронизация в данном случае осуществляется посредством отправки сообщений. Реализованы операторы select, accept, waituntil, которые компилируются в обычный JavaScript-код при помощи Java-приложения. Программа разбивается на части между синхронизациями, которые не подвергаются изменениям, а для синхронизаций генерируется дополнительный код. Как уже упоминалось, для готовых программ создан отладчик, который позволяет проследить за синхронизацией.

В целом, такое решение больше подходит для учебных целей, а не для решения реальных задач, поскольку является достаточно громоздким и требует перекомпиляции при каждом изменении параллельного участка кода. Сам по себе оператор waituntil, который прерывает исполнение потока до указанного времени, возможно, и является тем средством, которого многим не хватало в JavaScript, но для удобства его использования следовало реализовать компилятор на самом языке JavaScript, как в прошлом случае. Приведём фрагмент программы на synchronous JavaScript:

```
process Channel(name) {
  var fifo = new Array(0)
  this.put = function(data) {
    fifo.push(data)
  }
  this.get = function(data) {
    return fifo.shift()
  }
  this.run = function() {
    for (;;) {
      select {
      case
        when (fifo.length>0)
        accept get
      case
        when (fifo.length<6)
        accept put
      }
      showChannel(fifo.length)
    }
  }
}
.
```

В примере описывается процесс, реализующий FiFo-очередь.

JQUERY DEFERREDS[5]

Достаточно популярная библиотека jQuery позволяет создавать объекты, которые помогают организовывать вычисления по готовности данных. Как правило, в JavaScript готовность данных означает, что закончился процесс их получения с сервера. Концепция не является сложной: создаётся объект, в который передаются функции, выполняемые в случае различных статусов при разрешении (в случае успеха, неуспеха, в любом случае). Как правило, такой объект возвращается функцией, которая выполняет загрузку данных, объект разрешается при помощи замыкания, контекста реального обработчика событий на контекст функции, вернувшей deferred-объект. Функции асинхронных запросов JQuery всегда возвращают deferred-объекты.

[5] http://api.jquery.com/category/deferred-object/

```
$.get("test.php").done(function() {
   alert("$.get succeeded"); });
```

В приведённом примере функция, выполняющая AJAX запрос типа get, возвращает deferred-объект, который позволяет назначить функцию, выполняемую в случае успешности запроса.

Также могут быть реализованы конвейеры из методов JavaScript, выполняемые асинхронно:

```
var defer = $.Deferred(),
    filtered = defer.pipe(function( value ) {
      return value * 2;
    });
defer.resolve( 5 );
filtered.done(function( value ) {
  alert( "Value is ( 2*5 = ) 10: " + value );
});
```

.

ВЫВОДЫ

Из рассмотренных способов организации параллельных процессов на JavaScript только Concurrent.Tread подходит для реализации вычислений. Возможно, разработчиками стандартов JavaScript будет выбран какой-то другой способ, но в связи с развитием браузерных приложений всё сложней обходить вниманием тот факт, что процессоры стали многоядерными.

СПИСОК ЛИТЕРАТУРЫ:

1. Edwards J. Multi-threading in JavaScript. — 2008. — Available at: http://www.sitepoint.com/multi-threading-javascript/ (accessed on 01.10.1012)
2. Petitpierre C. Multithreading for Javascript. Available at: http://ltiwww.epfl.ch/sJavascript/(accessed on 01.10.1012).
3. Maki D., Iwasaki H. JavaScript Multithread Framework for Asynchronous Processing: PhD thesis [online pdf document]. –15 p. – Available at: http://mirror.transact.net.au/sourceforge/j/project/js/jsthread/doc/thesis-en.pdf.
4. Petitpierre C. Synchronous C++, a Language for Interactive Applications // IEEE Computer. – 1998 – N. 9. – P. 65–72.

Г.П. Несговорова

БИОИНФОРМАТИКА: ПУТИ РАЗВИТИЯ И ПЕРСПЕКТИВЫ

ВВЕДЕНИЕ

Со времени своего появления, во второй половине 20-го века, наука информатика начала широко внедряться и сотрудничать (и продолжает это и в наши дни) с другими науками: физико-математическими, техническими, гуманитарными и пр. Перефразируя всем известные слова М.В. Ломоносова, можно сказать, что «широко простирает информатика руки свои в дела человеческие». Сейчас трудно найти такую область науки, которая бы обходилась без методов информатики. Не избежали этого и естественные науки.

В конце 60-х – начале 70-х годов прошлого века ЭВМ стали активно применяться в биологии: к этому времени возросла их память, увеличилась скорость операций, уменьшились размеры. И к тому же накопилось большое количество экспериментальных данных по биологии, требующих осмысления и обработки. Например, уже к 2003 г. объединенными усилиями ученых многих стран был в общих чертах прочитан геном человека.

Таким образом, на стыке XX-го и XXI-го веков появилась бурно развивающаяся область биомедицинской науки биоинформатика, которой к настоящему времени насчитывается уже около 30 лет. Биоинформатика обязана своим появлением накоплению обширных экспериментальных данных. Особенно заметно это накопление стало проходить в 70-х гг. XX-го века. Информации, получаемой в биологических экспериментах, было значительно больше, чем возможности человека к запоминанию фактов и их анализу. Возникла необходимость хранения все быстрее увеличивающегося объема информации. Первые несколько сотен расшифрованных последовательностей белков были опубликованы в виде книги-атласа. Однако уже в начале 70-х г. число расшифрованных последовательностей возросло настолько, что из-за их объема оказались невозможными публикации в виде книг. Стало понятно, что нужны специальные программы для сравнения последовательностей, поскольку мозг человека не справляется с анализом такой информации.

В 90-е годы происходит расцвет геномики. К настоящему времени расшифрованы полные геномные последовательности человека, мыши, цып-

ленка, лягушки, отдельных видов рыб, круглых червей, нескольких сотен вирусов и бактерий и т.д. Прочтение генома бактерии – это теперь задача, посильная для группы из 2–3 исследователей за время, меньшее, чем один год. Геном человека составляет около 3-х миллиардов букв, что эквивалентно 15000 книжных томов. А значение факта его «прочтения» для биологов сравнимо с открытием Д.И. Менделеевым периодического закона для химиков.

Поэтому для обработки такой обширной биологической информации на помощь пришли компьютерные технологии. Первый алгоритм выравнивания генных последовательностей был предложен еще в 1970 г. Компьютеры позволили хранить информацию в виртуальных банках данных и оперировать ею с большой скоростью. Биоинформатика, как и многие другие современные науки, развивается на стыке разных наук: молекулярной биологии, генетики, математики и компьютерных технологий. Основная ее задача – разработка вычислительных алгоритмов для анализа и систематизации данных о структуре и функциях биологических молекул, прежде всего нуклеиновых кислот и белков.

Объем генетической информации, накапливаемый в банках данных, начал увеличиваться с возрастающей скоростью после того, как были разработаны быстрые методы секвенирования (расшифровки) нуклеотидных последовательностей ДНК. Достижения информатики, лингвистики и теории информации сделали возможным анализ генетических текстов. Взаимосвязанное развитие биоинформатики с другими областями науки позволяет рассчитывать на формирование нового уровня понимания биологических процессов, происходящих в клетках и организмах.

Если учесть, что первый персональный компьютер появился в 1981 г., а Интернет (World Wide Web) – в 1991 г., т.е. совсем недавно, то это имело колоссальное значение для развития не только предмета науки, но и ее организации. Один из главных принципов биоинформатики (как, впрочем, и других наук) – принцип единого мирового информационного пространства, объединяющего усилия ученых, работающих по всему миру.

Цель данной статьи – дать определение и понятия для широкого круга лиц, что такое биоинформатика, не вдаваясь в глубинные процессы, доступные для понимания лишь узким специалистам-биоинформатикам. В конце статьи в качестве приложения приводится словарь специфических биологических терминов, не всегда известных читателям других специальностей.

1. ЧТО ТАКОЕ БИОИНФОРМАТИКА

Истоки зарождения биоинформатики восходят к давним временам, к 13 веку. Молодой итальянец Леонардо из Пизы, вошедший в историю математики под именем Фибоначчи и знаменитый одноименными числами, описал решение задачи о размножении кроликов, построив тем самым первую математическую модель биологического процесса. А в 20-е годы XX века другой итальянский математик, Вито Вольтерра, создал модель совместного существования двух биологических популяций типа «хищник–жертва». После Второй мировой войны, в конце 40-х годов прошлого столетия, в биологию пришли математики и физики. Современная история биологии начинается в 1953 году, когда американскими учеными Уотсоном и Криком была открыта двойная спираль ДНК.

К настоящему времени существуют разные определения биоинформатики, но в основном под биоинформатикой понимают любое использование компьютеров для обработки разного рода биологической информации. Сегодня поле термина «биоинформатика» значительно расширилось и включает все реализации математических алгоритмов, связанных с биологическими объектами, и информационно-коммуникационных дисциплин, применяемых в биологических исследованиях. В биоинформатике, помимо самой информатики, используются методы прикладной математики, статистики и других точных наук. Биоинформатика используется также в биохимии, биофизике, экологии, генетике и в ряде других областей естественных наук.

Биоинформатика включает в себя:

1) математические методы компьютерного анализа в сравнительной геномике (геномная биоинформатика);

2) разработку алгоритмов и программ для предсказания пространственной структуры белков (структурная биоинформатика);

3) исследование стратегий соответствующих вычислительных методологий, а также общее управление биологическими системами информационной сложности.

Модное среди современных ученых слово «биоинформатика» – это уже почти отделившаяся ветвь молекулярной биологии. На сегодняшний день существует множество его определений и интерпретаций. Пока это еще не совсем устоявшийся термин. Рассмотрим это подробней.

Одни говорят, что биоинформатика –это раздел современной науки, занимающейся раскодированием человеческого генома.

Другие утверждают, что биоинформатика – это системная биология, позволяющая рассмотреть, изучить и систематизировать глобальную картину биологии.

Третьи считают, что биоинформатика – это аналог молекулярной биологии с той лишь разницей, что молекулярная биология занимается научными исследованиями в пробирке, а биоинформатика – при помощи мощных компьютерных систем.

Четвертые верят, что биоинфоматика – это возможность по структуре макромолекулы очень быстро найти гены-мишени и создать новые лекарственные препараты.

И каждый из них по-своему прав.

Существуют такие биологические утверждения, которые сначала были предсказаны, а потом проверены, и оказалось, что так и есть. Отсюда появляются такие ветви биоинформатики, как предсказательная, аналитическая, эволюционная.

В практическом смысле биоинформатика – это прикладная наука, обслуживающая интересы биологов. К области технической биоинформатики относится первичная обработка данных. Полученные данные надо где-то хранить, обеспечивать к ним удобный доступ и т.д. Более сложное и интересное занятие биоинформатиков – получать на основе данных о геноме конкретные утверждения: белок А обладает такой-то функцией, ген В включается в таких-то условиях и т.п. В этом и состоит практическое применение науки биоинформатики.

Биоинформатика находит свое применение в следующих направлениях биологической науки:

– геномика, транскриптомика и протеомика;
– компьютерное моделирование в биологии развития;
– компьютерный анализ генных сетей;
– моделирование в популяционой генетики.

Биоинформатика легко вписалась также в фармакологию, позволяя снизить срок проектирования препарата с 5-6 лет до нескольких месяцев, а также интегрировалась во многие другие медицинские и биологические науки.

На сегодняшний день существуют следующие разделы биоинформатики:

– биоинформатика в целом;
– клиническая биоинформатика;
– структурная геномика;
– функциональная геномика;

– фармакогеномика;
– клиническая протеомика;
– функциональная протеомика;
– структурная протеомика.

С помощью методов биоинформатики возможно не просто обрабатывать огромный массив различных биологических данных, но и выявлять закономерности, которые не всегда можно заметить при обычном эксперименте, предсказываать функции генов и зашифрованных в них белков, строить модели взаимодействия генов в клетке, конструировать лекарственные препараты.

2. ОСНОВНЫЕ ОБЛАСТИ ИССЛЕДОВАНИЙ В БИОИНФОРМАТИКЕ

Рассмотрим подробнее основные области исследований в биоинформатике.

2.1. Анализ генетических последовательностей

С тех пор как в 1977 г. был секвенирован фаг Phi-X 174, последовательности ДНК всё большего числа организмов были дешифрованы и сохранены в базах данных. Эти данные используются для определения последовательностей белков и регуляторных участков. С ростом количества данных уже давно стало невозможным анализировать последовательности вручную. В наши дни для поиска по геномам тысяч организмов, состоящих из миллиардов пар нуклеотидов, используются компьютерные программы.

Сборка фрагментов ДНК может быть довольно сложной задачей для больших геномов. Сейчас этот метод применяется практически для всех геномов, и алгоритмы сборки геномов являются одной из острейших проблем биоинформатики на сегодняшний день. Обработка гигантского количества данных, получаемых при секвенировании, является одной из важнейших задач биоинформатики. Другим примером применения компьютерного анализа генетических последовательностей является автоматический поиск генов и регуляторных последовательностей в геноме.

2.2. Аннотация геномов

В контексте геномики аннотация – это процесс маркировки генов и других объектов в последовательности ДНК. Первая программная система аннотации геномов была создана еще в 1955 году Оуэном Уайтом.

2.3. Вычислительная эволюционная биология

Эволюционная биология исследует происхождение и появление видов, их развитие с течением времени. Информатика помогает эволюционным биологам в нескольких аспектах:

1) изучать эволюцию большого числа организмов, измеряя изменения всех ДНК;

2) сравнивать целые геномы, что позволяет изучать более комплексные эволюционные события;

3) строить компьютерные модели популяций, чтобы предсказывать поведение системы во времени;

4) отслеживать появление публикаций, содержащих информацию о большом количестве видов.

2.4. Оценка биологического разнообразия

Биологическое разнообразие экосистемы может быть определено как полная генетическая совокупность определенной среды, состоящая из всех обитающих видов, будь это капля воды или горсть земли или вся биосфера планеты Земля. Специализированное программное обеспечение применяется для поиска, визуализации и анализа информации и, что особенно важно, для предоставления ее другим людям.

Помимо биоинформатики в современной научной биологической литературе встречается также термин «вычислительная биология». Считается, что вычислительная биология – это не область науки, а подход к использованию компьютеров для изучения биологических процессов. Термины «биоинформатика» и «вычислительная биология» пока еще часто употребляют как синонимы, хотя «вычислительная биология» чаще указывает на разработку алгоритмов и конкретные вычислительные методы. Считается, что не всё использование вычислительной биологии является биоинформатикой, например, математическое моделирование – это не биоинформатика, хотя и связано с биологическими задачами.

Существует еще и математическая биология, которая, как и биоинформатика, также решает биологические задачи, но используемые при этом методы не являются численными, и им не требуется реализация в программном или аппаратном обеспечении.

Отдельно выделяется структурная биоинформатика, к которой относится разработка алгоритмов и программ для предсказания пространственной структуры белков. Биоинформатика, таким образом, входит в разделы био-

логии наряду с анатомией, ботаникой, вирусологией, микробиологией, цитологией, палеонтологией, физиологией и т.д.

3. МЕСТО БИОИНФОРМАТИКИ В ЦЕПИ БИОЛОГИЧЕСКИХ ИССЛЕДОВАНИЙ

Биоинформатика анализирует факты, которые получены в экспериментальной биологии. Полученные данные экспериментатор сравнивает со всем набором, уже имеющимся в банке данных. Если он не нашел в этом банке открытой им последовательности, он заносит ее туда, пополняя тем самым банк. В функции банков включается хранение, систематизация, обновление информации и обеспечение доступа к ней. Эти операции требуют огромных компьютерных мощностей.

Существуют также банки научных публикаций по биологической тематике. Каждая статья, выходящая в номере любого научного журнала по биологии, помещается в банк и аннотируется так, чтобы любой другой ученый мог легко найти ее через Интернет. Крупнейшая on-line библиотека медико-биологических публикаций PubMed содержит более 16 млн. статей, вышедших в течение последних 50 лет.

Интегральные банки данных и энциклопедии выполняют важную функцию, объединяя в себе всю известную информацию о конкретном гене, белке, организме и т.д. Они обобщают информацию из большого числа других банков данных и постоянно обновляют ее.

Рассмотрим, какие исследовательские цели ставит перед собой биоинформатика.

3.1. Анализ геномов, поиск в них генов

Любой вновь прочитанный геном представляет собой огромную последовательность повторяющихся в разных комбинациях букв. Биоинформатика позволяет выделить гены в этом текстовом многообразии. Такая операция выделения гена из генома называется разметкой генома.

3.2. Предсказание функции генов

Выяснить функции всех генов опытным путем достаточно трудоемко. В этом случае биоинформатика помогает предсказывать их, опираясь на сравнение с теми генами, функции которых уже определены.

3.3. Оценка роли отдельных участков последовательности в функционировании белка

В молекуле белка есть участки, отвечающие за решение разного рода биологических задач. Распознавание этих участков с помощью методов биоинформатики расшифровывает весь спектр функций конкретного белка.

3.4. Построение молекулярных моделей белков на основе их последовательностей

Структуры белков определяют опытным путем, например, облучая микроскопический кристалл, состоящий из молекул белка, рентгеновскими лучами. Это достаточно долгий и дорогой процесс. Некоторые белки вообще невозможно анализировать так, поскольку они не имеют кристаллической структуры. Биоинформатика с помощью компьютерного моделирования помогает воссоздать пространственную модель белка, если известна структура белка с хотя бы отдаленно похожей последовательностью.

3.5. Исследование механизма функционирования макромоделей, исходя из их моделей

Зная пространственную структуру молекулы, полученную благодаря методам биоинформатики, можно предсказать, как она работает, и как на ее работу можно повлиять.

3.6. Компьютерное конструирование лекарств

Лекарства можно конструировать, моделируя в пространстве взаимодействие различных химических соединений с белком-мишенью. При этом надо перебрать огромное количество соединений и выбрать оптимальное.

3.7. Анализ информации

Соединение биологии, химии, физики, математики и информатики позволяет разносторонне описывать биологическую систему. Использование компьютерных ресурсов помогает многократно ускорить процесс анализа и повысить точность и скорость получения результатов.

3.8. Использование достижений информатики

Новые открытия в биологии с использованием технологий биоинформатики быстро находят свое применение в медицине, фармакологии, косметологии, биотехнологии, сельском хозяйстве, экологии и других областях. Биоинформатика самостоятельно дает результаты, имеющие практическую значимость, а также обеспечивает условия для работы разных областей биологии.

3.9. Основные технологии биоинформатики

Большая часть работ по биоинформатике сконцентрирована вокруг технологии использования баз данных для хранения биологической информации с последующей ее обработкой. Такие базы данных могут быть публичными или приватными. Очень важным является предоставление публичного доступа к таким базам данных через открытые стандарты. Получает развитие использование онтологий и логических методов для обработки биологической информации, хотя этот подход менее распространен, чем использование баз данных.

4. БИОИНФОРМАТИКА НА СОВРЕМЕННОМ ЭТАПЕ РАЗВИТИЯ

4.1. Развитие биоинформатики в мире

На рубеже 20-21 веков биоинформатика превратилась в бурно развивающуюся область мировой биомедицинской науки. Наряду с исследователями, ведущими фундаментальные разработки, потребителями биоинформационных технологий являются медицинские, фармакологические, биотехнологичные и учебные учреждения. Эта область науки определена в качестве приоритетной как в США, так и во всех других развитых странах.

Количество центров биоинформатики постоянно растет во всех странах Европы, Азии, США и Австралии. Наряду с государственными, академическими и образовательными центрами биоинформатики, в последние годы возникло значительное число организаций и проектов, ориентированных на коммерческое использование результатов исследований в области биоинформатики. Это прежде всего организации, деятельность которых ориентирована на структурный, функциональный и сравнительный анализ геномов, включая геном человека. Наряду с применением уже созданных ме-

тодов биоинформатики интенсивно развивается техническая и программная база для решения прикладных задач, особенно в фармакологии. Быстрыми темпами совершенствуется также и индустрия программного обеспечения для решения таких задач.

4.2. Состояние биоинформатики в России

Биоинформатика относится к интеллектуальным, высокотехнологичным разделам науки, где получаемые результаты в значительной степени зависят от развитого творческого мышления ученых, а не определяются в основном затратами на их техническую вооруженность. Таким образом, учитывая достаточно высокий интеллектуальный и образовательный уровень российских ученых и практическую невозможность больших финансовых затрат в современной экономической ситуации, биоинформатика имеет все основания стать одним из приоритетных направлений науки в Российской федерации.

Уже к 1984 г. стало ясно, что в России (тогда бывшем СССР) образовалась достаточная «критическая масса» специалистов в области применения математических методов в биологии. К этому же времени скопилось достаточно большое количество биологической информации, обработка, осмысление и анализ которой стало невозможно проводить без компьютерной поддержки. Все это и явилось предпосылкой для возникновения новой отрасли биологической науки – биоинформатики.

Количество публикаций по биоинформатике в последние годы стремительно нарастает. Работы в области биоинформатики активно печатались в 90-е годы, но сами эксперименты, описываемые в статьях, были проведены ранее. База для реализации комплексного подхода к проблеме и написания статей создавалась в течение многих лет усилиями многочисленных исследователей. Первые работы по теоретическому анализу аминокислотных последовательностей белков были опубликованы в 50-х годах прошлого века.

В 1988 г. был начат проект по расшифровке генома человека, ставивший своей целью определение полной последовательности ДНК, составляющей хромосомы человека. Этот проект общими усилиями ученых разных стран был завершен к 2003 г.

Отметим ряд институтов, где в России ведутся работы по биоинформатике.

В Москве организован Институт математических проблем биологии РАН. Там занимаются решением разных биологических проблем при под-

держке информационных технологий, таких как структурная и сравнительная геномика, основные молекулярно-биологические механизмы, молекулярно-генетические системы управления, протеомика, метаболика, базы биологических данных, математическое обеспечение биологических экспериментов.

При Институте проблем передачи информации РАН (ИППИ РАН) в 2003 г. основан учебно-научный центр «Биоинформатика». Также к научным институтам такой тематики относятся следующие учреждения:
ВНИИ «Генетика» (Москва);
Институт белка РАН (Пущино);
Институт биоорганической химии РАН (Москва);
Институт молекулярной биологии РАН (Москва);
Институт биомедицинской химии РАМН (Москва);
Институт цитологии и генетики СО РАН (Новосибирск) и ряд других.

Web-сайты имеются у следующих институтов:
Институт биомедицинской химии РАМН (Москва) –
http://www.ibmh.msk.su/bioinform
Институт физико-химической биологии МГУ им. Белозерского –
http://www.genebee.msu.su/
Институт цитологии и генетики СО РАН (Новосибирск) –
http://www.mgs.bionet.nsc.ru/ и другие.

Ряд оригинальных компьютерных программ и баз данных, созданных российскими учеными, приведен и на других сайтах. Ведется работа по подготовке отечественных специалистов в области биоинформатики. Во многих университетах и вузах биолого-медицинской тематики созданы кафедры биоинформатики и организованы учебные курсы по биоинформатике, такие как «Биоинформатика и компьютерное конструирование лекарств».

4.3. Создание Центра Коллективного Пользования по биоинформатике в СО РАН

В СО РАН (Новосибирский Академгородок) существует центр для разработки экспериментальных технологий в области биоинформатики. Для биологов НГУ и СО РАН создали единую информационную базу по биоинформатике. Совместными усилиями нескольких институтов СО РАН и Новосибирского государственного университета открыт Центр Коллективного Пользования (ЦКП) «БИОИНФОРМАТИКА» Суперкомпьютер центра один из самых быстрых в мире. Его производительность 16,5 терафлопс

(16,5 триллионов операций в секунду), по мощности он не уступают аналогичным, установленным в центре Vital-IT в Лозанне (Швейцария), система хранения данных рассчитана на 48 терабайт.

Центр «Биоинформатика», по сути, – это огромная вычислительная система, суперкомпьютер, с помощью которого можно моделировать лекарства от рака и СПИДа, расшифровывать геномы растений и живых организмов. Прочитать миллиарды и миллионы букв в геномах человека, пшеницы, вируса гепатита или энцефалита сегодня позволяет ДНК-секвенатор. В будущем ЦКП сможет помочь и специалистам смежных областей науки. Например, ученые планируют сравнить геном древнего гоминида из алтайской пещеры с геномом современного человека

В ЦКП более 400 пользователей, это около 20 академических институтов СО РАН. Есть и учебные институты, и фирмы, например, «Унипро», «Шлюмберже» и др.

Цель данного центра – создать сетевую инфраструктуру, которая объединит в технологический комплекс экспериментальные установки с вычислительными комплексами и хранилищами данных. Задачами самой структуры являются биоинформационная поддержка научно- исследовательских работ, подготовка специалистов в области компьютерной системной биологии и биоинформатики, разработка новых экспериментально-теоретических технологий.

Активное использование вычислительных мощностей ЦКП предполагается в рамках проекта СО РАН «Геномика, протеомика, биоинформатика». Организациями-учредителями проекта стали Институт цитологии и генетики СО РАН, Институт вычислительной математики и математической геофизики СО РАН, Международный Томографический центр СО РАН, Институт химической биологии и фундаментальной медицины СО РАН, Институт математики СО РАН, Институт вычислительных технологий СО РАН и Новосибирский государственный университет.

С 1998 г. в Институте цитологии и генетики СО РАН регулярно проводятся Международные конференции по биоинформатике регуляции и структуры геномов и системной биологии (Bioinformatics of Genome Regulation and Structure System Biology – BGRS/SB). На регулярно проходящем с недавних пор в Новосибирске Международном молодежном инновационном форуме Интерра также уделяется внимание вопросам биоинформатики.

Новое направление в науке требует и специалистов нового профиля. Поэтому в Академгородке осуществляется их подготовка: в школе N162 совместно с Институтом цитологии и генетики СО РАН создан профиль-

ный 10-й класс по направлению «биоинформатика» с углубленным изучением биологии, информатики, математики, физики на базе Института цитологии и генетики СО РАН и кафедры биоинформатики Новосибирского государственного университета.

В Сибирском государственном медицинском университете (СибГМУ, Новосибирск) в учебном плане появился новый предмет – биоинформатика. Со второго семестра студенты 5-го курса начинают изучать эту интересную и сложную науку, одно из направлений молекулярной биологии, которое исследует молекулярные процессы, но не в in vitro, а in silico, т.е. не в пробирке, а при помощи компьютера.

Следует также отметить и заслуги сибирских ученых в создании новой науки. К ним относится А.А. Ляпунов, выдающийся русский ученый-математик. В 2011 г. отмечалось 100 лет со дня его рождения. Последние годы своей жизни он жил и работал в Академгородке. Надо сказать, что будучи «отцом отечественной кибернетики», он заинтересовался биологией еще в 30-е годы прошлого века. Позднее, в 50-е годы, он снова обратился к биологии. Сложность биологических систем и процессов их эволюции привлекала его как объект приложения методов исследований, характерных для дескриптивной теории множеств. Таким образом, он стоял у истоков создания новой науки на стыке математики и биологии.

5. ПРИМЕРЫ КОНКРЕТНЫХ ПРОГРАММНЫХ КОМПЛЕКСОВ БИОИНФОРМАТИКИ

В системной биологии важнейшую роль играют методы биоинформатики. Они дают возможность с помощью компьютеров

– накапливать и интегрировать в банки данных экспериментальную информацию;

– осуществлять ее компьютерный анализ;

– проводить математическое моделирование структурно-функциональной организации живых систем;

– предсказывать новые свойства живых систем;

– на этой основе планировать новые этапы экспериментальных исследований.

В этом разделе приводятся конкретные примеры баз и банков данных, базовых пакетов, программных средств для полного анализа макромолекул, а также других программных продуктов по биоинформатике с их ад-

ресами в Интернете, разработанных как отечественными, так и зарубежными учеными.

GenBank – http://www.ncbi.nlm.nih.gov/GenBank
Банк данных по нуклеотидным последовательностям (3400000000 пар оснований в 461000 последовательностей).

SWISS-PROT – http://www.expasy.ch/sprot/sprot-top.html
Аннотированный банк данных по аминокислотным последовательностям белков.

PIR – http://www.nbrf.georgetown.edu/pir/searchdb.html
Аннотированный банк данных по аминокислотным последовательностям белков, организованных в соответствии с гомологией и таксономией.

PBD – http://www.rcsb.org/pdb/
Банк данных по 3D структуре биологических макромолекул.

NDB – http://ndbserver.rutgers.edu
Банк данных по нуклеиновым кислотам. Включает структуры ДНК и РНК вместе с их 3-хмерными изображениями.

ProDom – http://protein.toulouse.inra.fr/prodom.html
Банк данных по доменам белков.

Web-серверы, предоставляющие пользователю генетическую информацию, оснащены комплексом программных средств для поиска информации в банках данных и анализа нуклеотидных и аминокислотных последовательностей. В качестве запросов при поиске последовательностей в банках данных могут использоваться номенклатурные названия генов, организмов, ключевые слова и др.

В качестве примера предложим программу Auto Dok, которая является программой для автоматического докинга. С ее помощью можно посмотреть, как молекулы лекарств или кандидатов на роль лекарств взаимодействуют в известной 3D-структуре. В частности, программа применяется для разработки лекарств, специфически связывающихся с тем или иным белком.

Здесь же приведем примеры основных программ сравнения аминокислотных и нуклеотидных последовательностей.

ACT – (Artemis Comparison Tool) – геномный анализ;

Arlequin – анализ популяционно-генетических данных;

Bio Edit – редактор множественного выравнивания аминокислотных и нуклеотидных последовательностей;

Bio Numerics – коммерческий универсальный пакет программ по биоинформатике;

BLAST – поиск родственных последовательностей в базе данных аминокислотных и нуклеотидных последовательностей;

ClustaIW – множественное выравнивание аминокислотных и нуклеотидных последовательностей;

FASTA – набор алгоритмов определения схожести аминокислотных и нуклеотидных последовательностей;

Mesquite – программа для сравнительной биологии на языке Java;

Muscle – множественное сравнение аминокислотных и нуклеотидных последовательностей. Более быстрая и точная программа в сравнении ClustaIW;

Pop Gene – анализ генетического разнообразия популяций;

Populations – популяционно-генетический анализ.

Примером интегрированного инструмента биолога является также Unipro UGENE. Это свободно распространяемое программное обеспечение для работы молекулярного биолога. Пользовательский интерфейс этого продукта обеспечивает:

– простую и удобную работу с последовательностями;
– визуализацию хроматограмм;
– использование редактора множественного выравнивания последовательностей;
– просмотр трехмерных моделей PDB и MMDB с поддержкой стереорежима;
– просмотр филогенетических деревьев;
– применение конструктора вычислительных схем, автоматизирующего процесс анализа;
– поддержку сохранения изображений в векторные форматы для удобства публикаций.

Ряд оригинальных компьютерных программ, баз и банков данных, созданных российскими учеными, можно также найти по разным поисковикам на других многочисленных сайтах по биоинформатике.

ЗАКЛЮЧЕНИЕ

Сегодня в биологии происходит глобальная смена модели науки. Еще недавно эксперимент подтверждал или опровергал те или иные гипотезы.

Теперь же биология становится наукой, которая оперирует с данными, получаемыми в промышленных масштабах. Использование методов биоинформатики позволяет ученым делать содержательные выводы, которые опираются на сопоставление разнородных данных или на объединение согласованных наблюдений.

Хотя биоинформатике как науке всего лишь около 30 лет, в ней уже существуют свои традиционные направления: компьютерный анализ ДНК, РНК и белковых последовательностей, реконструкция пространственных структур биополимеров, теоретический и компьютерный анализ структурно-функциональной организации геномов и белков. Сюда же относится и развитие баз данных по молекулярно-генетической тематике, структурированию экспериментальных данных. Биоинформатика по своей сути – это интегративная наука, являющаяся инструментом в руках тех, кто занимается биологией, молекулярной биологией, молекулярной генетикой, медицинской генетикой, фармакологией, а также биохимией, биофизикой и т.д.

Подводя итоги, можно сказать что биоинформатика это широко применяющийся и быстро развивающийся способ заниматься биологией, не наблюдая живые существа, как зоологи, не делая опытов в пробирке, как в экспериментальной биологии, а анализируя результаты массовых данных или целых проектов, используя компьютерные технологии, можно делать совершенно конкретные, проверяемые биологические утверждения.

Биоинформатика, геномика и протеомика – это науки о жизни, определяющие начало 21 века, так же как молекулярная биология, иммунология и биотехнология определяли конец 20 века. Не так давно биохимики тратили всю жизнь, чтобы определить структуру одного белка и выявить его функцию. С помощью методов биоинформатики можно предсказать функции тысячи белков. Таким образом, в триаде геномика–протеомика–биоинформатика по сути заложены все новые подходы к созданию принципиально новой медицины будущего: новые методы диагностики, новые лекарства, так как биоинформатика – это путь от гена к лекарству через структуру макромолекулы.

Современное понимание биоинформатики приводит нас к мысли о том, что все те задачи, которые до недавнего времени решались биохимией и молекулярной биологией в реальных экспериментах, в будущем могут быть решены с той или иной степенью точности в виртуальных компьютерных экспериментах.

ПРИЛОЖЕНИЕ

Словарь биологических терминов

Биоинформатика, геномика, протеомика и прочие «-омы» и «-омики» – все эти слова прочно вошли в современную биологическую науку, она буквально пестрит красивыми названиями. При этом «-ом» обозначает множество однородных элементов (ген–геном), а «-омик» – методы изучения (геном–геномика). Посмотрим, что же означают основные термины современной биологии.

Аминокислота (аминокарбоновая кислота) – органическое соединение, в молекуле которого одновременно содержатся карбоксильные и аминные группы. К настоящему времени в различных объектах живой природы обнаружено до 200 аминокислот.

Аннотация – процесс маркировки генов и других и других объектов в последовательности ДНК.

Генетика изучает механизмы изменчивости и наследственности в живых организмах, определяет развитие современной науки о жизни. Изучает структуру и функцию генов, инвентаризует гены, создавая геномные карты живых организмов.

Геном (нем. Genom, англ. genome) – совокупность генов, локализованных в одиночном наборе хромосом данного организма. Термин «геном» предложен Гансом Винклером в 1920 г.

Геномика – раздел молекулярной генетики, изучает геном и гены живых существ. Занимается созданием карты метаболических сетей.

Гомология (греч. gomologia – соответствие) – сходство основных структур и органов организмов, основанных на общем генетическом наследстве.

Докинг – молекулярный докинг – это метод молекулярного моделирования, который позволяет предсказать наиболее выгодную для образования устойчивого комплекса ориентацию и положение одной молекулы по отношению к другой.

Метаболизм (греч. metabole – перемена, превращение) – совокупность химических реакций, протекающих в живых клетках и обеспечивающих организм веществами и энергией для его жизнедеятельности, роста и размножения.

Метаболомика дает возможность понять, как происходит обмен веществ в клетке, изучить и смоделировать метаболизм, исследовать совместимость функций элементов биологической системы и, как следствие, ускорить процесс создания лекарственных препаратов.

Нуклеотиды – фосфорные эфиры нуклеозидов, нуклеозидфосфаты. Свободные нуклеотиды, в частности, АТФ, АДФ, играют важную роль в энергетических и информационных внутриклеточных процессах.

Онтология – раздел философии, в котором рассматриваются всеобщие основы, принципы бытия, его структура и закономерности.

Пептиды (греч. peptos – сваренный, переваренный) – органические вещества, состоящие из остатков одинаковых или различных аминокислот, соединенных пептидной связью. Другими словами, пептиды – это универсальные белки.

Протеомика изучает и получает данные о сотнях тысяч пептидов и белок-белковых взаимодействиях, иначе – занимается инвентаризацией белков, т.е. реально работающих молекулярных машин в клетке. Это новая наука, выросшая из геномики.

Секвенирование (англ. sequence – последовательность) – это определение первичной или аминокислотной, или нуклеотидной последовательности белков и нуклеиновых кислот: ДНК и РНК.

Таксономия (греч. taxis – расположение, порядок и nomos – закон) – теория классификации и систематизации сложноорганизованных областей действительности, имеющих обычно иерархическое строение.

Транскриптомика – изучает активность генов, получает данные о концентрации десятков тысяч матричных РНК.

Фаг (греч. fagos – пожиратель) – вирус, избирательно поражающий бактериальные клетки.

Фармакогенетика (греч. pharmacon – лекарство + генетика) – раздел медицинской генетики и фармакологии, изучающий характер реакций организма на лекарственные средства в зависимости от наследственных факторов.

Филогенетика, или филогенетическая систематика – область биологической систематики, которая занимается идентификацией и прояснением эволюционных взаимоотношений среди разных видов жизни на Земле, как современных, так и вымерших.

ССЫЛКИ ПО ТЕМЕ

1. http://www.bioinformatics.ru
2. http://www.rusbiotech.ru
3. http://elementy.ru/lib/430895
4. http://www.sibai.ru
5. http://lake.baikal.ru
6. http://www.jcbi.ru
7. http://www.rtcb.iitp.ru/index_r.html
8. http://medi.ru/pbmc/8800101.htm
9. http://www.mgs.bionet.nsc.ru/mgs/
10. http://www.BioinforMatix.ru

Г.П. Несговорова

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ГУМАНИТАРНЫХ ИССЛЕДОВАНИЯХ И ГУМАНИТАРНОМ ОБРАЗОВАНИИ

ВВЕДЕНИЕ

На рубеже XX–XXI вв. в общественной жизни всего мира и нашей страны в частности произошли существенные изменения, связанные с формированием нового типа общественного устройства – информационного общества. Если до этого времени наука была озабочена приумножением и накоплением знаний, то теперь она сосредотачивается на способах овладения накопленными знаниями, признавая главенствующую роль информатики в своем дальнейшем развитии. Само понятие «информационное общество», ставшее ключевым понятием социальных наук, означает: ориентация на знания, цифровая форма представления объектов, инновационная природа и виртуализация производства, динамизм социальных процессов, оценка эффективности личности как человека, владеющего информационно-коммуникационными технологиями. Создание в послевоенные годы нового научного направления в виде методов и средств сначала кибернетики, а затем информатики существенно изменило образ мира. Рейтинг и могущество страны в мировом сообществе стали определять ее информационные возможности. С процессом развития информационного общества и новых информационных технологий связаны также и интенсивные процессы становления новой образовательной парадигмы, идущей на смену классической.

Внедрение персональных компьютеров, развитие информационно-коммуникационных технологий оказывает заметное влияние на развитие человека, на его мировоззрение, систему личностных ценностей. Все более погружаясь в виртуальные миры, человек сталкивается с необходимостью изменения стиля жизни, образа мышления, характера взаимоотношений с окружающим миром. Дальнейшая информатизация требует не только компьютерной грамотности, но и определенного уровня информационной культуры, основанной на понимании закономерностей развития информационного общества.

Таким образом, современный период развития общества характеризуется сильным влиянием на него компьютерных технологий, которые про-

никают во все сферы человеческой деятельности, обеспечивают распространение информационных потоков в обществе, образуя глобальное информационное пространство. Неотъемлемой и важной частью этого процесса является компьютерное образование и использование информационных технологий. Прежде чем говорить об информационных технологиях, неформально определим их.

Информационные технологии – это процесс, использующий совокупность средств и методов сбора, обработки и передачи данных (первичной информации) для получения информации нового качества о состоянии объекта, процесса или явления. В последние годы термин «информационные технологии» выступает синонимом термина «компьютерные технологии», так как все информационные технологии связаны с применением компьютера. Но при этом термин «информационные технологии» намного шире и включает в себя то, что связано с «компьютерными технологиями».

Внедрение современных информационно-коммуникационных технологий в науку и образование инициировало рост прикладных исследований во многих гуманитарных и социальных областях знаний. В конце прошлого века применение компьютеров и информационных технологий было прерогативой специалистов и студентов естественнонаучных дисциплин. С развитием самой вычислительной техники и программного обеспечения к ней информационные технологии все уверенней стали использоваться в гуманитарных областях знаний.

В современном обществе работу специалиста любого профиля невозможно представить без применения средств вычислительной техники и информационных технологий. Использование информационных технологий позволяет повысить эффективность принятия многих решений за счет своевременного получения необходимой информации. В наши дни информационные технологии играют связующую роль между естественными и гуманитарными науками.

Гуманитарные науки и гуманитарное образование, естественно, являются неотъемлемой частью всех других наук и образования как такового. Им присущи те же проблемы, что и всему остальному миру познания, а также некоторые особые проблемы, сугубо гуманитарные, такие как сохранение и систематизация знаний, например, с целью общедоступности культурного наследия человечества, представленного древними рукописями, книгами, разными видео- и аудиодокументами, как в виде артефактов, так и в цифровом.

Таким образом, гуманитарные области знаний обладают своей спецификой. С учетом этой специфики надо определить наиболее эффективные

пути использования информационных технологий для гуманитарных научных исследований и гуманитарного образования. Темпы внедрения информационных технологий в науку, образование, культуру да и в саму повседневную жизнь возрастают с каждым днем. Информационные технологии, связанные с применением компьютеров в научных исследованиях и в образовании, становятся достоянием гуманитариев не в меньшей степени, чем представителей естественных и технических наук. Сейчас происходит сближение, а часто и синтез гуманитарного и естественно-научного знания. В новой информационной среде все знания представляются как единая динамичная система.

1. ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ГУМАНИТАРНЫХ ИССЛЕДОВАНИЯХ

Российские специалисты все активнее включаются в разработку перспективных проблем использования информационных технологий и компьютерной техники в гуманитарных исследованиях, методов искусственного интеллекта, мультимедиа технологий и т.д. Именно это направление и является определяющим в развитии гуманитарных наук в наши дни и в будущем.

Период становления информационного общества совпал с процессом глобализации, который характеризуется сверхбыстрым развитием электронной коммерции, скоординированностью финансовых рынков, развитием наднациональных организаций. Глобализм есть результат важных процессов обмена информацией всех со всем обществом. В информационном обществе сегодня сотрудничают разные ученые: экономисты, психологи, лингвисты, естествоиспытатели, физики, математики, информационные специалисты и люди многих других профессий.

1.1. Метод междисциплинарного анализа как основа применения информационных технологий в гуманитарных исследованиях

В основе применения информационных технологий в гуманитарных исследованиях прежде всего лежит метод междисциплинарного анализа. Междисциплинарный характер исследования информационного общества предполагает комплексный подход не только к теории информации и коммуникации, но и к таким гуманитарным наукам, как социология, психология, политология, культурология, этика, экономика и др. Например, среда Интернет («киберпространство», «виртуальная реальность») – есть объект

психологии. С точки зрения психологии, Интернет – это взаимосвязь 3-х основных человеческих потребностей: коммуникативной, когнитивной, игровой.

Логика развития гуманитарных наук сегодня движется в направлении междисциплинарности, интеграции, что позволяет не просто объединять инструментарии отдельных наук, но и вырабатывать некие общие основания для гуманитарных и естественнонаучных исследований. Междисциплинарные исследования наиболее эффективны, если существует общий объект, на который нацелены методы различных наук. И таким объектом междисциплинарных исследований являются информация и информационные технологии, направленные на эффективную обработку, хранение и поиск нужных данных.

Академическое направление компьютерной этики, сформировавшееся в 80-е гг. прошлого века в США, ставит и решает проблемы предотвращения несанкционированного доступа к приватной информации, этических аспектов интеллектуальной собственности, неконтролируемого распространения заведомо ложных данных, т.е. тут мы имеем дело с этической картиной сети с позиции поведения ее пользователей, демонстрирующих взаимосвязь технологий с моральными и социальными ценностями.

Поликультурный контекст сетевых технологий представляет интерес для всех специалистов, изучающих человека и осмысливающих гуманитарные эффекты Интернета. Так, социология Интернета делает социальный анализ формирующейся информационной среды в обществе. Предмет ее изучения – аудитория Интернета и формы социокультурного взаимодействия между людьми при обмене социальной информацией. Настораживающие тенденции здесь обозначены как риски информационного общества, связанные с замещением духовной культуры узкопрофессиональными знаниями, деформацией досуга, ориентацией на развлечения, вытеснением реального живого общения, изменением характера человеческого мышления от творческого к формализованному.

Актуальная и новая тема исследования социологической науки – «цифровой раскол» («digital divide») – расслоение общества по принципу вовлеченности в мир информационно-коммуникационных технологий, и как следствие – некоторая маргинализация населения. Элитарная (в некоторой степени) система Интернет-услуг вызывает некий вид социального неравенства внутри общества. Информационно-коммуникационные технологии меняют ситуацию в фундаментальной науке, появляются новые методологические подходы, такие как «управление знаниями» («knowledge management»), «понятийные сети» («knowledge networks»).

Социальная теория сегодня – это анализ существующих форм общественной жизни, объемный обзор повседневных событий, понятийное формулирование новых понятий и реалий жизненного мира, к которым можно отнести:

1) социальные отношения в информационном обществе (интернетный стиль жизни, информационное поведение, информационная грамотность и культура, сетевое общение, компьютерофобия);

2) социальные общности, возникающие в информационно-коммуникационном пространстве, или в иносфере (экологические, когнитивные, гендерные);

3) социальные процессы в информационном обществе (глобализация социальных процессов и их столкновение с приватной жизнью, зомбирование общества, информационные войны, компьютерная преступность, антиглобализм).

Таким образом, информационные технологии, с одной стороны, способствуют развитию смежных дисциплин, внедряясь в них своими методами и обогащая их изучение, а с другой – сами являются предметом изучения с точки зрения информационно-коммуникационных технологий. Информационные взаимодействия входят в предметные области самых разных дисциплин, т.е. изучение информационного общества является принципиально мультидисциплинарной областью науки.

Информационные технологии долго не использовались в гуманитарных исследованиях. Это объясняется тем, что компьютерные технологии развивались исходя из потребностей физико-математических и технических наук, а связи между ними и гуманитарными науками не всегда были столь тесными, какими они становятся теперь. Какое-то время гуманитарии не рассматривали компьютер как реальный научный инструмент в своих исследованиях. Только в конце 20-го века иформационные технологии начали достаточно активно проникать в гуманитарные дисциплины. Хотя уже после появления микрокомпьютеров наиболее распространенной сферой их применения в гуманитарных науках стала лингвистика, а именно: обработка текстов с цель автоматизации создания частотных словарей, автоматическое реферирование статей, машинный перевод с одного языка на другой и пр.

Гуманитарии для начала получили доступный и удобный инструмент для хранения и поиска информации. Сначала это были базы данных, информационно-поисковые системы, потом – базы знаний, экспертные системы, системы искусственного интеллекта, например, когнитивные компьютерные модели понимания текста. Появилось новое понятие – образова-

тельная, информационная среда. Глобальная мировая информационная среда постоянно увеличивает число источников, которые могут быть использованы в гуманитарных исследованиях, но это порождает проблемы технического и нравственного плана (как найти нужный материал и как при этом не украсть его).

Сейчас, например, в РГГУ (Российский государственный гуманитарный университет), одном из главных гуманитарных учебных заведений России, разработана новая концепция организации научно-учебного процесса. Основная идея этого подхода – создание прежде всего с помощью информационных и информационно-коммуникационных технологий специальной среды, особого пространства информации, обеспечивающей полное усвоение материала изучаемой проблемы студентами при одновременном усилении творческой и методологической составляющей исследовательского и преподавательского труда.

Представляется, что современное исследовательское информационное пространство должно включать в себя 4 главных компонента:

1) информационные ресурсы – документированные данные и знания, циркулирующие по информационным каналам и накапливаемые в информационных хранилищах: архивах, фондах, базах данных и т.д. в виде, пригодном для широкого использования;

2) информационное сообщество – множество субъектов (организаций и физических лиц), действующих в едином информационном пространстве сферы различных исследований в качестве генераторов информации и различного рода потребителей этой информации;

3) информационная структура – информационные каналы и хранилища, информационные технологии, правовая и финансово-экономическая деятельность информационного общества;

4) средства и методы, обеспечивающие информационную деятельность, совокупность процессов и действий, осуществляемых информационным сообществом с использованием информационных ресурсов и инфраструктуры с целью производства научных исследований.

Мы все живем в эпоху, когда информационные технологии проникли абсолютно во все отрасли человеческой деятельности. Рассмотрим, к примеру, такую отрасль, как экономика. Существенной частью управления хозяйством являются информационные технологии. Без них невозможно ни экономическое планирование производства, ни распределение ресурсов, ни выявление пропорций и связей в экономике, ни осуществление руководства, управления и контроля на предприятиях в отрасли, в регионе и в целом в экономике. С помощью компьютеров осуществляют имитационное

моделирование. Модели этого класса реализуются в виде алгоритмов и программ, отражающих довольно сложные зависимости, не поддающиеся простому аналитическому методу. Этот способ моделирования широко применяется для исследования проблем развития городов, регионов, различных экономических, экологических и других сложных проблем. Благодаря сети Интернет при проведении научных исследований осуществляется быстрое обращение к разным электронным источникам с целью получения необходимой информации.

Для решения экономических задач применяют автоматизированные системы управления и автоматические системы обработки данных. Использование таких систем помогает находить оптимальные варианты, позволяющие разрешать различные вопросы, требующие в процессе поиска не только скорости и больших объемов вычислений, но и гибкости, динамизма, неординарных подходов. Сейчас существует множество программных продуктов, позволяющих решать различные задачи в гуманитарных исследованиях от бухгалтерской деятельности в экономике до сложных социологических, экологических, археологических, исторических, этнографических, лингвистических, правовых и других гуманитарных задач. Информационные технологии в этих случаях предоставляют инструментарий, позволяющий многократно ускорить процесс проведения таких исследований, повысить их точность и сократить трудоемкость.

В настоящее время компьютеры и связанные с ними информационные технологии проникают везде, в том числе во все области гуманитарных знаний. Подробный анализ применения информационных технологий во всех гуманитарных отраслях не укладывается в рамки данной статьи, поэтому ниже остановимся лишь на некоторых гуманитарных дисциплинах из всего их многообразия.

1.2. Информационные технологии в исторической науке

В качестве примера использования информационных технологий в гуманитарных исследованиях остановимся на исторической науке. Ранее, в [1], речь шла о применении компьютеров и программных средств при обработке текстов в компьютерной лингвистике.

Информация как универсальная категория играет очень важную роль в современном знании и сама становится в последние годы центральным объектом исследования гуманитарных наук. Усложнение общественной жизни сопровождается расширением объемов информации, что, в свою очередь, ведет к увеличению потребности в обмене информацией. Таким

образом, чем шире потребность в информации и информационном обмене, тем больше информации распространяется в обществе, т.е. она представляет собой фактор, управляющий общественным развитием.

Наиболее актуальной информация становится в переломные моменты истории, когда происходят качественные преобразования в природе, обществе, науке и т.д. Все переломные моменты в истории общества связаны с накоплением информации и появлением новых способов и средств информационного обмена. Таким образом, информация оказывается главным источником движения общества. Информационная деятельность составляет основу механизма развития социальной инфраструктуры и потому предваряет, сопровождает и завершает любую деятельность субъекта.

Внедрение новых информационных технологий в исторические исследования началось с археологии. Например, с помощью информационных технологий можно создавать историко-хронологические схемы развития культур разных периодов, основанные на методах многомерного статистического анализа с использованием стандартного и оригинального программного обеспечения. Освоение компьютерных технологий обработки и хранения данных, в том числе работы с системами управления базами данных, позволяет вплотную подойти к созданию банков данных, которые могут в совокупности составить единый архив машиночитаемой исторической информации для образования и исследований, например, банк данных археологических изысканий, к параметрам которых относится как информация о расположении памятников, так и информация об обследованиях, и, что особенно важно, – информация о состоянии памятников на момент их выявления. Можно создавать разные базы данных: от демографического состава сибирской купеческой семьи 19-го – начала 20-го вв. до всех населенных пунктов какого-то определенного края.

Историческое компьютерное картографирование связано с созданием карт определенного региона. Неоценима роль компьютерного картографирования в исторических исследованиях и его соотношение с исторической географией и исторической картографией.

Для систематизации и накопления исторического материала с помощью информационных технологий формируются разного рода библиотеки электронных ресурсов, т.е. собственно исторические источники. В такую библиотеку входят следующие разделы:

1) базы данных – в виде архивных файлов;

2) электронные изображения – с цифровыми изображениями страниц старинных книг;

3) набор электронных текстов исторических источников – здесь собраны такие источники, которые повсеместно используются и в научных работах, и в учебном процессе и которые издавались либо давно, либо ограниченным тиражом.

Таким образом, информационные технологии в исторической науке используются как при создании, систематизации, накоплении, обработке и поиске рабочего материала по заданной научной тематике историка, так и для применения уже готовых электронных отцифрованных данных в виде текстов и мультимедийных средств. Примером методических материалов нового поколения является, например, для историков – энциклопедия на CD ROM «История России» и множество других электронных справочных материалов, которые можно найти в Интернете, что значительно ускоряет и улучшает работу над конкретной темой.

2. ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ГУМАНИТАРНОМ ОБРАЗОВАНИИ

В настоящее время все более возрастает роль информационно-коммуникационных технологий во всем образовании, и, в частности, в гуманитарном. Исторически сложилось так, что информационные технологии развивались сами по себе, находя свое применение в точных дисциплинах, а гуманитарное образование – само по себе, независимо от них. Делались попытки приобщить гуманитариев к информационным технологиям посредством преподавания им программирования, но ничего из этого не получилось: психологически люди с гуманитарным складом ума не желали (или не могли) осваивать, как они считали, «точные» науки. И только с развитием вычислительной техники и программного обеспечения в виде языков программирования высоких уровней, информационные технологии стали находить свое применение не только в гуманитарных исследованиях, но и в гуманитарном образовании. Сейчас информационные технологии обеспечивают всеобщую компьютеризацию преподавателей и учащихся гуманитарного профиля на уровне, позволяющем решать три основные задачи:

1) обеспечение выхода в Интернет каждого участника учебного процесса, в любое время и из любого места;

2) развитие единого информационного пространства образовательной индустрии и присутствие в нем в различное время и независимо друг от друга всех участников образовательного процесса;

3) создание, развитие и эффективное использование управляемых информационных образовательных ресурсов.

Образовательную среду, в которой осуществляются образовательные информационные технологии, определяют следующие работающие в ней компоненты:

1) техническая (вид используемой компьютерной техники и применяемые средства связи);

2) программно-техническая (программные средства поддержки реализуемой технологии обучения);

3) организационно-методическая (инструкции учащимся и преподавателям, организация всего учебного процесса).

Процессы информатизации современного гуманитарного образования не должны пониматься только как формирование технических средств, повышающих внешнюю эффективность образовательного процесса. Они представляют те социокультурные изменения, которые меняют задачи и образ гуманитарного образования в современном мире. Формирование в процессе образования творческой личности, способной к эффективной деятельности, определяет содержание целевых установок всего образования. Существует понятие модели компетентности личности как цели образовательной системы. Эта модель включает:

– социальную компетентность, которая состоит в освоении правил и норм базовых социальных практик;

– осознание собственной социальной и культурной идентичности в историко-культурном окружении;

– интеллектуальную и коммуникативную компетентность, которая состоит в способности отстаивать свою точку зрения, критически ее проверять, изменять, в умении использовать современные способы и технологии коммуникации;

– мировоззренческую компетентность, которая предполагает терпимость к ценностям и нормам других культур, принятие ценностей социальной и культурной инициатив;

– профессиональную компетентность.

Данная модель основывается на представлении о гуманитарном измерении информационного общества и приоритетах современного гуманитарного образования. В индустриальном обществе приоритет отдавался прагматическому, технократическому образованию. Особенностью современной ситуации становления информационного общества является то, что приоритет приобретает гуманитарное образование, которое рассматривает раз-

витие личности и создание условий ее безопасного и достойного существования как цель, а не как средство.

Образованный человек должен быстро и эффективно решать задачи, определяющие уровень его личной компетентности, от этого зависит его социальное качество. Кроме того, уровень функционального совершенствования специалиста зависит от того, насколько успешно соединены функциональные знания и гуманитарная подготовка в его профессиональном образовании. Традиционное представление о гуманитарном образовании состоит в том, что оно нацелено на формирование фундаментальных основ, которые позволяют человеку решать мировоззренческие задачи, ориентироваться в современной социокультурной обстановке.

Становление информационного общества связано с изменением задач гуманитарной подготовки. В современных условиях эти задачи могут быть искажены. Это связано с двумя крайностями в понимании функций и особенностей гуманитарного образования.

Во-первых, гуманитарное образование понимается как способ конструирования человеческого бытия, как универсальное средство решения идеологических задач. Оно рассматривается как форма манипулирования человеческим сознанием.

Во-вторых, утверждается вторичность и внешний характер гуманитарного образования. Этот подход базируется на представлении о технократическом обществе, где гуманитарное образование не определяет уровень и качество развития технологий, а потому не влияет на качество жизни и не составляет главный ресурс развития общества.

Изменение ориентиров гуманитарного образования связано с тем, что оно становится фундаментальной подготовкой. Социальные практики становятся интеллектуально насыщенными, поэтому фундаментальность современного гуманитарного образования нацелена на формирование фундаментальной структуры человеческого бытия.

Гуманитарное познание в информационном обществе приобретает, как уже упоминалось выше, междисциплинарный характер. Комплексный характер определяет эффективность, теоретическую продуктивность и направление исследований. Гуманитарная наука развивается по пути возникновения специализаций, появления смежных областей исследования. Информационные процессы и структуры, их обеспечивающие, меняют не только техническую, инструментальную сторону исследования, формы и способы работы с гуманитарным материалом, но и сами становятся предметом исследования и фактором, определяющим представление о гуманитарном исследовании и его тематической области.

Развитие гуманитарного образования в условиях информационного общества направлено на его интеграцию с естественнонаучным образованием. Это обеспечивает строгость и точность методической и технической сторон гуманитарного образования, что во многом определяет его объективность и результативность. Преобразование и развитие гуманитарного образования во многом определяется процессами информатизации и становления информационной техники. Внедрение информационных технологий в структуру гуманитарного образования приводит к появлению новых отраслей научного знания, специализаций гуманитарного направления. Одним из основных направлений развития гуманитарного образования является повышение его эффективности посредством достижения строгости и точности гуманитарного познания, возможности обработки большого массива информации, которая обеспечивается использованием математического аппарата и средствами информационных технологий.

2.1. Единая образовательная модель гуманитарного образования

Современный человек должен не только обладать неким объемом знаний, но и уметь учиться: искать и находить необходимую информацию, чтобы решать те или иные проблемы, использовать разные источники информации для решения этих проблем, постоянно приобретать новые знания в области информационных технологий, чтобы использовать их в учебе и исследованиях. Необходимо построение единой образовательной модели гуманитарного образования, основанной на современных информационных и педагогических технологиях, на методологических принципах информационного общества. В настоящее время существует множество определений понятия модели, остановимся на следующем: образовательная модель – это некоторая структура знаний, которая играет существенную роль в планировании наших предстоящих действий и позволяет нам в таких действиях использовать ранее приобретенные знания. Информатизация выступает как основной механизм реализации новой образовательной парадигмы, как новое качество системного образования, как средство реализации функции прогнозирования образовательной системы. Информатизация способствует новому синтезу гуманитарных и естественно-технических наук. Современное образование должно строиться на междисциплинарности, которая должна проявляться и в методологии, и в образовательной практике.

Интерес к теории информации тесно связан с процессами информатизации, которые на рубеже 80-90 гг. прошлого века вовлекли и социально-гуманитарные науки. Подтверждением тому стало появление в 1990 г. «от-

раслевых» информатик в целом ряде научных гуманитарных областей (социальная, экономическая, правовая, историческая, археологическая, психологическая, лингвистическая и т.д.). Но в развитии любой отраслевой информатики происходит перекос в сторону прикладных исследований.

Информатизация гуманитарных наук позволила выявить глубокие социальные изменения, вызванные этим процессом. Это привело к необходимости переосмысления понимания информатики, изменения научных взглядов на саму науку информатику, выяснение механизмов влияния информатизации на развитие общества, на формирование нового этапа информационной культуры, построение теорий информационного общества и т.д.

Надо отметить также, что использование информационных технологий в преподавании гуманитарных дисциплин предъявляет повышенные требования к профессиональным качествам преподавателей, требует от них приобретения новых профессиональных знаний. Информационное пространство само по себе является важнейшим объектом гуманитарного исследования: такова, например, виртуальная среда, которая интенсивно развивается, но не всегда однозначны социальные, психологические и нравственные последствия этого явления.

2.2. Дистанционное обучение как применение информационных технологий в образовании

В качестве одной из информационных технологий, применяемых в образовании, и надо сказать не только в гуманитарном, следует отметить дистанционное обучение, которое стало возможным и получило широкое развитие в наши дни благодаря сети Интернет, которую определяют как «гипертехнологию, включающую в себя все остальные, и ее успех объясняется тем, что она может дать всем все»... Дистанционное образование уверенно приходит на смену заочному.

Разработчики дистанционного образования конкретизируют индивидуализацию образовательного поведения следующим образом (считая, что в дистанционном обучении наиболее ярко проявляются черты личностно-ориентированного способа обучения):

1) гибкость – обучающий сам планирует время, место и продолжительность занятий;

2) модульность – материалы для изучения используются в виде модулей, что позволяет обучаемому выбирать самому, что ему предпочтительнее;

3) доступность – независимость от географического и временного положения обучающегося;

4) рентабельность – экономическая эффективность проявляется за счет уменьшения затрат на содержание площадей образовательных учреждений и на печать, размножение (включая бумагу) учебных и методических материалов;

5) мобильность – эффективная реализация обратной связи между преподавателем и обучаемым;

6) охват – одновременное обращение ко многим источникам информации (электронным библиотекам, банкам данных, базам знаний, экспертным базам и т.д.) большого количества обучающихся;

7) технологичность – использование в образовательном процессе новейших достижений информационных и телекоммуникационных технологий.

Современные компьютеры позволяют с большой эффективностью воспроизводить все виды передачи информации. Только они могут реализовать адаптивные алгоритмы в обучении и обеспечить преподавателя объективной и оперативной обратной связью о процессе усвоения учебного материала. Поэтому принципиальное отличие дистанционного обучения в сегодняшнем его понимании от традиционного заочного заключается не только в том, что «перо и бумагу» заменяет компьютер, а почту – Интернет, но и в том, что мультимедийный компьютер, будучи интегрированным носителем информации, наиболее полно и адекватно отображает модель «face to face». И только в компьютерах могут быть реализованы информационно-справочные системы на основе гипермедийных ссылок, что является одной из важных составляющих индивидуализации обучения.

Дистанционное обучение в наши дни не обходится без электронных учебников, являющихся аналогом «бумажных» учебников и разного рода методической литературы. Это, в свою очередь, породило и способствовало развитию широкой системы электронного библиотечного обслуживания, которое включает:

– сеть информационных хранилищ, электронных изданий с развитой системой доступа как с отдельных рабочих мест, так и через электронные читальные залы;

– обширную сеть информационно-поисковых систем по библиотекам;

– базу подготовки и выпуска электронной информации (электронные издания);

– единую систему электронных каталогов.

Все это способствует созданию комплексов рабочих мест с открытым доступом в сети Интернет.

Что же касается собственно электронных библиотек, то они обслуживают не только гуманитариев, но и специалистов разных областей знаний. В своей деятельности они сами, с одной стороны, пользуются информационными технологиями, а с другой, – этими технологиями и являются для научных сотрудников и студентов.

ЗАКЛЮЧЕНИЕ

Подводя итог сказанному выше, хочется надеяться, что информационные и информационно-коммуникационные технологии и в дальнейшем будут шире и глубже проникать в гуманитарные исследования и гуманитарное образование, будут создаваться новые программные продукты, помогающие гуманитариям-исследователям и студентам различных гуманитарных специальностей в их работе и учебе. Развитие современного российского общества во многом зависит от реформы образования, в частности, от внедрения и использования информационно-коммуникационных технологий в образовательном процессе.

Удивительным образом новая информационная среда полностью отвечает самым насущным потребностям гуманитарного образования. Она в полной мере может быть использована в гуманитарном вузе. Потребность в гуманитарной культуре будет только возрастать. Получая доступ к современным базам данных, будущий гуманитарий может строить свое индивидуальное информационное пространство. Информационная свобода, таким образом, является условием и формой свободы политической и творческой. В обеспечении этой свободы – главный смысл информатизации гуманитарного образования. Создание программного инструментария для гуманитарных наук, использующего современные способы работы с информацией, носит, в свою очередь, вполне гуманитарный характер.

Следует также отметить, что идеями гуманизации и сопутствующей гуманитаризации образования должно быть пронизано и вузовское прикладное физико-математическое образование. Уместно напомнить общеизвестное положение о том, что гуманитарные знания формируют целостный образ мира, в котором такие понятия как любовь, забота о мире и человеке, ощущение красоты мира продолжают существовать.

СПИСОК ЛИТЕРАТУРЫ

1. Несговорова Г.П. Информатизация гуманитариев, гуманитаризация информатиков // Проблемы системной информатики. – Новосибирск: ИСИ СО РАН, 2010. – С.179-187.
2. Лукина Н.П. Информационное общество: состояние и перспективы социально-философского исследования. – Открытый междисциплинарный электронный журнал ТГУ «Гуманитарная информатика». – http://huminf.tsu.ru
3. Можаева Г.В. Роль исторической информации в современном источниковедении. – Открытый междисциплинарный электронный журнал ТГУ «Гуманитарная информатика». – http://huminf.tsu.ru
4. Завьялова М.П., Сухушин Д.В. Изменение установок гуманитарного образования в условиях становления информационного общества. – Открытый междисциплинарный электронный журнал ТГУ «Гуманитарная информатика». – http://huminf.tsu.ru
5. Титова С.В. Информационно-коммуникационные технологии в гуманитарном образовании: теория и практика. – Пособие для студентов и аспирантов языковых факультетов университетов и вузов. – М., П-Центр, 2009.

А.П. Стасенко

ТЕСТИРОВАНИЕ ИЗМЕНЕНИЙ В ПРОГРАММНОЙ СИСТЕМЕ НА ОСНОВЕ ПОКРЫТИЯ ИСХОДНОГО КОДА

ВВЕДЕНИЕ

Регулярное тестирование сборок программной системы является основой современных практик разработки ПО [1]. Широкое распространение получила технология непрерывной интеграции, которая предполагает тестирование каждой правки исходного кода в дополнение к традиционному тестированию ночных сборок. Развитием данной идеи является использование набора тестов, запускаемого перед отправкой изменений в основную ветвь разработки; при этом 100% прохождение данных тестов является необходимым условием для попадания правок разработчика в хранилище исходного кода. Такой подход позволяет избегать ситуаций, когда серьезная ошибка, допущенная одним из разработчиков, блокирует работу целой команды.

Для сложных программных систем, состоящих из множества взаимодействующих компонент, характерно использование объемного предварительного тестирования, в связи с необходимостью проверки работоспособности каждой из составляющих частей системы. Ручное определение состава тестирования не всегда надежно, так как правки в одной из компонент могут отразиться на работе системы совершенно неожиданным для разработчика образом. На практике это приводит к существенному замедлению процесса разработки, так длительность предварительного тестирования является одним из факторов, ограничивающих скорость работы над проектом.

Подходы к сокращению объемов тестирования, основанные на сопоставлении изменений в исходном коде с данными о тестовом покрытии, описаны в литературе и хорошо изучены с теоретической стороны [2–3]. Их основная идея состоит в исключении из тестового прогона тестов, не покрывающих изменившиеся части программы. Внедрение подобных методик в процесс разработки больших программных систем сталкивается с рядом трудностей, которые во многом определены объемами данных о тестовом покрытии. Наибольшую проблему представляет поддержание этих данных в актуальном состоянии, так как тестирование инструментированных сборок идет в несколько раз (как правило, в 3–5 раз) дольше обычных тестирований [4]. Таким образом, возрастающая нагрузка на тестирующую систему может свести на нет выигрыш, достигнутый сокращением объемов

тестирования. Передача, хранение и оперативный анализ данных о тестовом покрытии также вызывает затруднения, так как их объем может достигать сотен гигабайт [5].

В данной статье будет рассказано об опыте внедрения этой технологии в процесс разработки существующего программного продукта, написанного на языках Си и Си++. Описанные выше проблемы были решены при помощи снижения детализации данных о тестовом покрытии. Вместо традиционного покрытия базовых блоков используется покрытие процедур (модулей, классов либо других крупных частей программы). Данный подход позволяет инструментировать только точки входа в процедуры, что существенно (до 80%) сокращает стоимость обновления данных о покрытии. Вместо отдельных тестов рассматриваются логические группы тестов (исходя из предположения, что тестовая база имеет некоторую структуру), что сокращает объемы данных и повышает прозрачность работы алгоритма. Общее снижение детализации данных о тестовом покрытии влечет их меньшую изменчивость, что позволяет обновлять эти данные не для каждой правки в исходном коде, а на периодической основе. В статье будут освещены и другие вопросы, связанные с практическим внедрением данной методики, такие как разработка алгоритма анализа изменений в исходном коде.

ОБЩАЯ СХЕМА СИСТЕМЫ

На рис. 1 приведена общая схема генерации оптимизированного набора тестов.

Предполагается, что при разработке программной системы используется система контроля версий (svn, cvs, git). Под ветвью разработки системы подразумевается английский термин branch. Под локальной копией системы – термин checkout. Под изменением кода системы – термин checkin.

Как видно из схемы, в качестве большой подготовительной работы должна быть собрана информация о покрытии кода программной системы на всём наборе имеющихся тестов или как минимум на наборе тестов, использующихся для обычного предварительного тестирования изменений. В результате регулярного сбора информации о покрытии кода генерируется отображения имён функций исходного кода программной системы во множество тестов, исполнение которых приводит к вызову этих функций.

Далее разработчик формирует локальную копию системы и вносит изменение, которое требуется протестировать. На основании этого изменения

и информации о процессе построения системы находится список измененных функций. Информация о процессе построения системы требуется, чтобы получить точный набор опций компиляции (в частности, препроцессора) и сгенерированные заголовочные файлы, от которых могут зависеть измененные единицы трансляции, без чего невозможно получить исходный текст единицы трансляции без директив препроцессора.

Автоматическое определение имён изменившихся функций необходимо для исключения возможных ошибок, которые может допустить программист при ручном формировании. Также в случае изменений объявлений или определения модульных переменных список зависящих от них функций не всегда очевиден.

На финальном этапе происходит генерация оптимизированного набора тестов как объединения множеств тестов, полученных на этапе сбора информации о покрытии кода, для каждой измененной функции.

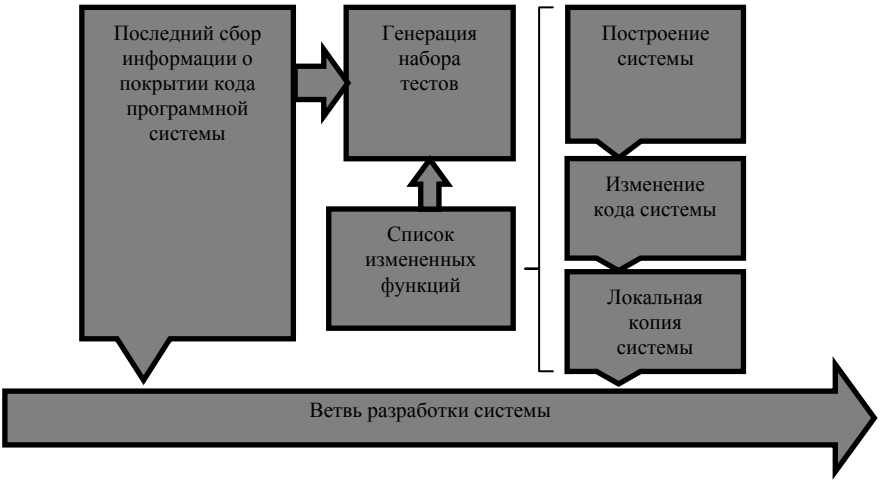


Рис. 1. Общая схема использования информации о покрытии кода для оптимизации набора тестов при тестировании изменений программной системы

РЕГУЛЯРНЫЙ СБОР ИНФОРМАЦИИ О ПОКРЫТИИ КОДА

Сбор информации о покрытии кода основывается на функциональности компилятора по сбору профилировочной информации (опция -prof-gen для

компилятора Intel, опция /profile для компилятора Microsoft, опция -p для компилятора gcc). Таким образом, требуется построить специальную версию программной системы с использованием этих ключей компилятора. Дальнейшее использование этой специальной версии системы при прогоне тестового набора будет вызывать создание специальных файлов с профилировочной информацией.

Как правило, из профилировочной информации можно извлечь подробные данные о покрытии линейных участков, однако для упрощения алгоритма поиска изменений в исходном коде и для сокращения объемов хранимой информации считаются только изменённые функции. Однако такое огрубление анализа сокращает количество тестов, которое можно удалить из регрессионного тестирования. Тест может покрывать функцию, затронутую изменением в исходном коде, но не покрывать ту часть её управляющего графа, где это изменение случилось.

Также для целей сокращения объемов хранимой информации было принято решение объединять информацию о покрытии кода для отдельных тестов в существующих тестовых сюитах (наборах тестов, имеющих общее назначение). Как уже было отмечено, регулярный сбор информации о покрытии кода требуется для генерации отображения имён функций в исходном коде программной системы во множество тестовых сюит, исполнение которых приводит к вызову этих функций.

Для целей сбора информации о покрытии кода нет необходимости выполнять все действия тестов, отличные от вызовов тестируемой программной системы, что позволяет существенно ускорить процесс сбора в некоторых случаях. Например, в случае тестирования компилятора можно не исполнять скомпилированные и слинкованные им тесты. Несмотря на эти возможности по ускорению прогона тестов, использование инструментированной программной системы для сбора информации о покрытии кода может замедлять исполнение тестов в несколько раз.

Как правило, сбор информации о покрытии кода требует незначительных модификаций тестовой системы для внедрения этапа объединения профилировочной информации после каждого теста тестовой сюиты, иначе в случае тестовых сюит с большим количеством тестов есть опасность переполнения дискового пространства машины профилировочной информацией индивидуальных тестов, что для каждого теста может занимать десятки мегабайт. Также в случае больших тестов, состоящих из тысяч вызовов программной системы, возникает потребность более раннего объединения профилировочной информации при достижении её определенного количе-

ства, что обычно реализуется с помощью оберток (wrapper) для вызовов программной системы.

Сбор информации о покрытии кода проводится на регулярной основе для ветви разработки системы, и его оптимальная частота зависит от типичной скорости добавления нового кода (функций) и времени, необходимого для сбора. Чем выше частота добавления новых функций или удаления существующих функций, тем чаще надо проводить новый сбор информации о покрытии кода. Однако, если время работы сбора занимает существенное время, то его частое исполнение может занять доступные тестовые ресурсы и свести на нет экономию от последующей оптимизации тестирования. Следует отметить, что регулярный сбор информации о покрытии кода может быть ценен не только для целей оптимизации тестирования изменений, но и, например, для планирования разработки новых тестов для слабо покрытых участков кода, поэтому в этом смысле для целей оптимизации тестирования регулярный сбор информации о покрытии кода может быть «бесплатным».

Временной разрыв между последним регулярным сбором информации о покрытии кода и изменением, для которого выполняется оптимизация регрессионного тестирования, строго говоря, нарушает свойство безопасности такой оптимизации. Безопасные (англ. *safe*) подходы либо не сокращают объемы тестирования, либо гарантируют, что исключение тестов не приведет к сокрытию дефектов. Как правило, происходит исключение тестов, результат исполнения которых не может быть затронут конкретными изменениями в исходном коде.

Для того чтобы явным образом нарушилось свойство безопасности, т.е. произошло сокрытие дефекта из-за сокращения объема регрессионного тестирования, необходимо выполнение следующих условий [4]:

1) агрегированные данные о тестовом покрытии должны были измениться с момента их последнего обновления, причем в «б**о**льшую» сторону;
2) рассматриваемая модификация исходного кода должна затрагивать часть программы, данные о покрытии которой устарели;
3) рассматриваемая модификация не должна затрагивать части программного кода, для которых данные о покрытии еще актуальны;
4) модификация должна привести к падению теста из группы тестов, которая была ошибочно исключена из тестирования;
5) дефект, вызвавший данное падение, должен быть специфическим для исключенной группы тестов и не проявиться на других тестах.

Сочетание данных событий возможно на практике, причем его вероятность тем выше, чем уже область программы, в которой в данный момент ведется активная разработка. В ходе практических экспериментов воспроизвести такую ситуацию не удалось. Следующие меры помогут снизить вероятность возникновения такой проблемы и опасность ее последствий: выбор удачного разбиения тестовой базы на группы, применение метода на подходящих стадиях жизненного цикла программного продукта, регулярное обновление данных о тестовом покрытии, периодическое тестирование на полной тестовой базе.

ПОИСК СПИСКА ИЗМЕНЁННЫХ ФУНКЦИЙ

После внесения изменения в исходный код программной системы для получения списка измененных функций требуется:

1) получить список измененных исходных текстов, которые для языков Си/Си++ можно разделить на два класса: единицы трансляции (translation unit) и заголовочные (header) файлы;
2) для каждого измененного заголовочного файла, используя данные о зависимостях между целями построения, определить множество зависимых от него единиц трансляции и объединить его с общим множеством измененных единиц трансляции;
3) препроцессировать текст единицы трансляции, для чего используются данные об опциях препроцессора и сгенерированных заголовочных файлах, полученные во время построения программной системы;
4) получить абстрактное синтаксическое дерево (AST) [6] для оригинальной и измененной версии единиц трансляции.

Использование сравнения абстрактных синтаксических деревьев позволяет игнорировать несущественные изменения в тексте программы, такие как изменения форматирования. В то же время препроцессирование встроенных макросов, таких как __LINE__ и __DATE__ добавляет неоднозначность, зависящую от форматирования программы. Для их устранения было принято решение предварительно обрабатывать все заголовочные файлы программной системы, текстуально заменяя эти макросы константными значениями, так как их переопределение через опции компилятора не всегда представляется возможным.

Изначально для построения абстрактного синтаксического дерева для поиска списка измененных функций был использован имеющийся синтакси-

ческий распознаватель (parser) языка Си/Си++ на языке Python, сгенерированный системой построения компиляторов ANTLR3. Однако высокое потребление памяти и низкая скорость работы такого Python распознавателя в реальных исходных текстах привела к поиску альтернативных решений, основанных на генерации внутреннего представления существующими компиляторами промышленного уровня.

Компилятор gcc предоставляет возможность получать абстрактные синтаксические деревья (AST) через опцию -fdump-syntax-tree, а специальная версия компилятора Intel позволяет генерировать дампы внутреннего представления и таблиц символов. Работающая версия системы основывается на дампах внутреннего представления компилятора Intel. Основное отличие данных дампов от абстрактного синтаксического дерева заключается в наличии отдельной модульной и локальной для каждой функции таблицы символов и заменой всех вхождений переменных на ссылки в эти таблицы.

С одной стороны, дампы компилятора позволяют дополнительно игнорировать некоторые несущественные изменения, такие как, например, переопределения типов, лишние скобки и иные несущественные для семантики элементы программы. С другой стороны, к именам переменных в дампах компилятора добавляется счётчик, который используется компилятором для разрешения неоднозначностей одинаковых имён в разных областях видимости. Таким образом, требуется исключать влияние случайного изменения счётчика для оригинальной и измененной версии программы. Также требуется устранять различия в декорировании имён Си++ (по крайней мере, в Windows), которые включают в себя дату изменения файла единицы трансляции.

Например, рассмотрим следующую простую Си-программу:

```
int i1 = 0;
int main() {
  int i2 = 0;
  return i1+i2;
}
```

Значительно усеченный дамп компилятора функции main и её символьная таблица будет выглядеть примерно следующим образом (жирным шрифтом выделены имена переменных с суффиксами добавленными компилятором для разрешения неоднозначности):

```
PACK| (i2.1) align: 0 MOD 4, size: 4, …
    VAR| (i2.1_V$1) type: SCALAR, size: 4,
       | offset: 0, esize: 4, …, edtype: SI32,  …
…
3       0           entry extern SI32  main
```

```
                    {
4       1               i2.1_V$1 = 0(SI32);
5       2               return ( (SI32) i1_V$0 + i2.1_V$1 );
6       3               return ;
                    }
Root Context C0.1 {
} C0.1
```

Модульная таблица символов выглядит примерно так:

```
PACK| (i1) align: 0 MOD 4, size: 4,
    | …, offset: 0, …
    VAR| (i1_V$0) type: SCALAR, size: 4,
           | offset: 0, esize: 4, …,
           | edtype: SI32,  …
    INIT| offset: 0, repeat: 1, ...,
           |data: 0(SI32)
```

Алгоритм нахождения изменённых функций между оригинальной и измененной версиями единицы трансляции работает следующим образом:

1) заменяет все вхождения имени переменной в функциях на её тип, её инициализацию и другие существенные семантические свойства в оригинальной и измененной версии единицы трансляции;
2) сличает все видимые извне единицы трансляции переменные с совпадающими именами между оригинальной и измененной версиями единицы трансляции и в случае нахождения различия в их семантических свойствах, сигнализирует, что уменьшение тестового набора не может быть осуществлено за приемлемое время, так как в этом случае требуется анализ всех единиц трансляции в программной системе;
3) текстуально сличает функции с совпадающими именами между оригинальной и измененной версиями единицы трансляции и сообщает функции с различиями в качестве искомых.

РЕЗУЛЬТАТЫ ОПТИМИЗАЦИИ ТЕСТИРОВАНИЯ

На рис. 2 приведены результаты оптимизации тестового набора, полученные для изменений, взятых за несколько месяцев в большой программной системе. Серый график показывает сокращение относительно полного набора тестов, а черный график – относительно набора тестов, обычно использующегося при тестировании изменений. График показывает процентное сокращение в наборе тестов и не учитывает их временные затраты, что, конечно, не совсем верно.

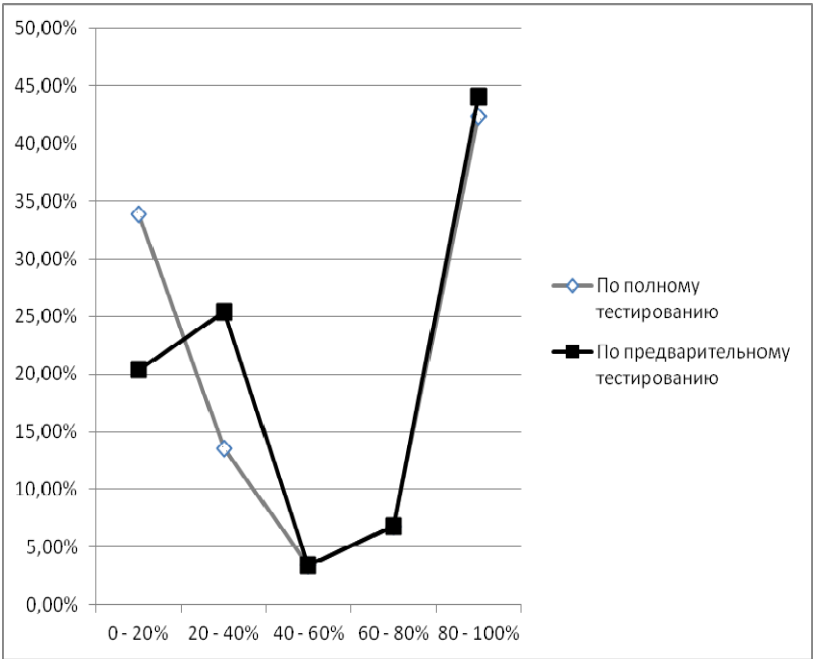


Рис. 2. Эффективность оптимизации тестирования изменений.
По оси X – процент тестов, оставшихся после оптимизации.
По оси Y – процент от общего количества изменений

Видно, что примерно в 45% случаев не удалось достичь существенного сокращения набора тестов (группа 80–100%). Однако для примерно 50% случаев удалось сократить объем тестов более чем в два раза, что является весьма неплохим результатом. Эксперименты также показали, что ожидаемо хорошо сокращалось тестирование у небольших изменений, тогда как тестирование больших изменений практически не сокращалось.

ЗАКЛЮЧЕНИЕ

Был налажен регулярный сбор информации о покрытии кода существующими тестами сложной программной системы, являющейся компилятором. Был разработан инструмент для определения измененных функций в

измененном исходном тексте программной системы, который реагирует только на фактические изменения синтаксической (AST дерева) или семантической (изменения определенных извне переменных) структуры функций. Для реализации инструмента использовались возможности по генерации внутреннего представления существующими компиляторами промышленного уровня. В результате применения данной системы в половине случаев удалось сократить объем требуемого тестирования изменений в два раза.

СПИСОК ЛИТЕРАТУРЫ

1. Leung H.K., White L. Insights into regression testing // IEEE Conf. on Software Maintenance. — IEEE Computer Society Press, 1989. – P. 60–69.
2. Rothermel G., Harrold M.J. A Safe, Efficient Regression Test Selection Technique // ACM Transactions on Software Engineering and Methodology. – 1997. –Vol. 6. – P. 173–210.
3. Rothermel G., Harrold M.J. Empirical Studies of a Safe Regression Test Selection Technique // IEEE Transactions on Software Engineering. – 1998. – Vol. 24.
4. A Framework for Reducing the Cost of Instrumented Code. Arnold M., Ryder B.G. б.м. // Proc. of the ACM SIGPLAN 2001 Confe. on Programming language design and implementation, 2001.
5. Салмин А.И. Исследование качества и эффективности функционального тестирования компилятора.– Новосибирск: НГУ, 2011.
6. Ахо А. Компиляторы: принципы, технологии, инструменты / Ахо А., Сети Р., Ульман Дж. — М: Издательский дом «Вильямс», 2003.

Т.В. Шманина

ИНФОРМАЦИОННАЯ СИСТЕМА ДЛЯ ПОДДЕРЖКИ ПРОЦЕССА ПРОВЕДЕНИЯ ИССЛЕДОВАНИЙ НА ОСНОВЕ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

ВВЕДЕНИЕ

Над решением технологических и исследовательских задач работает огромное число коллективов по всему миру, непрерывно производя большие объемы полезного знания. В ходе исследований коллективами могут применяться два основных метода: выбор оптимального решения текущей задачи из множества существующих решений (либо его построение на основе последних) и создание принципиально нового, оригинального решения проблемы.

Зачастую наиболее рациональным способом решения поставленных задач является первый путь. Однако он требует наличия информации об имеющихся разработках, которая чаще всего доступна в виде научных публикаций и технической литературы, объемы которой чрезвычайно велики и продолжают расти. Кроме того, с ростом объема производимого знания возрастает степень специализации исследователей, что не позволяет им быть осведомленными о многих потенциально полезных для их практики методах, подходах, а возможно, и целых областях знания.

Возникает проблема создания автоматических или полуавтоматических средств, упрощающих процесс поиска потенциальных решений поставленной перед исследователем задачи. Наилучшие потенциальные решения задач могут быть обнаружены в не связанных напрямую с тематикой решаемой задачи областях знаний. Поэтому такие методы поиска информации, как обычный (возможно, семантический) информационный поиск, кластеризация документов с целью группировки их по основной тематике и некоторые другие, зачастую лишь в незначительной степени упрощают задачу поиска решения, так как остается необходимость поиска взаимосвязей между методами, подходами и т.п.

Возможным ответом в вопросе поиска потенциальных решений некоторой задачи может стать автоматизация (полная или частичная) метода исследования на основе литературных источников, предложенного Доном Свенсоном в 1986 году.

В процессе исследования на основе литературных источников новое знание не генерируется на основе экспериментальных данных или путем логического вывода. Вместо этого производится попытка найти взаимосвязь между существующими знаниями, уже полученными экспериментальным или дедуктивным путем, посредством выделения ранее незамеченных взаимосвязей между сущностями, описанными в литературных источниках. Эта техника широко используется на практике уже долгое время [2].

Выделение таких скрытых взаимосвязей получило название «связывание Свенсона». Чаще всего этот процесс рассматривался исследователями в контексте биомедицинских задач. Типичным примером связывания Свенсона является следующий. Предположим, исследователь занимается разработкой лекарства от заболевания A (например, синдрома Рейно) и известно, что заболевание вызывается веществом B. Кроме того, известно, что препарат C уменьшает концентрацию вещества B в организме, и, таким образом, C может быть лекарством от болезни A. Однако последние данные публиковались отдельно от литературы, посвященной лечению A, поэтому взаимосвязь между болезнью A и препаратом C могла быть упущена (см. рис. 1). Связывание Свенсона имеет своей целью выделение таких взаимосвязей и предоставление их исследователю в явном виде.

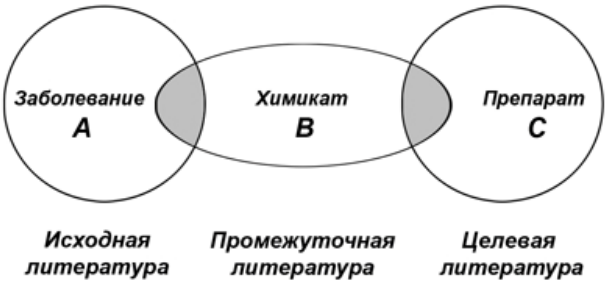


Рис. 1. Связывание Свенсона для заболевания A и препарата C

В ряде работ была предпринята попытка создания инструментов, автоматизирующих процесс построения связываний Свенсона для нахождения потенциальных решений задач непосредственно в виде концептов. Большинство этих работ было ориентировано на биомедицинскую область и

шинство этих работ было ориентировано на биомедицинскую область и основывалось на идее поиска взаимосвязей между разрозненными литературными источниками по общим ключевым понятиям, содержащимся в текстах. Упомянутые подходы, в отличие от предлагаемого в данной работе, стремятся выделить точные термины, определяющие потенциальное решение задачи, и предоставить их пользователю в виде упорядоченного по релевантности списка. К числу инструментов, применяющихся при создании таких моделей, относятся методы информационного поиска, латентное семантическое индексирование, использование внешних тезаурусов, онтологий и различных статистических характеристик для ранжирования потенциальных решений и выявления взаимосвязи между терминами [2].

Целью проводимого автором исследования является разработка подхода, позволяющего осуществлять анализ конечной локальной коллекции текстовых документов с целью построения сети взаимосвязи тем и проблем, затрагиваемых в литературе, для упрощения процесса исследования на основе имеющихся литературных источников.

МЕТОД АВТОМАТИЗАЦИИ ПРОЦЕССА ИССДЕДОВАНИЯ НА ОСНОВЕ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

В целях автоматизации процесса исследования на основе литературных источников автор данной работы предлагает метод, стремящийся имитировать процесс исследования научной и технической литературы человеком.

Принцип действия предлагаемого метода следующий. Алгоритм получает на вход формулировку некоторой проблемы в виде одного или нескольких ключевых терминов, а также конечную локальную коллекцию документов научного или технического плана. В процессе работы алгоритм сначала выделяет в коллекции документов темы, непосредственно связанные с введенной проблемой, а затем итеративно выделяет в еще не просмотренной литературе темы, связанные с темами, выделенными на предыдущей итерации, пытаясь таким образом имитировать поведение исследователя. В результате алгоритм строит ориентированный граф взаимосвязи основных тем, затронутых в литературе (Рис. 2). В этом графе ориентированные маршруты, ведущие от вершины, соответствующей исходной проблеме, реализуют цепочку шагов, предпринимаемых исследователем при последовательном поиске и изучении литературы, касающейся вопросов, связанных напрямую или косвенно с введенной проблемой. В качестве результата работы алгоритма пользователю будет предоставляться построен-

ный граф взаимосвязи тем, наличие которого в готовом виде может помочь ускорить подбор необходимой для исследования литературы.

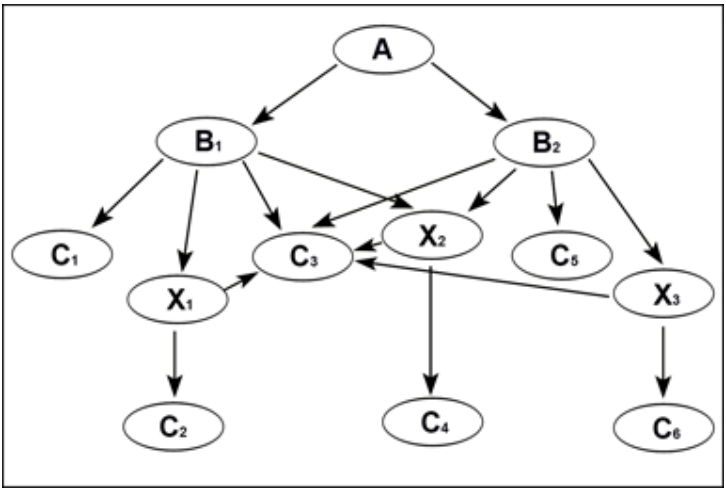


Рис. 1 Граф взаимосвязи тем, построенный в процессе поиска решения проблемы A

В основе описанного метода лежит подход, предложенный Р. Костоффом в 2003 году [1]. Однако, в силу ряда особенностей последнего, таких как предполагаемый полуавтоматический режим работы и ограничение на глубину строящегося графа тем, было принято решение произвести модификацию данного подхода.

Алгоритм построения графа взаимосвязи тем

Определения:

Коллекция документов D – набор неструктурированных текстов произвольной тематики. Предлагаемый в данной работе подход рассчитан на работу с локальной конечной коллекцией документов научного или технического плана на английском языке.

Ключевая фраза – слово или словосочетание w, извлеченное из текста $d \in D$ и имеющее высокую содержательную значимость для текста. В наилучшем случае является термином, отражающим одну из тем, содержащихся в тексте.

Тема T **–** набор семантически тесно связанных между собой ключевых фраз, $T = \{w_1, \ldots, w_N\}$. В наилучшем случае все ключевые фразы, образующие тему T, относятся к одной и той же достаточно узкой тематике, и поэтому определенная таким образом тема должна быть способна дать исследователю четкое представление о тематическом содержании документа.

Проблема P – с точки зрения пользователя представляет собой текстовую формулировку задачи, связанные тематики для которой необходимо найти. Задается в виде словосочетания или набора словосочетаний на естественном языке. С точки зрения алгоритма проблема есть тема, составленная из нескольких ключевых фраз.

Граф взаимосвязи тем – ориентированный помеченный граф $G = (V, E)$, где V – множество вершин графа, и каждая вершина $v \in V$ помечена множеством ключевых фраз, образующих некоторую тему, построенную алгоритмом, а $E \subset V \times V$ – множество ребер, такое, что $\forall v_1, v_2 \in V : e = (v_1, v_2) \in E \leftrightarrow$ тема, которой помечена вершина v_2, была получена алгоритмом при поиске литературы, непосредственно касающейся темы v_1.

Уровень темы T в графе G – номер итерации алгоритма построения графа тем, на котором была сгенерирована тема *T*.

Множество просмотренных документов $Prohib(D)$ – множество документов коллекции, на которых в ходе работы алгоритма уже производилось выделение тем. Документы из этого множества не принимают участия в дальнейшем процессе построении множества тем (то есть документы «изымаются» из коллекции после первого просмотра). Данная мера позволяет реализовать поддержку идеи связывания Свенсона: система стремится выделить связи между темами, описанными в разрозненной, непересекающейся литературе. Для таких тем вероятность иметь неизвестные исследователю взаимные связи выше.

Множество запрещенных ключевых фраз при построении тем уровня k $Prohib(k)$ – набор ключевых фраз, которые не должны присутствовать в документах, непосредственно связанных с некоторой темой *T* уровня *k*–1. Во множество запрещенных ключевых фраз при построении тем уровня *k* попадают все ключевые фразы, составляющие темы уровней 0, …, $k - 2$. Таким образом, система ограничивает возможность выделения тех документов, в которых упоминаются темы, по которым уже производился поиск. Последнее требуется методологией проведения исследований на основе литературных источников: алгоритм имеет целью выделить те

связи между проблемой и темами, которые ранее еще не были построены (если и не в исследовательской практике, то, по крайней мере, самим алгоритмом). Поэтому темы, выделенные в процессе построения графа взаимосвязи тем на более ранних итерациях, считаются известными и далее не представляют интереса. Кроме того, посредством применения этой меры производится попытка избежать построения чрезмерно разветвленного и зашумленного итогового графа тем.

Дополнительно каждой теме сопоставляется набор документов, из которых были выделены составляющие ее ключевые фразы, что позволяет пользователю осуществлять навигацию по коллекции документов при изучении построенного графа.

Предлагаемый алгоритм:

Вход: формулировка проблемы в виде набора фраз на естественном языке и локальная коллекция документов научного или технического плана.

Выход: граф взаимосвязи тем.

Алгоритм:

Шаг 0. Формулировка проблемы в терминах алгоритма.

Набору фраз, введенному пользователем, сопоставляется тема T_{01} уровня 0.

Шаг *k*. Построение множества тем $\{T_{k,1},\ldots,T_{k,m_k}\}$ уровня *k* по множеству тем $\{T_{k-1,1},\ldots,T_{k-1,m_{k-1}}\}$ уровня *k*–1.

- Для каждой темы $T_{k-1,i}$ уровня *k*–1 выполнить:
  - Поиск литературы $D_{k,i}=\left\{d_{k,i,1},\ldots,d_{k,i,j_i}\right\}\subset D\setminus Prohib(D)$, непосредственно связанной с темой $T_{k-1,i}$ (процесс поиска подробно описан далее). Если $D_{k,i}=\varnothing$, завершаем данную итерацию и переходим к теме $T_{k-1,i+1}$.
  - Выделение множества тем $[T]_{k,i}=\{T_{k,i_1},\ldots,T_{k,m_{ki}}\}$, затронутых в найденной литературе $D_{k,i}$ (метод выделения тем подробно описан далее).
- Поместить множество документов, просмотренных на данной итерации, во множество всех просмотренных документов: $Prohib\left(D\right)=Prohib(D)\cup\cup_i D_{k,i}$.

- Поместить множество ключевых фраз, составляющих темы уровня k–1, во множество запрещенных ключевых фраз: $Prohib(k+1) = Prohib(k) \cup \cup_i T_{k-1,i}$.
- Произвести перекомпоновку тем уровня k для получения множества тем $\{T_{k,1},\ldots,T_{k,m_k}\}$:
  - Если $\{[T]_{k,1},\ldots,[T]_{k,m_{k-1}}\} = \varnothing$, итеративный процесс выделения тем завершается на шаге $k = N$. Иначе:
  - Для каждой темы $T_{k,n} \in \cup_i [T]_{k,i}$ поочередно пытаемся объединить ее с другими темами $T_{k,m} \in \cup_i [T]_{k,i}$ (критерий объединения приведен далее). Таким образом, устраняются дубликаты тем, которые могли быть построены для некоторых $T_{k-1,n}$ и $T_{k-1,m}$, $n \neq m$ (внутри же любой $[T]_{k,i}$ темы заведомо не пересекаются по построению). Получаем множество тем $\{\tilde{T}_{k,1},\ldots,\tilde{T}_{k,m_k}\}$.
  - Для всех пар различных тем $\tilde{T}_{k,n},\tilde{T}_{k,m} \in \{\tilde{T}_{k,1},\ldots,\tilde{T}_{k,m_k}\}$ устраняем их пересечение (метод описан далее). Получаем множество тем $\{T_{k,1},\ldots,T_{k,m_k}\}$.

Шаг N+1. Перекомпоновка графа тем.

Для каждой пары соседних уровней k-1 и k графа тем, для каждой темы $T_{k-1,i} \in \{T_{k-1,1},\ldots,T_{k-1,m_{k-1}}\}$ и $T_{k,j} \in \{T_{k,1},\ldots,T_{k,m_k}\}$ определяется необходимость объединения этих тем, и производится объединение в случае такой необходимости (критерий объединения тем приведен далее).

На этом построение графа тем завершается.

Приведенный алгоритм конечен в силу того, что

1. Коллекция текстовых документов, на основе которой строится граф, конечна.
2. Каждый раз после завершения процесса выделения тем на шаге k вся найденная и обработанная на k-й итерации литература помещается в «запрещенное» множество, по которому поиск на дальнейших итерациях не производится. Таким образом, рано или поздно коллекция документов будет исчерпана.
3. Алгоритм чаще всего заканчивает работу до момента исчерпания коллекции документов в силу пополнения множества запрещенных

ключевых фраз и особенности формирования запроса на поиск связанной литературы (см. следующий раздел).

Поиск литературы по заданной теме

Поиск литературы по заданной теме производится стандартными поисковыми средствами. В частности в данной работе для поиска документов используется поисковый сервер Apache Solr, реализованный на базе библиотеки полнотекстового поиска Apache Lucene [11].

Поиск производится следующим образом. Имея некоторую тему на *k*-й итерации алгоритма, для которой необходимо найти связанную литературу, система генерирует запрос к поисковой системе по принципу:

1. В запрос включаются все ключевые слова, образующие тему.
2. К запросу добавляются все термины, образующие темы, построенные на итерациях 0 – *k*–2, с запрещающим поисковым оператором. Иначе говоря, эти фразы либо совсем не должны содержаться в найденных документах, либо ранг таких документов в общем списке понижается.

Например, в случае системы Solr, которая поддерживает стандартный синтаксис поисковых запросов для ИПС общего назначения (таких как Google), запрос Q, сгенерированный по теме $B = \{b_1, \ldots, b_N\}$, при условии, что множество ключевых фраз, образующих темы на итерациях 0 – *k*–2, равно $D = \{d_1, \ldots, d_M\}$, будет иметь вид:

$$Q = "b_1\ b_2 \ldots b_N\ \ -d_1 -d_2 \ldots -d_M\ ".$$

Из множества документов, полученных от поисковой системы, выбираются только те документы, оценка релевантности запросу которых, вычисленная поисковой системой, превышает некоторый порог. Данный порог релевантности должен регулироваться пользователем и, вообще говоря, может зависеть от объема и состава текстовой коллекции, по которой производится поиск.

Выделение тем для множества документов

Для того чтобы по выделенному набору документов сформировать набор затрагиваемых в них тем, необходимо выполнить следующие шаги:

1. Выделить ключевые фразы из каждого документа.
2. Произвести кластеризацию на полученном наборе ключевых фраз с целью группировки семантически связанных ключевых фраз.

Выделение ключевых фраз. Библиотека Maui

Качество сформированных тем в значительной степени зависит от качества ключевых фраз, выделенных из текста и подаваемых на вход алгоритму кластеризации. В случае если важные термины и ключевые фразы будут упущены, темы, построенные по множеству выделенных фраз, будут иметь неполную структуру, либо некоторые из них вовсе могут быть упущены. Если будет выделено слишком много общеупотребительных фраз, возникнет опасность генерации искусственных кластеров, которые будут сформированы за счет перекрывания множеств общеупотребительных фраз из разных текстов. Кроме того, в этом случае могут быть построены ошибочные связи между полученными темами за счет таких «общеупотребительных» связок. Такое явление может послужить причиной необоснованного разрастания конечного графа тем по количеству вершин и ребер и снижению «содержательности» тем, соответствующих вершинам.

Поэтому для выделения ключевых фраз из текстов было решено использовать Maui – библиотеку для автоматического индексирования научной литературы. В основе работы Maui лежит использование статистических методов и алгоритмов машинного обучения для выделения наиболее значимых ключевых фраз из текстовых документов. Maui обладает высокими показателями качества извлечения терминов из документов, сравнимыми с показателями качества индексирования текстов людьми-индексаторами. Подробно с ее устройством и принципом работы можно познакомиться в работах [3] и [4].

Кластеризация

Для формирования тем, иными словами, для группировки тесно связанных ключевых слов, используется метод кластеризации [7]. В общем случае, кластеризация – процесс разбиения заданного множества точек на группы на основе атрибутов этих точек.

Существует несколько классов алгоритмов кластеризации, для которых понятие кластера варьируется и, следовательно, кластеры, построенные разными алгоритмами, значительно отличаются по свойствам. Автором работы был произведен обзор основных классов кластеризационных методов, с результатами которого можно ознакомиться в [5].

Учитывая полученные сведения об особенностях наиболее распространённых алгоритмов кластеризации, для решения поставленной задачи было решено применять центроидный метод кластеризации k-means++ в сочетании с заданием метрик для определения семантических расстояний между ключевыми фразами и заданием требуемого для формирования тем числа кластеров равным $\sqrt{n/2}$, где n – число кластеризуемых точек [8, 9]. При этом задаваемое число кластеров выражает интуитивное понятие об оптимальном выборе числа кластеров, который представляет собой баланс между максимальной компрессией данных путем помещения их в один кластер и максимальной точностью разбиения на кластеры, когда каждой точке множества соответствует свой кластер.

Семантическая дистанция между фразами

Для задания расстояний между точками при кластеризации предлагается использовать метрики на основе меры семантической дистанции между ключевыми фразами.

Существует два основных подхода к измерению семантической дистанции между фразами:

1. Основанный на использовании сторонних источников знаний, например, специализированных словарей, тезаурусов, онтологий.
1. Такие методы позволяют напрямую вычислить показатель близости двух слов или фраз и основаны на предположении, что используемый тезаурус содержит всю необходимую для расчетов информацию. Использование таких мер затруднено тем, что:
   a. не для всех языков построены тезаурусы;
   b. в случае, если тезаурус имеется, он может быть неполон.
2. С примерами тезаурусных мер семантической близости можно ознакомиться в [6].
3. Статистические методы.
4. Многими исследователями было показано, что семантику слова можно определить, пусть даже приближенно, через типичный контекст его употребления. Поэтому, а также в силу наличия богатого тренировоч-

ного материала, для реализации предлагаемого алгоритма было принято решение сосредоточиться на применении статистических методов.

5. Статистические методы опираются на на подсчет характеристик, основанных на совместной встречаемости слов. При этом все слова рассматриваются как точки в *N*-мерном пространстве, и задача определения семантической дистанции сводится к двум основным шагам – задание координат точек в пространстве и вычисление дистанции между точками. Последнее требует подбора подходящей метрики.

В целях уменьшения размерности решаемой методом кластеризации задачи *N*-мерное пространство, в котором производится представление и кластеризация точек, строится по набору T выделенных на данном шаге ключевых фраз следующим образом:

1. Набор T рассматривается в качестве фиксированного на данном шаге словаря V_C, $|V_C| = N$.
2. Все ключевые фразы $v \in V_C$ нумеруются.
3. Теперь каждое слово или фраза $w \in V$ (бесконечного словаря) может быть представлено в виде *N*-мерного вектора величин совместной встречаемости $\overrightarrow{f_w} = (f_1, \ldots, f_N)$, где f_i указывает, насколько часто слово w встречается совместно со словом v_i.

Таким образом, каждая ключевая фраза представляется в виде точки в *N*-мерном пространстве.

В качестве координат точек f_i необходимо брать величины, выражающие степень семантической близости фраз с фразами из V_C и обладающие свойством: величина тем больше, чем больше семантическая близость двух рассматриваемых слов или фраз. Для этого автором работы используется величина Pointwise Mutual Information (PMI). Для произвольных фраз v и w данная величина имеет вид:

$$PMI(v, w) = \frac{f_{v,w}}{f_v f_w},$$

где $f_{v,w}$ – частота совместной встречаемости фраз v и w в некотором корпусе текстов, f_v – частота фразы v в этом корпусе. В качестве корпуса текстов при этом берется подаваемая на вход алгоритму коллекция документов *D*. Слова v_i и w считаются встретившимися совместно, если v_i в некотором тексте встретилось на расстоянии, не превышающем *N* слов от

слова w Величина N должна задаваться пользователем. Для экспериментов N было взято равным 20.

Следует отметить, что именно PMI была выбрана для реализации предлагаемого метода в силу ее большей адекватности в качестве меры семантической близости по сравнению с распространенными бинарной и абсолютной частотами совместной встречаемости слов [6].

Метрики для вычисления семантической дистанции между ключевыми фразами

Введение метрики для вычисления расстояний между ключевыми фразами дает возможность производить кластеризацию слов на основе информации об их семантической близости.

Для измерения дистанции между двумя векторами, соответствующими ключевым фразам можно использовать различные известные метрики. Для изучения влияния выбора метрики на качество кластеров были выбраны 4 известные метрики:

1. Расстояние по Манхэттену (метрика, порождаемая нормой L_1):

$$dist_{L_1}(\vec{x},\vec{y}) = \sum_{i=1}^{N} |x_i - y_i|.$$

2. Евклидова метрика (метрика, порождаемая нормой L_2):

$$dist_{L_2}(\vec{x},\vec{y}) = \left(\sum_{i=1}^{N} (x_i - y_i)^2\right)^{\frac{1}{2}}.$$

3. Метрика на основе величины угла между нормированными векторами ключевых фраз:

$$dist_{cos}(\vec{x},\vec{y}) = \frac{\arccos(sim_{cos}(\vec{x},\vec{y}))}{\pi},$$

где величина sim_{cos} выражает степень семантической близости между ключевыми фразами на основе близости их векторов и равна косинусу угла между векторами $\vec{x}$ и $\vec{y}$:

$$sim_{cos}(\vec{x},\vec{y}) = \frac{\vec{x} \bullet \vec{y}}{\vec{x}_2 \vec{y}_2},$$

где $\vec{x} \bullet \vec{y} = \sum_{i=1}^{N} x_i \bullet y_i$ – скалярное произведение векторов $\vec{x}$ и $\vec{y}$.

4. Дивергенция Дженсена–Шеннона.
   В случае двух векторов дивергенция Дженсена–Шеннона задается следующей формулой:

$$dist_{JS}\left(\vec{x},\vec{y}\right)=\sqrt{\sum_i\left(x_i log_2\frac{2x_i}{x_i+y_i}+y_i log_2\frac{2y_i}{x_i+y_i}\right)}.$$

   Доказательство того, что функция $dist_{JS}\left(\vec{x},\vec{y}\right)$ действительно обладает всеми свойствами метрики, можно найти в [10].

Операции на множестве тем

Первые два этапа алгоритма – это перекомпоновка тем и перекомпоновка графа. Перекомпоновка тем производится с целью устранения дубликатов тем в конечном графе. Такие дубликаты могут возникать среди тем, построенных на одной итерации (во множествах тем, построенных по различным темам из предыдущей итерации), либо же среди тем, построенных на двух последовательных итерациях.

Определим две операции:

1. Объединение тем производится в случае, если мера близости двух тем превышает установленный порог $t, 0 \le t \le 1$.
2. Устранение пересечения тем производится в противном случае. При этом множество ключевых фраз, попавших в пересечение, относится к меньшей по мощности теме.

В качестве меры близости тем берется коэффициент Жаккара: если A и B – два множества, то

$$sim_J\left(A,B\right)=\frac{\left|A\bigcap B\right|}{\left|A\bigcup B\right|},\ 0\le sim_J\left(A,B\right)\le 1.$$

Пороговое значение t может варьироваться и устанавливается вручную. Для экспериментов величина порога объединения тем бралась из интервала $0.6 \le t \le 0.8$.

Область применения алгоритма

В силу ряда особенностей метода, предложенного автором данной работы, а именно, применения статистических методов для формирования тем, описывающих коллекцию документов, а также использования ключевых фраз, извлекаемых из документов, как главный характеризующий их признак, данный метод ориентирован на анализ научной и технической литера-

туры. Кроме того, очевидно, что успех применения данного метода будет зависеть от степени сформированности терминологии, используемой в анализируемой литературе.

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Предложенный подход к автоматизации процесса исследования на основе литературных источников был реализован в прототипе информационной системы для поддержки процесса анализа литературных источников.

Разработанный прототип работает в трех режимах:

1. Режим обучения.
   Этот режим существует для тренировки Maui. При работе в данном режиме системе на вход подается набор текстов, в котором каждому тексту сопоставлен список ключевых слов. На основе поданной коллекции текстов система генерирует модель извлечения ключевых фраз для дальнейшего применения в режиме индексирования. Соответственно, если индексирование документов будет производиться для коллекции документов по фиксированной тематике, то для обучения рекомендуется брать коллекцию документов по этой же либо родственной ей тематике. Кроме того, обучение можно производить с использованием словаря предметной области, либо вообще без словаря.
2. Режим индексирования.
   В этом режиме система получает на вход коллекцию документов в произвольном текстовом формате, поддерживаемом системой. В данный момент система поддерживает такие форматы, как txt, html, xml, doc, pdf и некоторые другие. Также должна быть указана какая-либо из моделей извлечения, построенная на этапе обучения, и может указываться словарь, на основе которого необходимо извлекать терминологию из текстов (без словаря ключевые фразы извлекаются непосредственно из текста, в противном случае тексту приписываются термины из словаря). Для каждого поданного документа производится извлечение из него (или приписывание ему) ключевых фраз, которые сохраняются в базе данных. Таким образом, в процессе поиска извлечение ключевых фраз каждый раз не производится. Это ускоряет работу приложения ипозволяет использовать для извлечения ключевых фраз модель извлечения и, возможно, словарь, подхо-

дящие по тематике к индексируемой коллекции, что улучшает качество индексирования.

3. Режим поиска.
   В этом режиме системе на вход подается запрос от пользователя в виде набора фраз на естественном языке. Система производит построение графа взаимосвязи тем по алгоритму, изложенному в данной работе, на множестве проиндексированных документов (фактически, на множестве ключевых фраз, сохраненных в базе данных). Построенный граф выдается пользователю в текстовом формате.

Архитектура приложения

Реализованный прототип информационной системы написан на языке Java и имеет стандартную трехуровневую архитектуру: пользовательский интерфейс, алгоритмическую компоненту и интерфейс доступа к данным. Схематично архитектура приложения изображена на Рис. 2.

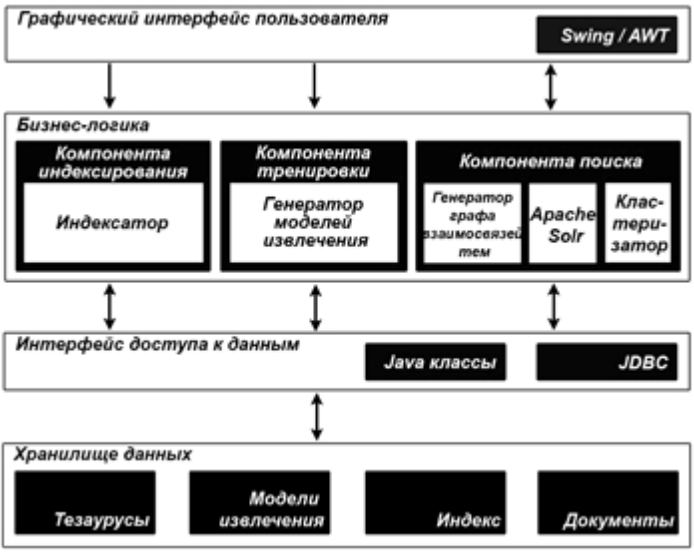


Рис. 2. Архитектура приложения

ТЕСТИРОВАНИЕ

Для тестирования приложения была взята коллекция, состоящая из 50 научных публикаций по тематике обработки сигналов и изображений (материалы конференции ICASSP 2002).

Предварительно приложение было обучено на множестве научных статей, состоящем из 800 документов по тематике обработки сигналов и изображений (материалы научной конференции ICASSP 2011). Обучающая выборка была подготовлена в полуавтоматическом режиме и представляла собой пары: публикация, преобразованная в текстовый формат, и файл, содержащий ключевые слова, которыми данная публикация была проиндексирована человеком-индексатором.

Тестирование производилось по 5 запросам. Варьировались следующие параметры:

- число ключевых фраз, сопоставляемых родительской теме при формировании списка дочерних тем (от него зависели число и размеры кластеров);
- метрика для алгоритма кластеризации;
- пороговое значение коэффициента Жаккара для операций над темами.

Для построенных графов оценивались следующие количественные параметры:

- общее число тем, выделенное по запросу из коллекции документов;
- число документов коллекции, вовлеченных в процесс построения графа тем;
- число итераций, проделанное алгоритмом при построении графа тем («глубина графа тем»).

Из результатов можно заключить, что при фиксированных метрике и пороге объединения тем и увеличении числа ключевых фраз, извлекаемых для каждой темы, глубина графа практически не варьируется, несмотря на разрастание графа в ширину, которое обусловлено прежде всего методом формирования кластеров. С другой стороны, разрастание графа связано также с увеличением общего числа вовлеченных в процесс поиска ключевых фраз, дающим большую возможность вариации тем, что подтверждается увеличением числа привлеченных к построению графа документов.

По результатам тестирования было также отмечено, что при фиксированном нижнем пороге числа ключевых фраз для получения более разветвленной структуры графа тем в случае метрики Дженсена-Шеннона необходимо принимать порог объединения тем равным 0.6, для метрики Манхет-

тен – 0.7, а для угловой метрики – 0.8. В случае метрики Евклида выбор порога неоднозначен, однако наибольшая вовлеченность текстовой коллекции была получена при величине порога объединения тем, равной 0.8.

Экспертом также была произведена качественная оценка информационной связности построенных графов тем. В целом было отмечено хорошее качество работы системы, а именно, информационная связность и содержательность построенных графов взаимосвязи тем с точки зрения потенциального исследователя. Была отмечена невысокая степень промахов системы при формировании тем – число фраз в кластерах, не подходящих к их основной тематике, было крайне незначительным. При этом сформированные темы-кластеры давали эксперту представление о подразумевающейся тематике и, таким образом, были содержательными с точки зрения человека-исследователя.

Наиболее эффективным сочетанием параметров по результатам качественной оценки работы системы являются:

- нижний порог числа ключевых фраз, равный 50;
- мера Дженсена-Шеннона;
- порог слияния тем по мере Жаккара, равный 0.6 (это значение дает наиболее разветвленный граф для метрики Дженсена-Шеннона, однако не влияет на информационную связность тем).

Также было проведено тестирование системы на большой выборке научных статей (1012 документов по тематике обработки сигналов и изображений – материалы конференции ICASSP 2002). Тестирование производилось с обозначенными выше оптимальными параметрами: нижний порог числа ключевых фраз, равный 50, порог слияния тем, равный 0.6, метрика Дженсена–Шеннона. Тестирование производилось по запросам «Compressing multi-component digital maps» и «Digital image watermarking», для которых при тестировании на малой коллекции были получены наихудшие качественные результаты в силу отсутствия в малой коллекции достаточной доли документов, связанных непосредственно или косвенно с тематикой запросов. Были получены следующие результаты:

1. Глубина построенных графов взаимосвязи тем была равна 8 в обоих случаях.
2. Оба графа взаимосвязи тем содержали порядка 2000 вершин.
3. Число вовлеченных файлов было равно 772 и 737 (из 1012), соответственно.

Заключение эксперта о качестве построенных графов взаимосвязи тем было следующим:

1. Для запроса «Compressing multi-component digital maps» были получены очень качественные результаты: очень удачная классификация ключевых фраз и документов по темам, адекватно отражающая структуру предметной области.
2. Результаты по второму запросу «Digital image watermarking» оказались менее очевидными. Полученная структура тем оказалась сложной и труднопонимаемой. Результаты работы системы на данном запросе требуют дополнительного пристального изучения экспертом, но также являются интересными.

ЗАКЛЮЧЕНИЕ

Итогом работы стала реализация прототипа информационной системы, в перспективе способной играть роль ассистента исследователя при поиске потенциальных решений исследовательских задач. В частности, разрабатываемая система имеет потенциал для выявления ранее не замеченных косвенных взаимосвязей между подходами, методами и технологиями, информация о которых была опубликована; система также способна выявлять и группировать литературу, которая может быть интересна исследователю при решении его задач.

Результаты работы, в частности, разработанный прототип информационной системы, могут служить основой для проведения экспериментальных исследований, касающихся проблемы автоматизации процесса исследования на основе литературных источников и дальнейшего развития методов автоматизации этого процесса.

Однако предложенный метод требует дальнейшей доработки. А именно, в дальнейшем могут быть исследованы вопросы:

- эффективности различных методов кластеризации в контексте решаемой задачи и методов улучшения качественных характеристик генерируемых тем;
- возможности и целесообразности комбинации статистических характеристик ключевых фраз с их характеристиками, вычисленными на основе тезаурусов и Википедии, при формировании тем;
- эффективности и целесообразности операций над темами в зависимости от их свойств (плотность тем, степень пересечения тем и т.д.);
- возможности и целесообразности разработки подходов для сокращения мощности генерируемого графа тем.

Таким образом, в будущем предполагается совершенствование предложенного метода и разработанного программного средства.

СПИСОК ЛИТЕРАТУРЫ

1. Kostoff R., Boylan R., Simons G. Disruptive technology roadmaps / Technol. Forecast. Soc. Change. – 71. – 2004. – P. 141–159 – Mode of access: http://www.cuaed.unam.mx/puel_cursos/cursos/d_gcfe_m_tres/modulo/modulo_3/m3-3.pdf
2. Ganiz M., Pottenger W., Janneck C. Recent Advances In Literature Based Discovery / Lehih University Technical Report LU-CSE-05-027. – 2005. – Mode of access: http://www.cse.lehigh.edu/~billp/pubs/JASISTLBD.pdf
3. Medelyan O., Frank E., Witten I. Human-competitive tagging using automatic keyphrase extraction / Proc. of the Internat. Conference of Empirical Methods in Natural Language Processing EMNLP-2009. – Singapore. – 2009. – 10 P. – Mode of access: http://www.cs.waikato.ac.nz/~olena/publications/emnlp2009_maui.pdf
4. Medelyan O. Human-competitive automatic topic indexing. PhD Thesis. – University of Waikato, New Zealand. – 2009. – 241 P. – Mode of access: http://www.cs.waikato.ac.nz/~olena/publications/olena_medelyan_phd_thesis_July2009.pdf
5. Шманина Т. В. Информационная система для поддержки процесса проведения исследований на основе литературных источников. Магистерская диссертация. – Новосибирский Государственный Университет, Россия. – 2012. – 64 С.
6. J. Hockenmaier Introduction to Natural Language Processing. Lectures at University of Illinois at Urbana-Champaign [Electronic resource]. – 2008. – Mode of access: http://www.cs.uiuc.edu/class/fa08/cs498jh/
7. Cluster analysis [Electronic resource]. – Mode of access: http://en.wikipedia.org/wiki/Cluster_analysis
8. D. Arthur, S. Vassilvitskii k-means++: the advantages of careful seeding / Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms SODA'07. – 2007. – pp. 1027-1035.
9. Determining the number of clusters in a data set [Electronic resource]. – Mode of access: http://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set
10. D. Endres, J. Schindelin A New Metric for Probability Distributions / IEEE Transactions on Information Theory. – 49 (7). – 2003.
11. Apache Solr Documentation [Electronic resource]. – Mode of access: http://lucene.apache.org/solr/

Т.А. Золотухин

ВИЗУАЛИЗАЦИЯ ГРАФОВ ПРИ ПОМОЩИ ПРОГРАММНОГО СРЕДСТВА VISUAL GRAPH[1]

ВВЕДЕНИЕ

Визуализация информации играет большую роль в жизни человека. Считается, что около 90% всей получаемой информации человек получает через зрение. Человечество за тысячи лет преодолело путь от простейших способов визуализации в виде наскальных рисунков до карт, схем и диаграмм. В настоящее время визуализация – неотъемлемый элемент обработки сложной информации о строении объектов.

Многие структуры данных, представляющие практический интерес в математике и информатике, могут быть представлены в виде графов. Одним из основных классов является класс иерархических и/или атрибутированные графов[1].

Преимущества графов во многих случаях становятся ощутимыми только при наличии хороших инструментальных средств для их визуализации и обработки. Поэтому в настоящее время в мире происходит значительный рост интереса к методам и средствам визуализации графов, о чем свидетельствует рост числа публикаций, содержащих описания новых алгоритмов и способов визуализации графов, а также их реализации в программных средствах.

В данной статье будет рассмотрено одно из таких программных средств, которое предназначено для визуализации и поиска информации в иерархических атрибутированных графах (далее – графовая модель). Данное программное средство называется Visual Graph и разрабатывается при содействии лаборатории НГУ-Интел в рамках проекта по созданию системы для визуализации графовых моделей[2].

Также стоит отметить, что Visual Graph использует лицензию BSD, что позволяет применять возможности данного проекта практически в любых целях.

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

ОБЛАСТЬ ПРИМЕНЕНИЯ

В ходе развития компиляторов было создано несколько структур, которые представляются в виде графов: синтаксические деревья, графы потоков управления и графы вызовов. Каждый из этих типов графов имеет свое практическое применение. Так, синтаксические деревья используются при построении внутреннего представления программы в компиляторах или интерпретаторах. Графы потока управления – для оптимизации в компиляторах и утилитах статического анализа кода, а графы вызовов – при отладке программ. Все эти графы объединяет то, что их можно рассматривать как иерархические и атрибутированные графы.

Представление этих графов в разных компиляторах может быть разным, причем нет гарантии, что оно не будет отличаться в разных версиях одного и того же компилятора.

Для решения данной проблемы разработчики выбрали способ, в котором либо сам компилятор, либо вспомогательная программа переводит граф из внутреннего представления в файл одного из поддерживаемых программным средством Visual Graph форматов.

После описанной выше процедуры сохранения программное средство Visual Graph сможет прочитать эту графовую модель из файла, визуализировать ее и предоставить пользователю средства навигации по ней.

СУЩЕСТВУЮЩИЕ ПРОБЛЕМЫ И ИХ РЕШЕНИЯ

При разработке программного средства Visual Graph разработчики ь столкнуться с массой проблем. В данном разделе будут изложены основные из них, а так же то, как они были решены в рамках программного средства Visual Graph.

Начать стоит с проблемы хранения графов вне программного средства Visual Graph, т.е. представление графов в файловой системе, в текстовом виде. Для представления графов в текстовом виде существует множество языков описания обычных графов, но для иерархических атрибутированных графов количество таких языков невелико. Одним из таких языков является GraphML[3].

GraphML – это язык описания графов, основанный на XML. Этот формат позволяет описывать направленные, ненаправленные и смешанные графы, гиперграфы и иерархические графы, специфичные для приложений

атрибуты. Соответственно, язык GraphML полностью поддерживает иерархические атрибутированные графы.

Другой проблемой является хранение графов внутри программного средства Visual Graph. Размер входного графа может достигать сотен мегабайт. Следовательно, хранение такого количество информации в оперативной памяти компьютера может привести к тому, что она быстро закончится. Одно из решений этой проблемы – это кэширование данных на жесткий диск, но в связи с тем, что перед проектом стояла задача навигации, которая подразумевает под собой также выборку всевозможной информации из графовых моделей, было принято решение об использовании реляционной базы данных.

В качестве реляционной базы данных была выбрана встраиваемая база данных SQLite[4]. В отличие от большинства популярных реляционных баз данных, для SQLite не требуется установка сервера, а вся клиент-серверная архитектура сводится к работе с файлами.

Следующей проблемой, с которой пришлось столкнуться при разработке, стала проблема расширяемости системы. Данная проблема возникает из-за того, что такие части программного средства, как навигация, визуализация и анализ графов, представляются набором средств, который может расширяться как разработчиками самого программного средства Visual Graph, так и сторонними разработчиками.

Для построения расширяемой системы было принято решение о добавлении поддержки плагинов. На данный момент вся функциональность программного средства Visual Graph является набором плагинов, которые могут взаимодействовать друг с другом. Взаимодействие происходит за счет системы уведомлений и запросов. Таким образом, каждый плагин может отправить системе запрос или уведомление, и та, в свою очередь, разошлет его по всем плагинам. Плагин, исполняющий чей-либо запрос, также может отослать заказчику (плагину, породившему этот запрос) уведомление о состоянии его запроса. Разработчику плагина необходимо только прописать реакцию своего плагина на те, или иные запросы и уведомления.

Следующей проблемой является проблема навигации по графовым моделям. Как известно, нет какого-то универсального набора средств навигации, который бы удовлетворял любым потребностям любого пользователя для решения любой его задачи. Поэтому было принято решение ориентироваться на область применения, описанную выше, и разработать для нее следующие инструменты:

1. Рабочий стол и миникарта (рис. 1). Рабочий стол состоит из набора вкладок, которые пользователь открывает для визуализации выбран-

ной части графовой модели. Каждая вкладка состоит из рабочей области и нерабочей вкладки, соответственно. Рабочая область – это область, которую пользователь видит непосредственно в данный момент, и в которой визуализируются вершины и ребра графа, а также атрибуты и их значения, связанные с этими вершинами и ребрами. Нерабочая область – область, которую пользователь не видит, но может туда попасть, используя скроллинг. Миникарта позволяет увидеть весь граф целиком в текущей вкладке, а также текущую рабочую область.

2. Навигатор, визуализирующий графовые модели, с которыми в данный момент работает пользователь в виде дерева (рис. 1). В данном дереве отображены только вершины, т.к. отображение ребер не будет нести никакой смысловой информации, но будет очень сильно засорять рисунок. Для быстрого поиска по дереву реализована строка быстрого поиска, которая позволяет пользователю без труда найти интересующую его вершину по имени. Так же из навигатора есть доступ к визуализации интересующих пользователя вершин графовой модели с помощью рабочего стола. Т.е. пользователь может выделить интересующую его вершину или группу вершин и открыть их в новой вкладке. При этом будут достроены все ребра между выбранными вершинами.
3. Атрибутная панель, которая позволяет управлять визуализацией атрибутов для выбранных вершин и ребер в текущей вкладке. Для этого пользователю необходимо выбрать интересующие его вершины и ребра в текущей вкладке, после чего отметить в атрибутной панели галочками те атрибуты, которые он хочет визуализировать (рис.10).
4. Фильтр – инструмент, который использует тот факт, что пользователь работает с атрибутированными графами (рис.2). Данный инструмент осуществляет поиск вершин и ребер по заданным условиям в текущей вкладке. Построение условий осуществляется за счет атрибутов и их значений. На рис.2 показан механизм задания выражения, состоящего из логических связок, скобок и условий вида <имя атрибута = его значение (возможно подстроки)>. После того как пользователь закончит формирование выражения, оно исполнится на элементах (вершины и ребра) графовой модели, находящихся в текущей вкладке.

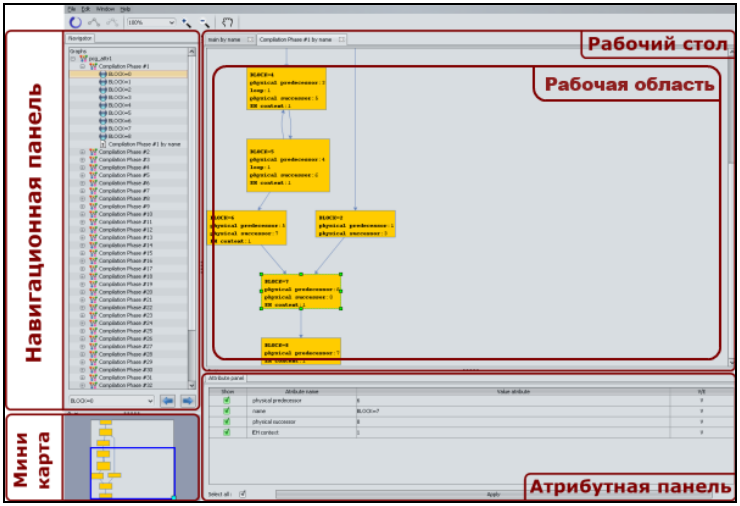


Рис. 1. Пользовательский интерфейс программы Visual Graph

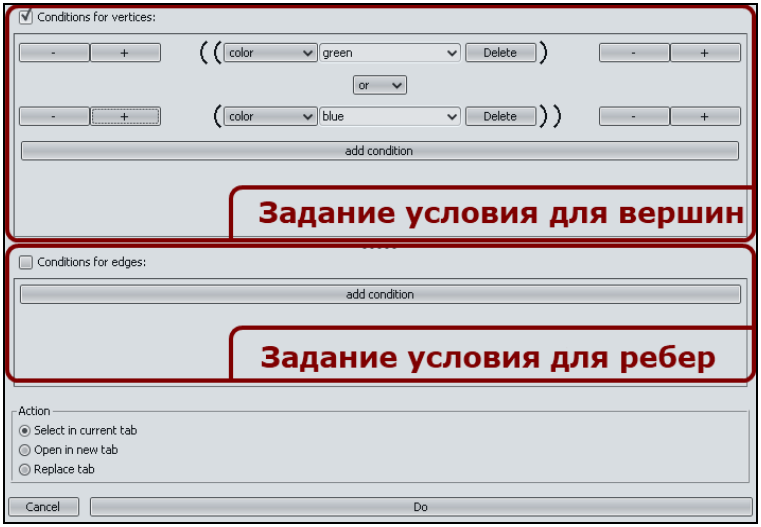


Рис. 2. Фильтр

В результате пользователь получит набор вершин и ребер, которые будут удовлетворять заданным им условиям (рис. 3).

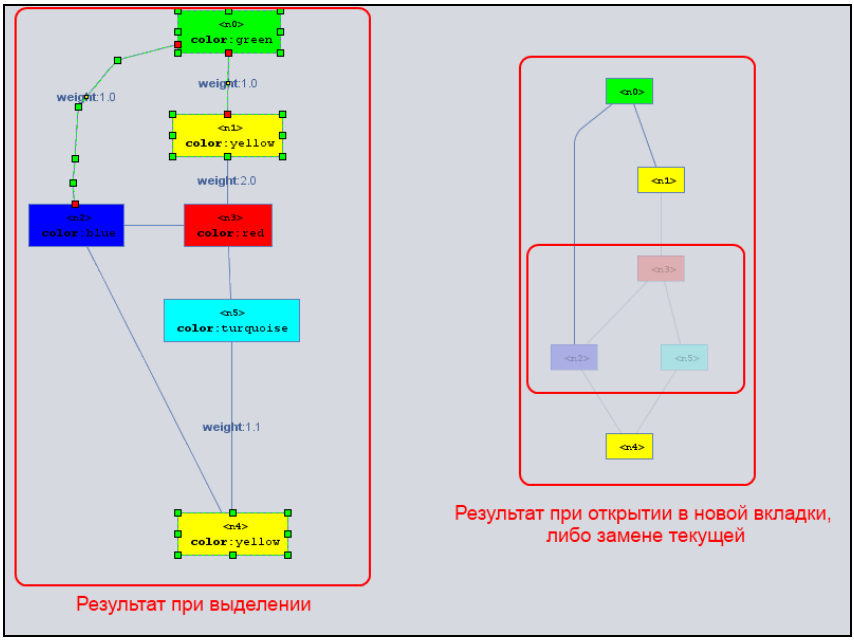


Рис. 3. Результат работы фильтра

5. Поисковая панель – инструмент, который, как и фильтр, использует тот факт, что пользователь работает с атрибутированными графами. В отличие от фильтра, данный инструмент позволяет задавать условия только для вершин и осуществляет поиск не только для графа в текущей вкладке, но и для всех его внутренних графов и их внутренних графов и т.д. по иерархии вниз. Такой поиск даёт т.н. результирующее дерево (0), в котором красными крестиками отмечены вершины, не удовлетворяющие заданному условию.

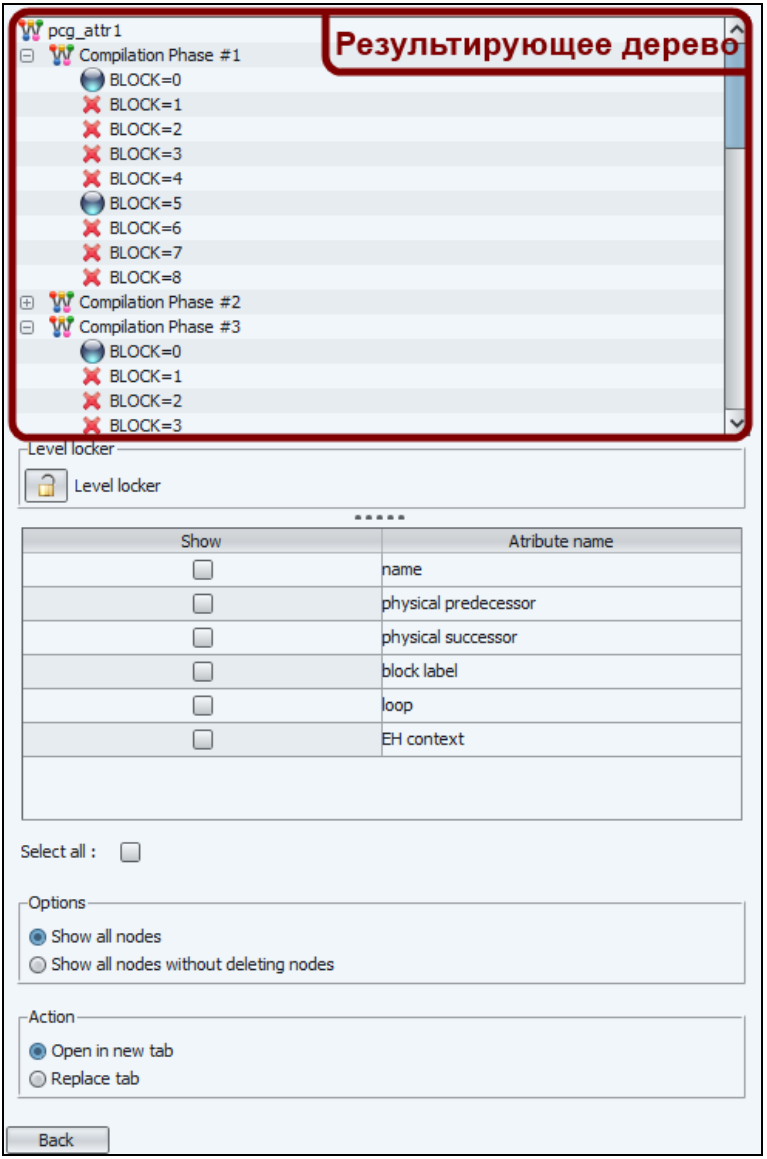


Рис. 4. Результат работы поисковой панели

Как и в случае с навигатором, пользователь может выделить интересующую его вершину или группу вершин и открыть их в новой вкладке. При этом будут достроены все ребра между выбранными вершинами.

6. Блокнот – инструмент, позволяющий загружать файлы с дополнительной информацией и связывать их с графовыми моделями. После этого пользователь может перейти из элемента графовой модели к дополнительной текстовой информации. Например, это можно использовать для связи графовой модели с исходным кодом.

К средствам навигации также можно отнести и средства структурного анализа, т.е. различные алгоритмы, применяемые к графам и помогающие пользователю получить различного рода информацию: подсветку кратчайшего пути в графе, подсветку циклов и т.п.

В программе Visual Graph к средствам структурного анализа можно отнести подсветку кратчайшего пути, подсветку циклов, а также сравнение графов с подсветкой различий в них.

Отдельно хотелось бы остановиться на инструменте, позволяющем проводить сравнительный анализ двух графов и показывать наибольший общий подграф, а в случае, если графы изоморфны – сообщать об этом. Данного инструмента нет ни в одной из трех рассмотренных ниже программ-аналогов, и, в отличие от инструментов поиска пути и циклов, он наиболее сложен в реализации.

Как известно, алгоритм поиска изоморфизма двух графов принадлежит классу NP и в основном решается полным перебором с применением эвристик для уменьшения числа вариантов для перебора. Алгоритм, разрабатываемый для данного инструмента, идет по тому же пути и работает следующим образом:

1. На вход принимается два атрибутированных (возможно, и без атрибутов) графа.
2. Строится двудольный граф, вершинами которого являются вершины первого и второго графа (первая доля и вторая доля, соответственно). Далее от каждой вершины одной доли проводится ребро к вершине другой доли с весом, равным проценту совпадения этих двух вершин (от нуля до единицы, соответственно). При этом нуль означает кардинальное несовпадение вершин, а единица, наоборот, полное их совпадение
3. К полученному двудольному графу применяется венгерский алгоритм решения задачи о назначениях для получения максимального паросочетания максимального веса [8, 9]. Очевидно, что такое па-

росочетание может быть не в единственном числе и далее остается перебрать все эти варианты.

Несмотря на то, что в конце алгоритм сводится к перебору, данная эвристика позволяет отсеять большое количество неверных решений, тем самым уменьшив общее время работы алгоритма. Происходит это главным образом за счет того, что графы, с которыми работает пользователь программы Visual Graph, в большинстве своем атрибутированные, и пользователю разрешено выставлять вес того или иного атрибута, который будет использован при подсчете веса ребра в двудольном графе, описанном выше.

Последней проблемой, которая будет рассмотрена в данном разделе, является проблема отображения графов, а именно раскладка графа на плоскости.

Задача раскладки не имеет и не может иметь оптимального решения в связи с тем, что наборы критериев для оценки качества раскладки различны для различных графов. Для некоторых графов одни критерии будут вносить большую значимость в читаемость графа, чем другие. Тем не менее, можно подобрать виды раскладок, их примерные параметры и алгоритмы для конкретных классов графов, которые обычно визуализируются.

Рассмотрим графы, которые обычно генерируются при работе компилятора:

1. Графы управления. Они обычно имеют относительно небольшое количество ребер и структуру, близкую к иерархической. Как правило, поуровневая раскладка хорошо справляется с такими видами графов. В зависимости от объема графа можно использовать эвристики или алгоритмы для минимизации ширины или высоты получаемой раскладки, а также пересечений между ребрами.
2. Синтаксические деревья. Существует множество техник для отображения деревьев. Можно воспользоваться алгоритмами для упорядочивания узлов или алгоритмами радиальной раскладки дерева.

ОСОБЕННОСТИ ПРОГРАММНОГО СРЕДСТВА VISUAL GRAPH И ЕГО АНАЛОГОВ

В настоящее время на рынке представлен достаточно широкий круг программных средств для работы с графами. Наиболее известными являются aiSee[5], yEd[6] и cytoscape[7].

Область применения этих программных средств частично совпадает с областью применения программного средства Visual Graph. Следует пояснить слово «частично». В данном контексте оно означает, что их область применения шире, т.к. многие из них рассчитаны на редактирование существующих графовых моделей и создание новых. Программное средство Visual Graph ориентировано исключительно на просмотр существующих графовых моделей без возможности их редактирования.

Основные задачи, решаемые подобными программными средствами – это отображение графов и навигация по ним. Поэтому в рамках этих двух задач и будут рассмотрены особенности каждого из вышеперечисленных средств.

Вначале стоит сказать несколько слов о каждом из рассматриваемых нами программных средств.

aiSee – программное средство, которое автоматически рисует раскладку графа, описанного на языке GDL. Затем пользователь может интерактивно исследовать, данный граф, отпечатать его и сохранить в различных форматах. Первоначально aiSee был разработан для визуализации структур данных, обрабатываемых компиляторами. На сегодняшний день его используют десятки тысяч людей по всему миру в самых различных областях, таких как бизнес-менеджмент (структурные схемы предприятий, визуализация бизнес-процессов), генеалогия (семейные деревья), разработка программного обеспечения (блок-схемы, графы потока управления, графы вызовов функций, анализ объёма стека) и так далее.

Cytoscape – свободное программное обеспечение с открытым исходным кодом для визуализации и анализа сетей. Основная область применения данной программы – это биоинформатика. Данное программное средство стало очень популярным благодаря тому, что оно легко расширяется, позволяя сторонним разработчикам писать для него различные плагины.

yEd представляет собой программное средство, которое может быть использовано для быстрого создания и редактирования высококачественных диаграмм. Создание диаграмм происходит вручную или с помощью импортирования из внешних данных. После этого к полученной диаграмме можно применить богатый набор алгоритмов для проведения анализа с последующим получением необходимой информации.

Каждое из рассматриваемых программных средств так или иначе отображает графы и имеет свои методы для управления отображением этих графов.

В aiSee присутствует несколько раскладчиков и множество настроек для них, которые в незначительной степени влияют на результат. Качество раскладки определенных типов графов очень высокое.

В yEd присутствует большое количество раскладчиков и опций для них, которые предоставляют возможность тонкой настройки визуализации каждого графа пользователем.

Cytoscape имеет как собственные алгоритмы раскладки, так и сторонние, среди которых можно отметить присутствие алгоритмов из yFiles, которые используются в yEd. Стоит также отметить, что результаты, получаемые в yEd, намного лучше, чем аналогичные результаты в Cytoscape с настройками раскладчиков по умолчанию.

Программное средство Visual Graph предоставляет несколько раскладчиков, главным из которых является модифицированный иерархический раскладчик, максимально адаптированный для работы с графами, используемыми в компиляторах. Но при желании раскладчик может быть изменен и/или настроен под другие типы задач.

Закончив с задачей отображения графов, мы можем приступить к рассмотрению особенностей средств навигации.

Выделим основные средства навигации, которые предоставляет программа aiSee для решения данной задачи:

1. Рабочий стол, который визуализирует графовую модель целиком, выдавая пользователю статичную картинку, которую нельзя изменять, например, передвигая элементы.
2. Поисковый инструмент, позволяющий пользователю искать элементы графовой модели с помощью их имен. Он поддерживает задание регулярных выражений, выбор категорий элементов для поиска и сохранение предыдущих поисковых запросов.

Выделим основные средства навигации, которые предоставляет программа yEd для решения данной задачи:

1. Рабочий стол схожий с рабочим столом, который предоставляет aiSee. Но, в отличие от aiSee:
   a. есть возможность работать с несколькими графовыми моделями одновременно в одном экземпляре программы (создается вкладка для каждой графовой модели),
   b. есть возможность редактирования элементов графовой модели, начиная от смены их положения и размеров до задания атрибутов, влияющих на их визуализацию
2. Навигатор, отображающий графовую модель в виде дерева. Стоит отметить, что данный инструмент отображает дерево только для

той графовой модели, с которой в данный момент работает пользователь.

3. Миникарта, показывающая всю графовую модель целиком.

Выделим основные средства навигации, которые предоставляет программа Cytoscape для решения данной задачи:

1. Рабочий стол, наподобие того, что предоставляет yEd. Отличие заключается в том, что программа Cytoscape не умеет работать с иерархическими графами.
2. Миникарта, наподобие миникарты в программном средстве yEd.
3. Атрибутная панель, которая позволяет отображать текущие атрибуты и задавать новые. Данный инструмент выглядит как таблица, по горизонтали которой расположены имена атрибутов, по вертикали – список выделенных вершин, а в ячейках находятся соответствующие значения того или иного атрибута для той или иной вершины.
4. Фильтр, позволяющий осуществлять поиск вершин и ребер по заданным условиям на атрибутах.

Выделим основные средства навигации, которые предоставляет программное средство Visual Graph (подробное описание всех нижеперечисленных инструментов можно найти в разделе “ Существующие проблемы и их решения”):

1. Рабочий стол, визуализирующий части графовой модели. К особенностям данного инструмента можно отнести:
   1) возможность влиять на визуализацию элементов посредством задания определенных атрибутов;
   2) возможность работать с несколькими графовыми моделями одновременно, в одном экземпляре программы;
   3) возможность редактирования элементов графовой модели (изменения положения элементов и их размеров);
   4) возможность просмотра интересующей пользователя области графовой модели в другом окне или вкладке. Это одна из важнейших особенностей данного инструмента. Человеку для понимания чего-то сложного (а графовые модели, безусловно, сложны) нужно сложное разделить на множество маленьких простых частей, которые он сможет легко проанализировать;
   5) возможность визуализации пользовательских атрибутов.
2. Навигатор, наподобие навигатора в программном средстве yEd. Основным отличием является то, что в yEd навигатор отображает графовую модель, находящуюся в текущей вкладке, а не все графо-

вые модели, с которыми работает пользователь, как это сделано в Visual Graph. Несложно понять, что в случае с yEd во главе угла стоит рабочий стол, тогда как в Visual Graph главным элементом является навигатор.

3. Миникарта, наподобие миникарты в программном средстве yEd.
4. Атрибутная панель, отображающая набор атрибутов у выделенных элементов графовой модели, а также позволяющая управлять их визуализацией на рабочем столе.
5. Фильтр и поисковая панель осуществляют поиск элементов по заданному выражению. К особенностям данного инструмента можно отнести:
   1) возможность задания сложных булевских выражений, содержащих скобки и разнотипные знаки между ними;
   2) использование типов атрибутов при задании выражения.
6. Блокнот. Несмотря на то, что данный инструмент не обладает большинством стандартных функций, таких как подсветка синтаксиса или поиск с использованием регулярных выражений, его функциональности вполне хватает для решения тех задач, которые встают перед пользователем программного средства Visual Graph. Функциональность включает:
   1) возможность открывать множество текстовых файлов и связать их с графовыми моделями для последующего перехода из элемента графовой модели к фрагменту в текстовом файле;
   2) подсветку результатов поиска.
7. Средства структурного анализа, такие как поиск циклов и кратчайших путей, а также сравнительный анализ двух графов. Сложно охарактеризовать плюсы и минусы данных инструментов, но можно с уверенностью сказать, обойтись без них практически невозможно: стоит лишь представить, сколько времени уйдет на то, чтобы вручную сравнить два графа со 100 вершинами, не говоря уже о более крупных графах.

ЗАКЛЮЧЕНИЕ

В данной статье были рассмотрены проблемы, возникшие в ходе разработки программного средства Visual Graph, его область применения, а также было произведено исследование схожих программных средств. Результаты исследования показали, что программное средство Visual Graph обла-

дает множеством достоинств и не уступает аналогам в области визуализации графов, используемых в компиляторах.

Также стоит отметить, что использование программного средства Visual Graph не ограничено только визуализацией графов, применяемых в компиляторах. Данное программное средство может быть использовано и в других смежных областях, в которых необходима визуализация графов и навигация по ним.

СПИСОК ЛИТЕРАТУРЫ

1. Касьянов В.Н. Иерархические графы и графовые модели: вопросы визуальной обработки // Проблемы систем информатики и программирования. – Новосибирск, 1999. – С. 7–32.
2. Касьянов В.Н., Евстигнеев В.А. – Графы в программировании: обработка, визуализация и применение. – Санкт-Петербург, 2003. – 1104 с.
3. Спецификация GraphML. URL: http://graphml.graphdrawing.org/specification.html (дата обращения: 15.05.2011)
4. Домашняя страница проекта SQLite. URL: http://www.sqlite.org (дата обращения: 15.05.2011)
5. Проприетарный аналог aiSee. URL: http://www.aisee.com (дата обращения: 15.05.2011)
6. Проприетарный аналог yEd. URL: http://www.yworks.com (дата обращения: 15.05.2011)
7. Cytoscape. URL: http://www.cytoscape.org/ (дата обращения: 15.05.2011)
8. Kuhn H. W. The Hungarian Method for the assignment problem. Naval Research Logistics Quarterly, 2:83—97, 1955. Kuhn's original publication.
9. Munkres J. Algorithms for the Assignment and Transportation Problems, Journal of the Society for Industrial and Applied Mathematics, 5(1):32—38, 1957 March.

Д.С. Гордеев

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

ВВЕДЕНИЕ

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

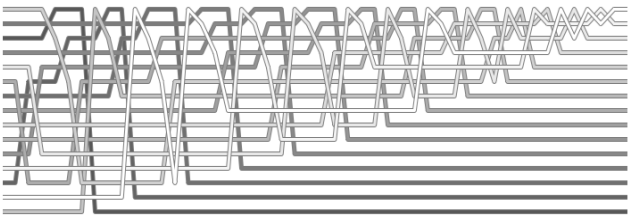


Рис. 1. Визуализация пирамидальной сортировки массива с помощью статичного изображения

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

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

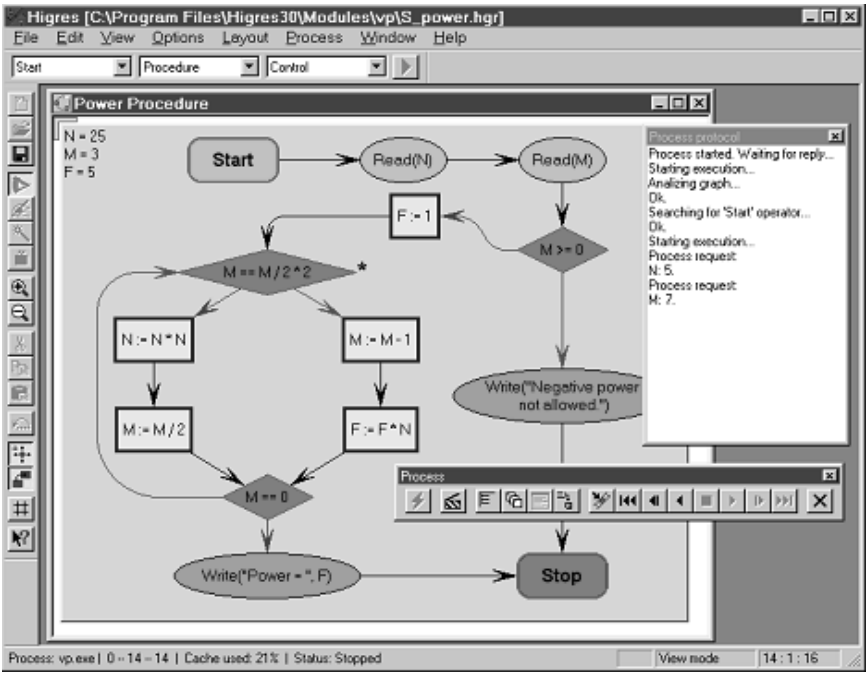


Рис. 2. Выполнение алгоритма, реализованного в виде внешнего модуля к системе Higres

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

МЕТОДЫ ВИЗУАЛИЗАЦИИ АЛГОРИТМОВ

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

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

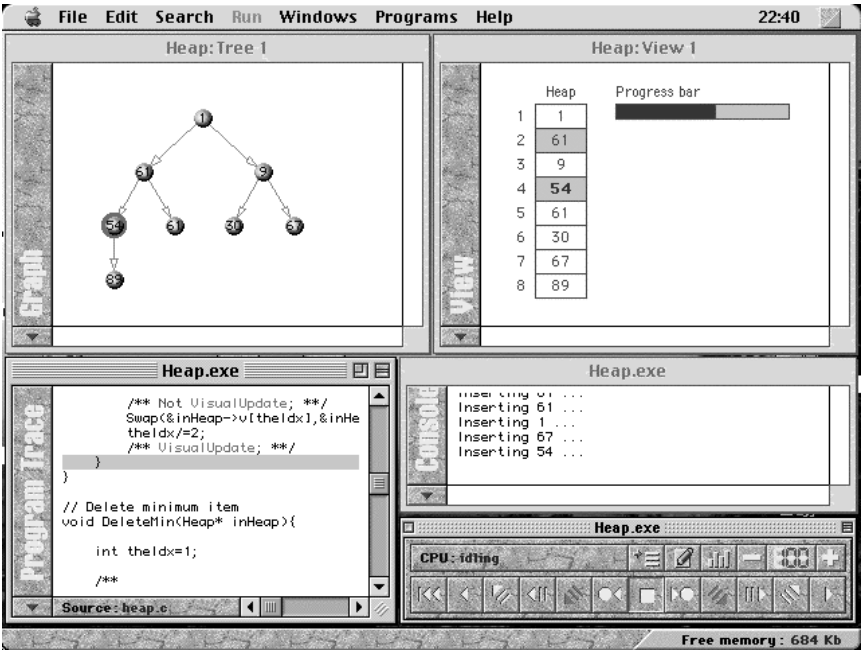


Рис. 3. Пример визуализации операций над binary heaps в системе Leonardo, A C Programming Environment for Reversible Execution and Software Visualization

МОДЕЛЬ ВИЗУАЛИЗАЦИИ

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

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

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

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

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

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

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

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

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

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

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

СИСТЕМА ВИЗУАЛИЗАЦИИ

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

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

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

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

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

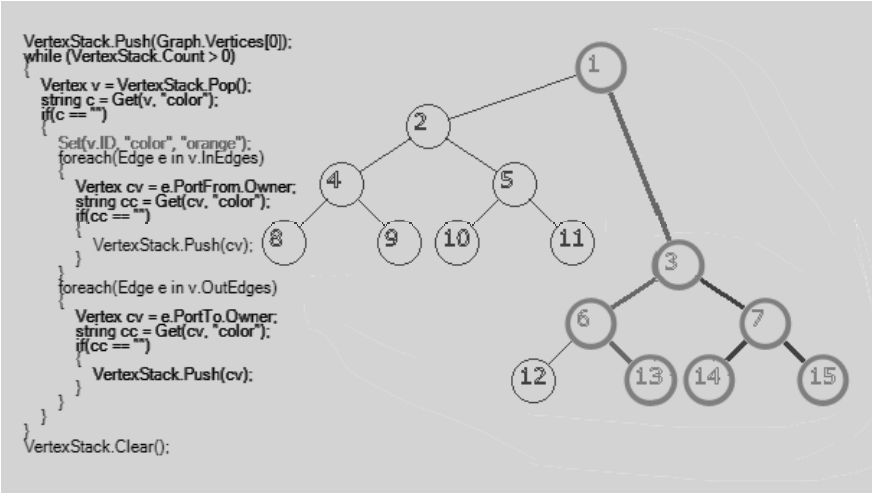


Рис. 4. Пример визуализации алгоритма поиска в глубину, один из промежуточных кадров

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

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

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

ЗАКЛЮЧЕНИЕ

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

СПИСОК ЛИТЕРАТУРЫ

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

Ю.В. Малинина

АВТОМАТИЧЕСКОЕ ВЫЯВЛЕНИЕ ТЕМАТИЧЕСКОЙ КАРТЫ ДОКУМЕНТА

ВВЕДЕНИЕ

Рост массивов полнотекстовых документов, публикуемых в электронном виде, требует новых средств организации доступа к информации. И одной из фундаментальных проблем в этой области является то, что иногда пользователь не знает точно, какую именно информацию ему хотелось бы получить, имея лишь общее представление о границах своих интересов. Это особенно актуально для научных публикаций, так как в конечном счете научное знание воплощается в тексты и познается через тексты. Избыток научной информации в текстовой форме имеет преимущества и проблемы.

1. С одной стороны, можно допустить, что абстрактная модель научного знания является производной от множества реальных текстов.
2. С другой стороны, с каждым годом становится все очевиднее, что потребность в компактификации текстовой информации только увеличивается. По мере накопления знаний мы вынуждены передавать информацию во все более сжатом виде, сжимать имеющиеся знания все сильнее и сильнее.

На сегодня существует несколько подходов к сжатию информации.

1. Классификация (classification). Задача заключается в отнесении документа к одной из нескольких заранее определенных категорий, основываясь на содержании документа. Уже с середины XX в. для упрощения поиска материала по определенной тематике используются периодические реферативные издания, названия и тематическая направленность рубрик которых изменялась в соответствии с тенденциями развития отдельных областей науки. Это наглядный пример представления информации в сжатой форме на основе классификации.
2. Кластеризация (clustering) отличается от классификации тем, что мы заранее не знаем, какие существуют категории в рассматриваемой области, или список этих категорий неполон. Документы объединяются в подмножества динамически на основе выбранных критериев. Примером этого подхода может служить индекс цити-

рования “Science Citation Index” (SCI), ориентированный на поиск новых научных публикаций в мировой системе периодических изданий по системе научных ссылок. В этом случае для кластеризации документов используется естественная исторически сложившаяся система группировки научных работ по ссылкам автора на работы его предшественников в определенной тематической области.

3. Онтологии являются эффективным средством представления и систематизации знаний и еще одним подходом к сжатию информации. Онтологии используются для формальной спецификации понятий и отношений, которые характеризуют определенную предметную область и фактически представляют экстракт знаний об определенной области. В свою очередь, тематические карты (XTM) и RDF/OWL форматы являются наиболее важными стандартами для представления знаний в онтологиях.

ТЕМАТИЧЕСКИЙ АНАЛИЗ ТЕКСТА

Все обозначенные выше идеи сжатия текстовой информации направлены на выявление того, что стоит за словами в документе, т.е. на определение смысла и темы текста. Считается, что тема позволит представить пользователю информацию в сжатом и облегчающем понимание сути документа виде. В то же время основная трудность заключается в идентификации темы документа. Нахождение механизмов автоматического определения основной темы и подтем документа могло бы значительно улучшить эффективность классификации, кластеризации, извлечения и поиска информации.

ТЕМА ДОКУМЕНТА

Идентификация темы часто рассматривается как высокоразвитая техника извлечения метаинформации из текста документов. Тем не менее, перед рассмотрением процесса идентификации темы необходимо определить само понятие темы документа. Согласно Брауну и Юлу [1], обычно под темой документа понимается интуитивно удовлетворительный способ описания основного принципа, который связывает различные смыслы (дискурсы) в рамках его текста. Многие исследователи заняты поиском кратких идентификаторов содержания, которые бы эффективно представляли первичные

смыслы документа. На сегодня приходится констатировать факт, что под темой документа понимается проблема или ряд проблем, которым посвящен документ, т.е. то, о чем идет речь (в самом общем смысле), и определяется это интуитивно.

В нашем случае зафиксируем формулировку основной темы текста как некоторую совокупность слов, наиболее значимых для передачи смысла текста.

ТЕМАТИЧЕСКИЕ КАРТЫ

Тематические карты являются (синтаксически) стандартизированной формой семантических сетей. В концепции тематических карт больше внимания уделяется навигации между темами, чем на связь между ними. В отличие от обычных семантических сетей тематические карты предоставляют дополнительную возможность навигации и поиска.

Наглядно тематическая карта может быть представлена в виде ориентированного графа, состоящего из вершин типа «тема» (topic), соединённых рёбрами типа «ассоциация» (association). Также имеется множество «информационных ресурсов» (occurrences). Некоторые «темы» ссылаются на нужные им «информационные ресурсы». Таким образом, «информационные ресурсы» отделяются от графа «тем» и «ассоциаций», который представляет собой только каталог информации.

Происхождение тематических карт можно проследить еще в начале 1990-х годов, когда группа Дэвенпорт (Davenport group) обсуждала пути, которые позволили бы производить обмен цифровых документов. В последние годы развитие технологии тематических карт сделало ее пригодной для включения в области управления знаниями. Сегодня тематические карты (Topic Maps) – активно развиваемая XML-технология моделирования знаний, задача которой – сделать опубликованную в Web информацию более доступной людям и компьютерам. XTM – открытый стандарт представления тематических карт в формате XML.

Формат тематических карт стандартизирован в ИСО 13250:2003 (ISO/IEC 13250:2003) – http://www.topicmaps.org.

Концепция тематических карт имеет много преимуществ и включает следующее [9].

1. Тематическая карта представляет собой структуру, независимую от местонахождения ссылок, т.е. может ссылаться на любой документ.

Поэтому тематическая карта может быть использована для навигации по нескольким ресурсам.

2. По сравнению с тезаурусами нет жестко определяемых типов связей, например, ассоциативных, иерархических и эквивалентности. Таким образом, возможно создать любой желаемый тип связи. Прирост числа ассоциативных связей позволяет описывать более сложные отношения, но в то же время такую развитую сеть связей труднее создать. Следует подчеркнуть, что связи должны создаваться очень точным и согласованным способом.
3. В тематической карте можно представить несколько точек зрения по любому вопросу. Это стало возможным благодаря характеристике темы «scope», задающей контекст, в котором имя или местоположение присваивается данной теме, и контекст, в котором темы ассоциативно связаны. Эта характеристика тематической карты позволяет поддерживать объективность в коллекциях документов.
4. Тематические карты легко визуализируются. Эта функция обеспечивается приложениями тематических карт.
5. Благодаря Public Subject Identifiers, которые идентифицируют (определяют) предметные темы, несколько тематических карт могут объединяться в одну карту.
6. Механизм, называемый веб-сервисом тематических карт, позволяет обмениваться фрагментами тематических карт. Он может быть использован для создания сетевой кооперации веб-приложений .
7. Прослеживается большая связь между тематическими картами и RDF, следовательно, проекты, основанные на тематических картах, могут быть гармонично включены в семантический веб.

ПРЕДЛАГАЕМЫЙ ПОДХОД

Идентификация темы документа

Для построения тематической карты документа необходимо идентифицировать темы и подтемы документа. Эта задача требует определения синтаксической структуры, а также семантики текста. На практике это означает глубокий анализ синтаксиса и семантики, что является трудной задачей из-за сложностей естественного языка [6]. С другой стороны, поскольку мы заинтересованы только в семантических связях, которые являются особенностями содержания документа, то определенные языковые закономерно-

сти, обнаруженные в документах, ограничены. Подходы к построению семантических сетей и выявление терминов для научных текстов были более подробно рассмотрены в предыдущих работах [7, 8].

Рассмотренная ниже технология тематического анализа позволяет дополнительно автоматически выявлять ключевые темы текста. В ходе тематического анализа устанавливаются ассоциативные связи между темами. Совокупность тем со связями образует ассоциативную семантическую сеть, представляющую модель текста, которая в дальнейшем представляется в виде тематической карты текста или совокупности текстов

Связанность текста (лексические цепочки)

Для любого данного документа слова или фразы должны быть всегда лексически связаны в цепочки вокруг центральной темы. Понятие лексической цепочки вытекает из работы, связанной с понятием текстовой сплоченности в лингвистике [4]. Согласно этой теории, смысл документа заключается в том, чтобы быть связанным и удерживать эту связь сквозь всю структуру текста через грамматическое единство, которое достигается с помощью таких средств языка, как ссылки, замены и семантически связанные слова. Лексические сплоченности неявно существуют не только между парами терминов, но и между последовательностями слов. Для последнего случая используется понятие лексической цепочки, впервые предложенное Morris J. и Hirst G. [5].

Если рассмотреть текст, то можно заметить, что слова и словосочетания, близкие по смыслу к словам основной темы, образуют лексические цепочки, которые пронизывают весь текст. Естественно предположить, что, если имеется лингвистический ресурс, в котором описаны разнообразные смысловые связи между словами (WordNet, RusNet), то можно двигаться по тексту, находить связанные по смыслу слова и формировать лексические цепочки [2]. Самые частотные (или выделенные по другим критериям) цепочки могли бы показать, чему именно посвящен конкретный текст [3].

Измерение семантической близости

Традиционный метод расчета сходства рассматривает частоты слов, но не принимает во внимание семантические отношений между словами. Подобный подход может привести в результате к неточной кластеризации. Вместо того, чтобы использовать TF и TFIDF метрики сходства, предлагается измерить семантическое сходство терминов при помощи лексической цепи. Этот метод был предложен Barzilay и Elhadad [3] и он анализирует

смысл не-терминов, итеративно измеряя семантическое сходство между парами терминов и группируя термины в кластеры, которые в конечном итоге представляют концепции каждого документа или набора документов.

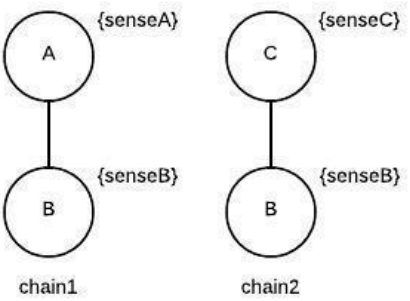
Нахождение лексических цепочек

Лексическая цепь, согласно [10], представляет собой набор семантически связанных слов в тексте. На этом шаге мы ищем семантику слов в WordNet (RusNet) и вычисляем вес сходства в соответствии с отношением гипонимии в WordNet (RusNet). Шаги построения лексической цепи в основном следуют алгоритму, предложенному в работах [3, 11]:

Первоначально слово-кандидат выбирается из общих слов документа, которые не являются фразами [3]. Затем эти слова сравниваются друг с другом в соответствии с порядком слов в этом документе. Кроме того, для слова-кандидата смысл раскрывается на основе WordNet (RusNet) словаря [11].

На этом этапе рассматриваются все смыслы слова, и каждый смысл сохраняется на разных уровнях. Первый уровень представляет собой набор синонимов и антонимов, а второй – набор первых гипонимов/гиперонимов, а также их вариаций (т. е. меронимы/холонимы и т.д.).

Предположим следующую ситуацию: извлечены три слова, A, B и C; их смысловой набор, соответственно, senseA, senseB и senseC. Если для первого слова успешно найдено соответствие со вторым словом B, то они производят два вида перестановок. Первый вид перестановки – это комбинация senseA ∪ senseB, которая является первой цепочкой chain1. Вторая комбинация senseB ∪ senseC является второй цепочкой chain2. Затем сопоставляем третье слово C с A и B. Когда они совпадают успешно, они производят четыре вида перестановок, таких как chain1 ∪ senseC и chain2 ∪ senseC. При помощи этого способа нужно найти соответствие для каждого слова в документе. Если они явно совпадают, они будут производить другие лексические цепочки.



Пока этот метод не вычисляет вес связи. Если обнаруживается подходящая лексическая цепь, алгоритм просто подсоединит слово к лексической цепочке. Если связь не определена, то будет строиться новая цепочка. Эта процедура будет работать итеративно, пока все соответствия не будут найдены.

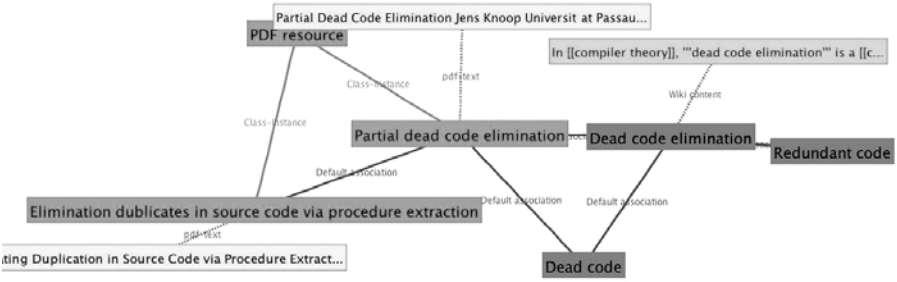
Когда обнаружение всех лексических цепей закончится, в качестве следующего шага необходимо будет проверить идентичность смыслов найденных лексических цепей. На этом этапе некоторые из них могут быть объединены, поскольку они являются однородными по семантике.

Далее мы просто суммируем все связи для каждой лексической цепи, чтобы указать вес каждой. Когда лексические цепочки приобретают вес, становится возможным выбирать старшую по весу лексическую цепь и считать, что ее слова представляют основную концепцию (тему) документа, а остальные являются вспомогательными темами

ПОСТРОЕНИЕ ТЕМАТИЧЕСКОЙ КАРТЫ

Для того чтобы привязать полученную семантическую модель к интересуемой предметной области, используем словарь соответствующей тематики. В нашем случае обратимся к онтологиям, описанным в работах Князевой М.А. [12,13]. В итоговой онтологии фиксируются только те семантические конструкции, в которых участвуют термины из словаря предметной области. Далее, воспользовавшись тем фактом, что тематические карты являются (синтаксически) стандартизированной формой семантических сетей, и сходством структуры семантической сети и тематической карты, мы можем построить относительно просто отображение узлов семантической сети в темы карты, а ссылки рассматривать как ассоциации.

На приведенной схеме изображен пример отрывка полученной тематической карты, построенной на основе множества публикаций об удалении неиспользуемого кода.



СПИСОК ЛИТЕРАТУРЫ

1. Brown, G., Yule G. Discourse Analysis // Cambridge Textbooks in Linguistics Series. – Cambridge University Press, 1983.
2. Morris J., Hirst G. Lexical cohesion computed by thesaural relations as an indicator of the structure of a text // Computational Linguistics. – 1991. – Vol. 17 (1). – P. 21–48.
3. Barzilay R., Elhadad M. Using Lexical Chains for Text Summarization // ACL/EACL Workshop Intelligent Scalable Text Summarization. – Madrid, 1997.
4. Halliday M., Hasan R. Cohesion in English. – London: Longman, 1976.
5. Morris J., Hirst G. Lexical cohesion computed by thesaural relations as an indicator of the structure of text // Computational Linguistics. – 1991. – Vol. 17 (1). – P. 21–48.
6. Glasgow B., Mandell A., Binney D., Ghemri L., Fisher D. An information-extraction approach to the analysis of free-form text in life insurance applications // AI Magazine. – 1998. – Vol. 19. – P. 59–71.
7. Малинина Ю.В. Семантическая сеть как формальный метод описания и обработки текстов по преобразованиям программ // Методы и инструменты конструирования и оптимизации программ. – Новосибирск, ИСИ СО РАН, 2005. – С. 137–144.
8. Малинина Ю.В. Автоматическое выявление таксономии в области преобразований программ на основе анализа семантических связей в публикациях // Методы и инструменты конструирования и оптимизации программ. – Новосибирск, ИСИ СО РАН, 2008.

9. Włodarczyk B. Topic map library = better library: an introduction to the project of the National Library of Poland // IFLA World Library and Information Congress 78th IFLA General Conference and Assembly 11–17 August 2012, Helsinki, Finland.
10. Li S.C., Wang H.C. Document Topic Detection Based On Semantic Feature // The 18th International Conference on Information Management, 2007. – P 56–73.
11. Chen K.J., Chen C. J.,Automatic Semantic Classification for Chinese Unknown Compound Nouns // Coling 2000: Proc. – P. 173–179.
12. Артемьева И.Л. Модель онтологии предметной области «оптимизация последовательных программ». Термины для описания объекта оптимизации. / И.Л. Артемьева, М.А. Князева, О.А. Купневич. – Владивосток. – 43 с. – (Преп. / НТИ 29-2000).
13. Князева М.А., Гужавин В.Д. Онтология низкоуровневой оптимизации программ // Научная сессия МИФИ-2002.

СОДЕРЖАНИЕ

CONTENTS

УДК 519.68 + 681.3.06

Язык представления графов GraphML: базовые средства / Касьянов В.Н. // Информатика в науке и образовании: Сб. науч. тр. / Под ред. В.Н. Касьянова. – Новосибирск, 2012. – С. 7–22.

В статье рассматривается текущее состояние работ по международному проекту GraphML, инициированному сообществом по рисованию графов в 2000 г. с целью создания стандартизованного языка описания графов на основе XML. Описываются базовые средства языка GraphML, достаточные для представления графовых моделей в большинстве приложений.. – Библиогр.: 17 назв.

Graph ML language for graph presentation: basic means / Kasyanov V.N. // Informatics in research and education / Ed. Victor Kasyanov. – Novosibirsk, 2012. – P. 7–22.

The paper deals with the current state of the GraphML project, initiated by the graph drawing community in 2000 with a view to developing a standard graph description language based on XML format. Basic GraphML concepts sufficient for graph models presentation in most applications are described. – Refs: 17 titles.

УДК 519.68 + 681.3.06

Язык представления графов GraphML: дополнительные возможности / Касьянов В.Н. // Информатика в науке и образовании: Сб. науч. тр. / Под ред. В.Н. Касьянова. – Новосибирск, 2012. – С. 23–46.

В статье рассматривается текущее состояние работ по международному проекту GraphML, инициированному сообществом по рисованию графов в 2000 г. с целью создания стандартизованного языка описания графов на основе XML. Описываются дополнительные возможности языка GraphML, расширяющие базисные средства языка. – Библиогр.: 23 назв.

Graph ML language for graph presentation: additional functionalities / Kasyanov V.N. // Informatics in research and education / Ed. Victor Kasyanov. – Novosibirsk, 2012. – P. 23–46.

The paper deals with the current state of the GraphML project, initiated by the graph drawing community in 2000 with a view to developing a standard graph description language based on XML format. Additional functionalities of GraphML like nested graphs, hypergraphs and ports are described. – Refs: 23 titles.

УДК 519.68 + 681.3.06

Средства поддержки применения графов в информатике и программировании / Касьянов В.Н., Касьянова Е.В. // Информатика в науке и образовании: Сб. науч. тр. / Под ред. В.Н. Касьянова. – Новосибирск, 2012. – С. 47–56.

Статья посвящена средствам поддержки применения теоретико-графовых

методов в информатике и программировании, которые разрабатываются в Институте систем информатики им. А.П. Ершова СО РАН. В ней рассматриваются системы WikiGRAPP, WEGA, Higres и Visual Graph. – Библиогр.: 11 назв.

Tools for graph application support in informatics and programming / Kasyanov V.N., Kasyanova E.V. // Informatics in research and education / Ed. Victor Kasyanov. – Novosibirsk, 2012. – P. 47–56.

The paper is devoted to tools for supporting the application of graph-theory methods in informatics and programming which are under development at A.P. Ershov Institute of Informatics Systems. The WikiGRAPP, WEGA, Higres and Visual Graph systems are considered. – Refs: 11 titles.

УДК 519.68 + 681.3.06

Облачный сервис для научных вычислений и образования / Идрисов Р.И. // Информатика в науке и образовании: Сб. науч. тр. / Под ред. В.Н. Касьянова. – Новосибирск, 2012. – С. 57–64.

На сегодняшний день всё большую популярность набирают облачные сервисы, которые предоставляют различные возможности. Конечно, под большим количеством слова «облачный» подчас скрываются обычные вещи, которые просто были названы по-новому. Согласно последней редакции российской Википедии на момент написания статьи, облачный сервис – это просто некоторый доступный ресурс в сети, который может быть использован без знания его внутренней структуры.

Разрабатываемая система ставит перед собой две цели: научную и образовательную. Для научной цели более критична масштабируемость, а для образовательной – доступность. Кроме того, было бы неправильно ориентироваться только на один язык программирования, поскольку не существует единого мнения о наилучшем учебном языке. Для масштабируемости требуется универсальность описания параллелизма, это значит, что программа не должна быть адаптирована для структуры конкретной вычислительной системы. Согласно работам А. П. Ершова это достигается, если язык программирования приближается к языку описания задач, а не к языку описания алгоритмов. В Институте систем информатики СО РАН мы продолжаем разработку потокового языка программирования Sisal, эта работа ставит перед собой именно такие цели. – Библиогр.: 2 назв.

Cloud service for research computing and education / Idrisov R.I. // Informatics in research and education / Ed. Victor Kasyanov. – Novosibirsk, 2012. – P. 57–64.

Cloud services, offering various capabilities to potential users, are becoming increasingly popular. Of course, often “cloud” service appears due to rebranding. According to the latest definition found in the Russian Wikipedia, cloud service is just a service which can be used without any knowledge of its internal structure and which is available via the internet.

Our system has two purposes: research and educational. Scalability is more

significant for research and availability is critical for education. We suppose that it would be wrong to use just one input language for this system, because there is no single opinion about the best computer language for educational purposes. Scalability requires universality of parallelism description, which means that the program should not be adapted to the structure of a specific computing system. According to A.P. Ershov's works, this is achieved when the language is closer to the task definition language and not to the algorithm language. At our institute, we are developing the dataflow programming language Sisal which sets the same aims. – Refs: 2 titles.

УДК 519.68 + 681.3.06

Параллелизм в JavaScript / Идрисов Р.И. // Информатика в науке и образовании: Сб. науч. тр. / Под ред. В.Н. Касьянова. – Новосибирск, 2012. – С. 65–70.

О преимуществах параллельного исполнения на сегодняшний день уже можно подробно не рассказывать, поскольку эта тема стала достаточно очевидной. Тенденции таковы, что даже мобильные телефоны оснащаются многоядерными процессорами. Есть так же и другая тенденция, заключающаяся в том, что большое количество приложений переходят в браузеры (программы просмотра html-документов). В статье рассматриваются различные подходы к организации параллельного исполнения на языке JavaScript. – Библиогр.: 4 назв.

Parallelism in JavaScript / Idrisov R.I. // Informatics in research and education / Ed. Victor Kasyanov. – Novosibirsk, 2012. – P. 65–70.

The advantages of parallel programming are well known today and there is no need to describe them in detail. The current trend is that even cell phones can contain multi-core processors. Another trend is a rapid growth of browser applications. This paper briefly overviews different approaches to concurrency implementation in JavaScript. – Refs: 4 titles.

УДК 519.68 + 681.3.06

Биоинформатика: пути развития и перспективы / Несговорова Г.П. // Информатика в науке и образовании: Сб. науч. тр. / Под ред. В.Н. Касьянова. – Новосибирск, 2012. – С. 71–89.

В статье рассматриваются история возникновения, развития и перспективы достаточно новой науки – биоинформатики. Дается ее определение, перечисляются основные области исследований в биоинформатике, определяется ее место в цепи биологических исследований. – Библиогр.: 10 назв.

Bioinformatics: development and prospects / Nesgovorova G.P. // Informatics in research and education / Ed. Victor Kasyanov. – Novosibirsk, 2012. – P. 71–89.

This paper deals with the history of origin, development and prospects of the new science of bioinformatics. The paper provides the definition of bioinformatics, enumerates its fundamental fields of research, and determines its place among

other areas of biological research. – Refs: 10 titles.

УДК 519.68 + 681.3.06

Информационные технологии в гуманитарных исследованиях и гуманитарном образовании / Несговорова Г.П. // Информатика в науке и образовании: Сб. науч. тр. / Под ред. В.Н. Касьянова. – Новосибирск, 2012. Дается обзор информационно-коммуникационных технологий, применяемых в гуманитарных исследованиях и гуманитарном образовании в наши дни. – Библиогр.: 5 назв.

Information technologies in humanities research and education / Nesgovorova G.P. // Informatics in research and education / Ed. Victor Kasyanov. – Novosibirsk, 2012. – P. 90–105.

The authors review the use of information-communication technologies used in humanities research and education. – Refs: 5 titles.

УДК 519.68 + 681.3.06

Тестирование изменений в программной системе на основе покрытия исходного кода / Стасенко А.П. // Информатика в науке и образовании: Сб. науч. тр. / Под ред. В.Н. Касьянова. – Новосибирск, 2012. – С. 106–115.

В больших программных системах используется предварительное тестирование каждой правки исходного кода. Длительность предварительного тестирования является одним из факторов, ограничивающих скорость работы над проектом. Данная статья описывает подход к сокращению объема предварительного тестирования, основанного на сопоставлении изменений в исходном коде с данными о тестовом покрытии.

Ключевые слова: покрытие кода, оптимизация регрессионного тестирования изменений. – Библиогр.: 6 назв.

Testing of changes in a software system on the basis of source code coverage / Stasenko A.P. // Informatics in research and education / Ed. Victor Kasyanov. – Novosibirsk, 2012. – P. 106–115.

The development of complex software systems usually requires extensive pre-commit testing of every change in the source code. The duration of the pre-commit testing is one of the factors limiting the speed of working on the software project. This paper describes an approach to the testing volume reduction based on comparison between the source code changes and the source code coverage.

Keywords: source code coverage, pre-commit regression testing optimization. – Refs: 6 titles.

УДК 519.68 + 681.3.06

Информационная система для поддержки процесса проведения исследований на основе литературных источников / Шманина Т. В. // Информатика в науке и образовании: Сб. науч. тр. / Под ред. В.Н. Касьянова. – Новосибирск, 2012. – С. 116–134.

В статье приводится описание подхода, автоматизирующего процесс исследования на основе литературных источников путем построения графа взаимосвязи тем на локальной конечной коллекции текстовых документов. Приводится описание прототипа информационной системы, реализующей данный подход и в перспективе способной выполнять роль ассистента исследователя при поиске потенциальных решений исследовательских задач. Приводятся результаты тестирования системы, показывающие применимость предложенного метода для решения данной задачи. – Библиогр.: 11 назв.

Information system for the support of research based on literary sources / Shmanina T.V. // Informatics in research and education / Ed. Victor Kasyanov. – Novosibirsk, 2012. – P. 116–134.

The paper presents an approach to the automation of research based on literary sources. For a local textual document collection, the proposed method constructs a topic interrelation graph. The prototype of the information system based on this approach is described. The system is designed to be able to play the role of a research assistant in finding potential solutions for research problems. Test results showing good performance of the method described are listed. – Refs: 11 titles.

УДК 519.68 + 681.3.06

Визуализация графов при помощи программного средства Visual Graph / Золотухин Т.А. // Информатика в науке и образовании: Сб. науч. тр. / Под ред. В.Н. Касьянова. – Новосибирск, 2012. – С. 135–148.

В статье рассматривается визуализация атрибутированных иерархических графов при помощи интерактивной системы Visual Graph. Приведены несколько иллюстраций, показывающих общий вид системы, а так же отдельных ее частей. – Библиогр.: 9 назв.

Graph visualization using the program tool Visual Graph / Zolotukhin T.A. // Informatics in research and education / Ed. Victor Kasyanov. – Novosibirsk, 2012. – P. 135–148.

The paper considers visualization of attributed hierarchical graphs with the interactive system Visual Graph. Several illustrations showing a general view of the system as well as its separate parts are presented. – Refs: 9 titles.

УДК 519.68 + 681.3.06

Визуализация алгоритмов на графах: интерпретация алгоритма в качестве программы / Гордеев Д.С. // Информатика в науке и образовании: Сб. науч. тр. / Под ред. В.Н. Касьянова. – Новосибирск, 2012. – С. 149–160.

В статье описывается метод визуализации алгоритмов на графах, обеспечивающий задание алгоритма в качестве параметра. Под алгоритмами на графах следует понимать алгоритмы, обрабатывающие графы. Основными преимуществами данного подхода являются возможность задания алгоритма в текстовой форме в качестве параметра, возможность задать граф в качестве параметра, а также гибко настраивать

результирующую визуализацию. Визуализация алгоритмов осуществляется с помощью множества настраиваемых эффектов. В качестве входных графов рассматривается класс иерархических графов. Использование таких графов позволяет не ограничивать множество рассматриваемых графов, а также позволяет облегчить некоторые аспекты визуализации дополнительной информации. Описываемый подход может использоваться как для создания образовательных систем, так и для исследования графовых алгоритмов. – Библиогр.: 8 назв.

Graph algorithm visualization: interpretation of an algorithm as a program / Gordeev D.S. // Informatics in research and education / Ed. Victor Kasyanov. – Novosibirsk, 2012. – P. 149–160.

This paper describes a new method of graph algorithm visualization based on a dynamic approach. By graph algorithms we mean graph processing algorithms. The main advantages of this approach are the possibility of setting an algorithm in a textual form as an input parameter, of setting a graph as an input parameter, and of adjusting visualization flexibly. Visualization of algorithms is carried out by means of a set of configurable visual effects. The class of hierarchical graphs is used as input parameters. This allows us to use input graphs of any type and to represent additional data appearing during the algorithm run as a part of a single visualized graph model. This approach can be used both for research and educational purposes. – Refs: 8 titles.

УДК 519.68 + 681.3.06

Автоматическое выявление тематической карты документа / Малинина Ю.В. // Информатика в науке и образовании: Сб. науч. тр. / Под ред. В.Н. Касьянова. – Новосибирск, 2012. – С. 161–169.

Статья рассматривает подход к удобной организации текстовых ресурсов и быстрого доступа к информации на основе автоматического выявления тематической карты документа при помощи лексических цепей. Выделенные концепции и связи совместно образуют тематическую карту. Таким образом, происходит интеграция онтологии и тематических карт для развития информационной системы по преобразованиям программ. – Библиогр.: 13 назв.

Automated identification of a document’s topic map / Malinina Yu.V. // Informatics in research and education / Ed. Victor Kasyanov. – Novosibirsk, 2012. – P. 161–169.

The paper considers an approach to convenient organization of textual resources and quick access to information on the basis of automated identification of a document’s topic map using lexical chains. The concepts and associations selected together form a topic map. Thus, the ontology and topic maps are integrated for further development of the information system on program transformations. – Refs: 13 titles.

ИНФОРМАТИКА В НАУКЕ И ОБРАЗОВАНИИ

Под редакцией
проф. Виктора Николаевича Касьянова

Рукопись поступила в редакцию 15. 08.2012
Ответственный за выпуск Г.П. Несговорова
Редактор Т.М. Бульонкова

Подписано в печать 27. 11. 2012
Формат бумаги 60 × 84 1/16 Объем 10,2 уч.-изд.л., 11,1 п.л.
Тираж 75 экз.

Центр оперативной печати “Оригинал 2”, г.Бердск,
ул. Островского, 55, оф. 02, тел. (383) 214-45-35